Université Grenoble Alpes

Magistère Informatique de Grenoble

Conférences de fin de stage

Le magistère est une formation complémetaire ayant pour objectif d'initier des étudiants au monde de la recherche en informatique

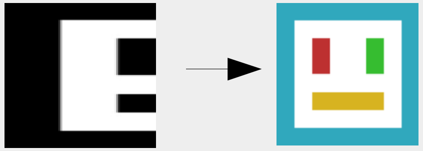
Proceedings

Parrallel Connecting Component Labeling

Hugo Bantignies

Connecting Component Labeling

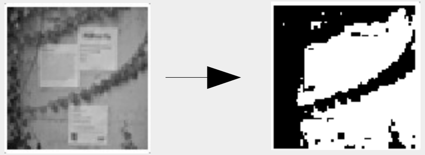
Connecting Component Labeling (CCL) :
Assigning a label to each connected component -blob- of a binary image.



Binarization

We built an optimized binarization to generate a binary image using intrasecs function in C.

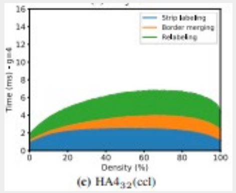
Converting a pixel image to a binary image depending on a threshold.



Algorithm Comparizon

CCL Algorithm is already implemented. For example, NVIDIA, built the algorithm with CUDA.

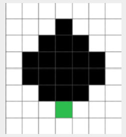
We wanted to compare our implementation idea with them, about the execution time.



Labeling execution time of 2048x2048 on GPU, CUDA

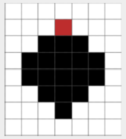
Root and Terminaison

A root is the start of a component -blob-, it will be stocked into an array during the first step of the algorithm.



The root is the green run

A terminaison is the end of the component -blob-.

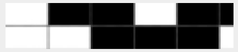


The terminaison is the red run

Propagation

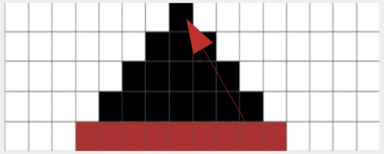
During the CCL, there are two propagation : Horizontal and Vertical.
Horizontal is the first propagation. During this step, each thread will create an array of label for his line and the next line. Each label will be connected to another label. At the end, we have got a horizontal propagation stored in arrays.

1	2	3	4
1	1	1	4



This array represents the first pixel line

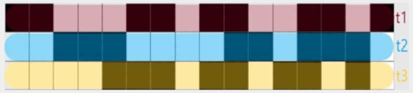
The second propagation is the Vertical.
An array of root was built during the horizontal propagation. Our goal is to propagate each root, from the bottom to the top, throw the arrays built during the first step.



Vertical propagation of the red root

Parallel and Multithreading

To optimize the execution, we have to use parallelization and threads. That is why, for the horizontal propagation, each line of the image is associated to a thread.



Each line is runned by a different thread (t1,t2,t3)

This allows us the entire scanning of the image in a short time. The horizontal propagation is done at the same time.

References

[1] HAL, *A new direct CCL and CCA Algorithms for GPUs*, Arthur Hennequin and Lionel Lacassagne

[2] *How to speed CCL up with SIMDRLE algorithms*, Florian Lemaitre, Arthur Hannqeuin, Lionel Lacassagne

Extended Abstract : Parallel Connecting Component Labelling Algorithm on Multi-Core CPU

Hugo Bantignies
Supervised by: Dominique Houzet

Abstract

This document is an extended abstract of ”Parallel Connecting Component Labeling Algorithm on Multi-Core CPU”.

1 Introduction

Currently, in image processing, the direct *Connecting Component Labeling* algorithm (CCL) was recently implemented in parallel. We wanted to submit our own version of the algorithm -a parallel version- to the *OpenCV* library and compare it with other implementations already existing on GPU -like NVIDIA with CUDA- as a bibliographic review. Before that, we wanted a Multi-core CPU version of our code in C programming language.
CCL is an algorithm consisting in providing a unique label to each connected component of a binary image used primarily in image detection. This algorithm can be enhanced in *Connecting Component Analysis* (CCA) to get more information about the image.

2 Binarization

The CCL algorithm needs a binary image. For this reason, we implemented our version of an optimized binarization by thresholding. This is an algorithm to convert a pixelized image into a binary image. We used intrinsics functions from the instruction set *Streaming SIMD Extensions 2* (SSE2) to optimize it. To read and stock the image, the *__m128i* type and his logical operations is an example. Also, we applied a byte compression to recreate the binary image.

3 New Parallel CCL Algorithm

3.1 Definitions

There are some terms to know about the algorithm :

- run : A sequence of ”1” in a row in a line of the binary image.
- root : The first run encounter of a connected component during the reading direction. It is the start of the component.
- ending : The last run encounter of a connected component during the reading direction. It is the end of the component.
- 4-connectivity : A pixel is only connected -or not- with his top, right, left and bottom neighbors.

3.2 Label Propagation

Propagation will detect and keep in table every connections between different run, to find every connected component at the end. There are two propagations during the execution of our CCL version : horizontal and vertical.

Horizontal Propagation

To have an optimized propagation we used multithreading. Each line of the binary image is associated to a thread. Each thread explores his current line and the next one, to have a common line with the next thread.
Every run in a binary line is identified by a unique label. The thread will explore his two lines switching when the end of a run is farther than the other one on the other line. Two tables -A and B- will keep all run connections following 4-connectivity. A root table is filled during this propagation.

Vertical Propagation

Every root stocked in the table, will be propagated through A and B tables generated by threads during the horizontal propagation. The root propagation will be stopped on a ending run. There is a particular situation called a conflict, when a root, during his propagation, arrives on an update run by a previous root propagation. A conflict notified will be saved in an equivalence table like that : (updated root, new root) which means that every run updated by the previous root or by the current one are in the same component.

3.3 Equivalence table

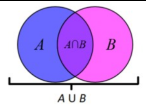
At the end of propagation, all missing root labels inside the equivalence table are a unique label of a connected component. Otherwise, for each equivalent root, the connected component has several equivalent roots but one single label -the minus one-.

Validation of Presburger Sets operations

Objective of the internship: test ISL operations between Presburger sets using Z3 SMT-Solver

During the internship, ISL and Z3 were used in their python version

Integer Set Library (ISL)	***Z3 (SMT-Solver)***
-Stores sets defined by Presburger Formulae *-Allows users to compute operations between sets or within sets:*	-Checks the satisfiability of first order logic formulae, with integers operations.



How to extract the formula defining an ISL set S in order to store it in Z3 format ?

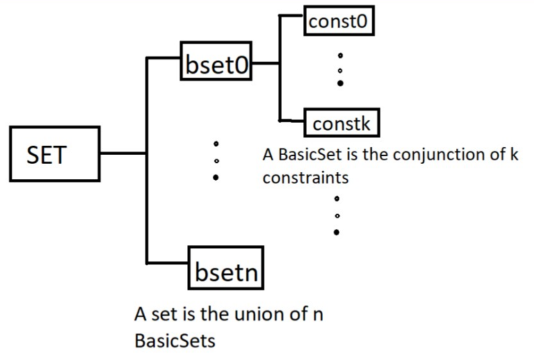
```
Input: ISL set S
Output: A logical formula in Z3 format F

F := False
forEach bset in S.get_basic_sets():
    b := True
    forEach constraint in bset.get_constraints():
        c := 0
        forEach x in constraint.get_variables_by_name():
            c:= c + Int(x)*constraint.get_variables_by_name()[x]
        if constraint.is_equality:
            b := b and(c = 0 )
        else:
            b := b and (c >= 0)
    F := F or b
```

How are ISL Sets stored and defined ?

Let S be an ISL Set. If S.get_basic_sets() returns the array: [b0, ... , bn], then S = b0 union b1 ... union bn.
Let B be anISLBasicSet. If B.get_constraints() returns the array: [c0, ..., cn], then B = { x : c0(x) and c1(x) ... and cn(x) }

Let c be an ISL Constraint. If c.get_variables_by_name() returns the dictionnary : D = { x0 : a0 , ..., xn: an, 1 : a(n+1) }, then the constraint corresponds to the formula:
a0*x0 + a1*x1 + ... + an*xn + a(n+1) >= 0 if c.is_equality(),
a0*x0 + a1*x1 + ... + an*xn + a(n+1) = 0 else.



SET
bset0
const0
constk
A BasicSet is the conjunction of k constraints
bsetn
A set is the union of n BasicSets

ISL Set data structure

How to create an ISL set defined with a Z3 formula and how to test Operations between and within sets ?

S = Set(str(F)).

How to test an union or intersection operation in ISL ?

-Create Z3 polyhedra.
-Create ISL sets defined by these polyhedra
-Compute the operations between these sets
-Extract the formula defining the resulting set in Z3 format
-The resulting formula has to be equivalent to "A and B" if the operation tested is the intersection (Set(str(A)).intersect(Set(str(B)) and it has to be equivalent to "A or B" if the operation tested is the union.

Cooper's method of quantifier elimination, and its use to test ISL projection:
Let A = { [x0, ... xi , ... xn] : F(x0, ..., xi, ..., xn) }
Let B be A's projection in the xi axis.
Then
B = { [x0, ... x(i-1), x(i+1) , ..., xn] : Exists xi, F(x0, ... x(i-1), x(i+1),...xn) }
In order to test this operation in ISL, a method of extracting constraints with existance quantifiers would have to be programmed.
Biggest issue:
Z3 can induce an infinite loop when checking formulas with existance quantifiers
Solution:
Implement Cooper's method of quantifier elimination = >
It allows to find a finite set in which we can check the validity of the formula. If the formula is true with each element of the set, then the whole formula is verified.

Validation of Presburger Sets operations

Nicolas Besson
UGA and Verimag
Grenoble, France
nickxbesson@gmail.com

Abstract

The *Validation of Presburger Sets operations* ~~will be printed from electronic manuscripts submitted by the authors. The electronic manuscript will also be included in the online version of the proceedings. This paper provides the style instructions.~~

1 Introduction

1.1 ISL

Integer Set Library (ISL) is a C library used for storing sets of vectors of integers, and computing different operations such as union, intersection and projection on these sets.

1.2 Presburger Set definition

A set can be defined with a logical formula, i.e. , if an object satisfies the formula, then it belongs to the set.
In ISL, these formulas are defined with the Presburger arithmetic language.

1.3 Main objective

The main objective of the internship was to validate the main operations offered by ISL, using an SMT solver (Z3). During the internship, the python libraries ISLpy and Z3py were used in order to facilitate the task.

1.4 Main issues

After a projection operation, in many cases the formula ends up having an existential quantifier. The main problem is that Z3 may induce an infinite loop when checking the satisfiability of this kind of formula. This is why Cooper's method of quantifier elimination had to be implemented. This allows to find a finite set, in which if the formula is satisfied, then the original formula is also satisfied.
The "Set" data structure of the ISL library had to be understood in order to program robust tests.

2. Testing the operations

2.1 Creation of Z3 random polyhedra

The first objective of the internship was to create a function creating a random polyhedron, i.e, an array containing formulas of the form : $a_1 x_1 + \ldots + a_n x_n + a_{n+1} \geq 0$ with every a_i created randomly . The formulas are objects of the class "Expression" declared in the Z3 library, allowing to keep in memory operations with variables.

2.2 Creation of ISL sets

Once the polyhedron is created, the conjunction of these inequalities is computed. In order to create an ISL set defined by this formula, it suffices to give as a parameter to the initializer Set(), the string: " *{ [$x_1, \ldots, x_n$] : Conjunction(P) }* ", assuming "*Conjunction(P)*" computes the conjunction of the inequalities forming the array P.

2.3 Computing Union and Intersection of ISL sets

Let *A = { x : P(x) }* and *B = { x : Q(x) }*. In set theory, the intersection of A and B corresponds to :
A ∩ B = { x : P(x) and Q(x) }. The ISL operation A.intersect(B) is supposed to keep in memory the simplification of "P(x) and Q(x)". The tests consist in seeing if in fact, the simplification is equal to the expected formula. *Idem* for the union of sets (A.union(B)).

2.4 Main idea for the tests

With two sets A and B created with two Z3 random polyhedra P_A and P_B, the intersection or the union is computed. Then the simplification of the formula F, defining the resulting set is extracted as a Z3 expression. Z3, being a SMT-Solver allows to prove if two formulas are equivalent :
If Xor (F , P_A and P_B) is unsatisfiable, then the formulas "F" and "P_A and P_B" are equivalent. The expected result to consider that the operation works, is that for each randomly created P_A and P_B, its conjunction has to be equivalent to the resulting F if the tested operation is the intersection, and its disjunction has to be equivalent to F if the tested operation is the union.

2.5 Results

As expected, in every tested case (with 10 000 tests), the result was what was expected. In order to conclude that the operations always work, the code of the operations has to be mathematically proven (which was not the point of the internship).

References

[Aaron Bradley, Zohar Manna, 2007] The Calculus of Computation: Decision Procedures with Application to Verification

Effective implementations of Information-Set Decoding algorithms

Dorian Biichlé supervized by Pierre Karpman

Team CASC @ Laboratoire Jean Kuntzmann, University of Grenoble-Alpes

Introduction

Information-Set Decoding forms a class of exponential probabilistic algorithms whose goal is to find a low-weight codeword in a given random linear code.

ISD Algorithms

- These algorithms operates on either the **generator** or the **parity-check matrix** of the linear code.
- By manipulating the matrix and checking its rows, they try to find a codeword (a vector) with a weight as close as possible to a **theoretical minimum bound.**
- Two main algorithms have been implemented: **Prange** [1] and **Stern** [3]. Prange idea is to analyze rows of the generator matrix in systematic form (when the left part of the generator matrix is the identity), while Stern is an improvement of Prange using a time-memory trade-off.
- Two implementation of Stern have been done; a first one using only Canteaut and Chabaud improvements [2], and a second one adding optimizations and recommended parameters of Bernstein, Lange and Peters [4]

Implementations

- Implementations have been tailored to solve $[1280, 640]$ binary codes. Thus, **many optimizations rely on the fixed size of the code's parameters to get faster**.
- These algorithms have been implemented to make the most of the **AVX-512 instructions set**. Hence, the **implementations have been done in C, using AVX-512 intrinsics**.
- Some linear algebra operations are done using the M4RI library, which is specialized in fast arithmethic operations over $\mathbb{F}_2$.

Results

Evaluating performances

- Solving the low-weight codeword problem in practice has implications in cryptography as some cryptosystems rely on its hardness.
- In fact, our **implementations have been tested against decoding-challenge.org's challenges**, where the goal is to find a codeword with the smallest weight possible for the given instance of a random linear binary code.
- The purpose of the website is to assert the practical hardness of the problem, with real-world cryptographic parameters.

- The code has been **launched on one of GRICAD's cluster**, *Dahu* (dahu110 & dahu111 nodes, 2x8 cores Intel(R) Xeon(R) Gold 6244 CPU 3.60GHz).
- As ISD algorithms are probabilistic, the problem is embarassingly parallel, which means one can run 16 instances of the program simultaneously on 16 cores without much effort.

