



MAGISTÈRE INFORMATIQUE DE GRENOBLE 2014

Proceedings

Supervised by Michaël Périn and Cyril Labbé

August 27, 2014

- 1 Rémy Boutonnet**
WCET analysis : a counter-based approach
Supervized by Fabienne Carrier and Claire Maiza
Verimag
- 2 Alexandre Maréchal**
A linearization technique for multivariate polynomials using convex polyhedra based on Handelman's theorem
Supervized by David Monniaux and Michaël Périn
Verimag
- 3 Raphaël Jakse**
DBnary: Extracting Inflected Forms from the French Wiktionary
Supervized by Gilles Sérasset
LIG
- 4 Jérémie Suzan**
DBnary : Extracting German inflected Form from German Wiktionary
Supervized by Gilles Sérasset
LIG
- 5 Thomas Baumela**
MPSoC Virtual Platform Generation
Supervized by Nicolas Fournel and Frederic Petrot
TIMA Laboratory - SLS Group

- 6 Mehdi Makhlouf**
PWM Driver (Pulse Width Modulation)
Supervized by Nicolas Fournel and Frederic Petrot
TIMA Laboratory - SLS Group
- 7 Luc Libralesso**
Resolution of Bin Packing Problem on GPU
Supervized by Michaël Gabay
G-SCOP
- 8 Lea Albert**
Performance analysis of accelerators caches
Supervized by Guillaume Huard
Inria - MOAIS
- 9 Myriam Clouet**
VSS-Simulator Stats Viewer
Supervized by Vincent Jost
G-SCOP
- 10 Marion Dalle**
Analytical model on memory and cache coherence
(Confidential internship, abstract only)
Supervized by Florence Perronin and Jean-Marc Vincent
CEA
- 11 Jérémy Wambecke**
Distributed Interactive Model Checking for Hybrid Systems
Supervized by Goran Frehse and Olivier Lebeltel
Verimag

- 12 Anaïs Durand**
Self-Stabilizing Leader Election in Polynomial Steps
Supervised by Karine Altisen and Stéphane Devismes
Verimag
- 13 Maxime Puys and Lionel Rivière and Thanh-Ha Le and Julien Bringer**
High-Level Simulation for Multiple Fault Injection Evaluation
Supervised by Marie-Laure Potet
Verimag
- 14 Carole Plasson**
Human detection in indoor environment with 3D sensor
Supervised by Olivier Aycard
LIG - Team AMA
- 15 Antoine Faravelon**
Fast and Accurate Branch Predictor Simulation
Supervised by Nicolas Fournel and Frederic Petrot
TIMA Laboratory - SLS Group
- 16 Hugo Guiroux**
Towards simpler performance troubleshooting for kernel-intensive workloads
Supervised by Renaud Lachaize and Vivien Quéma
LIG - Team ERODS

- 17 Jérémy Seguin**
Evaluation of Xeon Phi's communications means
Supervized by Vincent Danjean
Inria - MOAIS
- 18 Simon Moura**
Hierarchy construction in large scale taxonomies
Supervized by Eric Gaussier
LIG - Team AMA
- 19 Maxime Portaz**
Comparison of geometrical and model based retargeting methods for imitation games using an RGB-D sensor
Supervized by Etienne Balit and Dominique Vaufreydaz
INRIA - PRIMA
- 20 Gregory Cano**
Combining interactive visualisation and UI plasticity to go beyond conventional homescreens
Supervized by Gaelle Calvary
LIG - Team IIHM
- 21 Geoffrey Danet**
Analysis of spatial phenomena using multi-scale aggregation
Supervized by Jean-Marc Vincent and Christine Plumejeaud-Perreau
LIENSs La Rochelle - INRIA Grenoble

WCET analysis : a counter-based approach

Rémy BOUTONNET

VERIMAG, University of Grenoble, France

13 août 2014

Hard real-time systems have strict timing constraints that must be satisfied to avoid catastrophic events at runtime, major injuries or death. Safe upper-bounds on the execution time of real-time tasks must be derived to insure that the system meets its constraints. The counter approach is proposed to tighten the Worst-Case Execution Time (WCET) estimation, by adding special variables called counters to C programs. This internship shows the interest of the counter approach for semantic property extraction in the WCET context and how it can improve WCET estimations with a set of workflow and tools.

Unfortunately, it is not possible in general to compute upper-bounds on the execution time of programs [11], otherwise one could solve the halting problem. Hopefully, we restrict ourselves to programs used in embedded hard real-time systems that must terminate and where iteration have to be bounded. However, the real WCET cannot be known exactly due to the size of the state space of any non-trivial program, and components like caches and branch prediction used in modern processors [10]. This is why WCET analysis aims to compute an upper-bound on the real WCET, that must be safe (no underestimation so as to reject programs that may violate their constraints) and tight (to be of any use).

The counter approach [2] is proposed to refine the WCET estimation computed by existing tools via the extraction of semantic properties of programs, like infeasible paths or loop bounds, with special variables called counters added at key program points (e.g. basic blocks) and incremented each time the control flows through that point.

This internship shows how the WCET estimation computed by existing tools can be improved effectively by adding these counters to C programs.

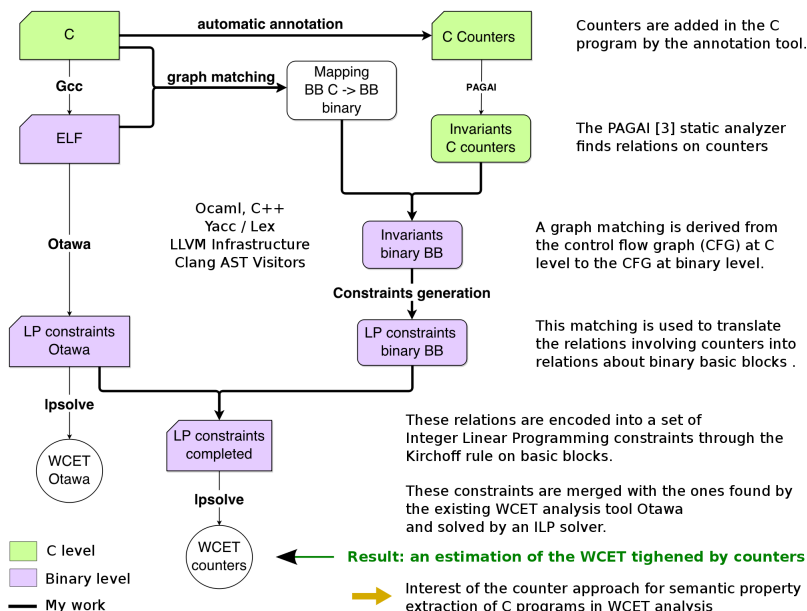


FIGURE 1 – The counters workflow. The tools corresponding to arrows in bold have been developed during this internship.

that the Otawa and oRange [5] tools were not able to detect, thus enabling the computation of a WCET bound. These benchmarks are illustrating that the counter approach can be used to improve the WCET estimation of existing tools giving two main kinds of semantic properties : the presence of infeasible paths and more accurate loop bounds, especially with iteration variables modified in the loop body and while loops containing breaks and continues.

The C program is first compiled with Gcc and analyzed by the WCET tool Otawa [1] that outputs a set of Integer Linear Programming (ILP) constraints [9] which are solved by an ILP solver to obtain a WCET estimation.

Counters variables are automatically added to the C program by an annotation tool that I developed, based on the LLVM infrastructure [8] and the Clang C compiler. A second tool derives a graph matching from the Control Flow Graph (CFG) of the program at C level to the CFG at binary level to rewrite the invariants on counters found by the PAGAI static analyzer [7] into invariants on binary basic blocks.

These invariants are translated into ILP constraints, added to the ones produced by Otawa and solved by the ILP solver to obtain a new WCET estimation.

The WCET computed by Otawa and the WCET computed through the use of counters have been compared with a set of benchmarks, showing improvements ranging from 2 % to 43 % and even loop bounds

Références

- [1] Clément Ballabriga, Hugues Cassé, Christine Rochange, and Pascal Sainrat. Ottawa : An open toolbox for adaptive wcet analysis. In Sang Lyul Min, Robert G. Pettit IV, Peter P. Puschner, and Theo Ungerer, editors, *SEUS*, volume 6399 of *Lecture Notes in Computer Science*, pages 35–46. Springer, 2010.
- [2] R. Boutonnet and M. Asavaoe. The wcet analysis using counters - a preliminary assessment. *8th Junior Researcher Workshop on Real-Time Computing - JRWRTC 2014 (Submitted)*, 2014.
- [3] P. Cousot and R. Cousot. Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.
- [4] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, NY.
- [5] Marianne de Michiel, Armelle Bonenfant, Clément Ballabriga, and Hugues Cassé. Partial flow analysis with orange. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification, and Validation*, volume 6416 of *Lecture Notes in Computer Science*, pages 479–482. Springer Berlin Heidelberg, 2010.
- [6] Julien Henry, Mihail Asavaoe, David Monniaux, and Claire Maïza. How to compute worst-case execution time by optimization modulo theory and a clever encoding of program semantics. In *Proceedings of the 2014 SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems, LCTES '14*, pages 43–52, New York, NY, USA, 2014. ACM.
- [7] Julien Henry, David Monniaux, and Matthieu Moy. Pagai : A path sensitive static analyser. *Electronic Notes in Theoretical Computer Science*, 289(0) :15 – 25, 2012. Third Workshop on Tools for Automatic Program Analysis (TAPAS' 2012).
- [8] Chris Lattner and Vikram Adve. Llvm : A compilation framework for lifelong program analysis & transformation. In *Proceedings of the International Symposium on Code Generation and Optimization : Feedback-directed and Runtime Optimization, CGO '04*, pages 75–, Washington, DC, USA, 2004. IEEE Computer Society.
- [9] Yau-Tsun Steven Li and Sharad Malik. Performance analysis of embedded software using implicit path enumeration. In *Proceedings of the 32Nd Annual ACM/IEEE Design Automation Conference, DAC '95*, pages 456–461, New York, NY, USA, 1995. ACM.
- [10] Reinhard Wilhelm, Sebastian Altmeyer, Claire Burguière, Daniel Grund, Jörg Herter, Jan Reineke, Björn Wachter, and Stephan Wilhelm. Static timing analysis for hard real-time systems. In Gilles Barthe and Manuel Hermenegildo, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 5944 of *Lecture Notes in Computer Science*, pages 3–22. Springer Berlin Heidelberg, 2010.
- [11] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3) :36 :1–36 :53, May 2008.

WCET analysis: a counter-based approach

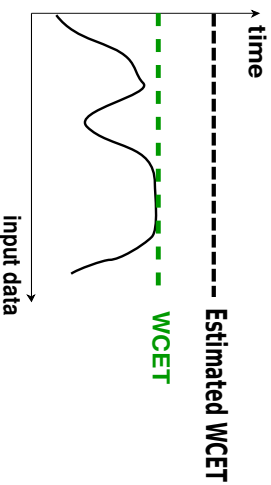
Supervised by Fabienne Carrier and Claire Maiza

Rémy BOUTONNET - UJF VERIMAG Synchrone



The Worst-case execution time

Hard real-time systems have to satisfy strict timing constraints. Deriving a safe upper-bound on the execution time of a task is essential for the schedulability of the whole system.



We propose to add special variables called counters on C programs to refine the estimated worst-case execution time (WCET) [1, 2] of real-time tasks computed by existing tools.

An example C program: infeasible.c

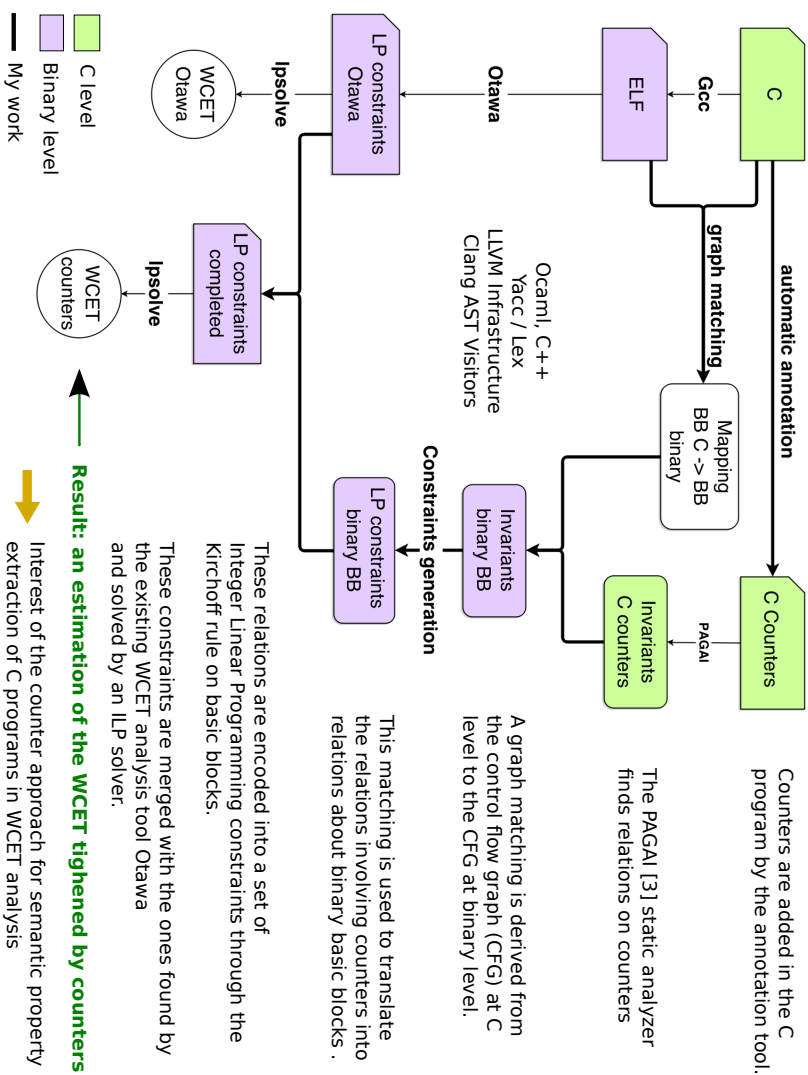
This simple C program has an infeasible path that existing tools like Otawa [4] and oRange does not detect.

```
int infeasible(int x, int y) {
  int result;
  cptr_infeasible1:
  if (x && y) {
    cptr_infeasible2:
    result = 0xIBADB001;
  }
  if (x &&& !y) {
    cptr_infeasible3:
    result = 0xIBADB002;
  }
  return result;
}
```

Mutually exclusive conditions

Variables `xi_infeasible` are counting the number of executions of block `i`

The workflow of counters



```
infeasible.c
1 - cptr_infeasible2 - cptr_infeasible3 >= 0
cptr_infeasible3 >= 0
cptr_infeasible2 >= 0
```

Graph matching and constraints generation

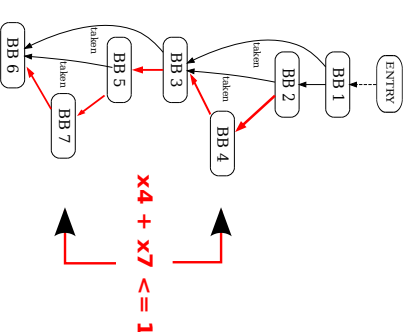
```
Matching
cptr_infeasible2 : block 4
cptr_infeasible3 : block 7
```

```
1 - x4_infeasible - x7_infeasible >= 0
x7_infeasible >= 0
x4_infeasible >= 0
```

Constraints solving

```
WCET counters = 46 cycles
WCET Otawa = 49 cycles
```

An infeasible path in infeasible.c



The tool using the counters approach finds the invariant stating that the path BB 2 - 4 - 3 - 5 - 7 is infeasible.

Automatic annotation of C programs

A source-to-source transformation tool based on the Clang API has been developed to automatically annotate C / C++ programs.

In the context of the counters approach, it provides automatic counters adding for control structures, regardless of the syntactic form of the program.

References

- [1] The worst-case execution time problem - overview of methods and survey of tools, R.Wilhelm et al., ACM Transactions on Embedded Computing Systems, Vol. 8, Issue 3, April 2008.
- [2] Static timing analysis for hard real-time systems, R.Wilhelm, S.Altmeppen, C.Bourjuela, D.Grunu, J.Herder, J.Henneke, B.Wachter, S.Wilhelm, VMCAI 2010, LNCS 5944, pp. 3-22, 2010.
- [3] PAGAI: a path sensitive static analyzer, J.Henry, D.Monniaux, M.Moy, TAPAS12.
- [4] OTAWA: An Open toolbox for Adaptive WCET Analysis, C.Ballabriga, H.Casé, C.Rochange, P.Sarrac, 8th IEP WG 10.2 International Workshop, SEUS 2010, Waidhofen/Ybbs, Austria, October 13-15, 2010, Proceedings.

A linearization technique for multivariate polynomials using convex polyhedra based on Handelman's theorem

Alexandre Maréchal

Supervised by David Monniaux and Michaël Périn

Abstract

We present a linearization method to over-approximate non-linear multivariate polynomials with convex polyhedra. It is based on Handelman's theorem and consists in using products of constraints of a starting polyhedron to over-approximate a polynomial. We also implemented two other linearization methods that we will not detail in this paper, but that we shall use as comparison. As a part of the VERASCO project, the goal is to prove such methods with the proof assistant Coq.

1 Toward Certification of a C compiler

1.1 The VERASCO project

The CompCert project (2006-2010) from Inria Paris-Rocquencourt and Rennes consisted in the formal verification of a C compiler. The goal was to avoid miscompilation which is the production of an executable that does not match the source code. The project led to CompCert, the first C compiler certified using the proof assistant Coq. This compiler is proved to produce a correct executable code if the corresponding source code can not lead to a runtime error. Thus, the correctness of the executable code depends on an assumption on the source code. The VERASCO project follows CompCert and gathers Inria Paris-Rocquencourt, Inria Saclay, VERIMAG laboratory, Airbus, and the University of Rennes. VERASCO aims at developing a certified static analyzer capable of proving the absence of error in the source code, hence discarding the CompCert assumption. The principle of verification based on static analysis is to compute an over-approximation of all possible reachable states of the program, examining only the source code, then to check that no error state is reachable. As every program, such a verification tool is not protected from a failure and a bug in the analyzer can make it miss errors. That's why the correctness of the analyzer must be certified in Coq too.

1.2 VERIMAG's contributions

Static analysis by abstract interpretation

The VERIMAG laboratory performs research about program verification using static analysis by abstract interpretation [1]. Static analysis differs from test since, first, the source code

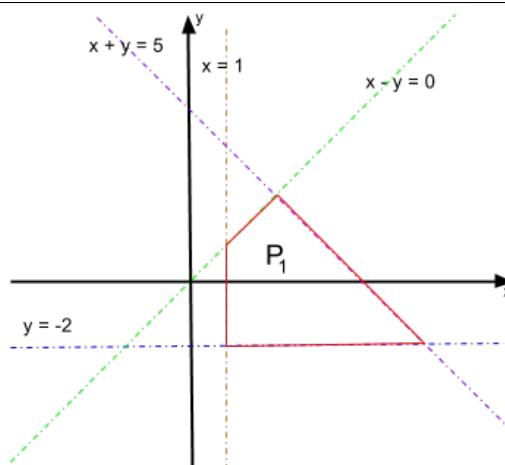


Figure 1: Graphical representation of $P_1 : \{x \geq 1, y \geq -2, x - y \geq 0, x + y \leq 5\}$

is not actually executed, and second, abstract interpretation ensures that no reachable state is omitted. It uses abstract domains (such as intervals, polyhedra, etc.) instead of usual variables to approximate the sets of program states. The effect of each instruction is captured by symbolic computations using abstract domains. Static analysis catches the (potentially infinite) set of possible behaviours of the source code at the price of approximating the reachable states. As a result, some states considered during the analysis are not reachable in practice. When precision is needed, programs can be analyzed using the domain of convex polyhedra which is able to reason about linear relations between program variables.

A certified library for convex polyhedra

A convex polyhedron¹ is defined as a conjunction of linear constraints of the form $\sum_{i=1}^n \lambda_i x_i \leq c$ where $\forall i \in \{1, \dots, n\}, x_i$ is a variable, $\lambda_i \in \mathbb{Q}$ and $c \in \mathbb{Q}$ are constants. For instance, the polyhedron P_1 defined by the system $\{x \geq 1, y \geq -2, x - y \geq 0, x + y \leq 5\}$ defines a geometrical space represented in the plane (Figure 1). A polyhedron which is fully bounded is called a *polytope*.

¹We only deal with convex polyhedra. For readability, we will omit the adjective convex in the following.

Polyhedra are used in static analyzers to automatically discover linear relations between variables of the source code. Those relations help deducing necessary properties to prove the correctness of the program. A Coq library was recently created by Fouihié [2], allowing safe handling of polyhedra operators such as intersection and convex hull, that is the smallest polyhedron containing the union of P_1 and P_2 .

Polyhedra suffer from an ontological limitation: they cannot deal with non-linear relations, *i.e.* expressions containing products of variables. Hence, the analyzer cannot exploit the information on a variable z when encountering a non-linear assignment $z := x * y$. Moreover, after such an assignment, all the previously determined linear constraints containing z do not hold anymore. That's why a linearization technique that approximates $x * y$ by a linear expression is necessary to avoid dramatic loss of precision during the analysis.

Linearization techniques We developed and tested three linearization techniques, but only the last one will be detailed in this paper. The two other ones shall be mentioned as comparison.

The first linearization technique uses the variable intervalization introduced by Miné [3], which consists in replacing some variables of the non-linear expression by intervals to remove products of variables. We implemented and proved this algorithm in Coq.

The second linearization method consists in representing polynomials in the Bernstein basis which allows to deduce a bounding polyhedron from its Bernstein coefficients. We tested it in SAGE (a free open-source mathematics software system similar to maple or mathematica).

The technique that we shall present in this paper is a new linearization method based on Handelman's theorem. Given a starting polyhedron P , Handelman's method consists in using products of constraints of P to obtain a linear over-approximation of the polynomial constraint $f \geq 0$. It requires a parametric simplex that we implemented in Ocaml. The rest of the method was developed in SAGE for tests.

Overview of the paper We shall begin with a brief summary about linearization in Section 2. The linearization method based on Handelman's theorem is described in Section 3. The parametric simplex algorithm, which is used in Handelman's method, is presented in Section 4. We shall give some clues about the Coq certification of this method in Section 5. The three methods are compared in terms of precision in Section 6 where we also discuss about future works.

2 Linearization

The goal of linearization is to approximate non-linear relations by linear ones. In this work we do not consider expressions formed of non-algebraic functions like \sin, \log, \dots ² Our linearization methods only address polynomial expressions containing products of variables. Thus, in this paper f represents a polynomial expression on the variables of the

²We could in principle treat analytic functions by considering their Taylor polynomials.

```

1 int x, y, z;
2 if (x >= 1 && y >= -2 && x >= y && x <= 5 - y)
3 {
4     if (x*x + y*y <= 4)
5         z = y*x;
6     else
7         z = 0;
8 }

```

*Example 2: C program containing two non-linear expressions $x * x + y * y \leq 4$ and $y * x$*

program (x_1, \dots, x_n) . The lines 4 and 5 of Example 2 contain such non-linear relations.

Let us explain how the effect of a guard or an assignment on a polyhedron is computed, and how the linearization process is involved. Without loss of generality, we focus in this paper on the treatment of a guard $f \geq 0$ where f denotes a polynomial in the (x_1, \dots, x_n) variables of the program. For instance the guard $x^2 + y^2 \leq 4$ corresponds to the case $f(x, y) \triangleq 4 - x^2 - y^2$.

The effect of an assignment $x := f$ on a polyhedron P corresponds to the intersection of P with a polyhedron G formed of two inequalities $\tilde{x} - f \geq 0 \wedge f - \tilde{x} \geq 0$ (encoding the equality $\tilde{x} = f$). This computation uses a fresh variable \tilde{x} that is renamed in x after elimination of the old value of x by projection. We denote by $P_{/x}$ the polyhedron P where the variable x has been projected. Formally, the effect of $x := f$ on P is the polyhedron $\left((P \cap G)_{/x} \right) [\tilde{x}/x]$

The effect of a guard $f \geq 0$ on a polyhedron P consists in the intersection of the points of P with the set of points (x_1, \dots, x_n) satisfying the condition $f(x_1, \dots, x_n) \geq 0$. This intersection is not necessarily a polyhedron. Let us denote by \mathcal{P} the set of points after the guard, *i.e.* $\mathcal{P} = P \cap \{ (x_1, \dots, x_n) \mid f(x_1, \dots, x_n) \geq 0 \} = \{ (x_1, \dots, x_n) \in P \mid f(x_1, \dots, x_n) \geq 0 \}$.

When the guard is linear, say $x - y \geq 0$, we simply add the constraint $x - y \geq 0$ to P and obtain a polyhedron. With the polyhedron P_1 from Figure 1, we obtain the polyhedron $\mathcal{P} = P_1 \wedge (x - y \geq 0) = \{ x \geq 1, y \geq -2, x - y \geq 0, x + y \leq 5, x \leq y \}$.

When the guard is not linear, we find its effect on P by computing a convex polyhedron P' such that $\mathcal{P} \subseteq P'$. There is no unique polyhedron P' , any choice of P' satisfying $\mathcal{P} \subseteq P'$ would give a correct approximation.

As an example, the effect of the non-linear guard $\mathcal{G} \triangleq x^2 + y^2 \leq 4$ on the polyhedron P_1 of Figure 1 is represented on Figures 3 and 4. The green polyhedron G of Figure 4 is a linear approximation of \mathcal{G} by an octagon. We obtain the red polyhedron P' by computing the intersection of polyhedra $P_1 \cap G$.

Shape of real-life non linear expressions We tested the linearization techniques on statements taken from the benchmarks *deb1*, based on a satellite control software, and *pa-*

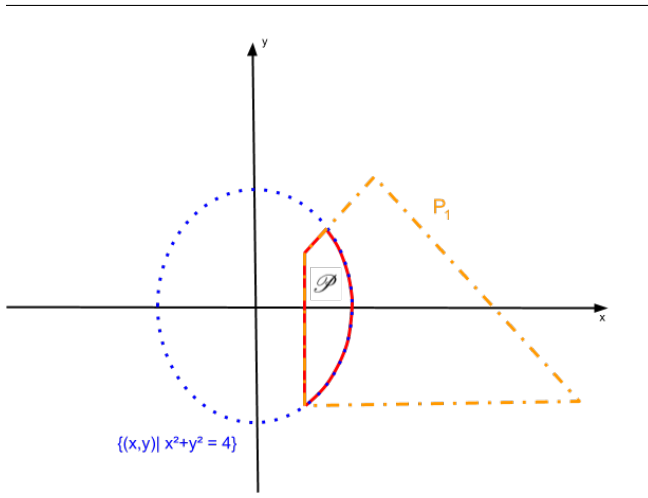


Figure 3: Graphical representations of $P_1 \triangleq \{x \geq 1, y \geq -2, x - y \geq 0, x + y \leq 5\}$ in orange and the guard $\mathcal{G} \triangleq \{(x, y) \mid x^2 + y^2 \leq 4\}$ in blue. The set $\mathcal{P} = P_1 \cap \mathcal{G}$ is represented in red.

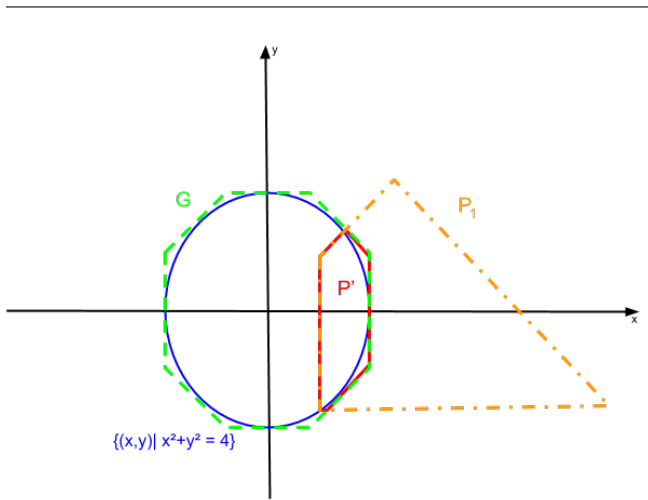


Figure 4: Graphical representations of $P_1 \triangleq \{x \geq 1, y \geq -2, x - y \geq 0, x + y \leq 5\}$ in orange and the guard $\mathcal{G} \triangleq \{(x, y) \mid x^2 + y^2 \leq 4\}$ in blue. A linear over-approximation G' of \mathcal{G} is drawn in green. The approximation P' of $P \cap \mathcal{G}$ is represented in red.

pabench which is a flight control code. In general, polynomials of such programs contain less than four variables and their power rarely exceed two. Indeed, most non-linear expressions appear in computation of Euclidian distances, that's why we encounter square roots as well. As a consequence, the exponential complexity of some algorithms is manageable.

3 Linearization on polytopes using Handelman representation

In this section, we explain how to exploit Handelman's theorem [4] as a new linearization technique. This theorem gives a characterization of positive polynomials over a compact set. This kind of description is usually called a positivstellensatz. As said in the introduction, we focus on the treatment of a guard $f \geq 0$ where f denotes a polynomial in the (x_1, \dots, x_n) variables of the program.

Consider a compact polytope $P = \{(x_1, \dots, x_n) \mid C_1 \geq 0, \dots, C_p \geq 0\}$ where C_i are linear polynomials over (x_1, \dots, x_n) . Suppose P describes the possible values of (x_1, \dots, x_n) at a program point before the guard, we seek a polyhedron that approximates $P \wedge f \geq 0$. We will use as a running example

$$P = \{(x, y) \mid x - 1 \geq 0, y + 2 \geq 0, x - y \geq 0, -x - y + 5 \geq 0\} \text{ and } f = 4 - x^2 - y^2.$$

The affine approximation problem A naive call $P \sqcap f \geq 0$ to the intersection operator of the polyhedral domain would return P not exploiting the constraint $f \geq 0$ which is not affine. Our approximation problem is to find an affine constraint $\alpha_0 + \alpha_1 x_1 + \dots + \alpha_n x_n$, denoted by $\text{aff}(f)$, such that $P \Rightarrow \text{aff}(f) > f$ meaning that $\text{aff}(f)$ bounds f on the polyhedron P . By transitivity of \geq we will conclude that $P \wedge f \geq 0 \Rightarrow P \wedge \text{aff}(f) > 0$. Thus, $P \sqcap \text{aff}(f) > 0$ will be a polyhedral approximation of the program state after the polynomial guard $f > 0$.

3.1 Handelman representation of positive polynomials

Notations Tuples and multi-indexes are typed in boldface. Let $\mathbf{I} = (i_1, \dots, i_p) \in \mathbb{N}^p$ be a multi-index. Let us define the set of Handelman products

$$\mathcal{H}_P = \{C_1^{i_1} \times \dots \times C_p^{i_p} \mid (i_1, \dots, i_p) \in \mathbb{N}^p\}$$

$$\text{where } P \triangleq C_1 \geq 0 \wedge \dots \wedge C_p \geq 0$$

This set contains all products of constraints C_i of P . Given a multi-index $\mathbf{I} = (i_1, \dots, i_p)$, $H^{\mathbf{I}} \triangleq \prod_{j=1}^p C_j^{i_j}$ denotes an element of \mathcal{H}_P . Note that the $H^{\mathbf{I}}$ are positive polynomials on P as products of positive constraints of P .

Example 1. Considering our running example, $H^{(0,2,0,0)} = (y + 2)^2$, $H^{(1,0,1,0)} = (x - 1)(x - y)$ and $H^{(1,0,0,3)} = (-x - y + 5)^3(x - 1)$ belongs to \mathcal{H}_P .

The Handelman representation of a positive polynomial $Q(x)$ on P is

$$Q(x) = \sum_{\mathbf{I} \in \mathbb{N}^p} \underbrace{\lambda_{\mathbf{I}}}_{\geq 0} \underbrace{H^{\mathbf{I}}}_{\geq 0} \text{ with } \lambda_{\mathbf{I}} \in \mathbb{R}^+$$

This representation is used in mathematics as a certificate ensuring that $Q(x)$ is positive on P . Obviously if a polynomial can be written in this form, then it is necessarily positive on P . Handelman's theorem [4], that we summarize here, concerns the non-trivial opposite implication:

Theorem 1 (Handelman's Theorem). *Let $P = \{(x_1, \dots, x_n) \in \mathbb{R}^n \mid C_1 \geq 0, \dots, C_p \geq 0\}$ be a compact polytope where each C_i is a linear polynomial over $x = (x_1, \dots, x_n)$. Let $Q(x)$ be positive polynomial on P . Then there exists $\lambda_I \in \mathbb{R}^+$ and $H^I \in \mathcal{H}_P$ such that*

$$Q(x) = \sum_{I \in \mathbb{N}^p} \lambda_I H^I$$

Usually, the Handelman representation of a polynomial $Q(x)$ is used to determine a constant lower bound of $Q(x)$ on P thanks to Schweighofer's algorithm [5] that focuses on iteratively improving the bound by increasing the degree of the H^I . We shall present here another use of Handelman's theorem: we are not interested in just one (tight) bound but in a polytope wrapping the polynomial $Q(x)$.

3.2 Handelman approximation as a Parametric Linear Optimization Problem

We are looking for an affine constraint $\text{aff}(f)$, such that $\text{aff}(f) \geq f$ on P , which is equivalent to $\text{aff}(f) - f \geq 0$ on P . Then, Handelman's theorem applies:

The polynomial $\text{aff}(f) - f$ which is positive on the polytope P has an Handelman representation as a positive linear combination of products of the constraints of P , i.e.,

$$\text{aff}(f) - f = \sum_{I \in \mathbb{N}^p} \lambda_I H^I, \lambda_I \in \mathbb{R}^+, H^I \in \mathcal{H}_P \quad (1)$$

The relation 1 of Handelman's theorem ensures that there exists some positive combinations of f and some $H^I \in \mathcal{H}_P$ that remove the monomials of total degree >1 and lead to affine forms:

$$\begin{aligned} & \alpha_0 + \alpha_1 x_1 + \dots + \alpha_n x_n \\ & = \text{aff}(f) = 1 \cdot f + \sum_{I \in \mathbb{N}^p} \lambda_I H^I \end{aligned}$$

Note that the polynomials of \mathcal{H}_P are generators of the positive polynomials on P but they do not form a basis. Indeed, it is possible to have one $H \in \mathcal{H}_P$ being a positive linear combination of other elements of \mathcal{H}_P .

Example 2. Consider $P = \{(x, y) \mid x \geq 0, y \geq 0, x - y \geq 0, x + y \geq 0\}$.

Then, $H_{(2,0,0,0)} \triangleq x^2$, $H_{(1,1,0,0)} = xy$, $H_{(0,2,0,0)} = y^2$, $H_{(0,0,0,2)} = (x+y)^2$ belongs to Handelman products and they are not independent.

Indeed, $H_{(0,0,0,2)} = x^2 + 2xy + y^2 = H_{(2,0,0,0)} + 2H_{(1,1,0,0)} + H_{(0,2,0,0)}$

As a consequence, a positive polynomial $\text{aff}(f) - f$ can have several Handelman representations, even on a given set of Handelman products. Actually, we exploit the non-uniqueness of representation to get a precise approximation

of the guard: we look for many affine constraints $\text{aff}(f)$ that bound f on P . Their conjunction forms a polyhedron that over-approximates f on P .

We now explain how the determination of all affine constraints bounding f can be expressed as a Parametric Linear Optimization Problem (PLOP).

Notations Given a multi-index I and n variables x_1, \dots, x_n , let x^I be the monomial $x_1^{i_1} \times \dots \times x_n^{i_n}$. We define the total degree of the monomial x^I as $\text{deg}(I) = \sum_{j=1}^n i_j$. For monomial of degree ≤ 1 we simply write $\alpha_i x_i$ instead of $\alpha_I x^I$ when $I = (0, \dots, 0, 1, 0, \dots, 0)$ with 1 in its i^{th} coordinate. With this settings, the relation 1 can be rephrased:

For some choice of λ_I , the coefficient α_I of the monomial x^I in the polynomial $f + \sum_{I \in \mathbb{N}^p} \lambda_I H^I$ is null for all multi-index I with $\text{deg}(I) > 1$.

Now, let $d_f \in \mathbb{N}^n$ be the maximal degree of the monomials of f . We restrict our search to finding a Handelman representation of $\text{aff}(f) - f$ on the subset $\{H_1, \dots, H_q\}$ of all the Handelman products of degree $\leq d_f$, instead of the whole set \mathcal{H}_P . If we fail with monomials of degree $\leq d$ we increase d . Handelman's theorem ensures that we will eventually succeed.

Example 3. With $f = 4 - x^2 - y^2$, $d_f = 2$. Therefore, we shall consider the 15 following Handelman products:

$$\begin{aligned} H_1 &= H_{(0,0,0,0)} = 1 \\ H_2 &= H_{(1,0,0,0)} = x - 1 \\ H_3 &= H_{(0,1,0,0)} = y + 2 \\ H_4 &= H_{(0,0,1,0)} = x - y \\ H_5 &= H_{(0,0,0,1)} = -x - y + 5 \\ H_6 &= H_{(2,0,0,0)} = (x - 1)^2 \\ H_7 &= H_{(0,2,0,0)} = (y + 2)^2 \\ H_8 &= H_{(0,0,2,0)} = (x - y)^2 \\ H_9 &= H_{(0,0,0,2)} = (-x - y + 5)^2 \\ H_{10} &= H_{(1,1,0,0)} = (x - 1)(y + 2) \\ H_{11} &= H_{(1,0,1,0)} = (x - 1)(x - y) \\ H_{12} &= H_{(1,0,0,1)} = (x - 1)(-x - y + 5) \\ H_{13} &= H_{(0,1,1,0)} = (y + 2)(x - y) \\ H_{14} &= H_{(0,1,0,1)} = (y + 2)(-x - y + 5) \\ H_{15} &= H_{(0,0,1,1)} = (x - y)(-x - y + 5) \end{aligned}$$

With the restriction to $\{H_1, \dots, H_q\}$, finding the λ_I can be formulated as a PLOP.

The relation 1 on $\{H_1, \dots, H_q\}$ amounts to find positive $\lambda_1, \dots, \lambda_q \in \mathbb{R}^+$ such that

$$\begin{aligned} \text{aff}(f) &= 1 \cdot f + \sum_{i=1}^q \lambda_i H_i \\ &\parallel \\ & \lambda \cdot (f, H_1, \dots, H_q)^\top \\ &\parallel \\ \alpha_0 + \alpha_1 x_1 + \dots + \alpha_n x_n & \lambda \cdot \mathcal{H}_f \cdot \mathbf{m}^\top \\ &\parallel \\ \mathbf{m} \cdot (\alpha_0, \dots, \alpha_n, 0, \dots, 0)^\top &= \mathbf{m} \cdot \mathcal{H}_f^\top \cdot \lambda^\top \end{aligned}$$

where:

- \mathcal{H}_f is the matrix of the coefficients of f and the H_i in the canonical basis of monomials of degree $\leq d_f$ denoted by $\mathbf{m} = (1, x_1, \dots, x_n, \mathbf{x}^{d_1}, \dots, \mathbf{x}^{d_f})$
- the vector $\boldsymbol{\lambda} = (\lambda_f, \lambda_1, \dots, \lambda_q) = (1, \lambda_1, \dots, \lambda_q)$ characterizes the Handelman's positive combination of f and the H_i
- we associated a coefficient $\lambda_f = 1$ to f just to get convenient notations.

Finally, the problem can be rephrased as finding $\boldsymbol{\lambda} \in \{1\} \times (\mathbb{R}^+)^q$ such that

$$\mathcal{H}_f^\top \cdot \boldsymbol{\lambda}^\top = (\alpha_0, \dots, \alpha_n, 0, \dots, 0)^\top$$

The result of the matrix-vector product $\mathcal{H}_f^\top \boldsymbol{\lambda}^\top$ is a vector $\boldsymbol{\alpha} \triangleq (\alpha_0, \alpha_1, \dots, \alpha_n, \alpha_{d_1}, \dots, \alpha_{d_f})$ that represents the constraint $\alpha_0 + \alpha_1 x_1 + \dots + \alpha_n x_n + \sum_{I \leq d_f} \alpha_I \mathbf{x}^I$ in the \mathbf{m} basis. Since we seek an affine constraint $\text{aff}(f)$ we are interested in finding $\boldsymbol{\lambda}$ such that $\mathcal{H}_f^\top \boldsymbol{\lambda}^\top = (\alpha_0, \dots, \alpha_n, 0, \dots, 0)^\top$. Each such $\boldsymbol{\lambda}$ gives an affine constraint $\text{aff}(f)$ that bounds f on P . Therefore, the conjunction of all constraints $\text{aff}(f) \geq 0$ form a polyhedron \mathcal{A}_f that approximates the guard $f \geq 0$ on P .

Example 4. The transposed matrix \mathcal{H}_f^\top of f and our 15 Handelman products with respect to the basis $\mathbf{m} = (1, x, y, xy, x^2, y^2)$ is shown on Figure 5.

With $\lambda_f = \lambda_6 = \lambda_7 = 1$ and every other $\lambda_i = 0$, we obtain

$$\boldsymbol{\alpha}^\top = \mathcal{H}_f^\top \boldsymbol{\lambda}^\top = \begin{pmatrix} 9 \\ -2 \\ 4 \\ 0 \\ 0 \\ 0 \end{pmatrix} \begin{matrix} 1 \\ x \\ y \\ xy \\ x^2 \\ y^2 \end{matrix}$$

Thus, $\text{aff}(f) = -2x + 4y + 9$ is a constraint that bounds f on P , as shown on the Figure 6(a). Indeed, we can see that the plane $-2x + 4y + 9$ is above the polynomial f . Figure 6(b) shows the approximation of f that we obtain with the constraint $-2x + 4y + 9 \geq 0$.

In order to obtain a better approximation, we need to find others affine approximations bounding f on P . For instance, with $\lambda_f = \lambda_8 = 1$, $\lambda_{11} = \lambda_5 = 2$, and every other $\lambda_i = 0$, $\boldsymbol{\alpha}^\top = (21, -2, -8, 0, 0)$. The Figure 7 shows the approximation of f with the two constraints $-2x + 4y + 9 \geq 0$ and $-2x - 8y + 21 \geq 0$.

The Parametric Linear Optimization Problem Finding all and the tightest approximations $\text{aff}(f)$ that bounds f on P can now be expressed as the following PLOP which can be solved using a Parametric Simplex in the spirit of [6].

Given a set $\{H_1, \dots, H_q\} \subseteq \mathcal{H}_P$ of Handelman products,

minimize $\text{aff}(f)$, that is, $\alpha_0 + \alpha_1 x_1 + \dots + \alpha_n x_n$
under the constraints

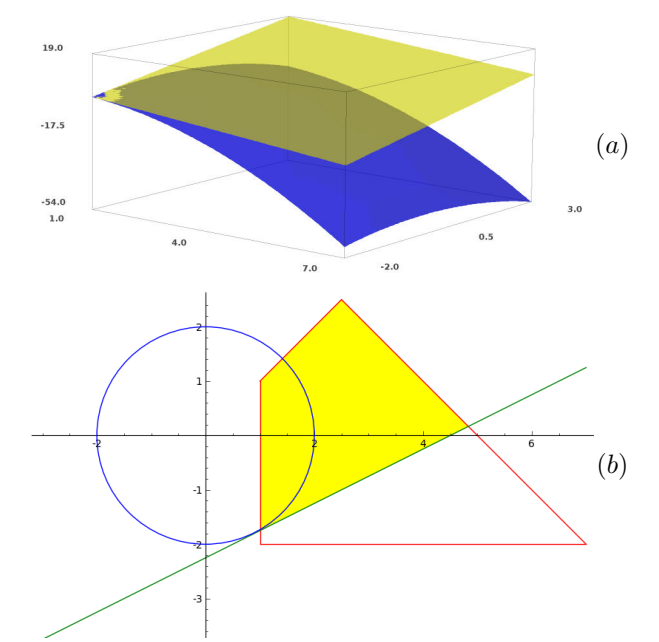


Figure 6: (a): The surface $z = f(x, y) \triangleq -x^2 - y^2 + 4$ is in blue, the plane $z = \text{aff}(f)(x, y) \triangleq -2x + 4y + 9$ in yellow. (b): The representation of the polyhedron $P \triangleq \{(x, y) \mid x - 1 \geq 0, y + 2 \geq 0, x - y \geq 0, -x - y + 5 \geq 0\}$ is in red and the area of the circle in blue represents $\{(x, y) \mid f(x, y) \geq 0\}$. The green line is the constraint $-2x + 4y + 9 = 0$. The yellow area is the polyhedron $P \cap \{(x, y) \mid -2x + 4y + 9 \geq 0\}$ that over-approximates the conjunction $P \wedge f \geq 0$.

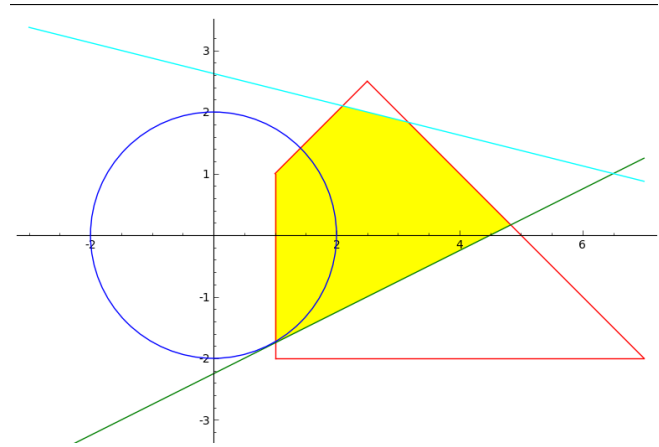


Figure 7: The material is that of Figure 6 augmented with a cyan line which corresponds to the constraint $-2x - 8y + 21 = 0$. The yellow area is the polyhedron $P \cap \{(x, y) \mid -2x + 4y + 9 \geq 0, -2x - 8y + 21 \geq 0\}$ that over-approximates the conjunction $P \wedge f \geq 0$.

	f	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8	H_9	H_{10}	H_{11}	H_{12}	H_{13}	H_{14}	H_{15}
1	4	1	-1	2	0	5	1	4	0	25	-2	0	-5	0	10	0
x	0	0	1	0	1	-1	-2	0	0	-10	2	-1	6	2	-2	5
y	0	0	0	1	-1	-1	0	4	0	-10	-1	1	1	-2	3	-5
xy	0	0	0	0	0	0	0	0	-2	2	1	-1	-1	1	-1	0
x^2	-1	0	0	0	0	0	1	0	1	1	0	1	-1	0	0	-1
y^2	-1	0	0	0	0	0	0	1	1	1	0	0	0	-1	-1	1

Figure 5: Transposed matrix \mathcal{H}_f^\top of f and the 15 Handelman products of Example 3 with respect to the basis $\mathbf{m} = (1, x, y, xy, x^2, y^2)$

$$\begin{cases} \mathcal{H}_f^\top(\lambda_f, \lambda_1, \dots, \lambda_q)^\top = (\alpha_0, \dots, \alpha_n, 0, \dots, 0)^\top \\ \lambda_f = 1 \\ \lambda_i \geq 0, i = 1..q \end{cases} \quad (\text{PLOP 1})$$

where

- The $\lambda_1, \dots, \lambda_q$ are the variables of the PLOP.
- The $\alpha_0, \dots, \alpha_n$ are kept for the sake of presentation ; in practice they are substituted by their expression issued from $\mathcal{H}_f^\top \lambda^\top$.
- The x_1, \dots, x_n are the parameters of the PLOP.

Remark that each instantiation (x_1, \dots, x_n) of the parameters defines a standard Linear Optimization Problem (LP) which can be solved by the simplex algorithm, providing the optimum associated to the given parameters. In the next section, we describe a parametric simplex able to find every optimal solutions of PLOP 1 as a function of the parameters (x_1, \dots, x_n) .

4 The parametric simplex algorithm

The simplex algorithm is used to find the optimal solution to a LP of the form

$$\begin{aligned} & \text{minimize } \mathbf{c}^\top \mathbf{t} \\ & \text{under the constraints} \\ & A\mathbf{t} \leq \mathbf{b} \\ & \mathbf{t} \geq 0 \end{aligned} \quad (\text{LP 1})$$

where $\mathbf{t} = (t_1, \dots, t_q)$ are decision variables, $\mathbf{c} = (c_1, \dots, c_q)$ is the cost, $A \in M_{m,n}(\mathbb{R})$ and $\mathbf{b} \in \mathbb{R}^m$. In this section, we shall describe the algorithm of a parametric simplex, *i.e.* a simplex able to find the optimal solution of linear programming problems whose objective contains parameters. Instead of $\mathbf{c}^\top \mathbf{t}$, the objective of such problems is a linear combination of the decision variables t_1, \dots, t_q where coefficients are linear forms of the parameters β_1, \dots, β_n .

Before entering into details of the parametric simplex algorithm, we shall begin by running an example of the standard simplex.

Classical simplex execution

As explained in [7], the principle of the simplex method is to move from a feasible solution of the problem to another one that improves the objective value. Iterating this process shall lead to the optimal solution.

The example that we shall run further is the following one:

minimize $-2t_1 + t_2$
under the constraints

$$\begin{aligned} t_1 + t_2 &\leq 5 \\ -t_1 &\leq 1 \\ -t_2 &\leq 2 \\ -t_1 + t_2 &\leq 0 \\ t_1, t_2 &\geq 0 \end{aligned} \quad (\text{LP 2})$$

Initialization: finding a feasible solution The first step is to add *slack variables* s_i to the problem. We add one slack variable s_i for each constraint to denote the slack between its left-hand side and its right-hand side – the constant. For instance, from the first constraint $t_1 + t_2 \leq 5$ of LP 2, we define $s_1 = 5 - t_1 - t_2$. The constraint can now be replaced by $s_1 \geq 0$. Indeed, $t_1 + t_2 \leq 5 \Leftrightarrow s_1 \geq 0$. Repeating this for all constraints, we obtain the following definitions:

$$\begin{aligned} s_1 &= 5 - t_1 - t_2 \\ s_2 &= 1 + t_1 \\ s_3 &= 2 + t_2 \\ s_4 &= t_1 - t_2 \end{aligned} \quad (\text{DEF})$$

The problem LP 2 can therefore be rewritten into

minimize $-2t_1 + t_2$
under the constraints

$$\begin{aligned} s_1 &= 5 - t_1 - t_2 \\ s_2 &= 1 + t_1 \\ s_3 &= 2 + t_2 \\ s_4 &= t_1 - t_2 \\ s_1, s_2, s_3, s_4, t_1, t_2 &\geq 0 \end{aligned} \quad (\text{LP 3})$$

Vocabulary The system composed of the equations and the objective function is called a dictionary. The variables appearing on the left-hand side of the equalities are called non-zero variables, or *basic* variables. The set of these variables is a *basis*. The variables appearing on the right-hand side are null variables or *nonbasic*.

Note that every feasible solution (t_1, t_2) of LP 2 leads to a unique feasible solution of LP 3 thanks to the equations of DEF. Reciprocally, from a feasible solution of LP 3, a feasible solution of LP 2 can be deduced by removing the slack variables.

A starting feasible solution can now be found easily in our example, setting t_1 and t_2 at 0. We obtain the solution ($t_1 = 0, t_2 = 0, s_1 = 5, s_2 = 1, s_3 = 2, s_4 = 0$). The value of the objective associated to this solution is $-2*0+0 = 0$. If LP 2 is feasible but the solution obtained by setting t_1 and t_2 at 0 is not (*i.e.* $s_i < 0$ for some i), there exists an *auxiliary* problem able to find a correct dictionary. It is detailed in [7].

Finding better solutions Now we have found a feasible solution, the next step is to find another one that has a better objective value. The idea is to pick a variable that has a negative coefficient in the objective and increase it as much as possible. In LP 3, the objective is to minimize $-2t_1 + t_2$. By increasing t_1 , the objective decreases and we obtain a better solution. Hence we increase t_1 while the solution remains feasible. We must now find the constraint that restrain the value of t_1 the most. $s_1 = 5 - t_1 - t_2$ gives an upper bound for t_1 that is 5. Indeed, if $t_1 > 5$, then $s_1 < 0$, which leads to an unfeasible solution. $s_2 = 1 + t_1$ and $s_4 = t_1 - t_2$ give lower bounds on t_1 , and $s_3 = 2 + t_2$ does not speak about t_1 . Therefore, the first constraint is the limiting one, and setting $t_1 = 5$, we obtain the solution ($t_1 = 5, t_2 = 0, s_1 = 0, s_2 = 6, s_3 = 2, s_4 = 5$) then the objective becomes -10. Note that the value of t_1 is no longer zero and that on the contrary, s_1 has become null. That's why s_1 must leave the basis while t_1 enters it (this switch between two variables is called a *pivot*). It means that we must change our dictionary LP 3 so that the basic variables t_1, s_1, s_2, s_4 are expressed in terms of the non-basic ones t_2, s_3 . From $s_1 = 5 - t_1 - t_2$ we deduce $t_1 = -s_1 - t_2 + 5$, thus we obtain the following dictionary:

$$\begin{aligned} & \text{minimize } 3t_2 + 2s_1 + 10 \\ & \text{under the constraints} \\ & t_1 = -s_1 - t_2 + 5 \\ & s_2 = -s_1 - t_2 + 6 \\ & s_3 = 2 + t_2 \\ & s_4 = -2t_2 - s_1 + 5 \\ & s_1, s_2, s_3, s_4, t_1, t_2 \geq 0 \end{aligned} \quad (\text{LP 4})$$

We can see that there is no variable in the objective associated to a negative coefficient anymore, it means that the optimal solution has been found.

Parametric simplex execution

To illustrate the parametric simplex, let us replace the coefficients of t_1 and t_2 in LP 2 by parameters β_1 and β_2 :

$$\begin{aligned} & \text{minimize } \beta_1 t_1 + \beta_2 t_2 \\ & \text{under the constraints} \\ & t_1 + t_2 \leq 5 \\ & -t_1 \leq 1 \\ & -t_2 \leq 2 \\ & -t_1 + t_2 \leq 0 \\ & t_1, t_2 \geq 0 \end{aligned} \quad (\text{PLOP 2})$$

The initialization step is the same as before, and we obtain the dictionary

minimize $\beta_1 t_1 + \beta_2 t_2$
under the constraints

$$\begin{aligned} s_1 &= 5 - t_1 - t_2 \\ s_2 &= 1 + t_1 \\ s_3 &= 2 + t_2 \\ s_4 &= t_1 - t_2 \\ s_1, s_2, s_3, s_4, t_1, t_2 &\geq 0 \end{aligned} \quad (\text{PLOP 3})$$

Once we have a feasible dictionary, the principle is to determine the optimal solution depending on the sign of the parameters β_i . Indeed, to improve the objective in PLOP 3, since we look for a variable with a negative coefficient in the objective to perform a pivot, we need to know the sign of the coefficient of t_1 , which is β_1 . First, we shall assume $\beta_1 < 0$ and obtain a branch leading to one or several optimal solutions depending on the sign of β_2 . Second, we shall assume $\beta_1 \geq 0$ and get a second branch. Let us execute the algorithm for the branch $\beta_1 < 0$ to clearly see what type of results appear. The variable leaving the basis is t_1 , hence as in the standard algorithm, we must find the entering variable by looking for the constraint bounding the value of t_1 the most. Again, this limiting constraint is $s_1 = 5 - t_1 - t_2$ which prevents t_1 to exceed 5. The pivot of t_1 and s_1 gives the solution ($t_1 = 5, t_2 = 0, s_1 = 0, s_2 = 6, s_3 = 2, s_4 = 5$) and the objective value is now $5\beta_1$. The associated dictionary is the following one:

$$\begin{aligned} & \text{minimize } -\beta_1 s_1 + (\beta_2 - \beta_1)t_2 + 5\beta_1 \\ & \text{under the constraints} \\ & t_1 = -s_1 - t_2 + 5 \\ & s_2 = -s_1 - t_2 + 6 \\ & s_3 = 2 + t_2 \\ & s_4 = -2t_2 - s_1 + 5 \\ & s_1, s_2, s_3, s_4, t_1, t_2 \geq 0 \end{aligned} \quad (\text{PLOP 4})$$

For the next iteration, we look again for a variable whose coefficient is negative in the objective. $-\beta_1$ cannot be negative because we made the assumption $\beta_1 < 0$. However, none of our assumptions prevents $\beta_2 - \beta_1$ from being negative. Again, we will create two branches where the first one shall assume $\beta_1 < 0 \wedge \beta_2 - \beta_1 < 0$ whereas the other one shall assume $\beta_1 < 0 \wedge \beta_2 - \beta_1 \geq 0$. In the first of these two branches, the variable leaving the basis is t_2 , and the limiting constraint is $s_4 = -2t_2 - s_1 + 5$, thus the entering variable is s_4 and the maximum value for t_2 is $\frac{5}{2}$. Hence the new feasible solution is ($t_1 = \frac{5}{2}, t_2 = \frac{5}{2}, s_1 = 0, s_2 = \frac{7}{2}, s_3 = \frac{9}{2}, s_4 = 0$), the objective value is $\frac{5\beta_1 + 5\beta_2}{2}$ and we end with the following dictionary:

$$\text{minimize } \frac{-\beta_1 - \beta_2}{2} s_1 + \frac{\beta_1 - \beta_2}{2} s_4 + \frac{5\beta_1 + 5\beta_2}{2}$$

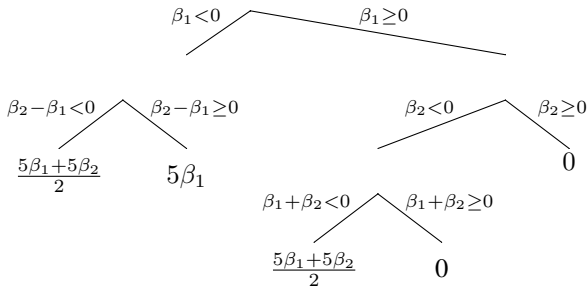


Figure 8: Optimal solutions of the problem PLOP 2 in terms of β_1 and β_2

under the constraints

$$\begin{aligned}
 t_1 &= -\frac{s_1}{2} + \frac{s_4}{2} + \frac{5}{2} \\
 s_2 &= \frac{s_4}{2} - \frac{s_1}{2} + \frac{7}{2} \\
 s_3 &= -\frac{s_4}{2} - \frac{s_1}{2} + \frac{9}{2} \\
 t_2 &= -\frac{s_4}{2} - \frac{s_1}{2} + \frac{5}{2} \\
 s_1, s_2, s_3, s_4, t_1, t_2 &\geq 0
 \end{aligned} \tag{PLOP 5}$$

Recall that our current context is $\beta_1 < 0 \wedge \beta_2 - \beta_1 < 0$. With these assumptions, neither $-\frac{\beta_1 - \beta_2}{2}$ nor $\frac{\beta_1 - \beta_2}{2}$ can be negative. Thus, the optimal has been found and the algorithm stops **for this context**. For the second branch, going back to PLOP 4 and assuming $\beta_1 < 0 \wedge \beta_2 - \beta_1 \geq 0$, there is no remaining variable with a negative coefficient in the objective. Thus the optimal for this context is $5\beta_1$. The region where $\beta_1 < 0$ has been fully explored and leads to two different optimal solutions depending on β_2 . At this point, the space where $\beta_1 \geq 0$ still needs to be traveled, so the next iteration starts from PLOP 2 with the assumption $\beta_1 \geq 0$.

