Formal and Informal Methods for Multi-Core Design Space Exploration

Jean-Francois Kempf
VERIMAG
University of Grenoble
France
kempf.jf@gmail.com

Olivier Lebeltel
CNRS-VERIMAG
University of Grenoble
France
Olivier.Lebeltel@imag.fr

Oded Maler
CNRS-VERIMAG
University of Grenoble
France
Oded.Maler@imag.fr

We propose a tool-supported methodology for design-space exploration for embedded systems. It provides means to define high-level models of applications and multi-processor architectures and evaluate the performance of different deployment (mapping, scheduling) strategies while taking uncertainty into account. We argue that this extension of the scope of formal verification is important for the viability of the domain.

1 Introduction

Consider an application program to be executed on a *multi-core platform*. The application is modeled as a *task graph*, a collection of tasks partially-ordered according to precedence and annotated by execution times and data transfer volumes. We assume that these durations, as well as the arrival times of new jobs to execute, admit some *bounded uncertainty*. We want to evaluate the influence of different deployment strategies (mapping, scheduling, etc.) on the overall *performance* of the system. We start by explaining why this research direction constitutes a fruitful and important extension of the scope of formal verification.

Correctness vs. Performance

Algorithmic formal verification is concerned with proving *functional correctness* of certain systems, most notably finite-state systems such as communication protocols and digital hardware. This is often done by abstracting away from *data* and focusing on *control* (synchronization). However, functional correctness in the strict sense often used in verification is not a *necessary* nor *sufficient* condition for the usefulness of a system. A bullet-proof correct system with an extremely slow response is not likely be ever used, while systems that work well *most* of the time are all around us. To keep formal verification and its insights alive and make it applicable and relevant to system design beyond the very narrow context in which it is currently used, one should rethink some of the basic premises of the field, in particular:

1. The qualitative logical models of systems;
2. The qualitative *yes/no* nature of the questions asked and the answers provided;
3. The *universal quantification* over behaviors.

Relaxing the first premise is of course not new. Models of automata augmented with numerical variables are used extensively in *software verification* as well as in *hybrid systems*. Timed automata [2], the model most relevant to the present paper, have been invented to model delays and execution times in a quantitative way. The second relaxation which has been argued under the banners of *quantitative*

Submitted to:
QAPL 2014

analysis/synthesis [16, 11] consists of decorating transition systems with numerical costs and tracking their evolution. Such costs typically admit a simpler dynamics than more general numerical variables in programs or hybrid systems. For example, the model of linearly-priced timed automata [29, 12], which are timed automata augmented with costs that can grow at different rates at different states, is simpler to analyze than other hybrid systems with constant slope [27, 25, 4] because the cost variables are *passive observers* of the dynamics. The relaxation of universal quantification is what underlies *statistical model-checking* [41, 17, 19] and can be viewed cynically as the verification community discovering what practitioners have known all along. We argue and demonstrate in this paper that a *combination* of all these relaxations has a great potential in solving real problems in modern *systems design* including a central problems related to the multi-core revolution: how to evaluate and optimize the *performance* of application programs on such execution platforms.

Functional correctness and good performance are complementary and sometimes conflicting evaluation criteria. In *hard* real-time systems, performance is hardwired into correctness: a feedback function of a controller should be computed between every two consecutive sensor readings which puts a *deadline constraint* on its computation time. Using a *timed* model of the software/hardware architecture, which represents the execution times of the tasks as well as the scheduling policy, one can verify that such a deadline is never missed. In other words, the quantitative timing information about the system participates in the proof of a functional *yes/no* property. In certain simple situations studied extensively by the real-time community [15, 32, 28] one can do the calculation [31] without invoking an explicit dynamic "executable" model at all. For other, increasingly more popular, classes of embedded systems, the real-time constraints are *softer* and the system is expected to give a best effort performance depending on the system load and resource availability. A typical example would be video streaming where a good trade-off between response time and image quality is sought. For such systems, the actual response time is a performance measure of the system, together with additional criteria such as system price or power consumption. Unlike what is common in verification, the quantitative measures are not "Booleanized" via predicates/constraints into a yes/no answer but remain *quantitative* and can be used to *compare* the relative performance of different designs.