Results in numbers

	Prange	Stern	Stern 2
48h runs/core done	16	61	96
evaluated codewords/s	58 400 000	9 800 000	15 295 500
Av. smallest codeword/run	228.4	227.3	225
Smallest codeword found	224	222	219

Figure: Overall stats for each implementations

After many runs, the best codeword found has a weight of 219 (theoretical minimum bound is 144 for the given instances).

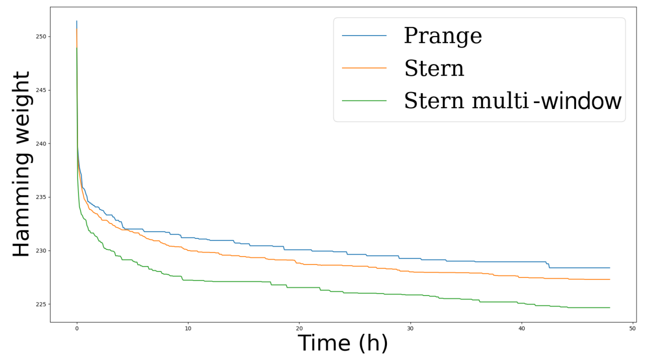
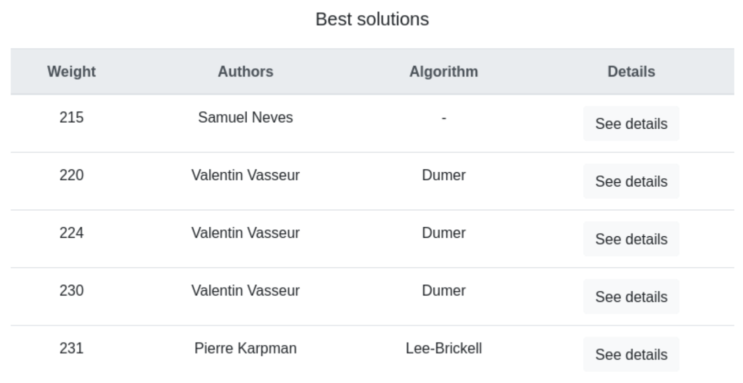


Figure: Average minimum weight found over time for each implementation



Best solutions

Weight	Authors	Algorithm	Details
215	Samuel Neves	-	See details
220	Valentin Vasseur	Dumer	See details
224	Valentin Vasseur	Dumer	See details
230	Valentin Vasseur	Dumer	See details
231	Pierre Karpman	Lee-Brickell	See details

Figure: decoding-challenge.org's scoreboard

That was close but not enough to get into decoding-challenge.org's scoreboard.

- The code is accessible on github at github.com/Antoxyde/isd.

References

- Prange, E. *The use of information sets in decoding cyclic codes* IRE Transactions IT-8(1962) S5–S9
- Canteaut, A., Chabaud,F *A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511.* EEE Transactions on Information Theory44(1) (January 1998) 367–378
- Stern, J. *A method for finding codewords of small weight* Coding theory and applications, volume 388 of Lecture Notes in Computer Science, 1989.
- Bernstein, DJ., Lange, T., Peters, C. *Attacking and defending the McEliece cryptosystem* 2008

Effective implementations of information-set decoding algorithms

Dorian Biichlé, supervised by Pierre Karpman
Team CASC, Laboratoire Jean Kuntzmann, University of Grenoble-Alpes

1 Introduction

ISD (**I**nformation-**S**et **D**ecoding) is a class of probabilistic algorithms whose goal is to find a low-weight codeword in a random linear code. This problem is considered hard, as these algorithms all have an exponential complexity. Some cryptosystems rely on this hardness. The goal of this work is to ensure the practical hardness of the low-weight codeword problem with real-world cryptographic parameters against ISD algorithms.

1.1 Linear codes

A random linear $[n, k]$ code is a linear subspace $C \subset \mathbb{F}_q^n$ of dimension k. We will only consider the binary case, ie $q = 2$. C can be characterized by a $k \times n$ generator matrix G, whose rows forms a basis of C. This matrix is said to be in systematic form if its in the form $G = [I_k|R]$, where R is a random $k \times (n-k)$ matrix.

An information set of G is a subset of length k of $\{1, .., n\}$, corresponding to indexes of linearly independant columns of G.

The Gilbert-Varshamov bound on linear codes tells us that there exist on average a unique codeword of a certain given weight. By "low-weight codeword", we thus means a codeword whose weight is as close as possible to that bound.

2 ISD algorithms

The first algorithm implemented is Prange [3]. It aims to find low-weight codewords by using a generator matrix in systematic form. It uses the fact the rows have a maximum weight of $n - k + 1$ (and expected $\frac{n-k}{2} + 1$).

The second algorithm implemented is Stern [4]. The idea is the same as Prange, but using a time-memory trade-off. It strives to lower the weight of analysed codewords by finding linear combinations having the same value on a specific window. The maximum weight of analysed codewords is thus $n - k - \ell + 2p$ (and expected $\frac{n-k-l}{2} + 2p$), where p is the number of rows in the linear combinations, and ℓ the size of the window.

2.1 Improvements

Over the years, researchers have found several improvements to these algorithms. We implemented some of their ideas:

- Canteaut and Chabaud [1], who lowered the complexity of Stern algorithm by making single steps of the Gaussian elimination algorithm, instead of all the steps.
- Bernstein, Lange and Peters [2], who gives interesting optimizations, for instance forcing several windows to zero in one iteration of the Stern algorithm, instead of only one.

3 Implementations

Implementations were tailored to decoding-challenge.org's challenges, which are $[1280, 640]$ binary codes. That means that the code heavily rely on the fact that these parameters are fixed. We strove to make full use of the AVX-512 instructions set, as the codewords are 1280 bits long. An optimization made consist of storing only the redundant part of the generator matrix, meaning storing a single row takes 640 bits, so 512 bits and 128 bits vectors. Some of the linear algebra operations are done using the M4RI library, which is specialised in arithmetic over $\mathbb{F}_2$. The whole code is accessible on a github repository at github.com/Antoxyde/isd

4 Results

The code has been launched on one of the GRICAD's cluster, Dahu (specifically on the dahu110 & dahu111 nodes, 2x8 cores Intel(R) Xeon(R) Gold 6244CPU 3.60GHz).

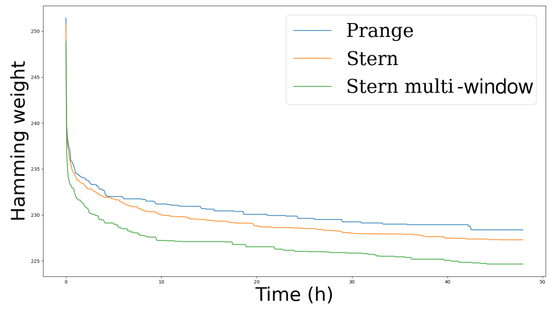


Figure 1: Average min. weight found over 48h runs

The G-V bound for $[1280, 640]$ binary codes is 144, and after running our Stern implementation for approximately 192 core-days, the best codewords we obtained has a weight of 219.

References

[1] F. Chabaud A. Canteaut. “A new algorithm for finding minimum-weight words in a linear code: Application to McEliece’s cryptosystem and to narrow-sense BCH codes of length 511”. In: *IEEE Transactions on In-formation Theory44* (1998), pp. 367–368.

[2] C. Peters D. J. Bernstein T. Lange. “Attacking and defending the McEliece cryptosystem”. In: (2008).

[3] E. Prange. “The use of information sets in decoding cyclic code”. In: *IRE Transactions IT-8* (1962).

[4] J. Stern. “A method for finding codewords of small weight”. In: *Coding theory and applications* 388 (1989).

Lattice based resolution for the Hidden Number Problem

Gaspard Anthoine[1] and Pierre Karpman[1]

***Abstract*— We are interested in the problem of finding an unknown value in an interval $[0, p-1]$ for a prime p given a small number of relations of the form $y_i + A_i y_0 + B_i \equiv 0 \mod p$ only partially known and according to a uniform law. Where A_i and B_i are uniformly random public values and y_i are either well unknown according to a non-uniform law, or only partially known and according to a uniform law.**

I. INTRODUCTION

In cryptography digital signatures are used for verifying authenticity of a message, a document. They are in reality widely used for software distribution, financial transactions, cryptocurrencies, website certificates. Digital signatures relies on asymmetric cryptography, the DSA standard was proposed by NIST and adopted in 1994. This scheme is based on modular exponentiation and a discrete logarithm problem. There is now some new scheme for digital signature for example based on elliptic curves named ECDSA. Attacking theses signatures algorithms when a certain amount of bits leaks can be linked to what is called the Hidden Number Problem. There are two principals methods used to solve this problem, the first is called Bleichenbacher [1]. The Bleichenbacher's method permits to attack leaks with a small number of bits but requires a lot of signatures. The method we will study in this paper will be based on lattices. The HNP can be solved via lattice as shown by Boneh and Venkatesan [2]. Howgrave-Graham & Smart [3] showed in 2001 how to attack DSA using lattice and building an appropriate basis. The main concern of this work was to replicate the results of Howgrave-Graham and Smart and to see if the recent progress of Albrecht et al. [4] for short vector search in a lattice could improve the parameters (signature size and number of leaking bits). The advantage of the lattice approach compared to the Bleichenbacher one is that fewer signatures are needed but in return more bits have to leak.

II. RELATED WORKS

Lattice based attacks on electronic signature have been first introduced by Howgrave-Graham & Smart [3] in 2001. In 2002 Nguyen & Shparlinski [5] improved the results of Howgrave-Graham & Smart by recovering the private key on a 160 bits signature with only 3 bit of leaked nonce. Lattice based attacks have then shown to have real world application. For example Benger & all [6] showed in 2014 how to attack OpenSSL ECDSA with a side channel on cache (FLUSH+RELOAD) and lattice based attack to recover the private key. In 2019 Breitner & Heninger [7] show how to attack cryptocurrencies using transactions signature with biased nonce and à lattice based private key recovery.

III. PRELIMINARIES

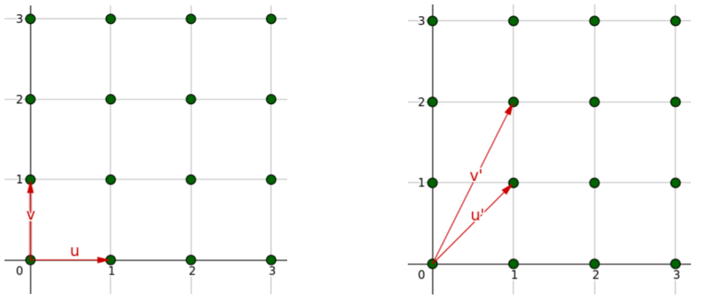
A. Lattices

Definition 1: A lattice is a discrete subgroup of space, with a finite rank n. Let $v_1, \dots, v_n \in \mathbb{R}^m$ be n linearly independents vectors. A lattice spanned by $\{v_1, \dots, v_n\}$ is the set of all linear combination of $v_1, \dots, v_n$ such that :

$$\mathcal{L} = \left\{ \sum_{i=1}^{n} a_i v_i, a_i \in \mathbb{Z} \right\}$$

$\{v_1, \dots, v_n\}$ will be called the basis of L. When $m = n$ we can construct a matrix of the basis by the vectors down row by row.

This formal definition can be seen more simply as a regular arrangement of points. We will define the dimension of L as $dim(\mathcal{L}) = n$.



(a) Lattice with basis vector u, v (b) Lattice with basis vector u', v'

Fig. 1: Same lattice spanned by different vectors

B. Lattice problems

1) Shortest Vector Problem: Given a basis, find the smallest vector in the lattice. This problem is a problem considered as difficult (NP-HARD).

Definition 2: We will call an orthogonality defect for a basis B of a lattice Λ:

$$\delta(B) = \frac{\Pi_{i=1}^{n} \|b_i\|}{\sqrt{\det(B^T B)}} = \frac{\Pi_{i=1}^{n} \|b_i\|}{d(\Lambda)}$$

In the case of a perfectly orthogonal basis we will have $\delta(B) = 1$.

A SVP problem can however be approached by several basis reduction algorithms. Notably LLL [8] or BKZ. We will then call this problem a γ-approximation that we will note SVP_γ with $\gamma > 1$ and $\delta(B) < \gamma$, for large enough gamma we can use LLL (of the order of $\gamma = 2^{\Omega(n)}$ with n

*This work was not supported by any organization
[1] Laboratoire Jean kuntzmann

the size of the lattice). If you want a better reduction you will have to use BKZ (or even HKZ [4]) but the reduction will then be slower (will depend on some parameters). If you use LLL you will have a bad approximation ratio $((\frac{2}{\sqrt{3}})^n)$ but the algorithm will be polynomial.

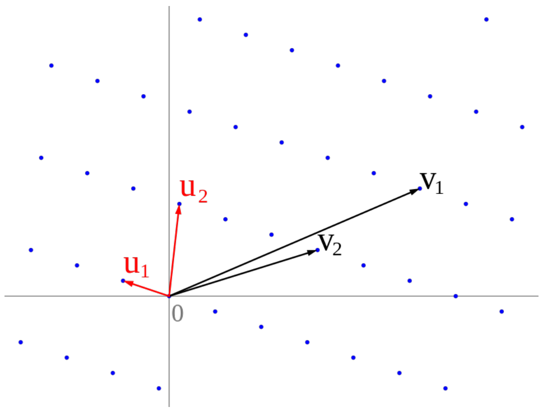


Fig. 2: Example of basis reduction

As you can see in figure 2 the vector of the basis are shorter and more orthogonal (closer to the orthogonality defect) after the reduction.

2) Closest Vector Problem: Given a lattice, find the closest lattice's vector to a given vector which does not belong to the lattice. We can define given a target vector $t \in \mathbb{R}^n$ and a lattice $\mathcal{L} \subset \mathbb{R}^n$:

$$\underset{x\in\mathcal{L}}{dist}(\mathcal{L}, t) = \|x - t\|$$

The hardness of this problem is known to be NP-complete (first proved by Emde Boas [9]) and highly related to hardness of SVP. There is an approximation problem called γ-CVP for any $gamma > 1$ approximation factor, $t \in \mathbb{R}^n$, a lattice $\mathcal{L} \subset R^n$, the goal is to output $x \in \mathcal{L}$ with :

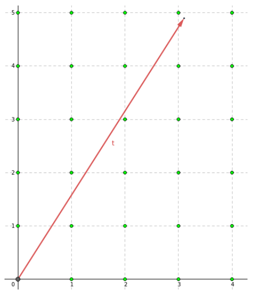
$$\|x - t\| \leq \gamma \cdot dist(\mathcal{L}, t)$$

This problem is also known to be NP-complete for any $\gamma > n^{c/\log\log n}$ for some constant c [10]. This problem can be solved easily if one has a good base of the lattice with Babai's algorithm [11] ~~and~~ the quality of the vector we found will depend on the quality of the reduction basis.