Figure 8 summarizes the optimal solutions depending on the different contexts encountered throughout the algorithm.

Solutions of the parametric simplex Let us denote by z^* the optimal solution of a PLOP. As explained earlier, z^* is a function of the parameters β_1, \dots, β_n and is divided into regions. For instance, the optimum solution of PLOP 2 is the function

$$z^*(\beta_1, \beta_2) = \begin{cases} \frac{5\beta_1 + 5\beta_2}{2} & \text{if } \beta_1 < 0 \wedge \beta_2 - \beta_1 < 0 \\ 5\beta_1 & \text{if } \beta_1 < 0 \wedge \beta_2 - \beta_1 \geq 0 \\ \frac{5\beta_1 + 5\beta_2}{2} & \text{if } \beta_1 \geq 0 \wedge \beta_2 < 0 \wedge \beta_1 + \beta_2 < 0 \\ 0 & \text{if } \beta_1 \geq 0 \wedge \beta_2 < 0 \wedge \beta_1 + \beta_2 \geq 0 \\ 0 & \text{if } \beta_1 \geq 0 \wedge \beta_2 \geq 0 \end{cases}$$

z^* is a continuous, piecewise affine and concave function[8]. Note that z^* can be equal to the same affine function in several disjoint regions. For instance in both spaces $\beta_1 <$

$0 \wedge \beta_2 - \beta_1 < 0$ and $\beta_1 \geq 0 \wedge \beta_2 < 0 \wedge \beta_1 + \beta_2 < 0$, $z^*(\beta_1, \beta_2) = \frac{5\beta_1 + 5\beta_2}{2}$. Thanks to these properties, the following one can be shown:

Property 1. Let $f_a : \mathbb{R}^n \rightarrow \mathbb{R}$ be an affine function, if $z^* = f_a$ on $I \subset \mathbb{R}^n$ with non-empty interior, if $z^* = f_a$ on $J \subset \mathbb{R}^n$ such that $J \cap I = \emptyset$, then $z^* = f_a$ on the convex hull of $I \cup J$.

Property 1 could be used during the parametric simplex algorithm to avoid the creation of a new branch. For instance with one parameter β , knowing that

$$z^*(\beta) = \begin{cases} 2\beta - 1 & \text{if } \beta \in [0, 2] \\ 2\beta - 1 = 5 & \text{if } \beta = 3 \end{cases}$$

we can directly deduce that $z^*(\beta) = 2\beta - 1$ for all $\beta \in [0, 3]$.

Application to Handelman's linearization To solve PLOP 1, we execute the parametric simplex described previously. As explained earlier, the decision variables are the λ_i while the parameters are the x_i . Moreover, we specify as initial context the starting polyhedron P , meaning that we only consider $(x_1, \dots, x_n) \in P$. We obtain a tree with linear forms on x_i at leaves. Each of these linear forms is a constraint over-approximating $P \wedge f \geq 0$. As said previously, it is possible to find the same constraint at different leaves, and some constraints can be redundant compared to others. Hence, the over-approximating polyhedron P' is defined as the conjunction of all these constraints. Figure 9 shows P' for our guiding example, where we can clearly see that one green line does not constraint P' , meaning that it is redundant.

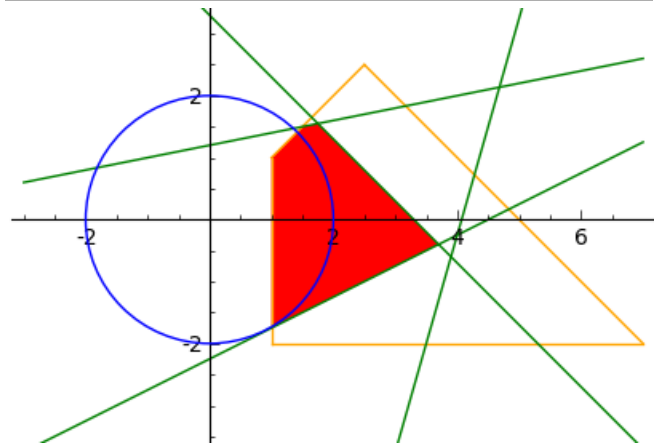


Figure 9: The polyhedron $P = \{ (x, y) \mid x - 1 \geq 0, y + 2 \geq 0, x - y \geq 0, -x - y + 5 \geq 0 \}$ is drawn in orange. The area delimited by the blue circle represents the guard $\{ (x, y) \mid x^2 + y^2 \leq 4 \}$. The red surface is P' , the approximation of $\{ (x, y) \mid x^2 + y^2 \leq 4 \} \cap P$. The green lines are the constraints returned by the parametric simplex

5 Toward certification in Coq

In this part, we give intuitions about the certification in Coq of the linearization using Handelman.

We want to prove that $\text{aff}(f) > f$, or equivalently that $\text{aff}(f) - f > 0$, on the polytope P . Given indexes $(i_{1,1}, \dots, i_{1,p}), \dots, (i_{q,1}, \dots, i_{q,p})$ and the coefficients $\lambda_1, \dots, \lambda_q$, we have to check in Coq that:

- (1) $f + \sum_{j=1}^q \lambda_j C_1^{i_{j,1}} \dots C_p^{i_{j,p}}$ is linear. This can be done by expanding the polynomial and looking at the coefficient of monomials of degree > 1 .
- (2) The product of positive polynomials is positive. This can be proven in Coq once for all. We obtain finally that $f + \sum_{j=1}^q \lambda_j C_1^{i_{j,1}} \dots C_p^{i_{j,p}}$ is a linear over-approximation of f .

The computation of $\text{aff}(f)$ does not have to be done in Coq. Thus, $(i_{1,1}, \dots, i_{1,p}), \dots, (i_{q,1}, \dots, i_{q,p})$ and $\lambda_1, \dots, \lambda_q$ are used as a certificate.

6 Comparison of the linearization methods and future work

We implemented and proved in Coq the linearization algorithm based on intervalization. The Bernstein’s method was implemented in SAGE. We developed the parametric simplex algorithm in Ocaml and the rest of Handelman’s algorithm was done in SAGE. In order to compare the three methods, we realised a simple analyzer in SAGE. It is able to handle C programs containing guards, assignments and if-then-else but no function call nor loop. Given a starting polyhedron P and a list of statements s , the analyzer computes the effect of s on P with the three techniques. The intervalization algorithm is performed by an Ocaml program obtained by automatic extraction from our Coq development. This Ocaml code is called by the SAGE script. We are then able to measure and compare the resulting polyhedra volume using an existing SAGE library.

Comparison We realized experiments on statements taken from the satellite code. In general, intervalization is the fastest but the less accurate of the three methods. Bernstein’s method can be as accurate as needed, but at the price of an high algorithmic cost. Handelman’s method is about as accurate as the Bernstein one. Up to now, it is the most expensive method, mainly because it is new, we implemented it in a naive way and no attention was paid to improve the computations as it has been done, for decades, for Bernstein approximations.

The Bernstein approximation relies on the interval where ranges each variable. Extracting the interval of a variable from a polyhedron requires to solve two linear optimization problems to get the maximum and minimum value of the variable in the polyhedron. This overhead is avoided in Handelman’s method which reasons directly on the constraints of the polyhedron. Hence, the Bernstein’s method is convenient when the polyhedron is in fact an hypercube – that is the product of the interval of each variable – whereas the Handelman’s method is promising at program point associated with a general polyhedron. Specifically, we think that

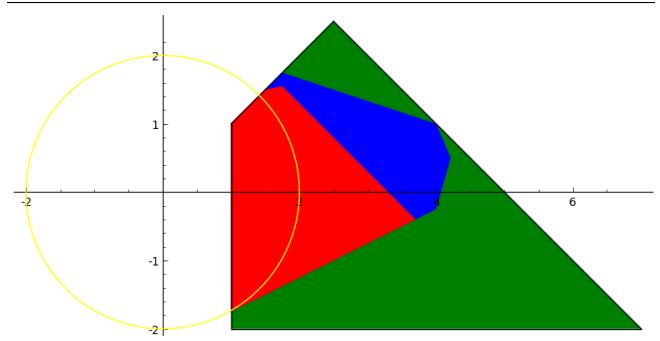


Figure 10: Representation of the effect of the guard $\{(x, y) \mid x^2 + y^2 \leq 4\}$ (yellow circle) on the polyhedron $\{(x, y) \mid x - 1 \geq 0, y + 2 \geq 0, x - y \geq 0, -x - y + 5 \geq 0\}$ (black outline). The green surface is the result of the linearization using intervalization. The blue one is the result of Bernstein’s linearization. The red one is the linearization with the Handelman’s method.

Handelman’s method can be more suitable in terms of precision, even in complexity, in case of successive linearizations. Indeed, where the Bernstein’s method stacks approximation errors at each new linearization, the Handelman one does not degrade. Moreover, in order to certify these methods, the Bernstein one requires to switch from the polyhedron representation using vertices to the polyhedron representation by linear constraints, which is not the case for Handelman.

In practice, the three linearization methods can be combined: analysis is an iterative process that switches to finer methods when the cheapest ones failed to prove correctness of the program. We can imagine starting an analysis with intervals which are cheap and deal with non-linear expressions. Then, switching to the domain of polyhedra if more precision is required. This second phase can reuse the intervals computed by the first one and apply intervalization or Bernstein’s linearization without paying the overhead of extracting intervals. This time the analysis associates polyhedra to program points. Then, to gain more precision, a third phase can run the analysis with Handelman’s linearization. Other combinations are possible and Handelman can be directly used at any phase since a product of bounded interval is a special case of polytope, called an hypercube.

We show on Figure 10 the results of the three methods to approximate the guard $\{(x, y) \mid x^2 + y^2 \leq 4\}$ on $P \triangleq \{(x, y) \mid x - 1 \geq 0, y + 2 \geq 0, x - y \geq 0, -x - y + 5 \geq 0\}$. We can see that intervalization is not precise enough to approximate the guard. Indeed, the resulting polyhedron is the same as the initial one, and the guard does not add any information in the analysis.

We compute Bernstein’s method without any interval splitting or degree elevation. Even without any refinement process, Bernstein is more accurate than intervalization but slower.

Handelman’s polyhedron is the most precise of the three techniques in this example. We chose as subset $\{H_1, \dots, H_q\}$ the 15 possible products of constraints of P of degree ≤ 2 , meaning that we are faced with a 15-variables LP. Industrial linear solvers are able to deal with hundreds of

variables, but this is obviously the shortfall of Handelman’s linearization.

Future work Along the document we identified several points that still need work prior to the integration of our linearization methods in the VERASCO analyzer. We review them quickly and sketch direction of improvement.

Unbounded polyhedron Up to now, we considered only the linearization with Bernstein or Handelman on polytopes. Indeed, the Bernstein basis is defined only on $[0, 1]^l$ and Handelman’s theorem applies on a compact polyhedron. However, we do not necessarily have full bounds on each variable of the polynomial expression f , therefore we need to be able to manipulate unbounded polyhedra. This case has already been treated for intervalization. There exists a method to handle partially unbounded intervals during the transformation of a polynomial into the Bernstein basis [9]. It is based on a bijective transformation of the partially unbounded polynomial to a bounded one which has the same sign. Handelman’s theorem addresses only compact polyhedra. Obviously, if a polynomial grows non-linearly in the unbounded direction, it cannot be bounded by any affine function. However, the growth of f with respect to the unbounded variable can be bounded, *e.g.* $f(x) = 1 - x^2$. The adaptation of Handelman’s method to this case is an open question.

Certification in Coq Recall that as a part of VERASCO project, the linearization techniques need to be certified. We have done this for the intervalization algorithm, but it still requires to be included in the VERASCO analyzer. The two other linearization methods have not been certified yet. We shall certify them in Coq using certificates, meaning that parts of the computations are done outside of Coq. We do not plan to implement the Handelman’s method in Coq but to use certificates that drive the verification in Coq that $\text{aff}(f)$ is an affine constraint and an approximation of f .

Handelman’s Linearization The main improvements of Handelman’s linearization that we shall work on are:

- **The choice of the subset** $\{H_1, \dots, H_q\}$. On the one hand, considering a lot of H_i allows lots of Handelman’s representations, therefore an improved accuracy. On the other hand, each new H_i adds a variable λ_i in the simplex. In order to minimize the number of H_i to consider, starting with a small subset $\{H_1, \dots, H_q\}$, we could imagine an incremental approach that adds new H_i when no solution is found. We must pay attention to the algorithm in order to exploit the computations of the previous attempt.
- **The size of the output tree returned by parametric simplex algorithm** As explained in Section 4, the same optimal solution can appear several times in the tree. Thanks to Property 1 defined in Section 4, we could sometimes avoid some branch creation, therefore decrease the algorithm execution time and obtain a smaller number of regions. Even so, it would require to fastly

determine if a region belongs to the convex hull of two other regions.

Experiments in the large Once all the three linearization techniques are integrated in the VERASCO analyzer, series of code benchmarks and testings shall be realised. Then, combination of the three methods shall be adjusted, as well as the heuristics we are using. Indeed, we have to adapt our linearization techniques depending on the type of code we want to analyse. For instance, the satellite code contains lots of sums of squares, thus we must adjust our heuristics to be more effective on this kind of expressions. Similarly, if an expression appears many times in the program, we should pay a special attention to linearize it precisely.

References

- [1] P. Cousot and R. Cousot, “Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints,” in Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, pp. 238–252, ACM, 1977.
- [2] A. Foulhé, D. Monniaux, and M. Périn, “Efficient generation of correctness certificates for the abstract domain of polyhedra,” in SAS2013, 2013.
- [3] A. Miné, “Symbolic methods to enhance the precision of numerical abstract domains,” the 7th International Conference on Verification, Model Checking and Abstract Interpretation, vol. 3855, pp. 348–363, January 2006.
- [4] D. Handelman, “Representing polynomials by positive linear functions on compact convex polyhedra,” Pac. J. Math, vol. 132, no. 1, p. 35–62, 1988.
- [5] Markus Schweighofer, “An algorithmic approach to schmüdgen’s positivstellensatz,” Elsevier Preprint, June 2001.
- [6] P. Feautrier, “Parametric integer programming,” RAIRO Recherche opérationnelle, vol. 22, no. 3, p. 243–268, 1988.
- [7] V. Chvatal, Linear Programming. Series of books in the mathematical sciences, W. H. Freeman, 1983.
- [8] T. Gal and J. Nedoma, “Multiparametric linear programming,” Management Science, vol. 18, no. 7, pp. 406–422, 1972.
- [9] C. Muñoz and A. Narkawicz, “Formalization of a representation of Bernstein polynomials and applications to global optimization,” Journal of Automated Reasoning, vol. 51, pp. 151–196, August 2013.

DBnary: Extracting Inflected Forms from the French Wiktionary

Extended Abstract

Raphaël Jakse

Magistère internship with Gilles Sérasset (LIG)

Natural Language Processing (NLP) is ubiquitous. Speech recognition, speech synthesis, automated translation and speech analysis are examples of NLP.

Inflected forms, which are forms derived from canonical forms, are an important part of many natural languages. In French, for example, the first person of conditional present form “parlerai” is an inflected form of the canonical form “parler” (“to speak”).

Writing programs related to NLP therefore requires a way to recognize and produce inflected forms and link them with their grammatical information, their canonical forms and potentially other data as definitions, usage, examples. NLP-oriented dictionaries provide such linkings but are usually hard to maintain and to keep up-to-date with languages which are constantly getting new words and neologisms.

Wiktionary is a participative dictionary working like Wikipedia, constantly getting updated by hundreds of people each month. Unfortunately, this valuable resource contains data about inflected form but is unexploitable as is: information is contained in a presentation-oriented syntax which builds lists, tables, paragraphs, titles, rather than being represented in a semantic way. Wiktionary is designed to be read and written by humans who do not necessarily have a background in computer science. Added difficulty also comes from the fact that this presentation is not always very consistent.

The goal of this work is to build a maintainable dictionary of French inflected forms from Wiktionary by extracting and adding them in DBnary. DBnary is what DBpedia is to Wikipedia: a set of data extracted from Wiktionary in the RDF format which is updated when Wiktionary is updated, which makes it possible to use data from Wiktionary in a automated way.

First part of the work was to find out how inflected forms and related data appear in Wiktionary. An inflected form can appear at three places in Wiktionary:

- for nouns and adjectives: in the page of its canonical form, in which a wikicode macro produces a table containing all inflected forms of the form. This table can also be present in the pages of these inflected forms.

- for verbs: in the conjugation tables of the verb, present in a special page of the Wiktionary.
- for all forms: in its own page, in which it is clearly identified as being an inflected form of a given morpho-syntactical type. Its canonical form and morpho-syntactical information is found in the definition present on the page, written in a French following a quite strict pattern (e.g. “Première personne du présent de l’indicatif de [[parler]]”). This French sentence also gives morpho-syntactical information on this inflected form.

Then, parsers had to be written for:

- the French sentence describing the inflected form and giving its canonical form on the page of the inflected form.
- HTML conjugation tables
- HTML tables of inflected forms of adjectives and noun
- extracting information that is given in inflected and canonical form pages with macros like `{{m}}` (masculine), `{{f}}` (feminine), `{{mf}}` (masculine and feminine), `{{s}}` (singular) `{{p}}` (plural), `{{sp}}` (singular and plural).
- finding macros which expand to HTML tables.

Another part of the work was to find out how to represent these newly extracted elements in DBnary; in data structures of its extractor as well as in its RDF model. Previous research on representation of lexical (including morphological) information of a form led to RDF ontologies like Olia and Lexinfo. These ontologies are directly usable in DBnary thanks to RDF functioning and choice was made to use Lexinfo, based on the Lemon model on which DBary is also based.

Last part of the work is a comparison of this extraction with other existing resources: Morphalou, Leff, GLAFF. This part involves conversion of these resources in a common format, writing a comparison algorithm finding common and conflicting forms (form which should have the same morphological information but which have not across resources) and some manual work to understand these differences.

References

- [Buitelaar *et al.*, 2009] Paul Buitelaar, Philipp Cimiano, Peter Haase, and Michael Sintek. Towards linguistically grounded ontologies. In Lora Aroyo, Paolo Traverso, Fabio Ciravegna, Philipp Cimiano, Tom Heath, Eero Hyvönen, Riichiro Mizoguchi, Eyal Oren, Marta Sabou, and Elena Simperl, editors, *The Semantic Web: Research and Applications*, volume 5554 of *Lecture Notes in Computer Science*, pages 111–125. Springer Berlin Heidelberg, 2009.
- [Francopoulo *et al.*, 2006] Gil Francopoulo, Nuria Bel, Monte George, Nicoletta Calzolari, Monica Monachini, Mandy Pet, and Claudia Soria. Lexical markup framework (LMF) for NLP multilingual resources. In *Proceedings of the Workshop on Multilingual Language Resources and Interoperability*, pages 1–8, Sydney, Australia, July 2006. Association for Computational Linguistics.
- [Lehmann *et al.*, 2014] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 2014.
- [Romary *et al.*, 2004] Laurent Romary, Susanne Salmon-Alt, and Gil Francopoulo. Standards going concrete: from LMF to Morphalou. In *The 20th International Conference on Computational Linguistics - COLING 2004*, Genève/Switzerland, 2004. coling.
- [Sajous *et al.*, 2013] Franck Sajous, Nabil Hathout, and Basilio Calderone. GLÁFF, un Gros Lexique Á tout Faire du Français. In *Actes de la 20e conférence sur le Traitement Automatique des Langues Naturelles (TALN'2013)*, pages 285–298, Les Sables d’Olonne, France, 2013.
- [Sérasset, 2014] Gilles Sérasset. DBnary: Wiktionary as a Lemon-Based Multilingual Lexical Resource in RDF. *Semantic Web Journal - Special issue on Multilingual Linked Open Data*, pages –, 2014. To appear.

DBnary : Extracting German inflected Form from German Wiktionary

J r mie Suzan

Magist re internship with Gilles S rasset

1 Abstract

Machine translation, speech synthesis and speech recognition often required data like texts and dictionaries. Good linguistic data should reflect language and language changes, meaning be regularly updated and have a wide coverage. And maintaining and keeping data up to date is difficult.

Wiktionary is a dictionary working like Wikipedia, constantly getting updated by hundreds of people each month. This resource contains inflected forms, but has a presentation syntax for building paragraphs, tables, lists, titles, and all things needed to present a text. The Syntax is easily learned by humans who didn't learn computer sciences.

In German an inflected form is a form derived from canonical form, it can be a declination or conjugation forms. For example, "*kenne*" is the first person present for the verb "*kennen*" (to know), and "*blauster*" is the superlative nominative masculine singular form for the adjective "*blau*" (blue).

Wiktionary contains inflected forms of German, but the resource is unexploitable because of the presentation's syntax. DBnary[3] already extracts information from Wiktionary, the work involves adding German inflected forms into DBnary.

The first step of the work was to find out German inflected forms in Wiktionary. the inflected forms was contained :

- in the canonical form page in a little table.
- in their own page.
- in specials pages, linked to the canonical form page, for example Wiktionary links the page "*wissen*" (to know) with the page "*wissen (Konjugation)*".

In Wiktionary source, there are tables directly built in wikicode, and HTML tables built by macros expansion. For example `{{Deutsch Personalpronomen 1}}`

build a HTML-table containing the Form of the German personal pronouns "*ich*" and "*wir*" in the German's four cases.

This work need to :

- Evaluate Strategy of extraction for conjugation and declination.
- Find Wiktionary's Macro and expanding them.
- Parse Wiktionary HTML tables.
- Extract inflection's information from HTML tables.

The declinated and conjugated forms was separated for their handling. To evaluate extraction's strategies, it was a comparison between inflected forms extracted in their own page, and forms generated with a macro (in special pages or in canonical form's pages).

When extraction operated, the work was to describe inflected forms in RDF format with an Ontology (for example *Lexinfo* or *Olia*). And creating links between inflected forms, their characteristics and canonical forms into DBnary representation.

Last part of the work was to compare the data extracted with other data like Zmorge [2] and morphy [1]. And know how many common forms and conflicting forms they have.

References

- [1] W. Lezius, R. Rapp, and M. Wettler. A freely available morphological analyzer, disambiguator and context sensitive lemmatizer for german. pages 743–748, 1998.
- [2] R. Sennrich and B. Kunz. Zmorge: A german morphological lexicon extracted from wiktionary. may 2014.
- [3] G. S rasset. Dbnary: Wiktionary as a lmf based multilingual rdf network. may 2012.

MPSoC Virtual Platform Generation

Extended abstract

Thomas Baumela. Supervisors : Nicolas Fournel and Frederic Petrot
TIMA Laboratory - SLS Group - Grenoble, France

Nowadays, MultiProcessor System-On-Chip (MPSoC [5]) are widely used because of their multiple advantages (low power consumption, usability in heterogeneous platforms, meeting performances in multimedia, network architectures, security, etc). These systems are becoming more complex to design, thus we use simulation to develop software before hardware platform availability. MPSoC virtual platforms allow to design, run, debug and test software faster than on physical hardware platforms using sophisticated tools. Developing such virtual platforms (VP) is a complex and a time consuming task.

Today, Virtual platform environment (VPE) conceptors, VP conceptors and software conceptors are all depending on each others. These dependencies cause the following issues.

a) Complex and low-level VP implementations: Virtual platform implementations are complex programs, only about 5% (in our working environment) of the information in these is about architecture design. The rest is about syntax, declarations, instantiations, etc. Platform designers need a more high-level representation.

b) VP implementations depend on VPE: VPE updates cause most of the time VP updates. Such operations are manually made by platform conceptors, which results in a long and complex process. If the way that every components are connected to a data bus is modified by VPE writers, this must be changed for every component of every platform.

c) Data redundancy: The set of programs and tools needed to simulate a platform (including the software), contain redundant data and parameters. These data have multiple heterogeneous forms (different formats, languages, usage). The problems caused by data duplications are obvious (complex update, introduction of bugs, etc).

d) Found consistency problems: VP are difficult to implement and debug. Debugging wastes time while the causes of the majority of platform failures are often a simple consistency error in the platform design (mis or unconnected wires, addressing conflicts, etc).

The result work aims at solving every of the highlighted issues by generating platform and tools from simple descriptions. We have defined an architecture description language (ADL) to provide to platform designers the ability to describe their architecture easier. The language can describe a modular representation of data, fitting to describing architecture. It allows to, define components, link between them, give them parameters, and embed them in other components.

ADL descriptions are then taking by the following generation model. This model is composed of three main processes. First, parsing the description language into a specific data structure making easier data access. Second, processing it

by consistency analyser which run a semantic analyser and enriches the data structure to fit it to generator specification. Finally a template processor reads this structure and generates final outputs processing templates (substituting tags by variable contents). Template processing allows to externally define output structure and constant elements without modify the implementation. A data manager can optionally be added to enhance the given description.

Using this model for MVPG, we specialize it by defining an analyser, and a template processor. They are both based on a component-based software model making them loosely coupled. The analyser is composed of a central analysis engine, on which component analysers can self-registering them as a plugin-like system. Each component analyser knows how to analyse the platform component corresponding with. The template processor is based on the same model, each component generator knows how to generate component specific snippets of program (declaration, instantiation, etc). In addition, the data manager of this specialization provides an addressing auto-management tool. This model made analysis and generation processes completely extensible without any modification of existing software.

In addition to VP generator we have developed a component generator. Using the generation model (demonstrating its reusability), it generates component skeletons (without behaviour) to easily write implementation of new components for the VPE. It also generates component analysers and generators to extend the VP generator describe before.

Our work environment is composed of the following tools. YAML [1] in ADL parsing, helping to store the general description structure into hash tables and arrays. Ruby [4] for generator model implementation. C/C++, RABBITS [2] and SystemC [3] for virtual platform implementations. Our experimental works are the following : Generate pre-existent platforms and compare them to validate generated platform. Generate Idscripts as software parametrization.

To conclude, our work was to generate virtual MPSoC platforms to provide higher representations of them to designers and provide tools to help to build softwares. Our generation model is generic and will can be used every time generation is needed. In future, we would fully integrate generation process in all our works about MPSoC simulations (generate all existing platforms, integrate all existing components to our generator, etc). Generate complete diagrams and documentation about a platform. Build a graphical interface to easier manipulate descriptions. Find a way to integrate component behavior in our descriptions.

REFERENCES

- [1] Oren Ben-Kiki, Clark Evans, and Brian Ingerson. Yaml ain't markup language (yaml) version 1.1. Working Draft 2008, 5:11, 2001.
- [2] Tima laboratory SLS team. Rabbits : an environment for fast and accurate mpsoc simulation. <http://tima.imag.fr/sls/research-projects/rabbits/>.
- [3] Thorsten Grötter and Stan Liao, Grant Martin, Stuart Swan, and Thorsten Grötter. System design with SystemC. Springer, 2002.
- [4] Yukihiro Matsumoto. Ruby programming language. <https://www.ruby-lang.org>.
- [5] W. Wolf, AA Jerraya, and G. Martin. Multiprocessor system-on-chip (mpsoc) technology. Computer-Aided Design of Integrated Circuits and Systems, *IEEE Transactions on*, 27(10):1701–1713, Oct 2008.

PWM Driver (Pulse Width Modulation)

Extended abstract

Mehdi MAKHLOUF

TIMA Laboratory - SLS Group - Grenoble, France

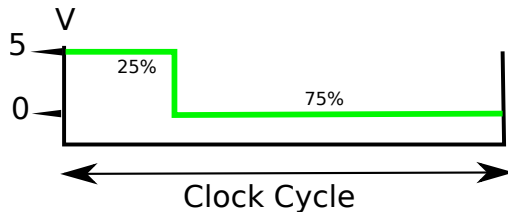
Nicolas Fournel, Frederic Petrot

I. INTRODUCTION

Today Embedded Software become a must. A lightweight OS allowing an adapted material, so less expensive. If we talked about adapted OS, we talk about adapted driver too and for us the pwm driver.

II. PWM

In our material, pins cans just product a digital value (On 5v, Off 0v) in their initial mode. But in severals situations we needed to product numerics values. take a servomotor in example, it has 3 cable (2 for alimentation and one to control it), we want to control it with the third cable, who wait digital values fluctuations: PWM algorithme need a value to push(pull), a range of values and finally a clock divisor to know the frequency. the chip and the motor have a deal for the values of the frequency and range too are sure. now a ratio is calculate with the value push divided by the range. this ratio represents the percent of the cycle where the digital values takes On, the last part of the cycle it is Off. exemple:
range: 1024 frequency:X mHz value push: 256



III. PROBLEMS

To initialize the pwm mode on the chip, we need to access on the physicals adress to modified the device register. We much to say at the pin what mode to use, and many others parameters. Or it's better if the application don't access on the physicals addresses directly, cause of possible errors. A bad adress can cause many problems on the chip, if you write on a other register for example. More the initialization of the device is not very variable, maybe can we activate the device before the application?

IV. SOLUTIONS

A solution exist, driver using. it's initialized before the application main and all access are doing by the access function of the driver(open, read, write). A file is mounted for this access, without have to know addresses of the device registers.

V. EXPERIMENTATIONS

We have begun to write in the raspberry pi [2] registers into application's main. After have understood a part of the documentation, we have find more informations in wiring pi library [3] who offers access of a lot off possibility with the raspberry pi, but on a os who give you the access by a file mounted "/dev/mem". We have finally find lasts missings offsets in this library, we have do our first steps with an oscilloscope and make a driver with this code. Initialize before the main, run with a open on our mounted file "/devices/pwm" and assign values and parameters on the device with primitives write and ioctl (change range, frequency or data to push), with a file descriptor return by the open function. the application is write in c language with Apes [1]

VI. CONCLUSION

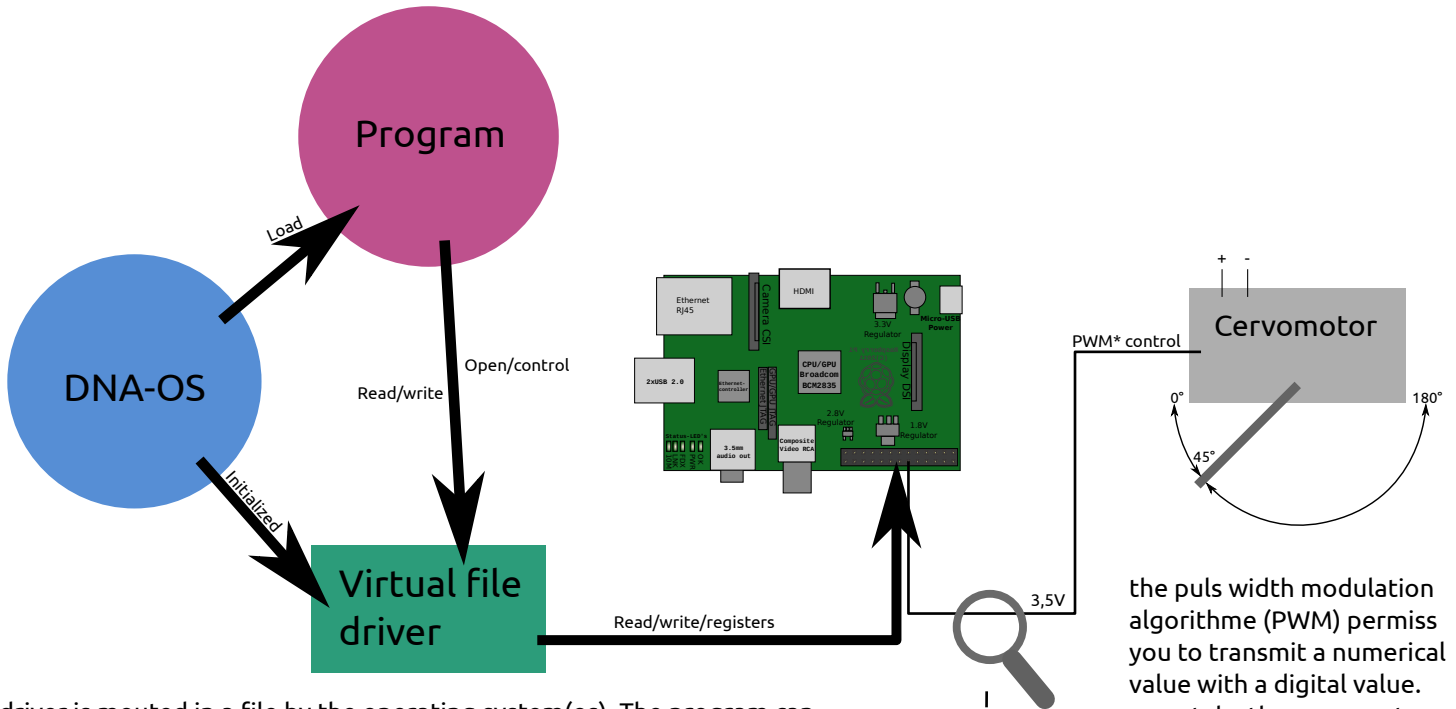
In conclusion we have do a driver of pwm generation for raspberry pi model b, with settings for a total utility without know more information about the raspberry pi registers.

REFERENCES

- [1] Apes: Application elements for socs. <http://tima.imag.fr/sls/research-projects/application-elements-for-socs/>.
- [2] Raspberry pi b : is a credit-card sized computer that plugs into your tv and a keyboard. <http://www.raspberrypi.org/>.
- [3] Wiring pi: gpio interface library for the raspberry pi. <http://wiringpi.com/>.

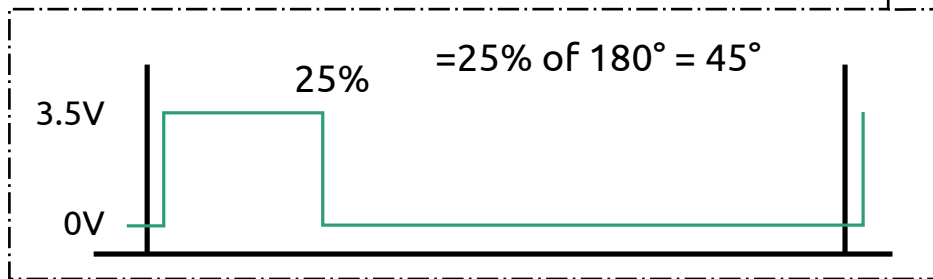
Output device drivers for lightweight OS

Mehdi Makhlouf

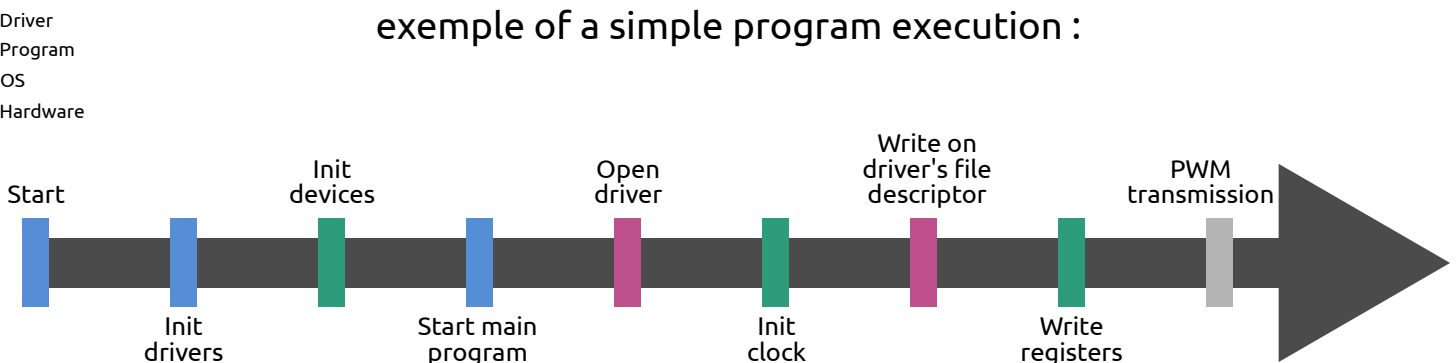


The driver is mounted in a file by the operating system (OS). The program can read, write or call an other control function with the file descriptor return by an opening of this file. File mounted take a gate responsibility and call the driver function when a read, write or control function was call on the program.

the pul width modulation algorithm (PWM) permitt you to transmit a numerical value with a digital value. pwm take the cervomotor range and frequency on parameters to transmit a utilisable value. the digit begin at 1(3.5v) and end at 0(0v), the value transmit is equal as the percent of the time where the digit stay at 1. exemple on left.



exemple of a simple program execution :



when OS starts, it mounts all drivers and initialize them, all of them initialise the device who it control with read/write on the devices registers. after all initializations it will start the programm's main. the programm open the driver's file mounted , who permitt to init the PWM's clock register for us, and after transmit a value with a write to give an order at the driver to write on the pwm's data register to permitt the hardware to execute the algorithme and transmit the value with the gpio. Finally the cervomotor take the value and on this exemple turns to have the same angle as the data recieved.

Resolution of Bin Packing Problem on GPU

Luc Libralesso

Supervised by: Michaël Gabay

Extended Abstract

In the bin-packing problem, a finite set of items is to be packed into a minimum number of bins [Martello and Toth, 1990].

It has many applications like cutting metal bars, filling trucks, virtual machines placement...

Thanks to some work on *bin-packing problem*, a company reduced its production costs of paper for an amount of 56 million euros with reduce wastes of about 1% [CEPI, 2012].

The bin-packing problem is a hard combinatorial optimization problem (*NP-Complete*) Hence, it is very challenging to solve it optimally.

We can use different approaches :

1. Find structures on instances. With this properties, we can design efficient algorithms for special cases.
2. Use and design heuristics. Heuristics are fast algorithms but they do not always return an optimal solution. Sometimes we can even show that the value of the heuristic solution cannot worse than a ratio to the optimal solution value. For example, *First Fit* solution for the bin packing problem is guaranteed to be less than $\frac{71}{60} \times OPT + 1$.
3. Solve it with a exact – smarter – algorithm (whose complexity) is exponential in the worst case.

In this internship, we focused on the third option and saw what can we do to solve bin packing instances in a faster way.

In a first time, we developed and implemented ways to generate in parallel, partitions and permutations by finding a bijection between an integer set and solutions space. Moreover, partitions are standard mathematical objects and can be used for other problems.

The index based object generation makes it easy to implement the approach on *Graphics Processing Unit (GPU)*.

Permutations : This is the first version of parallel generation [Wong, 2002].

Partitions : This method use the same principle than Permutation method (i.e. generation by index). It also use partition representation with *restricted growth strings* [Orlov, 2002] that breaks a lot of symetries.

Partitions with sizes : With this method, we only generate partitions with a given size..

Another way to find the optimal solution is to make decisions on partial solutions. Partial solutions are extended until we can guarantee that we will not give a better solution than the best known solution.

Idea : Represent problem by a tree in which each edge is a choice and each leaf is a solution. The *Branch and Bound* approach finds a leaf minimizing the value of the solution.

Key Steps :

- **Choose** the “most interesting” node
- **Generate Children** of the selected node
- **Compute bounds** with heuristics and relaxations
- **Update** global bounds (if feasible)
- **Prune** nodes by comparing their bounds to global bounds

We designed some ways to make choices and made an implementation than can solve instances with up to 200 items in less than one second.

References

- [CEPI, 2012] Key Statistics CEPI. European pulp and paper industry. 2012.
- [Martello and Toth, 1990] Silvano Martello and Paolo Toth. *Knapsack problems*. Wiley New York, 1990.
- [Orlov, 2002] Michael Orlov. Efficient generation of set partitions. *Engineering and Computer Sciences, University of Ulm, Tech. Rep*, 2002.
- [Wong, 2002] SV Wong. Permutations with cuda and opencl. Article on <http://www.codeproject.com/>, May 2002.

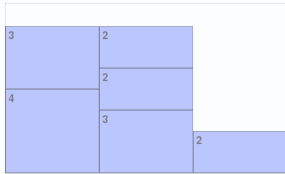
Problem Presentation

Bin packing has many applications :

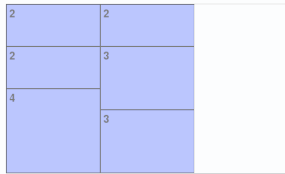
- Cut cables to minimize waste
- Load Trucks to minimize lost space

It's an *NP-Complete* problem.

examples of solutions for instance with
bin capacities = 8 and items = (4, 3, 3, 2, 2)



(a) Heuristic solution
3 bins



(b) Optimal solution
2 bins

Integer Linear Programming

Data :

- c (integer > 0) bin capacity
- n (integer > 0) number of items
- w_i (integer > 0) weight of the item i

Variables :

- $x_{i,j}$ (boolean) true if item i is inside bin j false otherwise
- u_j (boolean) true if bin j is open false otherwise

$$\text{Minimize } \sum_{j \in \{1, \dots, n\}} u_j$$

$$\sum_{i \in \{1, \dots, n\}} x_{i,j} \times w_i \leq c \quad \forall j \in \{1, \dots, n\} \quad (1)$$

$$\sum_{j \in \{1, \dots, n\}} x_{i,j} = 1 \quad \forall i \in \{1, \dots, n\} \quad (2)$$

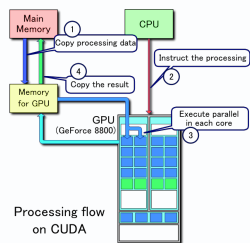
$$u_j \geq x_{i,j} \quad \forall i, j \in \{1, \dots, n\} \quad (3)$$

$$u_j \geq u_{j+1} \quad \forall j \in \{1, \dots, n\} \quad (4)$$

$$x_{i,j} \in \{0, 1\} \quad \forall i, j \in \{1, \dots, n\} \quad (5)$$

$$u_j \in \{0, 1\} \quad \forall j \in \{1, \dots, n\} \quad (6)$$

Graphics Processing Unit



Processing flow on CUDA
CUDA processing flow (En) by Tosaka - Own work

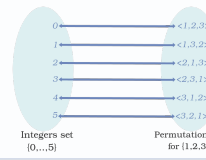
We use *GPU* to compute solutions. It often uses about 500 cores which have a frequency about 700Mhz.

It's also possible to improve global performance by using a good type of memory.

Permutations and Partitions

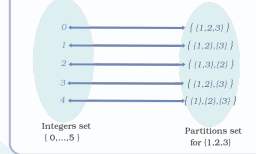
Start Point

We generate index-based permutations [1]



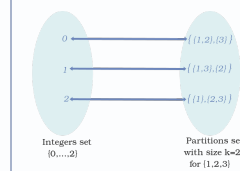
Index for Set Partitions

We combine permutation generation with restricted growth strings to generate partitions [2]



Size Matters

We keep only interesting partition sizes.



Branch & Bound

Idea : Represent problem by a tree in which each edge is a choice and each leaf is a solution. The *Branch and Bound* approach finds a leaf minimizing the value of the solution.

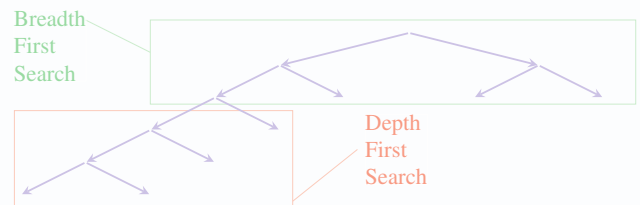
Choose the "most interesting" node

Generate Children of the selected node

Compute bounds with heuristics and relaxations

Update global bounds (if feasible)

Prune nodes by comparing their bounds to global bounds



Perspective

Permutations and Partitions

- Can we generate partitions of *multisets* ?

Branch and Bound

- Experiment other branching rules
- Improve node selection heuristics (Pricing)
- Implement additional heuristics and reductions
- Combine *CPU* and *GPU* to further accelerate the *Branch and Bound* (Hybrid Computing)

Bibliography

- [1] SV Wong. Permutations with cuda and opencl. Article on <http://www.codeproject.com/>, May 2002.
- [2] Michael Orlov. Efficient generation of set partitions. *Engineering and Computer Sciences, University of Ulm, Tech. Rep.*, 2002.
- [3] Silvano Martello and Paolo Toth. *Knapsack problems*. Wiley New York, 1990.

Performance analysis of accelerator caches

Lea Albert, Guillaume Huard

Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France
CNRS, LIG, F-38000 Grenoble, France, Inria - MOAIS

Nowadays, computers are composed of accelerators in addition to their CPUs (central processing unit), that enable the execution of parallel tasks. GPUs, are composed of grids of cores. Each such grid possesses eight to thousands cores that work synchronously. They have a L1 cache shared between their cores and are connected to the same global L2 cache. MICs have also many cores that work asynchronously and have a private L1 cache. Here we work on the only currently existing MIC, the Intel Xeon Phi coprocessor (60 cores). It has a particularity: each core has a local L2 cache shared with the other cores using a bidirectional ring interconnect.

In this work we aim at finding out which of these architectures is the most efficient regarding the memory. We want to evaluate them regarding the raw memory efficiency and the efforts required from the programmer to reach their full potential.

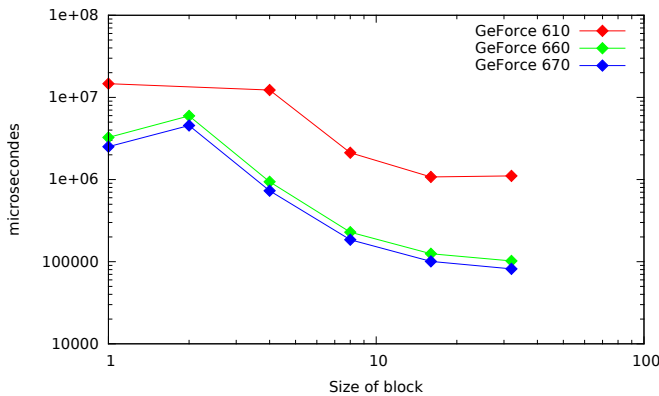


Fig. 1. Matrix product of 2048×2048 matrices on GPUs

In our experiments, we have decided to work with an algorithm that can be efficiently parallelized, the matrix product, in which the same data in memory are reused multiple times. In a naive implementation of this algorithm, the locality is poor: only the lines of the first input matrix are a source of spatial locality. We will compare it to a matrix product by block [1]: the algorithm iterate over blocks of the three matrices to compute a part of the product in which each element of each input block is reused a number of times equal to the block size.

In our first experiment, presented in figure 1, we compare three different GPUs of the same generation of Nvidia GeForce. We study the effect of the size chosen for the blocks on the efficiency of the matrix product by block. Notice that the naive implementation is equivalent to a matrix product by

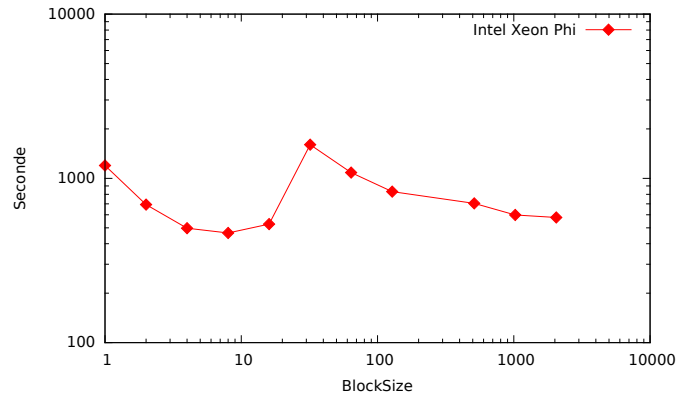


Fig. 2. Matrix product of 2048×2048 matrices on the Intel Xeon Phi

block using 1 as the size of the blocks. Although the 610M is improved by up to 5,7% when the blocksize increases, it stays the least efficient. It has a L1 and L2 caches smaller than the other accelerators. There are almost not differences between the GeForce 660 and GeForce 670: Both of them are improved by 3%. They are composed of the same type of memory hierarchy and of caches of almost the same size.

In our second experiment, presented in figure 2, we evaluate the same algorithm running on Intel Xeon Phi coprocessor: Until a block size of 16×16 , the performance is improved: for each result block, we can store all the sub-matrices in the L1 cache. After this size, there is a degradation: with a block size of 32×32 , the degradation is significant due to it uses the L2 cache instead of the L1. We can note that from the 64×64 we start to become more efficient, probably because of the increased locality brought by the larger block.

To conclude, we have seen the cache behaviour of the GPUs and MICs is the same for the matrix product. An effort is required from the programmer to reach the full potential of the memory hierarchy and, regarding the gains, it is especially important on the Xeon Phi. We have performed these experiments on small matrices (2048×2048) and it would be interesting to extend it to larger sizes that do not fit into the L2 cache. Also, here we have only evaluated one program, the matrix product. It would be interesting to test and compare our accelerators on others programs.

REFERENCES

- [1] Ulrich Drepper. What every programmer should know about memory. Technical report, November 2000.

Performance analysis of accelerator caches

Léa Albert
Guillaume Huard
INRIA MOAIS

I) The context

In recent years, the accelerators were developed in the market: There are cheap and powerful. There are two types of accelerators:

a) MICs (Many Integrated Cores)

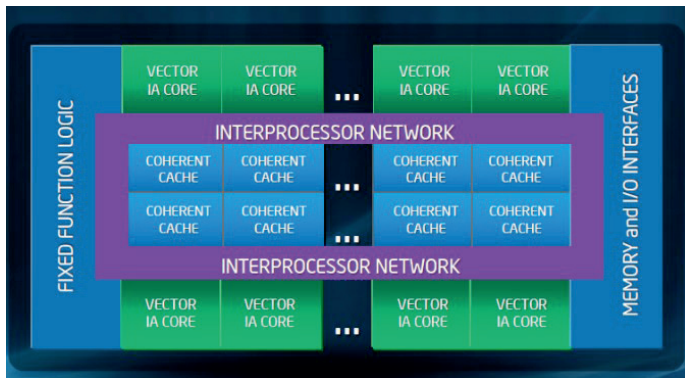


Fig 1: The Intel Xeon Phi coprocessor used for our experiences

It has many asynchronously cores. Each cores have a private l1 cache. The l2 cache is a bidirectional ring interconnect and each core is connected by this cache.

b) GPUs (Graphics Processing Units)

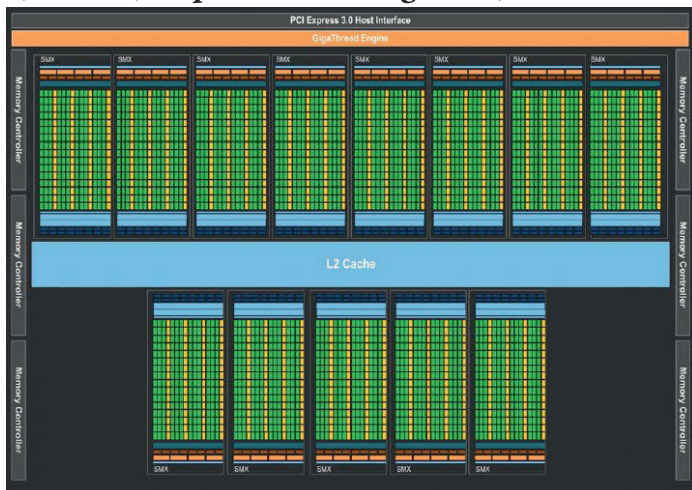


Fig2: The NVIDIA's GPU

GPUs are composed of grids. Each grid has synchronously cores and a l1 cache. A l2 cache connect all the grids.

II) The goal

Memory has many consequences on the running time of a program. The goal of the experience is to evaluate the local memory of these two architectures.

We use the matrix product program:

- Classic linear algebra
- Handle much data to be re-used

- C_{3,5} and C_{3,6} use U
- C_{4,5} and C_{4,6} use V
- C_{3,5} and C_{4,5} use W
- C_{3,6} and C_{4,6} use X

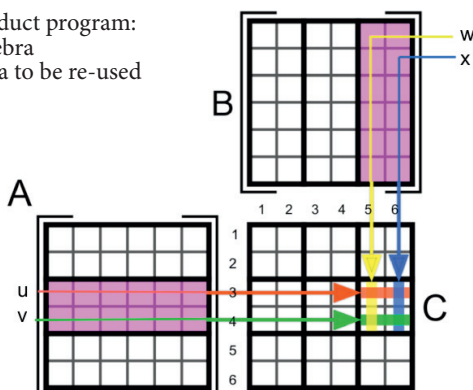


Fig3: Matrix product per block [1]

III) Experiences

a) Program with and without shared memory

The GPU used hasn't a l2 cache. By copying in local memory, we can go faster. It is important to allocate explicitly memory to gain performance.

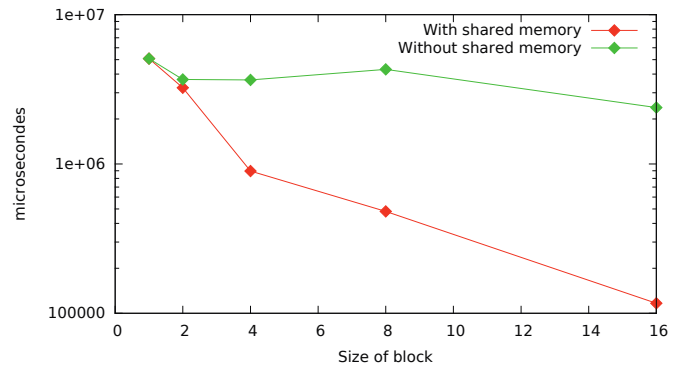


Fig 4: Running of matrix product of size 1024x1024 on Quadro FX580

b) Program on several GPUs

- By cutting the matrix result on block, we are more efficient
- The GeForce 610 is less powerful (l1 and l2 cache smaller).
- Few differences between GeForce660 and GeForce 670 (almost similar capacities).

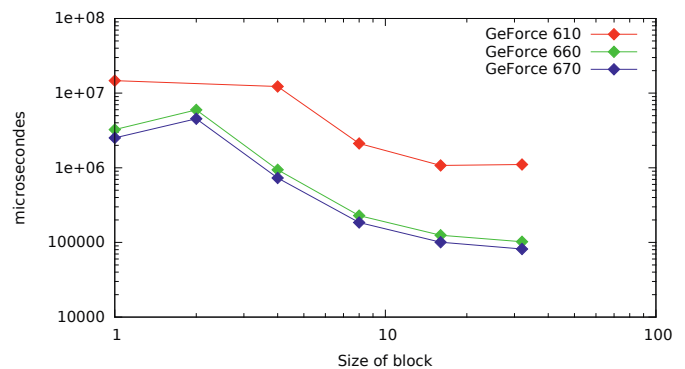


Fig 5: Running matrix product of size 2048x2048 on GPUs

c) Program on the Intel Xeon Phi

- The Intel Xeon Phi performances are improved.
- Capacity to store all data in l1 cache (with block size 16x16 or -)
- Need to use the l2 cache after this size (peak on 32x32)
- Using l1 and l2 cache for compute less block (go faster).

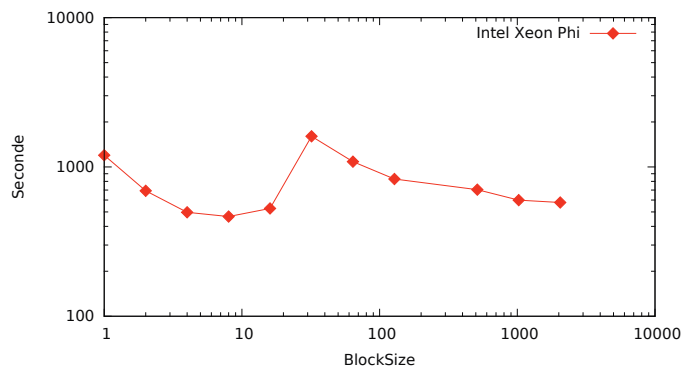


Fig 6: Running matrix product of size 2048x2048 on the Xeon Phi

IV) Conclusion

It is important from the programmer to work on his code to benefit fully from the memory whatever the architecture.

VSS-Simulator Stats Viewer

Myriam CLOUET

31 juillet 2014

During the last years, Vehicle Sharing Systems (VSS) have expanded dramatically. Those systems are interesting for a variety of reasons (ecological, economical and political). People are led to think that Vehicle Sharing System will continue to develop.

There is a need for the operating analysis of VSS, their return on investment and tools for deciding where to implement them and with which design parameters, such as station location and sizing, fleet sizing, incentives and pricing policies.

That's why G-SCOP laboratory developed a Vehicle Shareing System simulator (VSS-Simulator) for analyzing virtual VSS thoroughly.

The simulator can provide data in CSV (Comma Separated Values) format. This file can be converted into a tabular within the needed data.

But this format is not really human readable and there is a real need for its analysis. There is a need for a program that is able to convert CSV format into a graphic output would therefore useful.

During my internship, I developed such a program. For that, i used Java and some libraries like JFreeChart and Swing.

This program works as follows : the user indicates the path to a CSV file output by the simulator, keeps the important data structure from the CSV file, then the user can choose which data display. Finally, the program computes the required data, extracts it and displays it with diverse charts.

However, there are some improvements needed for the program. One of them is to allow the user to treat other data from the simulator, annex data. Another improvement can be generating other chart types from the user request

VSS-Simulator Stats Viewer

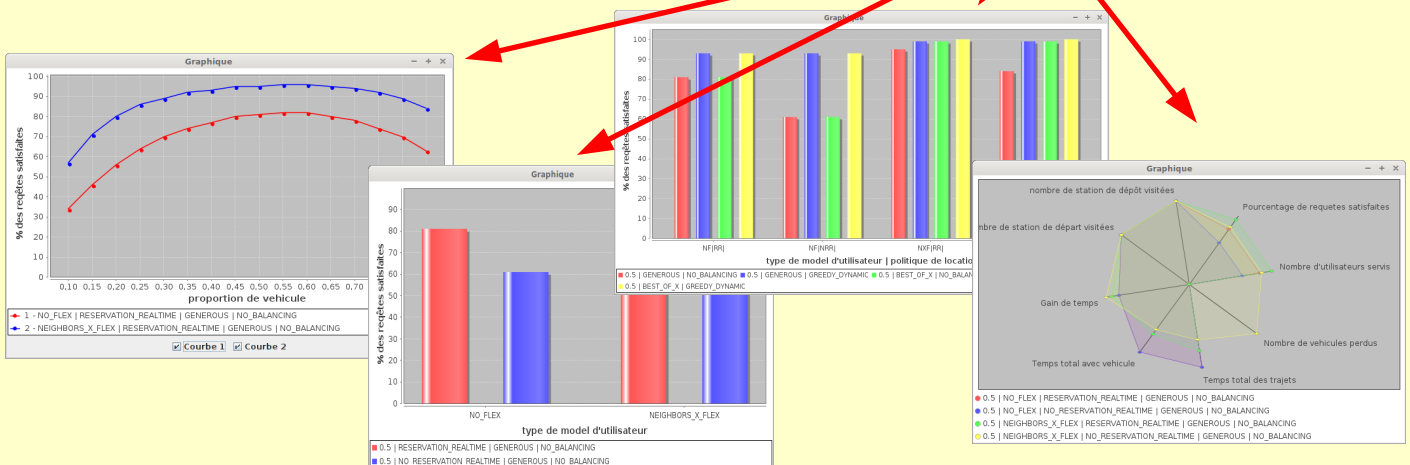
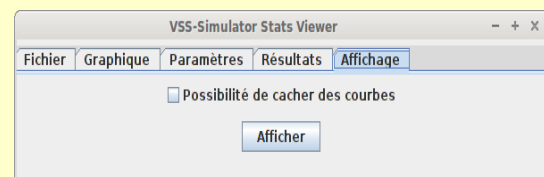
Introduction and Objectives

- Recently **increasing** interest in **Vehicle Sharing Systems (VSS)**
- Already existing **VSS-Simulator**
- Necessity to analyze data from VSS-Simulator

Equipments and Methods

- Equipments :
 - langage : **Java**
 - library : **jFreeChart, Swing, ...**
- Methods :
 - Program** creation that parse file **CSV** from VSS-Simulator, store **data** and generate **Charts** depending **user** request.