The major contrast with the tradition of safety-critical verification is that soft systems are *not* evaluated according to their *worst-case* behavior but in a more *probabilistic* fashion. The traditional verification approach to the problem of performance evaluation based on "classical" timed automata technology [42, 20, 13, 30, 40, 9, 3] is exhaustive: it can compute performance measures such as termination time and other costs for *all* possible values of the uncertainty space, thus compute lower- and upper-bounds on termination time. For soft real-time systems this is, at the same time, too much and too little. The lower and upper-bounds represent very extreme cases which are realized only when all the tasks take their extremal duration values. Under very reasonable assumptions they are less likely than termination times that admit many realizations (as 7 is more likely than 12 in dice). In contrast with the exhaustive approach, in Monte-Carlo simulation the uncertainty space is finitely *sampled* according to some distribution and each sampling point induces a single deterministic behavior whose performance is evaluated by (cheap) simulation. Such an approach is weaker than formal verification because it does not cover *all* behaviors: it can, at most, put bounds on the probability of error or a deadline miss. On the other hand it is stronger as it can give an estimation of the *distribution* and *expected value* of the termination times, which can be much more useful for this type of applications than the very conservative bounds computed by the exhaustive approach.

The present paper is thus yet another step toward a pragmatic fusion of formal verification and performance evaluation (see also the dedicated volume [14] and proceedings of some related conference [35, 23, 34]) to produce a tool-supported methodology for high-level performance analysis (and eventu-

ally, synthesis) of applications programs running on multi-core architectures. In particular, this framework provides:

1. A formal description language for applications, hardware platforms, external environments as well as mapping and scheduling policies;
2. A translation of these objects into timed automata and employing both *set-theoretic* and *probabilistic* interpretation of timing uncertainty in their semantics;
3. Performance evaluation procedures based on either standard zone-based timed automata verification (when size permits) or statistical simulation.

Industrial Context

Platform 2012 (P2012, [37]) is an ongoing project of ST Microelectronics (the largest European semiconductor manufacturer) and CEA-LETI to develop a multi-core architecture to serve as an accelerator (computation fabric) for high throughput computational tasks (video processing, radio sensing, image analysis) for embedded (smart phones) and other (TV set top boxes) devices. P2012 is viewed as an alternative to GPUs as a replacement of dedicated hardware currently used for these functions. The flexibility and productivity gains of software are supposed to compensate for a tolerable degradation in performance compared to hardware. However, writing parallel software is not a trivial matter and deploying it efficiently on the multi-core platform (mapping, memory allocation, scheduling of computations and data transfers) is a hard combinatorial optimization problem with a significant variations in performance over its feasible solutions. In some sense, the multi-core revolution brings application software developers back to earlier and darker days where they had to reason about low-level architecture dependent details in order to meet performance requirements.[1]

The present work has been carried out within the French regional project ATHOLE (2008-2012). One of the goals of the project was to provide high-level tools to analyze (and optimize) the performance of applications on the P2012 architecture. Current performance evaluation tools used on the hardware side, at least based on our experience, work in a very *low granularity*, that is, they simulate the execution of the code on the processor in a *cycle-accurate* manner. This leads to very costly simulation whose extreme precision is an overkill, especially given that often this simulator is combined with much rougher models of the interconnect infra-structure. Moreover, such an analysis requires that the application is already written and that the architecture exists, at least virtually. Our work suggests a complementary approach in which:

- Applications are modeled at the *task*, rather than *instruction*, level. This means that a piece of code is modeled as a *timed process* characterized by a quantitative estimations of its duration and the amount of data it exchanges with other tasks. Such a description is compatible in spirit with numerous data-flow and component-based frameworks [8, 22, 38, 39, 5] advocated for writing such applications;
- We model high-level performance related features of the architecture such as processor speeds, bandwidth and latency of communication mechanisms, static and dynamic power consumption of architecture elements, etc.
- Task durations, as well as arrival rates, are modeled as admitting *bounded uncertainty*, thus compensating for the lack of detail and accuracy in the application and architecture models.

[1] Or one can look at it more positively as pulling developers of hardware IP upward toward the joy of high-level software development.

As a result we provide hardware-software co-designers with a tool for rapid design-space exploration: based on profiling or past experience, the designer may decorate the application with performance numbers (intervals and distributions alike) and compare the performance figures obtained using different platforms, mapping decisions and scheduling strategies. Such procedures accelerate feasibility checks at early design stages and can be eventually integrated into the compilation and deployment chain.

The rest of the paper is organized as follows: Section 2 presents the principles of our (extendable) system description language, describes the analysis techniques (formal and statistical) supported by our tool and provides some implementation details. Section 3 demonstrates the whole approach on case study concerning power/performance tradeoff of a skeleton of an application inspired by image processing. A short discussion concludes the paper.

2 System Modeling and Analysis

Our guiding modeling principle is to abstract away as much as possible from low-level details such as the application code itself or hardware protocols and compensate for the lack of precise information by increasing the uncertainty margins and taking this uncertainty[2] seriously in the analysis. There will be several types of under-determination in the durations of tasks and data transfers or their arrival rates. These can be due to various phenomenological origins: tentative ignorance in early development stages, true data-dependent variability in the algorithms or unmodeled variability in the architecture workload and physical conditions.