3) Closest Vector Problem reduction to Shortest Vector Problem: It's possible to reduce ~~a~~ CVP problem to an SVP by building a new lattice with a particular basis :

$$B' = \begin{pmatrix} B & 0 \\ u & n \end{pmatrix}$$

Where B is the basis of the lattice for the CVP problem with vector u which where not in the lattice and B' is a basis for an SVP problem. Applying a basis reduction algorithm to B will give you γ-CVP solution with gamma depending on the quality of the reduction. In figure 3 you can see an example of CVP for a lattice $\mathcal{L}$ with basis $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and a vector $t = \begin{pmatrix} 3.1 \\ 4.9 \end{pmatrix}$. The solution vector is $x = \begin{pmatrix} 3 \\ 5 \end{pmatrix}$.


(a) Vector $t \notin \mathcal{L}$

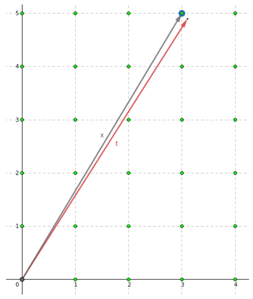

(b) vector $x \in \mathcal{L}$

Fig. 3: Example of CVP

C. Hidden Number Problem

Definition 3: The hidden problem number can be described as follows : considering $T = (t_1, \ldots, t_n), t_i \in \mathbb{F}_p$, $\alpha \in F_p$, and $MSB_l(\alpha t_i)$ the l most significant bits of αt_i. Having T and $(MSB_l(\alpha t_1 \mod p), \ldots, MSB_l(\alpha t_n \mod p))$ can we found α ?
This model is a generalisation of some attack problems on DSA (digital signature Algorithm). This problem can be solved via a lattice as shown by Boneh and Venkatesan [2] or by Bleichenbacher's method [1]. The complexity of the Hidden Number Problem depends on several parameters :

- Time complexity : the runtime of an algorithm to solve an instance of the problem.
- Data complexity : the number of samples that will be required to solve an instance.

In this paper we will only study solving HNP with lattice, this requires less sample than a Bleichenbacher's method but will only work for more bits leaking (the parameter l of the problem).

D. Digital Signature Algorithm

DSA or Digital Signature Algorithm is an algorithm proposed by NIST and adopted in 1994 as a standard for digital signatures. The algorithm is in three parts :

1) Alice publish a group $\mathbb{G}$ of cardinal p (prime) and a generator $g \in \mathbb{G}$. Alice get a random x and publish $h = g^x$. We imagine there is a bijective function $f : \mathbb{G} \to \mathbb{Z}/p\mathbb{Z}$ that everybody knows. x will be the private key of Alice that she has to keep secret.
2) If Alice wants to sign a message $m \in Z/p\mathbb{Z}$ she computes $r = f(g^y)$ and s such that :

$$m \equiv sy - xr \mod p$$

for some randomly chosen $y \in\in Z/p\mathbb{Z}$.
3) Alice can now send $(m, r, s) to$ Bob.

To verify the signature Bob can compute $f(g^{ms^{-1}} h^{rs^{-1}}) = r$.

IV. Attacking DSA with Lattice

It's possible to translate an HNP instance into equations modulo a prime p. You can write $MSB_l(\alpha t_i \mod p)$ as :

$$\alpha t_i - b_i \equiv y_i \mod p$$

with b_i the least significant bits of αt_i.

If we got h signatures of the form :

$$m_i - s_i y_i + x r_i \equiv 0 \mod p$$

With x and y_i unknown we can write :

$$y_i + A_i x + B_i \equiv O \mod p$$

With $A_i = (-s_i)^{-1} \cdot r_i$ and $B_i = (-s_i)^{-1} \cdot m_i$. If we know some of the most significant bits of the y_i we now have a HNP instance : $MSB_l(-y_i - B_i \mod p) = MSB_l(A_i x \mod p)$ and thus recover x. We can build the basis B for à lattice $\mathcal{L}$ as follows :

$$B = \begin{pmatrix} p \times 2^{l+1} & 0 & 0 & \dots & 0 & 0 \\ 0 & p \times 2^{l+1} & 0 & \dots & 0 & 0 \\ \vdots & & & \ddots & & \\ 0 & \dots & \dots & \dots & p \times 2^{l+1} & 0 \\ A_1 \times 2^{l+1} & A_2 \times 2^{l+1} & \dots & A_n \times 2^{l+1} & 1 & 0 \end{pmatrix}$$

And applying Babai algorithm with a vector $t = (MSB(\alpha \times A_1) \times 2^{l+1}, \dots, MSB(\alpha \times A_n) \times 2^{l+1})$ which is outside the lattice will give with good probability a vector $y = (\dots, \alpha)$. As we have seen before we can transform this instance of a closest vector problem into a shortest vector problem by building this basis :

$$B' = \begin{pmatrix} p \times 2^{l+1} & 0 & 0 & \dots & 0 & 0 \\ 0 & p \times 2^{l+1} & 0 & \dots & 0 & 0 \\ \vdots & & & \ddots & & \\ 0 & \dots & \dots & \dots & p \times 2^{l+1} & 0 \\ A_1 \times 2^{l+1} & A_2 \times 2^{l+1} & \dots & A_n \times 2^{l+1} & 1 & 0 \\ t_1 \times 2^{l+1} & t_2 \times 2^{l+1} & \dots & t_n \times 2^{l+1} & 0 & p \end{pmatrix}$$

If we apply a basis reduction algorithm on B' we get a good probability to find $y' = (y, -p)$ as one of the vectors of the basis.

It's also possible to slightly improve the dimension of the matrix by rearranging the diffrents $y_i + A_i x + B_i$ as follow :

- $x \equiv -A_n^{-1} y_n - A_n^{-1} B_n \mod p$
- so we have $y_i + C_i y_0 + D_i \equiv 0 \mod p$ with $C_i = -A_n^{-1} \cdot A_i$, $D_i = -A_n^{-1} B_n + B_i$ and $y_0 = y_n$

At the end we have $n - 1$ equations so this reduce the dimension of the lattice.

V. Results

This results are the bests results we got using 100 signatures samples.

Nombre de bits	CVP LLL	CVP BKZ	SVP
160	6	5	4
192	6	5	5

For the 4 bits of nonce leak and a 160 bits signature we got à 50% success.For the 5 bits of nonce leak and a 192 bits signature we got à 100% success.

VI. Conclusions

To conclude ~~we~~ we managed to get better results than Howgrave-Graham & Smart [3] but not as good as Nguyen & Shparlinski [5] who managed to have 100% success on a 4 bits leak for a 160 bits signature and even some success for 3 bits leak. Maybe we could try to understand more how the scaling factor is working in the basis (2^{l+1}). We got also another idea would be to use the knowledge about h_n with the last optimisation to get a bigger advantage. Further work should be done to understand how the different factors works with the algorithm.

References

[1] D. Bleichenbacher, "Chosen ciphertext attacks against protocols based on the rsa encryption standard pkcs #1," in *Advances in Cryptology — CRYPTO '98* (H. Krawczyk, ed.), (Berlin, Heidelberg), pp. 1–12, Springer Berlin Heidelberg, 1998.

[2] D. Boneh and R. Venkatesan, "Hardness of computing the most significant bits of secret keys in diffie-hellman and related schemes," in *Advances in Cryptology — CRYPTO '96* (N. Koblitz, ed.), (Berlin, Heidelberg), pp. 129–142, Springer Berlin Heidelberg, 1996.

[3] N. A. Howgrave-graham and N. P. Smart, "Lattice attacks on digital signature schemes," *Designs, Codes and Cryptography*, vol. 23, pp. 283–290, 1999.

[4] M. R. Albrecht, L. Ducas, G. Herold, E. Kirshanova, E. W. Postlethwaite, and M. Stevens, "The general sieve kernel and new records in lattice reduction," in *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II* (Y. Ishai and V. Rijmen, eds.), vol. 11477 of *Lecture Notes in Computer Science*, pp. 717–746, Springer, 2019.

[5] P. Q. Nguyen and I. E. Shparlinski, "The insecurity of the digital signature algorithm with partially known nonces," *Journal of Cryptology*, vol. 15, pp. 151–176, 2000.

[6] N. Benger, J. van de Pol, N. P. Smart, and Y. Yarom, ""ooh aah... just a little bit" : A small amount of side channel can go a long way," in *CHES*, pp. 75–92, Springer, 2014.

[7] J. Breitner and N. Heninger, "Biased nonce sense: Lattice attacks against weak ecdsa signatures in cryptocurrencies." Cryptology ePrint Archive, Report 2019/023, 2019. https://eprint.iacr.org/2019/023.

[8] A. K. Lenstra, H. W. Lenstra, and L. Lovasz, "Factoring polynomials with rational coefficients," *MATH. ANN*, vol. 261, pp. 515–534, 1982.

[9] P. Boas, "Another np-complete problem and the complexity of computing short vectors in a lattice," 1981.

[10] I. Dinur, G. Kindler, R. Raz, and S. Safra, "Approximating cvp to within almost-polynomial factors is np-hard," *Combinatorica*, vol. 23, p. 205–243, Apr. 2003.

[11] L. Babai, "On lovász' lattice reduction and the nearest lattice point problem," in *STACS 85* (K. Mehlhorn, ed.), (Berlin, Heidelberg), pp. 13–20, Springer Berlin Heidelberg, 1985.

Curve-based stylization of 3D animations [*]

FARHAT Amine
Under the supervision of
VERGNE Romain • THOLLOT Joëlle

Abstract

Expressive rendering can be done in multiple ways. One of them involves the repetition of a simple 2D mark. The usage of a map of 3D positions as an input allows for a placement of these marks that follows the geometry of the objects, and stays coherent with their deformations as well as with the position of the camera. This aspect makes this method extremely effective for the stylization of 3D animations. In this project, these marks are procedurally generated in order to follow the shape of a curved path, which facilitates the reproduction of brush-based or line-based art styles. The path is computed from a series of points and directions sampled from the tangent map of the scene. This computation is done by fitting a quadratic function. Since the sampled point are determined by the geometry, the path remains coherent with the movements of the objects, while still maintaining a 2D painted aspect.

1 Introduction

When it comes to rendering images on a computer, one might seek to stray away from the usual pursuit of photo-realism, aiming instead for a stylized appearance. In computer graphics, the process of stylization usually consists in taking a photograph and applying a series of filters – that can be implemented as programmable shaders – in order to recreate the desired art style. The past two decades have witnessed an ever-increasing mainstream usage of these stylization algorithms, notably in real time photo manipulation.

While most of these methods yield more than satisfying results, their main limitation manifests itself when being applied to animated scenes. Since the stylization process is generally applied to one frame at a time, this can lead to important discrepancies between two consecutive images. For example, an oil-paint filter might represent an object with horizontal strokes in one frames and vertical strokes in the next, which leads to a noticeable increase in popping. This issue was formalized as the temporal coherence by Bénard et al. [4].

According to Bénard, a stylization method can be appraised on three distinct factors : Flatness, Motion coherence and Temporal continuity. But since these three factors are unelikely to be achieved at the same time, each method has to settle on a compromise [figure 1].

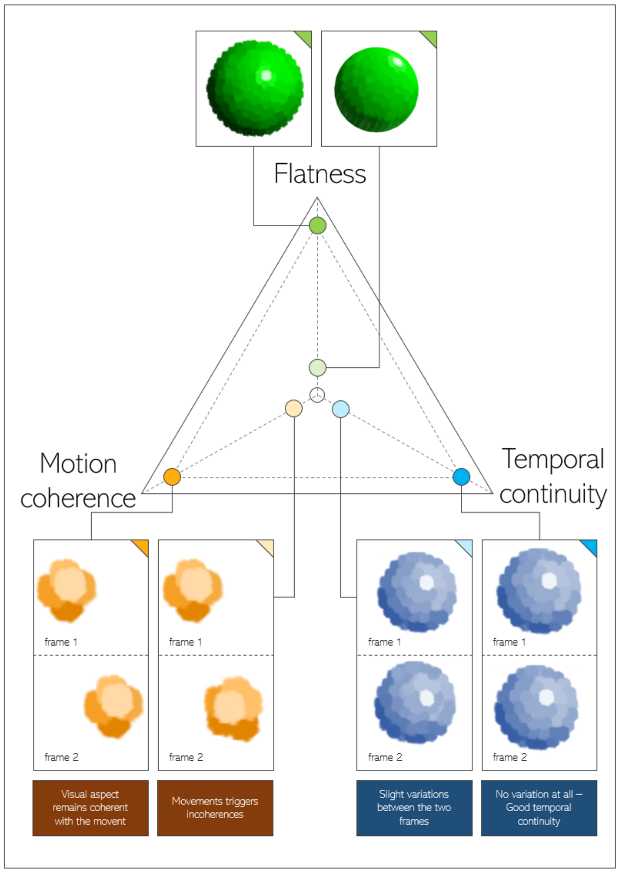

Flatness
Motion coherence
Temporal continuity
frame 1
frame 2
frame 1
frame 2
frame 1
frame 2
frame 1
frame 2
Visual aspect remains coherent with the movent
Movements triggers incoherences
Slight variations between the two frames
No variation at all – Good temporal continuity

Figure 1: Visual representation of the Temporal coherence problem.

Last year, during my internship, I briefly contributed to the

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

development of a new stylization method, the groundwork of which was laid down by my supervisors.

Our method ensures motion coherence by taking 3D scenes as an input instead of 2D images. Since the input is a 3D scene, it can provide more information on what is to be represented on the canvas, including but not restricted to the objects on the scene, their positions, the orientation of their surfaces, the textures and colors of their materials, etc. This data is stored in a stack of images, referred to as G-buffers: In a normal rendered image, each pixel is a projection of a section of a 3D scene onto the 2D screen. Usually, the pixel takes the color of the object it is representing, such as to be understood by the human brain. However, it is also possible to store other kinds of information in that pixel, such as the orientation of the corresponding point in the 3D space.

In broad terms, this process will procedurally create a 3D cloud of "anchor" points, generated within the volume of the objects on the scene. These anchor points are thus coherent with the transformations of the objects, since they are generated from the information gathered from the G-buffers. Then, the image is recreated by exclusively "pasting" a simple image on the canvas, depending on the position of the anchor points on the screen. These images, henceforth referred to as "splats", can vary in terms of shape, size, color, and texture. These parameters are the defining factors of the type of art style that is going to be outputted.

This information can also be used throughout the stylization pipeline to obtain different results depending on the user's will. This allows for an extremely wide scope of art styles, ranging from pointillism to watercolor.

However, the splats themselves are limited to 2D square images, which can be too restrictive in certain scenarios: For example, when representing a moving object, some artists rely on motion lines - an artistic abstraction of motion blur - to better express the trajectory of the object. Additionally, many styles rely on long, uninterrupted strokes that bend and twists depending on the geometry of the object.

By building on this method, it is possible to allow the splats to be molded by a curved path, the shape of which could be determined by a number of factors based on the characteristics of the object. And since the splats are placed on the canvas depending on the geometrical properties of the objects, we can ensure a satisfying level of motion coherence.

2 Related Work

The field of non-photorealistic rendering offers a good amount of attempts at curve-based stylization methods. Rendering algorithms, and according to Bénard's survey[4], they can be divided in two categories: texture and mark based.