42	17	817	-36	13,325	4,676	0,37
43	20	988	147	14,349	4,682	0,37
44	24	1187	312	14,349	4,682	0,37
45	29	1445	690	14,349	4,682	0,37
46	37	1795	989	13,837	4,682	0,37
47	47	2256	1475	14,093	4,682	0,37
48	57	2758	1948	14,349	4,682	0,37
49	66	3209	2406	14,349	4,682	0,37
50	75	3604	2768	14,349	4,682	0,37
51	82	3934	3093	14,349	4,682	0,37
52	88	4213	3321	14,093	4,682	0,37
53	92	4423	3548	13,837	4,682	0,37
54	97	4641	3748	14,349	4,682	0,37
55	99	4742	3812	14,349	4,682	0,37
56	100	4758	3836	14,349	4,682	0,37
57	99	4756	3818	14,349	4,682	0,37
58	99	4736	3823	14,349	4,682	0,37



Conclusion

- The **program** allows to **user** to **choose** various **charts** from **data** that he want to analyse in the aim to **assist** their **analyse**.
- **Improvements** :
 - Process more data
 - Generate more charts.

Analytical model on memory and cache coherence

Marion Dalle

Univ. Grenoble Alpes,
F38000 Grenoble, France
CEA, LETI, MINATEC
campus, F38054 Grenoble, France
marion.dalle@e.ujf-grenoble.fr

Supervised by: Yves Durand, Florence Perronnin and Jean-Marc Vincent

Abstract

Designing multicore systems with shared memory attracts much attention from the designers in the area. Analytic models allows to evaluate the performance of these systems. Thanks to this insight the designers can dimension the architecture. The existing models consider average service times. Unfortunately the system with shared memory uses a cache coherence protocol. These protocols can cause the service times to vary and the existing models do not take into account this variability. In this paper we develop a detailed study of the service time variability for a novel architecture developed at CEA Grenoble. Some models are developed to evaluate the performance of the architecture studied. By building increasingly complex models of the architecture and analyzing these models.

Distributed Interactive Model Checking for Hybrid Systems

Jeremy Wambecke
M1 Informatique Verimag

Supervised by: Goran Frehse and Olivier Lebeltel

I understand what plagiarism entails and I declare that this report is my own, original work. Jeremy Wambecke, 21/08/14:

Abstract

Hybrid systems describe the evolution of a set of continuous variables with both discrete and continuous dynamics. They are studied since they can be applied to model a lot of systems, relative to computer science or to other fields. SpaceEx is a state of the art model checking tool for these systems, based on efficient algorithms to check the models of real dimension systems. Its current user interface, through a web browser, gives a limited interaction with the tool.

An interactive way to control the analysis, associated to a visualization system, will make easier the checking of hybrid systems and so their modeling. Some examples of features that can be useful are a step by step analysis mode and a pause command, to visualize partial results. This system has to be distributed, to allow both local and distant usages. For this, a client-server architecture provides a flexible and easy to maintain solution, while allowing these usages.

The result is a single-user prototype which computes trajectories and sends them to the interface to be displayed. It can react to some control commands, such as pause and provides a step by step mode for the analysis. Further work is needed to integrate the SpaceEx analysis core and to add multi user functionalities.

1 Introduction

Hybrid systems are used to represent continuous systems in which discrete events occur. They deal with the evolution of real variables, which is continuous but can change according to discrete events. The reachability of these systems, i.e. trying to know which values the variables can have, is known to be a hard problem due to this particularity of both discrete and continuous properties [Alur *et al.*, 1995]. SpaceEx [Frehse *et al.*, 2011] is a tool platform for the Verification of such systems, developed at Verimag laboratory under the Hybrid

team of Oded Maler. It consists of an analysis core based on a reachability algorithm, associated to a model editor for creating hybrid systems models in analysis core format. These models can then be sent to the analysis core to be checked through a web interface.

The main goal of this internship is to design and implement a new form of interaction with the SpaceEx analysis core, which will be more powerful, user friendly and convenient than the current one. An important requirement for this new interface is the reactivity. Indeed, a computation can take a large amount of time and we want to get a feedback on it. For example, if an user launches a computation and waits a lot of time while it's not expected, it would be better that he can have a way to ask what is happening with this. Within this scope, some ways to control the computation have to be added, such as a step by step mode which allows the user to incrementally check a model.

This new form of interaction with SpaceEx would ideally be fitted to the two main usages, namely local and distant ones. So, an architecture which seems appropriate is a client/server one, where the server can be accessed locally or distantly. The server encapsulates the SpaceEx analysis core and provides to it a way to react to control commands, and to send results to the client. This last includes a way to send these commands and the visualization system for the analysis results. One approach for this kind of architecture is remote procedure calls (RPC), which allows the client to call functions of the server. This approach has several advantages, such as the encapsulation of low-level network aspects.

The implementation results in a server embedding a computation core which generates trajectories. The SpaceEx core was not yet embedded in the server, as a simplest analysis core which simulates its behavior was used for the tests. This server can react to some control commands sent by the client, such as pausing the computation, computing a step in the algorithm or stopping it. It sends partial results, i.e., the trajectories that have been computed, when it's going in pause, on demand or after a step was executed. In the other side, the client provides a way to send the commands to the server, receiving the computation status to inform the user. It displays the received trajectories with some options, such as variables to display selection or zoom features.

This report is structured as follows. Section 2 presents the hybrid systems and the notion of reachability. Then section

3 evokes the client-server architecture and the possible applications provided by it. Section 4 discuss about the interests to have an interactive model-checking, and presents the relative functionalities to be provided. Section 5 explains some implementation choices, such as the choice of a RPC library. Finally, section 6 comes as a conclusion and presents further work to be done on the project.

2 Hybrid systems

This part describes the concepts of hybrid systems, their representation as automata and the concept of state in this field. Then, we approach the modeling and model checking concepts and discuss why it is an important issue. For more detailed description, see [Branicky, 2005].

2.1 Description

A hybrid system consists of a set of real valued variables which evolve according to continuous equations. It has a set of discrete states, called the locations, each having its own equations. At a given time, the system is in one location, which determines the equations that are applied on the variables. Thus the equations that are applied to the variables depend on the location in which the system is. The equations of a hybrid system are called the flow equations. A hybrid system is associated to discrete events occurring according to the variables values, which lead to a location change. These events are called transitions. A hybrid system can be represented by a hybrid automaton, with the locations as the automaton states and the discrete events as the transitions.

We use the following syntax in the hybrid automata representation. The equality is represented by a '=' symbol, while the '=' symbol represents an affectation. The prime represents either the derivative of a variable or the new value of it, depending on whether it is in a location or in a transition. For example, x' represents the derivative of x in a flow equation, whereas it represents its new value if it is placed in a transition. We will precise this meaning in the following.

2.2 Locations

A hybrid automaton is composed of discrete states called locations, each having its own flow equations and applying them to the variables. So, the dynamics of the variables, i.e., their evolution, is defined by the flow equations of the current system location. The locations are associated to a label, and have an invariant on the variables which defines when it is possible to remain in it.

Figure 1 describes the parts of a location of an automaton with two variables. In this example, x must be positive and y lower than 10 to stay in this location. The dynamics of y and x are expressed by their derivative, the one of y is equals to x while the one of x is 1.

In a hybrid system, the expression of the flow equations is of the form $x' = a1.x + a2.x + a3.x \dots + b$. It is possible to represent inequalities in this form in placing restrictions in invariants. For example, an inequality $-1 < x' < 1$ can be expressed in introducing a variable e , with $x' = e$ as the flow equation and $-1 < e < 1$ as the invariant.

Figure 1: Location example

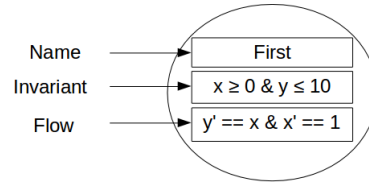
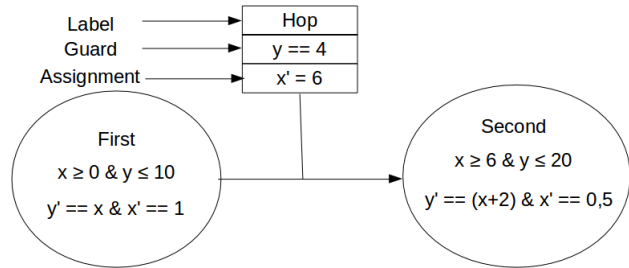


Figure 2: Transition example



2.3 Transitions

The locations are associated with transitions. They have a label to be identified and can have a guard, which is a condition on the variables to take the transition. The transitions optionally have some assignments on the variables, represented with a prime symbol. Notice that these assignments can lead to a discontinuity in a variable dynamics, as its value changes arbitrary at a given time.

Figure 2 describes the parts of a transition between two locations. In this example, this transition is possible when y reaches the value of 4. During this transition, x takes 6 as value, as described with $x' = 6$. Note that x' in the assignment part of a transition is not the derivative, but the representation of the new value of x .

Two important rules about transitions have to be known :

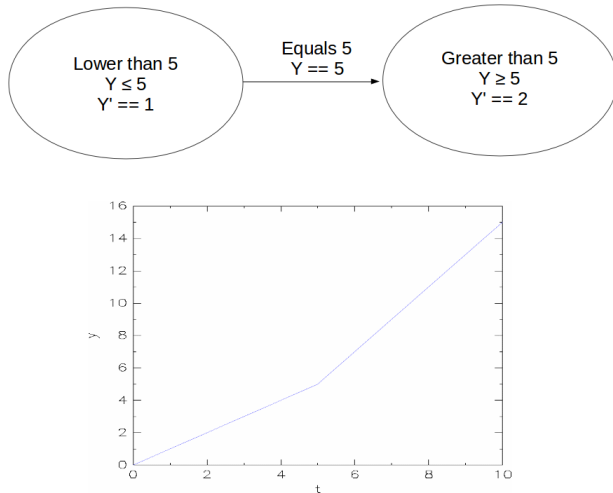
- The invariant of a location must be satisfied to stay in it.
- To take a transition from a location to another, the invariant of this latter has to be satisfied, as well as the guard of the transition.

Hybrid automata can be non-deterministic, as if some transitions are possible the system can take any of them or stay in the same location if its invariant is respected. The guard of a transition is optional, as it can be deduced from the invariants of the source and destination locations. However, for clarity reasons, a common practice is to always specify the guard of a transition.

2.4 States and reachability

Let's introduce the concept of state. A state of a hybrid system is defined by a couple (loc, val) where loc is the current location of the system and var is a set which gives each variable a value. This must not be confused with a discrete state, called location. A hybrid automaton is associated to an initial state, which corresponds to the location in which it begins

Figure 3: Example of a hybrid system describing the evolution of a variable y . The curve represents the beginning of its execution with $(y \leq 5, y==0)$ as initial state. Notice that this system is non-deterministic, as when y is equals to 5 the two locations are acceptable for the system. However, a hybrid system is in one location at the same time.



and a set of values for the variables. These values correspond to the initial ones of the variables.

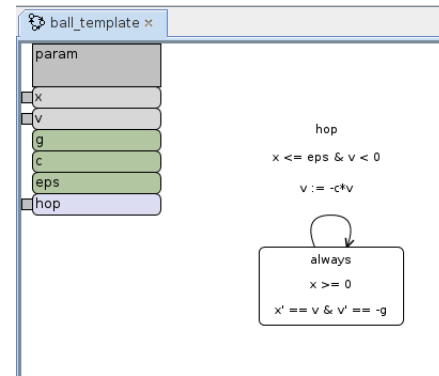
An execution of a hybrid automaton is a sequence of associations of transitions and sets of values for the variables, according to the dynamics of the locations covered by the hybrid system. An execution originates in a given initial state. A state is reachable if an execution leads to it. In other words, the reachability can be defined by, from a given location and having an initial values for variables, it is possible to reach a given location with expected values for the variables. For example, in figure 3 the state (Greater than 5, $y==6$) is reachable with (Lower than 5, $y==0$) as initial state.

We define a trajectory as the sets of values taken by a hybrid system variables during the time passed in a given location, after a given transition and before it takes another. An execution can be represented with a set of trajectories, corresponding to the different locations reached by the system. For example, in figure 3 the execution can be represented by two trajectories. The first corresponds to the time passed on a first location, when y is lower than 5. The second corresponds to the time passed on the second location, where y is greater than 5.

2.5 Modeling and checking

A critical step in these systems is the modeling one, as its result will make the foundation of a product. That is why a lot of time is dedicated to ensure that the model will not reach unacceptable states, especially in critical systems. For example, a helicopter controller must ensure that the vertical orientation will not exceed a given threshold for the aircraft not pitching down. That demonstrates the need for a tool that provides not only reachability features but also a way to find where the model behavior has began to be wrong.

Figure 4: Bouncing ball model defined in the Model Editor of SpaceEx



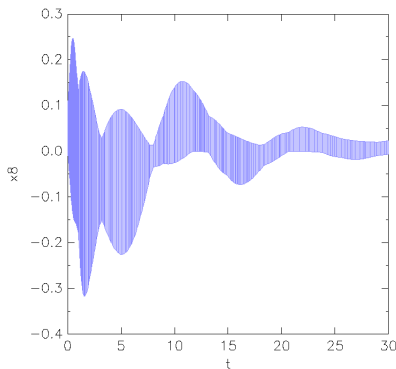
SpaceEx[Frehse *et al.*, 2011] [Frehse, 2010] is a model checking tool for hybrid systems, developed by the Hybrid team of Oded Maier at Verimag. It enables to define a hybrid automaton model in its own language, with the help of a model editor. Figure 4 shows the model of a bouncing ball defined in the Model Editor.

Then the user can specify some options as the initial state and then he can check the model. It provides several algorithms such as Phaver [Frehse, 2009] or Le Guernic-Girard (LGG) [Guernic and Girard, 2010], and features to develop new ones. These embedded algorithms allow to check real models, with hundred of variables, in an acceptable time. This scalability is provided by the algorithms properties, especially the data structures used, which allows to perform some frequently used operations in model checking. This particularity makes SpaceEx attractive for companies, as a scalable checking tool for a wide range of systems which depend on real variables. It supports the concept of forbidden states, that can be defined. A forbidden state is one that must not be reached in the execution. If it is the case, it is notified to the user. Figure 5 shows the results of a helicopter controller analyze in SpaceEx.

2.6 SpaceEx algorithms

To understand the design of interactive model checking, we describe the form of the SpaceEx algorithms. It uses a fixed point computation, that is, the result obtained at a step is used as the basis of the next one. Figure 6 represents informally the form of the algorithms used in SpaceEx. Two sets of states are used. The first one, called *waiting*, represents the states which remain to be analyzed. The second one, called *passed*, contains the states produced with the previous steps. The states in the *passed* set are so the reachable states of the analyzed model. At the beginning, the waiting set contains the initial state of the system and the passed set is empty. At the end, the passed set is sent to the output as it represents the system evolution. The function *computeSuccessors* returns the successors of a state. This function depends on the used algorithm, which differ mainly in set representation and operations on it. It checks if the states that results from this were not already reached. If it is the case, i.e., if a state pro-

Figure 5: Results of a helicopter controller analyze in SpaceEx. This shows the aircraft stabilization by the controller.



```

passed =  $\theta$ 
waiting = { initial_state }
while(waiting  $\neq$   $\theta$ ) loop
  s = pop(waiting)
  waiting = waiting  $\cup$  computeSuccessors(s)
  passed = passed  $\cup$  {s}
end loop
output(passed)

```

Figure 6: Form of the SpaceEx algorithms

duced is already in the *passed* set, it is not added. So, the algorithm ends only if, at a given time, all the produced states were already reached.

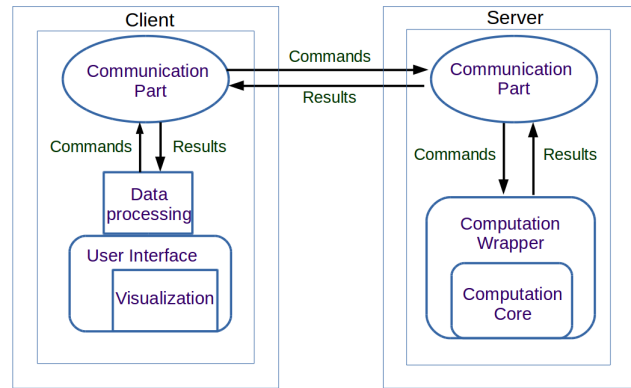
3 Architecture for Local and Distributed Usages

This section describes the architecture used for the system. The chosen architecture model is a client-server one (see [Janssen, 2014] for a definition). The server side is composed of the computation core with a wrapper around it to interpret received commands, while the client side has a part of data processing, a visualization system and a graphical interface for the user to interact with ergonomics. Both client and server have a communication layer, which interact according to defined protocols for the client to connect to the server, to launch a computation, to control it and to exchange data in a certain format.

3.1 Local and distributed usages

- **Local usage** corresponds to the scenario in which an user downloads SpaceEx on his own machine for working on it, just by launching it like another simple software. In particular, the user does not have to configure anything for running SpaceEx. What's more, several

Figure 7: System architecture



instances of this same software can be launched at the same time without having conflicts. With this scenario, the server can be seen as having a single client.

- **Distributed usage** corresponds to a scenario in which the server is running on a machine, and several users connect to it to request model checking, each having his own client on his machine. The main example for this case is a classroom session, where the server was configured and runs on a machine while some students are launching computations on it. This scenario brings up some issues, like getting the IP/port of the server for the clients or identifying them.

3.2 Advantages of a Client Server Architecture

The main advantage of this architecture is the independence of both parts, with the exception of the communication layers which depend on communication protocols. Client and server can be written in different languages, which is the case in this system. The SpaceEx computation core was developed in C++ for performance, as model checking operations are costly. Since the server side has to encapsulate this computation core, it has to be written in this language to encapsulate it. The client side has to be portable for being executed on different systems, so the choice of Java was made. In addition it makes easier the development of graphical user interfaces with the embedded swing library.

3.3 Remote Procedure Calls for Communication

One approach for this kind of architecture is remote procedure calls (RPC), which allows the client to call functions (or methods in case of object oriented programming) of the server. This approach has several advantages, such as the reuse of network-dealing code, which furthermore encapsulates low-level aspects. Furthermore, RPC give more semantics than a raw flow of bytes, as when handling with read/write on sockets. This method is used for message passing between the client and the server, for example for the client to send the model to check or for the server to send the results. The RPC are used to implement the communication protocols that initiate an analysis.

3.4 Applications

This architecture provides several ways of usage for SpaceEx :

- **Local**

The communication can be established with client and server on the same machine, allowing the user to launch both as a local software.

- **Distributed**

Client and server can communicate through a network. In this way, it is possible to launch the server on a machine and to access it distantly. This makes the usage of SpaceEx easier, as the server can be configured once for all users, which have just to specify the machine coordinates to access the server.

- **Classroom work**

The typical applied example of the previous point, where the server is running on a machine in a computer room, while students are launching some computations on it in the context of a practical work.

- **Company usage**

The model checking of hybrid systems interests the industry, so with this architecture the server can be installed on a shared machine of a company, accessed by its members. This will allow to share computation results, and to simplify the usage of SpaceEx.

- **Cloud**

The most advanced usage of the system is to let the server run on a machine accessible through Internet and to distribute a preconfigured version of the client, which allows to connect without configuration. It is in consideration at Verimag, but requires a lot of work and means to apply it.

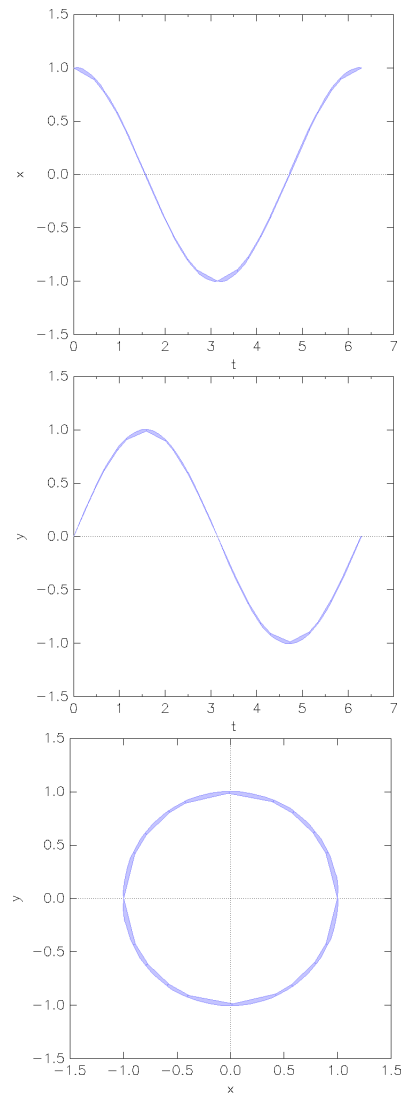
4 Interactive model checking

A tool which provides a way to check a hybrid model with interactivity can be very useful. In fact, with only the final results of the model checking, it is difficult to visualize the evolution of a model. If the results are not valid, i.e., a not acceptable state was reached, it is important to discover at which step the system has began to go wrong. So this tool could act like a debugger, for example in automatically pausing the computation when a wrong state is reached or executing it step by step. In addition, some controls are given to the user through the client, as making the core to pause the computation, to execute a step or to restart it.

4.1 Visualization system

Added to these controls, an interactive visualization system can help to understand the model behavior. This can provide a way to select some variables, whose trajectories will be displayed. A zoom can be applied on these. Furthermore, the understanding of what is going on with the system is sometimes easier when displaying the variables as a function of a certain another one, which is provided by this system. Figure 8 shows an example of a such system, which is composed of

Figure 8: Computation results with variables as function of another



three variables : x , y and the time t . The two first curves are representations of respectively x and y as functions of time, while the third is the representation of y as a function of x . We can see that this system is perfectly periodic, as this makes a circle and not an ellipse, and so a fixed point was found. This is not obvious with only the two first curves.

4.2 Step by Step Computation

Hybrid systems can be very complex and be composed of hundred of variables, which makes their modeling and above all the detection of errors very difficult in these cases. Moreover, an error in the modeling can lead to the reach of an unacceptable state, but not always at the next step. In fact, this wrong state can be reached after many steps was executed. When it occurs, the initial error is impossible to find in the results, due to the large number of trajectories. To address this issue, a step by step mode provides a way to visualize the

intermediate results, in forms of trajectories, whenever a new state is reached. With the displaying of these intermediate results, the user can immediately see that sometimes is going wrong in the model, in which location, and can correct it.

4.3 Forbidden states

The concept of forbidden states is supported by the SpaceEx analysis core. We can use this informations to display the trajectories of the forbidden states that have been reached in a way to differentiate them. If a forbidden state is defined with an inequality on a variable, for example $x \leq 5$ which explains that the variable x must never exceed 5, a line can be displayed to represent this limit. In addition, every trajectory of x that exceeds this limit will be differentiated. Furthermore, it is possible to pause the analysis as soon as a forbidden state is reached, for the user to immediately see it.

4.4 User-Assisted Checking

Another functionality which can help the user to check a model is to provide to it a way for directly interacting with the model checking algorithm. Thus, based on the algorithm given in figure 6, the user could choose the next state to compute successors. For doing this, the *waiting* set, which contains the states that remain to be analyzed, have to be sent to the client and to be displayed. Then, the visualization system displays these states in a different way than the *passed* ones, for them to be differentiated. The user can so choose the next state to be analyzed, for example in clicking on the trajectory that represents it. Furthermore, a representation of the locations can be added to select the next transition in a non-deterministic scenario.

5 Implementation

This section gives some implementation details and choices. It talks about some developed features, then evokes the communication implementation using remote procedure calls (RPC) and the data relative components implementation. So far, about 3K lines of C++ for the server and 6K of Java for the client have been written. Some unit tests have been developed on the client side, for critical parts that concern the trajectories treatment. As these parts have been optimized due to the huge amount of data that can be received, the code is the more complex part and had to be rigorously tested.

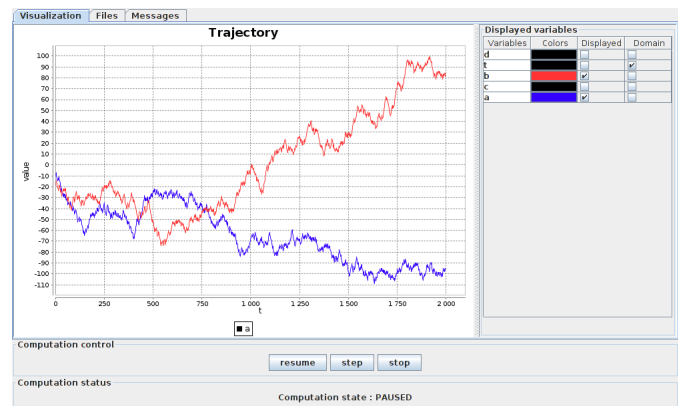
5.1 Developed features

The aimed use case in this internship is the local one, where the user launches both client and server as a single program. To do this, a script has been developed that allows this behavior. A terminate command has been added for this case, which leads to the end of the server when the client terminates. The coordinates of the server, namely IP address and port number, can be passed either by an option file or by argument, the latter being used in the script.

For both client and server, logging systems have been implemented for debugging, allowing severity options (error, warning and debug) and in-file saving. The displayed messages can be filtered by this severity option.

The system was incrementally designed and implemented in making prototypes, each version adding a new feature. For

Figure 9: Screenshot of the current version of the prototype. A trajectory with 5 variables was sent after the computation was paused. This can be seen in the status bar in the bottom of the window.



these prototypes, the analysis core was simulated with one that generates trajectories. For example, the first version just consisted of the connection protocol, another more advanced version allowed to send sine and cosine point evaluations and so on.

For now, some commands was implemented to control the computation. The server provides a control manager to the analysis core, which takes care of the control commands. So the user can launch, pause, stop the computation and execute a step on it in sending the command via the client. On the client side, the visualization system was implemented to display the trajectories. The displayed variables can be chosen, such as the variable used in abscissa. With this feature, each variable can be displayed as a function of any other. The zoom features have also been implemented.

This prototype allows some features of interactive model checking, such as the visualization system and the step by step computation. Furthermore, it is possible for the analysis core, through the server, to send a trajectory as soon as it is computed.

5.2 Used Libraries

Communication

The first objective was to choose a modern, open source and cross language RPC library to support the development of the client/server architecture. A survey of such libraries was made, which is given in the appendix. This survey led to the choice of Apache Thrift, a library which was originally developed for Facebook. The purpose of Thrift is to provide a way for creating services available between different programming languages, and to make the development of clients and servers easy. This is done via a specific IDL (interface definition language) from which some code is generated for providing remote methods implementation. Some reasons led to the choice of Thrift :

- **Modernity** : Thrift was open sourced in 2007, integrated to Apache in 2008 and is still in development.

- **Reliability** : The fact that this library was used for Facebook and taken over by Apache inspires confidence about its quality.
- **Flexibility** : The IDL of Thrift allows to write our own data types without having to serialize them manually, as the generated code does this automatically for us. What's more, this library brings a lot of features, for example threaded servers, SSL sockets...
- **Ease of use** : Thrift is simple to install and use. We just have to write a description of needed data types and services, and then generate the code with the Thrift command. It then generates this code for client, server and sending data, encapsulating all network aspects. We then just have to write the handlers for the method calls we defined in the Thrift file.

Visualization

A plotting library was needed to implement the visualization system of the client. We chose JFreeChart, a library for Java which provides some plotting functionalities. It facilitates the development of the visualization part, for example in providing methods for displaying some curves of different colors, adding legends for them. An adapter has been written to transfer the data, received from the server, to the library form. This adapter provides an independence between the data representation and the used library, allowing to change it with minimum effort.

6 Conclusion

This work is an attempt to improve the interaction with the SpaceEx analysis core, in designing some features such as a step by step mode and a visualization system. Furthermore, these actions can be done through the network with the client-server architecture. That will bring a new way of using SpaceEx, which will make easier the checking of hybrid system models. With these features the user is more implicated in the process and has a better feedback on the analysis. All of this is made in one purpose, which is to facilitate the detection of errors in hybrid systems models, above all the complicated ones with hundred of variables.

Some functionalities relative to interactive model checking has to be added. For example, the user assisted checking has to be developed. For this, more information from the analysis core has to be sent to the client, such as the locations reached by the model. A multi-user functionality has to be added to satisfy all the use cases of the project, such as using the tool in a classroom for a practical session. This will bring new challenges, mainly the definition of a protocol for client identification.

Acknowledgements

I would like to thank Goran Frehse, who was an attentive supervisor, and Olivier Lebeltel for bringing his help. Both gave me a lot of time and some useful advice, not only for this internship but which will also be suitable for further works.

References

- [Alur *et al.*, 1995] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical computer science*, 138(1):3–34, February 1995.
- [Branicky, 2005] Michael S. Branicky. Introduction to hybrid systems. *Handbook of Networked and Embedded Control Systems*, pages 91–116, 2005.
- [Frehse *et al.*, 2011] Goran Frehse, Colas Le Guernic, Alexandre Donze, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, and Antoine Girard Thao Dang Oded Maler. Spaceex : Scalable verification of hybrid systems. *CAV11*, 2011.
- [Frehse, 2009] Goran Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. *International Journal on Software Tools for Technology Transfer (STTT)*, 10(3), June 2009.
- [Frehse, 2010] Goran Frehse. An introduction to spaceex v0.8. 2010.
- [Guernic and Girard, 2010] Colas Le Guernic and Antoine Girard. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems*, 4(2):250–262, 2010.
- [Janssen, 2014] Cory Janssen. Client/server architecture, 2014.

Self-Stabilizing Leader Election in Polynomial Steps^{*†}

Anaïs Durand

VERIMAG UMR 5104, Université Joseph Fourier, Grenoble, France
anaïs.durand@e.ujf-grenoble.fr

Supervised by: Karine Altisen and Stéphane Devismes

Joint work with: Alain Cournier (MIS Lab., Université Picardie Jules Verne, France) and
Franck Petit (LIP6 UMR 7606, INRIA, UPMC Sorbonne Universités, France)

Abstract

In this paper, we propose a silent self-stabilizing leader election algorithm for bidirectional connected identified networks of arbitrary topology. This algorithm is written in the locally shared memory model. It assumes the distributed unfair daemon, the most general scheduling hypothesis of the model. Our algorithm requires no global knowledge on the network (such as an upper bound on the diameter or the number of processes, for example). We show that its stabilization time is in $\Theta(n^3)$ steps in the worst case, where n is the number of processes. Its memory requirement is asymptotically optimal, *i.e.*, $\Theta(\log n)$ bits per processes. Its round complexity is of the same order of magnitude — *i.e.*, $\Theta(n)$ rounds — as the best existing algorithm [Datta *et al.*, 2011b] designed with similar settings. To the best of our knowledge, this is the first self-stabilizing leader election algorithm for arbitrary identified networks that is proven to achieve a stabilization time polynomial in steps. By contrast, we show that the previous best existing algorithm designed with similar settings [Datta *et al.*, 2011b] stabilizes in a non polynomial number of steps in the worst case.

1 Introduction

In distributed computing, the *leader election* problem consists in distinguishing one process, so-called the leader, among the others. We consider here identified networks. So, as it is usually done, we augment the problem by requiring all processes to eventually know the identifier of the leader. The leader election is fundamental as it is a basic component to solve many other important problems, *e.g.*, consensus, spanning tree constructions, implementing broadcasting and convergecasting methods, *etc.*

^{*}This work has been partially supported by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01) funded by the French program Investissement d'avenir and the AGIR project DIAMS.

[†]A preliminary version of this work has been published in SSS'2014 [Altisen *et al.*, 2014a].

Self-stabilization [Dijkstra, 1974; Dolev, 2000] is a versatile technique to withstand *any* transient fault in a distributed system: a self-stabilizing algorithm is able to recover, *i.e.*, reach a legitimate configuration, in finite time, regardless the *arbitrary* initial configuration of the system, and therefore also after the occurrence of transient faults. Thus, self-stabilization makes no hypotheses on the nature or extent of transient faults that could hit the system, and recovers from the effects of those faults in a unified manner. Such versatility comes at a price. After transient faults, there is a finite period of time, called the *stabilization phase*, before the system returns to a legitimate configuration. The *stabilization time* is then the maximum time to reach a legitimate configuration starting from an arbitrary one. Notice that efficiency of self-stabilizing algorithms is mainly evaluated according to their stabilization time and memory requirement.

We consider (deterministic) asynchronous silent self-stabilizing leader election problem in bidirectional, connected, and identified networks of arbitrary topology. We investigate solutions to this problem which are written in the locally shared memory model introduced by Dijkstra [Dijkstra, 1974]. In this model, the distributed unfair daemon is known as the weakest scheduling assumption. Under such an assumption, proving that a given algorithm is self-stabilizing implies that the stabilization time must be finite in terms of atomic steps. However, despite some solutions assuming all these settings (in particular the unfairness assumption) are available in the literature [Datta *et al.*, 2010; 2011a; 2011b], none of them is proven to achieve a polynomial upper bound in steps on its stabilization time. Actually, the time complexities of all these solutions are analyzed in terms of rounds only.

Related Work. In [Dolev *et al.*, 1999], Dolev *et al* showed that silent self-stabilizing leader election requires $\Omega(\log n)$ bits per process, where n is the number of processes. Notice that *non-silent* self-stabilizing leader election can be achieved using less memory, *e.g.*, the non-silent self-stabilizing leader election algorithm for unoriented ring-shaped networks given in [Blin and Tixeuil, 2013] requires $O(\log \log n)$ space per process.

Self-stabilizing leader election algorithms for arbitrary connected identified networks have been proposed in the message-passing model [Afek and Bremler-Barr, 1998;

Awerbuch *et al.*, 1993; Burman and Kuten, 2007]. First, the algorithm of Afek and Bremner [Afek and Bremner-Barr, 1998] stabilizes in $O(n)$ rounds using $\Theta(\log n)$ bits per process. But, it assumes that the link-capacity is bounded by a value B , known by all processes. Two solutions that stabilize in $O(\mathcal{D})$ rounds, where \mathcal{D} is the diameter of the network, have been proposed in [Awerbuch *et al.*, 1993; Burman and Kuten, 2007]. However, both solutions assume that processes know some upper bound D on the diameter \mathcal{D} ; and have a memory requirement in $\Theta(\log D \log n)$ bits.

Several solutions are also given in the shared memory model [Arora and Gouda, 1994; Dolev and Herman, 1997; Datta *et al.*, 2010; 2011a; 2011b; Kravchik and Kuten, 2013]. The algorithm proposed by Dolev and Herman [Dolev and Herman, 1997] is not silent, works under a *fair* daemon, and assume that all processes know a bound N on the number of processes. This solution stabilizes in $O(\mathcal{D})$ rounds using $\Theta(N \log N)$ bits per process. The algorithm of Arora and Gouda [Arora and Gouda, 1994] works under a *weakly fair* daemon and assume the knowledge of some bound N on the number of processes. This solution stabilizes in $O(N)$ rounds using $\Theta(\log N)$ bits per process.

Datta *et al* [Datta *et al.*, 2010] propose the first self-stabilizing leader election algorithm (for arbitrary connected identified networks) proven under the distributed unfair daemon. This algorithm stabilizes in $O(n)$ rounds. However, the space complexity of this algorithm is unbounded. (More precisely, the algorithm requires each process to maintain an unbounded integer in its local memory.)

Solutions in [Datta *et al.*, 2011a; 2011b; Kravchik and Kuten, 2013] have a memory requirement which is asymptotically optimal (*i.e.* in $\Theta(\log n)$). The algorithm proposed by Kravchik and Kuten [Kravchik and Kuten, 2013] assumes a synchronous daemon and the stabilization time of this latter is in $O(\mathcal{D})$ rounds. The two solutions proposed by Datta *et al* in [Datta *et al.*, 2011a; 2011b] assume a distributed unfair daemon and have a stabilization time in $O(n)$ rounds. However, despite these two algorithms stabilize within a finite number of steps (indeed, they are proven assuming an unfair daemon), no step complexity analysis is proposed. Finally, note that the algorithm proposed in [Datta *et al.*, 2011a] assumes that each process has a bit of memory which cannot be arbitrarily corrupted.

Contribution. We propose a silent self-stabilizing leader election algorithm for arbitrary connected and identified networks. Our solution is written in the locally shared memory model assuming a distributed unfair daemon, the weakest scheduling assumption. Our algorithm assumes no knowledge of any global parameter (*e.g.*, an upper bound on \mathcal{D} or n) of network. Like previous solutions of the literature [Datta *et al.*, 2011a; 2011b], it is asymptotically optimal in space (*i.e.*, it can be implemented using $\Theta(\log n)$ bits per process), and it stabilizes in $\Theta(n)$ rounds in the worst case. Yet, contrary to those solutions, we show that our algorithm has a stabilization time in $\Theta(n^3)$ steps in the worst case.

For fair comparison, we have also studied the step complexity of the algorithm given in [Datta *et al.*, 2011b], noted

here $\mathcal{D}\mathcal{L}\mathcal{V}$. This latter is the closest to ours in terms of performance. We show that its stabilization time is not polynomial, *i.e.*, there is no constant α such that the stabilization time of $\mathcal{D}\mathcal{L}\mathcal{V}$ is in $O(n^\alpha)$ steps. More precisely, we show that fixing α to any constant greater than or equal to 4, for every $\beta \geq 2$, there exists a network of $n = 2^{\alpha-1} \times \beta$ processes in which there exists a possible execution that stabilizes in $\Omega(n^\alpha)$ steps. Due to the lack of space, this latter result is not presented here. Refer to the technical report online [Altisen *et al.*, 2014b] for more details.

Roadmap. The next section is dedicated to computational model and basic definitions. In Section 3, we propose our self-stabilizing leader election algorithm. In Section 4, we outline the proof of correctness and the complexity analysis. A detailed proof of correctness and a complete complexity analysis are available in the technical report online [Altisen *et al.*, 2014b]. In Section 5, we detail some experiments. Finally, we conclude in Section 6.

2 Computational model

Distributed systems. We consider *distributed systems* made of n processes. Each process can communicate with a subset of other processes, called its *neighbors*. We denote by \mathcal{N}_p the set of neighbors of process p . Communications are assumed to be bidirectional, *i.e.* $q \in \mathcal{N}_p$ if and only if $p \in \mathcal{N}_q$. Hence, the topology of the system can be represented as a simple undirected connected graph $G = (V, E)$, where V is the set of processes and E is a set of edges representing (direct) communication relations. We assume that each process has a unique ID, a natural integer. IDs are stored using a constant number of bits, b . As commonly done in the literature, we assume that $b = \Theta(\log n)$. Moreover, by an abuse of notation, we identify a process with its ID, whenever convenient. We will also denote by ℓ the process of minimum ID. (So, the minimum ID will be also denoted by ℓ .)

Locally shared memory model. We consider the *locally shared memory model* in which the processes communicate using a finite number of locally shared registers, called *variables*. Each process can read its own variables and those of its neighbors, but can only write to its own variables. The *state* of a process is the vector of values of all its variables. A configuration γ of the system is the vector of states of all processes. We denote by \mathcal{C} the set of all possible configurations.

A distributed *algorithm* consists of one *program* per process. The program of a process p is a finite set of *actions* of the following form: $\langle \text{label} \rangle :: \langle \text{guard} \rangle \rightarrow \langle \text{statement} \rangle$. The *labels* are used to identify actions. The *guard* of an action in the program of process p is a Boolean expression involving the variables of p and its neighbors. If the guard of some action evaluates to true, then the action is said to be *enabled* at p . By extension, if at least one action is enabled at p , p is said to be enabled. We denote by $Enabled(\gamma)$ the set of processes enabled in configuration γ . The *statement* of an action is a sequence of assignments on the variables of p . An action can be executed only if it is enabled. In this case, the execution of the action consists in executing its statement.

The asynchronism of the system is materialized by an adversary, called the *daemon*. In a configuration γ , if $Enabled(\gamma) \neq \emptyset$, then the daemon selects a non empty subset S of $Enabled(\gamma)$ to perform an (*atomic*) step: $\forall p \in S$, p atomically executes one of its actions enabled in γ , leading the system to a new configuration γ' . We denote by \mapsto the relation between configurations such that $\gamma \mapsto \gamma'$ if and only if γ' can be reached from γ in one (atomic) step. An *execution* is then a *maximal* sequence of configurations $\gamma_0, \gamma_1, \dots$ such that $\gamma_{i-1} \mapsto \gamma_i, \forall i > 0$. The term “maximal” means that the execution is either infinite, or ends at a *terminal* configuration γ in which $Enabled(\gamma)$ is empty.

In this paper, the daemon is supposed to be *distributed* and *unfair*. “Distributed” means that while the configuration is not terminal, the daemon should select at least one enabled process, maybe more. “Unfair” means that there is no fairness constraint, *i.e.*, the daemon might never permit an enabled process to execute, unless it is the only enabled process.

Rounds. To measure the time complexity of an algorithm, we also use the notion of *round*. This latter allows to highlight the execution time according to the speed of the slowest process. If a process p is enabled in a configuration γ_i but not enabled in the next configuration γ_{i+1} and does not execute any action between γ_i and γ_{i+1} , we said that p is *neutralized* during the step $\gamma_i \mapsto \gamma_{i+1}$. The first round of an execution e , noted e' , is the minimal prefix of e in which every process that is enabled in the initial configuration either executes an action or becomes neutralized. Let e'' be the suffix of e starting from the last configuration of e' . The second round of e is the first round of e'' , and so forth.

Self-stabilization. Let \mathcal{A} be a distributed algorithm. Let \mathcal{E} be the set of all possible executions of \mathcal{A} . A *specification* SP is a predicate over \mathcal{E} .

\mathcal{A} is *self-stabilizing* for SP if and only if there exists a non-empty subset of configurations $\mathcal{L} \subseteq \mathcal{C}$, called *legitimate* configurations, such that:

- *Closure*: $\forall e \in \mathcal{E}$, for each step $\gamma_i \mapsto \gamma_{i+1} \in e$, $\gamma_i \in \mathcal{L} \Rightarrow \gamma_{i+1} \in \mathcal{L}$.
- *Convergence*: $\forall e \in \mathcal{E}, \exists \gamma \in e$ such that $\gamma \in \mathcal{L}$.
- *Correction*: $\forall e \in \mathcal{E}$ such that e starts in a legitimate configuration $\gamma \in \mathcal{L}$, e satisfies SP .

Every configuration that is not legitimate is called *illegitimate*. The *stabilization time* is the maximum time (in steps or rounds) to reach a legitimate configuration starting from any configuration.

Self-stabilizing leader election. We define SP_{LE} the specification of the leader election problem. Let $Leader : V \mapsto \mathbb{N}$ be a function defined on the state of any process $p \in V$ in the current configuration that returns the ID of the leader appointed by p . An execution $e \in \mathcal{E}$ satisfies SP_{LE} if and only if:

1. For all configuration $\gamma \in e$, $\forall p, q \in V$, $Leader(p) = Leader(q)$ and $Leader(p)$ is the ID of some process in V .

2. For all step $\gamma_i \mapsto \gamma_{i+1} \in e$, $\forall p \in V$, $Leader(p)$ has the same value in γ_i and γ_{i+1} .

An algorithm \mathcal{A} is *silent* if and only if every execution is finite [Dolev *et al.*, 1999]. Let γ be a terminal configuration. The set of all possible executions starting from γ is the singleton $\{\gamma\}$. So, if \mathcal{A} is self-stabilizing and silent, γ must be legitimate. Thus, to prove that a leader election algorithm is both self-stabilizing and silent, it is necessary and sufficient to show that: (1) in every terminal configuration γ , $\forall p, q \in V$, $Leader(p) = Leader(q)$ and $Leader(p)$ is the ID of some process; (2) every execution is finite.

3 Algorithm \mathcal{LE}

In this section, we present a silent and self-stabilizing leader election algorithm, called \mathcal{LE} . Its formal code is given in Algorithm 1. Starting from an arbitrary configuration, \mathcal{LE} converges to a terminal configuration, where the process of minimum ID, ℓ , is elected. More precisely, in the terminal configuration, every process p knows the identifier of ℓ thanks to its local variable $p.idR$. This means that, in particular, we instantiate the function $Leader$ of the specification as follows: $Leader(p) = p.idR$, $\forall p \in V$. Moreover, a spanning tree rooted at ℓ is defined using two variables per process: par and $level$. First, $\ell.par = \ell$ and $\ell.level = 0$. Then, for every process $p \neq \ell$, $p.par$ points to the parent of p in the tree and $p.level$ is the level of p in the tree.

We now present a simple algorithm for the leader election in Subsection 3.1. We show why this algorithm is not self-stabilizing in Subsection 3.2. We explain in Subsection 3.3 how to modify this algorithm to make it self-stabilizing.

3.1 Non Self-Stabilizing Leader Election

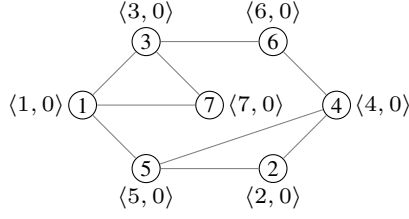
We first consider a simplified version of \mathcal{LE} . Starting from a predefined initial configuration, it elects ℓ in all idR variables and builds a spanning tree rooted at ℓ . Initially, every process p declares itself as leader: $p.idR = p$, $p.par = p$, and $p.level = 0$. So, p satisfies the two following predicates: $SelfRoot(p) \equiv (p.par = p)$ and $SelfRootOk'(p) \equiv (p.level = 0) \wedge (p.idR = p)$. Note that, in the sequel, we say that p is a *self root* when $SelfRoot(p)$ holds. From such an initial configuration, our non self-stabilizing algorithm consists in the following single action:

$$\begin{aligned} J\text{-Action}' &:: \exists q \in \mathcal{N}_p, (q.idR < p.idR) \\ &\rightarrow p.par \leftarrow \min_{\preceq} \{q \in \mathcal{N}_p\}; \\ & \quad p.idR \leftarrow p.par.idR; \\ & \quad p.level \leftarrow p.par.level + 1; \end{aligned}$$

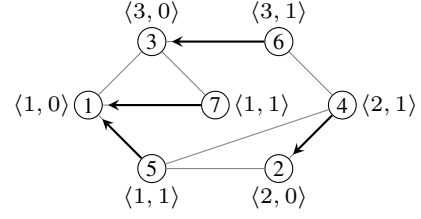
$$\text{where } \forall x, y \in V, x \preceq y \Leftrightarrow (x.idR \leq y.idR) \wedge [(x.idR = y.idR) \Rightarrow (x < y)]$$

Informally, when p discovers that $p.idR$ is not equal to the minimum identifier, it updates its variables accordingly. Let q be the neighbor of p having idR minimum. Then, p selects q as new parent ($p.par \leftarrow q$ and $p.level \leftarrow p.par.level + 1$) and sets $p.idR$ to the value of $q.idR$. If there are several neighbors having idR minimum, the identifiers of those neighbors are used to break ties.

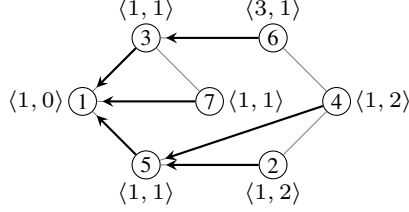
Hence, the identifier of ℓ is propagated, from neighbors to neighbors, into the idR variables and the system reaches a



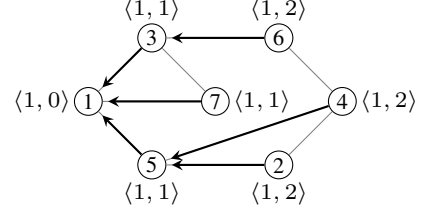
(a) Initial configuration. $SelfRoot(p) \wedge SelfRootOk'(p)$ holds for every process p .



(b) 4, 5, 6, and 7 have executed $J-Action'$. Note that $J-Action'$ was not enabled at 2 because it is a local minimum.



(c) 2, 3, and 4 have executed $J-Action'$. 3 joins the tree rooted at 1, but the new value of $3.idR$ is not yet propagated to its child 6.



(d) 6 has executed $J-Action'$. The configuration is now terminal, $\ell = 1$ is elected, and a tree rooted at ℓ is available.

Figure 1: An example showing an execution of the non self-stabilizing algorithm. Process identifiers are given inside the nodes. $\langle x, y \rangle$ means $idR = x$ and $level = y$. Arrows represent par pointers. The absence of arrow means that the process is a self root.

terminal configuration in $O(D)$ rounds. Figure 1 shows an example of such an execution.

Notice first that for every process p , $p.idR$ is always less than or equal to its own identifier. Indeed, $p.idR$ is initialized to p and decreases each time p executes $J-Action'$. Hence, $p.idR = p$ while p is a self root and after p executes $J-Action'$ for the first time, $p.idR$ is smaller than its ID forever.

Second, even in this simplified context, for each two neighbors p and q such that q is the parent of p , it may happen that $p.idR$ is greater than $q.idR$ —an example is shown in Figure 1c, where $p = 6$ and $q = 3$. This is due to the fact that p joins the tree of q but meanwhile q joins another tree and this change is not yet propagated to p . Similarly, when $p.idR \neq q.idR$, $p.level$ may be different from $q.level + 1$.

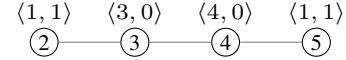
According to those remarks, we can deduce that when $p.par = q$ with $q \neq p$, we have the following relation between p and q :

$$\begin{aligned} GoodIdR(p, q) &\equiv (p.idR \geq q.idR) \wedge (p.idR < p) \\ GoodLevel(p, q) &\equiv (p.idR = q.idR) \\ &\Rightarrow (p.level = q.level + 1) \end{aligned}$$

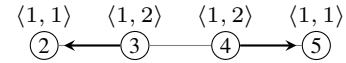
3.2 Fake IDs

The algorithm presented in Subsection 3.1 is clearly not self-stabilizing. Indeed, in a self-stabilization context, the execution may start in any arbitrary configuration. In particular, idR variables can be initialized to arbitrary natural integer values, even values that are actually not IDs of (existing) processes. We call such values *fake IDs*.

The existence of fake IDs may lead the system to an illegitimate terminal configuration. Refer to the example of execution given in Figure 2: starting from the configuration



(a) Illegitimate initial configuration, where 2 and 5 have fake idR .



(b) 3 and 4 executed $J-Action'$. The configuration is now terminal.

Figure 2: Example of execution that does not converge to a legitimate configuration.

in 2a, if processes 3 and 4 move, the system reaches the terminal configuration given in 2b, where there are two trees and the idR variables elect the fake ID 1. In this example, 2 and 5 can detect the problem. Indeed, predicate $SelfRootOk'$ is violated by both 2 and 5. One may believe that it is sufficient to reset the local state of processes which detect inconsistency (here processes 2 and 5) to $p.idR \leftarrow p$, $p.par \leftarrow p$ and $p.level \leftarrow 0$. After these resets, there are still some errors, as shown on Figure 3.

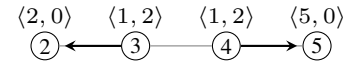


Figure 3: One step after Figure 2b, 2 and 5 have reset.

Again, 3 and 4 can detect the problem. Indeed, predicate $GoodIdR(p, p.par) \wedge GoodLevel(p, p.par)$ is violated by both 3 and 4. In this example, after 3 and 4 have reset, all inconsistencies have been removed. So let define the following action:

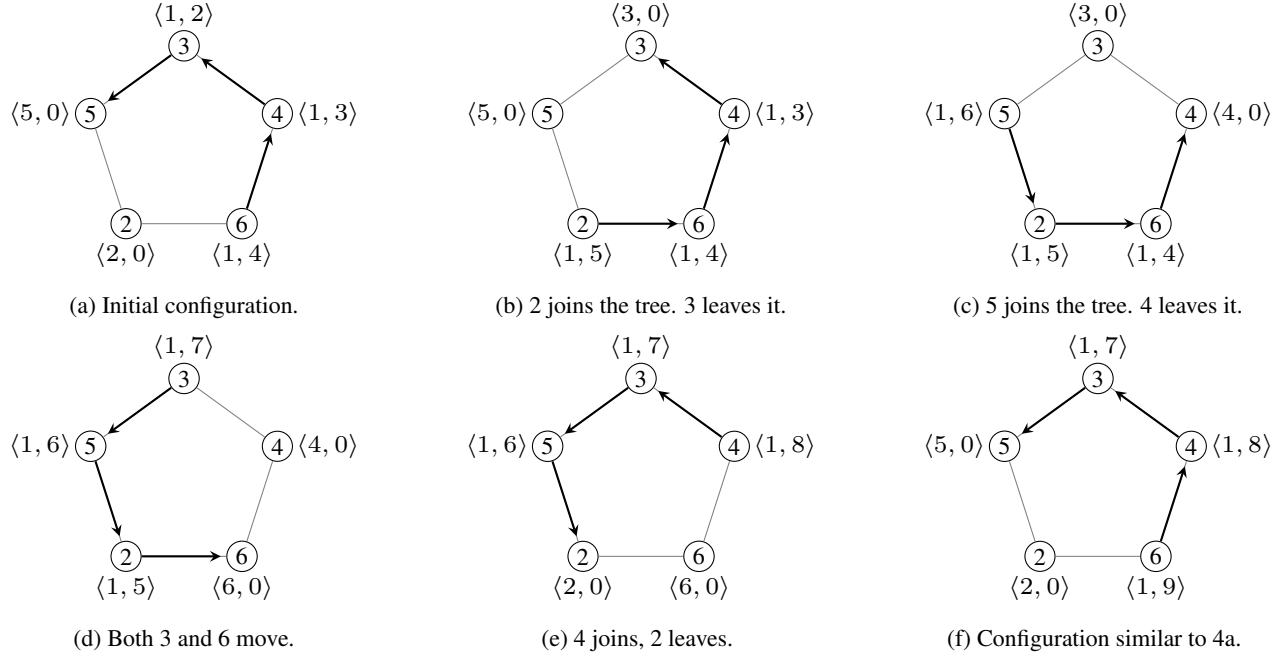


Figure 4: The first process of the chain of bold arrows violates the predicate $SelfRootOk'$ and resets by executing $R-Action'$, while another process joins its tree. This cycle of resets and joins might never terminate.

$$\begin{aligned}
R-Action' &:: (SelfRoot(p) \wedge \neg SelfRootOk'(p)) \\
&\vee (\neg SelfRoot(p) \wedge \neg (GoodIdR(p, p.par) \\
&\wedge GoodLevel(p, p.par))) \\
&\rightarrow p.par \leftarrow p; p.idR \leftarrow p; p.level \leftarrow 0;
\end{aligned}$$

Unfortunately, this additional action does not ensure the convergence in all cases—refer to the example in Figure 4. Indeed, if a process resets, it becomes a self root but this does not erase the fake ID in the rest of its subtree. Then, another process can join the tree and adopt the fake ID which will be further propagated, and so on. In the example, a process resets while another joins its tree at lower level, and this leads to endless erroneous behavior, since we do not want to assume any maximal value for *level* (such an assumption would otherwise imply the knowledge of some upper bound on *n*). Therefore, the whole tree must be reset, instead of its root only. To that goal, we first freeze the “abnormal” tree in order to forbid any process to join it, then the tree is reset top-down. The cleaning mechanism is detailed in the next subsection.

3.3 Cleaning Abnormal Trees

To introduce the trees, we define what is a “good relation” between a parent and its children. Namely, the predicate $KinshipOk'(p, q)$ models that a process p is a *real child* of its parent $q = p.par$. This predicate holds if and only if $GoodLevel(p, q)$ and $GoodIdR(p, q)$ are true. This relation defines a spanning forest: a *tree* is a maximal set of processes connected by *par* pointers and satisfying $KinshipOk'$ relation. A process p is a root of such a tree whenever $SelfRoot(p)$ holds or $KinshipOk'(p, p.par)$ is false. When $SelfRoot(p) \wedge SelfRootOk'(p)$ is true, p is a *normal root* just as in the non self-stabilizing case. In other cases, there is

an error and p is called an *abnormal root*: $AbRoot'(p) \equiv (SelfRoot(p) \wedge \neg SelfRootOk'(p)) \vee (\neg SelfRoot(p) \wedge \neg KinshipOk'(p, p.par))$. A tree is said to be *abnormal* (resp. *normal*) when its root is abnormal (resp. normal).

We now detail the different predicates and actions of Algorithm 1.

Variable *status*. Abnormal trees need to be frozen before to be cleaned in order to prevent them from growing endlessly (see 3.2). This mechanism is achieved using an additional variable, *status*, that is used as follows. If a process is clean (*i.e.*, not involved into any freezing operation), then its *status* is C . Otherwise, it has status EB or EF and no neighbor can select it as its parent. These two latter states are actually used to perform a “Propagation of Information with Feedback” [Chang, 1982] into the abnormal trees. Status EB means “Error Broadcast” and EF means “Error Feedback”. From an abnormal root, the status EB is broadcast down in the tree. Then, once the EB wave reaches a leaf, the leaf initiates a convergecast EF -wave. Once the EF -wave reaches the abnormal root, the tree is said to be *dead*, meaning that there is no process of status C in the tree and no other process can join it. So, the tree can be safely reset from the abnormal root toward the leaves. Notice that the new variable *status* may also get arbitrary initialization. Thus, we enforce previously introduced predicates as follows. A self root must have status C , otherwise it is an abnormal root:

$$SelfRootOk(p) \equiv SelfRootOk'(p) \wedge (p.status = C)$$

To be a real child of q , p should have a status coherent with the one of q . This is expressed with the

Algorithm 1 Algorithm \mathcal{LE} for every process p

Variables: $p.idR \in \mathbb{N}; p.par \in \mathcal{N}_p \cup \{p\}; p.level \in \mathbb{N}; p.status \in \{C, EB, EF\};$ **Macros:**

$$\begin{aligned} Children_p &\equiv \{q \in \mathcal{N}_p \mid q.par = p\} \\ RealChildren_p &\equiv \{q \in Children_p \mid KinshipOk(q, p)\} \\ p \preceq q &\equiv (p.idR \leq q.idR) \wedge [(p.idR = q.idR) \Rightarrow (p \leq q)] \\ Min_p &\equiv \min_{\preceq} \{q \in \mathcal{N}_p \mid q.status = C\} \end{aligned}$$

Predicates:

$$\begin{aligned} SelfRoot(p) &\equiv p.par = p \\ SelfRootOk(p) &\equiv (p.level = 0) \wedge (p.idR = p) \wedge (p.status = C) \\ GoodIdR(s, f) &\equiv (s.idR \geq f.idR) \wedge (s.idR < s) \\ GoodLevel(s, f) &\equiv (s.idR = f.idR) \Rightarrow (s.level = f.level + 1) \\ GoodStatus(s, f) &\equiv [(s.status = EB) \Rightarrow (f.status = EB)] \vee [(s.status = EF) \Rightarrow (f.status \neq C)] \\ &\quad \vee [(s.status = C) \Rightarrow (f.status \neq EF)] \\ KinshipOk(s, f) &\equiv GoodIdR(s, f) \wedge GoodLevel(s, f) \wedge GoodStatus(s, f) \\ AbRoot(p) &\equiv [SelfRoot(p) \wedge \neg SelfRootOk(p)] \vee [\neg SelfRoot(p) \wedge \neg KinshipOk(p, p.par)] \\ Allowed(p) &\equiv \forall q \in Children_p, (\neg KinshipOk(q, p) \Rightarrow q.status \neq C) \end{aligned}$$

Guards:

$$\begin{aligned} EBroadcast(p) &\equiv (p.status = C) \wedge [AbRoot(p) \vee (p.par.status = EB)] \\ EFeedback(p) &\equiv (p.status = EB) \wedge (\forall q \in RealChildren_p, q.status = EF) \\ Reset(p) &\equiv (p.status = EF) \wedge AbRoot(p) \wedge Allowed(p) \\ Join(p) &\equiv (p.status = C) \wedge [\exists q \in \mathcal{N}_p, (q.idR < p.idR) \wedge (q.status = C)] \wedge Allowed(p) \end{aligned}$$

Actions:

$$\begin{aligned} EB\text{-action} &:: EBroadcast(p) &&\rightarrow p.status \leftarrow EB; \\ EF\text{-action} &:: EFeedback(p) &&\rightarrow p.status \leftarrow EF; \\ R\text{-action} &:: Reset(p) &&\rightarrow p.status \leftarrow C; p.par \leftarrow p; p.idR \leftarrow p; p.level \leftarrow 0; \\ J\text{-action} &:: Join(p) \wedge \neg EBroadcast(p) &&\rightarrow p.par \leftarrow Min_p; p.idR \leftarrow p.par.idR; p.level \leftarrow p.par.level + 1; \end{aligned}$$

predicate $GoodStatus(p, q)$ which is used to enforce the $KinshipOk(p, q)$ relation:

$$\begin{aligned} GoodStatus(p, q) &\equiv [(p.status = EB) \Rightarrow (q.status = EB)] \vee [(p.status = EF) \Rightarrow (q.status \neq C)] \\ &\quad \vee [(p.status = C) \Rightarrow (q.status \neq EF)] \\ KinshipOk(p, q) &\equiv KinshipOk'(p, q) \wedge \\ &\quad GoodStatus(p, q) \end{aligned}$$

Precisely, when p has status C , its parent must have status C or EB (if the EB -wave is not propagated yet to p). If p has status EB , then the status of its parent must be EB because p gets status EB from its parent q and q will change its status to EF only after p gets status EF . Finally, if p has status EF , its parent can have status EB (if the EF -wave is not propagated yet to its parent) or EF .

Normal Execution. Remark that, after all abnormal trees have been removed, all processes have status C and the algorithm works as in the initial version. Notice that the guard of J -action has been enforced so that only processes with status C and which are not abnormal root can execute it, and when executing J -action, a process can only choose a neighbor of status C as parent. Moreover, remark that the cleaning of all abnormal trees does not ensure that all fake IDs have been removed. Rather, it guarantees the removal of all fake IDs smaller than ℓ . This implies that (at least) ℓ is a self root at the end of the cleaning and all other processes will elect ℓ within the next \mathcal{D} rounds.

Cleaning Abnormal Trees. Figure 5 shows how an abnormal tree is cleaned. In the first phase (see Figure 5a), the

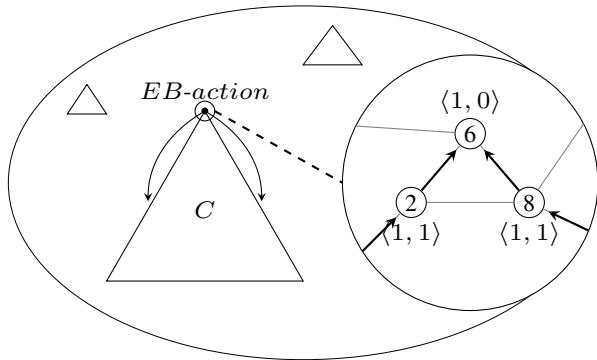
root broadcasts status EB down to its (abnormal) tree: all the processes in this tree execute EB -action, switch to status EB and are consequently informed that they are in an abnormal tree. The second phase starts when the EB -wave reaches a leaf. Then, a convergecast wave of status EF is initiated thanks to action EF -action (see Figure 5b). The system is asynchronous, hence all the processes along some branch can have status EF before the broadcast of the EB -wave is done into another branch. In this case, the parent of these two branches waits that all its children in the tree (processes in the set $RealChildren$) get status EF before executing EF -action (Figure 5c). When the root gets status EF , all processes have status EF : the tree is dead. Then (third phase), the root can reset (safely) to become a self root by executing R -action (Figure 5e). Its former real children (of status EF) become themselves abnormal roots of dead trees (Figure 5f) and reset.

Finally, we used the predicate $Allowed(p)$ to temporarily lock the parent of p in two particular situations — illustrated in Figure 6 — where p is enabled to switch its status from C to EB . These locks impact neither the correctness nor the complexity of \mathcal{LE} . Rather, they allow us to simplify the proofs by ensuring that, once enabled, EB -action remains continuously enabled until executed.

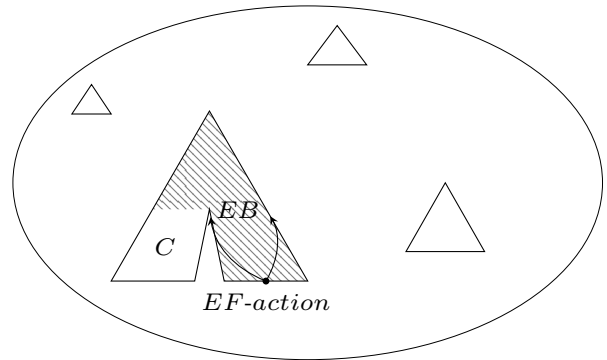
4 Correctness and Complexity Analysis

First, remark that idR and $level$ can be stored in $\Theta(\log n)$ bits. So, the memory requirement of \mathcal{LE} is $\Theta(\log n)$ bits per process.

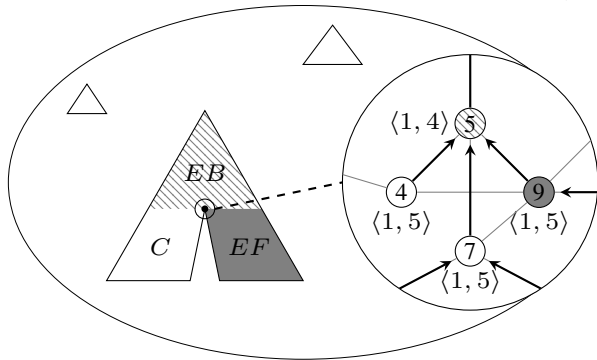
Let us first distinguish between *clean* and *dirty* configurations. Given any configuration γ , γ is *clean* if and only if in γ , $\forall p \in V, \neg EBroadcast(p) \wedge p.status = C$. In other



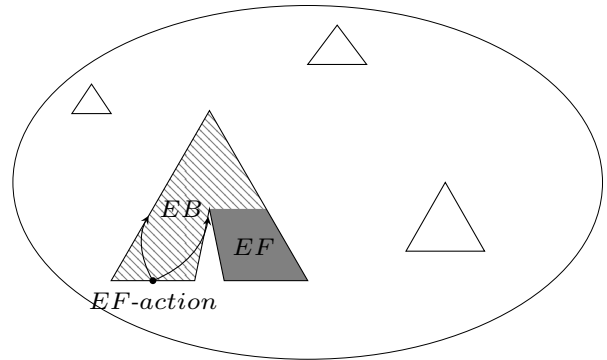
(a) When an abnormal root detects an error, it executes *EB-action*. The *EB-wave* is broadcast to the leaves. Here, 6 is an abnormal root because it is a self root and its *idR* is different from its ID ($1 \neq 6$).



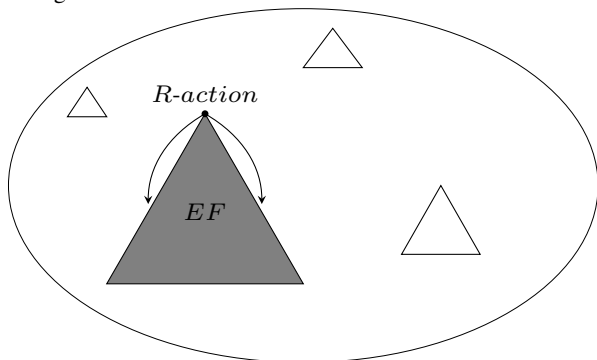
(b) When the *EB-wave* reaches a leaf, it executes *EF-action*. The *EF-wave* is propagated up to the root.



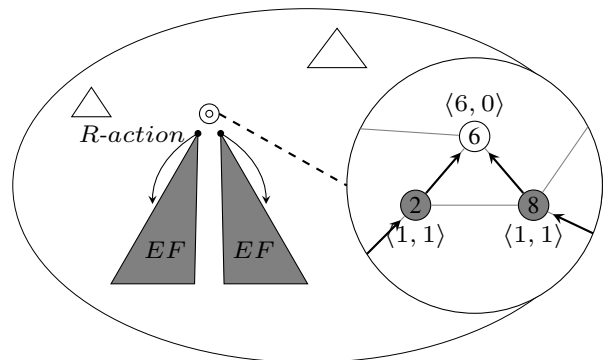
(c) It may happen that the *EF-wave* reaches a node, here process 5, even though the *EB-wave* is still broadcasting into some of its proper subtrees: 5 must wait that the status of 4 and 7 become *EF* before executing *EF-action*.



(d) *EB-wave* has been propagated in the other branch. An *EF-wave* is initiated by the leaves.

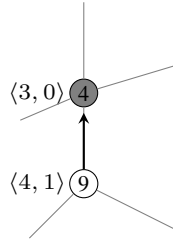


(e) *EF-wave* reaches the root. The root can safely reset (*R-action*) because its tree is dead. The cleaning wave is propagated down to the leaves.

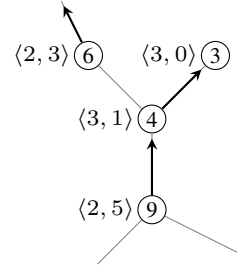


(f) Its children become themselves abnormal roots of dead trees and can execute *R-action*: 2 and 8 can clean because their status is *EF* and their parent has status *C*.

Figure 5: Schematic example of the cleaning mechanism. Trees are filled according to the status of their processes: white for *C*, dashed for *EB*, gray for *EF*.



(a) 4 and 9 are abnormal roots. If 4 executes R -action before 9 executes EB -action, the kinship relation between 4 and 9 becomes correct and 9 is no more an abnormal root. Then, EB -action is no more enabled at 9.



(b) 9 is an abnormal root and Min_4 is 6. If 4 executes J -action before 9 executes EB -action, the kinship relation between 4 and 9 becomes correct and 9 is no more an abnormal root. Then, EB -action is no more enabled at 9.

Figure 6: Example of situations where the parent of a process is locked.

words, a configuration is clean if and only if it contains no abnormal trees. In particular, such a clean configuration does not contain fake IDs smaller than ℓ . Any configuration that is not clean is said to be *dirty*.

4.1 Correctness and Stabilization Time in Steps

Convergence from a clean configuration. Let us first consider any *clean* configuration, γ . As γ is clean, γ may contain some fake IDs, but all of them (if any) are greater than ℓ . This implies, in particular, that ℓ is a self root and $\ell.idR = \ell$ forever from γ . Moreover, in γ there are at most n different values disseminated into the idR variables. Every process $p \neq \ell$ can only decrease its own value of idR by executing J -action (all other actions are disabled forever at p because they deal with abnormal trees). Hence, overall after at most $\frac{(n-1) \times (n-2)}{2}$ executions of J -action, the configuration is terminal and ℓ is elected.

Convergence from an arbitrary configuration. The remainder of the proof consists in showing that, from any arbitrary configuration, a clean configuration is reached in $O(n^3)$ steps. So, let consider a dirty configuration γ . Then, γ contains some abnormal trees. In the following, we say that a process p is called *alive* if and only if $p.status = C$. Otherwise, it is said to be *dead*. By extension, a tree T is called an *alive tree* if and only if $\exists p \in T$ such that p is alive. Otherwise, it is called a *dead tree*.

We first show that no abnormal alive tree can be created from γ . So, as there are at most n abnormal alive trees in the initial configuration, and each of them may contain up to n processes, at most n^2 EB -action, EF -action, and R -action respectively are sufficient to freeze and remove all them. Notice that this way we clean abnormal trees is the main difference between our algorithm \mathcal{LE} and the algorithm proposed in [Datta *et al.*, 2011b], \mathcal{DLV} . Indeed, we have shown that, contrary to \mathcal{LE} , the correction mechanism implemented in \mathcal{DLV} can involve a non-polynomial number of correction actions (see [Altisen *et al.*, 2014b]).

Nevertheless, processes can execute J -action during the removal of abnormal trees. In particular, a process p can leave an abnormal alive tree T by executing J -action to

join another (normal or abnormal) tree. However, in this case the value of $p.idR$ necessarily decreases. Later p can join T again, but this may happen only if p executes actions EB -action, EF -action, and R -action at least once in the meantime. This means that p participates to the removal of some abnormal tree. Thus, each time p joins T again, the number of abnormal trees decreases, *i.e.*, p can join and leave T at most $n - 1$ times.

Thus, each process (n) can join each abnormal tree (at most n) at most $n - 1$ times using J -action which gives an overall number of J -actions in $O(n^3)$.

To sum up, starting from any configuration, a terminal configuration where ℓ is elected is reached in $O(n^3)$ steps. (We prove a tighter bound in [Altisen *et al.*, 2014b].)

4.2 Stabilization Time in Rounds

Let us consider a clean configuration γ . Again, γ may contain some fake IDs, but all of them (if any) are greater than ℓ . This implies, in particular, that ℓ is a self root and $\ell.idR = \ell$ forever from γ . ℓ being the minimum value in idR variables, ℓ is propagated, from neighbors to neighbors, into the idR variables and the system reaches a terminal configuration in $O(\mathcal{D})$ rounds.

Consider now a dirty configuration γ . From γ , all abnormal trees are frozen and removed in parallel using three waves: (1) the broadcast of the value EB from the abnormal roots to the leaves, (2) a convergecast of the value EF from the leaves to the abnormal roots, and (3) finally, the cleaning is performed top-down. As the maximum height of a tree is n , each of these waves is done in at most n rounds. Overall, abnormal trees are removed in at most $3n$ rounds.

Hence, the stabilization time is at most $3n + \mathcal{D}$ rounds.

5 Experimentation

In this section, we evaluate the average performances of algorithm \mathcal{LE} in terms of rounds and steps. This work is still in progress.

5.1 Experimentation Protocol

We generate pools of random graphs using the *Unit Disk Graph* (UDG) model [Huson and Sen, 1995]. In a UDG, a

node is connected with all the other nodes in a disk around it. In other words, two nodes are connected if and only if the Euclidean distance between them is smaller than some radius. Wireless sensor networks can be roughly modeled using UDG where the radius of the disk is the transmission range of the sensor emitter.

We use a simulator dedicated to locally shared memory model. An enabled process p is selected by the daemon with an exponential distribution of parameter $\frac{1}{\alpha} = \frac{1}{4}$ (The expected time before activation is 4.) for every process.

The initialization of the processes is also randomized:

- The number of fake ids smaller than ℓ , denoted n_f , equals to 10% of n .
- Each process has a unique random id between n_f and $n + n_f - 1$.
- n_f processes (uniformly chosen) have a random idR between 0 and $n_f - 1$. The other processes have a random idR between n_f and $n + n_f$.
- The par pointer is uniformly chosen among the neighbors of the node and itself.
- The $level$ is uniformly chosen between 0 and an arbitrary value.
- The $status$ is uniformly chosen.

5.2 Average stabilization time in rounds

An experimental analysis was realized to evaluate the average performances of \mathcal{LE} in terms of rounds.

We generate a pool of twenty random UDGs of $n = 1000$ nodes for each value of the diameter between 4 and 27. We execute \mathcal{LE} five times on each graph of the pool, until the confidence interval is smaller than 2% of the average stabilization time in rounds.

Figure 7 shows the results. The average stabilization time in rounds is drastically smaller (the order of magnitude is the diameter) than the analytical bound in the worst case of exactly $3n + \mathcal{D}$ rounds. So the worst case seems to be rare in this class of graph.

5.3 Average stabilization time in steps

An experimental analysis was realized to evaluate the average performances of \mathcal{LE} in terms of steps.

We generate a pool of ten random UDGs of diameter close to 15 (15 ± 1) for $n = 100, 200, \dots$, to $n = 1000$ nodes. We execute \mathcal{LE} five times on each graph of the pool, until the confidence interval is smaller than 2% of the average stabilization time in steps.

Figure 7 shows the results. The average stabilization time in steps is drastically smaller than the analytical bound in the worst case of $\Theta(n^3)$ steps. So the worst case seems to be rare in this class of graph.

5.4 Work in progress

Other experiments have been done by inserting faults in a terminal configuration in order to measure the impact of the number of faults on the stabilization time. But, this work needs further investigation.

The same experimental analysis was also done on another model of random graphs: *Barabási-Albert graphs*. The Barabási-Albert model [Albert and Barabási, 2001] generates

random *scale-free* networks (*i.e.*, networks with a power-law degree distributions) similar to a lot of actual systems, the Internet for example. It models *preferential attachment*: a node with high degree receives new links with a bigger probability than a node with smaller degree. Again, the average stabilization time in rounds and in steps is drastically smaller than the analytical bounds but this work needs further investigation, in particular on the link between the performances of \mathcal{LE} and the density of the graphs.

6 Conclusion

We proposed a silent self-stabilizing leader election algorithm, called \mathcal{LE} , for bidirectional connected identified networks of arbitrary topology. Starting from any arbitrary configuration, \mathcal{LE} converges to a terminal configuration, where all processes know the ID of the leader, this latter being the process of minimum ID. Moreover, as in most of the solutions from the literature, a distributed spanning tree rooted at the leader is defined in the terminal configuration.

\mathcal{LE} is written in the locally shared memory model. It assumes the distributed unfair daemon, the most general scheduling hypothesis of the model. Moreover, it requires no global knowledge on the network (such as an upper bound on the diameter or the number of processes, for example). \mathcal{LE} is asymptotically optimal in space, as it requires $\Theta(\log n)$ bits per process, where n is the size of the network. We analyzed its stabilization time both in rounds and steps. We showed that \mathcal{LE} stabilizes in at most $3n + \mathcal{D}$ rounds, where \mathcal{D} is the diameter of the network. We have also proven in the technical report [Altisen *et al.*, 2014b] that for every $n \geq 4$, for every \mathcal{D} , $2 \leq \mathcal{D} \leq n - 2$, there is a network of n processes in which a possible execution exactly lasts this complexity. Finally, we proved that \mathcal{LE} achieves a stabilization time polynomial in steps. More precisely, we have shown in the technical report [Altisen *et al.*, 2014b] that its stabilization time is at most $\frac{n^3}{2} + 2n^2 + \frac{n}{2} + 1$ steps. Still in [Altisen *et al.*, 2014b], we have shown that for every $n \geq 4$, there exists a network of n processes (and of diameter 2) in which a possible execution exactly lasts $\frac{n^3}{6} + \frac{5}{2}n^2 - \frac{11}{3}n + 2$ steps, establishing then that the worst case is in $\Theta(n^3)$.

We have also implemented \mathcal{LE} in a high-level simulator to empirically evaluate its average performances. Experimental results tend to show that its worst cases in terms of rounds (exactly $3n + \mathcal{D}$ rounds) and in terms of steps ($\Theta(n^3)$ steps) are rare.

Perspectives of this work deal with complexity issues. In [Datta *et al.*, 2011b], Datta *et al* showed that it is easy to implement a silent self-stabilizing leader election which works assuming an unfair daemon, uses $\Theta(\log n)$ bits per process, and stabilizes in $O(\mathcal{D})$ rounds (where \mathcal{D} is an upper bound on \mathcal{D}). Nevertheless, processes are assumed to *know* \mathcal{D} . It is worth investigating whether it is possible to design an algorithm which works assuming an unfair daemon, uses $\Theta(\log n)$ bits per process, and stabilizes in $O(\mathcal{D})$ rounds without using any global knowledge. We believe this problem remains difficult, even adding some fairness assumption.

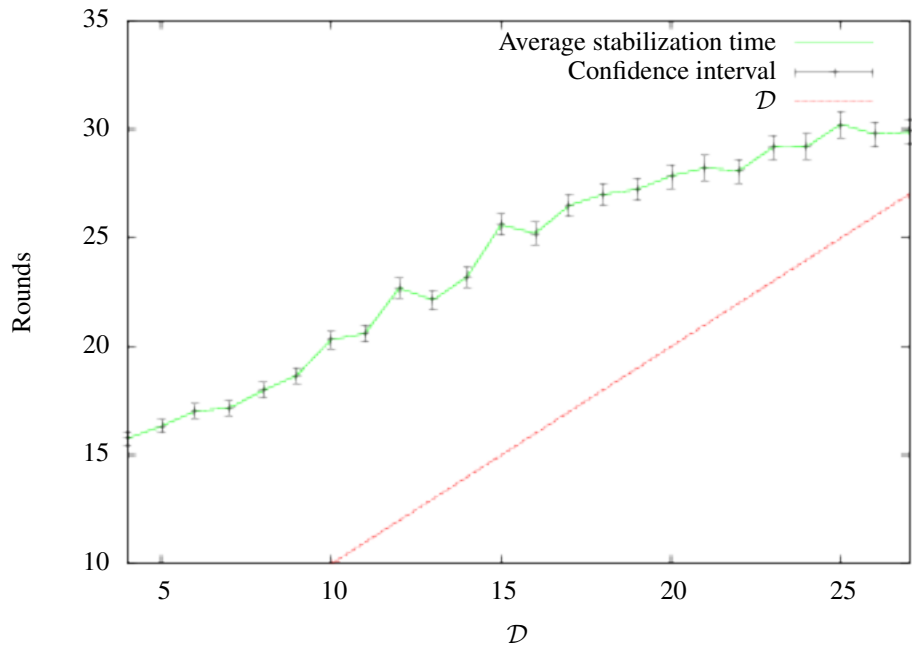


Figure 7: Average stabilization time in rounds ($n = 1000$).

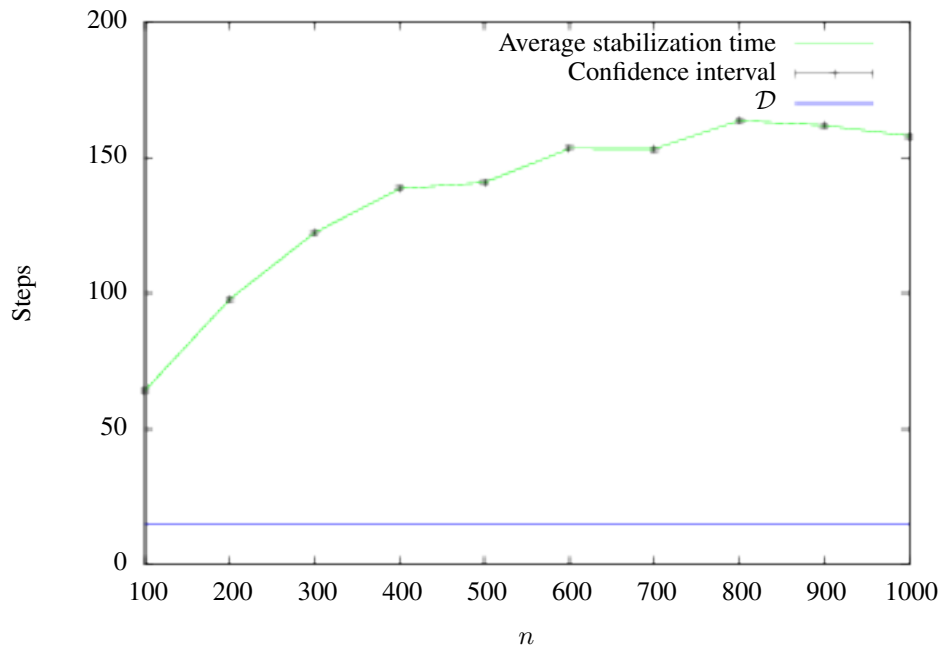


Figure 8: Average stabilization time in steps ($D = 15 \pm 1$).

References

- [Afek and Bremler-Barr, 1998] Yehuda Afek and Anat Bremler-Barr. Self-Stabilizing Unidirectional Network Algorithms by Power Supply. *Chicago J. Theor. Comput. Sci.*, 1998, 1998.
- [Albert and Barabási, 2001] Réka Albert and Albert-László Barabási. Statistical Mechanics of Complex Networks. *CoRR*, cond-mat/0106096, 2001.
- [Altisen *et al.*, 2014a] Karine Altisen, Alain Courmier, Stéphane Devismes, Anaïs Durand, and Franck Petit. Self-Stabilizing Leader Election in Polynomial Steps. In *SSS*, 2014.
- [Altisen *et al.*, 2014b] Karine Altisen, Alain Courmier, Stéphane Devismes, Anaïs Durand, and Franck Petit. Self-Stabilizing Leader Election in Polynomial Steps. Technical report, CNRS, 2014.
- [Arora and Gouda, 1994] Anish Arora and Mohamed G. Gouda. Distributed Reset. *IEEE Trans. Computers*, 43(9):1026–1038, 1994.
- [Awerbuch *et al.*, 1993] Baruch Awerbuch, Shay Kutten, Yishay Mansour, Boaz Patt-Shamir, and George Varghese. Time Optimal Self-stabilizing Synchronization. In *STOC*, pages 652–661, 1993.
- [Blin and Tixeuil, 2013] Lélia Blin and Sébastien Tixeuil. Brief Announcement: Deterministic Self-stabilizing Leader Election with $O(\log \log n)$ -bits. In *PODC*, pages 125–127, 2013.
- [Burman and Kutten, 2007] Janna Burman and Shay Kutten. Time Optimal Asynchronous Self-stabilizing Spanning Tree. In *DISC*, pages 92–107, 2007.
- [Chang, 1982] Ernest J. H. Chang. Echo Algorithms: Depth Parallel Operations on General Graphs. *IEEE Trans. Software Eng.*, 8(4):391–401, 1982.
- [Datta *et al.*, 2010] Ajoy Kumar Datta, Lawrence L. Larmore, and Hema Piniganti. Self-stabilizing Leader Election in Dynamic Networks. In *SSS*, pages 35–49, 2010.
- [Datta *et al.*, 2011a] Ajoy Kumar Datta, Lawrence L. Larmore, and Priyanka Vemula. An $O(n)$ -time Self-stabilizing Leader Election Algorithm. *J. Parallel Distrib. Comput.*, 71(11):1532–1544, 2011.
- [Datta *et al.*, 2011b] Ajoy Kumar Datta, Lawrence L. Larmore, and Priyanka Vemula. Self-stabilizing Leader Election in Optimal Space under an Arbitrary Scheduler. *Theor. Comput. Sci.*, 412(40):5541–5561, 2011.
- [Dijkstra, 1974] Edsger W. Dijkstra. Self-stabilizing Systems in Spite of Distributed Control. *Commun. ACM*, 17(11):643–644, 1974.
- [Dolev and Herman, 1997] Shlomi Dolev and Ted Herman. Superstabilizing Protocols for Dynamic Distributed Systems. *Chicago J. Theor. Comput. Sci.*, 1997, 1997.
- [Dolev *et al.*, 1999] Shlomi Dolev, Mohamed G. Gouda, and Marco Schneider. Memory Requirements for Silent Stabilization. *Acta Inf.*, 36(6):447–462, 1999.
- [Dolev, 2000] Shlomi Dolev. *Self-stabilization*. MIT Press, March 2000.
- [Huson and Sen, 1995] Mark L. Huson and Arunabha Sen. Broadcast Scheduling Algorithms for Radio Networks. In *Military Communications Conference, 1995. MIL-COM'95, Conference Record, IEEE*, volume 2, pages 647–651, 1995.
- [Kravchik and Kutten, 2013] Alex Kravchik and Shay Kutten. Time Optimal Synchronous Self Stabilizing Spanning Tree. In *DISC*, pages 91–105, 2013.

High-Level Simulation for Multiple Fault Injection Evaluation

Maxime Puys and Lionel Rivière and Thanh-Ha Le and Julien Bringer

SAFRAN Morpho

Firstname.Name@morpho.com

Magistère supervisor: Marie-Laure Potet

Verimag

University of Grenoble

Marie-Laure.Potet@imag.fr

This article is an extended version of one we submitted and got accepted to the QASA 2014 workshop affiliated to ES-ORICS 2014.

Abstract

Faults injection attacks have become a hot topic in the domain of smartcards. This work exposes a source code-base simulation approach designed to evaluate the robustness of high-level secured implementations against single and multiple fault injections. In addition to an unprotected CRT-RSA implementation, we successfully attacked two countermeasures with the high-level simulation under the *data* fault model. We define a filtering criteria that operates on found attacks and we refine our simulation analysis accordingly. We introduce a broader fault model that consists in skipping C lines of code and exhibit benefits of such high-level fault model in term of simulation performance and attack coverage.

1 Introduction

Effects of physical attacks on secure implementations were first described in 1997. In particular, fault injection attacks aim at modifying a program's state using an external event such as laser beams, voltage glitches or electromagnetic waves. Among the numerous possibilities opened by such attacks, an attacker can perform a Differential Fault Analysis (DFA) [Boneh *et al.*, 1997; Biham and Shamir, 1997] and retrieve secret information such as embedded cryptographic keys.

Smartcard-based products, which are widespread in the daily life, can be a profitable target for attackers. They may be sensitive to such fault attacks and therefore require a high security certification standard such as the Common Criteria [CSE *et al.*, 2012]. One of the very first step of a certification process is the security code review. In order to bring out vulnerabilities, evaluators and developers perform high-level security code reviews at the source code level. Suspected points

are then audited at the assembly level, but not systematically. However, mostly manual, this task is time consuming and error prone. A code reviewer could miss critical errors that might lead to a major security breach.

This two reasons encourage the development of automated high-level code analysis to help evaluators and developers. Complete and exhaustive approaches are often used at the binary level where all impacts of a fault can be considered (such as code operation modification) but this takes a long time to perform. In order to rapidly point out vulnerabilities in the security evaluation process, the high-level fault simulation becomes definitely useful. It constitutes a complementary step to other following low-level simulated or practical analyses.

Contributions. We propose an efficient high-level approach to analyze source codes against multiple fault injections. To achieve this end, we built tools to exhaustively explore a *data* fault model defined at the C variable level. Helped by an oracle, they can for each combination of faults tell if an attack worked on the program. Thanks to the testing approach we are able to produce detailed counterexamples or state on the program's robustness with respect to the chosen fault model. We demonstrate the validity of our approach on three implementations of the CRT-RSA [Quisquater and Couvreur, 1982] algorithm in the signature process by performing BellCoRe attacks [Boneh *et al.*, 1997]. Moreover, we propose two attack classification criteria aiming at regrouping them which becomes time-saving when dealing with multiple faults on realistic implementations. Finally, we study the results of the analysis with a *line-skip* fault model, lighter than the *data* fault model.

Organization of the paper. In section 2, we define useful terms widely used to express several concepts and security notions. We also recall the CRT-RSA algorithm and the BellCoRe attack. Section 3 further explains the considered fault model and the approach we use to evaluate implementations. Section 4 shows the results of our tests on the three CRT-RSA implementations. Finally, section 5 defines two classification criteria and discusses outcomes obtained from the three examples under the *line-skip* fault model.

2 State-of-the-art and definitions

2.1 Terminology

This section proposes succinct definitions of four notions: an attack, a fault, a security breach and a vulnerability. We explain how these notions are related and we will refer to these definitions all along this paper.

A **security breach** is the deviation of a program from its expected behavior in terms of security. A **vulnerability** names the presence of an error or lack of security in the program that might lead to a security breach. This term will be defined more precisely in section 5.1 using our classification criteria. A **fault** represents an external event changing the program's state. Its behavior and effects are formalized by a fault model using parameters such as space and temporal aspects, or persistence. An **attack** is the exploitation of a vulnerability by a fault. In [Potet *et al.*, 2014], authors refined this definition with the presence of a goal for the attacker. However, we will confine to the basic definition.

2.2 CRT-RSA

We choose to analyze the well known asymmetrical CRT-RSA algorithm [Quisquater and Couvreur, 1982]. Let's say that Alice wants to send the message m to Bob. She has to sign m using her RSA private key (d, N) and then she computes the signature $S = m^d \bmod N$. The idea behind the CRT algorithm is to replace the costly modular exponentiation of RSA with two sub-exponentiation with half the size of the original exponent. This roughly speeds up the computation by a factor of four.

Hence, the RSA private key d is split in two parts d_p and d_q . The inverse of q modulo p is denoted i_q . We obtain:

$$d_p = d \bmod (p - 1)$$

$$d_q = d \bmod (q - 1)$$

$$i_q = q^{-1} \bmod p$$

The two modular sub-exponentiation are realized as following :

$$S_p = m^{d_p} \bmod p$$

$$S_q = m^{d_q} \bmod q$$

$$S = S_q + q \cdot (i_q \cdot (S_p - S_q) \bmod p)$$

This last step *recombines* the two sub-signatures S_p and S_q in the final signature S . It can be performed using either Gauss or Garner's formula. The latter is the most used because as it provides better memory performances. This is the one we presented in the algorithm.

2.3 BellCoRe Attack on CRT-RSA

This attack has been discovered in 1997 by Boneh, DeMillo and Lipton [Boneh *et al.*, 1997] from BellCoRe (Bell Communications Research). An attacker is able to retrieve a prime factor p or q of N if he is able to inject a fault in the signature computation in order to obtain a faulty signature \widehat{S} such as:

$$1. |\widehat{S}| \neq |S|$$

$$2. \text{And } |\widehat{S} \bmod p| = |S \bmod p| \text{ or } |\widehat{S} \bmod q| = |S \bmod q|$$

The attacker is then able to retrieve either p or q by computing $\gcd(N, S - \widehat{S})$. In 1999, Joye, Lenstra and Quisquater [Joye *et al.*, 1999] showed that this attack can use only the faulty signature \widehat{S} and the message m and retrieve either p or q by computing $\gcd(N, m - \widehat{S}^e)$ with an overwhelming probability.

2.4 Code Security Properties

Security metrics aim at defining quantitative and objective criteria in order to gauge various aspect of security. It can be considered in multiple ways [Miani *et al.*, 2013] and takes several form [Vaughn *et al.*, 2003; Savola, 2007]. In the context of smartcards implementation robustness testing against fault injection and considering a manual security code review, they are designed to facilitate decision making and improve the code robustness. Measuring how the targeted implementation deviates from its functional specification under fault injection constitutes the *correctness* aspect of security. It is the assurance that the targeted function carries out its task precisely to the specification with the expected behavior. Another metric to assess the *efficiency* of a simulation tool is needed in order to confront specific tools.

Measurement requires realistic assumptions and inputs to attain reliable results [Jansen, 2010]. The qualitative and quantitative properties of a security objective must be defined. In our case, the security objective is to preserve the correctness of an ongoing RSA ciphering or signature under fault injections to avoid BellCoRe attacks. The quantitative aspect of such an objective lies in the number of deviation from the expected behavior. According to Section 2.1, it corresponds to the number of security breaches. The qualitative aspect corresponds to the nature of the attack (the fault model), and the way it deviates from the reference.

The classification of deviant cases permit to define the criticality level of found attacks. Thereby, we can measure the code sensitivity or robustness of a targeted code under fault attack, and determine potential vulnerabilities according to a measurement system, a fault model and a simulation tool.

2.5 Existing Works

In [Christofi, 2013; Christofi *et al.*, 2013], Christofi et al. propose a formal method to validate cryptographic implementations against first order fault injections relying on the orem proving. The fault targets a C variable and sets it to zero. Their studies lead them to the implementation of a Framac plugin named TL-Face and using the Jessie plugin in order to solve *weakest precondition* problems. A case study has been made on the Vigilant implementation of CRT-RSA, revealing possible BellCoRe attacks.

In [Rauzy and Guilley, 2013], Rauzy et al. study the effects of a first order fault on several CRT-RSA implementations. Their fault model consists in replacing any intermediate value with either zero or a random value. No mathematical properties such as co-primality or equivalent modulo is

considered. Their analyses lead to the implementation of an OCaml tool testing exhaustively every possible faults. It has been used to compare an unprotected implementation of CRT-RSA with the Shamir [Shamir, 1999] one and the Aumüller one [Aumüller *et al.*, 2002]. In [Rauzy and Guilley, 2014], they extend their approach on Vigilant and Coron’s counter-measures and provide high-order attacks.

In [Kauffmann-Tourkestansky, 2012], Kauffmann-Tourkestansky works on a first order fault model targeting control flow in order to skip instructions (using NOP or JUMP instructions). He uses a mutation analysis of C source codes and try to fill the gap between high-level and low-level implementations.

In [Heydemann *et al.*, 2013], Heydemann *et al.* also focus on an instruction skip based fault model. They propose a set of counter-measures applicable to every instruction of the Thumbs2 instruction set of the ARM language [arm, 2005]. They suppose that it is hard for an attacker to reproduce twice the same fault in a few cycles delay and give a way to duplicate each instruction. Finally, they prove that this mutated program (with all its instruction duplicated) has the same behavior than the original using the Vis model-checking tool [Brayton *et al.*, 1996].

In [Berthier *et al.*, 2014], Berthier *et al.* propose a brand new approach for evaluating smartcards security against first order fault injection. It consists in embedding the fault simulator itself directly on the smartcard. This way, faults are tested on the final product which is more reliable than on a software model. Moreover, it also enable the possibility to study the behavior of the card after injections using side-channel analysis. Their fault model targets byte skipping of an arbitrary length. They put it into practice on an implementation of a DES cipher, revealing for instance a fault able to skip a function call which compromised the security of the implementation.

In [Potet *et al.*, 2014], Potet *et al.* study the effects of a test inversion based fault model. The analysis is objective guided, in term of reaching or not basic blocks. The fault model is exhaustively explored by a mutation approach. Moreover, in order to take into account higher order faults, that would cause path explosion (and in their case, mutants number explosion), they only create one higher level mutant. It embeds on its own the *possibility* of injecting each possible fault or not. The paths are then covered by the concolic execution tool Klee [Cadar *et al.*, 2008].

Those existing works emphasize the importance of considering both *data* and control flow fault models in secure implementation robustness evaluation. With our high-level fault simulation approach presented in the next section, we consider both models while ensuring a very efficient detection and high performances.

3 High-Level Simulation Approach

Section 3.1 defines two reachable fault models considered with fault simulator, namely the *data* and the *line-skip* fault models. Section 3.2 defines the testing protocol used to evaluate implementations.

3.1 Mechanisms

The fault simulator operates at the source code level, considered fault models are defined with this granularity accordingly.

- Granularity:* : C variable
- (Spatial | temporal) control* : Complete | Limite
- Persistence:* : Permanent or transient
- Multiplicity:* : First order or higher
- Type:* : Set to 0 or 1

FAULT MODEL 1: Data

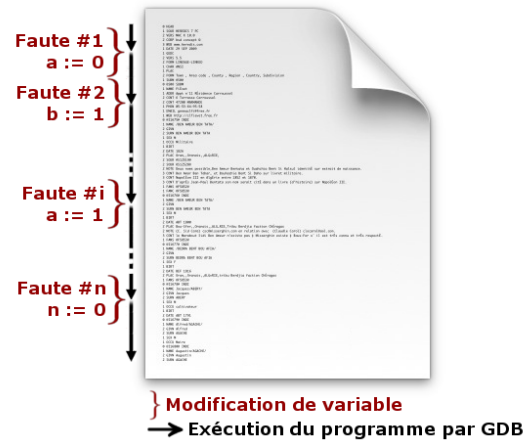


Figure 1: Data fault model

As showed in figure 1, the black arrows represent the program’s execution by GDB (launched by the `run` command). Using a `breakpoint`, it is interrupted each time a variable’s value has to be modified. The value modification (represented by a red bracket) is performed using the `set var` command followed by the new value. Finally, execution is resumed using the `continue` command.

This model is a subset of the one proposed in [Vigilant, 2008; Rauzy and Guilley, 2013]. Both of them allow the attacker to perform permanent and transient faults on every variable and intermediate values. On the opposite, our model only allow permanent faults on variable and transient faults on intermediate values, which is more realistic. It also assumes that an attacker can not modify the secret key, the message to sign or the signature, which should pass integrity checks at any time.

The *instruction-skip* fault model is explored in previous works [Berthier *et al.*, 2014; Moro *et al.*, 2013; V. K. Kosuri and N. Fazal, 2013], but almost exclusively at the assembly code level. To our knowledge, no experiment targets CRT-RSA under such fault model in the literature. As there is no assembly instruction notion at the C code level, we propose a high-level extension of the *instruction-skip* fault model as follows:

<i>Granularity</i>	:	C code line
<i>Skip Width</i>	:	One C code line
<i>(Spatial temporal) control</i>	:	Complete Limite
<i>Persistence</i>	:	Transient
<i>Multiplicity</i>	:	First order or higher
<i>Type</i>	:	Skip of lines in C source code

FAULT MODEL 2: Line-skip

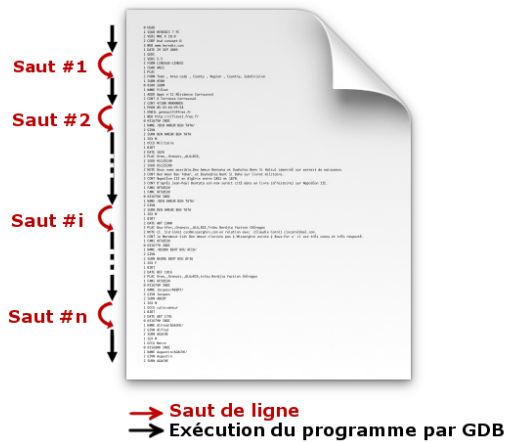


Figure 2: Line-skip fault model

As described in figure 2, the black arrows represent the program’s execution by GDB (launched by the `run` command). Using a `breakpoint`, it is interrupted each time a line-skip is requested. The skip is performed using the `jump` command (represented by a red arrow), allowing direct or indirect jumps to a line of the source code. Finally, execution is resumed using the `continue` command.

In practice, the fault simulator has been designed to permit arbitrary width line skips. However, a C line of code is often represented by several lines of assembly. Knowing that in the current state of the art, it remains difficult to skip multiple assembly lines, an attacker will unlikely be able to skip multiple C lines of code. Then, we will only consider faults with a width of one line.

Table 1 summarizes the differences of our fault models with state of the art, on several general criteria in order to show the diversity of existing analysis on this topic.

The simulator mostly relies on a testing approach in the sense that the targeted implementation is not modified between simulations. However, a single mutation might be needed to decompose computations and let the intermediate values appear as one time variables.

Faults are injected using the well known Gnu Project Debugger (GDB) [GDB, 1988] that makes it possible to pause the execution via breakpoints, change any variable’s value

and then resume the execution. Moreover, we enhanced the control and efficiency of our simulator by providing automation through Python scripts which allow way more possibilities than GDB scripts as they only have few conditional structures available.

3.2 Test Protocol

Our testing protocol currently targets any single function in an implementation. Several parameters can be tuned to specify the fault model such as the fault multiplicity or which variable to attack or not. If not specified, every global variable of the file, local variable and parameter of the function will be faulted at each line of the function.

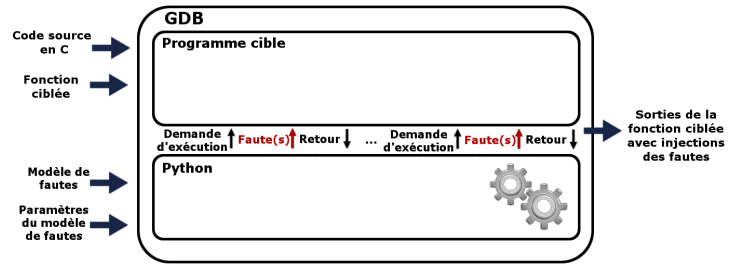


Figure 3: Inner-workings of the simulator

As described in figure 3, GDB commands are controlled by Python scripts which will for each combination of faults (aka targeted variables, injection lines and new values set), ask GDB to:

1. Execute the target;
2. Set breakpoints where the fault shall be injected;
3. Inject the faults;
4. Get the system state post execution;
5. Repeat this sequence until the fault model is exhaustively explored.

All functional outputs (such as function return value or prints) are logged in a single file split by executions. Moreover, as these outputs might be given to an oracle telling if they correspond to an attack or not, the log file also keeps track of such verdicts automatizing request to oracles. This will spot possible attacks with respect to an oracle defined by the user. As we will see, there might be many of them.

We choose to allow the simulator to change the value of a variable even if this variable is not used at the targeted line. It means that every variable will be forced to every possible values (zero or one) at each line. This possibility could be realistic for example in a system where the values of the variables are stored in memory and loaded each time they are read. In a system where this assertion does not stand (basically all system with registers), only the realistic attacks will be included in the total set of attacks found.

Instinctively, such a flexibility will create a relation between some attacks in a way that they can be regrouped as one generic attack and several ways to reproduce it. Thus, in

Reference	Abstraction level	Type of fault model	Persistence	High order
[Christofi, 2013] [Christofi <i>et al.</i> , 2013]	High level (C)	Data-flow	Permanent	
[Rauzy and Guilley, 2013] [Rauzy and Guilley, 2014]	High level (OCaml)	Data-flow	Transient	✓
[Kauffmann-Tourkestansky, 2012]	High level (C) / Low level	Control-flow	Transient	
[Heydemann <i>et al.</i> , 2013] [Moro <i>et al.</i> , 2013]	Low level	Control-flow	Transient	
[Berthier <i>et al.</i> , 2014]	Low level	Control-flow	Transient	
[Potet <i>et al.</i> , 2014]	Intermediate (LLVM)	Control-flow	Transient	✓
Our approach	High level (C)	Data-flow Control-flow	Permanent Transient	✓

Table 1: Comparison of our approach with state-of-the-art

section 5.1, we will define precisely what we call redundant attacks. Then we will detail classification criteria in order to regroup such attacks.

4 Case Study

In this section we present a case study in order to show the validity of our tool. We concentrate on the process of signature using CRT-RSA. The objective is to ask a system to sign a random message m with its own key (p, q, d_p, d_q, i_q, N) and to obtain a prime factor p or q of N using a BellCoRe attack.

We will study the results of the simulator on an unprotected CRT-RSA implementation and on one using the Aumüller *et al.* counter-measure. Both of them will be tested with the *data* fault model and the *line-skip* fault model explained in Section 3.1.

4.1 Study of an Unprotected Implementation of CRT-RSA

```

1 int CRT_RsaSign(int M, int p, int q, int d_p,
  int d_q, int i_q)
2   S_p = M^{d_p} mod p /* Signature modulo p */
3   S_q = M^{d_q} mod q /* Signature modulo q */
4
5   /* Recombining */
6   S = S_q + q · (i_q · (S_p - S_q) mod p)
7   return S

```

Listing 1: Unprotected implementation of CRT-RSA

SIMULATION 1 targets the unprotected CRT-RSA given in Listing 1 with first order attacks under the *data* fault model. For the rest of this paper, we will describe each simulation experiment with the following structure:

Target function : Unprotected CRT_RsaSign
Success oracle : Success of a BellCoRe attack on the signature
Fault model : *Data*
Fault multiplicity : 1 (first order)
Result : **11 attacks found**

SIMULATION 1: Data model on unprotected CRT-RSA (first order faults)

Data attack example 1 The unprotected implementation of CRT-RSA is prone to numerous attacks. For instance, forcing the value of S_p to zero prior to the execution of line 6 reveals the prime factor q of N . Indeed,

$$S - \widehat{S} = q \cdot ((i_q \cdot (S_p - S_q) \bmod p) - (i_q \cdot (-S_q) \bmod p))$$

and

$$\gcd(N, S - \widehat{S}) = q$$

Data attack example 2 An even clearer attack consists in zeroing the whole intermediate value $q \cdot (i_q \cdot (S_p - S_q) \bmod p)$ prior to the execution of line 6 will result in $\widehat{S} = S_q$, thus we have a BellCoRe attack:

$$|\widehat{S}| \neq |S| \text{ and } |\widehat{S} \bmod q| = |S \bmod q|$$

SIMULATION 2 targets the same unprotected CRT-RSA with first order attacks according to the *line-skip* fault model. Results are shown below:

Line attack example 1 Skipping the line computing $i_q \cdot (S_p - S_q) \bmod p$ (intermediate computation included in line 6 of listing 1) led to $\widehat{S} = S_q + q \cdot (S_p - S_q)$ (as the former intermediate value of the signature was $(S_p - S_q)$ and we skipped its multiplication to i_q) which allows a BellCoRe attack.

The *line-skip* fault model detected four vulnerable lines. Listing 1 shows a generic code of the naive RSA where several computations are gathered on few lines. There is a strong

Target function : Unprotected CRT_RsaSign
Success oracle : Success of a BellCoRe attack on the signature
Fault model : Line-skip
Fault multiplicity : 1 (first order)
Result : **4 attacks found**

SIMULATION 2: Line-skip model on unprotected CRT-RSA (first order faults)

dependency between the implementation and the attack success rate with the line skip fault model. The latter can also recover several attacks found by the *data* fault model and count them as a single one, which explains the lower number of found attacks in SIMULATION 2.

For instance, if we consider a variable a that is used in an attacked exponentiation, with the *data* fault model, we can set it either to 0 or 1. However, with the *line-skip* fault model, the attack output will depend on the initialization value of a . Therefore, if a was initialized to 0, we would recover a *set-to-zero* data fault model. Moreover, if a was not initialized, we would recover a *random* data fault model and finally, if a was initialized to a constant value, we would recover a *set-to-value* data fault model. Even if the two latter *data* fault models are not directly considered by our simulation, the *line-skip* fault model can detect them.

4.2 Study of the Shamir Implementation of CRT-RSA

The counter-measure of Shamir [Shamir, 1999] introduces a new factor r co-primed with p and q , random and small (less than 64 bits). Computations are thus performed modulo $p \cdot r$ (resp. modulo $q \cdot r$), which allows to retrieve the result by reducing modulo p (resp. modulo q). A verification is possible by reducing modulo r . The Shamir implementation of CRT-RSA is given in listing 2.

```

1 int CRT_RsaSign(int M, int p, int q, int d,
2   int iq)
3   r = rand()
4   p' = p * r
5   dp = d mod (p - 1) * (r - 1)
6   Sp' = M^dp mod p' /* Signature modulo p' */
7
8   q' = q * r
9   dq = d mod (q - 1) * (r - 1)
10  Sq' = M^dq mod q' /* Signature modulo q' */
11
12 /* Recombining */
13  Sp = Sp' mod p
14  Sq = Sq' mod q
15  S = Sq + q * (iq * (Sp - Sq) mod p)
16
17  if (Sp' != Sq' mod r) { takeCounterMeasure() }
18  else { return S }

```

Listing 2: Shamir implementation of CRT-RSA

Target function : Shamir CRT_RsaSign
Success oracle : Success of a BellCoRe attack on the signature
Fault multiplicity : 1 (first order)
Result : **15 attacks found**

SIMULATION 3: Data model on Shamir CRT-RSA (first order faults)

SIMULATION 3 targets the Shamir implementation of CRT-RSA with the *data* fault model. Our simulator shows that a first order fault is enough to break the implementation.

Data attack example 3, A *Set-to-zero* on the value of S_p prior to execution of line 15 of listing 2 allows us to obtain the exact same attack than Data attack example 1, as the integrity test only relies on S'_p and S'_q .

Data attack example 4 The same clear attack presented on Data attack example 2 is still feasible. Setting the whole intermediate value $q \cdot (i_q \cdot (S_p - S_q) \bmod p)$ to zero prior to the execution of line 15 will trigger a BellCoRe attack the exact same way.

SIMULATION 4 targets the same Shamir implementation of CRT-RSA under the *line - skip* fault model. Here again, our simulator shows that a first order fault is enough to break the implementation.

Line attack example 2 Skipping the line computing $i_q \cdot (S_p - S_q) \bmod p$ (intermediate computation included in line 15 of listing 2) led to $\widehat{S} = S_q + q \cdot (S_p - S_q)$ which allows a BellCoRe attack as showed in Line attack example 1. As explained in Data attack example 3, Shamir's counter-measure is not triggered due to the fact that the integrity test only relies on S'_p and S'_q and not on S .

4.3 Study of the Aumüller Implementation of CRT-RSA

The counter-measure of Aumüller [Aumüller *et al.*, 2002] has been developed in order to enhance the version of Shamir against first order attacks. It stills introduce a new factor t co-primed with p and q . However, the computation of d_p et d_q is performed outside of the function which removes the use of d . Moreover, the ending verification introduced by Shamir is now *asymmetrical* and intermediate verification are also added. The Aumüller implementation of CRT-RSA is given in listing 3.

On the Aumüller implementation of CRT-RSA, our results matches the one exposed in [Rauzy and Guilley, 2013]. No first order attack is found by setting any variable or intermediate data to zero.

```

1 int CRT_RsaSign(int M, int p, int q, int dp,
    int dq, int iq)
2     t = rand()
3
4     p' = p · t
5     d'p = dp + random1 · (p - 1)
6     S'p = Md'p mod p' /* Signature modulo p' */
7
8     if ((p' mod p ≠ 0) or (d'p ≠ dp mod (p - 1))) {
        takeCounterMeasure() }
9
10    q' = q · t
11    d'q = dq + random2 · (q - 1)
12    S'q = Md'q mod q' /* Signature modulo q' */
13
14    if ((q' mod q ≠ 0) or (d'q ≠ dq mod (q - 1))) {
        takeCounterMeasure() }
15
16    /* Recombining */
17    Sp = S'p mod p
18    Sq = S'q mod q
19    S = Sq + q · (iq · (Sp - Sq) mod p)
20
21    if ((S - S'p ≠ 0 mod p) or (S - S'q ≠ 0 mod q)) {
        takeCounterMeasure() }
22
23    Spt = S'p mod t
24    Sqt = S'q mod t
25    dpt = d'p mod (t - 1)
26    dqt = d'q mod (t - 1)
27
28    if (Sptdqt ≠ Sqtdpt mod t) { takeCounterMeasure() }
29    else { return S }

```

Listing 3: Aumüller implementation of CRT-RSA

SIMULATION 5 targets the CRT-RSA implementation protected with the Aumüller countermeasure shown in Listing 3 above. Second order attacks are performed according to the *data* fault model, we obtain:

<i>Target function</i>	: Aumüller CRT_RsaSign
<i>Success oracle</i>	: Success of a BellCoRe attack on the signature
<i>Fault model</i>	: <i>Data</i>
<i>Fault multiplicity</i>	: 2 (second order)
<i>Result</i>	: 802 attacks found

SIMULATION 5: Data model on Aumüller CRT-RSA (second order faults)

The Aumüller implementation is not robust against second order attacks using this *data* fault model. The first fault is used to corrupt the computation while the second avoids the counter-measure to be triggered.

Data attack example 5 A *Set-to-one* fault on $(p - 1)$ before the execution of line 5 sets up a BellCoRe attack. Secondly,

performing the same fault on $(p - 1)$ before the execution of line 8 avoids triggering the counter-measure.

Data attack example 6 The attack presented for unprotected and Shamir implementations consisting in setting the whole intermediate value $q \cdot (i_q \cdot (S_p - S_q) \bmod p)$ to zero before the execution of line 19 stills enables a BellCoRe attack. However, it will also trigger the counter-measure of line 21 (through the test $S - S'_p \not\equiv 0 \pmod{p}$). A second fault will disable it by setting the intermediate value $S - S_p$ to zero.

Such a huge number of attacks (802 in SIMULATION 3) makes results impossible to analyze by hand. Moreover it is obvious that most of these attacks found can be regrouped into some generic attacks with different ways to reproduce them. This example definitely shows the necessity of classification criteria and metrics.

SIMULATION 6 targets the same protected CRT-RSA with the Aumüller countermeasure but second order attacks are performed according to the *line-skip* fault model. Results are shown below:

<i>Target function</i>	: Aumüller CRT_RsaSign
<i>Success oracle</i>	: Success of a BellCoRe attack on the signature
<i>Fault model</i>	: <i>Line-skip</i>
<i>Fault multiplicity</i>	: 2 (second order)
<i>Result</i>	: 13 attacks found

SIMULATION 6: Line-skip model on Aumüller CRT-RSA (second order faults)

Line attack example 3 Skipping the line computing $i_q \cdot (S_p - S_q) \bmod p$ (intermediate computation included in line 19 of listing 3) led to $\hat{S} = S_q + q \cdot (S_p - S_q)$ which allows a BellCoRe attack as showed in Line attack example 1 and 2. This time, it also triggers the counter-measure on line 21 which can be easily skipped by our fault model at the second order. This attack is very similar to the Data attack example 6.

Thirteen attacks are spotted by the *line-skip* fault model. Interestingly, the number of attacks found by the *line-skip* fault model in Simulation 4 is drastically lower to the one found by the *data* fault model in simulation 3. This can be explained knowing that it only depends on the lines while the *data* fault model also depends on variables and values. We will present deeper analyses of the link between these two models in section 5.2.

To the best of our knowledge, some studies showed that the Aumüller implementation of CRT-RSA is weak against multiple fault injection such as [Kim *et al.*, 2011] but none provides detailed experimental results.

5 Advanced Analysis

5.1 Regrouping Criteria

We recall that a successful attack is the exploitation of a code vulnerability, which is induced by the fault injection. According to the transient value modification fault model, we

provide a variable-centric criteria C_{val} by which we measure the code sensitivity. C_{val} only depends on the value taken by the targeted variable regardless the attacked line of code. It is defined as:

$$C_{val}(targeted_var, value) := \#\{line \mid \mathcal{O}(S, \widehat{S}, n) = true\}$$

For a given couple $(targeted_variable, value)$, C_{val} describes the number of successful attacks obtained by setting $targeted_variable$ to $value$ regardless of the lines. We define as non-redundant, the successful attack that has the greatest injection line number.