When it comes to mark based approaches, we will focus on two papers with comparable process. Hertzmann et al. proposed an algorithm where the brush strokes are generated in a path that follows the directions with the least color difference[1] on a blurred version of the input. Huang et al. [2] had a similar approach, where the strokes are instead generated by a grid of features. The cells of this grid have variable sizes, and each one of them regroups small clusters of pixels with similar values. The stroke once again follows the neighboring paths with the least color difference. In both of these methods, the shape of the strokes can thus only be defined by the colors on the input 2D image. However, since the splatting method is always based on a 3D scene, we have a wider range of data to choose from. One possible solution would be to use the tangent information on each pixel. In that way, the brush strokes would follow the general shape of the object - the principal curvature of the object - which might be close to the painting process of a human artist.

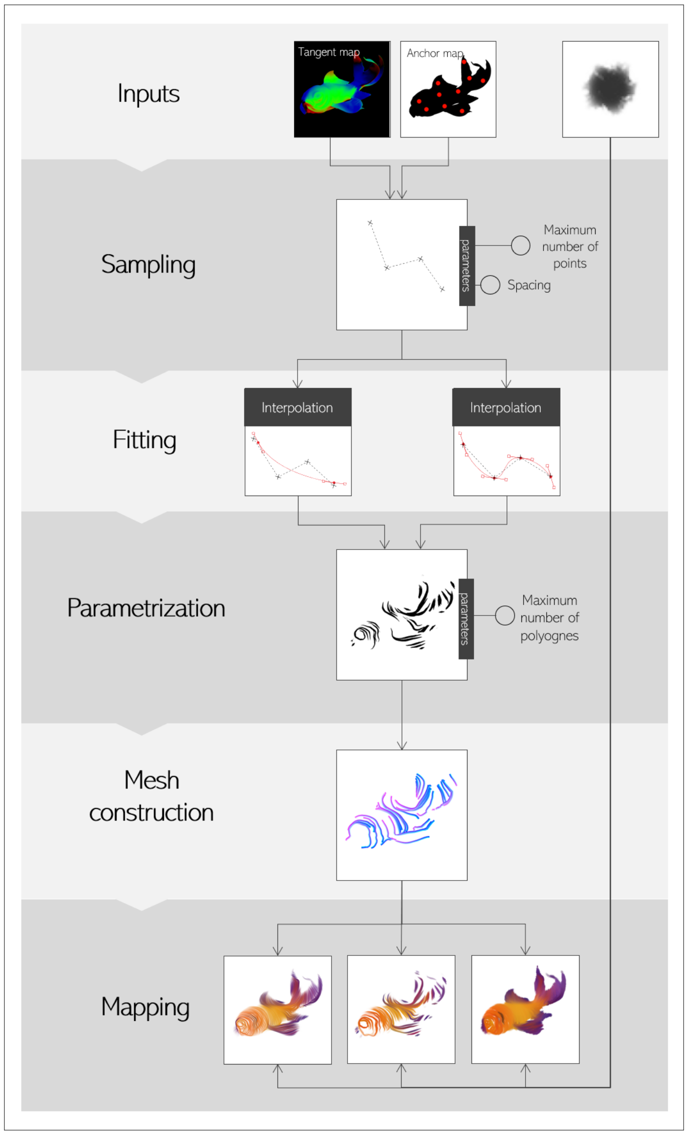


Figure 2: The sampling process with a high and a low density value

Schmid et al. [3] proposed a groundbreaking technique to draw paint-looking brushes in 3D space, by allowing the artists to draw 3D curves directly on the 3D geometry. These curves are transformed into stripes of polygons, which are then used to map the texture of the brush. The resulting meshes can then be rendered as any classic textured object.

However, the process is not automated, since every strokes needs to be hand drawn. Which means that the result cannot be easily animated. Additionally the curves are oriented in 3D space, which can limit the movements of the camera, and negatively impacts the flatness of the result

3 Method

The objective of our method is to recreate the scene using a set of brush strokes on a 2D canvas. Each brush stroke follows a path that represents the shape of an object on the screen. As such, these curves are constructed from points that are sampled from the surface of these objects. The process is done in for steps. First, for each anchor point given as an input, we need to sample a number of points in the 3D space that follow the general shape of the geometry. Second, these points are projected onto a 2D canvas, and transformed into a curve. Thirdly, we generate a stripe that follows the shape of the curve. Finally, we can map the input splat onto the shape in order to obtain the final render.

3.1 Point sampling

The sampling process is the first and most important step of our stylization method. In order to draw a curve, one must first determine the points it is going to follow. We must ensure that these points are coherent with the geometry, and remain coherent when the object moves or when the camera is animated. Additionally, since each anchor point is going to generate a curve, we must ensure that the silhouette cases - the cases where a curve cannot be fully drawn, if at all - are taken into consideration.

Description

Sampling is done in three steps: First, the 2D coordinates of the anchor point on the screen is projected onto the surface of the 3D object using the world position map. The resulting 3D coordinate corresponds to the point on the geometry of the object that appears on the position of the anchor point to the viewer. This point also carries the value of the tangent of the surface at that position. This tangent is then used to perform a "step" in the 3D space in that direction, with an offset d determined by the user [figure 4]. The resulting 3D step is then reprojected onto the 2D canvas. Since the whole process started with a 2D coordinate, we can re-iterate using that last point, and keep drawing the rest of the curve. A single step can be described with the following formula:

$$V_0 = V_{anchor}$$

$$V_x = M_{mvp} * (M_{mvp}^{-1} * V_{x-1} + T(V_{x+1}) * d)$$

where V_x is the x^{th} sampled point, V_{anchor} is the 2D position of the anchor on the screen, $T()$ the function that returns the value of the tangent map at a given 2D position, and M_{mvp} the projection matrix.

Once the process is complete, a series of 2D points is obtained, one that follows the geometry of the object in a curve-like pattern [figure 3].

Tangent map

The use of a tangent map was mentioned in the previous paragraph. The tangent map is a G-buffer that stores, for each pixel on the map, the tangent on the surface of the corresponding point in the 3D scene. These tangent vectors can be painted by the artist himself directly on the geometry, or globally computed from the normal map and a direction vector.

Fidelity and precision

The steps are performed using a user-defined offset. This offset determines how spaced out are the sampling points in 3D space. The density of the sampling points directly affects the shape of the curve [figure 3]: The higher the density, the closer the curve is to the actual geometry of the object. The reason for such a phenomenon arises from the re-projection step. Indeed, when projecting a 2D point into 3D coordinates and vice-versa, the projection is done on a vector that originates from the camera. [figure 4] The error gets higher the more the object's surface is curved, i.e the higher the difference between the normals of the origin point and the arriving point. The camera position is also critical in determining the intensity of the error. However, a high precision may negatively impact the flatness of the result. This is why this factor is left to the user's discretion, as these sampling errors might be desirable.

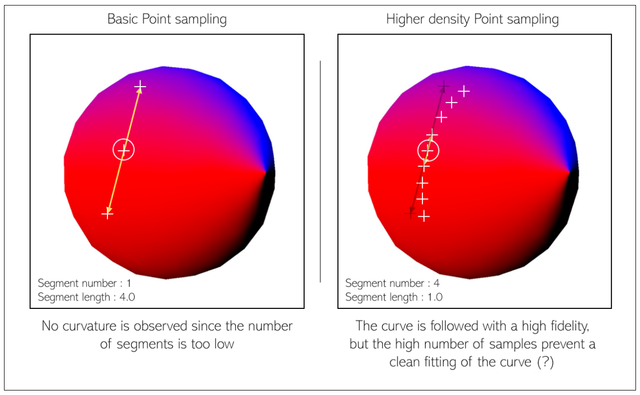

Basic Point sampling
Higher density Point sampling
Segment number : 1
Segment length : 4.0
Segment number : 4
Segment length : 1.0
No curvature is observed since the number of segments is too low
The curve is followed with a high fidelity, but the high number of samples prevent a clean fitting of the curve (?)

Figure 3: The sampling process with a high and a low density value

3.2 Curve fitting

Once the sampling parameters are satisfactory, the points need to be processed in order to obtain a continuous path. Note that there is no single method for processing a series of points into a path. The following methods are simply three different approaches with their own advantages and drawbacks. They have been chosen mainly for exploratory purposes, and alternative methods can always be proposed.

Polyline interpolation

This is one of the simplest ways to convert a series of points into a continuous line. The path is a compound of straight segments linking every two consecutive points. This fitting method offers the highest fidelity possible, since the path goes through every single point available.

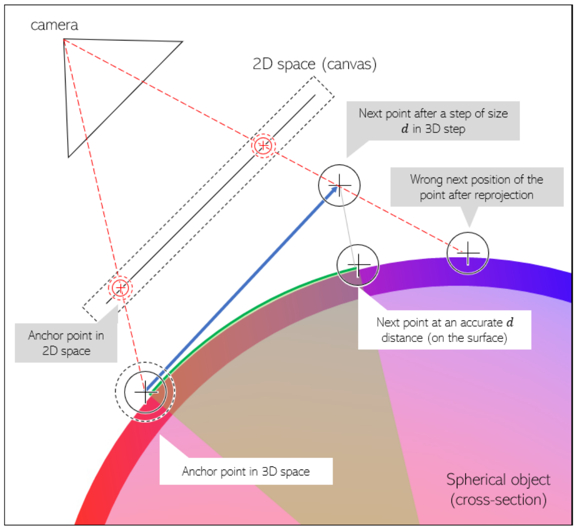

camera
2D space (canvas)
Next point after a step of size d in 3D step
Wrong next position of the point after reprojection
Next point at an accurate d distance (on the surface)
Anchor point in 2D space
Anchor point in 3D space
Spherical object (cross-section)

Figure 4: A 2D explanation of the sampling error on a cross-section of a spherical object.

Since the polyline has a high fidelity to the input data, the result is more satisfying with a high number of points, that are close to each other on the 2D space. Lowering the number of points can lead to apparent straight lines, which might be interesting only for a handful of art-styles.

3.3 Spline iterpolation

This time, the path is no longer a series of lines, but a series of cubic splines linking the points. There are many ways of obtaining a spline interpolation of a series of points, one of which is centripetal Catmull-Rom interpolation. The result is a continuous path where each point is an anchor point to two consecutive splines. Even though the splines are different, the formula ensures that the transition from one to the other is consistent, thanks to the usage of the previous anchor point as a control point. However, it is interesting to note that, by opposition to the polyline interpolation, the spline interpolation does not benefit from a high number of input points. Since each segment can result in the formation of one to two bumps, having to many points may lead to a path with a jagged appearance. Instead, it is wiser to reduce the number of input points, as well as spacing them out. On the other hand, we demonstrated in the previous section that increasing the distance between the sampling points may lead to an increased error factor. Thus, the correct way to proceed is to simplify the input data, by using a curve decimation algorithm such as the Ramer-Douglas-Peucker algorithm (see Algorithm 1).

This way, we ensure that the remaining points are still on the geometry. The number of iterations of the algorithm and the intensity of the decimation is once again left to the user [figure 6].

Polynomial curve fitting

Another way to proceed would be to rely on an approximate function instead of an interpolation. The main difference is that, contrary to an interpolation, an approximation does not necessarily pass by all the points. Its shape is instead defined by the rough distribution of the available data.

Algorithm 1: Ramer–Douglas–Peucker algorithm

```
Result: A decimation of the input set of points
dmax = 0;
index = 0;
end = length(PointList);
for i = 2 to (end - 1) do
    d = perpendicularDistance(PointList[i],
      Line(PointList[1], PointList[end]));
    if d >dmax then
        index = i;
        dmax = d;
    end
    ResultList[] = empty; // If max distance is greater
      than epsilon, recursively simplify;
    if dmax >epsilon then
        // Recursive call;
        recResults1[] =
          DouglasPeucker(PointList[1...index],
          epsilon);
        recResults2[] =
          DouglasPeucker(PointList[index...end],
          epsilon);
        // Build the result list;
        ResultList[] =
          recResults1[1...length(recResults1) - 1],
          recResults2[1...length(recResults2)];
    else
        ResultList[] = PointList[1], PointList[end];
    end
    // Return the result return ResultList[]
end
```

3.4 Splat aspect

Parametrization

The final step is to find a satisfying parametrization of the path in order to map a 2D texture on it. While there are many approaches to this problem, the objective is always to generate a 2D map where each pixel corresponds to a position on the final render.

The parametrization process is always roughly the same, and can be sumarized in the following formula:

$$\alpha_x = \frac{x}{N_{segments}} * N_{points}$$

$$V_x = P_{\lfloor \alpha_x \rfloor} + \varphi(C_{\lfloor \alpha_x \rfloor}, \{\alpha_x\})$$

where N_s the number of segments, N_p the number of polygons, P_x the x^{th} sampled point,and $\varphi()$ a local parametrization function that depends on the nature of the chosen fitting. For example, $\varphi_{polyline}()$ is a simple linear equation, while $\varphi_{spline}()$ is a cubic curve parametrization function.

Splat Mesh

We use the parametrization function to build a mesh that follows the path previously computed.

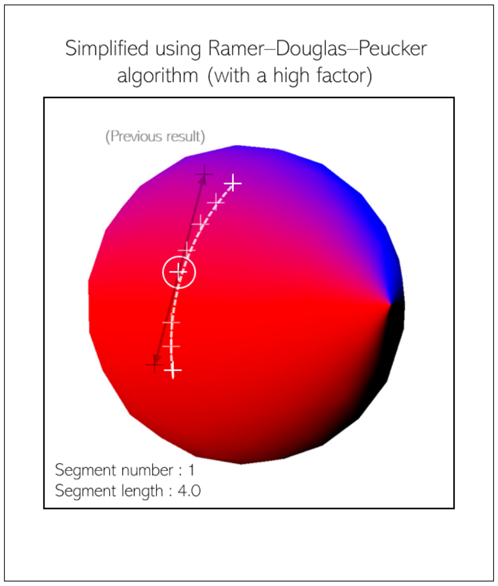

Simplified using Ramer–Douglas–Peucker algorithm (with a high factor)
(Previous result)
Segment number : 1
Segment length : 4.0

Figure 5: The resulting three sampling points after a Ramer-Douglas-Peucker decimation compared to the same 3 points obtained after a basic sampling

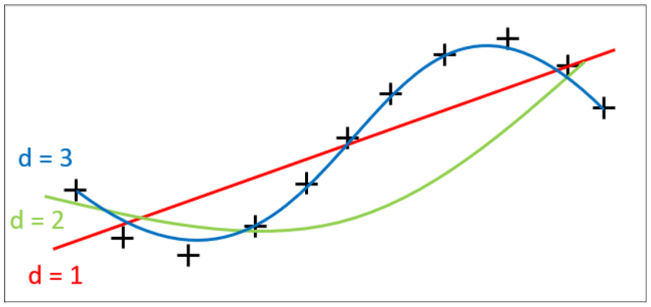


Figure 6: Fitting attempts with curves of different degrees

The number of polygons can be determined by the user, and affects the performances of the algorithm and the smoothness of the end result.