Table 2 summarizes the experimental results obtained by the fault simulator on CRT-RSA implementations with the *data* fault model on C variables. The first line shows how many attacks have been found by the simulator on each implementation. The second line displays how many non-redundant attacks are found. When the same targeted variable is modified to the same value at several different lines of the code, it describes the same attack. We call such groups of lines a vulnerability. This is why the second line of Table 2 report a lower number of found attack.

5.2 Link Between The Fault Models

We now provide a line-centric criteria C_{line} that for each line of the implementation under testing returns a boolean value depending on the presence of an attack with an injection on this line. In the end, for a given implementation, this criteria returns the set of lines where injecting a fault results in an attack. It is defined as follows:

$$C_{line}(l, m) := \begin{cases} true & \text{if } \exists \mathcal{A}(l, m) \mid \mathcal{O}(S, \widehat{S}, N) = true \\ false & \text{otherwise} \end{cases}$$

For a given line l , C_{line} returns true if there exist a successful attack \mathcal{A} on line l according to the fault model m denoted as $\mathcal{A}(l, m)$. For each implementation that we studied, we retrieve vulnerable lines filtered by C_{line} under the *data* fault model. Then, we check how many of these lines are also vulnerable under the *line-skip* fault model. For readability purpose, we define

$$Vulnerable\ lines_{model} := \{l \mid C_{line}(l, model) = true\}$$

Thereby, we compute the following ratio:

$$Gain := \frac{\#\{Vulnerable\ lines_{data} - Vulnerable\ lines_{lskip}\}}{\#\{Vulnerable\ lines_{lskip}\}}$$

It happened that for each of our implementations, this gain ratio was 100% even considering redundant attacks. This means that each vulnerable line found with the *data* fault model is also vulnerable with the *line-skip* fault model.

Moreover, as we can see in table 3, the first (resp. second) line displays the time taken by our tool for each implementation using the *data* (resp. *line-skip*) fault model. The third line is the ratio of the first and second lines. We can here

conclude that simulations using the *line-skip* fault model are much quicker than using the *data* fault model. This is specially true when dealing with higher order faults.

In our experiments, every attack found by the *data* fault model seems to be reproducible using the *line-skip* fault model. As mentioned in section 4.1, the general link between the two fault models is not straightforward. On one hand, the general effects of a line skip on a variable is a *Set-to-last-value*. It differs from *Set-to-0/1*. If we consider that an attacker has no control on intermediate values, the *line-skip* fault model can be seen as a *uncontrolled* value fault model.

On the other hand, considering a single line of code, the *data* fault model targets only one variable while the *line-skip* fault model targets every modified variables of the line. Despite these differences, it still becomes really useful to run *line-skip* based simulations prior to data based simulation at high-level as it produces similar effects. Moreover, the large simulation performance improvement justifies this choice.

Finally, high-level *line-skip* allows to discover attacks pretty invulnerable to *data* fault models such as removing a `break` statement in a `switch` or adding/removing loop iterations. As a matter of fact, the last has been put into practice in [Dehbaoui *et al.*, 2013] where the authors are able to increase the leakage of an AES key for side channels analysis. Generally, these high-level faults models can be considered as the consequence of low-level faults encompassing several of them.

6 Conclusion

We proposed an approach to evaluate the robustness of secured implementations against multiple fault injections. This approach works on high-level source codes (such as C). We proposed a first fault model relying on data modification with the granularity of a C variable. This fault model has been automated in a Python tool which is able to try it exhaustively on a given implementation and log out the functions outputs. Helped by oracles, it can tell whether a combination of faults results in an attack or not.

We demonstrated the validity of our tool on three examples of CRT-RSA implementations and obtained results according to the current state-of-the-art. Moreover, we proposed a second fault model relying on line skipping that is faster. In our experiments, we found that it covers entirely the attacks found by the *data* fault model with a huge speed gain.

Finally, we proposed a set of criteria and metrics in order to regroup attacks found and quantify in term of security the robustness of an implementation.

6.1 Perspectives

Further work is to refine the link we found between our *data* and control-flow fault models. This shall start by analysing more example such as more complexe counter-measures (for instance [Vigilant, 2008] and its enhancement [Coron *et al.*, 2010] or [Kim *et al.*, 2011]). We also already studied non-cryptographic example such as *VerifyPIN* implementations, and we would like to continue to expand in this way.

	Unprotected (order 1)	Shamir (order 1)	Aumüller (order 1)	Aumüller (order 2)
Attacks found	11	15	0	802
Non-redundant attack	9	11	0	85

Table 2: Found Attacks with the Fault Simulation with Data Fault Model

	Unprotected (order 1)	Shamir (order 1)	Aumüller (order 1)	Aumüller (order 2)
Data	3,53 s	38,75 s	143,93 s	1361,45 mn
Line Skip	2,18 s	2,39 s	14,31 s	7,41 mn
Gain ratio	162 %	1621 %	1006 %	18373 %

Table 3: Line-skip Fault Model And Fault Simulation Timing Performances

Then a deeper and more generic analysis of our two fault models could formally establish a link between the two of them and definitely promote the use of *line – skip* models to increase simulations speed.

References

- [arm, 2005] ARM Architecture Reference Manual - Thumb-2 Supplement, 2005.
- [Aumüller *et al.*, 2002] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert. Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 260–275. Springer, 2002.
- [Berthier *et al.*, 2014] M. Berthier, J. Bringer, H. Chabanne, T.-H. Le, L. Rivière, and V. Servant. Idea: Embedded Fault Injection Simulator on Smartcard. In J. Jürjens, F. Piessens, and N. Bielova, editors, *ESSoS*, volume 8364 of *LNCS*, pages 222–229. Springer, 2014.
- [Biham and Shamir, 1997] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.
- [Boneh *et al.*, 1997] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). In W. Fumy, editor, *EUROCRYPT*, volume 1233 of *LNCS*, pages 37–51. Springer, 1997.
- [Brayton *et al.*, 1996] R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S.-T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. Ranjan, S. Sarwary, T. R. Staple, G. Swamy, and T. Villa. VIS: A system for verification and synthesis. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 428–432. Springer Berlin Heidelberg, 1996.
- [Cadar *et al.*, 2008] C. Cadar, D. Dunbar, and D. R. Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *OSDI*, pages 209–224, 2008.
- [Christofi *et al.*, 2013] M. Christofi, B. Chetali, L. Goubin, and D. Vigilant. Formal verification of a CRT-RSA implementation against fault attacks. *J. Cryptographic Engineering*, 3(3):157–167, 2013.
- [Christofi, 2013] M. Christofi. *Preuves de sécurité outillées d’implémentation cryptographiques*. PhD thesis, Laboratoire PRiSM, Université de Versailles Saint Quentin-en-Yvelines, France, 2013.
- [Coron *et al.*, 2010] Jean-Sebastien Coron, Christophe Giraud, Nicolas Morin, Gilles Piret, and David Vigilant. Fault attacks and countermeasures on vigilant’s rsa-crt algorithm. In *Proceedings of the 2010 Workshop on Fault Diagnosis and Tolerance in Cryptography*, FDTC ’10, pages 89–96, Washington, DC, USA, 2010. IEEE Computer Society.
- [CSE *et al.*, 2012] CSE, SCSSI, BSI, NLNCSA, CESG, NIST, and NSA. Common Criteria for Information Technology Security Evaluation Version 3.1 Release 4, 2012. <https://www.commoncriteriaportal.org>.
- [Dehbaoui *et al.*, 2013] A. Dehbaoui, A.-P. Mirbaha, N. Moro, J.-M. Dutertre, and A. Tria. Electromagnetic Glitch on the AES Round Counter. In E. Prouff, editor, *COSADE*, volume 7864 of *LNCS*, pages 17–31. Springer, 2013.
- [GDB, 1988] 1988. <http://www.sourceware.org/gdb/>.
- [Heydemann *et al.*, 2013] K. Heydemann, N. Moro, E. Encrenaz, and B. Robisson. Formal verification of a software countermeasure against instruction skip attacks. In *PROOFS 2013*, Santa-Barbara, États-Unis, Aot 2013.
- [Jansen, 2010] W. Jansen. *Directions in security metrics research*. DIANE Publishing, 2010. NISTIR 7564.
- [Joye *et al.*, 1999] M. Joye, A. K. Lenstra, and J.-J. Quisquater. Chinese Remaindering Based Cryptosystems

- in the Presence of Faults. *J. Cryptology*, 12(4):241–245, 1999.
- [Kauffmann-Tourkestansky, 2012] X. Kauffmann-Tourkestansky. *Analyses securitaires de code de carte a puce sous attaques physiques simulees*. PhD thesis, Université d’Orléans, 2012.
- [Kim *et al.*, 2011] S.-K. Kim, T. H. Kim, D.-G. Han, and S. Hong. An Efficient CRT-RSA Algorithm Secure Against Power and Fault Attacks. *J. Syst. Softw.*, 84(10):1660–1669, Octobre 2011.
- [Miani *et al.*, 2013] R.-S. Miani, M. Cukier, B. B. Zarpelão, and L. de Souza Mendes. Relationships Between Information Security Metrics: An Empirical Study. In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop, CSIIRW ’13*, pages 22:1–22:4, New York, NY, USA, 2013. ACM.
- [Moro *et al.*, 2013] N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, and E. Encrenaz. Electromagnetic Fault Injection: Towards a Fault Model on a 32-bit Microcontroller. In *FDTC*, pages 77–88. IEEE, 2013.
- [Potet *et al.*, 2014] M.-L. Potet, L. Mounier, M. Puys, and L. Dureuil. Lazart: a symbolic approach for evaluation the robustness of secured codes against control flow fault injection. In *ICST*, 2014.
- [Quisquater and Couvreur, 1982] J.-J. Quisquater and C. Couvreur. Fast decipherment algorithm for rsa public-key cryptosystem. *Electronics Letters*, 18(21):905–907, 1982.
- [Rauzy and Guilley, 2013] P. Rauzy and S. Guilley. A Formal Proof of Countermeasures against Fault Injection Attacks on CRT-RSA. volume 2013, page 506, 2013.
- [Rauzy and Guilley, 2014] P. Rauzy and S. Guilley. Formal Analysis of CRT-RSA Vigilant’s Countermeasure Against the BellCoRe Attack: A Pledge for Formal Methods in the Field of Implementation Security. In S. Jagannathan and P. Sewell, editors, *PPREW@POPL*, page 2. ACM, 2014.
- [Savola, 2007] R. Savola. Towards a taxonomy for information security metrics. In G. Karjoth and K. Stølen, editors, *QoP*, pages 28–30. ACM, 2007.
- [Shamir, 1999] A. Shamir. Method and apparatus for protecting public key schemes from timing and fault attacks. Patent Number 5,991,415, Novembre 1999. also presented at the rump session of EUROCRYPT ’97.
- [V. K. Kosuri and N. Fazal, 2013] V. K. Kosuri and N. Fazal. FPGA Modeling of Fault-Injection Attacks on Cryptographic Devices. In *IJERA*, volume 3, pages 937–943, 2013.
- [Vaughn *et al.*, 2003] R. B. Vaughn, R. R. Henning, and A. Siraj. Information Assurance Measures and Metrics - State of Practice and Proposed Taxonomy. In *HICSS*, page 331, 2003.
- [Vigilant, 2008] D. Vigilant. Rsa with crt: A new cost-effective solution to thwart fault attacks. In *Proceeding Sof the 10th International Workshop on Cryptographic Hardware and Embedded Systems, CHES ’08*, pages 130–145, Berlin, Heidelberg, 2008. Springer-Verlag.

Human detection in indoor environment with 3D sensor

Carole Plasson
AMA, LIG
Grenoble, France

Supervised by: Olivier Aycard

I understand what plagiarism entails and I declare that this report is my own, original work.
Plasson Carole, 21 august 2014

Abstract

Human detection is an open and complex problem in computer vision. Indeed, creating an accurate 3D detection in real time is a research area which remains under study. In industry, it can be used when robots work side-by-side with human workers. In this case, detection has to be accurate for security.

This internship deals with human detection in indoor environment. It is performed with RGB-D data provided by 3D sensor. RGB-D datasets are composed of color and depth images. The purpose is to obtain an accurate 3D detection in real time.

To achieve this, we first at all perform a motion detection. It allows us to obtain 2D bounding boxes. These are sent to a classifier to determine whether the boxes contain a human. After that, we obtain a 2D human detection. With this method, it is possible to detect several people at the same time.

Once 2D detection performed, we work with depth data to compute the orientation of a human. We obtain an angle which allows us to realize a 3D box around human. For a real time detection, we add parallelism.

1 Introduction

The human detection is an essential part in computer vision and is used, for example, in video-surveillance. The main problem of this detection is the variation of human features (physical features but also different postures they may have). In some cases, for example in industry where humans work side by side with robots, there is a new problem : detection has to be efficient, in the interests of security but also efficiency. Indeed, a robot has to be able to detect a human. With this detection, it will avoid him and will not stop unnecessarily if the robot detects a human enough away [Knight, 2013].

During this internship, we are interested in the detection in constrained environment (indoor environment) and with a fixed camera. This camera is a 3D laser sensor which allows us to get RGB (Red Green and Blue) and depth data. (see figure 1)

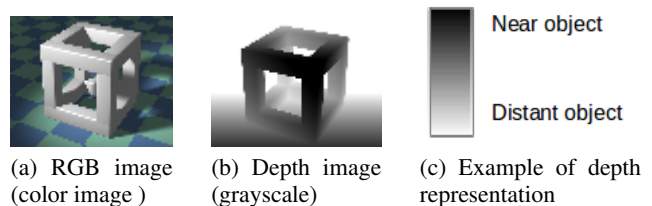


Figure 1: Data sent by sensor

For this, we first use color images to create a 2D box around human. The figure 2 shows the steps to realize this box. These images will be analyzed to detect moving regions that we represent by white pixels (images in black and white).

Then, we use a classifier which allows us to recognize a human in an image thanks to different features. With this classifier, we know if the bounding box contain an human or not (image with red rectangle).

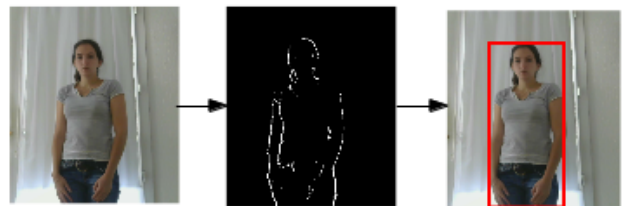


Figure 2: Steps to perform a bounding box

Once the 2D box obtained, we perform this one in 3D. To achieve this, we compute the orientation angle of human in using depth images. With this angle, we can create a 3D bounding box. Sometimes, this 3D representation is not sufficient, too large and imprecise. We can adjust and cut this one in several boxes to resolve this problem.

The construction of this 3D box requests a lot of computations. For a detection in real time, our computations in the program have to be optimal. For best performances,

parallelism has been added.

This report is structured as follows. Section 2 presents the general scheme for performing a human detection. Then, the section 3 explains the different steps to obtain a bounding box and it contains a description of classifier. This description introduces the methods currently used for a human detection. In section 4, we present the construction of a 3D box around a human. We show also, in section 5, the results obtained by our program, the advantages and limitations of this one and the possible improvements for the detection. Finally, section 6 comes as a conclusion.

2 Steps to perform a 3D human detection

This section introduces the different steps of the detection realized during the internship. These ones will be explained in the following sections.

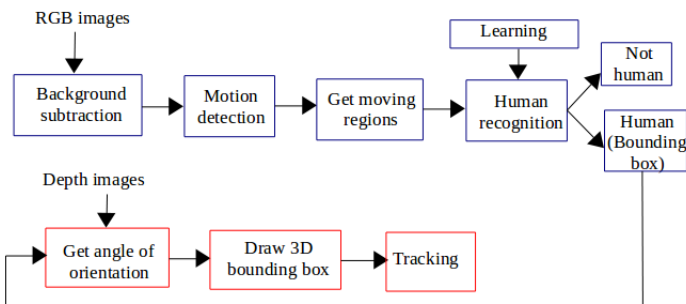


Figure 3: Steps to perform a 3D human detection

Figure 3 shows, in blue, performed steps during TER and, in red, performed steps during Magistere. 2D bounding box is obtained with these following blue steps :

1. Acquisition of the background image.
This image is the room without people and is used to detect motion.
2. Motion detection.
We perform a comparison pixel by pixel between background image and color images sent by the sensor. We obtain a set of moving points.
3. Get moving region.
We assemble points to obtain a detection of several moving regions.
4. Human recognition.
This step allows us to determine whether moving regions are humans. Step learning, show in the figure, was made before execution.

After these performed steps, we have one or several 2D bounding box(es) containing a human. We use it to make a 3D detection.

1. Maximal optimization of the previous steps.
Before construction of 3D box, we have to add parallelism. Without this, the 3D detection can not be performed in real time.

2. Computation of the depth in millimeter.
To know the position of the human, we have to compute the depth in mm thanks to depth images.
3. Computation of orientation angle.
With the 2D bounding box and depths in mm, we can know the orientation of the human.
4. Processing of 3D box.
With the orientation angle, we can create the 3D bounding box.

After these steps performed, we have a 3D box around human.

3 2D bounding box creation

In this part, we explain the steps to get the bounding box. We start with RGB detection to find regions of interest in an image and to send them to the classifier. After that, we explain the methods and algorithms used by the classifier.

3.1 Motion detection with RGB data

The simplest method to detect a motion is using RGB images (color images). Indeed, a 3D sensor sends 30 images per second. To perform detection, we use the background image which represents the room without any human [Wikipedia, 2014a]. With a simple comparison, it allows us to detect changes like, for example, a human displacement in front of the sensor.

A first method to realize this comparison is : for each pixel of each image, we compute the color difference with the corresponding pixel of the background image. If this difference is greater than a given threshold, we suppose that the pixel is moving.

This method is precise enough but we have some errors. Indeed, during brightness change, pixels of illuminated regions change. With this method, these pixels will form a moving region and it is an error.

We can prevent this happening in comparing each image provided by the sensor with the previous one. These errors will be here at the time of lighting but quickly removed in next images. Indeed, the difference with the previous image will be too low in following images. For this second method, we use two thresholds : one for the comparison with the background image and an other for the comparison with the previous image. If the differences are greater than these thresholds, pixels belong to moving region. Difference with the background image is preferred to difference with the previous image since this one is more accurate to detect whole human. In the event of a human is almost static, he will be detected after the comparison with the background image. However, with the previous image, he will not be entirely detected, the difference being too low between 2 successive images. Images in figure 4 show results of RGB detection.

Image 4a is the background image and image 4b represents the RGB image. The white pixels of image 4c are moving pixels and the black pixels identify static regions.

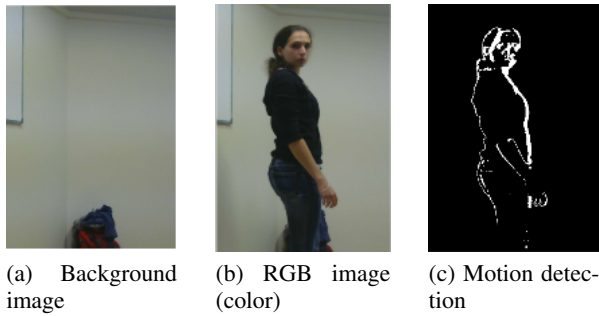


Figure 4: RGB image processing to detect motion

3.2 Computation of connected components

Once moving pixels found, we have to separate the moving regions and to create a bounding box for each of them. For example, in figure 6, we obtain a box around the human and another around the object at the left.

This step allows us, in addition to boxes creation, to remove errors like white pixels which represent this object at the left in the image. So, we realized an algorithm to determine connected components of moving pixels.

With image 4c, we can see that the human is not always entirely detected. His detection depends on his motions. To obtain a single component which will represent the human, we cannot simply assemble neighbor pixels. In fact, the moving pixels are not always directly adjacent. So, we define a region of approximately 50 pixels around each of them. All pixels in this region will be in the same component. (see figure 5)

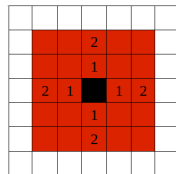


Figure 5: Region of 2 pixels

After defining components, we can remove those which can not represent a human (for example, those containing too few pixels). This process limits the number of images sent to the classifier. Then, we draw a rectangle, around each of them, computing minimum and maximum in width and height. The image part contained in the rectangle is sent to the classifier to be analyzed (see section 3.3).

We can see, on the figure 6, two rectangles, one for each found component. Those identify moving regions. Then, we can use the classifier to know whether the rectangles contain a human.

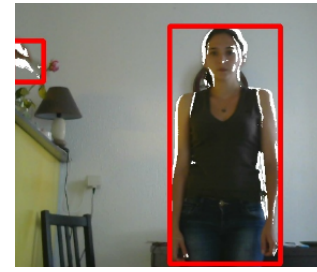


Figure 6: Results after connected components computation

3.3 Classifier

The used classifier is based on the machine learning algorithm Adaboost created by Yoav Freund and Robert Schapire in 1997 [Freund and Schapire, 1999] and [Freund and Schapire, 1996]. Adaboost uses the boosting method. The principle of this method is to use a set of binary classifiers more efficient than random (having a success rate higher than 50%), call weak classifiers. A binary classifier distributes the inputs in two categories. In our case, the inputs are the images and the categories are : “contains human” or “does not contain human”. A linear combination of these weak classifiers forms a final classification function. This function allows us to know whether an image contains human. These binary classifier are determined by HOG descriptors (Histogram of Oriented Gradients) which are really efficient for human detection. HOG are found by Navneet Dalal and Bill Triggs, researchers for INRIA [Dalal and Triggs, 2005] and [Wikipedia, 2014b] in 2005.

Histogram oriented gradients

This section explains HOG computation which allows us to determine the shape of a human in computing different gradients. A gradient is a vector which defines a variation of function. In our case, the gradients defines the color variation. First, the image is divided in several parts with same size, called cells. For each cell, the algorithm performs a histogram containing different orientations of gradients for each pixel of this cell. In order to be more efficient, we then realize blocks, which can overlap. These blocks contain several cells. For each block, we compute the local histograms which will be normalized. They will be the final descriptors used by Adaboost.

Computation of orientation gradient

For a pixel, this orientation is performed with Perwitt filter (a mask). This filter uses different 3x3 matrices like these :

$$G_x = \begin{pmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{pmatrix} G_y = \begin{pmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$$

G_x allows us to compute the horizontal orientation of gradient and G_y the vertical orientation. The gradient orientation of this pixel is computed in using color values of the adjacent pixels. We compute the sum of products between the adjacent pixels and the coefficients of the filter located in the matrix. We realize this computation on an example. This following

matrix contains values of 9 adjacent pixels. The aim is to compute orientation of gradient for the central pixel (this one with b_2 as value).

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{pmatrix}$$

The orientation of the gradient in the horizontal direction is found thanks to the matrix G_x as follows :

$$1 * a_1 + 0 * b_1 + (-1) * c_1 + 1 * a_2 + 0 * b_2 + (-1) * c_2 + 1 * a_3 + 0 * b_3 + (-1) * c_3$$

If the value is positive, the gradient is oriented to the left, otherwise to the right. We do likewise with G_y to obtain the vertical orientation (top or down direction).

Construction of the histogram cell

The computation of the orientation gradient is realized for all pixels of each cell. Once these directions obtained, we construct the corresponding histogram. (see figure 7).

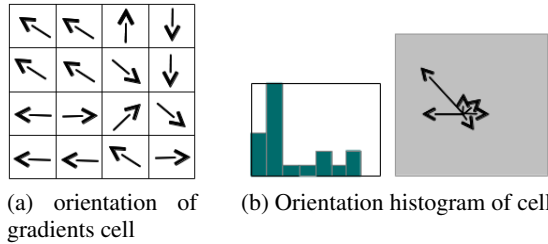


Figure 7: Steps to create histogram

This histogram represents the number of instances of each direction (beginning to the left, then top left, ...).

Blocks formation

In order to be more efficient, Navneet Dalal and Bill Triggs have thought to create blocks. A block contains several cells, so we must define their size. In the figure 8, we will take blocks containing 2x2 cells.

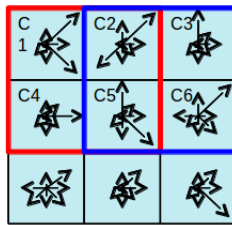


Figure 8: Creating 2 blocks : one red and another blue

We compute local histograms of the block. For this, we use histograms of cells contained in the block. In the example, the histogram of the red block is computed using cells C1, C2, C4 and C5. After, a normalization is made. Each block contains, at the end, a local histogram. They define binary classifiers used to create the final classifier.

Adaboost method

The aim of Adaboost is to perform progressively the final function of classification F . This final function is a linear combination of selected classifiers. Thanks to the algorithm, we obtain an efficient function of classification. It will allow us, from features obtained with HOG descriptors, to classify images containing humans and those without human. First, the Adaboost method uses a set of weak classifiers, H and a set of m examples. Training set is the one of (x_i, y_i) where x_i represents examples and y_i indicate if examples are positive (contain a human) or not. $y_i \in \{-1; +1\}$ for $i = 1, \dots, m$. Each example is assigned to a coefficient of difficulty $d_1(i)$ initialized to $\frac{1}{m}$. These classifiers return, for each example, $+1$ if example is found positive, -1 otherwise. Adaboost begins by choosing the first classifier h_1 in H . It will be added to F , the final function. We choose the classifier with best success rate, the one which minimizes the error of classification. This error is performed as

$$e_t = \sum_{i=1}^m d_t(i) [h_t(i) \neq y_i]$$

In the final function, each classifier is assigned to an integer α_t which is proportional to its success rate. Thanks to this, the classifier with the best success rate will have a weight more important than the others in the final function :

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1 - e_t}{e_t}\right)$$

Difficulty coefficients of examples are updated. Indeed, the examples where the classifier failed must have a coefficient more important than the other examples. So, we will oblige future classifiers to be more efficient with misclassified examples by previous ones. The new value of coefficients is computed as follows :

$$d_{t+1}(i) = \frac{d_t(i)}{Z_t} * \begin{cases} \exp -\alpha_t & \text{si } h_t(i) = y_i \\ \exp \alpha_t & \text{si } h_t(i) \neq y_i \end{cases}$$

where Z_t is a normalization vector. The algorithm chooses other classifiers in this way. It stops when the success rate is sufficient or a certain number of classifiers is reached.

4 Construction of 3D bounding box

To create a 3D bounding box, we use depth data and the 2D box created. In this section, we present the depth images and how to compute depths in millimeters stored in these images. Then, we compute the orientation of human thanks to these distances and finally, we realize a 3D box with this orientation.

4.1 Get depth in millimeter

A depth image contains depth values, in millimeters, for each pixel. It can be available in different resolutions : 640x480, 320x240 or 80x60. Here, we use depth images in 640x480. Generally, 3D sensors are efficient for distances between 80 cm and 400 cm. The depth data in millimeters is encoded on 16 bits. In an image, each pixel is associated with 3 values : blue, green and red values encoded on 8 bits. The depth in

millimeter can be store in different ways. In our case, we use only green and red values like this :

$$depth_mm = (value_green \ll 8) \mid value_red$$

This computation allows us to know the distance between human and the sensor, as well as the human orientation.

4.2 Orientation of detected human

In the section 3, we explain how to obtain a 2D bounding box. Once this one obtained, we get the depth of pixels which are on the right and left ends (see the figure 9).



Figure 9: Computation of human orientation

In the figure 9, the 2 circles represent the extremities on the horizontal axis of the bounding box. After, we compute depths associated with these extremities with the formula in section 4.1.

Thanks to these depths, we can know the orientation of human with a simple trigonometric formula.

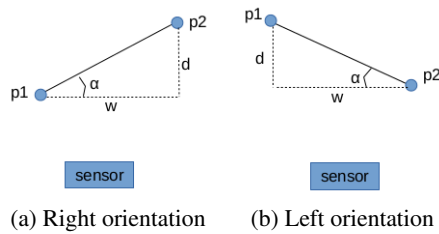


Figure 10: Representation of orientation with a view from the top

In the figure 10a, the depth of $p2$ is greater than the depth of $p1$. d represents the difference between the 2 depths and w is the width of the 2D bounding box. The orientation angle α is computed with these 2 values like this :

$$\alpha = \arctan(d/w)$$

4.3 Production of the 3D bounding box

With the previous subsections, we have the orientation of the human. We just need to draw this box on image, see figure 11.



Figure 11: A 3D bounding box

In the figure 11, the left edge of the box is closer to the sensor than the right edge. Therefore, the depth of point 1 (to the left) is less than the depth of the point 2 (to the right). The thickness of the box is fixed but, with more time, it can be dynamically computed.

5 Results obtained and possible improvements

In this section, we explain implementation choices, advantages and limitations of our program. Then, we present some possible improvements.

5.1 Implementation choices

This program has been realized in c++ in using Opencv library. Opencv is an open-source library created for the computer vision. It has a c++, python, c and java interfaces and can be used with Windows, Linux, Mac OS or Android. I used it for image processing, especially for HOG and Adaboost. For a program in real time, I used multi-threading. For the motion detection, several threads compare the frames. Then, when a motion region is found, the current thread sends this region to an other one which executes the classifier.

During this internship, I did not have a 3D sensor. Therefore, I used my web-cam for the implementation of the 2D bounding box. Indeed, I just need to have RGB images for the motion detection.

However, for the 3D detection, I needed depth data. Thus, I spent some time searching RGB and depth datasets on the Internet [Center, 2014]. Moreover, these images had to contain humans in indoor environment and the website had to indicate the sensor position (for example the height of camera), the maximal depth ... This research took a lot of time of the Magistere period.

5.2 Advantages and limitations of the program

Despite low image quality and depth image rather accurate, the results are relatively right. Thanks to the RGB detection, a moving human is quickly detected when he walks in front of sensor. It is also possible to detect several humans simultaneously (see figure 12). Parallelism has been introduced for the detection to be executed in real time.

Some errors are present if the human motion is too low. Images of the figure 13 represent these errors.

On these images, the human is not entirely detected. This error can be a real problem for the human recognition (see

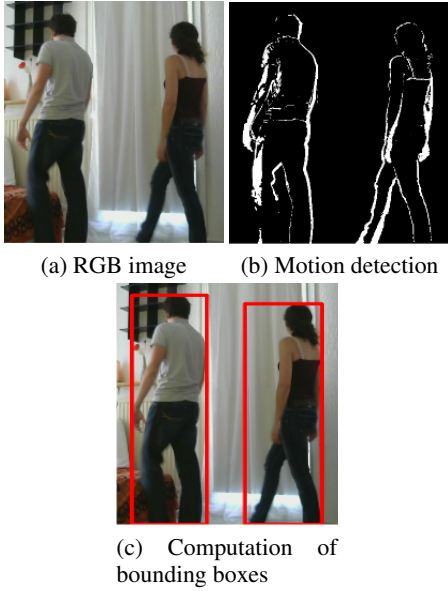


Figure 12: Detection of several humans



Figure 13: Errors of the detection

right image). We can see that there are only arms which are detected. Indeed, body parts detected are only the moving body parts. Therefore, the rectangles containing arms are sent to the classifier and the human is not recognized.

Moreover, the classifier does not always recognize a human. Overall, a standing human is recognized but if he is sitting or crouching, he is not.

5.3 Possible improvements

During this internship, we are interested in an accurate representation. When humans work side by side with robots, the detection has to be more accurate for security. For this, we have to improve the box and to refine the human contours. Figure 14 is a simple example to present this problem.



Figure 14: Insufficient box

In the example, a box is not sufficient. In this section, we explain how to improve the 3D detection. Due to lack of time, these next steps are not implemented. After the program execution, an imprecise 3D bounding box is returned. To obtain a 3D representation more accurate, we can cut this box in different parts (see figure 15).

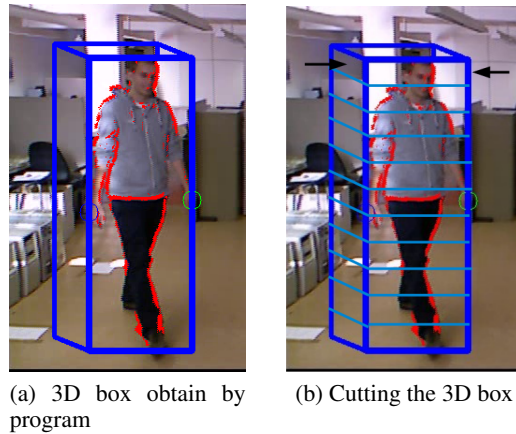


Figure 15: First step to improve the 3D representation

We can first cut the 3D box in several boxes. For example, the cut can be realized every 15 centimeters. A human is no longer represented by a single 3D box but by some boxes. We can after that adjust the different boxes, see figure 16. For this, we research the minimum and maximum in horizontal for each block. We then obtain several boxes with different sizes.

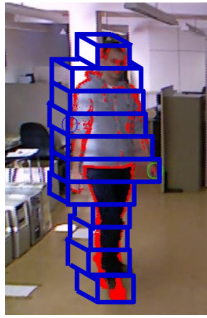


Figure 16: 3D representation

6 Conclusion

6.1 Interests of a human detection

The human detection is an important part in computer vision. It is in a lot of fields as video-surveillance, industry ... Currently, technology is evolving and more and more humans work side-by-side robots.

For example, in automobile manufacturing, robots can carry heavy loads and realize dangerous tasks instead of workers. The detection has to be accurate and in real time for security.

6.2 Analysis of this work

I realized a human detection with RGB and depth images. This internship allows me to find out the computer vision and to use libraries that I had never used.

To create these bounding boxes, I followed different steps. First, I had to learn the OpenCV library to work with images and videos. Once this work realized, I did the motion detection thanks to background subtraction and comparisons with successive frames. Second, I found the moving regions with the computation of connected components. Third, I worked on HOG that I did not know and used the classifier. Finally, if a human is detected, I perform the 3D bounding box around him thanks to computation of the angle of orientation and the approximation of the thickness. Some ideas to improve was found but I did not have time to implement them.

Despite some errors, the results obtained are interesting. Indeed, humans are quickly detected and the program can detect several ones simultaneously. Adding 3D brings more informations about the position and the orientation of the humans in the room.

The 3D bounding box improvement was not implemented but can be integrated without difficulty. With this improvement, a human would be represented by a set of boxes. This method would allow us to obtain a more accurate 3D representation of human.

Acknowledgements

I would like to thank my supervisor Olivier Aycard and Ricardo Omar Chavez Garcia for their attention and their help during this internship.

References

- [Center, 2014] Computer Vision Center. Dgait database, 2014.
- [Dalal and Triggs, 2005] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection, computer vision and pattern recognition. 2005.
- [Freund and Schapire, 1996] Yoav Freund and Robert Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. 1996.
- [Freund and Schapire, 1999] Yoav Freund and Robert Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 1999.
- [Knight, 2013] Will Knight. Smart robots can now work right next to auto workers. *MIT Technology Review*, September 2013.
- [Wikipedia, 2014a] Wikipedia. Background subtraction — wikipedia, the free encyclopedia, 2014. [Online; accessed 25-August-2014].
- [Wikipedia, 2014b] Wikipedia. Histogram of oriented gradients — wikipedia, the free encyclopedia, 2014. [Online; accessed 25-August-2014].

Fast and Accurate Branch Predictor Simulation *

Antoine Faravelon

UFR IM²AG, Grenoble

antoine.faravelon@e.ujf-grenoble.fr

Supervised by: Nicolas Fournel, Frédéric Pétrot

Tima, Team SLS

I understand what plagiarism entails and I declare that this report is my own, original work.



Antoine Faravelon, 21 august 2014

Abstract

Processors are becoming more and more powerful and a few MHz can change performances dramatically. Thus it becomes vital to be able to simulate a processor design as early in the design process as possible. This simulation needs to be accurate and performant as runtime is of utmost importance and so is the precision of the result. This article gives an overview of a simulation process for one particular part of a processor, the branch predictor. Indeed, current fast simulation technologies such as native simulation and Dynamic Binary Translation (DBT) ignore such architectural details. However, they may give precious information to a processor designer or an early developer in the case they would like to estimate the, possibly Worst Case, Execution Time (WCET) of their program. We focus on two predictors. The branch predictor modeling process and analyze of its runtime and accuracy are described in this article. With a simple predictor over 93% precision was achieved while causing an overhead that is inferior to 5%. A slightly more evolved version can raise accuracy by about 3% on average with no sensible further decrease in performances.

1 Introduction

A branch predictor is a component of the processor that guesses the outcome of a conditional branch, i.e if the branch will be taken or not taken. In this article we aim at modeling the behavior of predictors. The principle of simulation is to reproduce the execution of a program on a target architecture on a host machine with a different architecture. The process is actually slow and as such, fast simulation techniques have been developed. Currently, the state of the art in fast simulation ignores architectural details of processors, such as

*These match the formatting instructions of IJCAI-07. The support of IJCAI, Inc. is acknowledged.

branch predictors or pre fetchers. But, in order to calculate or estimate the WCET of a program on a given processor, this details are necessary. Simulating them though, in a fully detailed way is impractical as this would dramatically slow down the whole simulation process. Indeed, most predictors use large tables accessed non sequentially which means that making them run faithfully on the host machine would result in filling the L1 cache (and even L2) of its processor resulting in an important decrease of performances. For example, the ISL-TAGE [André Seznec, 2011] presented for the 3rd championship of branch prediction uses about 64KB of memory in hardware, another example would be any two level predictor. A 15 bits global history will require 2^{15} entries to be used (with a 2 bit counter and a memory wise perfectly optimized program it would be a 2^{16} bits wide table). To avoid that, the predictor should be simulated by approximation. Using information that are not available to the predictor to reduce memory usage or managing it in a different way can allow to simulate it with satisfying results while not diminishing the speed of the whole simulation in unreasonable proportions.

2 Related Work

Related work can be found in the domain of processor modeling and particularly for the computation of the WCET. A survey of simulation tools already used in the computation of the WCET of a program was given in [Reinhard Wilhelm et al., 2008]. From there, the first observation is that most tools use static analysis (after checking, even those that were said to be dynamic often turned out to have become static since this paper). Searching for the method used to simulate branch predictors in these tools, it can be seen that for a number of them there is no information on branch prediction. Dynamic branch prediction was only advertised in the Chronos simulator but it is a static analyzer in fact, working on source code and extracting real addresses from the binary obtained by compiling it. It advertises precise simulation of popular predictors, which seems more attractive than solution proposed in [Xianfeng Li et al., 2004] and [Colin and Puaut, 2000]. They basically tag branches as predicted always taken, never taken, unknown, (or a slightly more refined scheme for the second one). But static analysis methods are hardly applicable to a DBT scheme as it uses information on the whole CFG to simulate the predictor. Dynamic branch prediction was also addressed in simulation with for example [Björn

[Franke , 2008] and [Franke and Powell , 2009]. These are based on statistical models and machine learning. While they provide good results, they require prior knowledge of the processor and the possibility to execute training sets on them. It may still be possible for a programmer after the target architecture has been released, however it seems impractical both for early development and processor design as the final processor is not yet available and having to use a perfectly accurate simulator would reduce the usefulness of an approximate one. Time would anyway have been lost in training which may have to be repeated multiple times as design evolves.

3 Featured Predictor

We mainly study two predictors. The gshare, which is a simple bimodal predictor given as example in cbp3 and the TAGE family which is now considered to be state of the art in branch prediction [André Seznec , 2007].

3.1 A few definitions

Bimodal/n bit saturating counter

Branch predictors are most often based on counters. The counter represents the last outcomes of a branch and is used to establish a prediction. Usually, it is bi-modal (two bits). If the counter has a value greater or equal to 2, then the predictor will consider the branch to be taken, otherwise it will be considered to be not taken. The principle is the same with a counter of any length, if the value is more than half the maximum, then the branch is taken, otherwise it is not. Most of the predictors use a 2bit counter (gshare for example) or 3bit (TAGE for its tagged tables). When a counter's state is $\frac{2^{n-1}}{2}$ or $\frac{2^{n-1}}{2} + 1$ then the counter is said to be "weak" otherwise it is said to be strong. When counters are at their extrema they saturate, i.e. their value will not continue to increase/decrease.

Two Level Predictor

In complement to the counter, one needs some method to store them. The two level predictor model uses a table which contains counters for a given amount of branches. The way this table is indexed varies in function of the predictors. The two featured predictors derive from it. To have a more precise description of this predictor model and some of its derivatives, see [Yeh and Patt, 1992].

Global history

In a global history model, the predictor possesses information on the outcome of the last n branches. A n bit shift register is used, so that each branch outcome is introduced to the left of the register while its rightmost bit is dropped. The number thus formed is then used as an index for a bimodal counter table of size 2^n .

3.2 gshare

The gshare is a Two Level Predictor using bimodal counters and deriving from Global History model. The table index corresponding to the branch is calculated by XORing the Program Counter (PC) of the branch and the global history register. This predictor has been widely used in the past as the model is relatively simple to implement. However, as any

global history based predictor, table size grows exponentially as history size grows. It does offer satisfying results in a certain number of situations but with a reduced quantity of memory, a branch may access to an already in use entry. Correlation with a global history will not always be positive either.

3.3 TAGE

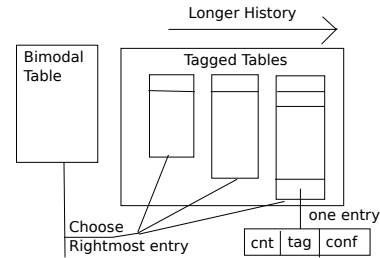


Figure 1: Base elements of the TAGE branch predictor

The TAGE predictor uses multiple model of predictors. One of the two main elements of this predictor is the tagged tables. N tagged tables, each of them using different global history sizes forming a geometric series. The other element is the default predictor [André Seznec , 2007] which is a bimodal table here. Each tagged table has K_n entries. Every entry contains, a 3bit saturating counter, a tag which is a part of a branch's PC and a confidence counter (describes age and usefulness of this entry). To give the prediction, an entry's tag should correspond to the PC of the current branch, its confidence should not be null and the counter not in a weak state. The tables are indexed in a way near that of the gshare : the PC of the branch is XOR'ed with the current global history. But the number of actually used bits of this history depends of the table (the table $N+1$ uses a larger history than table N , however it is hashed as explained in [Pierre Michaud , 2005]). Then, the chosen prediction will be the entry satisfying these conditions in the largest tagged table. If none respect these conditions, the default predictor will be used. An entry in the tagged table will be created when the default table or the current largest table fail to provide a correct prediction. Other components are used in various implementations. In the implementation that was chosen for this work, a loop predictor and statistical corrector were used. The later being a predictor which obeys pre inscribed statistics to bias the prediction if it seems unrealistic [André Seznec , 2011]. Figure 1 describes the base elements of the TAGE predictor. The main advantage of the tagged table over a simple gshare is that the size used for global history is hopefully more optimal as entries in this tables are allocated only when the default predictor and shorter history tables failed to provide a correct prediction. Conversely, for branches of which the outcome is independent from that of the previous ones, the bimodal table will be used which should reduce the number of cases where history is nocive to the prediction.

4 Proposition

This work proposes to add simulation of a branch predictor to existing DBT simulators. These simulators execute programs from target architecture by dynamically translating the binary code into instructions that are executable by the host. The functional core of this simulator will not depend on our predictor simulator. If the prediction is wrong, what will suffer is the runtime estimation of the said binary code. Indeed, while in hardware a wrong prediction degrades performances, here only the estimation of its performances will be affected. In the case of DBT, only runtime information can be used for simulation, basically, anything that can be readen in the execution unit of a processor, that is, the instruction itself, its arguments, its PC and the state of the registers before and after its execution. This simulator may also be extended to native simulators. In native simulation, the source code of the program is first compiled using specific techniques to obtain a binary of which the structure is the same as it would have been if it had been compiled for the target machine. Then it is executed on the host. This binary can be annotated during the compilation process with the original source code and any information static analysis can provide(A Control Flow Graph(CFG) for example). This annotations could then be used by the predictor simulator in order to make its prediction not only on runtime information(which would actually not be fully accessible as register state in original predictor or the real PC can not be found in this execution flow as it is compiled for the host machine). The estimated prediction given by this simulator will then be used to compute the time an instruction should take to execute, if the estimated prediction is not the same as the direction taken by the branch, then it will be estimated that the processor would have suffered penalties in term of performance due to fetching not useful instructions and polluting its cache as a consequence for example. The estimation of the execution time of each instruction will then be used to give an estimation of the WCET of the whole program by adding the execution time of each instruction of each path taken in the program.

In the simulator, just as in [Xianfeng Li et al., 2004], the program is modeled through a CFG. Here, it is generated dynamically as the trace runs. The end of each block is the conditional branch that is being predicted, the start is the target of a branch. Each edge corresponds to the target of a branch (the address reached if the branch is taken) or to the next instruction(that will be executed if the branch is not taken). This information is acquired by reading the PC(for the address of the branch) and the target that is given by the branch instruction. Each block is a "node". The most important part of this node being the branch(other instructions are ignored). Through the edges going to each node, the parents are also accessible, potentially allowing to relate branches to their predecessors. Nodes are numbered as they are found, a "distance" between nodes(and branches) can then be defined.

In order to obtain a predictor that is as near as possible to the reality, the simulated predictor was created from the original model. The simulator can be seen as a simplification of

the "optimal" simulator(e.g. a simulator precisely following hardware model) but developed in reverse. First, the "core" is implemented(instead of first the full feature). This core is the set of counters. When simulating a gshare it is particularly seen as a simple 2bits counter stored in a hash table indexed with the start address of the block(with no XOR'ing of the address) is enough to give good result. As for the TAGE, using the same table and adding a simple approximation of tagged tables to access 3bits counters is what allowed us to obtain the first version of the predictor simulator.

The main interest of this simulator is to be usable in fast simulation, it is oriented to a DBT scheme but could be extended to native simulation. It does not rely on prior training as the method used in [Franke and Powell , 2009] and the performance impact, according to current experiments, is acceptable for real use. Also, this simulator gives a solution that is universal which is limited in the case of machine learning as unknown situation that occurs may lead to imprecision on programs that are not yet known, while this simulator perform independently of prior knowledge and as such will depend only on the content of the program(though, as seen in the Experimentation section, it does not perform equally on every type of traces). Implementation in RABBITS also showed that such a simulator can be integrated to a DBT simulator quite naturally as the structure used by these are very similar as to the ones needed by the predictor simulator to compute its predictions.

This work however, also shows the limit of the approach simulating a branch predictor without static analysis or machine learning as in the pre cited works. Indeed, being in a situation that is near that of the real predictor, simulating it, using less memory and being faster becomes actually a form of optimization of the said predictor, which has many potential limits. Though there is still a few advantages over it such as more available memory and more flexibility in the organization of information, these are more an asset to make better predictions (see idealistic predictors of the branch prediction championship) than an help to simulate it.

5 Experimentation

In this section, the method used to create the simulator and the tools that were used will be described.

5.1 Tools

cbp3 simulator

The first tool was the program provided to participants of the 2011's championship branch prediction(organized by the Journal Of Instruction Level Parallelism). The default predictor(gshare) was used in the first phase of this work. For the second phase, the ISL-TAGE predictor provided by André Seznez was used. For both, the driver of the championship served as a basis. The driver is used to give the rhythm of the simulator. When the predictor report its prediction, the simulator is called to provide its own and they are then compared. At the end the driver will provide statistics on the simulators performances. These statistics were the ones that were used to provide the results presented in the next section.

DBT Simulator : RABBITS

In a second time, the work focused on RABBITS, the DBT simulator developed by TIMA and based on qemu. Both the simulator and the original predictor were implemented and are now executed in the translated code. This allows to monitor their execution in a realistic environment and thus measure the respective impact on execution time of the whole simulator(with potential for the misprediction costs).

It should be noted that annotations, that is code which is not part of the original program's translation, such as the branch prediction here have no influence on the functional's code execution. A misprediction will not change the way it is executed, neither will a prediction that is not the same as the one from the original predictor. The costs of such mistakes will be to reduce the accuracy of the estimation about runtime and performance in general. Execution of these annotations will have an influence on overall execution time of the simulation though, and this is what is measured here as an overhead.

Branch Predictor Simulator

In the functional simulator the simulator uses information provided by the driver in the same way as the original predictor. It gets the PC and the real outcome to compare with the predicted one from the driver's "uop" structures. It does not however use any information on the outcome of the original predictor as this will not be accessible anymore in a production system. Only Data available in a real DBT simulator are accessed. These information are stored in a Dynamically generated CFG along with static and dynamic information generated by the simulator. The main information that are used are two counters, a 2bit one and a 3bit one that are used to simulate the ones from the predictor, a block number which is used to simulate the distance between two last blocks and thus recreate the idea of tagged tables in a more simple way(and without the tags as the blocks are indexed by the PC of the branch leading to them). Other information that are kept are the parents of the block(the block that potentially can jump to it), the last (real) outcome of the branch and, if the branch is a loop, the length of the iteration and the confidence on the loop predictor's prediction. This last information allows to potentially simulate a L-TAGE instead of a TAGE. Some information will not be kept for final version for reasons that will be explained in Results section.

When simulating the gshare predictor in preliminary work, it has become immediately apparent that creating every entry of the table required for the gshare would lead to high performance overhead as memory consumption of the gshare is exponential with the size of the history which would fill the cache of the processor easily and dramatically reduce performances of the complete simulator.

The final version of the simulator(in RABBITS) principally uses the PC of the branch and its condition so as to be able to compute the outcome at runtime. RABBITS structures are used to store the predictor simulator's information instead of using its own data structures. This version of the simulator(which has been back ported to the functional simulator for tests) simulates the tagged tables by using an 8 bit history size as an index for a 3 bit counter table contained in each block(so one by branch). This allows to have a virtu-

ally unlimited number of entries as only the potentially useful ones are loaded at one given moment thus not filling the host caches for naught. The latest version is as in the figure 2 .

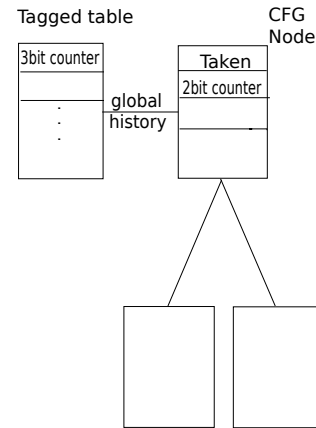


Figure 2: structure of the simulator

5.2 Evaluation

Functional Simulator

In order to determine the efficiency of the simulator, the metrics used were execution time and precision. For precision, the cbp3 executable was used. It executes both the predictor and the simulator, compare their results and then give a set of statistical values such as percentage of good prediction, false negatives and false positives. For the second metrics, three executable were created, cbp3-none which only executes the naked driver(no predictor or simulator), cbp3-sim which executes only the simulator and cbp3-pred which executes only the predictor. Then, each of them were launched on a asus S400CA laptop with a Intel(R) Core(TM) i3-3217U CPU @ 1.80GHz on ArchLinux(Rolling Release) and execution time(user) was captured using time command. The time itself may vary depending on processors and condition on the machine. This information is used as an indicator of performance as the ratio gives a good idea of performances(in term of execution time) of the simulator over that of the original predictor. Also, the predictor against which the program was tested is the ISL-TAGE as it is the latest and it seemed interesting to analyze the importance of added modules.

RABBITS

The same metrics and test machine were used as for the functional Simulator. Two platforms of the RABBITS simulator were used : roger(which runs linux and whose exact specifications can be found on TIMA's website), and thumper which allows to run bare bones programs(or with a minimal OS). Both synthetic benchmarks and realistic ones were launched on the thumper platform, the roger platforms was used only for the linux boot process.

5.3 Results

As a demonstration of feasibility, a simulation of the TAGE/L-TAGE/ISL-TAGE has been realized and evaluated

here. The result of this evaluation will be presented in this section.

Performances and Precision of Functional simulator

Following the methods explained in previous methods, the following result were obtained :

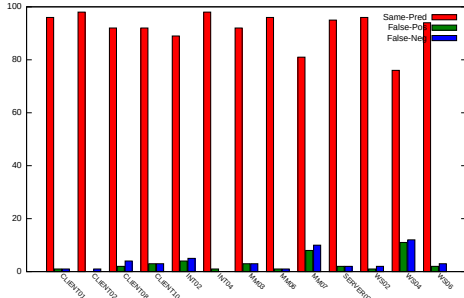


Figure 3: Precision of the Simulator Without Tagged table

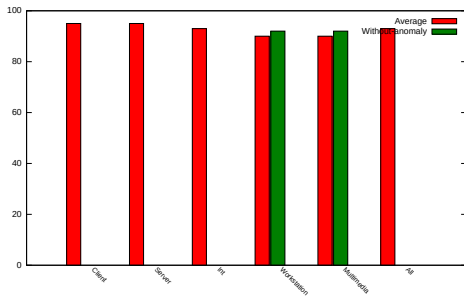


Figure 4: Precision of the Simulator on average

The result shown in figure 3 and figure 4 shows the precision of the simulator that is, the number of time where the simulator gives the same prediction as the original predictor. In most cases, the simulator is above 90% of good predictions(93,52%on average) and, for client and server programs, above 95% for almost all traces(95,40 % and 95,69% on average). Three anomalies appear : WS04, MM07 and, to a lesser extent, WS03. However, after deeper observation it revealed that mostly every errors in WS04 and MM07 comes from a few branches that appear in a cycle. These anomalies have actually been reduced with the use of a faithful emulation of one tagged table.

Another phenomenon to observe is that there seems to be no real bias on false positive and false negatives. On average it appears that there is near the same proportion of both.

It should be observed that even if anomalies are eliminated, the multimedia and workstation have result lower than the global mean. Still, with or without them the precision is still above 90% on average.

In table 5 it can be observed that the simulator's overhead(over driver only) is very low. It is also much lower

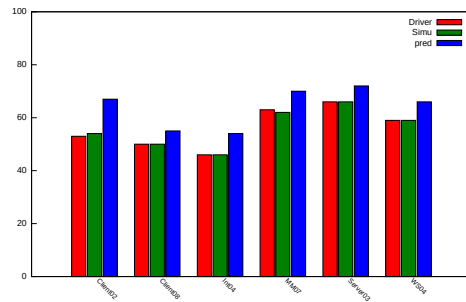


Figure 5: Performances of the simulator without table(seconds)

than that of the original predictor which has important overhead(especially considering that loading a trace takes multiple seconds).

Overhead of the original can be as high as 13 seconds, causing an overhead larger than 20% on the client02 trace while the simulator only give an overhead of 0.8s(less than 2%). On other traces it is around 6-7%.

Influence of Loop Prediction(and other precise details)

The simulator is actually able to simulate Loop prediction also. This however has been deactivated on final version as result is actually negative, Here are the result on a subset of the traces:

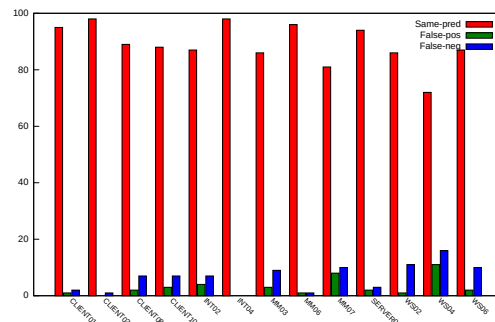


Figure 6: Precision of the Simulator with Loop Prediction

And the average by type of traces:

As it can be seen on the figure 6, on average precision of the simulation decreased slightly, many traces are now below the 90% of accurate simulated prediction threshold. This tends to show that when adding more precise details of the simulated predictors decrease its accuracy. The Workstation and multimedia traces type are the one that suffered the most from the loss of precision(see figure 7) while the server ones remain mainly unscratched.

Another interesting result is that false negatives seems to have raised consequently on some traces, thus giving an indication for the direction that analysis of this loss should take, that is, analyze case where the simulated loop predictor answers false. This should be done in the future as it could lead

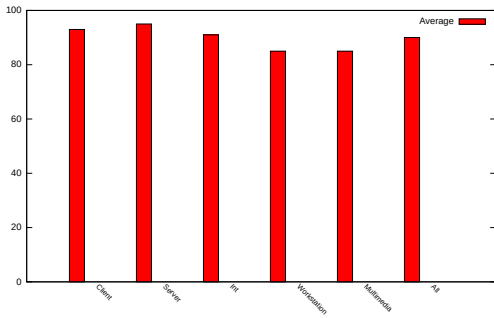


Figure 7: Precision of the Simulator with Loop Prediction on Average

to better overall precision of the simulator.

It is important to note that in most cases, simulating precise details(confidence factors, pseudo random number using bimodal counters etc.) will deteriorate the overall precision of the simulation in the same way as the Loop predictor. Performance remains relatively the same as in any mode the elements necessary for this simulation are still memorized and memory is the main limiting factor here. Eliminating this elements may however lead to increase in speed especially in the final simulator. This is why study of non useful information is important, it allows to save memory by eliminating it.

Table Simulation

On the other hand, simulating the tagged tables in a more faithful manner can lead to an important increase in precision. By simulating a tagged table using a global historic of size 8 leads to the results presented in figure 8. The corner cases where the simple predictor had its lowest precision yield better results, over 6% for the WS03 which gives the worst results for the simulator without a tagged table and about 3% in most other traces.

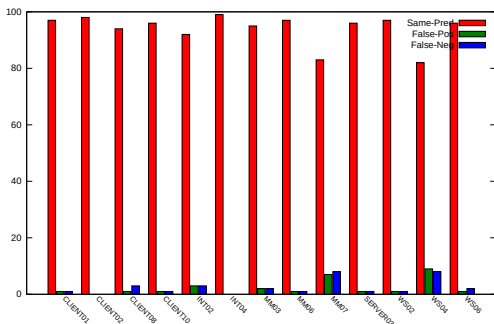


Figure 8: Precision of the Simulator With Tagged table

Since the main difference between this version of the simulator and the previous one is the use of a global history it can be deduced that the prediction(and thus outcome) of the cycle of branches mentioned earlier is dependent on the outcome of previous branches. Thus the reason why the simulator fails is that the history was not really emulated

before. The results now though better are not perfect. This can most probably be explained by the fact that the size of historic used in the table does not correspond to the one used for all these branches. Some of them are probably related to older branches which here are not present, or less likely a shorter relation is needed(the size of historic used here is the second shortest). This could then be solved by implementing more tables. This however would be costly as more tables will consume more memory and potentially increase the number of memory accesses due to the research of the matching entry in the longest table.

Performance And Precision of DBT Simulator

Firstly, precision of the branch prediction simulation in RABBITS was evaluated. This allows to check its implementation by verifying that it performs the same as in the functional simulator. The results are exposed in figure 9.

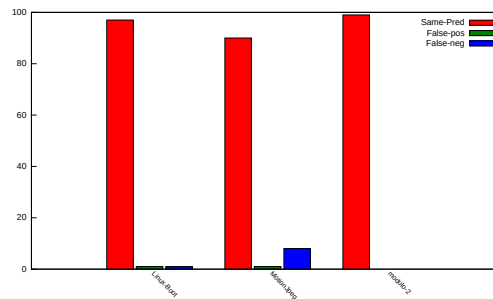


Figure 9: Precision of the simulator

The results we observed are near that of the ones in the functional simulator which tends to confirm the correctness of our implementation. Precision of the simulator in this reduced set of known programs are satisfying. The test of the branch taken when the iterator of a loop is multiple of two shows result that are the same with the original predictor for 99,92% of predictions. This shows the ability of the simulator, just as the tag, to choose the tagged table when needed as a bimodal table(and to some extent the first version of the predictor) would fail to predict this case. Indeed, the history of branches outcome is need here to determine patterns where the branch will be taken or not taken. The lowest precision is attained on the Motion Jpeg decoding(ParallelMJPEG software executed on thumper platform of RABBITS) where precision is slightly above 90%.

After verifying the precision and thus the correctness of implementation, the performances, that is the overhead of branch prediction was measured.

In the set of tests, performances of both version of the simulator are quite near. The most important observed difference in overhead is seen on the ParallelMJPEG test, with 36s more for the final version(less than 3%). The overhead of the final version being 4,76%. The fully implemented TAGE predictor's overhead is much more important. In

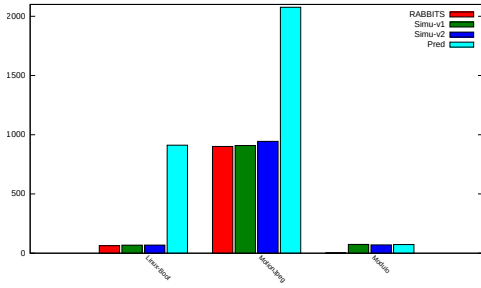


Figure 10: Performances of the predictor simulator's two versions against RABBITS alone and the TAGE predictor

case of a linux boot sequence, it is about 15 times slower than RABBITS alone. On the other tests, it is about 2 times slower as can be seen in figure 10.

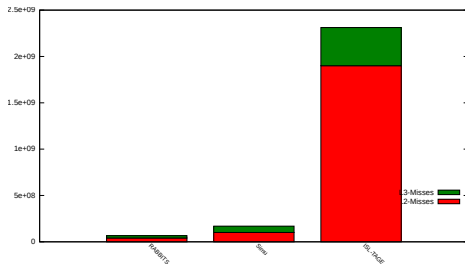


Figure 11: Cache Misses during Linux Boot

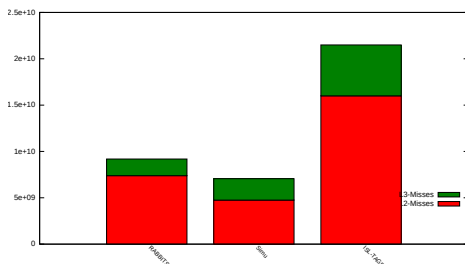


Figure 12: Cache Misses during Motion Jpeg Decoding

Figures 11, 12, 13 show the number of cache misses(L2 and L3 as L1 was not possible to be measured) in the different traces that were tested for performances. In each of them, the original predictor causes multiple times more cache misses than the branch predictor simulator or RABBITS without branch prediction. In particular, in the case of the linux boot there are 19 times more L2 cache misses with it enabled. This explains such a degradation of performances in this trace(about 15 times slower than RABBITS alone). For the simulator, the difference is relatively low. In the case of the Motion Jpeg decoding and modulo 2 there are even less L2 cache misses than without it. This could be explained by the fact that, by executing the branch prediction simulation

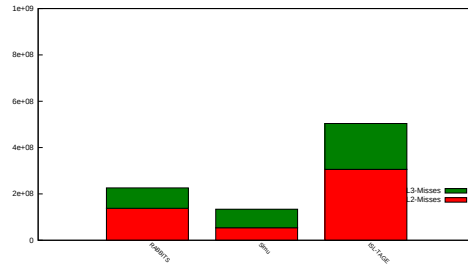


Figure 13: Cache Misses during Modulo2 test

before the branching itself, some data that will be useful to the following parts are fetched in the cache by the processor with a better timing.

In view of these results, correlated with the runtime, it seems that memory is indeed the main limiting factor to the performance of branch prediction simulation. It confirms the need to simulate the tables of these predictors in a way that is less costly in memory accesses and, more importantly, that causes less cache misses either in the simulator itself or after its execution.

Comparison to other predictors

Finally, in order to show that the simulator is not just a generic simulator but is adapted to the TAGE, a comparison was led with other predictors, the gshare and the bimodal. This has been done both with the functional simulator's traces and real programs in RABBITS.

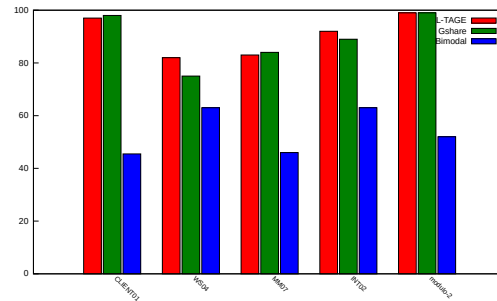


Figure 14: Comparison with multiple predictors

From the figure 14 one can first observe that the bimodal and simulated predictor results are quite far. For the trace CLIENT01 there is only 45,49% of matching results while 97,79% of predictions of the simulator are the same as for the ISL-TAGE.

In the case of the Gshare results are more mixed. While the trace WS04 exhibits 7% reduction in matching rate compared to the TAGE, it is slightly superior in the case of the trace CLIENT01. This can be explained by the fact that the gshare and tagged table function in a similar way. But there are two main differences, for the TAGE and simulator counter size is 3bit and, whenever the counters are in a weak state, an entry in the alternate predictor(a bimodal table) is used instead of the table. This means that if a branch outcome is not related to the previous branches then it will probably not be processed

by a table indexed using global history but only this branch PC thus obtaining results possibly different from the gshare's.

6 Conclusion And Future Work

6.1 Analysis of this work

Functional Simulator

From the obtained results, multiple points should be remarked:

First, to make the simulation perform satisfyingly, multiple aspects of the branch predictor were simplified. For instance, while the real predictor has multiple tagged tables indexed by a hash of the branch's PC and multiple global historic sizes (one for each table), the predictor simulator either simulate it using a condition on the distance between blocks and a PC indexing or use only one table with an historic of size 8 (but unlimited number of entries) indexed using the PC of the branch and the global historic see figure 2. In the end, the creation of the branch predictor simulator was mainly a work of simplification. Keeping most relevant information and mechanism. One information that appears to be the most important was the counter(s) size(s). Just having a counter of the good size with a PC indexed table was enough to simulate with very high precision a gshare and it is the main base for the TAGE. The main focus of this work was to analyze information so as to determine the useful ones. The second focus was to experiment in order to assess the precision and overhead caused by the resulting simulator, improving it when possible. Another important part was to organize information in an efficient way so as to reduce memory related overhead. A probable side effect of simplification was to damage the ability to simulate precise details of the original predictor as these are based on some of its behaviors which may not be replicated.

In the end, a first established method to simulate predictor is to extract the counters, their size and when they are used. Then, by simplifying the way they are accessed, by grouping all information together and then using notion such as distance between blocks (which the original predictor could not possess), it becomes possible to imitate various costly details in a less heavy way at the price of reducing precision.

It is important to not forget that while making a "perfect" simulator would allow to reach a state without any mistake, branch prediction is not the only part of the main simulator (binary Translation being an obvious other part) and as such should not monopolize all resources.

But this work also seems to exhibit limits in simulating predictors in this condition. In the end, this work is near to an optimization of the predictor while accepting a loss in precision in exchange for reduced overhead as in [Pierre Michaud, 2005] where the author processes by successive degradation of its idealistic predictor to reach a realistic one. The optimization has obvious limitation, one of them being that the original predictor has probably already been deeply analyzed and optimized leaving only a shorter margin. As for simplifications, each of them needs to be tested to analyze the damages caused by it, which in the end cause problem akin to the learning phase of a machine learning driven simulator such as the one from [Franke and Powell, 2009].

Real Simulator : RABBITS

Implementation of the branch predictor simulator in RABBITS has shown satisfying results. Precision wise, it performs the same as the functional algorithm as it uses the same algorithms. Performance wise, adding the branch predictor simulator had a relatively low impact. This is possible thanks to the simulator structures, as the information that are needed by the DBT simulator are near in term of organization (basic blocks are cut along branches) to what is needed for the branch predictor simulation. Thus information can be retrieved from this data structures when they are the same as the one used for by RABBITS and the missing ones can be added to them. This allows to improve performances compared to the functional simulator as the costly management of data structures is avoided and some memory consumption is potentially avoided.

On the other hand, the original predictor's impact on performances is important. On the roger platform, booting linux is about fifteen times slower (15 minutes on the test machine) when enabled. This shows that using it for realistic benchmarks is quite problematic, as these benchmarks tend to focus mostly on workloads after boot. Overall it causes multiple times more cache misses than its simulated counterpart in any of the benchmarks. This shows that in any program heavy performance loss is to be foreseen as L2 and L3 cache misses can be extremely costly. Thus, the need for approximation of predictors is shown to be quite important in a real DBT simulator.

6.2 Future Work

This work has shown that it was possible to simulate predictors such as the ones from the TAGE family with reasonable decrease in performance for a satisfying precision in a DBT simulator. It could be used, coupled with the appropriate data, to estimate the runtime of a program in a more precise way by computing misprediction penalties when they are predicted to occur.

However, while results are satisfying, it seems that integrating this work in a native simulator could lead to enhanced precision. Indeed, in native simulation, not only the CFG, but also the source code of the program is available which could allow to detect relations between branch thus enabling the simulator to predict statically in which table with which history size the branch should be placed. Thus reducing once again or eliminating the corner cases in which the simulator does not perform as satisfyingly as on average. These cases being seemingly related to branch history. The CFG and that knowledge coupled together could at least lead to predict the number of tagged entries needed in the simulator thus reducing or suppressing runtime memory usage when the prediction can be done by static analysis.

In a more general manner, tests should be led in order to determine how many tagged tables can be simulated, evaluate the gain of precision and the degradation of performance for each added table.

References

- [André Seznec , 2011] André Seznec. *A 64 Kbytes ISL-TAGE branch predictor*. JWAC-2 : Championship Branch Prediction, june , 2011.
- [André Seznec , 2007] André Seznec. *The L-TAGE Branch Predictor*. Journal of Instruction-Level Parallelism, July , 2007.
- [Pierre Michaud , 2005] Pierre Michaud. *A PPM-like, tag-based branch predictor*. Journal of Instruction-Level Parallelism, April , 2005.
- [Björn Franke , 2008] Björn Franke. *Fast Cycle-Approximate Instruction Set Simulation*. SCOPEs '08 Proceedings of the 11th international workshop on Software & compilers for embedded systems, 2008
- [Franke and Powell , 2009] Björn Franke, Daniel Christopher Powell. *Using continuous statistical machine learning to enable high-speed performance prediction in hybrid instruction-/cycle-accurate instruction set simulators*. SCOPEs '08 Proceedings of the 11th international workshop on Software & compilers for embedded systems, 2008
- [Reinhard Wilhelm et al. , 2008] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, Per Stenström. *The Worst-Case Execution Time Problem Overview of Methods and Survey of Tools*. ACM Transactions on Embedded Computing Systems ,April, 2008
- [Colin and Puaut, 2000] A. Colin and I. Puaut. *Worst case execution time analysis for a processor with branch prediction*. Journal of Real time Systems, May, 2000
- [Xianfeng Li et al., 2004] Xianfeng Li, Abhik Roychoudhury and Tulika Mitra. *Modeling Out-of-Order Processors for Software Timing Analysis* Proceedings of the 25th IEEE International Real-Time Systems Symposium, december, 2004
- [Yeh and Patt, 1992] Tse-Yu Yeh and Yale N. Patt. *Alternative Implementations of Two-Level Adaptive Branch Prediction* The 19th Annual International Symposium on Computer Architecture, May, 1992

Towards simpler performance troubleshooting for kernel-intensive workloads

Hugo Guiroux

MoSIG Student, UJF

ERODS Team, LIG

Supervised by: Renaud Lachaize and Vivien Quéma

I understand what plagiarism entails and I declare that this report is my own, original work.

Name, date and signature:

Abstract

The design of recent machines is inherently distributed and parallel. Such hardware architectures introduce performance problems, which are hard to discover and understand. We propose a set of profiling metrics helping programmers to easily and rapidly pinpoint performance hot-spots related to the usage of kernel mechanisms in their application. As a starting point, we study Hackbench, a massively multi-processes micro-benchmark making intensive use of communications channels and exhibiting non-trivial performance anomalies. This study allows us to show different metrics and visualization technique said to be resource-centric: we observe the actions leading to inefficient behaviour by taking point-of-views of resources (e.g., processes, communication channels).

1 Introduction

The hardware architecture of modern computers is extremely complex. Indeed, in order to overcome physical limitations, the design of recent machines is inherently distributed. Today, any machine has several cores (multi-core architecture) and can thus run different execution flows in parallel. Besides, the main memory is also split into several nodes, leading to non-uniform memory access times (NUMA): a memory access from a core located near the destination node is faster than from a remote core. Besides, the degree of distribution (number of cores and nodes) increases with each new hardware generation.

The operating system kernel allows programmers to exploit such diverse architectures by abstracting the underlying machine. It also provides different mechanisms easing the work of the application developers for common tasks (e.g., communication, process management) without worrying about the system complexity. But achieving an efficient usage of these mechanisms is hard, even for experienced programmers. In such complicated environments, understanding performance problems is often much harder than discovering and addressing them.

The work presented in this report aims at simplifying the analysis of performance problems that arise from the usage of kernel mechanisms. We are particularly focused on the context of server-side applications (e.g., web servers, storage services, data analytics). Such applications are generally very difficult to analyze because (i) they stress the kernel mechanisms and (ii) they have workloads that may greatly change over time.

A server-side application is structured as a set of concurrent execution flows (processes, threads), which interact both explicitly and implicitly. Many performance problems are actually caused by inefficient patterns for such interactions. An example of explicit interaction between execution flows is communication via IPC channels. An IPC channel is a kernel-managed facility enabling several processes that run on the same machine to exchange data. Well-known examples of IPC channels are the pipe (unidirectional) and the Unix domain socket (bidirectional). A common use of such sockets is on a server-side web application, in which a request is first received by a web server, which communicates with another process to do the business logic and so on: each process doing a part of the request handling and communicating with others using IPC channels. Several problems can arise from the misuse of IPC channels, which can be related to a producer-consumer pattern. For example, we can imagine one process producing data too fast or another one not consuming data fast enough. Thus, bottlenecks appear, leading to performance issues. The overall goal is to get a flow of data going through the chain as fast and as fluidly as possible.

An example of implicit interaction between execution flows is synchronization on a lock protecting a shared data structure. Different processes may concurrently invoke a system call, which internally has to access to a kernel shared data structure. Those accesses must be synchronized to maintain the consistency of the data. To do so, each process will try to acquire a shared lock. This lock will be held by only one process at a time, thus forcing the others to wait on it. The situation where multiple processes wait on a lock will create contention. And such a situation is one example of inefficient implicit interactions leading to performance degradation.

Overall, pinpointing inefficient interactions involving concurrent execution flows and kernel mechanisms is hard (both for explicit and implicit interactions) and the need for profil-

ing tools helping to do so is crucial.

Our long-term goal is to design and implement a profiling tool in order to help programmers detect and understand performance problems related to inefficient usage of kernel mechanisms.

There are several criteria for such a tool:

Obvious alerts: the tool must allow developers to easily and rapidly understand the cause of performance hot-spots thanks to precise indicators: indicating which resource (e.g., process, IPC channel) is subject to performance issues using different metrics to explain why and showing inefficient patterns pinpointing which processes are involved.

Overhead: as in all profiling tools, an overhead is to be expected: observing the execution of an application has a cost. The one of our tool must be the lowest possible. We are considering production server workload: we do not want to induce service degradation nor service outage. We can not let our tool stress the observed application in order to gather performance information. Besides, the behaviour observed with the tool must be close to the original one. Inducing too much overhead can remove performance issues (e.g., problems due to lock contention where overhead can reduce the stress on the lock acquisition).

Dynamic instrumentation: ideally, a user should simply be able to run the tool without having to modify or recompile the kernel or the application.

We tolerate at first to not fulfill all the requirements, especially the one about dynamic instrumentation. For the moment, we modify the kernel manually and recompile it, allowing us to validate our approach before trying to do it in a more flexible way for the end user.

Our approach is incremental: in this report, we focus on the first step towards the design of our profiling tool. More precisely, we identify which metrics provided by kernel tracing/debugging facilities to use. Indeed, we observed that taking a “heavy” approach by collecting as many metrics as possible and then processing them is not feasible. The induced overhead is too high. Thus, we need to select the necessary and sufficient set of metrics to build our diagnosis.

Then, we devise how to combine such metrics in order to get a point-of-view allowing us to discover and understand performance problems. Our approach is supported by the case study of discovering inefficient kernel mechanisms usages in Hackbench, a micro-benchmark for Linux that makes intensive use of communication and scheduling facilities in a massive multi-process context. This allows us to identify different metrics that can be used in a future profiling tool to detect performance issues. Our study describes the different techniques we used to understand performance problems at application and kernel level. We plan to generalize this manual approach as well as automatize it through a profiler tool.

This report is organized as follows. Section 2 presents available profiling tools and methods and their limitations in our case. Section 3 presents Hackbench, the configuration

we used and the behaviour we observed. Section 4 shows the approach taken to understand inefficient interactions patterns, which metrics and methods we used. Finally, sections 5 and 6 concludes about different metrics that can be used for automatically pinpointing problematic interactions patterns and presents future works.

2 Background & Related Works

As mentioned previously, the need for tools helping programmers to find performance issues in their programs is crucial. In this section, we first describe the main existing tools related to our work. Then, we explain why they are not sufficient to address the specific problem that we target: performance troubleshooting of kernel-intensive workloads.

Most profiling tools are based on event collection and processing [Wisniewski and Rosenberg, 2003]. Events are timestamped information carrying structured data that indicate something happened in some context (e.g., a function call with the corresponding arguments and call chain). A profiler operates in two successive phases:

Event collection: an online phase (also called tracing phase) where the profiler gathers all information it needs.

Event processing: an offline phase (i.e., after the execution of the application) where events are sorted and processed. Different kinds of processing can be applied to such a list: statistics aggregation, temporal flows construction, etc.

Decoupling these two phases allows easily implementing different strategies to process the data gathered before. It also reduces the profiling overhead while the application runs by doing the event-processing offline, as such a processing has generally a greater overhead than the collection phase.

The event processing phase will be the main part of our contribution. Concerning event collection, several techniques are available for the Linux kernel [de Rochefort, 2014]. Due to a lack of space, we only discuss two of the main ones below.

LTTng [Giraldeau *et al.*, 2011] is a set of tracing tools aiming to be highly efficient. It uses kernel and hardware facilities as event sources. Even if the main goal of LTTng is event collection, a plugin architecture is available to process these events. Previous work ([Fournier and Dagenais, 2010]) presents a use case with a LTTng plugin inferring performance hot-spots at application level. However, we observed that LTTng introduces a significant performance overhead on highly parallel applications. LTTng uses separate processes to collect events. These additional processes can impact the scheduling of a multi-process application. Moreover, adding new types of events in the kernel requires modification and recompilation of the tool, which is problematic.

Our choice for events collection is to use the Linux “perf” tool [Weaver, 2013]. perf uses the “perf_events” kernel mechanisms to gather different sources of events such as software counters (e.g., number of context-switches), hardware events (e.g., memory accesses) and kernel static tracepoints (interest points such as scheduling of a process, defined in the kernel source code). It allows building a timeline of events (called a trace) that can be processed offline. The integration of perf

inside the Linux kernel is a guarantee of its quality. It also insures that the tool will be maintained for a long time. Besides, changes inside the kernel like the insertion of new tracepoints do not force recompiling the tool, which is convenient. All the available tracepoints are dynamically discovered by perf (via the sysfs kernel interface).

Regarding the processing phase, a recent survey [Wang *et al.*, 2013] shows that profiling tools can be classified into two categories: code-centric and data/resource-centric tools. Most profiling tools such as VTune [Intel, 2013] and CodeXL [AMD, 2014] are code-centric: they correlate blocks of code to performance hot-spots such as the elapsed time spent within a function or the number of calls of a function.

The code-centric is poorly-suited for our problem: it does not allow us to identify the types and instances of the impacted resources. Resource-centric tools are better suited to solve these problems.

An example of the resource-centric approach is MemProf [Lachaize *et al.*, 2012], a memory profiler developed by the ERODS team. It is designed for NUMA machines, where inefficient memory accesses can be on remote nodes and thus lead to performance issues. Its goal is to allow application programmers to easily detect inefficient memory access patterns and pinpoint their cause. To do so, MemProf builds two kinds of temporal flows: (i) flows of memory accesses from a thread point of view: this allows knowing which thread or group of threads access which objects and (ii) flows from an object point of view: MemProf is able to semantically identify objects in memory and thus to deduce accesses to them. Using temporal flows for threads and objects can help programmers identifying the precise memory behaviour of an application, by highlighting which run-time entities and which execution phases are causing inefficiencies.

Several previous works on resource-centric profiling tools have been proposed by the HPC (High Performance Computing) community. Projections [Kalé and Sinha, 1993] is a framework helping programmers to understand performance problems of parallel applications. However it only targets applications written in Charm++, which is a programming language derived from C++. Paraver [Pillet *et al.*, 1995] and Scalasca [Geimer *et al.*, 2008] are two tools providing useful information to the programmer to investigate performance problems due to communication in highly parallel environments. However such tools are designed around the assumption that programmers have a great control and knowledge about machine resources and workloads: HPC applications either implement resources management in user mode (with better knowledge of resource usage) or give information to the kernel as hints (e.g., for scheduling, for future resource usage). In our case, the kernel has a priori no knowledge about the running application and the application relies on the kernel for efficient management of its mechanisms.

3 Case Study

In this section we describe a case study of the Hackbench micro benchmark. We chose Hackbench as a starting point for our work because of the following characteristics: (i) it

makes an intensive usage of IPC channels, (ii) it is massively multi-processes (iii) its code is simple to understand, and (iv) its execution yields non-trivial performance anomalies.

3.1 Hackbench

We study Hackbench¹, a micro-benchmark/stress test for the Linux scheduler. It was designed to stress the scheduler by spawning a large number of processes (or threads) doing numerous IPC operations (using either Unix domain sockets or pipes). In the remainder of this document we focus only on a configuration using processes and sockets with the default-settings (40000 processes).

The initial process (called the *master*) is in charge of managing the other Hackbench processes called workers.

The execution of Hackbench can be decomposed into three phases²:

Process creation: all workers are created by the master.

Synchronization phase: the master creates two sockets to provide synchronization: one to communicate from workers to master (socket 1), the other one from master to workers (socket 2). When a worker starts, it writes a byte into the socket 1 and then waits (using the `poll` syscall) for incoming data from the socket 2. When the master has created all the workers, it consumes every byte one by one (from socket 1). Once all the bytes are consumed, it writes one byte inside the socket 2, giving the “go” notification for all workers to start.

Reaping phase: the master waits for processes to finish before terminating.

This challenges the process scheduler due to the large number of processes competing on the same resource (*synchronization phase*).

3.2 Setup

Since we target multi-core architecture, we chose to pin (forcing a process to run on a set of cores) the master on the first core (core 0) and the workers on all other cores except this one. This kind of optimization is often used by experienced system programmers. Indeed, as the master is in charge of orchestrating the creation and the main execution phases of the worker processes, its reactivity is expected to have a strong impact on the application’s total execution time. Thus, it makes sense to provide it with a dedicated core, so that it does not compete for execution with the other processes.

We run the experiments on a 48-core Dell PowerEdge R815 server with 4x AMD Opteron 6344 chips (2,6 GHz, 12 cores), and 64 GB of RAM (DDR3 1600 MHz). We use the Ubuntu 12.04.4 operating system, with Linux kernel version 3.13. All experiments were run 60 times.

3.3 Observations

In this configuration, we observe a wide variation of the execution time with approximately 25% of the runs taking

¹<https://github.com/jlelli/rt-tests/blob/master/src/hackbench/hackbench.c>, [Online; accessed 10-June-2014]

²The initial Hackbench version contains a message exchange phase between workers after the synchronization phase. We chose to ignore this phase for simplicity.

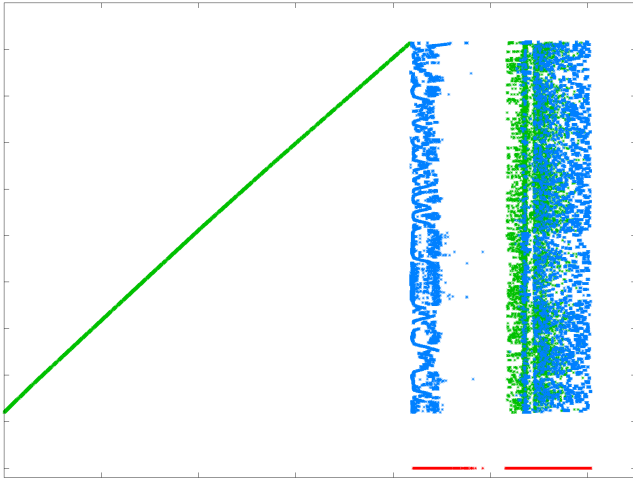


Figure 1: Process-view timeline for a long run

more than twice as long as the others (from 7.5 to 20 seconds). These variations are unexpected and surprising. Indeed, Hackbench is a deterministic micro-benchmark which does not depend on any input. It exhibits a “reproducible” workload and yet has highly varying execution times. Some can argue that this is mostly due to the non-determinism of the Linux scheduler. However, a twofold increase of the execution time is an uncommon behavior (at such a timescale) and, given that Hackbench uses 40000 concurrent and identical processes, we expect all CPUs to be always busy.

Our objective is to understand the root-cause of such variations and to provide solutions to guarantee a short execution time for all Hackbench runs.

4 Performance Troubleshooting Methodology

This section starts with Section 4.1, in which we present the methodology we employ to detect the root-cause of the Hackbench variations problem. At each step, we ask ourselves several questions that arise from the knowledge we have acquired previously. Then, we try to find a way to answer those questions by selecting metrics provided by the kernel and proposing resource-centric visualization techniques that enlighten us about the situation. Finally, with our understanding of the root-cause of the performance issue, in Section 4.2 we propose several solutions that can be used to avoid the slow Hackbench runs.

4.1 Methodology

Application Phases

As a starting point, we build a view called *process-view*. The process view is a timeline, where, on each line, we depict one process behaviour, using different colored dots marking an event.

Figure 1 is an example of such a view for an Hackbench run with a long execution time. The bottom line (with red dots) is the master line. One dot represents when the master enters a “blocking state” on the socket: Hackbench uses blocking read operations on the socket (i.e., the process reading on the

socket will be blocked and descheduled if there is no data to read). We chose to select the blocking events instead of every read event as the read event will happen 40000 times (once per worker) whereas a blocking is more unexpected. On top of this master line, we plot 40000 other lines, one per worker. A green dot represents a blocking of a worker when it tries to write inside the socket. This blocking appears when the socket is full. A blue dot is used to mark when a previously blocked worker is unblocked. We can observe that most of the workers are blocked just after their creation. Indeed, a small number of the first created workers will succeed in writing their byte inside the socket. For the other ones, the socket is full and they block on it. Secondly, some workers block more than once in the socket. More surprisingly, there exists an interval in the timeline of the master process during which there is no blocking. By comparing a short and a long Hackbench run, we conclude that the creation phase (from the first worker creation up to the last one) is constant across runs. Thus, the variation only comes from the synchronization phase.

Such a resource-centric view allowed us to detect several things that could not be or hardly be detected by another approach: (i) the constant time of the workers creation phase, (ii) the master blocking on an empty socket and (iii) the interval where the master is not blocked. The reason of master blocking is simple: when the master reads bytes from the socket, it reads faster than the awoken workers are capable of filling it back. Thus, the socket is empty and the master is forced to block on it, resulting in a descheduling. The void interval inside the master timeline can be the consequence of two things: either there is an interval where the master reads without being blocked, or there is a long phase where the master is not executing.

To answer the previous question about the cause of this long void interval inside the master timeline, we use another technique that allows us to know if a process is currently running or descheduled. Our approach looks for process states (e.g., running, blocked, descheduled), which is not done by code-centric approaches. Moreover, such information exists only at kernel level. A profiling tool that does not take advantage of the underlying operating system may not be able to fully understand some performance issues.

With the state information, we traced intervals for processes between the moment they are blocked and the moment they are scheduled back in Figure 2. A pink dot and a blue dot respectively show when the master is blocked and then descheduled on the empty socket and when it is unblocked (when there is new data on the socket and the master starts to run again on a CPU). We plot a green line between a blocking event and an unblocking event when this interval takes more than a given threshold. At the end of the synchronization phase, we can see that there is long interval where the master is blocked i.e., during which the master does not progress.

This is surprising: we chose to pin the master on its own CPU to avoid master descheduling. To understand the reason of these descheduling, we need to understand if the master is suspended for a long time because it waits a long time for new data inside the socket (and at this moment it will be marked

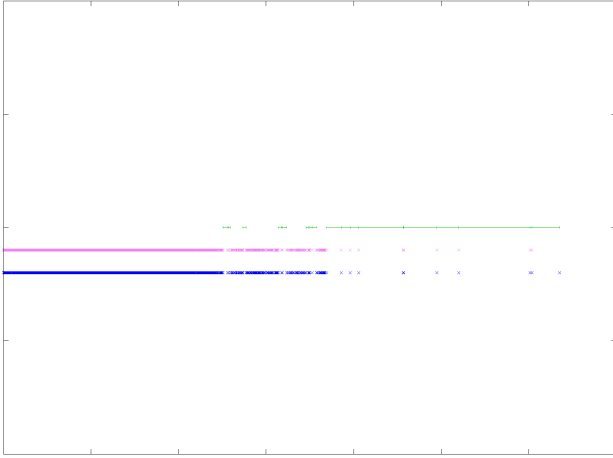


Figure 2: Timeline of master descheduling intervals

	Blocked-to-Runnable	Runnable-to-Scheduled
Average	98%	2%
Relative Standard Deviation	1%	0.06%

Table 1: Master blocking statistics for blocked – to – runnable and runnable – to – scheduled intervals

as runnable) or because it takes a long time for the scheduler to reschedule the master once it becomes runnable.

This time we again rely on kernel only information by capturing the timestamps for three kinds of events: when the master is blocked, when it is marked as runnable and when it is rescheduled. With these three events, we can detect easily if a process is waiting on a specific condition and thus understand why the condition takes so long to be satisfied or if a process is competing for a CPU to be scheduled (and then try to understand why there is a lack of available CPUs).

Table 1 shows the average time and the relative standard deviation of the time spent by the master in the *blocked-to-runnable* and in the *runnable-to-scheduled* time intervals. We observe that during 98% of the suspension time of the master, the master is blocked waiting for the condition (i.e., new data is inside the socket) to be satisfied. Thus, we now know that the master is not competing with other processes: our initial pinning configuration (master on first core, workers on all other cores except the first one) is effective. But we still do not understand why the master is waiting so long for new data: we saw previously that this long waiting interval appears at the end of the synchronization phase. At the end, there are less workers competing for CPUs as the ones which have written their byte are expected to be blocked inside the `poll` syscall, waiting for the “go” notification from the master.

We then focus on workers to know why the master waits so long for the arrival of new data. To do so, we add tracepoints

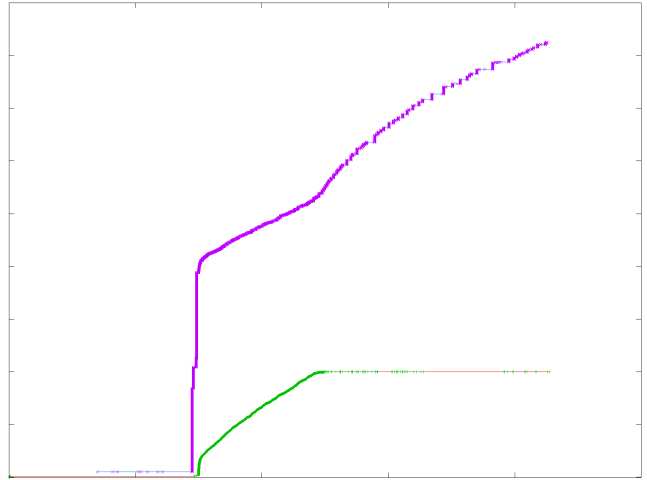


Figure 3: Timeline of runnable states of workers

directly inside the Linux kernel at the exact moment when a byte is written inside the socket and when a byte is read from the socket. We then build two timelines for the socket resource: when a byte is written inside this socket and when a byte is read from this socket.

We generate the timeline of the size of the socket buffer (not shown here due to a lack of space). At the beginning, there are the writes for the first workers, which succeed because the socket is not full yet. After that, a long interval without writes is seen: the master creates all workers but the workers can not write inside the socket as it is full. Then the master starts to free space on the socket and workers are awakened to write their byte. At the end, the frequency of reads and writes is low: the socket is not full, but the last workers nonetheless take time to write their byte inside the socket (we verified that a write happens just before a read by looking manually at timestamps).

We now know that the last workers take a very long time to write their byte. However, it should not be the case at the socket is not full and there are less workers, thus less competition for CPU time as workers having written their byte should be blocked on the `poll` syscall, waiting for the “go” notification from the master (after it has read all the workers bytes). This leads us to the following question: why are the last workers scheduled so late?

To understand why workers are scheduled so late, we chose to capture the event when a worker process is marked as runnable in addition to the reads of the master. With these timelines, we hope to see why the last workers are perturbed and by which processes.

We plot on Figure 3 a green dot and a purple dot respectively for a master read and for the moment where a worker is marked as runnable. The blue line joins two consecutive purple dots and the red line two consecutive green dots. Again, the height of timelines is here to visually separate consecutive points which are close in time.

The important thing to notice is the large amount of

```

ffffff8104f942 ttwu_do_wakeup
...
ffffff81118350 wait_on_page_bit_killable
ffffff81118406 __lock_page_or_retry
ffffff811187f7 filemap_fault
ffffff8111388b2 __do_fault
ffffff81113b8c7 handle_pte_fault
ffffff81113d5a9 handle_mm_fault
ffffff81164fce0 do_page_fault
ffffff81164c9f5 page_fault
    7f6272e089f0 __poll
        402ff2 create_worker

```

Figure 4: Call-chain of a runnable marking

runnable state marks at the end of the synchronization phase. Indeed, there should be only one mark after one master read: the master frees up one slot in the socket buffer and notifies a worker that was blocked waiting for a slot. But surprisingly, there are a lot of runnable marks that are not related to master reads. And each of these newly runnable workers will be scheduled, thus competing for CPU time. These workers will perturb the execution of the workers that still need to write their byte, and consequently will slow down the master read and the whole synchronization phase.

We do not expect other workers (workers that have already written inside the socket) to run during this period, they should be blocked inside the `poll` syscall. We now must understand why such workers are running: is it something related to the behaviour of the `poll` kernel mechanism? Or something else?

Taking advantage of the event tracing mechanism of the kernel, we observe the call-chains that lead a worker process to be set as runnable. Such call-chains are only available at kernel level and it is not common for profiling tools to use them.

A typical example of such a call-chain can be seen on figure 4. On the left column, the memory address where function is located. On right column, the function symbol name is provided. The call chain must be read bottom-up (e.g., `create_worker` calls `__poll`). Addresses starting with `ffffff` are inside the kernel. Thus the program enters the kernel via `page_fault`. Function `create_worker` at address `402ff2` is the Hackbench worker entry point. Finally, the function `__poll` at address `7f6272e089f0` is located inside the standard C library (i.e., `libc`).

From this call-chain we can understand that, after a worker has written its byte inside the socket, it calls the `poll` function which internally calls the `__poll` subroutine. The latter function triggers a minor page fault which will be handled by the kernel. We observed that all workers trigger this page fault³.

We believe that barely no experienced programmer could have intuitively guessed that such kinds of page faults could

³The page that triggers the page fault is a page of data used by the subroutine. Due to the copy-on-write kernel mechanism (which allows lazy/on-demand page duplication), it will generate a page fault to let the kernel duplicate the page.

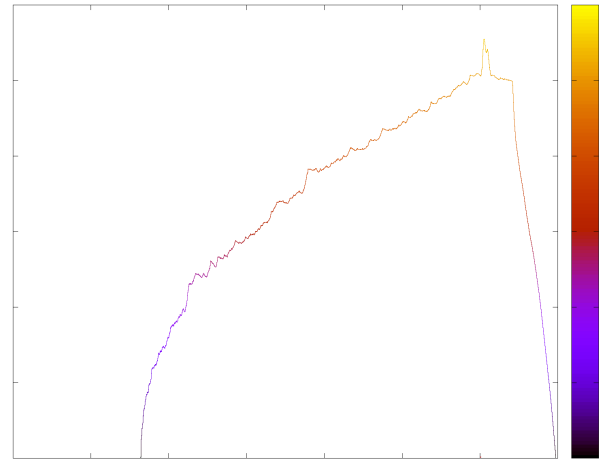


Figure 5: Timeline of the number of worker processes (running or blocked) within the page fault handler

be related to the initial performance problem. Yet, at this stage, there is not enough information to precisely understand how the page faults are related to the variations of the Hackbench execution times.

Lock Contention

The kernel function `__lock_page_or_retry` is called to acquire a lock on a shared kernel data structure called the page cache. The role of the page cache is to keep the data present in on-disk files (such as the `libc` shared library binary file) in memory as a performance optimization. In a massive multi-processes application, it is not uncommon to see lock contention: too many processes wait for the same lock, thus leading to process descheduling. And in our case, when the processes are rescheduled after finally obtaining the lock, they compete for CPU time with the workers waiting to write their byte inside the socket, causing performance degradation.

To validate our hypothesis about lock contention, we traced on Figure 5 the timeline of workers entering and leaving the page fault handler for a long run. When a worker enters the page fault, we increment the height of the curve, when a worker leaves the handler we decrement. The color gradient and the color of the curve is an indicator of the height.

In the case of a long Hackbench run, we observe that workers accumulate inside the handler without leaving it during up to 80% of the total execution time. These workers are serialized by the lock on the page cache and have to wait for the lock to be released, which is highly inefficient. On a short run (not shown in this report due to a lack of space), we observe that the curve goes up for a short time and then goes down, thus avoiding contention on the lock. This leads us to determine the key factor(s) that causes lock contention in the case of a long run: are there more workers requesting the lock at the same time or something else?

To evaluate the parallelism of our application across time, we build a view based on the scheduling events available from

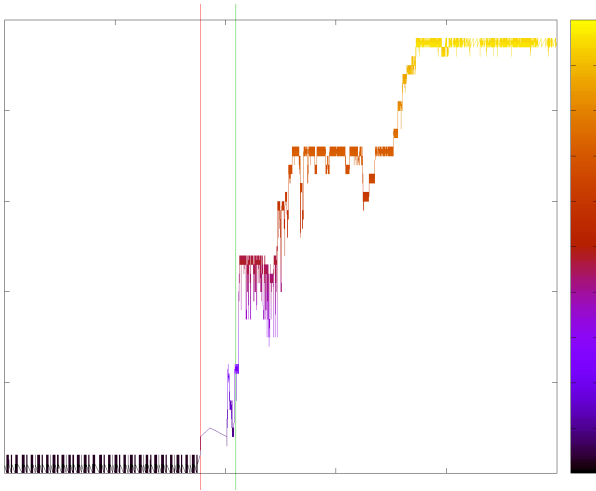


Figure 6: Timeline of the number of CPUs where workers are executing at the same time

the kernel. Such a view is highly valuable to observe the parallelism of the application *across* time and pinpoint where bottlenecks preventing good parallelism appear.

On Figure 6, the vertical red line indicates when the master reads the first byte from the socket (just after it has created all workers). The vertical green line indicates the first time the master blocks on the socket. The y-axis indicates the number of CPUs that are currently executing a Hackbench worker at a given point in time. The color gradient is a visual help for curve height. We zoomed on a 1 second interval, around the first master reads.

Before the first master read, we observe a few parallel processes: the master and the newly created worker, which performs its write and block on the socket. The time between the worker creation and the worker blocking on the full socket is short enough to avoid too many workers to run in parallel. In the interval between the first master read and the first master blocking, we can see up to 12 parallel execution flows. In a short run, we see at most 2 parallel execution flows.

From this, we can conclude that the more parallel execution flows there are on this interval, the more workers will trigger the `poll` page fault at the same time. And these workers will try to acquire the page cache lock at the same time, thus leading to contention. A lower degree of parallelism causes much less pressure on the lock, thus avoiding contention and reducing the average execution time.

Scheduling

A followup question that arises from the previous discovery is to understand why the workers are not always dispatched on the same number of CPUs when the master starts to read from the socket (in other words, why there is a greater degree of parallelism in the case of a long Hackbench run). We chose to look at the call-chains of the execution of `migrate_task` event. This event is raised when a task (i.e., an execution flow) waiting for execution on the runqueue (list of processes waiting for execution) of CPU A is migrated to another CPU B. Changing the runqueue of a process will always trigger

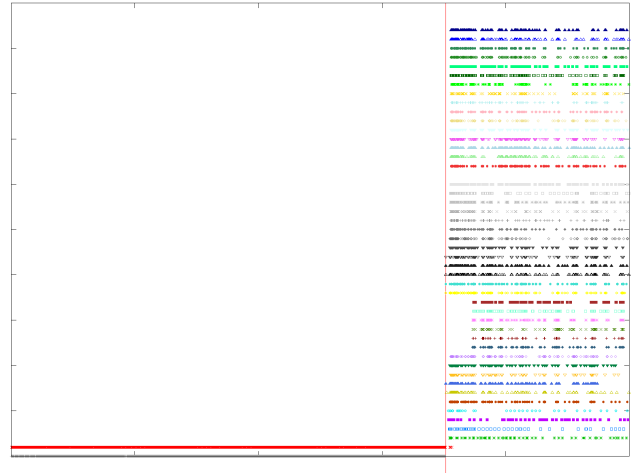


Figure 7: Timeline of migrations of workers per CPU

a `migrate_task`, thus tracking such events allows showing where processes go and why. In the case of Hackbench, processes are not dispatched on any CPU at creation but are executed on the CPU on which it has been created (the master CPU, the 0), until it forces itself into another CPU by pinning on any other CPU except the first one (as described in 3.2).

We observe from the call-chain of a migration procedure that the function to migrate a task is called by the function `loadbalance`. The load balancing mechanism is a scheduler mechanism trying to improve parallel execution by allowing a CPU which is not loaded to “steal” from the runqueue of a busy CPU runqueue in order to unload it. This kind of load balancing mechanism is run only on CPUs that are not idle. A CPU is put idle thanks to CPU frequency scaling: if a CPU has no work to do, the frequency of this core is lowered and the core goes to an idle state where some timer interrupts are disabled. This is done to save power. The load balancing mechanism uses interrupts to periodically run load balancing. Thus, if the CPU is idle, the load balancing will not happen. The final step and last question we ask ourselves is why in some cases CPUs are idle and sometimes not.

In Figure 7, we plot where the workers migrate. The vertical red line indicates the first master read. There is one line per CPU. We can see that, during the first master reads, there are migrations on 8 CPUs. In contrast, for a short run, we see only two CPUs where migrations happen. This confirms our hypothesis that the more migrations on different CPUs there are, the more contention there will be. However, to understand why CPUs are not idle (and perform migrations), we chose to list the different processes (not only Hackbench workers) running when the master does its first reads.

Several processes which are kernel helpers have been seen: `jbd2` (disk related), `kworker` (interrupts related), `rcu_sched` (kernel memory data structures related) and others. Such processes are regularly executed on the system in response to non-deterministic events: disk activity, timers, kernel garbage collector, etc. However, depending on which CPU is not idle

(activated by a kernel helper process) when the master starts reading from the socket, the migrations of the Hackbench workers will be done on a different number of CPUs.

We can now give a complete explanation of what happens in the case of a long Hackbench run. Many CPUs are activated when the master performs its first reads. Those first reads will mark workers as runnable. Load balancing on different CPUs is done, which will migrate workers on these CPUs and allow more workers to execute in parallel. Just after doing their writes, the workers will call `poll`, each causing a page fault. These page faults are all related to the same page inside the page cache, and accessing this page requires to take a lock. Thus, the workers are blocked waiting for the lock. When there are less workers that need to write their byte inside the socket (end of the synchronization phase), the contention on the lock will be reduced and workers will be unblocked. All these unblocked workers will be marked as runnable and will be scheduled on the different CPUs, just before being descheduled again waiting for the “go notification”. These workers will compete with the workers that have not yet written their byte inside the socket, thus slowing them down. As they take more time to write their byte, the master will wait longer between two reads, and the synchronization phase will take more time.

In the case of a short run, at the beginning there are less CPUs that execute a process, less workers will be migrated and less parallelism, less lock contention, less workers competing for CPU time and overall, a shorter synchronization phase.

This is surprising: in the case of Hackbench, better parallelism leads to performance anomalies. Besides, one would expect the scheduler to dispatch the worker processes on the CPUs at very early stages of the Hackbench execution. However, the scheduler is actually lazier: it waits for some CPU runqueues to become loaded before attempting to increase the degree of parallelism.

4.2 Solutions

Thanks to the insight that we have obtained through profiling, we propose three solutions that can be easily implemented to avoid the variations observed on Hackbench.

The first one is done by modifying the application. The solution is to add a spinning loop (a loop keeping the CPU busy without descheduling the process) between two master reads. This has the effect to wake up workers more slowly, letting them the time to do their page fault on the `poll`, avoiding the next process to wait on the lock of the page cache. Results show that this algorithm improves the average execution time by 13% and reduces the relative standard deviation of the execution times (metric for the variation) by 75%.

Another one is to enlarge the size of the socket. If the socket is large enough, there will be less workers that will do their page fault at the same time. However this solution can be not adapted to all situations as enlarging the default socket buffer size impacts all sockets and may lead to memory exhaustion.

The last solution is to statically link libraries at compile time. This allows workers to not trigger the problematic page

faults and avoid contention and thus long execution times. With this technique, the average execution is improved by 75% and the relative standard deviation by 99%. Such a great improvement is not only due to contention avoidance, but also to a decrease of the overall number of page faults.

5 Lessons Learned

Several techniques have been used to pinpoint performance issues. The process-view allows building timeline of processes and to spot unexpected behaviours, which can be unnoticed by profiling tools that use a statistical approach (i.e., tools that only provide metrics aggregated over the total execution time). Capturing descheduling intervals and the two different intervals (*blocked-to-runnable*, *runnable-to-scheduled*) is useful to discover if a process does useful work (is executed) or if it is either waiting on a condition (not runnable) or waiting for a free CPU. This knowledge enables performance optimizations either by giving more CPU time to the process or searching why the condition is not satisfied. Call-chains are invaluable to find the cause of an event. Evaluating lock contention by observing the number of processes trying to acquire a lock and the number of processes releasing the lock using a timeline enables focusing on a particular moment of the application where contention can happen. We do not think that statistical profilers can help to detect all lock contention patterns such as bursts. Looking for the number of parallel active execution flows is a good method to evaluate the speedup of a program and it can help pinpointing sub-optimal speedups (e.g., due to bad load balancing on the CPUs or hidden serialization points such as kernel locks) by looking at intervals of time where all expected CPUs are not fully utilized.

Our study of Hackbench highlights different patterns or inefficient interactions. The first one is related to the pinning of the master. The expected result was to give it more CPU time. However, the side-effect was to let it free the socket too quickly and then being blocked and descheduled. Even if this is not the root-cause of the variations, such a pattern is inefficient. Another pattern is to let the scheduler efficiently balance the processes over all the CPUs. The dispatching is highly sensible to how many CPUs are available (i.e., a process is running on the CPU). It is sometimes better to explicitly pin a process on a given CPU than to rely on the scheduler.

6 Conclusion and Future Works

Through the study of Hackbench, we have pinpointed different inefficient interactions (either explicit or implicit) between executions flows and resources. We used different methods to find the root-cause of such a problem that may be used, along with others, to automatically discover interactions performance bottlenecks in the context server-side applications.

We plan to generalize our approach by automatizing it in a profiling tool and analyzing real-world applications (e.g., the Apache web-server). We hope to find other metrics by studying such programs.

References

- [AMD, 2014] AMD. CodeXL - Powerful Debugging, Profiling & Analysis. <http://developer.amd.com/tools-and-sdks/opencv-zone/opencv-tools-sdks/codexl/>, 2014. [Online; accessed 01-June-2014].
- [de Rochefort, 2014] Xavier de Rochefort. Noyau Linux : À Propos Des Outils De Trace. In *Conférence en Parallélisme, Architecture et Système (ComPAS 2014)*, Neuchâtel, Switzerland, Avril 2014.
- [Fournier and Dagenais, 2010] Pierre-Marc Fournier and Michel R. Dagenais. Analyzing Blocking to Debug Performance Problems on Multi-Core Systems. *Operating Systems Review*, 44(2):77–87, April 2010.
- [Geimer *et al.*, 2008] Markus Geimer, Felix Wolf, Brian J. N. Wylie, Erika Ábrahám, Daniel Becker, and Bernd Mohr. The SCALASCA Performance Toolset Architecture. In *Proc. of the International Workshop on Scalable Tools for High-End Computing (STHEC)*, pages 51–65, Kos, Greece, June 2008.
- [Giraldeau *et al.*, 2011] Francis Giraldeau, Julien Desfossez, David Goulet, Michel R. Dagenais, and Mathieu Desnoyers. Recovering System Metrics from Kernel Trace. In *Linux Symposium*, pages 109–115, Ottawa, Canada, June 2011.
- [Intel, 2013] Intel. Intel VTune Amplifier XE 2013. <https://software.intel.com/en-us/intel-vtune-amplifier-xe>, 2013. [Online; accessed 01-June-2014].
- [Kalé and Sinha, 1993] L.V. Kalé and Amitabh Sinha. Projections: A Preliminary Performance Tool for Charm. In *Parallel Systems Fair, International Parallel Processing Symposium*, pages 108–114, Newport Beach, CA, April 1993. IEEE.
- [Lachaize *et al.*, 2012] Renaud Lachaize, Baptiste Lepers, and Vivien Quéma. MemProf: A Memory Profiler for NUMA Multicore Systems. In *USENIX Annual Technical Conference*, pages 53–64, Boston, MA, USA, June 2012. USENIX.
- [Pillet *et al.*, 1995] Vincent Pillet, Jesús Labarta, Toni Cortes, and Sergi Girona. Paraver: A Tool to Visualize and Analyze Parallel Code. In *Proceedings of WoTUG-18: Transputer and occam Developments*, volume 44, pages 17–31, Amsterdam, April 1995. IOS Press.
- [Wang *et al.*, 2013] Chengwei Wang, Soila P. Kavulya, Jiaqi Tan, Liting Hu, Mahendra Kutare, Mike Kasick, Karsten Schwan, Priya Narasimhan, and Rajeev Gandhi. Performance Troubleshooting in Data Centers: An Annotated Bibliography. *Operating Systems Review*, 47(3):50–62, January 2013.
- [Weaver, 2013] Vincent M. Weaver. Linux perf_event Features and Overhead. In *International Symposium on Performance Analysis of Systems and Software (ISPASS 2013)*, pages 80–87, Austin, TX, USA, April 2013. IEEE.
- [Wisniewski and Rosenburg, 2003] Robert W. Wisniewski and Bryan S. Rosenburg. Efficient, Unified, and Scalable Performance Monitoring for Multiprocessor Operating Systems. In *SuperComputing: Conference on High Performance Networking and Computing (SC 2003)*, pages 3–17, Phoenix, AZ, USA, November 2003. ACM.

Magistere report

Jérémy SEGUIN

Supervised by : Vincent DANJEAN

Abstract—Evaluation of Xeon Phi's communications means.

Keywords—Xeon Phi Intel MIC MPI

binary code executable by the Xeon Phi (-mmic for mpiicc compiler).

I. INTRODUCTION

The architectures of today's supercomputers are often prelude to tomorrow's public machines. We can note it with the appearance of multi cores processors in devices such as mobiles phones or personal computers.

The Xeon Phi, an Intel coprocessor available since early 2013, is announced as a basis element of supercomputers. It has many cores and a vectorial instruction set.

Xeon Phi is different from GPU accelerators because programs can be written like any application using parallel environment programming. It can be used as an accelerator or as a MPI cluster.

The objective here will be to evaluate the different means of communication of the Xeon Phi. We will restrict here to one Xeon Phi and a computer. However, means can not be all tested, only results of two of them will be presented here (MPI and socket).

II. THE XEON PHI

A. Presentation

The Xeon Phi is a new generation of accelerators that appeared in the year 2010. While more recent than GPU accelerators such as nVidia Tesla, though it is different by its architecture Intel MIC¹. This permits to execute code working on processor without having to rewrite it, contrary to actual GPU. Indeed, execute code on graphical card needs to rewrite it in its own language and use specific libraries.

With 60 cores at 1 Ghz and 4 threads per core (whether 240 threads), it is announced with a computing power of 1 Tflops². Moreover, vectorial instructions are available.

B. MPI programming models

We will take as convention "local" is the machine that possesses the Xeon Phi, "mic" is the Xeon Phi.

Coprocessor-only model :

As known as "native model". The program executes on Xeon Phi only. The code is the same as the one used for normal machine, only an option is specified at compilation to get

Symmetric model :

Program is divided in jobs using MPI. These jobs are executed in parallel on a list of hosts. Operation of these jobs is comparable to process. Communication between jobs is done using functions of the MPI library (Send, Receive, Broadcast, ...).

This model will be used in the experiments presented later in this paper.

Offload model :

A part of the calculations is offloaded either from local to mic (direct acceleration) or from the mic to local (reverse acceleration). Portions of code to offload are marked by pre-processor directives.

Example :

```
#pragma offload target(mic) in (n) \
      in (a:length(N)) out (b:length(N))
f (n, a, b);
```

This indicates that call to function f will be offloaded to the mic. The type of data (in, out or inout) used by the function and its size is specified. Others options can be specified for the data behaviour (allocation or not, ...). Programs thus coded must be compiled with specific options.

III. GRID'5000

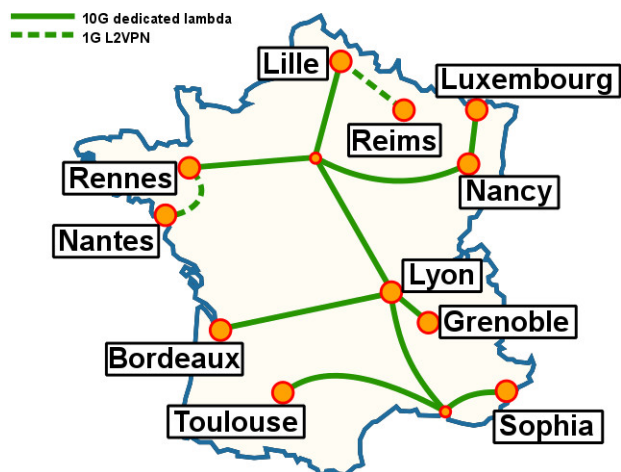
A. Presentation

Grid'5000 is a supercomputer network distributed on many sites. Each site provides hardware and software tools in order to conduct experiments simply, effectively and shared way. Sites includes clusters. For example, the Xeon Phi is accessible on Digitalis cluster of Grenoble site.

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see <https://www.grid5000.fr>).

¹Many Integrated Cores, x86 architecture

²FLoating Operations Per Second



In order to use a machine of the network, it is necessary to make a request (also called "submit") of job. This is done using *oarsub* command. Many options are available to precise the time needed for the job, the type of job (shared, reserved, deploy, ...), the start time, etc.

If nothing specifies otherwise, a reservation is shared. So several person can use the same resource together. Following experiences have been done on Grenoble site which possesses a Xeon Phi.

Here two examples of available tools :

chandler command to list running jobs and machines state.

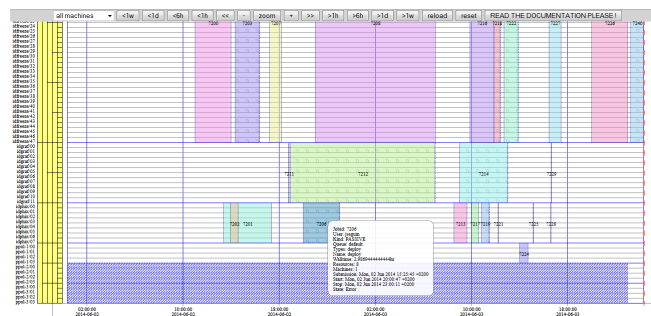
```
[jseguin@digitalis:~]$ chandler
160 resources, 56 used, by 2 job(s)
grimage-1: ██████████
grimage-2: ██████████
grimage-3: ██████████
grimage-4: ██████████
grimage-5: ██████████
grimage-6: ██████████
grimage-7: ██████████
grimage-8: ██████████
grimage-9: ██████████
grimage-10: ██████████
idfreeze: ██████████
idgraf: ██████████
idphix: ██████████
ppol-1: ██████████
ppol-2: ██████████
ppol-3: ██████████

Node state: █=Free █=Standby █=Suspected █=Absent █=Dead
Job kind: █=Exclusive job █=Shared job

idfreeze.grenoble.grid5000.fr
[7294] mettinger (exclusive), ends in 7h32m51s

idphix.grenoble.grid5000.fr
[7296] jseguin (shared), ends in 1h59m56s
```

Gantt permits to list jobs and machines state over time. At the left is the list of machines. Each coloured square represents a job. Colour is random and does not depends on the job. Detailed informations can be seen on mouse hover. The red bar represents the present moment (here at the left because further in time).



B. Deploy with kadeploy

kadeploy is one of the many software tools available in Grid'5000. This command permits to install a custom made system on a reserved machine and have the root access on it, very useful to install software or libraries not present in the base system.

However, a deployed system can not be shared, so only to use if necessary. For the following experiments using MPI, deploy was mandatory to have MPI libraries installed on the Xeon Phi.

IV. EXPERIMENTS

One program have been written to measure influences of communications on calculation. This programs is in two versions : one using MPI⁴ for communications, another using Sockets. The MPI version is designed for the symetric model, it needs to be launched with *mpirun*. The Socket version was written to compare results with the MPI version and be able to tell if the MPI layer adds time in communications.

These experiments have been done using a computer (local) connected to a Xeon Phi (mic).

There are 4 communications to measure : local -> local, mic -> mic, local -> mic and mic -> local. For each test, measures have been done a number of times to increase accuracy (here 32/64 times), Results are the average of these measures. Results are exported in csv⁵ file format.

Size of data are from 4B to 1BG, each time multiplying by two.