Mapping

Each vertex in the generated mesh has an UV value, that maps it to a point in the 2D texture of the splat. From that, many mapping options are available. The first and most basic method would be the simple mapping. It stretches the splat on the mesh with no further processing. This mapping is extremely cost efficient but may lead to the formation of undesirable artifacts due to the low resolution of the splat image. One other way to proceed would be to use the basic stamp. [ref] The idea is to keep the aspect ratio of the input image, and recreate the shape of the mesh by continuously repeating it and blending the consecutive sprites with one another. This technique is most commonly used in digital painting software.

The whole brush construction pipeline is summarized in [figure 2].

4 Implementation

The whole stylization process runs entirely on the GPU, using OpenGL 4.3 in the node-based compositing system "Gratin" [[5]]. The generation of the anchor points is done on a compute shader. The anchor points are stored in an array which is then passed to a vertex shader. In my implementation, the vertex shader serves no purpose beside liking an anchor point to a single vertex and fetching its attributes. This attribute list can contain the position of the anchor point in 3D and 2D spaces, its color, depth, opacity, and so on. These vertices are then passed as primitives to a geometry shader. The geometry shader is tasked with creating a mesh for each anchor point on the screen, which means that every step from the sampling to the parametrization is done in the geometry shader. The geometry shader will also assign, to each vector, a uv coordinate. Since these coordinates are interpolated between the vertices, we can directly call the fragment shader and map the splat with ease.

5 Limitation

Although the process is done in interactive time, it still suffers from performance issues that prevents it from running in real time. The use of a geometry buffer forces the shader to rely heavily on polygons, the number of which is set to an upper limit by the GPU. Moreover, since all the inputs are in 2D space, some of the spacial information is ineluctably lost, either because the corresponding points are hidden behind other objects or outside the view space.

Silhouette cases

Some of the anchor points are generated near the silhouettes of the objects they are attached to. In most mark-based stylization methods, this is one of the main causes of popping. Popping is an umbrella term describing any sudden appearance, disappearance or jittery movement that a mark can make from one frame to another. A stylization process with a lot of popping can be considered to have a low motion coherence, since the movements of the marks lack in fluidity, or are no longer characteristic of the movement they are supposed to be describing.

Our method is not immune to this issue. Since the number of sampled points is defined by the used offset as the precision factor, the sampler will often find itself in a situation where no spacial value is available, since the projected tangent lands on a pixel outside of the object's silhouette. When such a case arises, it is once again left to the user whether or not the lines should stop at the silhouette, or overshoot the boundaries of the object.

Another issue that arises with our sampling method is quite similar to the silhouette cases. Since the sampling is done on a 2D image, it is possible that the projected tangent lands on a pixel that does have a value, but that represents a different object than the one the anchor point originated from. Furthermore, two sampled points might land on the same object, but with depth or color discontinuities that are too significant to be painted on the with the same brush stroke.

6 Results

Using our method, it is possible to obtain a wide range of art styles by varying the input parameters. [figure 7] is a demonstration of our stylization method on simple sphere. The images on the top row are generated with a low mark density for clarity. The first render is generated using a procedural stroke that changes in thickness as it gets further along the curves. The curves themselves follow a normal directional map: the directional vectors that are perpendicular to the surface of the object. The second one in a simple attempt at a oil-paint art style. The marks are this time made to follow the latitude lines of the sphere, and assume the aspect of a pain brush stroke.

[figure 8] showcases how we can easily stylize a 3D model into a 2D-looking painting. The strokes are thick and the curves are short, which is supposed to emulate an oil-painting style. The figure also show the same object, but where the parameters dictate more precision in the rendering of the lines. The lines closely follow the geometry, which looks closer to what an artist would do with a pen.

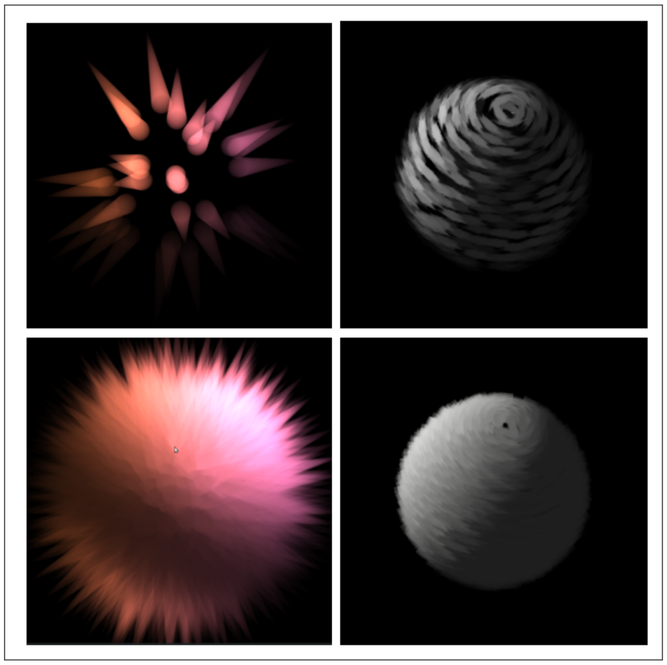


Figure 7: Stylization of a simple sphere using different direction maps : On the left, a normal map, on the right, an horizontal tangent map.

7 Future work

My internship still extends for a a month. I wish to tackle the approximation errors during the sampling process. Moreover, there are still a few popping issues that would require some attention. The idea would be to lower the opacity of the curves as they approach the limits of the object. The usage of curves as the main stylization technique also extends the possibilities of animation stylization, particularly when it comes to motion lines. Overall, this technique still offers a lot of possibilities that need to be explored, notably in terms of input parameters and possible art styles.

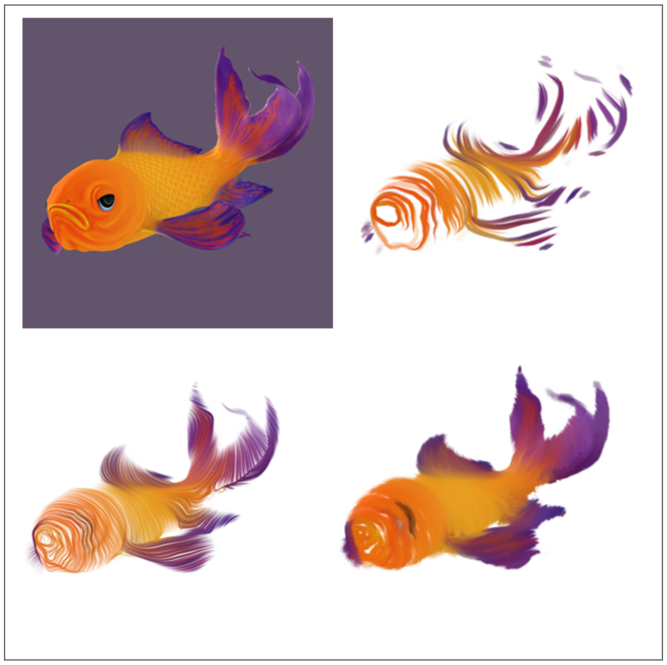


Figure 8: Stylization of a fish model with different thickness, length and precision parameters

8 Academic Acknowledgments

I would like to extend my sincerest gratitude to Joelle Thollot and Romain Vergne, who accepted to supervise my internship for the second time. I would also like to thank Maxime Garcia, Pierre Bénard, and Maxime Isnel, without whom this work would not have been possible. Lastly, I would like to thank the whole MAVERICK team for their warm welcome among their ranks, and all the advice and encouragements they've given me throughout these two years.

References

[1] HERTZMANN, A. Painterly rendering with curved brush strokes of multiple sizes.

[2] HUA HUANG, TIAN-NAN FU, C.-F. L. Painterly rendering with content-dependent natural paint strokes. *Springer-Verlag 2011* (2011).

[3] JOHANNES SCHMID, MARTIN SEBASTIAN SENN, M. G. R. W. S. Overcoat: An implicit canvas for 3d painting.

[4] PIERRE BÉNARD, A. B., AND THOLLOT, J. *Computer Graphics Forum*.

[5] VERGNE, R., AND BARLA, P. Designing gratin - a gpu-tailored node-based system. *Journal of Computer Graphics Techniques (JCGT)* (2015).

Relationships between Scheduling and Autonomic Computing Techniques Applied to Parallel Computing Resource Management

Manal BENAISSA
Master 2 MOSIG Data Infrastructure
Université Grenoble Alpes

Supervisors: Raphaël BLEUSE and Eric RUTTEN
Team CTRL-A

This Masters research project will be defended before a jury composed of:
Bruno RAFFIN (President of the jury)
Martin HEUSSE (Examiner)
Olivier RICHARD (External Expert)
Raphaël BLEUSE (Supervisor)

Abstract

The scheduling field regroups various methods by which work is distributed across available computational resources. Considering the complexity of modern infrastructures, particularly in Cloud and HPC computing, schedulers might face difficulties to propose an efficient scheduling while fitting to the system reality and it implied uncertainties. The autonomic computing field suggests a more practical approach, by controlling constantly a system and adjusting taken decisions at runtime, via feedback loops. Applying this strategy in the scheduling context bring a less complex model with a more modular structure. Likewise, scheduling community may bring a new vision of autonomic computing challenges. This work presents some possible models, from the most basic scheduling algorithm, to the most complex Cloud infrastructures.

***

Le domaine de l'ordonnancement regroupe diverses méthodes, où le travail est distribué à travers les différentes unités de calcul disponibles. En considérant la complexité des infrastructures actuelles, particulièrement dans le domaine du Cloud et du HPC, les ordonnanceurs peuvent rencontrer des difficultés à concilier la proposition d'ordonnancements efficaces et les incertitudes liées à la réalité du système. Le domaine de l'informatique autonome propose une approche plus pratique, en contrôlant régulièrement un système et en ajustant les décisions prises durant l'exécution, par l'intermédiaire de boucles de rétroaction. Appliquer cette stratégie dans le contexte de l'ordonnancement apporte un modèle moins complexe et une structure plus modulaire. De même, la communauté de l'ordonnancement pourrait apporter un regard nouveau sur les problèmes que rencontre la communauté de l'informatique autonome. Ce travail présente quelques modèles possibles, du plus basique algorithme d'ordonnancement aux plus complexes infrastructures Cloud.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

1. CONTEXT

Cloud and HPC computing are both based on a complex infrastructure composed of servers, data storage units, network connections and even software, dedicated to users. The system offer platform access as a service, for personal or professional purpose in Cloud computing, and mainly for scientific purpose in HPC context. More and more technologies in these fields need complex and time-consuming computations, which require to parallelize several tasks in highly complex infrastructures. Using classical scheduling algorithms that only take in account the amount of work and the number of compute units is not realistic in such systems. In fact, many other parameters like the cost of communications, dependencies between tasks or elasticity of the infrastructure bring complications in the search of optimal scheduling. Typically, Cloud infrastructure is generally composed of tens of data-centers scattered around the world and inter-connected by wide area networks (WAN). Every days, each data-center has to process some data, and execute many jobs. Most of the time, these jobs are dependent of each others, and need data from a far away data-center. Scheduler function is to distribute these jobs across different resources, taking into account Quality of Service and infrastructure constraints.

In 2000s, IBM proposed the *autonomic computing* approach to address such challenges. A controller is added to monitor the system and adjust decision when this one is no more adapted. This implies a constant checking of the system state and the computation of an adapted response of a non desired evolution, forming a feedback loop. Even if this strategy exists in Cloud computing, it is not always clearly defined. Most of the time, a static analysis is done, despite uncertainties due to the overall architecture. Nevertheless, the control loop concept can even be found in HPC systems, where the adopted approach is based on a more theoretical performance analysis. In this work, after a brief definition of each community politic, some possible strategies that promote collaboration between scheduling and autonomic community will be explored.

2. STATE OF THE ART: SCHEDULING

More and more computer science fields require to solve complex and time-consuming problems in short time, and the idea of dividing these kinds of problems in small blocks distributed across several compute units came quite intuitively. However, the parallelization of algorithms implies other problems like dependencies between these blocks, distribution between all available compute units, time and resource constraints and so on. R.L Graham [1] was the first to formalize scheduling constraints and to propose an approach in 1966 with the *List Scheduling*. After that, many strategies were explored, particularly in operational research, HPC and Cloud computing.

2.1 Scheduling problem

Scheduling problems are a part of constrained optimization problems, widely known in operational research field. Scheduling regroups methods by which work is distributed across available computational resources. Infrastructure is considered by following a model where work can be a task (block of code in program) or a job (an entire program). This work can be distributed across physical nodes like CPU, machines or even clusters, or logical nodes like threads.

Hence this notation allows to formalize the associated problem, by considering three aspects:

α the state of the system where parallelism is applied: Number of compute units, and whether all of them are identical.

β constraints on work: Number of task/job, dependencies between them, and processing time.

γ desired objective: minimizing execution time over all tasks, maximizing the total amount of work per time unit, or limiting lateness when a deadline is defined for example.

Graham gathered all these constraints in a notation system, the Graham notation [2]. In this work, only the main criteria will be described. SchedulingZoo [3] gathers a more complete description about this notation. Here, $< \alpha|\beta|\gamma >$ is chosen depending of the criteria described in Table 1 (this table is only an extract of the Graham notation).

For example, to describe a scheduling problem that minimizes the execution time and work on n identical parallel nodes, with k jobs linked by precedence constraints, we will write:

Machine environment α	
1	Single node
2	Two nodes
m	m nodes
P	All nodes are identical, aka execution time is the same, no matter the chosen node
R	Nodes are unrelated, which means execution time is not the same
Constraints β	
prec	Dependencies exist between tasks, which means task T1 has to be finished before starting task T2, if T2 is dependent of T1.
$pj = p$	Each task has the same processing time p.
pmtn	Allows preemption, tasks can be suspended and continued later, possibly by an other node.
Objective function γ	
C_{max}	Makespan, the total execution time between the start of the first job and the end of the last job.

Table 1: Criteria in Graham notation [2].

$$< Pn|prec|C_{max} >$$

A way to visualize this problem is to consider n identical nodes P_i, $1 \leq i \leq n$ and m tasks T_j, $1 \leq j \leq m$ with dependencies. These dependencies are specified with $\prec$ order relation: $T_i \prec T_j$ if T_j is dependent of T_i, which means that T_j can not start its execution if T_i is not finished. Each task needs a certain amount of time to be executed, given by the function μ. With this data, we obtain a dependency graph $G(\prec, \mu)$, where each vertex indicates a task and its processing time, and each edge gives existing dependency between two tasks.

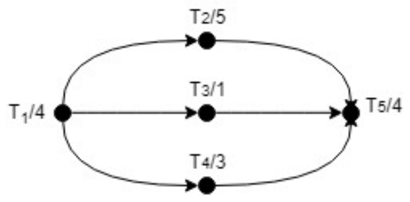


Figure 1: Dependency graph example [1]

For such a problem where the set of tasks and execution time are given, solutions exists and they are well known. Given a dependency graph $G(\prec, \mu)$,a certain number of computing nodes n and possibly some other elements, a Gantt diagram ξ can be obtained (cf Figure 2). This diagram will depend on the following algorithm, and must respect all specified constraints.