A. CommunicationInfluences

This program, written using MPI, must be launch with two jobs in parallel. Each job receive a rank. Rank 0 sends the data, does calculus and measures times while rank 1 only receive data. Here are the time calculated :

- t1 : time for calculus only
- t2 : time for transfert + wait
- tw : time for wait only
- ts : time for transfert only

³Images from grid5000.fr

⁴MPI (Message Passing Interface) is a message communication library available in C, C++ and Fortran

⁵Coma Separated Values

CommunicationInfluences structure

```

if (rank == 0) {
    //Calculus only
    for (i = 1; i <= count; i++) {
        gettimeofday (&tv1, &tz);
        calculation (0);
        gettimeofday (&tv2, &tz);
        t1 = getTime (tv1, tv2, 1);
    }

    for (j = size; j <= toSize; j*=2) {
        for (i = 1; i <= count; i++) {
            //Transfert only
            gettimeofday (&tv1, &tz);
            MPI_Isend (data, j/sizeof (int), MPI_INT, 1, 0,
                MPI_COMM_WORLD, &request);
            MPI_Wait (&request,&status);
            gettimeofday (&tv2, &tz);
            ts = getTime (tv1, tv2, 1);

            //Transfert + Calculus
            gettimeofday (&tv1, &tz);

            MPI_Isend (data, j/sizeof (int), MPI_INT, 1, 0,
                MPI_COMM_WORLD, &request);
            calculation (1);

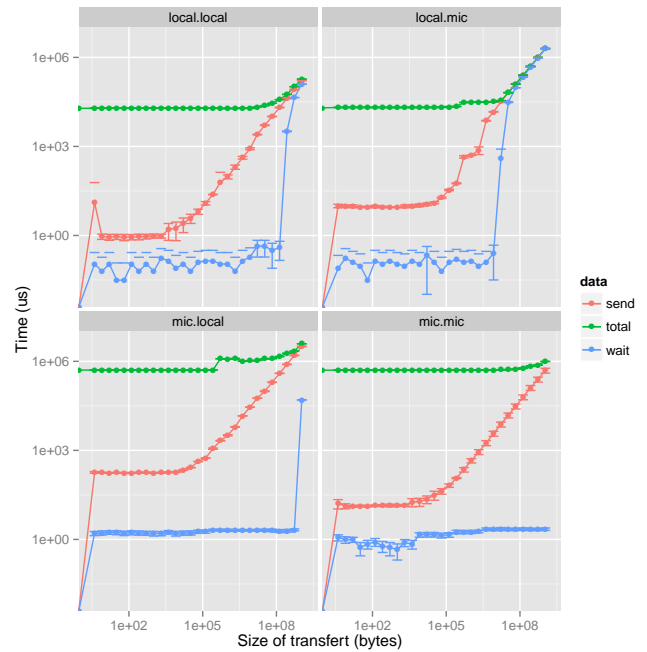
            gettimeofday (&tv3, &tz);
            MPI_Wait (&request,&status);
            gettimeofday (&tv4, &tz);

            gettimeofday (&tv2, &tz);

            t2 = getTime (tv1, tv2, 1);
            tw = getTime (tv3, tv4, 1);
        }
    }
}
else if (rank == 1) {
    for (...) {
        MPI_Recv (data, j/sizeof (int), MPI_INT, 0, 0,
            MPI_COMM_WORLD, &status);
    }
}

```

MPI_Isend is an asynchronous data send function. MPI_Wait must be use to wait for the end of the transfert. The argument 0 or 1 in the calculation function call is used to add or not calls to function MPI_Test periodically during calculus. This function used to test the completion of a transfer is used here to avoid transfer to "sleep" and be delayed.



At size 0, total time is calculus only time (no transfer), used as a reference. This is why send and wait time are null.

Total time is constant until send time begins too important. This is a proof that calculus and send are done in parallel and don't interfere each other. The additional time with big transfer is because transfer takes more time than calculus.

Wait time is also constant, another proof that transfer don't interfere with calculus. It only increases with big transfer because calculus ends first so more time is spent in the wait.

B. CommunicationInfluencesSocket

For this experience, MPI data transfer are replaced with sockets transfer. This is used to verify if MPI cost in time is important.

The program has two modes : server that only receives data and client that sends data, does calculus and measures times. Data sending are done in a specific thread to have asynchronous sends. Calculus and send are forced to be on the same CPU. For these results, tests were done on CPU 0 and CPU 1.

- t1 : time for calculus only
- t2 : time for transfert + wait

```

if (rank == CLIENT) {
    for (i = 1; i <= count; i++) {
        gettimeofday (&tv1, &tz);
        calculation ();
        gettimeofday (&tv2, &tz);
        t1 = getTime (tv1, tv2, 1);
    }

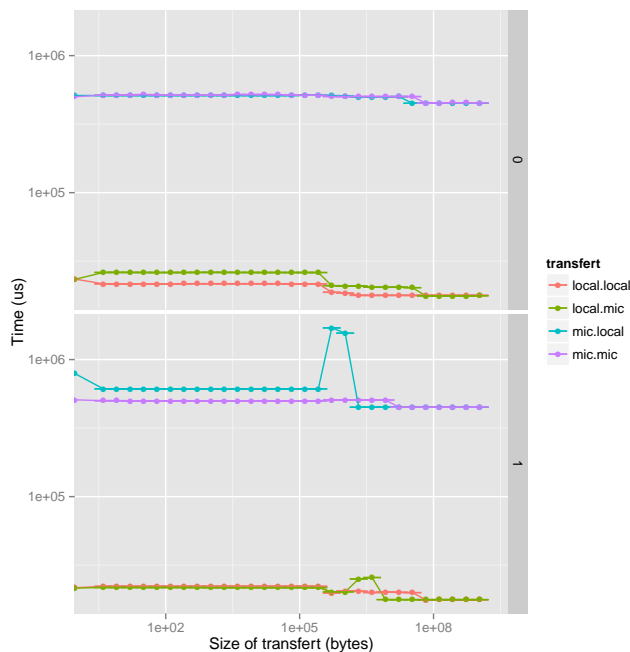
    for (j = size; j <= toSize; j*=2) {
        for (i = 1; i <= count; i++) {
            gettimeofday (&tv1, &tz);
            pthread_create(&thread, NULL,
                thread_write, NULL);
            calculation ();

```

```

pthread_join(thread, &retval) != 0)
gettimeofday (&tv2, &tz);
t2 = getTime (tv1, tv2, 1);
}
}
else if (rank == SERVER) {
while ( (nbData = read (sock_c, data, s)) > 0) {}
}

```



At size 0, time is still for calculus only.

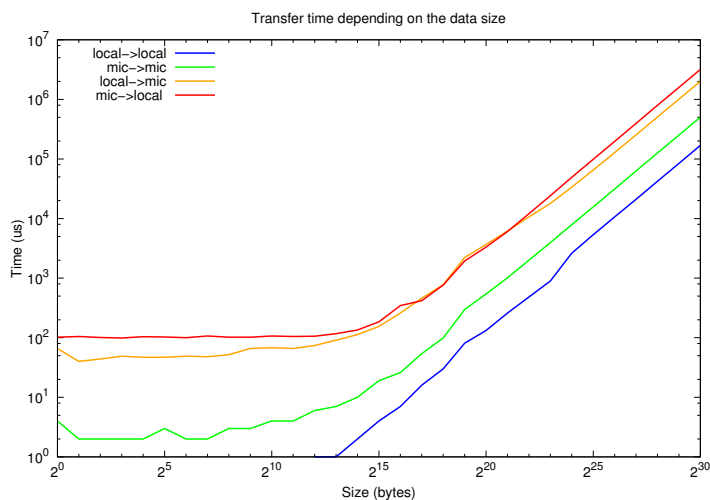
Times are almost constant for every transfer type and CPU. This proves that transfer don't interfere with calculus.

C. Other experiments

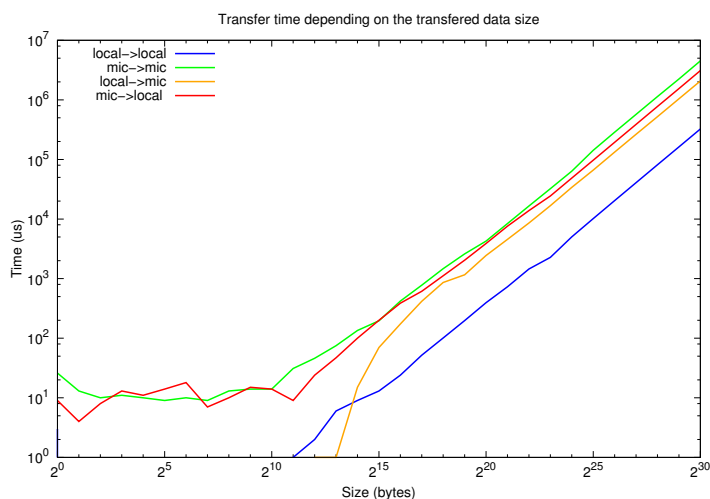
Others experiments were done before but not kept due to a lack of data or accuracy.

a) LatencyMeasurement: This program permits to measure time needed to transfer n bytes between 2 hosts. It can be compared to *ping* command. Size tested here are from 1B to 1GB with x2 step.

With MPI :



With sockets :



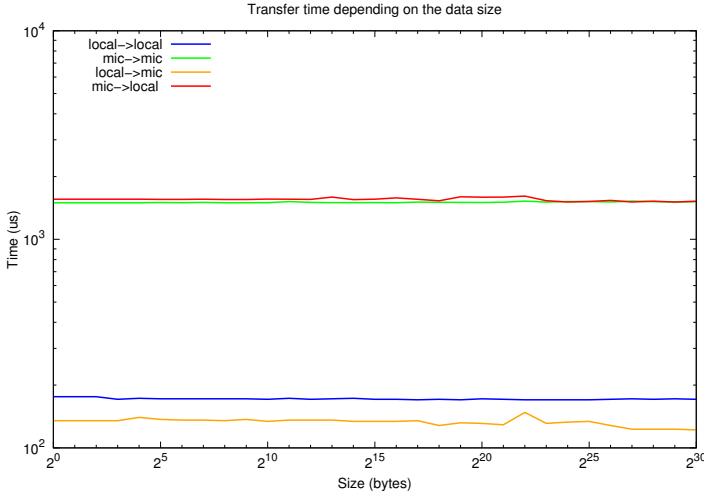
MPI and Socket version have both the same shape. We can see two parts in these graphics : a constant part (from 1B to 4-8KB) and a linear part (to 1GB).

In the constant part, transfers are faster with Sockets (except for mic->mic transfert). In the linear part, transfers to the exterior (mic->local and local->mic) are the same for both. However, for inter-hosts (mic->mic and local->local) are faster with MPI. The most remarkable is for mic->mic that takes 10 times longer with Socket.

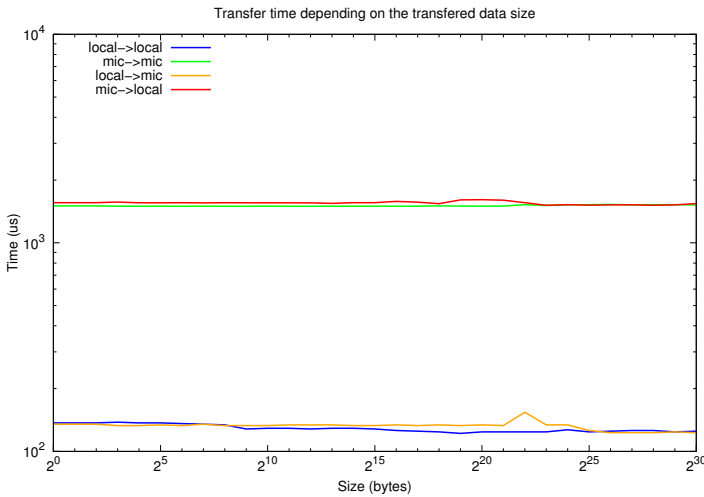
With these results, we can see that it is necessary to avoid as much as possible much inter-hosts communications as they are the slower. MPI has good results for sizes of data $\geq 8\text{KB}$ and Sockets are only good for non mic->mic transfers of short sizes ($< 8\text{KB}$).

b) CommunicationInfluences (the old version): This version of the program measured only the total time. This only measure is not enough to understand the behaviour with big data.

With MPI :

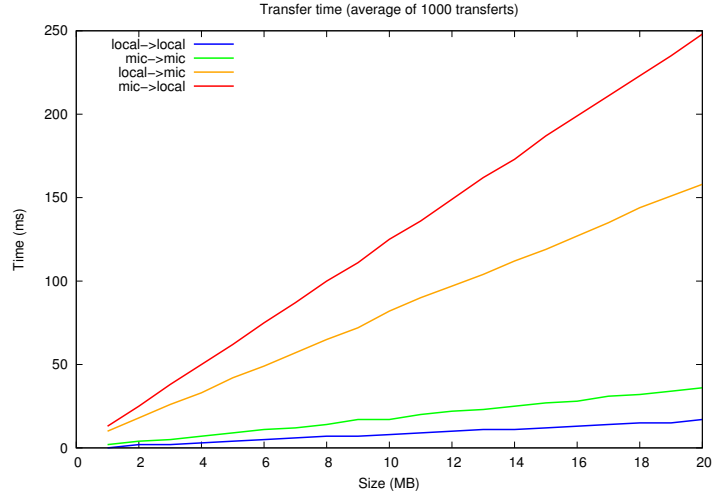


With sockets :



In both case, communications have negligible influences as the transfers times stay constant and even to base time.

c) *MatMultMPI*: The first program written. This program is a square matrix multiplication. A job called "root" generates two square matrix and sends it to each job that calculates a part of the result and sends it back to the root. Measured times are data generation, data sending, calculus, and result sent. These measure are not accurate because in seconds. So all measurements were done 1000 times.



The results tendencies are the same than in the linear part of LatencyMeasurement

d) *LatencyMeasurement*: First, this program were used to calculate latency for a 1B data transfer. Here is the average for 10^7 transfers.

local->local	mic->mic	local->mic	mic->local
0.2 μ s	2 μ s	1.7 μ s	50.3 μ s

local->local time is the best one but mic->local is the worst. This confirms the previous statements : it is necessary to avoid mic->local transfers. Average debit have been calculated from these results :

local->local	mic->mic	local->mic	mic->local
1228MiB/s	559MiB/s	123MiB/s	80MiB/s

V. CONCLUSION

Results of these experiments permit us to understand how to set the process to decrease communication time and the behaviour of data transfer using MPI and sockets in order to improve future programs using hybrid model.

CommunicationInfluences and CommunicationInfluences-Socket are the most accurate of the programs presented in this paper. An adaptation for other models and other communication means could be a good idea in order to compare the results. Moreover, only one Xeon Phi and one computer were used here, it could be useful to test others configurations.

It is necessary not to forget that experiments presented here were done in close environment, there were only one program running on a machine and a Xeon Phi, also close each other. Tests in disturbed environment (high latency, many running connections, ...) would be more realistic.

Such experiments are essentials for an understanding of the Xeon Phi communication specific. A little time spent testing could spare more time in future developments.

ACKNOWLEDGEMENTS

I wish to thank Vincent DANJEAN for providing this stage and helping me along it, Sébastien CURT and Pierre NEYRON for their help in my

research and expériences,
Inria for providing me new and interesting means of experimentation.

REFERENCES

- [1] www.grid5000.fr
- [2] software.intel.com
- [3] Intel® Xeon Phi™ Coprocessor System Software Developers Guide
June, 2013 - 163 pages
SKU# 328207-002EN
- [4] Jim JEFFERS & James REINDERS
Intel® Xeon Phi Coprocessor High-Performance Programming
2013, Published by Elsevier Inc - 430 pages
ISBN: 978-0-12-410414-3
- [5] Arnaud LEGRAND, CR CNRS, LIG/INRIA/Mescal
Vincent DANJEAN, MCF UJF, LIG/INRIA/Moais
HPC programming languages
October 2nd, 2013 - 197 pages

Hierarchy construction in large scale taxonomies

Moura Simon

Supervised by: Eric Gaussier

I understand what plagiarism entails and I declare that this report is my own, original work.
Name, date and signature:

Abstract

Classification is a core procedure in computer science and in others fields such as biology or statistics. The objective is to classify items (e.g., documents, images, videos) into a predefined set of categories. Several systems, like Wikipedia or the DMOZ repository, organize their data in a structured manner where the categories are organized in a hierarchical setting with parent-child relations among them. In this context, hierarchical classification methods aim to leverage this a-priori information in order to build scalable classification systems.

However the given hierarchy may not be well suited for automated classification as it may contain noise or be subject to personal preferences of the editors that designed it. A way to overcome this problem is to construct a new hierarchy starting from a flat setting or to alter the existing one.

In this article, we propose an heuristic algorithm to construct a hierarchy of classes that increase the speed of classification while minimizing the error of the process.

During this internship, we implemented this heuristic and tested it on a known data set (sampled from LSHTC challenge) in order to evaluate our results.

1 Introduction

The problem of identifying relevant categories from sets of objects and in which category an unknown object should belong is becoming more and more present in multiple scientific field.

In chemistry, we could classify molecules searching for shape patterns. In medicine, we could look for patterns in diseases. In a library we would like to classify books and so on.

A classical example in biology is the classification of a new species of animal. This kind of work could be useful to find the similarities between known and unknown species, get an

idea of possible parents (in the case of paleontology) or even get multiple information about specific animal.

In the following figure 1, we show an example of a hierarchy which could be used for that purpose.

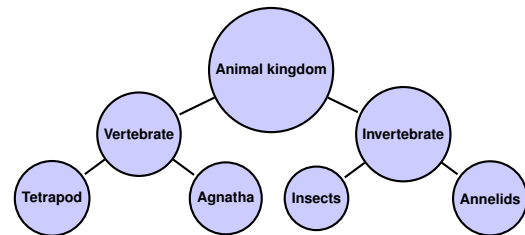


Figure 1: Animals taxonomy

One interesting point with this hierarchy is that if a new living being was discovered, we could classify it and obtain some useful information. For instance, if *dogs* were discovered, it would belong to tetrapod class, which is a sub-class of vertebrate class, which is a sub-sub-class of the animal kingdom root class. Thus, this structure help us to classify new animals regarding their similarities.

In computer science, classification is also widely used. For example, in image analysis we could look for patterns to recognize holiday, artistic or journalistic photos. In information retrieval, we would like to find the most relevant document regarding a large set of documents and so on.

In machine learning (ML), one objective is to design new methods for classification problems.

With the growth of available data, new problems are arising. Wikipedia has more than 4 million articles and more than 300.000 classes. Thus, the design of scalable applications, which are fast and accurate is a challenge.

Last subject will be the focus of the current paper.

This article will be divided as follow. Section 2 and 3 will explain how classification works and present the current approaches and ideas. Section 4 exposes the specific problem we are trying to tackle and how we managed it. Section 5 we will talk about our propositions for future work and conclude this paper.

2 Generalities on machine learning

In this section we present the notations used throughout this paper, the global idea of classification and supervised ML and finally we will formalize it in order to get a better understanding of what we are working on. Lastly, we will shortly discuss SVM models.

2.1 Notations

In classification the training set is typically denoted as $S = (x_i, y_i)_{i=1}^n$, where x_i is the feature vector (which represents the *attributes* of an object) of instance i and $y_i \subset C$ is the set of classes in which the instance belongs, where $C = c_1, \dots, c_k$.

w_i will represent a classifier for the class i .

2.2 Classification and ML

Machine learning is a broad subject but which can be divided in two main approaches :

- **Unsupervised** algorithms which operate on unlabeled data. The idea is to discover a structure in the data.
- **Supervised** ML algorithms which objective is to find a function which map a training set to their corresponding classes. Then this function is used to classify new unknown objects.

On this paper, we will focus on supervised machine learning and so, labeled datasets.

The general idea of supervised ML is the following : we consider a training set $S = (x_i, y_i)_{i=1}^n$ (e.g., a set of animals from living in the nature and their corresponding classes) and we try to discover a function $h(x_i) = y_i$ which generalize or map them the best. This function called a *classifier*.

Thus this classifier help us to determine in which class a new object belongs.

There are two common ways to use these classifiers : flat or hierarchical approaches.

Flat classification

For flat classification, we consider all the classes at once and decide directly where to put the object. With this method only one decision need to be taken but can be hard to take.

The classifier that we learnt from the training set will give the distance between each class and the object to classify. Thus, we will be able to decide where it belongs.

Hierarchical classification

On the other hand, hierarchical classification divide the problem of classification into several smaller problems. The idea is to use classifiers which consider less and less object, and thus are more and more accurate.

Then, the problem can be seen as a tree search. All nodes are classifiers and leaves are final classes.

As we can see in the figure 2, if we start from the root, we first have to take a decision between two sets using a first classifier. Then we iterate on lower levels until we reach a

leaf, which is the final class.

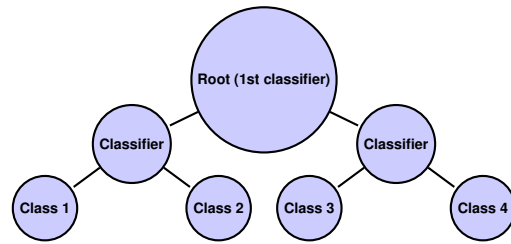


Figure 2: Hierarchical clustering

Accuracy vs Speed

When a method have been decided, hierarchical or flat classification, and a decision model have been established (for example SVM), the last step is to check the quality of the method using a *test set* (i.e. $S_{test} = (x_i, y_i)_{i=1}^n$). This dataset contains known mapping between objects and classes.

The errors made during the test phase is called the generalization error. It is on this error that we are going to work to build an accurate and fast hierarchy for classification algorithm.

Both of the previous classification methods have there advantages and drawbacks.

Three criterion are relevant to decide for the quality of a method: the time spent for pre-processing (such as hierarchy construction), the classification speed and finally, the accuracy.

Usually, we are looking for the best trade-off between these three measurement tools.

2.3 Support Vector Machines (SVM)

There is a lot of algorithms that aim to determine classifiers. On this paper, we are going to focus on SVM which are quite recent supervised ML models. It allows to recognize accurately patterns in sets of data.

It is based on two main ideas :

1. *Kernel trick* Consider an higher space than the original one which allow to separate the objects in a different way. Then the boundary decision will be an hyperplane of this new space. In the remainder of this paper, we will note w these boundaries.

The following picture illustrate the kernel trick. We can see that it would be hard to find an accurate frontier separating the data in a $2d$ space while it easier in three a $3d$ space : we just need to find an hyperplane.

2. Find the *maximum-margin* between objects : find the hyperplane which divide the objects in two sets with the highest distance possible with the hyperplane.

Then, the closest vectors of the hyperplane are called *support vectors*.

We can observe the principle of the maximum margin for a specific set on the image 4.

$$\epsilon(g_f) \leq \frac{1}{m} \sum_{i=1}^m A(g_f(x^{(i)}, y^{(i)})) + 2 * R_m(G_{F_B}, S) + 3\sqrt{\frac{\ln(2/\delta)}{2m}} \quad (1)$$

$$R_m(G_{F_B}, S) \leq \frac{2}{m} \mathbb{E} \left[\sup_{\|W\|_{\mathbb{H}}} \sum_{(v, v') \in V^2, v' \in D(v)} \|w_v - w_{v'}\| \sum_{i: c(f, x^{(i)}, y^{(i)}) = (v, v')} \sigma_i \phi(x^{(i)}) \right]_{\mathbb{H}} \quad (2)$$

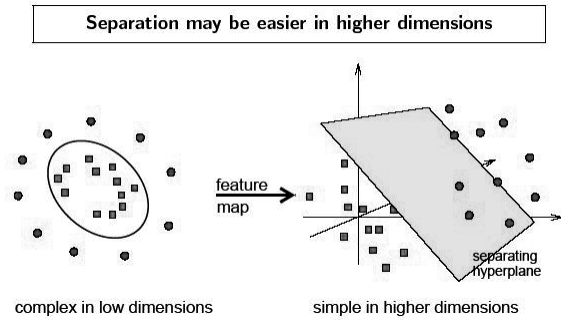


Figure 3: Kernel trick. (source: <http://www.dtreg.com/svm.htm>)

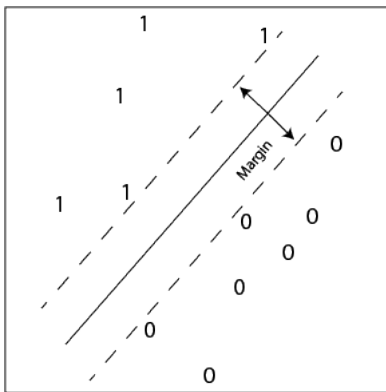


Figure 4: Max-margin (source: <http://www.sussex.ac.uk/Users/christ/crs/ml>)

3 State of the art and interesting ideas

When it comes to build a hierarchy, there is two main approaches :

- **Top-down** : we start considering the whole set of classes, train a classifier to divide it, iterate the process on both subsets and iterate until we have one class left per set.
- **Bottom-up** : we begin from leaf with one class per set and gather them in an upper levels until only one set left.

3.1 Common approach

[Bengio *et al.*, 2010] proposed an efficient solution to build a hierarchy reducing the generalization error.

The article is based on a top-down strategy and on two main ideas :

- For each node, they make a prediction on the subset of labels to be considered by its children;
- They used a method to reduce the feature space involved in the training in order to boost the learning speed and optimize the overall generalization error.

Both ideas have been re-used and improved in the following papers [Gao and Koller, 2011], [Deng *et al.*, 2011] and [Yang and Tsang, 2012].

3.2 Interesting improvement

Among all articles cited in last section, one of them had an innovative idea to handle the generalization error.

In fact, [Gao and Koller, 2011] proposed a method which aims to reduce the ambiguity while building a hierarchy in a top-down approach.

Basically, the idea consists in pushing back the ambiguous decisions and decide about them later.

As we can observe in 5 for the first node classifier, the boundary decision for classes 3 and 4 are not obvious. Thus, instead of cutting down the problem in two, we have to deal with an higher number of classes. It may increase the complexity but it is a trade for a better accuracy.

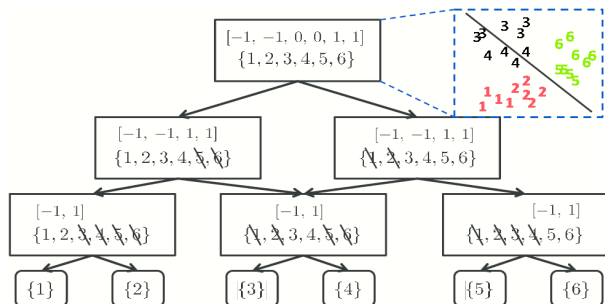


Figure 5: Image from [Gao and Koller, 2011]

4 Our contribution

In this section, we expose and formalize the proposed heuristic for the construction of an accurate and fast classification hierarchy based on recent results obtained by the AMA team.

$$\sum_{(v,v') \in D(\text{root}), v' \in D(v)} \|w_v^{(1)} - w_{v'}^{(1)}\|_2^2 + \sum_{v \in Lv_1(v)} \sum_{(v_i, v_j) \in D(v), v_i \neq v_j} \|w_{v_i}^{(1)} - w_{v_j}^{(1)}\|_2^2 \quad (3)$$

4.1 Problem Formalization

In a recent paper [Babbar *et al.*, 2013] published by the AMA team, a theorem have been proven giving a formal upper bound on error generalization for a given hierarchy.

The generalization error is the one made during classification of new objects using a predefined model.

Based on this proof, the generalization error, $\epsilon(g_f)$ (1), is upper bounded by three terms.

Where $D(v)$ denotes the daughters of node v , w_v is the classifier for the node v and V is the set containing all the nodes of the hierarchy, A is a Lipschitz function, $0 < \delta < 1$ and m is the number of objects in the training set.

The first and the third part of the bound are empirical errors, we are not working on them.

Thus, our efforts will be focused on the second term, 2. This term can itself be divided in two. The second one,

$$\left\| \sum_{i: c(f, x^{(i)}, y^{(i)}) = (v, v')} \sigma_i \phi(x^{(i)}) \right\|_{\mathbb{H}}$$

is linked to the input and fixed in advance. On the other hand, the first term usable for an optimization.

Finally, we are going to focus our interest on minimizing the following value while building the hierarchy :

$$\sum_{(v,v') \in V^2, v' \in D(v)} \|w_v - w_{v'}\| \quad (4)$$

4.2 Choices & hypothesis

Bottom-Up VS Top-Down choice

In regards to the equation 4 and considering that we sum on the daughters, the first intuition would be a top-down approach. Nevertheless, we spent some time thinking about this approach but it was computationally too costly and turned out to be an NP-complete problem.

Thus, we decided to use a bottom-up approach with a hierarchical agglomerative clustering method.

These methods are often used for such problems. The idea is to group up close elements step by step.

Hypothesis

At this point, we still need some hypothesis in order to use the formula 4 for hierarchy construction.

Starting from a flat hierarchy as in 6, the only parent for all the classes is the root. Thus, the formulas 4 will only be about the first flat level.

On the opposite, for upper level, we need to minimize the formula 3, which involve both, the current level and all the daughters in the lower structure. The figure 7 illustrate the idea and all the nodes on which we need to sum.

In order to reduce the computation time, we make a first hypothesis for levels upper than bottom most one. Regarding the formula 3 and considering that there is less sets at level $i+1$ than at level i , we admit that $A \ll B$ then $\min(A, B) =$

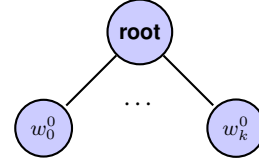


Figure 6: Level 0 of the construction

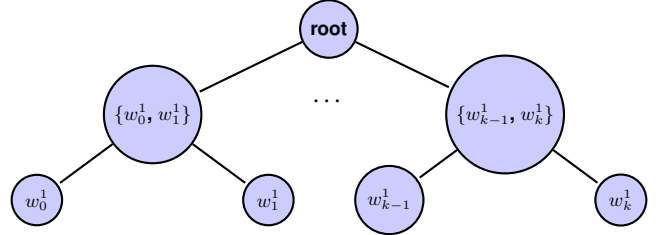


Figure 7: Level 1 of the construction

$\min(B)$. Where A represents the first term of the sum and B represent the second one.

The second hypothesis is that if two sets of classifiers are close in space i , then they will be close in space $i+1$.

With this two hypothesis, we are now able to create an heuristic. However, we still need a tool, in order to compare two sets of classifiers with a different cardinal.

Comparing sets of classifiers

In order to represent the distance between classifiers (which are vectors), we used a **similarity matrix**. A similarity matrix S is a matrix where $s_{(i,j)}$ represents the distance or the similarity between the object i and the object j .

Since we are going to group up some classifiers (which are vectors that represent a boundary decision between classes), we need a notion of distance to compare set of different size. For that purpose, we will use the *complete link* which is defined as follow in the book [Amini and Gaussier, 2013]:

$$\text{dist}(G_k, G_l) = \max_{d \in G_k, d' \in G_l} \text{dist}(w, w') \quad (5)$$

Where w and w' are classifiers in the set k and the set l . The idea behind this choice is to group two sets of classifiers in which all elements are close. Elements are going to be grouped only if they are not only close in average.

Dendrogram representation

We decided to group the classifiers two by two and represent them as dendrograms. A dendrogram is a representation of a binary tree with a notion of distance between the nodes. As we can see on the next picture 8, the further a node is from another one, the further they are in term of distance.

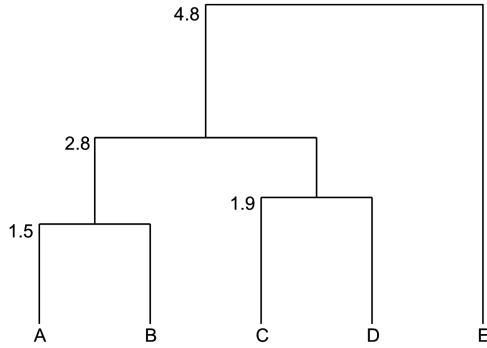


Figure 8: Dendrogram example

4.3 Proposed algorithm

With the help of the previous tools and based on decisions made earlier we proposed a new algorithm. In order to build it, we worked on classifiers instead of classes since the theorem in [Babbar *et al.*, 2013] give us terms to optimize. Building a hierarchy using classifiers was part of the challenge of this internship as we did not know if the bound was exploitable.

Proposed algorithm

1. For each class of the training set, train a classifier in one-vs-rest;
2. Compute a similarity matrix between the freshly trained classifiers;
3. Look for the two closest sets of classifiers w_i in the similarity matrix using the complete link distance;
4. Replace this two classifiers by the union in a same set considered as the parent and update similarity matrix;
5. Go to step 2. until there is only one set left. At his point we get a dendrogram representation based on classifiers.
6. Cut the dendrogram at an "arbitrary" height to keep only p sets.
7. Go to step 1. and learn classifiers considering the p sets until. Here, if a set contain more than one class, we consider them as a flat problem and apply *one VS rest* algorithm.

Highlights

1 vs rest is a usual method to compute classifiers while using SVM. The idea is the following. For each classes, compute a classifier considering the elements of all classes as a whole entity. Then, the hyperplane we obtain give an information about how to differentiate this specific class among all others.

For **step 6** we decided to *cut* the dendrogram at \sqrt{n} subsets. Where n is the total number of classes at start.

The cut down in the dendrogram in **step 7** is a key point for accuracy. The idea is to compute a new classifier using

a flat approach for classes which are the closest. As we are considering *close* classes, these classifiers are more precise to differentiate one class from another.

4.4 Complexity analysis

We can analyze each steps of the proposed algorithm and the classification of new documents considering that we have n classes and k features.

Hierarchy construction

1. Training classifiers for all classes in one-vs-rest is known to be a $O(k*n^2)$ problem [Chang and Lin, 2011];
2. Build dendrogram consists in two parts :
 - (a) Compute a similarity matrix between the freshly trained classifiers : we need to compare each classifier (one per class) to all the others. Thus the complexity will be : $O(n^2)$;
 - (b) Update the similarity matrix grouping the two closest sets w_i using the complete link distance, and do it 1 time for each grouping. The complexity of this part will be $O(n)$;

Classification of unknown object

In the worst case, if the dendrogram is totally unbalanced, the classification is not faster than the flat method. Only few objects will need to be checked once. Thus, classification time will be close to $O(n)$.

On the other hand, if the dendrogram is well balanced and the classification time will be closer to $O(\log(n))$, as the one of the binary search problem.

4.5 Training sets

In order to test our algorithm, we worked with two datasets which represent written documents with their corresponding classes. The features are the *tf-idf* (term frequencyinverse document frequency).

- LSTHC training data set. It can be found at <http://lshtc.iit.demokritos.gr/> and contain a training set and a test set of 1000 classes;
- A smaller dataset of documents with 64 classes and 50000 features, which is a sample of the last one.

4.6 Implementation

Implementation of an algorithm which use the bound of paper [Babbar *et al.*, 2013] was the second aim of this internship. This part have been divided in two phases :

1. Build a hierarchy;
2. Implement hierarchical classification for testing phase.

Choices

We used Python as programming language as it is widely used in ML for its simplicity to manipulate array and list. Thus, a large choice of library dedicated to learning are available.

We particularly used *scikit*, which is an open-source ML library. It allows to learn classifiers using a wide range of algorithm (including SVM), and much more.

Algorithm implementation

The algorithm discussed in 4.3 have been implemented in two parts.

First, parse and sort data in order to build dendrogram. Then build the hierarchy and store it for further computation.

Secondly, cut off in the dendrogram and consider subsets as a flat problem. Then, learn classifiers for new nodes and flat problems.

Testing phase

When it comes to testing phase, we had all needed classifiers yet. Thus, the idea was to evaluate all instance of the testing dataset regarding the current classifier until we meet a leaf.

This part needed to be done from scratch as no tool does it on generic data.

4.7 Results

In the following section, we analyze the results we obtained on the 64 classes training set.

Dendrograms

On the following figure 9 we can observe the results on a dataset of containing 64 classes and 49978 features for the dendrogram construction.

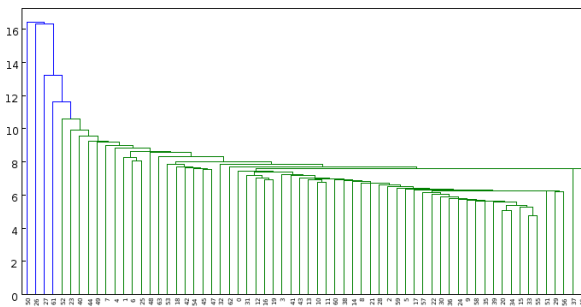


Figure 9: Dendrogram representation with 64 classes and complete link distance

We observe that the dendrogram is really unbalanced. These results are linked to the type of data we used. This is one of the worst case which could happen in term of process speed : we will need to compute classifiers twice in 57 cases.

The below picture 10 illustrate what we obtained at the step 6 of the algorithm after the *cut* in the dendrogram.

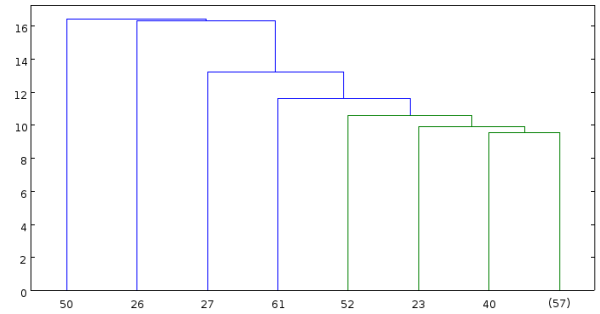


Figure 10: Dendrogram 8 classes (after dendrogram cut)

Accuracy and running time

In order to test our implementation, we used another sample from the same dataset. The purpose was to classify 506 new unknown objects in one of the 64 original classes.

Accuracy of SVM on flat hierarchy was 76.28% while the accuracy of our hierarchy implementation is 75.09%. The results are worse than flat SVM approach, but yet encouraging.

On the other hand, **running** time of both implementations were about 1.5s. These results are not relevant and hard to compare because of the small size of the test dataset.

We can note an important point here : the hierarchy construction takes time (about 20s) but it can be pre-computed for a given set of classes.

5 Conclusion and future work

This internship show some encouraging results and open some interesting perspectives for future work in the same subject field.

5.1 Apply algorithm on larger dataset

It would be interesting to test the proposed algorithm on a larger dataset for both : running time and precision checks. We could use the data from the DMOZ project which is a representation of the Internet.

5.2 Hierarchical classification library for *scikit*

Hierarchical classification is a widely used tool in ML. However, there is no tool yet in the *scikit-learn* library which handle this problem in a generic way. Even though it can be a substantial work to do, it would be interesting to implement a generic tool in order to fasten the future works.

5.3 Compare results with another approach

We could use another well known clustering algorithms to group up our classes and build the hierarchy. For example, the current approach could be compared with *kmeans* + + (an improved version of the *kmeans* algorithm).

In that case, if we start with n object, we could first pick \sqrt{n} centroids considering the euclidean distance. These

centroids would represent the first level of the hierarchy. Then we would apply the same principle on first level we just obtained which contain $\sqrt{\sqrt{n}}$ object to cluster. And so on until we have only one centroid where all objects are grouped : it would be the root.

5.4 Top-Down approach

As explained before, top-down approach are often used to avoid one-vs-rest algorithm in order to speed up the process of hierarchy construction. Thus, it would be interesting to look up for a new algorithm which allow both, a top-down approach and the use of the bound while building the hierarchy.

The idea of prediction on children developed in [Bengio *et al.*, 2010] could be useful as it would give us information the needed information about the lower tree level.

5.5 Conclusion

In this paper, we have presented a set of tools used in ML. We have seen that hierarchical SVM classification can be an efficient learning method, yet committing generalization error. We proposed an algorithm to minimize this error lowering an upper bound and the time spent for classification. After implementing this algorithm, we finally tested it on a small dataset. The results obtained show that further work in the same direction could theoretically lead to some improvements for classification speed, and to a lesser extent, for accuracy.

Acknowledgments

I would like to thank Eric Gaussier which allowed me to do my master internship and my magistere in the AMA team and helped all the way. Ioannis Partalas and Rohit Babbar for their patience and all the time they spent explaining me machine learning basis.

References

- [Amini and Gaussier, 2013] Massih-Reza Amini and Éric Gaussier. *Recherche d'Information - applications, modèles et algorithmes*. Eyrolles, 2013.
- [Babbar *et al.*, 2013] Rohit Babbar, Ioannis Partalas, Eric Gaussier, and Massih-Reza Amini. On flat versus hierarchical classification in large-scale taxonomies. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 1824–1832. Curran Associates, Inc., 2013.
- [Bengio *et al.*, 2010] Samy Bengio, Jason Weston, and David Grangier. Label embedding trees for large multi-class tasks. In J.D. Lafferty, C.K.I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 163–171. Curran Associates, Inc., 2010.
- [Chang and Lin, 2011] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [Deng *et al.*, 2011] Jia Deng, Sanjeev Satheesh, Alex Berg, and Li Fei-Fei. Fast and balanced: Efficient label tree learning for large scale object recognition. In *Proceedings of the Neural Information Processing Systems (NIPS)*, 2011.
- [Gao and Koller, 2011] Tianshi Gao and Daphne Koller. Discriminative learning of relaxed hierarchy for large-scale visual recognition. In *Proceedings of the 2011 International Conference on Computer Vision, ICCV '11*, pages 2072–2079, Washington, DC, USA, 2011. IEEE Computer Society.
- [Yang and Tsang, 2012] Jian-Bo Yang and Ivor W. Tsang. Hierarchical maximum margin learning for multi-class classification. *CoRR*, abs/1202.3770, 2012.

Comparison of geometrical and model based retargeting methods for imitation games using an RGB-D sensor

Maxime Portaz

Etienne Balit

Dominique Vaufreydaz

Abstract

Humanoid robots allow to imagine many applications for Human-Robot Interaction. Humanoid robots can be used as therapist tools in the treatment of social disorders. Imitation games are exploited to exercise social skills of children with autism spectrum disorder and evaluate their progression. When using a robot as training partners, the robot needs to perceive moves either to imitate or to validate mirror imitation from the user. The NAO robot has been chosen because it is well accepted by autistic children, thanks to its size, cuteness and facial neutrality. As a humanoid robot, NAO can perform a lot of human movements. To track the child skeleton, a RGB-D sensor is added to the setup.

This paper investigates different ways to compute the joint configuration of the NAO robot that best matches a tracked skeleton. A first method directly computes the joint configuration on the skeleton and applies it to NAO. A second approach uses inverse kinematics to find the NAO joint configuration. NAOqi and iTaSC inverse kinematics solvers are examined.

1 Introduction

In the context of serious games for child with Autism Spectrum Disorders (ASD), we want to create an imitation game enabling child to enhance their social skills. We need to measure their progresses, thus we have to maintain interaction for a while in a fluid and natural way.

Robots have shown to be efficient partners in such tasks [1]. With companion robots and moreover with humanoid robots, Human-Robot Interaction is of importance. As claimed by [6] and [8], one challenge concerns social competences of robots. While interacting with humans, they must perceive all social cues and act accordingly to enhance quality and pertinence of the interaction. For our serious game, looking at literature, we cannot envision a Wizard of Oz [3] experiment or a robot with staged behaviors [15].

The choice was made to focus on an open imitation game with a NAO robot. Due to its design and its size, it is a good choice to work with children. NAO can easily imitate a lot of human movements [4]. As a bipedal robot, NAO can fall, and it's a limitation for some applications. Our imitation game requires capture of behaviors from a human and control of the robot. Human skeleton is tracked using an RGB-D sensor. With this skeleton, articulation positions are given in 3D space. A conversion from the skeleton space to NAO moves, i.e. to control commands for the robot, must be computed.

In this paper, we will compare 3 different approaches to fulfill this task. The first one is to calculate angles for each articulation to control the robot. The two others use Inverse Kinematics (IK) to match as close as possible the robot stature to the human skeleton tracked using the RGB-D sensor. The first implementation uses NAOqi IK, the standard IK solver from the NAO SDK NAOqi. The second use Blender as an Integrated Development Environment for the iTaSC solver [12] for its ease of use and good performances.

Section II presents general context and motivation of these researches. The following section introduces related work and depicts our experimental setup. Section IV and V show joint configuration retargeting, and then model based retargeting, with NAOqi IK and iTaSC.

2 Serious game and autism

Serious games are entertaining activities with hidden goals like education, frailty evaluation or reeducation for elderly or people with disabilities [10], etc. In Human-Robot Interaction, games are interesting because they allow keeping people's interest without constraint. Particularly, children need to be focused, and a game seems to be the better way.

Children with Autistic Spectrum Disorders (ASD) have to be treated with effective interventions like behavioral therapies. Psychologist or therapist use different games and activities to improve child social skills. One of the most interesting games with these children is imitation game [2]. Usually done with the therapist, some games can be played with robots [11] as partners. Robots are often loved by children, and with autistic

children this is even more important. A human face is really expressive and human facial features can overwhelm those with an autism disorder [1]. In the case of an imitation game, a humanoid robot is adequate because of its likeness to a child.

The NAO humanoid robot has successfully been used in similar applications for these reasons [13]. NAO is a small humanoid robot with 25 degrees of freedom, 57 centimeters tall. He is equipped with two cameras, four microphones to compute sound azimuth and several bumpers. NAO can film people and do face recognition and tracking. We only use face tracking during interaction, due to a bad quality of the face recognition. There is the same issue with microphones. We have sound localization, but speech recognition is really difficult to use even more for children with ASD in a noisy room.

Another aspect of our work concerns the body tracking. Unlike other approaches, we cannot envision specific suit or motion capture techniques with autistic children. Using the embedded camera from the NAO will not provide body tracking. In imitation game, NAO may turn his head and not see the gamer anymore. For this reason, mounting an RGB-D sensor on the NAO like in [7] is not an option. Thus, we compute skeleton using an RGB-D sensor overhanging the robot. The setup is depicted in Fig. 1.



Figure 1: Setup for the imitation game. The RGB-D sensor overhangs the NAO and tracked the gamer skeleton.

3 From gesture to imitation

3.1 Related work

Several methods are described in the literature to implement a real time imitation of human on a humanoid robot. In [9], they developed an interaction between Zeno, a humanoid robot, and children with ASD. For

the motion capture and child gesture recognition, they equipped the person with sensors. This does not allow natural and fluent games, usable during psychological session.

In [15], a method to make an imitation game with a NAO using an RGB-D for posture tracking is depicted. They calculated every angle on the human skeleton to send them to the NAO. A problem rises for reverse imitation, i.e. skeleton position from joint configuration, as needed in a turn tacking imitation. We have to compare position in joint space only, and we want to keep both options, joint space and 3D space.

In [5] is presented a method to enable humanoid robots to imitate complex whole-body motions of humans in real time. They use an Xsens MVN, a suit for motion tracking, so people need to be equipped with sensors. This method has the same issue for reverse imitation as the previous one.

3.2 Experimental setup

NAO can be enacted using the NAOqi API provided by Aldebaran¹. NAOqi allows controlling each motor in position and speed. There is also a way to control hands, feet and head positions instead of joint angles. The NAO Inverse Kinematics (IK) solver finds the joint configuration to reach the target positions. There is no control on how the IK is resolved. NAO may fall or not move in case of unbalanced or unreachable posture. NAO include a fall manager that we can active or deactivate. But posture corrections are not perfect and not fast enough. So we have to avoid as possible wobbly positions.

As previously said, while moving, the NAO embedded sensors may not provide reliable information. To complete sensing facilities in our experimental setup, a static RGB-D sensor overhangs NAO (see Fig. 1). The RGB-D sensor is set in the room, behind NAO. RGB and depth streams are used to compute the player’s skeleton and face. A skeleton is composed of 20 points, one for each joint, in 3D position. Sound localization and speech recognition will enhanced our imitation game with vocal commands.

The main program runs on a computer connected to the RGB-D sensor and sends orders wirelessly to the robot (see Fig. 3.2). The latency depends of how angle and order are computed. In the three applications, latency is short enough to allow real time applications.

4 JOINT CONFIGURATION RETARGETING

The first approach to convert the skeleton given by an RGB-D camera in a joint configuration for NAO is to calculate each articulation angle on this skeleton. We can extract angles from the tracked skeleton and transpose them to NAO. NAO have relatively few degrees of freedom (fig. 3) associated to motors, so we can compute an angle for each one. We need to know every joint

¹<http://www.aldebaran.com/>

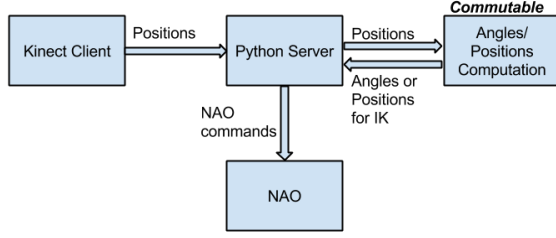


Figure 2: Software implementation scheme. The commutable component is made to allow comparison of different methods. There is a main application, named Server. It receive skeleton positions and send commands to NAO.

orientation and angle limitation. Formulas are specifics to NAO v3 and v4.

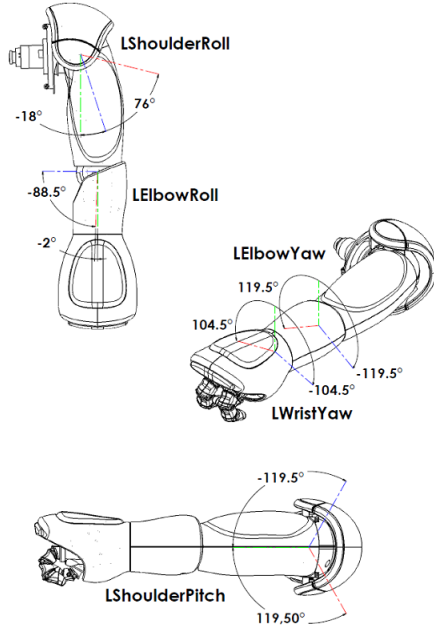


Figure 3: NAO's degrees of freedom for arms (source : <http://community.aldebaran.com/>)

We calculate the angles of both arms. First, we compute elbow vector (1) and shoulder vector (2) for the left side. Symmetrical formulas are used for the right side.

$$\vec{v_{ElbowL}} = \frac{elbowPosL - shoulderPosL}{\|elbowPosL - shoulderPosL\|} \quad (1)$$

$$\vec{v_{ShoulderL}} = \frac{hipPosL - shoulderPosL}{\|hipPosL - shoulderPosL\|} \quad (2)$$

These vectors allow the computation of ShoulderPitch left angle (3). We use geometrical angle between these two vectors. This value must be within a range from -2.857 to 2.857 rad, identically for the right side.

$$LShoulderPitch = \arccos \frac{\vec{v_{ElbowL}} \cdot \vec{v_{ShoulderL}}}{\|\vec{v_{ElbowL}}\| \cdot \|\vec{v_{ShoulderL}}\|} - \frac{\pi}{2} \quad (3)$$

We need the hip vector (4), between left and right hip, to find ShoulderRoll angle left (5) and right (6). This is in fact angle between arm vector and torso's plan.

This formula work in the most of case. But if the person use a torsion between pelvis and shoulder, imitation can fail. We can use a more complete formula, but in our case, we just need a simple imitation and see the limitation of this method. LShoulderRoll angle must be set between -0.3142 and -1.3265 rad and RShoulderRoll between -1.3265 and 0.3142 rad.

$$\vec{v_{Hip}} = \frac{hipPosR - hipPosL}{\|hipPosR - hipPosL\|} \quad (4)$$

$$LShoulderRoll = \frac{\pi}{2} - \arccos \frac{\vec{v_{ElbowL}} \cdot \vec{v_{Hip}}}{\|\vec{v_{ElbowL}}\| \cdot \|\vec{v_{Hip}}\|} \quad (5)$$

$$RShoulderRoll = -\frac{\pi}{2} + \arccos \frac{\vec{v_{ElbowR}} \cdot \vec{v_{Hip}}}{\|\vec{v_{ElbowR}}\| \cdot \|\vec{v_{Hip}}\|} \quad (6)$$

We compute hand vectors, between elbows and hands (7), to evaluate LElbowYaw (8) and RElbowYaw (9) angles. LElbowYaw and RElbowYaw angles must stay within the $[-2.0857, 2.0857]$ interval.

$$\vec{v_{HandL}} = \frac{ElbowPosL - HandPosL}{\|ElbowPosL - HandPosL\|} \quad (7)$$

$$LElbowYaw = \arccos \frac{\vec{v_{HandL}} \cdot \vec{v_{ShoulderL}}}{\|\vec{v_{HandL}}\| \cdot \|\vec{v_{ShoulderL}}\|} - \frac{\pi}{2} \quad (8)$$

$$RElbowYaw = \frac{\pi}{2} - \arccos \frac{\vec{v_{HandR}} \cdot \vec{v_{ShoulderR}}}{\|\vec{v_{HandR}}\| \cdot \|\vec{v_{ShoulderR}}\|} \quad (9)$$

LElbowRoll (10) and RElbowRoll (11) are calculated with forearm and arm vectors. LElbowRoll must stay between -1.5446 and 0.3142 rad and RElbowRoll between 0.0349 and 1.5446 rad.

$$LElbowRoll = -\arccos \frac{\vec{v_{HandL}} \cdot (\vec{v_{ElbowL}})}{\|\vec{v_{HandL}}\| \cdot \|\vec{v_{ElbowL}}\|} \quad (10)$$

$$RElbowRoll = \arccos \frac{\vec{v_{HandR}} \cdot (\vec{v_{ElbowR}})}{\|\vec{v_{HandR}}\| \cdot \|\vec{v_{ElbowR}}\|} \quad (11)$$

Using limits on motor angles permits to avoid the problem of unreachable positions.

Imitation and control game using this method is playable in real time. Nevertheless, every move made by humans is not possible for NAO (Fig. 4). Indeed, some movements seem to not be similar, because NAO do not have exactly same proportions and same member forms. An arm configuration on human does not necessarily match with the exact same configuration on NAO.

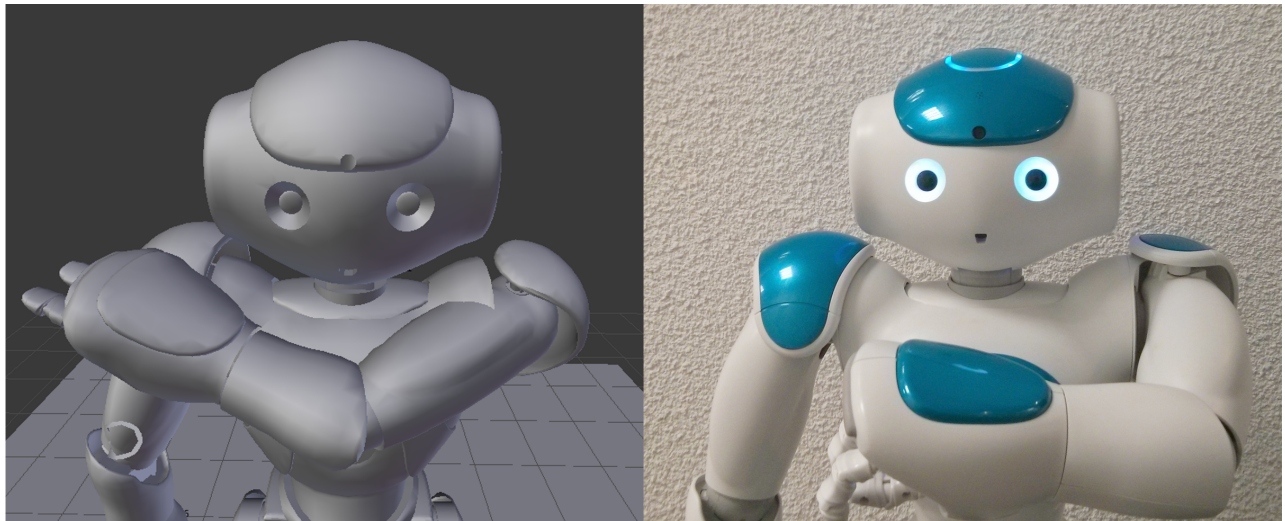


Figure 4: Impossible move. At left, one movement computed without constrains on joints, asking Nao to touch its shoulder. At right, the maximal corresponding position.

To produce a similar posture, sometimes joints configuration must be different. Looking at Fig. 1, one can see that raising NAO’s arms lead to non-human shoulder rotation. In this case, using directly joint configuration, it is impossible to compare NAO and human posture. Making a turn taking imitation game using this method is not relevant due to the need of posture comparison.

5 Model based retargeting

Inverse Kinematics can provide improved results. Movements are not exactly identical compared to human, but they are more adequate for NAO. Impossible configurations and joints limitations can be better handled.

5.1 NAOqi IK and scaling

NAOqi API provides an automatic inverse kinematics. The method takes hands, feet or head positions, and computes the path to reach them, this is an end-effector retargeting. In order to have a good control of NAO posture, we must have an accurate position. Arms, neck and legs must have the same size as NAO’s, to match its skeleton. To have these positions, we re-scale the skeleton provided by the RGB-D camera. We compute the scaling coefficient and use the Matrix (12) on the skeleton. It can be done for each member for better results. We have to know the exact size of the robot and suppose that the Kinect gives always exactly same point on same position on human body. Considering that NAO and Kinect space basis are different, we have to rotate space with the matrix (13) used to change space orientation. NAO basis is relative to robot position and tilt.

$$\begin{pmatrix} c & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (12)$$

with $c = \frac{Robot\ size}{Person\ size}$

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -\cos\theta & -\sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (13)$$

with $\theta = 90\text{ deg} + \text{robot tilt}$

This method have an important advantage compare to the previous one. Comparison between NAO’s and human end-effectors positions can be obtain directly. But we have to compute elbow or knee positions, so this method cannot be directly transpose to different robots. The main issue concerns unreachable positions for NAO. In these cases, the robot does not even try to reach them. The transformation between the two spaces must be accurate to avoid theses situation. But unreachable positions still exist, due to limited NAO’s DoF. Another problem is that the RGB-D sensor does not give exact joint position. On two consecutive frame, joint position can change, like elbow or wrist, not always placed on same position. This is a problem to compute joint length, to scale the skeleton. It is needed to compute length several time, to avoid unreachable positions. Another issue is that we have only an end-effector retargeting, so we cannot totally control NAO’s arms. A position of the hand can be reach with an infinity of joint configurations. As the elbow orientation is not controllable, we cannot be certain of correct imitation. If the goal is to reach a specific position for an end-effector, this method is applicable. But to match a posture for an imitation game, this is not usable.

5.2 iTaSC within Blender

Blender is an open-source software for 3D modeling and animation. Blender integrates a Game Engine and the iTaSC solver [12]. In this work, iTaSC is used to find the

robot joint configuration that most match the tracked skeleton.

We started from a 3D mesh of the NAO provided by Aldebaran, to have accurate proportions. We created a kinematic chain for this model, i.e. a chain of “bones” linked by universal joints on which meshes are fixated. Each of these joints can be constrained on each degree of freedom with maximum and minimum limits. Blender provides a 3D representation of each joint degrees of freedom to help in this process (Fig. 5). So, we copy and set NAO’s limits on each articulation in order to have the virtual skeleton of NAO.