Online Scheduling

In online scheduling, the scheduler receives jobs that arrive over time, and generally must schedule the jobs without any knowledge of the future. The lack of knowledge of the future precludes the scheduler from guaranteeing optimal schedules[4]. The easiest way to understand scheduling is to imagine a finite set of tasks where we know duration of each tasks. These problems where all constraints are fixed are well known. However, many systems don't know in advance

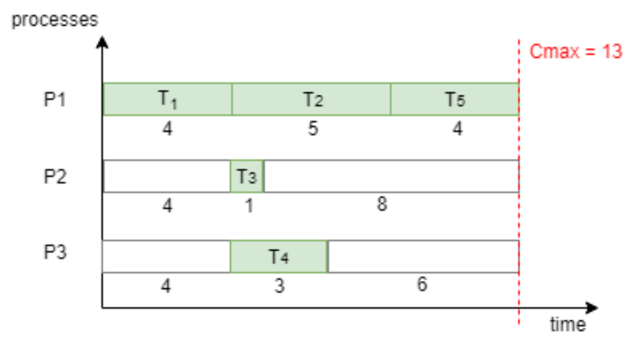


Figure 2: A possible Gantt diagram[1] of the dependency graph in the Figure 1, for $n = 3$

neither the number of tasks, nor their duration. In Cloud context for example, each client request come as a job, and they might arrive one by one as an input stream. On top of that, we distinguish non-clairvoyant problems which job computation time is not known. Despite this lack of information, solution exists and they are based on offline algorithm approximation. More precisely, we compare an online algorithm A to an optimal offline algorithm OPT that knows the entire input in advance.

Clairvoyant.
Clairvoyant scheduling constitutes a part of online scheduling, where processing requirement of each job is specified, but existence of jobs stay unknown until a certain *release time.*The jobs characteristics are not known until they arrive, restricting the scheduler to schedule jobs only with current information. However, once these ones are released, the job size (μ function) is known.

Non-Clairvoyant.
In more realistic models, the processing requirement of each job is also unknown, and can only be determined by processing the job and observing its execution time.

2.2 Scheduling algorithms

2.2.1 Offline scheduling

In the offline scheduling context, the *List Scheduling* is the most intuitive solution, particularly for the $< Pn|prec|C_{max} >$ problem described in section 2.1. Given a dependency graph $G(\prec, \mu)$, a certain number of processing nodes n, and an ordered list L, an optimal scheduling can be obtained by following some rules. We consider that a task T_j is available, and can be executed by a node if:

- It is not taken by an other node.
- It is not dependent of an other task.
- It is dependent, but all preceding tasks are already executed and finished.

The processing node P_i takes the **first available** task T_j in the list L. If no task is available in L, P_j become idle, by executing an empty task φ_k. This task ends when an other task finishes in an other node. We obtains the following Gantt diagram in the figure 3.

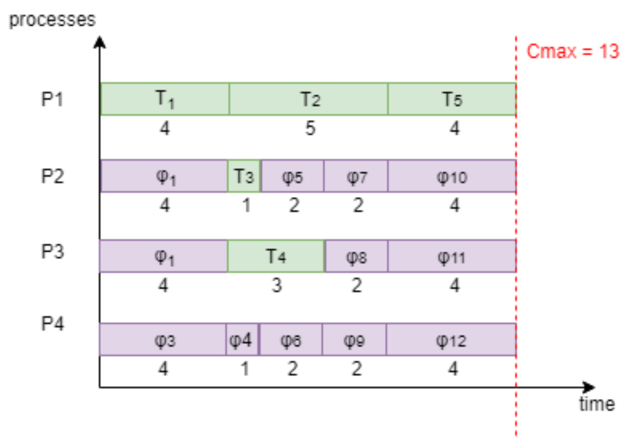


Figure 3: Gantt diagram[1] of the dependency graph in the Figure 1, for $n = 4$ **and** $L = [T_1, T_2, T_3, T_4, T_5]$

FIFO scheduling is the most used algorithm in Cloud computing. It's a variant of the List Scheduling, where the ordered list L is fixed: tasks are treated with the order they arrive. They are ordered in L by they release date, and the first task to be executed is the one with the earliest release time.

2.2.2 Online clairvoyant scheduling

To adapt offline solutions to an online model, Schmoys described in 1995 a method to converts algorithms that need complete knowledge of the input data into ones that need less knowledge. The idea is to use an offline algorithm (such as List Scheduling) to schedule a subset of jobs, released at each time interval. The system is defined by a set of computing nodes and a waiting queue for arriving jobs.
At time $t = 0$, a certain number S_0 of jobs was queued in the system queue. At $t = 0$, a snapshot of the queue state is taken by the scheduler. Since we are in a clairvoyant scheduling, we know the execution time of each jobs, and then we can estimate C_{max} of the S_0 scheduling. The offline scheduler schedules only these S_0 jobs in the queue, until this queue is considered empty by the snapshot view. In the same time, some jobs arrive in the queue and wait until S_0 schedule ends. These waiting jobs compose S_1 session. So when S_0 schedule ends (i.e at $t = C_{max}(S0)$), S_1 jobs are captured in an other snapshot and they are scheduled while S_2 jobs arrive in system queue. In the Figure 4, the release of S_0, S_1 etc (in red) corresponds to the moment when the snapshot is taken by the scheduler. Only jobs taken in each snapshot are scheduled (in green), the other arriving jobs wait until the next session.

2.3 Resources and Jobs Management System

Resources and Jobs Management Systems (RJMSs) [6] [7] are specific software specialized in distribution of computing power to user jobs within a parallel computing infrastructure. RJMS has a key role in HPC infrastructures: it collects all requests from users and scheduling them, while managing all resources of the system. An application (or executable) is grouped with some data like an estimation of the computation time and needed resources. All applications are queued to be scheduled next. The RJMS must manage these two part, to satisfy users demands for computation on one hand,

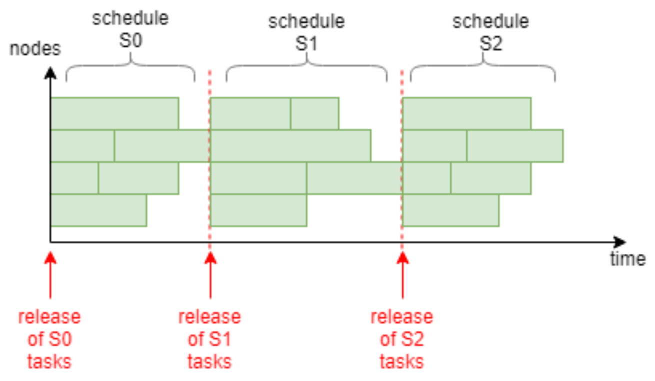


Figure 4: Shmoys algorithm [5]

and achieve a good performance in overall system utilization by efficiently assigning jobs to resources on the other hand.

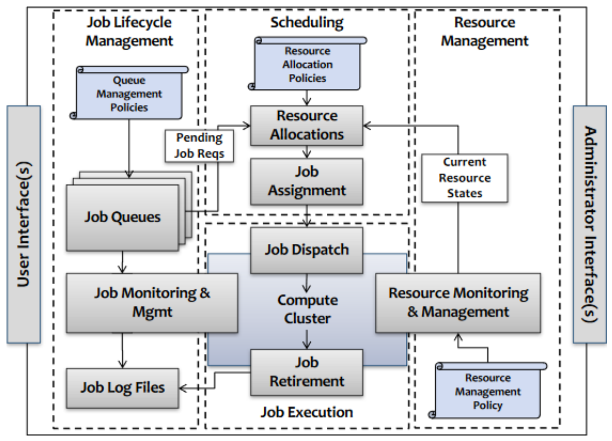


Figure 5: RJMS in HPC infrastructure [8]

The work of a RJMS is classically divided in three parts [8] [7] :

- Transforming a user request into an application, where all needed characteristics are specified.
- Scheduling these applications.
- Placement and execution of these applications in their allocated resources.

These three parts are respectively managed by the jobs manager, the scheduler and the resources manager (cf Figure 5).

2.3.1 *Resource Manager*

This part is responsible to collect and provide all information concerning computing nodes. This information is used by the scheduler and by users to inform about the availability and the state of the cluster. Resource Manager is responsible of these tasks:

Resource Treatment This part collect information about cluster structure and node characteristics. In fact, clusters are currently heterogeneous systems with different hardware components and different levels of hierarchies inside one node (cpu, core, thread...), for a given network.

Resource Deployment Resource Manager takes information about jobs from the Job Manager and the Scheduler, and initializes required resources.

Task placement Nowadays, a single computing node can have multiple cpu. In Cloud context particularly, since several users can share the same computing node, tasks need to be placed correctly, to avoiding collisions.

Resource Management Advanced Features Includes faults control and energy optimization.

2.3.2 *Job Manager*

Job Manager manages tasks related to the declaration and control of users jobs. Its role is decomposed in these functions:

Job declaration Users use RJMS interface to describe their jobs characteristics and select the resources of their preference. A job can be *interactive* (user want to be directly connected to the node, to launch his experiment manually), a *batch* job (used for direct script execution upon the allocated computing node), *bulk* jobs (one large job can be decomposed into smaller jobs, for a better scheduling), *best-effort* job (this kind of job has a low priority and can be killed if a normal job demands the resource) etc...

Job Control users can keep control on submitted jobs that are not executed yet, by changing initial parameters (priority, input/output files...).

Job Monitoring and Visualization Allows users to follow the execution of jobs upon the cluster.

Job Management Advanced Features Includes job preemption, faults control and security features.

2.3.3 *Scheduler*

The scheduler constitutes the main part of the RJMS: it assigns user jobs with chosen parameter to available nodes that match with the demands.

Scheduling Algorithms Different kind of scheduling policies (described in the section 2) can be used, but the most common in this context is FIFO, with which some optimization like the *backfilling* are possible.

Queues Management All tasks are placed in one queue or more, waiting to be scheduled. These tasks can be grouped in a specific queue when they sharing some similar characteristics. The scheduler takes task in these queues, depending of its policy. An important point is to avoid famine: tasks in the lowest priority queue might be never executed if the highest priority queue is always filled. A solution to bypass this problem is to set an increasing priority depending on the waiting time.

3. STATE OF THE ART: AUTONOMIC COMPUTING

Despite all efforts to optimize the system robustness, anticipating non desired behavior in complex infrastructure without controlling it regularly can be tricky. The notion of autonomic computing[9] was introduced by IBM in 2001, but the concept of feedback loop was commonly used in engineering. The aim was to tend toward a fully autonomous infrastructure, with a self-management of work. A security is added to control periodically system outputs and adjust inputs when these ones exceed permitted values. A common example is temperature control in machines: A controller will check constantly temperature evolution. When this one indicates an overheating, the controller adjust machine behavior by increasing its cooling system or decreasing energy consumption. Adequate responses to an anomaly are not necessarily based on a solid knowledge of the system, only few elements are enough. In this way, a feedback loop provides several advantages without changing all the system. Some models exist but we will focus on one of them: MAPE-K [10].

3.1 MAPE-K

The main characteristic of Autonomic computing is to offer self-management. The goal is to exempt system administrators from maintenance and to allow a continuous system working without a drop of performance. Autonomic systems adjust their behavior in the face of external changes. This characteristic includes:

Self-configuration Adding a new component in the system should not involve difficult manual operations. Instead, it must incorporate itself seamlessly, and lets the rest of the system to adapt to its presence.

Self-optimization The whole system has to improve its operations, in order to be more efficient. This process go through a learning phase, where the system monitors, experiments and adjust its behavior until a certain optimization level is reached.

Self-healing The system can detect and repair failures.

Self-protection The system must be prepared to handle malicious attacks or accidental failures, by anticipate them.

To fulfill these objectives, a controller is grafted to the managed system, forming a feedback look (cf Figure 6). The aim of the controller is to ensure that the system converges toward the desired behavior by controlling system when inputs are out of bounds, and without too large fluctuations in responses. To ensure that, we will focus on one model: *MAPE-K*. This model is divided in three main parts:

Monitor This part collects data y_k at time k, via sensors, and possibly makes some pre-processing. The main challenge here is to know collecting data frequency.

Analysis This part includes data analysis by making some estimations.

Planning This part makes an adequate action plan to adjust the system. To do that, controller follows control laws based on the knowledge of the controlled system. These control laws can follow various models based on machine learning, rule-based systems or even basic programming.

Executor This part applies decision-maker action plan, by sending data to modify u_k to the system, via actuators.

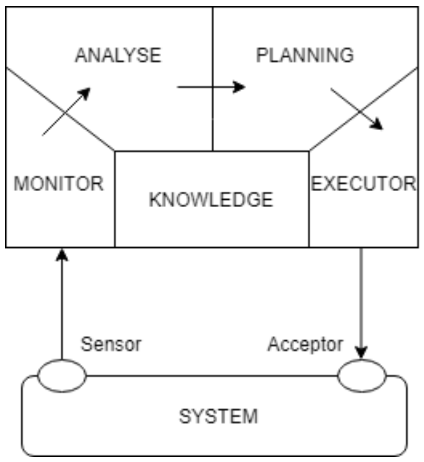


Figure 6: MAPE-K loop for control[10].

3.2 Control Theory as MAPE-K

On top of the MAKE-K model, control laws can be draw from Control theory[10] ideas. As we said before, the decision part can be based on various models, depending on the system needs, and the field of automatic control provides many potential control laws. The main interests of this field are:

Stability The system is considered as *stable*, if its inputs and disturbances stay bounded. Simply speaking, the system must converge toward the wanted behavior, by detecting inputs out of its bounds. With this approach, controller will naturally stabilize system, even when this one is instable. In fact, when inputs stay bounded, output and state will stay bounded too.

Robustness Because the system is periodically checked, with few amount of time between each control, an anomaly is quickly detected. With this method, we don't need all details about the system evolution. Only few data but regularly collected allows to correct system with enough precision.

Tracing of performance Controller checks *first* and adjust *next*. Because of that, we can draw evolution of the system at any moment.

Since the decision part is the controller core, and it will define the efficiency of this one. A bad decision will impact directly the system, and it's why the choice of control law is important. However, even if systems are different, in most case they follow the same rule: Only one variable is collected, and if its value falls outside the accepted range, only one other variable has to be changed. In the other hand, the chosen model has to be simple. A complex model will take time to compute and slow down the whole system. For these reasons, the *PID control law* (for Proportional, Integral, Derivative) is the most used. The u_k value is written as a function of the error signal $e_k = r_k - y_k$, where e_k is difference between the measured value y_k and the expected value r_k (also called *reference value*), at time k. We obtain u_k, expressed by the function:

$$u_k = K.e(t) + KT\frac{d}{dt}te(t) + KT_i \int_t^0 e(x)dx$$

In this function, we can identifies three terms:

Proportional term $K.e(t)$ where K controls the the rising time.

Derivative term $KT\frac{d}{dt}te(t)$: this part, based on change frequency, absorbs the oscillations and overshoots.

Integral term $KT_i \int_t^0 e(x)dx$: this part "memorizes" old values, to nullifies the static errors.

All factors (like K) are chosen depending on the system, by testing different values and estimate the most adapted ones. In many simplest models, only the proportional term is used and completed by derivative and integral term when it's required. Of course, PID model is not always sufficient, particularly if system behavior cannot be expressed by linear models.

4. PROBLEM STATEMENT

Scheduling computing field gathers many optimization methods to distribute work across all resources. As we saw in Graham notation, a precise representation of the system should be relevant (number and characteristics of computing nodes, description of the work...). But this representation reflects partially the reality of the system, despite a worse case analysis. Dues to this partial capture of the reality, the representation implies uncertainties. This variability, mainly explained by the architecture, can be captured with an online approach. Autonomic computing community suggests a more practical strategy, by adapting to the system. It gives a compromise between a more simple model but lacks of performance and stability guarantees.

In this way, autonomic computing and scheduling computing stay two distinct fields in computer science. However, scheduling community would benefit from control loop concept, autonomic computing community can use scheduling techniques as decision-maker in controller, and the merge of these two domains deserves further consideration. Scheduling community uses worst case analysis to evaluate a scheduling method, but these worst case can be avoided with a control loop. Some scheduling system already includes feedback loops in the structure, but in an implicit way. Highlighting these control loop in such a system will allow to fully exploit autonomic computing methods and formalize properly this kind of approach. In this way, autonomic computing community can bring a better expertise in scheduling computing field. In this paper, we will present some case where we can bring out a feedback loop in a scheduling system, by identifying each part of the potential MAPE-K pattern.

5. COMBINING SCHEDULING AND AUTONOMIC MANAGEMENT

The intersection between scheduling and autonomic computing field can be seen in different ways. In this part, we study existing cases and re-interpret them through the prism of autonomic computing. Each example will focus on a certain abstraction level, from the fine grain at the scheduler level to the Cloud infrastructure.

5.1 Feedback loop in a scheduling algorithm context

A first approach in the merge of these two domains is to reinterpret an online scheduler as a controller with MAPE-K pattern and scheduling policies as control laws. To do that, we will use Shmoys[5] technique seen previously: This method consists to use an offline scheduler at each cycle, defined by the state of system queue at a certain time. The controlled system is defined by a set of computing nodes and a waiting queue of jobs. In the clairvoyant context, released jobs and their execution times are known. Let us consider sensors that capture queue snapshots described in the section 2.2.2, allowing a monitoring of the queue state. An execution time C_{max} can be estimated (in the analysis part) and a decision (the scheduling process itself) can be made. This decision is injected directly on the nodes set via implicit actuators, thus closing the loop (cf Figure 7). MAPE-K pattern is clearly defined here, where the chosen control law is an offline scheduling algorithm.

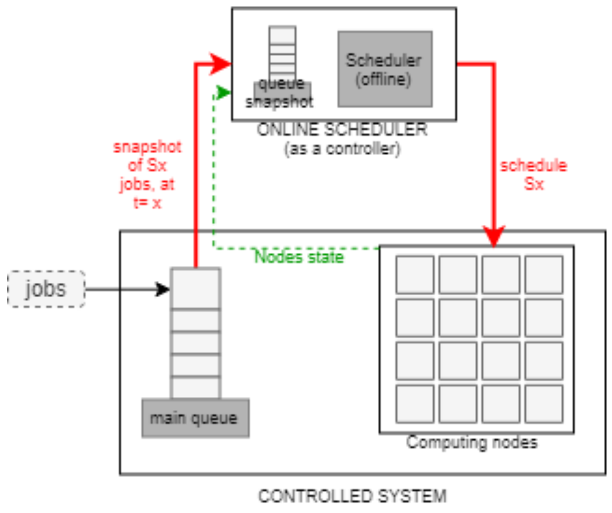


Figure 7: Online scheduler as a controller. Dotted arrows corresponds to possible additional monitoring. Plain arrows constitute the main control loop.

This approach can be adapted to pure online algorithms, by considering a sensor in the system queue and actuators that apply scheduler decisions. Capture frequency can be adjusted, depending on the computing cost of the scheduling algorithm and the jobs themselves. In the case above, capturing a snapshot not at each C_{max} but earlier may end up in a better result. It also may interesting to capture more information during monitoring, like the state of each node (green dotted arrow in the Figure 7). Knowing if some nodes crashed, or if they are overloaded may lead to a finer analysis, and thus to a better decision.

5.2 Feedback loop in a RJMS context

A scheduler can be interpreted as a closed loop, as shown in the previous section. It is therefore natural to consider feedback loops at the RJMS level, not only in the scheduling part itself but at the jobs and resources management level too. This approach was adopted in the HPC context, to exploit unused resources in a cluster. Grid5000[11] is a grid for computing experiments with more than 5000 cores dispatched in 11 sites, mainly in France. OAR[12] is its associated RJMS, highly configurable and based on a batch scheduler. Like many RJMS, OAR can use many queues to place user tasks, depending on the characteristic and the priority of the submitted task. This strategy allows to respect QoS constraints, while exploiting the maximum of the clusters capacity. However, some resources can remain unused despite these efforts. It's why CiGri was implemented in top of OAR.

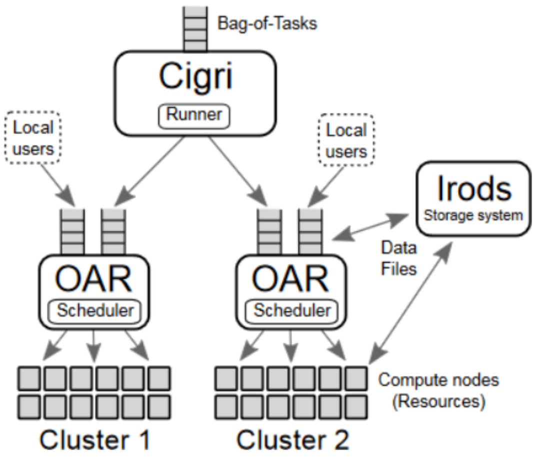


Figure 8: Overview of Grid5000 RJMS[13].

Each OAR scheduler is configured with a dedicated queue for CiGri. When a user submit a task, this task can be placed directly in OAR priority queue, to be executed in the best conditions possible. Otherwise, it can be placed in the CiGri queue, particularly if the task is small and it has a low priority. In this way, CiGri can place these numerous small tasks on the remaining resources. At each cycle, CiGri submits a number of tasks called *rate* to the clusters, and wait the end of the execution of all these tasks. Then, in the next cycle, Cigri submits an other number of tasks, with an higher *rate*, and wait. This rate increases at each cycle, depending on clusters availability. This method can be seen as a tasks flow, where CiGri is the tap. This flow is nevertheless mismanaged:

- CiGri submits tasks to OAR scheduler only if OAR priority queue is empty. Therefore, resources might remain unused while tasks in CiGri queue are still waiting (cf red dotted arrow in Figure 10).
- Clusters (or the storage system) are overloaded because too many tasks are submitted by CiGri.

To avoid these problems, a feedback loop was implemented (in green in Figure 9). This control loop reuses all MAPE-K and control theory concepts by monitoring the load in clusters queues and controlling the number of tasks submitted by CiGri. We recognize all parts of the MAPE-K pattern, with:

Monitoring The number of tasks y_k in clusters queues is measured.

Analysis The gap e_k between y_k and the expected value r_k is computed, giving the idea of CiGri behavior.

Planning Depending on e_k, more or less tasks will be submitted by CiGri. To know how many tasks u_k CiGri should submit, the controller use PI control law.

Executing At the end, the final decision is injected to CiGri.

These decisions are drawn on system knowledge that give the controller sensibility and responsiveness. This knowledge is mainly based on experimentation. Adding a controller in the system permitted to gain 8% in clusters usage, and increase efficiency of resources management.

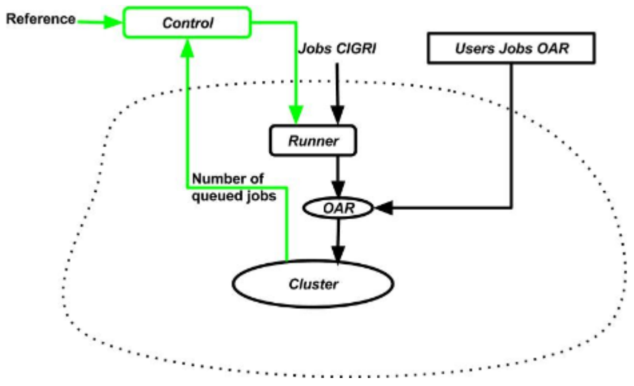


Figure 9: Feedback loop in CiGri cluster[13].

Another way to see this infrastructure is to consider CiGri as a single controller, and OAR as a basic RJMS. The controller seen previously becomes a integral part of CiGri: The number of tasks in clusters queues is monitored as before, but here, the number of submitted jobs in OAR low priority queue is directly monitored (cf. Figure 10).The system knowledge of this version is not static anymore, as long as the number of jobs in CiGri queue changes. However, we saw in the previous section that a scheduler can be seen as a controller too. Similarly, OAR can be interpreted as a feedback loop, where jobs and resources are monitored to influence scheduling decision. So here, the overall system can be viewed as the cooperation of the OAR controller and the CiGri controller. The OAR controller manages high priority jobs that they can not be preempted (i.e they can not be interrupted to be continued later by a possible other computing node). CiGri controller injects small preemptive jobs to maximize the resource usage without affecting QoS constraints. It's important to note that a feedback loop already existed before adding the controller: CiGri controls clusters and OAR state and injects more or less jobs in OAR low priority queue, depending on this information. This loop can be viewed in Figure 10 (with the red plain arrows).

As we said previously, more sensors can be added, to capture more information about OAR state. CiGri already controls the OAR high priority queue, but the only thing captured here is if this queue is empty or not. Rather than monitoring only that, CiGri may check the queue content: maybe some low priority jobs can be placed if this placement will not disrupt the execution of the future high priority jobs. The scheduling strategy adopted by OAR can be monitored too: we assume that this strategy don't change dynamically, depending on the arriving jobs. If this point changes, checking OAR behavior may be relevant, to adapt CiGri to each new strategy.

5.3 Feedback loop in the Cloud context

In previous sections, we saw that control loop can be represented at the scheduler level and it can be well used in

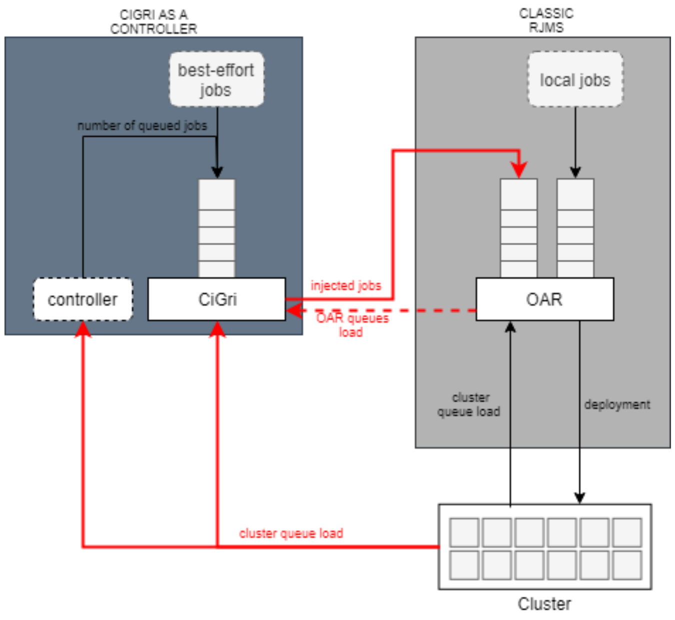


Figure 10: CiGri as an unique controller. Red plain arrows represent the CiGri control loop (even without the added intern controller). The red dotted arrow is additional information. Nothing excludes the possibility of monitoring more information.

the HPC context. Finding this control mechanism will be quite easy in a Cloud context in that autonomic computing was pretty popular in this kind of infrastructure. However, the previous section showed us two feedback loops that cooperate but stay independents. We will try to show in this section how control loops can cooperate efficiently. Cloud infrastructure benefits from the same RJMS model as HPC infrastructure: A job manager, the scheduler and a resource manager that communicate each other, forming two loops (see Figure 5). To show a possible finer interaction between these two loops, the model described by Hadrien CROUBOIS[7] in 2019 will be adopted. In his paper, the author suggests a modular structure for a fully autonomous workflow manager, with three main modules:

Static Analysis This module is in charge of the offline optimization of workflows. This part is the equivalent of the job manager seen in the RJMS section, apart from the fact that a workflow is a set of tasks with dependencies, and it can be viewed like an unique job. The objective is to precompute meta-data that will later help in the placement and execution of the relative workflows on the shared platform. In this part, all tasks will be grouped, depending on the dependencies and data locality, to optimize the execution.

Dynamic scheduling This module is based on the previous module meta-data, to suggest a scheduling while taking into account system elasticity.

Autonomic platform management This module is in charge of balancing the number of available machines to meet users demand. It will therefore control periodically state of the system and the workflows queue.

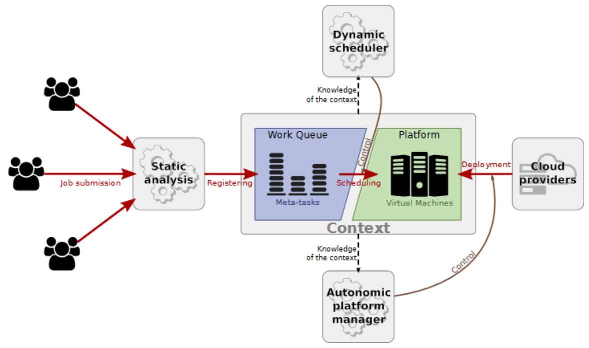


Figure 11: Overview of the system described in the paper[14].

One of the critical points is data-locality, which can be improved by placing tasks in the same location as their data dependencies, or by duplicating tasks that produce large datasets. The challenge here is to find a efficient placement to reduce communication cost, while preserving workflow structure. During the static analysis, tasks are grouped into clusters, correspondingly to their dependencies with data and other tasks: The aim is to execute one cluster in one single machine to achieve good data locality. After the mapping of the clusters onto the nodes has been performed, the nodes will be in charge of the scheduling of the tasks in the clusters they were assigned. We distinguish two kind of schedulers here: The core scheduler which assigns clusters to machine, and the node scheduler (in each machine) which assigns each tasks from the same cluster across all cpu of the said machine. As we said before, each scheduler can be viewed as a control loop.

On top of that, resources must be managed despite the complexity of the work structure. An other control loop is needed to manages all these resources in the highest level (core scheduler level). The estimation of a cluster needed resources is compared to the number of available resources. When a node is unused, it turn off itself. A single loop may be enough, but a finer approach was chosen here: The core scheduler is already aware of the resources state and the workload distribution across the platform. Since it has a global view of the system, information about machines can be reused. Binding the scheduler control loop with the resource management loop seems to be a better strategy. In the other hand, when more resource is needed, we should allocate new nodes, this part is managed by a dedicated control loop.