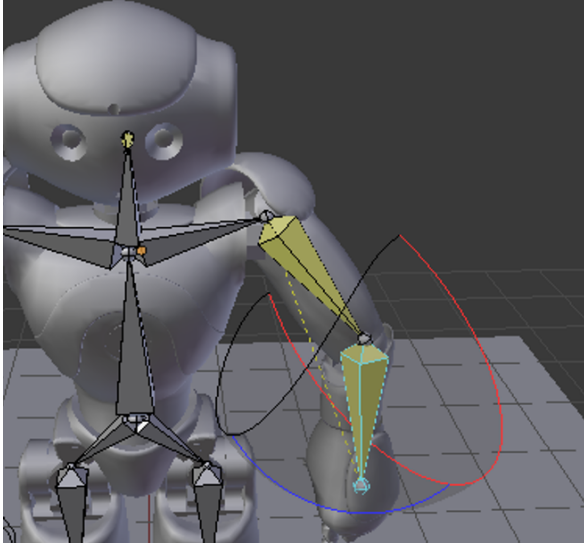


Figure 5: One Articulation set in Blender

Then, we created a virtual object for each joint of the tracked skeleton. In the beginning, before launching Blender Game Engine, they are placed on each bone extremity. They represent the positions that we want the robot joints to be in. We set each of them as target for the robot corresponding joint in the kinematic chain.

When we receive a skeleton, we consider shoulders center as the origin. Then, we compute the vector from this point to the left and right shoulders. We scale this vector to match the bones size in Blender. With this information, we are able to position the shoulder target. We repeat the same process to position elbow target relatively to the shoulder and wrist target relatively to the elbow.

Finally, the iTaSC solver is used to find a joint configuration with all the constraints defined. The kinematic chain degrees of freedom are hard constraints and joints targets are soft constraints.

The iTaSC solver tries to get the closest reachable position, considering every NAO’s limits. Then, the angles of the actual NAO joints are set to the value given by the iTaSC solver using the Joint Control API in NAOqi. The complete scheme of our implementation is described Fig. 6.

This method allow to simplify turn-taking imitation with a comparison between human and NAO positions made using targets positions. Using Blender add a latency. This is not an issue, because we use Blender for prototyping. Once the program is complete and connection between Kinect and iTaSC made, we can use iTaSC alone. Blender is a tool to simplify target positioning.

The inverse kinematic method is able to generalize the retargeting of motion to various robots. Moreover, using Blender to define the kinematic chain and iTaSC as inverse kinematic solver enable to rapidly reproduce the process described above to a different robot. To demonstrate this generalization, we use a virtual robot with tentacles, each one composed of 8 segments with universal joint on each articulation. On Fig. 7, the targets are placed at the same location. One can see that for a tracked skeleton, this method permits to compute controls for several robots.

6 Discussion and conclusion

To make strong social interaction between humans and robots, we have to find a way to measure how human and robots interact. To record it, we need to induce a natural interaction. Serious games seem to be a good way to provide a natural and constant interaction between a child and a humanoid robot. As a game for typical children and as a therapy for autistic children, imitation and control games are an easy way to do this.

In this paper, we described 3 different methods to implement an imitation game, particularly on making correspondence between the robot joint space and the skeleton tracked by an RGB-D sensor. Each method has their own advantages and drawbacks (table 1).

The direct retargeting of joint configuration is a light and fast way to solve this problem. But we can not easily control falling issues. There is no generalization without recalculating all formulas for a different robot. Also, to implement an imitation game that works both ways, we have to compare human and robot positions in joint space and not in 3D space.

Using NAOqi IK is a simple and reversible method. We can easily compare human posture to NAO posture. But the scaling is an issue and this method only implements end-effectors retargeting. Unreachable postures are not handle at all.

Using Blender and iTaSC as described is a general and simple method to control the robot. We can compare human position to robot position in 3D space. This method is not limited to the NAO robot and could be used for other humanoid robot. This method gives the best results, but has 3 principal limitations which will need future work for improvement:

- Our current system tracks the skeleton using the first version of the Kinect RGB-D sensor. The tracked skeleton lacks many degrees of freedom which would be useful in an imitation game, mainly the head pan and the wrist roll. The new Kinect 2 is able to track these joints and many others. Adding

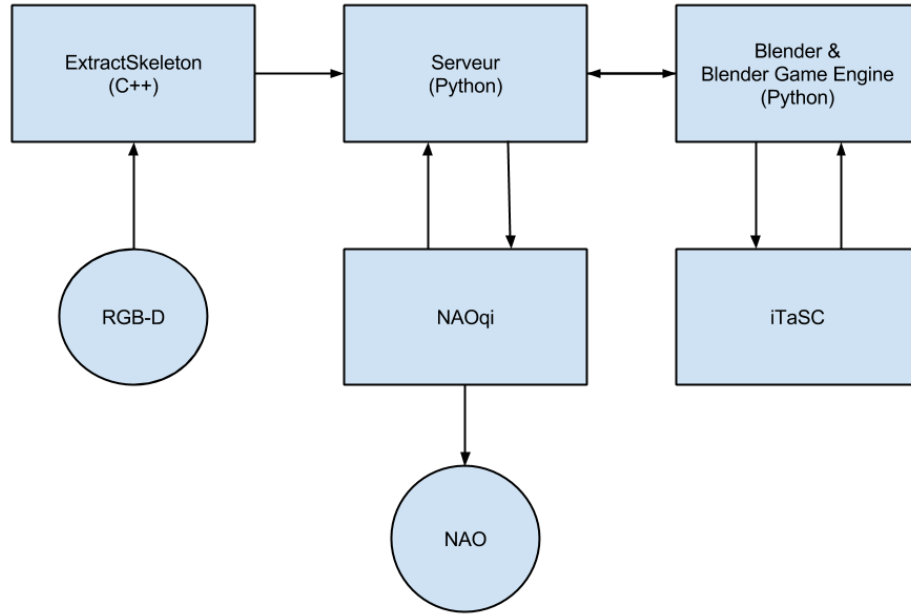


Figure 6: Software and hardware architecture of our implementation. Circles represent Hardware, and rectangles Software

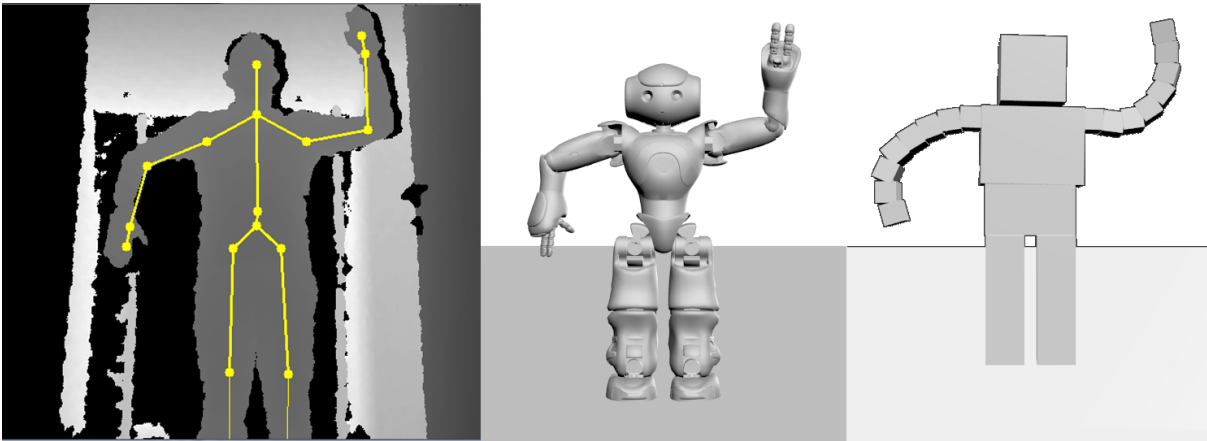


Figure 7: Full process and generalization. At left, the depth view and the skeleton extracted from the RGB-D sensor. In the middle, the NAO position computed using iTaSC and Blender. Last, retargeting on a robot with tentacular arms using the same method and same targets.

rotation to targets of the inverse kinematics solver, it could be possible to control every NAO's degrees of freedom, improving imitation.

- Our method, as presented in this paper, implements upper body imitation only. For the NAO to be able to imitate someone whole-body movement with this method, we need to tackle balancing of the NAO.

The NAOqi software suite provides a Fall Manager. Our preliminary tests using this tool were not conclusive. Another option would be to directly integrate the balancing constraint in the inverse kinematics task as done by Koenemann et al. [5].

- Currently, we need to manually scale the target at the size of the robot skeleton and to assign them the

Joint Configuration		Advantages	Inconvenients
			Light, Fast
Model Based Retargeting	NAOqi IK	Simple, Reversible	Bad results, not permissive, end effectors retargeting only
	Blender-iTaSC	Handle non reachable target, generalization, Reversible	3D mesh needed, External dependency

Table 1: Advantages and inconvenients of geometrical and model based retargeting methods

different bones of the kinematic chain. We plan to investigate the integration of an automated or partially automated method as developed in the context of motion capture retargeting to non-human characters [14].

The third retargeting method using iTaSC allowed us to implement a turn taking imitation game. Experiments with therapist are planned for a longitudinal study to train social skills of autistic children.

7 Acknowledgment

The author would like to thank Inria and French Ministry of Education and Researches. This project was conducted using facilities from the Amiquel4Home EquipEx (ANR-11-EQPX-0002).

References

- [1] John-John Cabibihan, Hifza Javed, Marcelo Ang Jr, and Sharifah Mariam Aljunied. Why robots? a survey on the roles and benefits of social robots in the therapy of children with autism. *International Journal of Social Robotics*, 5(4):593–618, 2013.
- [2] Angelica Escalona, Tiffany Field, Jacqueline Nadel, and Brenda Lundy. Brief report: Imitation effects on children with autism. *Journal of autism and developmental disorders*, 32(2):141–144, 2002.
- [3] Jing Guang Han, John Dalton, Brian Vaughan, Catharine Oertel, Ciaran Dougherty, Céline De Looze, and Nick Campbell. Collecting multi-modal data of human-robot interaction. In *Cognitive Infocommunications (CogInfoCom), 2011 2nd International Conference on*, pages 1–4. IEEE, 2011.
- [4] JingGuang Han, Nick Campbell, Kristiina Jokinen, and Graham Wilcock. Investigating the use of non-verbal cues in human-robot interaction with a nao robot. In *Cognitive Infocommunications (CogInfoCom), 2012 IEEE 3rd International Conference on*, pages 679–683. IEEE, 2012.
- [5] Jonas Koenemann, Felix Burget, Maren Bennewitz, F Burget, M Cenciarini, B Meier, H Bast, M Bennewitz, W Burgard, C Maurer, et al. Real-time imitation of human whole-body motions by humanoids. *Autonomous Robots*, 2013.
- [6] Nicole C Krämer, Sabrina Eimler, Astrid von der Pütten, and Sabine Payr. Theory of companions: what can theoretical models contribute to applications and understanding of human-robot interaction? *Applied Artificial Intelligence*, 25(6):474–502, 2011.
- [7] Daniel Maier, Armin Hornung, and Maren Bennewitz. Real-time navigation in 3d environments based on depth camera data. In *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, pages 692–697. IEEE, 2012.
- [8] Sylvie Pesty and Dominique Duhaut. Artificial companion: building a impacting relation. In *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, pages 2902–2907. IEEE, 2011.
- [9] Isura Ranatunga, Monica Beltran, Nahum A Torres, Nicoleta Bugnariu, Rita M Patterson, Carolyn Garver, and Dan O Popa. Human-robot upper body gesture imitation analysis for autism spectrum disorders. In *Social Robotics*, pages 218–228. Springer, 2013.
- [10] Philippe H Robert, Alexandra König, Hélène Amieva, Sandrine Andrieu, François Bremond, Roger Bullock, Mathieu Ceccaldi, Bruno Dubois, Serge Gauthier, Paul-Ariel Kenigsberg, et al. Recommendations for the use of serious games in people with alzheimer’s disease, related disorders and frailty. *Frontiers in aging neuroscience*, 6, 2014.
- [11] Ben Robins, Kerstin Dautenhahn, R Te Boekhorst, and Aude Billard. Robotic assistants in therapy and education of children with autism: can a small humanoid robot help encourage social interaction skills? *Universal Access in the Information Society*, 4(2):105–120, 2005.
- [12] Ruben Smits, Tinne De Laet, Kasper Claes, Herman Bruyninckx, and Joris De Schutter. itasc: a tool for multi-sensor integration in robot manipulation. In *Multisensor Fusion and Integration for Intelligent Systems, 2008. MFI 2008. IEEE International Conference on*, pages 426–433. IEEE, 2008.

- [13] J. Wainer, B. Robins, F. Amirabdollahian, and K. Dautenhahn. Using the humanoid robot kasper to autonomously play triadic games and facilitate collaborative play among children with autism. *Autonomous Mental Development, IEEE Transactions on*, PP(99):1–1, 2014.
- [14] Katsu Yamane, Yuka Ariki, and Jessica Hodgins. Animating non-humanoid characters with human motion data. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 169–178, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.
- [15] Fernando Zuher and Roseli Romero. Recognition of human motions for imitation and control of a humanoid robot. In *Robotics Symposium and Latin American Robotics Symposium (SBR-LARS), 2012 Brazilian*, pages 190–195. IEEE, 2012.

Combining interactive visualisation and UI plasticity to go beyond conventional homescreens

Gregory Cano
M1 Informatique
LIG - IIHM

Supervised by: Gaelle Calvary

I understand what plagiarism entails and I declare that this report is my own, original work.
Gregory Cano, 20/08/14:

Abstract

With the growing popularity of smartphones and mobile devices in general comes an increasing supply of applications and services. Therefore, it is legitimate to wonder how to display this much information on a small screen while keeping the interactivity. Enhancing the size of the screen is not a viable option, considering the difference in order of magnitude between the number of applications displayable at the same time and the amount of applications available. That's why we need a new approach that can display applications in a smart way, rather than displaying all the applications. Moreover, the majority of mobile devices only have a choice between a view of all the apps and a view of an actual running app. Our approach tries to break that in order to keep the context while running an application. The design space provides a general pattern displaying the application's most important features while keeping the context. It not only allows the user to see the context, but it also provides a more efficient way to use apps, given he doesn't need every function of it. The design space gives leads to the developer to build an adaptable interface while keeping it as stable as possible. This way, the user doesn't have to relearn the interface for every app. We also explored leads regarding the feed forward possibilities it offers. The device being aware of its environment can help providing relevant application choices automatically.

1 Introduction

With the increasing diversity of interaction devices together with the user's need for mobility, User Interfaces (UIs) must nowadays be usable everywhere, and capable of adaptation to the possibly changing context of use. This ability is named plasticity. Plasticity has so far mostly been studied from a methodological perspective. The research question was how to build UIs so that to ensure a potential for adaptation. The findings are:

- A clear distinction between the problem setting and the solution finding. The problem is defined as a set of models: the user's goal as rationale of the system, and the context of use as constraints for the solution. The solution is the outcome of an as large as possible exploration of the design space [Buxton, 2007]: it is modelled along four levels of abstraction (task, abstract, concrete and final UIs) [Calvary *et al.*, 2003]. Plasticity comes from the multiplicity of possible models at each level of abstraction;
- The promotion of Model Driven Engineering (MDE) and especially of models at runtime for supporting dynamic adaptation. The models can be carefully created at design time or generated at run time, e.g. for coping with contexts of use no pre-designed model is suited for;
- The need for combining MDE with other programming paradigms (e.g., service oriented approaches, component based approaches) so that to go beyond simple UIs made of classical widgets (e.g., labels, radio buttons).

These works have significantly contributed to maturing knowledge in the engineering of Human Computer Interaction (e.g., the W3C standardization of description languages of UIs). However progress made in plasticity remains rather limited as no convincing solution has emerged so far. This paper rewrites the problem of plasticity in the light of interactive visualization. It considers plasticity as a context-driven process of overviewing, zooming in, and filtering [Shneiderman, 1996]. Next section browses the state of the art in the field and related fields.

2 Related Work

2.1 Multi-Platform

Responsive design

Adapt the interface on the go as the display area grows/shrinks. Depending on the size of the area available, it uses a specific template. Once the size reaches a breakpoint, the interface switches to another template adapted to the new size. This way, several devices with different screen size can display the interface smoothly.

Graceful Degradation

This technic purpose is to develop UIs for multiple platforms. The idea is to build a base for the least constrained one and

apply transformation rules to develop the others [Florins *et al.*, 2006]. The service provided by UIs on several platforms has to be usable and consistent. Usable because it's not easy to cope with the changing capabilities from one platform to another. Consistent because users from a platform expect to take advantage of their knowledge on another UI of the same service.

2.2 Prompting

Feed forward

Environment can be used to enhance user experience with interfaces. For example, the android documentation website. As it is a development oriented website, it supposes you come here to get information about android development. As you type, it supplies several results that match what you search for.

Past experience can also improve the user experience by learning a user's habits. Automatic queries completion in search engines is a fine example. Prefixes used to generate completion can be associated to several propositions depending on gender, geographic position, and age [Shokouhi, 2013].

Ephemeral adaptation

When a collection of items is displayed, some of them are considered more relevant and appears directly. The others fade in a short while after [Findlater *et al.*, 2009]. This allows for a faster focus on the statistically more important items of the list hence better performance in using the application.

2.3 Interactive Visualisation

Mantra

[Shneiderman, 1996]

Overview : See the whole collection plus a detailed view. It is really helpful in map browsing. While the user looks at a detailed part, it is convenient to have a general view of the map. This way, he knows where he is and where he can go.

Zoom : Zoom in on parts of a collection. On large collection, it is often impossible to display enough details about every item. Zoom in on point of interest give more details and enables the user to browse conveniently among the collection.

Filter : Display only a specific kind of data in the collection. Filter by keyword or category narrows down the amount of items. It allows efficient and distraction-free data browsing.

Details-on-demand : Keep a view of the collection and get details about a subgroup when needed.

Semantic Zoom

In some datasets, zoom into parts of them isn't useful enough. Semantic zoom technic enables a specific amount of information to be displayed for a given zoom degree. I.e. map browsing [Frank and Timpf, 1994]. On a country scale, states, main roads and cities are displayed. On a city scale, the whole



Figure 1: Sketch of the camera magnified within the home screen

road network with streets names is displayed, as well as some points of interest such as a museum or a train station eventually. These levels of detail depend on the current zoom degree: the more the user zooms in, the more information he gets.

Focus+Context

Allows having a single continuous view displaying the context and one or more focus points [Cockburn *et al.*, 2009]. For instance, the Fisheye magnifies one or more areas within the context. The user can then know where he is among the collection while still be able to see more details on some areas.

3 Design space

3.1 Idea

The device screen size being limited, it has to be used as efficiently as possible. We chose to take a multi-scale approach to display applications. Today, they are, for the most part, used in full-screen, losing the context. To keep the context while using an application, we chose to give several representations of an application that don't take the whole screen. An application can be magnified to reveal some of its functions, but on a fraction of the screen. This magnification can be pushed automatically given the context is relevant, or triggered by the user. The goal is to provide an adaptable interface that will enhance user performance but it has to remain stable enough, so the user experience isn't altered. On Figure 1, we can see the home screen with a magnified camera app. We are then able to use the camera or interact with an-

All				
Multi				
Mono				
None				
Data / Task	None	Mono	Multi	All

Figure 2: Multi-scale responsive design space

other app instead. Figure 1 and Figure 2 are sketches of the phone home screen modified with an image editing software.

3.2 Multi-scale Data / Task

To magnify an app, we considered four breakpoints along two axes: Task and Data. Task represents the amount of features of the application available for the user, such as the camera trigger. Data represents the amount of data available for the user to browse, such as the photos he has taken. The breakpoints are as follow: None, Mono, Multi and All. Figure 2

- None: no additional Task/Data
- Mono: one additional Task/Data
- Multi: several (3+) additional Task/Data
- All: as much Task/Data as it is relevant to display without actually launching the app

The property of this representation is that we can easily apply it to another application with the same pattern. It helps the developer to adapt his application and keeping it coherent.

4 Working Prototype

4.1 Description

Through the process of implementation, some restrictions due to the Android language made me reconsider my project. There is no satisfying way in Android to scale an User Interface (UI) element both horizontally and vertically and keep

the global aspect of the interface. Considering that fact, I had to implement it horizontally only, thus lowering the multi-scale aspect of the solution. However, having it horizontally only make it possible to add multitasking easily, as we can see on Figure 3 Several applications are magnified and the user can interact with them independantly.

On Figure 4 we can see a magnified Camera application with one task (camera trigger) and one data (last photo taken). On the modified home screen, one tap on an application icon make it grow up to a certain point. Each tap adds a task or a data, and collapse when it is at maximum growth. A long touch on the application icon opens the application itself.

4.2 Architecture

The application is written in Android language, which mostly uses Java and XML languages. The interface (see Figure 5) is made of 5 horizontal *LinearLayout* put together by one vertical *LinearLayout*. Each horizontal *LinearLayout* contains 4 horizontal *LinearLayout*. They are the applications icons, coded with *TextView*. This way, we can easily add other UI elements such as buttons or images to each application. To add more space when we add UI elements, we increase the attribute *weight* of the *LinearLayout* child views. The more weight an element has, the more space it has compared to others of the same *LinearLayout*. The whole interface is put together in a XML file, to which we can add UI elements programmatically in the Java Activity. The elements added to an application can be simple elements such as buttons, or

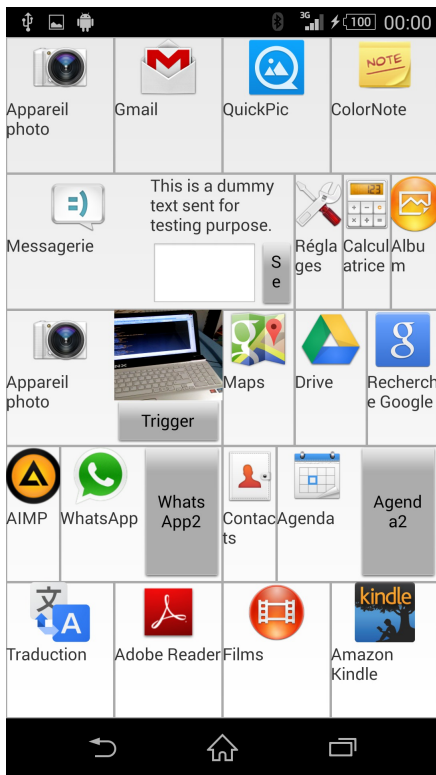


Figure 3: Multitask

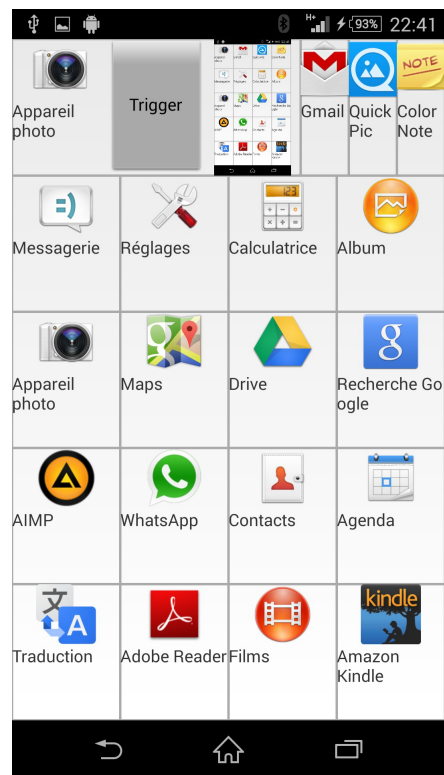


Figure 4: Camera Mono-Mono

new *LinearLayout* (as we can see on Figure 4 with the camera preview and trigger of the second Camera app).

5 Further Work

5.1 Context-aware system

The context can be the environment. The application is able to know where the user is and can act accordingly. For instance, given the environment is favourable (good exposition) and/or the spatiotemporal context is suitable (holiday). It may promote the Camera application by enhancing its dedicated space and adding Tasks/Data.

The context can be the device's position and movement in space, given by the accelerometer. As the smartphone is often, if not always, held horizontally, the vertical position can indicate a change in the user's intentions [Taylor and Bove, 2009].

The context can be the past experience, which implies some machine learning. The more we use the application, the more it learns about our habits and the way we use the different applications. That leads to better results regarding the relevance of each app/features for an app.

5.2 Predictions

Prediction choice

The applications have a function hierarchy. I.e. the camera trigger is not on the same level as the photo resolution choice. The fact of displaying several functions that are not on the

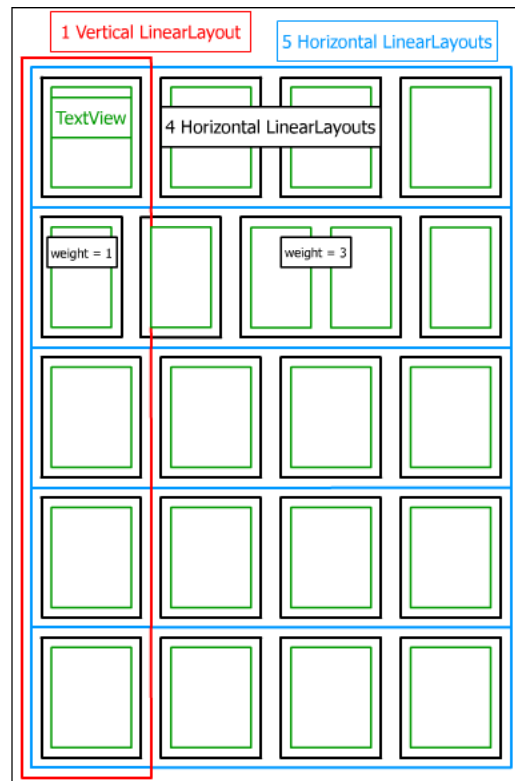


Figure 5: Architecture

same level may improve the usability. It can also confuse the user's representation of this hierarchy. However, displaying only functions of the same level can be too restrictive and lack flexibility, which is the opposite of plasticity. The same problem occurs in the case of applications structured with folders (iOS or Android desktop). The folder organisation results in a tree structure. If an application from the bottom of the tree is promoted, it comes to the same level as the one on the top of the tree. It may improve the efficiency but breaks the mental representation that the user has of the tree. The solution here would be to represent the application within its own context, i.e. with the folder that contains it. Therefore, the choice of applications to promote goes from 2D to a 3D representation.

Prediction use

The predictions can fail and the system must be able to learn from these mistakes. I.e. the context seems to give potential to an application in particular but the prediction's self-evaluation does not get a good score. Then no predictions are given. It is better not to do anything than use power, space and time resources for nothing. I.e. the user doesn't make use of the given predictions. There are two possibilities for next time here:

- change the predictions: it isn't stable and may confuse the user if they change every time
- do nothing: it makes the system pointless

They are not really satisfying and that's why we can allow the user to take control of these predictions.

User control

Give control of the display to the user is important for him: for both the number and the choice of predictions. The number of predictions stands for the level of granularity (Fig. 4). According to the context, the level given automatically may not satisfy the user. Be able to zoom in semantically can improve efficiency.

The choice of predictions may be wrong for the user. Be able to change the functions he wants can improve efficiency and can help the system to learn how to predict better.

This ability to control the display may improve efficiency but it needs interaction means to do so. Having too much parameter to set may burden the interface and make it unusable or make the user confused.

Ephemeral adaptation

The ephemeral adaptation [Findlater *et al.*, 2009] can also be an interesting way to improve this system: displaying only the most relevant applications for a short period of time to let the user decide, and fade in the others as time goes by. However, we have to adapt this technic to 2D, if not 3D, representation of items instead of 1D.

Evaluation

To define in what extent and at what point the user can have control over the predictions, we need to experiment it on multiple levels. I.e. the way of changing the number of tasks/datas displayed at once. It could be a pinch-to-zoom on the icon or buttons that we add to the representation. It could be automatic, or a combination of both. I.e. change the

tasks/datas displayed. It could be possible on a menu with a customisation option.

6 Conclusion

We have described a generic multi-scale pattern for smart-phone applications that provides adaptable yet stable representations. It uses a focus+context strategy combined with a semantic zoom that allows maximum details while keeping the context.

This is a cross-sectors project in the UI fields of research, we cannot achieve the conception with only one perspective. We need Interactive Visualisation to display a great amount of information, Plasticity to adapt the UI and enhance user's performance, Machine Learning to provide smart predictions to the user and Human Science to evaluate in what extent we can rely on the automatic nature of predictions and where is the user's place in this scheme.

7 Acknowledgements

I would like to thank Gaelle Calvary for the internship and for all the useful advices and support she gave me.

References

- [Buxton, 2007] Bill Buxton. *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.
- [Calvary *et al.*, 2003] Gaelle Calvary, Joelle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. A unifying reference framework for multi-target user interfaces. *INTERACTING WITH COMPUTERS*, 15:289–308, 2003.
- [Cockburn *et al.*, 2009] Andy Cockburn, Amy Karlson, and Benjamin B. Bederson. A review of overview+detail, zooming, and focus+context interfaces. *ACM Comput. Surv.*, 41(1):2:1–2:31, jan 2009.
- [Findlater *et al.*, 2009] Leah Findlater, Karyn Moffatt, Joanna McGrenere, and Jessica Dawson. Ephemeral adaptation: The use of gradual onset to improve menu selection performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '09*, pages 1655–1664, New York, NY, USA, 2009. ACM.
- [Florins *et al.*, 2006] Murielle Florins, Francisco Montero Simarro, Jean Vanderdonckt, and Benjamin Michotte. Splitting rules for graceful degradation of user interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces, AVI '06*, pages 59–66, New York, NY, USA, 2006. ACM.
- [Frank and Timpf, 1994] Andrew U. Frank and Sabine Timpf. Multiple representations for cartographic objects in a multi-scale tree - an intelligent graphical zoom. *Computers and Graphics*, 18(6):823–829, 1994.

- [Shneiderman, 1996] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. *IEEE Visual Languages (UMCP-CSD CS-TR-3665)*, pages 336–343, 1996.
- [Shokouhi, 2013] Milad Shokouhi. Learning to personalize query auto-completion. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '13*, pages 103–112, New York, NY, USA, 2013. ACM.
- [Taylor and Bove, 2009] Brandon T. Taylor and V. Michael Bove, Jr. Graspables: Grasp-recognition as a user interface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '09*, pages 917–926, New York, NY, USA, 2009. ACM.

Analysis of spatial phenomena using multi-scale aggregation

Geoffrey Danet


Grenoble, France

geoffrey.danet@e.ujf-grenoble.fr

Supervised by: Jean-Marc Vincent & Christine Plumejeaud-Perreau.

I understand what plagiarism entails and I declare that this report is my own, original work.

Name, date and signature:

Geoffrey Danet - 26/08/14 

Abstract

In order to use any aggregation process for a multi-scalar spatial analysis of landscape organization, the study of landscape metrics properties become primordial. According to the multi-scaling process used by the aggregation algorithm define by Lamarche-Perrin in his previous study [Lamarche-Perrin *et al.*, 2014], several parameter must be defined. In this paper, we identify the properties of metrics necessary for their usage on different scales and provide the definition of some calculation with their implementations in some languages (SQL, C++).

1 Introduction

Nowadays, in many disciplines, scientists have to use a vast quantity of data in order to work on their study. However, the huge quantity of data is often difficult to manipulate due to their microscopic and fragmented aspect. In order to resolve this problem, they use aggregation process to simplify calculations, but the result of those aggregations is not always pertinent and can induce too much loss of information for the analysis.

For example, in the ecological field, in rural areas with a predominance of agricultural activities, it has been proven that the study of environmental issues such as biodiversity preservation, soil erosion by water and tillage, erosive runoff, water pollution and gene fluxes may benefit from the long-term analysis of the crop mosaic resulting from farming practices. The arrangement, the shape and the nature of crops impact ecological processes at various scales [Lazrak *et al.*, 2010; Schaller *et al.*, 2012]. Thus ecology has developed methods for studying the spatial organization of landscapes based on the use of landscape metrics [Gustafson, 1998; Jiao and Liu, 2012]. In this paper, we take the case of the analysis of a crop mosaic resulting from farming practices for 5 years, observed on one municipality, la Foye-Monjault (Figure 1). We will show that the use of such metrics raise

many questions in term of scalability, interpretation, and even data representation.

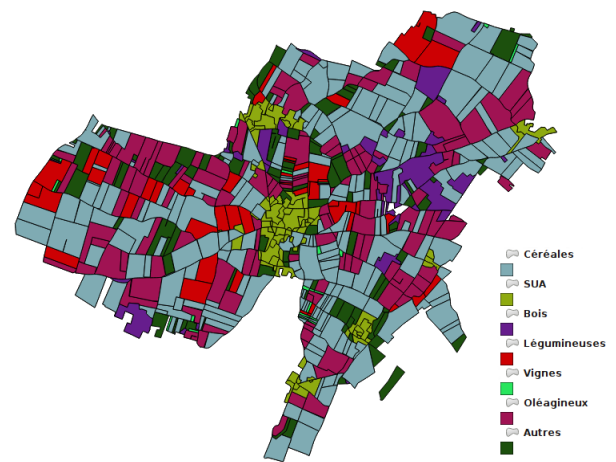


Figure 1: Categorized land lots of la Foye-Monjault for the year 2005.

1.1 Problematic

One of the main problem is that the use of metrics applied to agricultural parcels produces a vast quantity of data, that one has then to segmentize and re-aggregate, in order to make emerge interesting features in the structure of a landscape. Like in the case of the Modifiable Area Unit Problem [Openshaw and Taylor, 1979] we are facing a high complexity problem, not only due to the vast quantity of data, but also due to the meaning of the aggregations themselves. In fact, there is an infinite number of combinations to explore not only in the spatial dimension or temporal dimension, but even for any single classification of one metric measurement. Like in the approach developed by [Lamarche-Perrin *et al.*, 2014], it is very tempting to decompose the problem into smaller calculable pieces, and then to reaggregate. However, in spatial dimension, the variability of the produced signal is highly dependent upon the number, the size and the shape of the spatial division.

Furthermore, most of the previous studies were based on a matrix representation of the information, due to the nature of

data sources (mainly satellites imagery [Jiao and Liu, 2012]). Here we dispose of a spatio-temporal database with a vectorial representation of the information issued from an accurate observation on the ground. We are wondering whether an approach based on a vectorial representation of information could be more efficient for such calculus.

1.2 Goal

The objective of this study is to create synthetic representations of spatio-temporal data by using semantic aggregation of objects in order to characterize the spatial structures of an agricultural landscape and to be able to quantify what is stable or changing through time (looking for different recurrences of form/disposition according to the time). This raise firstly the question of the semantic that can be associated to the aggregation of metrics computed on parcels. It is necessary to find out measurements having scalable properties, in order to be able to quantify the loss of information each time we aggregate data together. The second question we are interested in is the data representation : which is the best suited for an efficient calculus ?

1.3 Data description

Since 1994, land uses of 19,000 land lots (mainly agricultural parcels) are recorded from the field each year by a program named *Zone Ateliers*¹ lead by the CNRS. A land lot is defined as a management unit, a polygon surrounded by entities having different land use in successive years (typically four). Each land lot is bounded by physical limits such as a road, a river, a field path or a single field boundary. It contains only one type of land use, and belongs to one unique person. It differs from the cadastral parcel, but also from the blocks stored in the RPG² which are updated every two years and distributed by IGN³.

For this study, a data sample of 4203 tuples describes the Foye-Monjault town in Deux-Sèvres county (France), between 2005 and 2010. Data are stored in a PostgreSQL database associate with PostGIS plugin for geometric manipulation. For each land lot, there is the following attributes:

- Plot identification: identify uniquely each land lot,
- Year: the year of the observation,
- Geometry: a polygon representing its shape in 2D,
- Land use code (OCS) : identify the type of culture or land use for the given year,
- Land use name: gives the name of the type of culture or land use for the given year.

Another table describes the various hierarchies of categories of land use. For instance, in the typology "biodiv", wheat, corn and sun flowers are grouped into the cereals category, and the cereal category belongs to the non-SUA category, opposed to the SUA category containing built parcels into the village.

¹<http://www.za-inee.org>

²Database recording the identification of agricultural parcels for the French government

³<http://www.geoportail.gouv.fr/donnee/48/registre-parcellaire-graphique-rpg-2010>

2 Measuring the landscape

In order to describe data of Crop Rotation System (CRS), we were interested in the usage of several landscape metrics. Landscape metrics are used to give a measurement of the forms, the configuration and the content of land lots. These metrics will allow to create a land lot inventory which increases the interpretation and the using of CRS data. For instance, it has been shown that landscape metrics can be used for the identification of land use [Jiao and Liu, 2012] : the intensive cultures of cereal are more often related to rectangular or simple land lots with big surface, whereas the vineyards with the same form are more often on lengthened and smallest land lots. What are particularities of the various landscape metrics? Do they possess an unique peculiarity according to a specific data type? Are they reusable on several scales? For example the ratio between the land lot area and the total area of the map are invariant whatever the scale. First of all, we need to better understand the nature of the information that landscape metrics provide to us, and their behavior during the aggregation process.

Among the various metrics, we can distinguish two types of metrics :

- The localization, the size and the form indexes concern the individual geometric properties of land lot.
- The arrangement and the neighborhood indexes which express spatial relations between several spatial objects groups (the arrangement correspond to the relative positions of objects, whereas neighborhood implies the use of a metric for distance computing).

In the following paragraphs, we take examples for each kind of index. There is an infinity of metrics, it is for this reason we choose to use the most used metrics to illustrate their behavior. We illustrate their spatial dispersion through a map using a divergent palette. We arbitrary choose to discretize the obtained index's range of values into 4 regular intervals classes.

2.1 Size indexes

The size indexes allow to describe characteristics related to the size of the land lot.

Perimeter index

The perimeter index allows to measure the external size of an entity. Like the area index, the perimeter is used in other metrics calculations.

The polygon perimeter is defined by the sum of its sides :

$$p = \sum_{i=0}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

With n being the count of points contained by the external ring of the polygon.

The Multipolygon perimeter correspond to the sum of each polygon's perimeter :

$$p = \sum_{i=0}^n p_i$$

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
72.06	401.90	607.50	690.70	882.80	5747.00

Table 1: Statistical summary of perimeter values.

With n being the count of polygon composed by the multi-polygon.

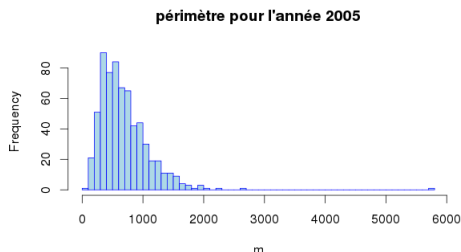


Figure 2: Perimeter histogram of la Foye-Monjault for 2005.

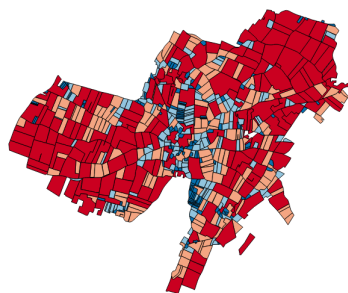


Figure 3: Spatial dispersion of the perimeter index of la Foye-Monjault for 2005. The palette is composed by 4 intervals (red for biggest perimeter and blue for smallest).

As we can see, land lots have mainly a perimeter included between 0 and 3,000 meters. The figure 3 shows that agricultural parcels, woods or hedge land lots have often a bigger perimeter than village land lots (which match the SUA of the figure 1).

Area index

The area index allows the surface measurement of an entity according to the units used to define the geometry. The area index is often used in other metrics calculations.

The area index calculation for a simple polygon follows the formula :

$$A = \frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i)$$

With n being the count of external points defining the shell of the polygon.

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
274.9	7065	17340	26440	33660	226600

Table 2: Statistical summary of area values.

However, this formula can not be used for certain polygons due to their complexity (like polygons which look like a "8"). In order to calculate the area of this type of polygons, it is needed to decompose it in more simple forms like for instance triangles. For that, the Delaunay triangulation is used to decompose the polygon into a set of triangles.

$$A = \sum_{i=0}^n a_i$$

With n being the count of triangles and a the area expressed by the following formula:

$$a = \frac{1}{2} |(x_B - x_A)(y_C - y_A) - (x_C - x_A)(y_B - y_A)|$$

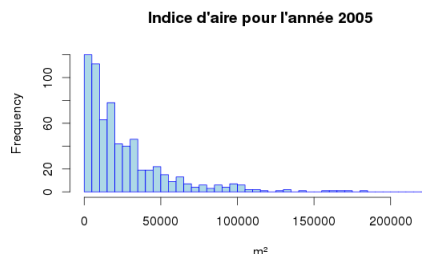


Figure 4: Area histogram of la Foye-Monjault for 2005.

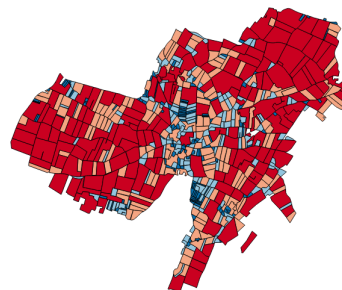


Figure 5: Area index of la Foye-Monjault for 2005. The palette is composed by 4 intervals (Red for biggest area and blue for smallest).

Like the perimeter, the area index shows a concentration between 0 and 50,000 square meters and a more uniform repartition between the interval 50,000 and 200,000 square meters. We observe that like the perimeter index, the area index have more big values on agricultural, hood and hedge

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.4161	0.9035	0.9733	0.9313	0.9973	1.0000

Table 3: Statistical summary of solidity values.

land lots than village land lots for instance.

Despite their apparent similarities, one should not conclude that the area and perimeter index are correlated : they can vary in opposite directions for the same object. For instance, a none convex land lot having several cavities on the circumference and an area of 10 square meters can have a more important perimeter than a land lot of the same area with a simple convex form.

Furthermore, the scale changes have a different impact on these metrics. In other words, if we multiply the scale by 2, the area is not multiplied by 2 but by 4, unlike to the perimeter which is multiplied by 2. Consequently, it is not possible to directly compare it on different scales.

2.2 Form indexes

Solidity index

The solidity index corresponds to the ratio between the entity with his convex envelope. Always positive, the index approaches 1 when the entity is convex. When, on contrary, an entity have one or more concavities (which we qualify as a complex form), the solidity index is much lower than 1. The solidity index is defined by the following formula :

$$I_s = \frac{Area}{ConvexEnvelopeArea}$$

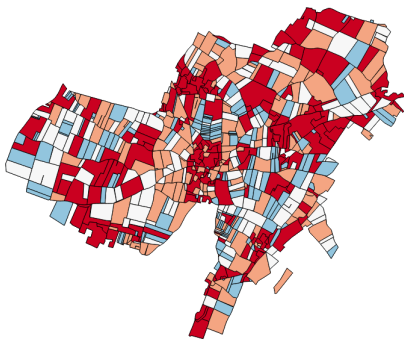


Figure 6: Solidity index of la Foye-Monjault for 2005. The palette is composed by 4 intervals (Red for convex land lot).

The repartition of the solidity index is concentrated in values closed to 1, which seems to show that land lots are most often of convex shape. The solidity index seems to show a more important concavity of shapes on forest, wood, build land lots. The index don't change on different scale.

Fractal index

The fractal index allows to determine the shape complexity of an entity (value between 1 and 2). More the land lot is complex, more its value approaches 2 and more it is simple

more the value approaches 1. For that, the formula compares the perimeter with the area. An important difference between the area and the perimeter shows that the external contour of the land lot is irregular and consequently complex.

$$I_f = \frac{2\ln Perimeter}{\ln Area}$$

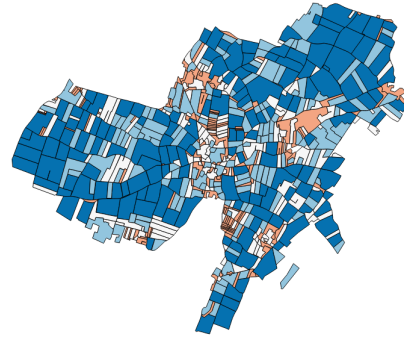


Figure 7: Fractal index of la Foye-Monjault for 2005. The palette is composed by 4 intervals (Red for complex land lots).

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.234	1.290	1.318	1.330	1.362	1.545

The fractal index seems be dispersed on a small interval : that shows a similarity of the land lots complexity (small complexity). As the solidity index, we can notice a bigger complexity on build or wood land lots for instance. However, the fractal index is not the same on different scales.

2.3 Arrangement indexes

Density index

The density index allows to determine the quantity of concentrated units into a zone. In our case, the density index is measured with the difference between the number of unity and the class area (External area of all entities). The density index is defined by the following formula :

$$MPS = \frac{ClassArea}{Numberofobjects}$$

3 Scalability of metrics

In order to use landscape metrics in an aggregation process as the one proposed in the previous study in trace aggregation for big distributed system by Robin Lamarche-Perrin [Lamarche-Perrin *et al.*, 2014], we must determine if landscape metrics can be scalable in order to be able to do a comparison on several scales. According to the previous part, we have seen that a metric is not always invariant at different scales.

3.1 Examples of scalable/non scalable metrics

The figure 8 is an aggregation example using the fractal mean of each land lot contained in the cell (zone of 1 square kilometer). This aggregation type have some problems,

particularly on the quantity of data loss and with his relevance. Indeed, as shown the figure 9, the land lot aggregation have been realized using the fractal index without taking into account the area. Consequently, if a zone with a big fractal value (thus complex) covering 80% of the total area is aggregated with another land lot with weak fractal value (simple) covering 20% of the area. The result are the average value of these two fractals indexes when the "weak" land lot represent only 1/5 of the Total Area of the Zone (TAZ). Furthermore, we must take in account the definitive loss of geometrical information after the aggregation process.

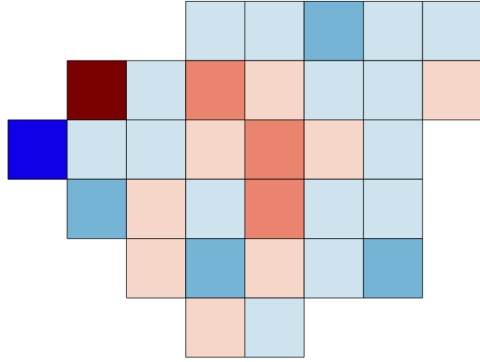


Figure 8: Example of fractal index aggregation with 1 square kilometer grid of La Foye-Monjault for 2005. (6 categories).

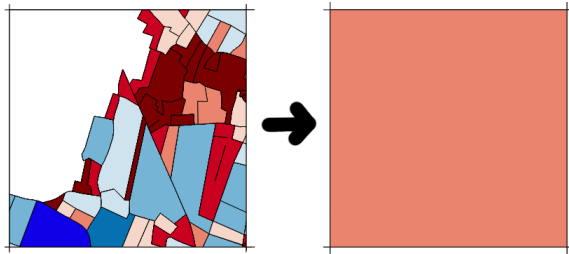


Figure 9: Example of fractal index aggregation with 1 square kilometer grid.

To solve this problem, it is required to proceed to a metric normalization. A good example of normalization is the Miller circularity index which uses the ratio between the land lot area and a circle with the same perimeter. The result is always limited between 0 and 1 where 1 mean a high circularity and 0 the opposite. Consequently, the index value never change whatever the scale and it is possible to compare land lots on different scales levels.

$$mc = \frac{4 * \pi * Area}{Perimeter^2}$$

The land lots aggregation must take in account spatial indicators. In other words, in the case of grid aggregation,

the land lot area aggregated is an essential factor to obtain a coherent spatial aggregation. In the case of aggregation using grid or quadtree model, it is necessary to normalize the information. For instance, we can use the area like a coefficient.

Let us take the following case :

Land lot 1 (p1): 20% of the TAZ, mc = 0,12 (Miller circularity index)

Land lot 2 (p2): 80% of the TAZ, mc = 0,85

If we do the aggregation process of the cell without taking into account the area, we obtain :

$$cel = \frac{0.12 + 0.85}{2} = 0,485$$

Now, if we apply the aggregation by taking into account the area :

$$cel = \frac{0.12 * 2 + 0.85 * 8}{10} = 0,704$$

thus we obtain the following formula :

$$p = \sum_{j=0}^n (I(j) * (\frac{a(j) * 100}{a(c)}))$$

I: Metric index, a: Area, c: Cell, j: Land lot

Be careful to the used method to define the land lot belonging's to a cell. In order to prevent false results, each land lot which intersects a set of cells must be divided in a set of little land lots according to each cells, otherwise a normalization of the surface can be useless.

3.2 Gravity center calculation

To find neighbors of a land plot, the most used method is to use the centroid of a land lot and take any polygon contained in the area of a circle. In this part, we are interested to find a specific point with additive property, unlike centroid, and which can be used with all kind of polygons. The gravity center has the additive property, but the main problem is the formula to calculate it on a polygon. in fact, the formula used to calculate the gravity center of a polygon depends of their form. Consequently, it is difficult to create an adaptive algorithm for all possible cases.

For this reason we have made a new approach to calculate the gravity center of any kind of polygon (multipolygon included). The solution is simple, knowing that gravity center has the additive property, we have chosen to divide the polygon into a set of triangles by using the Delaunay's triangulation. The triangles have the advantage to have only one formula to calculate it :

$$X_{TG} = \frac{1}{3}(X_A + X_B + X_C)$$

$$Y_{TG} = \frac{1}{3}(Y_A + Y_B + Y_C)$$

After the calculation of each gravity center of triangles, we weight each gravity center using its area. We obtain finally

the following formula to calculate the gravity center of a polygon :

$$X_G = \frac{1}{A(p)} \sum_{i=0}^n (X_{TG}(i) * A(i))$$

$$Y_G = \frac{1}{A(p)} \sum_{i=0}^n (Y_{TG}(i) * A(i))$$

Where A is the area, p the polygon and i the current triangle.

Gravity center additivity property

If we take the following example :

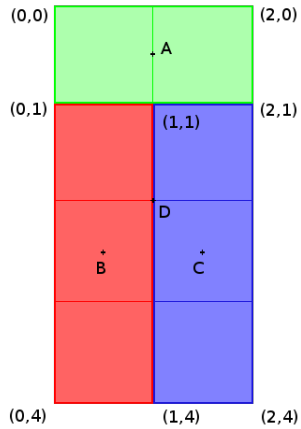


Figure 10: Set of polygon (each color represent a polygon).

On this example we have decomposed the land lot (Red + Green + Blue) in 3 land lot (Red, Green and Blue). To determine the additivity property of the gravity center, we calculate the gravity center of each part and weight the result by their respective area in order to obtain the gravity center of the initial polygon (see table 4 (R+G+B)) ; then, we compare the result with the gravity center calculate directly with the initial land lot (initial gravity center see table 4 (RGB)).

4 Computing landscape metrics

Now, we would like to study which kind of data representation optimizes the performance of the computation. That is to say we would increase the speed of the calculation, but also its accuracy, and lower the size of memory consumption.

Nowadays, spatial data representation are divided into two main approaches [Rigaux *et al.*, 2000]. The first representation, named grid or raster, allows to represent spatial data into a regular mesh, where each cell is localized by a row index and a column index, as shown on figure 11. This kind of representation accelerates the research of neighborhood data for instance, because the distance computation is simplified by the structure: there is only an addition of cells size to perform. However, the accuracy of the results is dependent from the scale of the mesh (the size of the cells). Moreover, there is a lot of redundancy in the data storage. This acts

Polygon	Area	Gravity center
Green	2	$X_G = \frac{0+0+1+1}{4} = 1$ $Y_G = \frac{0+1+1+0}{4} = \frac{1}{2}$
Red	3	$X_G = \frac{0+0+1+1}{4} = \frac{1}{2}$ $Y_G = \frac{1+4+4+1}{4} = \frac{5}{2}$
Blue	3	$X_G = \frac{1+1+2+2}{4} = \frac{3}{2}$ $Y_G = \frac{1+4+4+1}{4} = \frac{5}{2}$
RBG	8	$X_G = \frac{0+0+2+2}{4} = 1$ $Y_G = \frac{0+4+4+0}{4} = 2$
R+B+G	2+3+3=8	$X_G = \frac{1*2 + \frac{1}{2}*3 + \frac{3}{2}*3}{8} = 1$ $Y_G = \frac{\frac{1}{2}*2 + \frac{5}{2}*3 + \frac{5}{2}*3}{8} = 2$

Table 4: Calculation table.

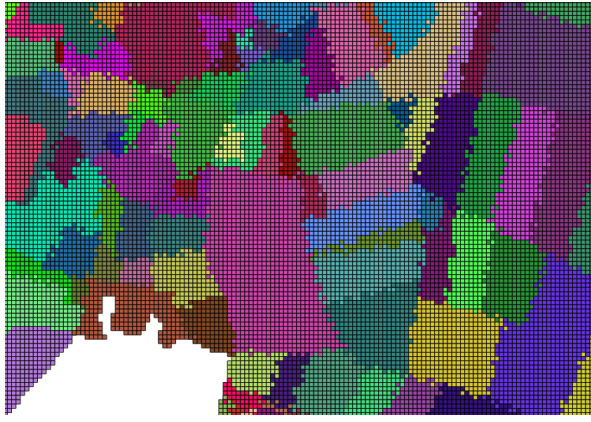


Figure 11: Extract of a 10 by 10 meters resolution grid of La Foye-Monjault for 2005. Coloration by distinct land lot.

	Vectorial	Grid 10m
Number of tuples	656	202,889
Weight	2 * size of float * n (number of points)	4 * size of float
Estimated Total Weight (bit)	2 (x and y) * 8 (float) * 14,959 (total number of points from all polygons) = 239,344 bits	202,889 (cells number) * 4 (xmin, xmax, ymin, ymax) * 8 (float) = 6,492 448 bits
Request time (ms) select * from table	365	5,598

Table 5: Storage comparing.

directly on the performance of the metrics computation.

The second representation is the vectorial representation, which allows to make precise spatial data using sets of points, lines, polygons or multipolygons. These geometric forms are constituted by points localized in the space by coordinates. This representation have the advantage of preserving a good information accuracy during the data entry and to store a minimum quantity of data. Furthermore, vectorial data have the advantage to allow the use of vectorial formulae already used in mathematics and physics without requiring modifications to adapt them.

4.1 Comparing vectorial to raster representation

In order to evaluate differences between these two methods, we have processed a set of indices to determine characteristics (execution time, complexity, ...) of each representation. All calculations are realized on our use case of La Foye-Monjault and with a C++ structure using the CGAL and GDAL library for the calculation of geometry.

As we can see, the quantity of information needed by vectorial representation is less important than the raster structure. It seems that the quantity of data required by grid directly im-

	Vectorial	Grid 10m
C++ Conversion (ms)	382	15,520
Gravity center calculation (ms)	2,052 (Delaunay's triangulation included)	704
Area calculation (ms)	9	263
Complete execution time (postgresql to c++ and gravity center calcul) (ms)	2,514	16,223
Area Precision	Depends of the data precision (+/- 0.00000001m from postGIS)	+/- 100m
Complexity	Delaunay triangulation: $O(n \log n)$ Gravity center: $O(n^2) + O(n \log n)$ Area: $O(n)$	Gravity center: $O(n^2)$ Area: $O(1)$

Table 6: Values of C++ execution.

pacts the quantity of memory and the performance in SQL request processing.

Execution of the C++ implementation

Despite a more important algorithm complexity, the vectorial representation seems faster than grid model. It can be explained by the smaller quantity of information used than the grid representation.

Spatial index calculation

Listing 1: Area SQL code

```
1 select sum(st_area(<geom_attribute>)) as CA
2 from <table_name>;
```

Index	Tuple	Execution time (ms)
Miller circularity	4203	82
Solidity	4203	82
Strain	4203	75
Fractal	4203	71
Dispersal	1	91
Area	1	11
Perimeter	1	11
Form	1	11
Density	1	11

Table 7: Values of metrics calculation.

Listing 2: Perimeter SQL code

```
1 select sum(st_perimeter(<geom_attribute>)) as  
  TE  
2 from <table_name>;
```

Listing 3: Miller circularity SQL code

```
1 select (4*pi()*st_area(<geom_attribute>))/pow  
  (st_perimeter(<geom_attribute>),2) as im  
2 from <table_name>;
```

Listing 4: Solidity SQL code

```
1 select st_area(<geom_attribute>)/st_area(  
  st_convexHull(<geom_attribute>)) as Is  
2 from <table_name>;
```

Listing 5: Fractal SQL code

```
1 select (2*ln(st_perimeter(<geom_attribute>))  
  /ln(st_area(<geom_attribute>))) as If  
2 from <table_name>;
```

Listing 6: Form SQL code

```
1 select sum(2*ln(st_perimeter(<geom_attribute  
  >))/ln(st_area(<geom_attribute>)))/count  
  (*) as MPFD  
2 from <table_name>;
```

Listing 7: Density SQL code

```
1 select sum(st_area(<geom_attribute>))/count  
  (*) as MPS  
2 from <table_name>;
```

5 Conclusion and perspectives

Basing on a real use case issues from the observation of agricultural land lot for many years in a town, we tried to discuss the various problems linked to the use of landscape metrics in an aggregation process, and to propose some ways of efficiently compute them.

Almost all of landscape metrics which can not be used directly in the aggregation process due to their none matching in several scale. To solve this problem, the metrics must be normalized.

Another question is the type of representation which must be used to represent and calculate all landscape metrics. For that, the vectorial representation should be the best solution according to the low quantity of data stored and the adaptive property with other vectorial formula. Furthermore, the difference of the execution time between raster and vectorial representation are more or less similar according to the type of calculation.

We have :

- Determined all potential useful indexes, and their behavior when changing of scale,
- Shown that vectorial representation is the best solution to represent geometric data for our purpose,
- Determined the best solution to find neighbors from polygonal representation, basing on gravity centers

This have been implemented inside a framework that can be easily reused by geographers or ecologists.

References

- [Gustafson, 1998] Eric J. Gustafson. Quantifying landscape spatial pattern: What is the state of the art? *Ecosystems*, 1(2):143–156, 1998.
- [Jiao and Liu, 2012] Limin Jiao and Yaolin Liu. Analyzing the shape characteristics of land use classes in remote sensing imagery. *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci.*, pages 135–140, 2012.
- [Lamarche-Perrin *et al.*, 2014] Robin Lamarche-Perrin, Lucas Mello Schnorr, Jean-Marc Vincent, and Yves Demazeau. Agrégation de traces pour la visualisation de grands systèmes distribués. *Technique et Science Informatiques*, 2014.
- [Lazrak *et al.*, 2010] ElGhali Lazrak, Jean-Francois Mari, and Marc Benot. Landscape regularity modelling for environmental challenges in agriculture. *Landscape Ecology*, 25(2):169–183, 2010.
- [Openshaw and Taylor, 1979] Stan Openshaw and P.J. Taylor. A million or so correlation coefficients: Three experiments on the modifiable areal unit problem. In N. Wrigley, editor, *Statistical methods in the spatial sciences*, pages 127–144. Pion, London, 1979.
- [Rigaux *et al.*, 2000] Philippe Rigaux, Michel Scholl, and Agnès Voisard. *Introduction to Spatial Databases: Applications to GIS*. Morgan Kaufmann, 2000.
- [Schaller *et al.*, 2012] Nomie Schaller, ElGhali Lazrak, Philippe Martin, Jean-Francois Mari, Christine Aubry, and Marc Benot. Combining farmers decision rules and landscape stochastic regularities for landscape modelling. *Landscape Ecology*, 27(3):433–446, 2012.