To keep it simple: the scheduler loop manage the work distribution and shut down useless nodes. The resource manager loop allocates new nodes, according to the information given by the core scheduler. In this way, these two loops share the responsibility of managing the platform resources.

However, this configuration stays complex, and many loops have to collaborate each other, adding more challenges. Rather than distribute decisions across all nodes, maybe a more centralized approach may be more efficient. Here, the decision of activating or not a node is made by two loops. Merging these two loops in one single loop may simplify the overall structure.

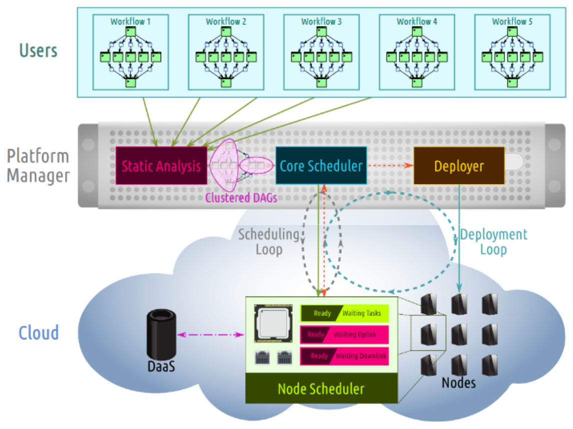


Figure 12: Autonomic scheduling and allocation loops[14].

6. CONCLUSION

In summary, this paper presents some possible explorations between scheduling and autonomic computing field. Although the intersection of these two domains was not really studied, a potential cooperation between these two fields can lead to interesting results. Some scheduling algorithms and RJMS already includes an autonomic approach without formalizing their feedback loop explicitly. Likewise, a scheduling algorithm can be seen as a control law by the autonomic computing community. Formalizing this merge allows a modular approach, quite appreciated in software engineering. Instead of seeing a scheduler as a monolithic constituent of an HPC or Cloud infrastructure, this one is divided on several sections with the system itself on one hand, and its controllers on the other hand. Proposing simpler solutions for scheduling problems adds an other good point at this collaboration. Adding a controller in an existing scheduler will not offer performance guarantees but may nevertheless increase its efficiency, by a constant control of each instance, and some adjustments when the scheduling doesn't meet requirements. However, creating multiple loops and trying to make them work together can make the system more complex.

Despite that, scheduling and autonomic community has much to gain by sharing their respective expertise, and this exploration can lead to many perspectives: Which classes of scheduling algorithms can be converted on control loop ? Which properties are required to convert such algorithms to control loops ? Which data can be added during monitoring for a better analysis ? Cohabitation between several loops must be more studied, and the feedback loops authentication in schedulers, started in this work, may be continued in next researches.

Acknowledgements

Since this project was performed during the covid-19 lockdown, a simple *"I want to thank..."* will be not enough to describe the kindness and the patience of Raphaël BLEUSE, who helped me and support me despite this context. He was present and helpful, like no one, and I will definitely recommend him to the futures trainees. His inspiring ideas and valuable discussions led to progress on this research, and his advice concluded my formation in style. I particularly want to honor Eric RUTTEN for his teaching skills. His clear approach, with a hint of humor, only highlight his unquestionable expertise. His freshness was a breath of air in these difficult times, and I want to thank him for making me discover the autonomic computing world with such an appreciable approach. This section will not be complete without mentioning all my colleagues, and particularly Quentin GUILLOTEAU, who shared with me all the vagaries of this internship. Thanks to him, my coffee times (during videoconferencing included) was really qualitative.

References

[1] Ronald Lewis GRAHAM. “Bounds for Certain Multiprocessing Anomalies”. In: *Bell System Technical Journal* (1966).

[2] Ronald Lewis GRAHAM, Eugene Leighton LAWLER, Jan Karel LENSTRA, and Alexander Hendrik George RINNOOY KAN. “Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey”. In: *Annals of Discrete Mathematics* 5.2 (1979), pp. 287–326. DOI: 10.1016/S0167-5060(08)70356-X.

[3] Christoph DÜRR, Sigrid KNUST, Damien PROT, and Óscar C. VÁSQUEZ. *The Scheduling Zoo*. URL: http://schedulingzoo.lip6.fr (visited on 2019-06-07).

[4] Joseph Y.-T. LEUNG, ed. *Handbook of Scheduling. Algorithms, Models, and Performance Analysis*. Chapman and Hall CRC, Apr. 2004. DOI: 10.1201/9780203489802.

[5] David B. SHMOYS, Joel WEIN, and David P. WILLIAMSON. “Scheduling Parallel Machines On-Line”. In: *SIAM J. Comput.* 24.6 (Dec. 1995), pp. 1313–1331. DOI: 10.1137/S0097539793248317.

[6] Raphaël BLEUSE. “Apprehending heterogeneity at (very) large scale. (Appréhender l’hétérogénéitéà (très) grande échelle)”. PhD thesis. Grenoble Alpes University, France, 2017. URL: https://tel.archives-ouvertes.fr/tel-01722991.

[7] Yiannis GEORGIOU. “Contributions for Resource and Job Management in High Performance Computing. (Contributions à la Gestion de Ressources et de Tâches pour le Calcul de Haute Performance)”. PhD thesis. Grenoble Alpes University, France, 2010. URL: https://tel.archives-ouvertes.fr/tel-01499598.

[8] Albert REUTHER et al. “Scalable system scheduling for HPC and big data”. In: *J. Parallel Distributed Comput.* 111 (2018), pp. 76–92.

[9] Jeffrey O. KEPHART and David M. CHESS. “The Vision of Autonomic Computing”. In: *IEEE Computer* 36.1 (2003), pp. 41–50. DOI: 10.1109/MC.2003.1160055.

[10] Éric RUTTEN, Nicolas MARCHAND, and Daniel SIMON. “Feedback Control as MAPE-K Loop in Autonomic Computing”. In: *Software Engineering for Self-Adaptive Systems III. Assurances - International Seminar, Dagstuhl Castle, Germany, December 15-19, 2013, Revised Selected and Invited Papers*. Ed. by Rogério DE LEMOS, David GARLAN, Carlo GHEZZI, and Holger GIESE. Vol. 9640. Lecture Notes in Computer Science. Springer, 2013, pp. 349–373. DOI: 10.1007/978-3-319-74183-3_12.

[11] Daniel BALOUEK et al. “Adding Virtualization Capabilities to the Grid’5000 Testbed”. In: *Cloud Computing and Services Science*. Ed. by Ivan I. IVANOV, Marten VAN SINDEREN, Frank LEYMANN, and Tony SHAN. Vol. 367. Communications in Computer and Information Science. Springer International Publishing, 2013, pp. 3–20. DOI: 10.1007/978-3-319-04519-1_1.

[12] N. CAPIT, G. DA COSTA, Y. GEORGIOU, G. HUARD, C. MARTIN, G. MOUNIE, P. NEYRON, and O. RICHARD. “A batch scheduler with high level components”. In: *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.* Vol. 2. 2005, 776–783 Vol. 2.

[13] Emmanuel STAHL, Agustín Gabriel YABO, Olivier RICHARD, Bruno BZEZNIK, Bogdan ROBU, and Éric RUTTEN. “Towards a control-theory approach for minimizing unused grid resources”. In: *Proceedings of the 1st International Workshop on Autonomous Infrastructure for Science, AI-Science@HPDC 2018, Tempe, AZ, USA, June 11, 2018*. ACM, 2018, 4:1–4:8. DOI: 10.1145/3217197.3217201. URL: https://doi.org/10.1145/3217197.3217201.

[14] Hadrien CROUBOIS. “Toward an autonomic engine for scientific workflows and elastic Cloud infrastructure. (Etude et conception d’un système de gestion de workflow autonomique)”. PhD thesis. University of Lyon, France, 2018. URL: https://tel.archives-ouvertes.fr/tel-01988995.

Transparent parallel algorithm composition

Louis Boulanger
M1 MoSIG
Grenoble, France
louis.boulanger@etu.univ-grenoble-alpes.fr

Supervised by: Frédéric Wagner

I understand what plagiarism entails and I declare that this report is my own, original work.
Name, date and signature: August 18th 2020, Louis Boulanger

[illegible]

Abstract

Parallel algorithms leverage multi-core processors' power to run faster; but using a parallel algorithm inside of another often leads to overheads, such as task creation. We designed policies for Rust's *Rayon* parallelism library, that regulate composed parallel algorithms and suppresses task creation overhead. In particular, we designed an adaptor for parallel iterators called *Composed*, which limits composed parallelism without fine-tuning from the user. These policies enable *transparent* composition.

1 Introduction

Nowadays, parallel algorithms and programs are ubiquitous in computing, as they enable the use of the entire multi-core processors capabilities. They vastly outperform sequential algorithms, and help resolve problems in research, as well as improve performance in industrial and commercial applications. This is why some sequential algorithms have parallel versions that can be used in order to improve performances.

These parallel algorithms are often designed and published as libraries, that users can use in their programs. However, using multiple parallel algorithms together can induce overhead, as each algorithm use all the available resources as if they were the only parallel algorithm running in the program. This leads to a multiplication of threads, which is detrimental to performances. For example, when searching through multiple files to find a text pattern, we have an iteration over the files, which can be parallel, as well as an iteration over the lines of each file, which can also be parallel. In these cases, it is often better to use a sequential version of an algorithm if it's used inside of a parallel one. For example, [Pan *et al.*, 2010] reported that Intel's Math Kernel Library (MKL) instructs to use the sequential version of their algorithms whenever running in parallel with another part of the application.

[Walker *et al.*, 2009] explored an extension of the BourneAgain Shell (bash) to use shell pipes with forks, joins and other parallel operations in order to compose different programs in a scripting language. [Pan *et al.*, 2010] proposed a low-level subtrate to efficiently allocate processing cores by using a *hardware thread* primitive instead of the virtual threads in order to efficiently compose parallel libraries. [Anderson, 2012] describes an extension of the C language, providing a new syntax to require certain resources (or a set of resources), which can help reduce composition overhead through the use of different policies and *allocation trees*.

Our solution relies on the *Rust* programming language, and more precisely the *Rayon* parallelism library, which uses work stealing for scheduling tasks. As a part of the *kvik* library, which adds several features to Rayon, we present several iterator adaptors that help compose parallel algorithms, and benchmarks assessing the performances of our solution compared to the use of the sequential inner algorithm.

2 Work stealing with Rayon

Work stealing is a strategy for scheduling tasks. [Blumofe and Leiserson, 1999] explains how work stealing functions. We consider *tasks*, which are sequences of computer instructions. A task can *spawn* another, and wait for its child tasks to to *join* back. Each processor has a queue in which it puts tasks spawned during its execution.

In work stealing, the inactive processors take the initiative to get a task to work on: whenever a task is available in the queue of a busy processor, it can be "stolen" by an idle processor; this way, there is less task migration than with *work sharing*, and processors will always try to work on a task.

This scheduling strategy goes back to early computing, where [Halstead, 1984] implemented a parallel version of Lisp using the same idea as work stealing. It has been used and implemented in many libraries, such as Cilk [Blumofe *et al.*, 1995], some implementations of the OpenMP C compiler [Tanaka *et al.*, 2000], [Hadjidoukas and Dimakopoulos, 2007], Kaapi [Gautier *et al.*, 2007], Intel's Thread Building Blocks [Kukanov, 2007], and Rayon.

2.1 Adaptors

The strength of Rayon is the concept of the *parallel iterator*. Rust's iterators are already very powerful and propose operations such as `map` and `reduce`, and Rayon's parallel iterator extends the concept to parallel algorithms. Non-reducing parallel iterator operations are implemented through the means of *adaptors*. Most adaptors in the library operates the same way: it adds its type onto the existing parallel iterator, and usually calls the function from this inner iterator and performs some computation or control over it.

For example, in order to execute a `map`, then a `fold`, on a vector of integers, the resulting parallel iterator would be of type `Fold<Map<Vec<i32>, ...>, ...>`. This means that any operation performed on the iterator would go through the `Fold` implementation, and then go down the iterators until we reach the implementation of the parallel iterator for vectors. Using this logic, we can control how parallelism is performed by implementing our custom reduction functions (which are implemented by default for all parallel iterators) such as `reduce`.

In order to illustrate, let's consider the EVEN SUM problem: with N vectors containing a varying number of integers, we want to find the number of vectors whose sum is even. A simple algorithm to solve this problem is presented in algorithm 1.

```
Data: a list of vectors $V$, of varying size
Result: the number $r$ of vectors in $V$ whose sum is
        even
$r \longleftarrow 0$;
foreach $v$ in $V$ do
    $s \longleftarrow 0$;
    foreach $i$ in $v$ do
        $s \longleftarrow s + i$;
    end
    if $s$ is even then
        $r \longleftarrow r + 1$;
    end
end
```

Algorithm 1: Computation of number of vectors having an even sum

This algorithm can be written with Rayon using the adaptors we mentionned previously: the code is detailled in listing 1. Note that in this version, we use a sequential iterator for the inner level.

This function first creates a parallel iterator from the vector of vector (using the `par_iter()` function), then uses `map` to transform each vector using a function (the *closure* parameter of the `map`). In this inner function, we take an iterator for each vector, using `iter()`, then sums it. Then we return `((sum + 1) % 2)`: this returns 1 if the sum is even. At the end, we can use a `sum` operation on the outer level, in order to sum up the results, and effectively get the number of vectors whose sum is even. We can see that there is very little boilerplate for this code: only the call to `iter` is replaced by `par_iter()`.

We compute in parallel this double sum by dividing the outer iterator into many small pieces. Each small piece contains a part of the elements from the original iterator, and we can perform the `map`, then the `sum`. Effectively, this will compute the number of even sums inside the small piece. Afterwards, it gets reduced to a single value. The code here doesn't specify how, and how much the outer iterator will be divided; this is up to the middleware. This abstraction has two benefits:

```rust
fn even_sum(vec: &Vec<Vec<u64>>) ->
↪   usize {
    let threads =
↪   rayon::current_num_threads();
    let limit = (((threads as
↪   f64).log(2.0).ceil()) as usize) + 1;

    vec.par_iter()
        .map(|v: &Vec<u64>| {
            let sum: u64 =
↪   v.iter().sum();
            ((sum + 1) % 2) as usize
        })
        .rayon(limit)
        .sum()
}
```

Listing 1: Rust code for computing the even sum problem

- The user doesn't have to fine-tune any threshold or value that would be required for specifying the division process,
- The middleware can decide how to divide the iterators, and we can design policies to specify a certain dividing scheme.

For this particular example, we used Rayon's policy: which uses a counter to limit the division to a certain number of iterator pieces. This approach is also used by Intel's TBB library [Kukanov, 2007]. The counter is initialized to a certain value; after a division, we decrement the counter. If the counter reaches 0, division stops, and the iterator piece is computed. The initial value can adjusted depending on the nature of the task and the number of threads. By default in Rayon, the threshold is set to $\lceil \log_2 P \rceil + 1$ where P is the number of processors: this creates $2P$ tasks in total. In the case where a piece is stolen, the counter is reset: this is because once a processor steals, this means that it doesn't have any other task to work on, and we need to create multiple tasks to keep it busy.