Applications

Applications are described by task-data graphs which are a simple generalization of the common task-graph model [18]. A task is an atomic computational entity which is characterized by an amount of work measured by instructions or cycles. Once a task is scheduled to execute on a processor with a given frequency, its amount of *work* is translated into duration. We allow *bounded uncertainty* (interval) for the amount of work. Another characteristic of a task is *precedence*: it cannot start before some other tasks terminate and its termination may be a pre-condition to the initiation of other tasks. Finally, we model the quantity of data that has to be communicated between a task and each of its successors. Depending on the mapping of the tasks onto the architecture and the data transfer mechanism used, e.g., DMA (direct memory access) or inter-process communication, this transfer is transformed into a special *communication task*. The whole task-data graph is called a *job type* and it is the basic unit of work whose instances arrive to be executed. Fig. 1 illustrates the modeling and translation to timed automata of a simple job consisting of two tasks T_1 and T_2 so that the former precedes the latter. We assume an architecture with processors that can have two speeds, 1 and 2. Automaton $\mathscr{A}_1$ models the first task. From a waiting state A, depending on a scheduler command, it can start executing in speed 1 (state B) or speed 2 (state C). In both cases the transition resets clock x and as long as the automaton is in such an active state, no valid scheduler will issue another *start* command for the same processor. Depending on the speed, the automaton may leave the active state when the clock is in the interval $[a,a']$ or $[a/2,a'/2]$ and move to final state D. The automaton $\mathscr{A}_2$ is similar except that it has a non-enabled state E which it can leave only when T_1 terminates, that is, when $\mathscr{A}_1$ is in final state D. Readers are referred to [1] for more detailed presentation of the modeling approach.

[2] Under-determination, using the terminology of [33].

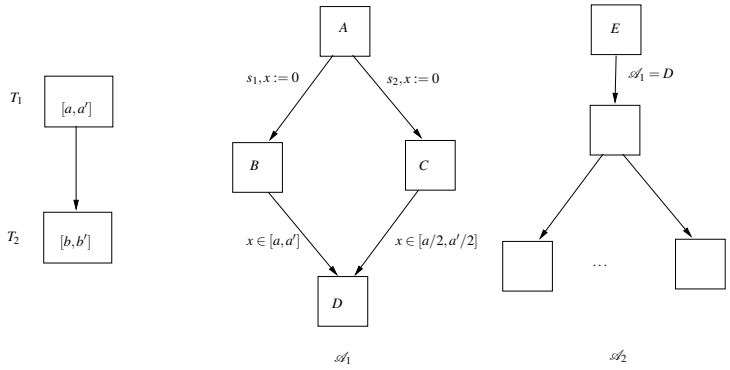


Figure 1: Translating a task graph into timed automata.

Scheduling and resource allocation in general are spread over many application domains, each focusing on specific features of the problem. The classical job-shop problem is restricted to precedence constraints that can be decomposed into a disjoint union of linear orders (jobs) and allows different types of resources (machine) so that some tasks can execute only on some machine type (which is useful for distinguishing between processors and data transfer mechanisms). On the other hand, the classical task graph problem allows a more general precedence structure (any partial order) but uses a single type of resource (processor) on which any task can be executed. Our model is a continuation of [1] where a generalized model, fusing job-shops and task-graphs and allowing different machines as well as partial orders, was introduced and translated naturally into timed automata. We allow several job types and model the process of arrival of a *stream* of job instances using input generators.

Input Generators

Another aspect of scheduling which is treated differently along communities is the *dynamic* aspect: in classical real-time scheduling new task instances arrive periodically or quasi-periodically but these are traditionally simple tasks without precedence constraints. In contrast, job-shop and task-graph problems typically do not handle the dynamic "reactive" aspect, that is, a stream of job instances that arrive one after the other, for example, a sequence of encoded image frames or web queries. This aspect is extremely important, first because it represents the real nature of these applications and, secondly, it favors solutions based on *pipelining*, that is, the concurrent execution of tasks that belong to different job instances (see some definitions and theoretical investigations in [21]).

One approach to treat this recurrence aspect is to use *cyclic* task-graphs admitting a loop from the last to the first task. While this might be suitable for modeling loops in programs where the termination of one instance enables the execution of the next one, it is not at all natural for jobs arriving from the *outside*, often independently of their processing by the system. To this end we use the concept of an *input generator*, a process that generates a timed sequence of job instances subject to some logical and timing constraints. The simplest generator is the *deterministic* periodic generator which produces an instance of a job every d time. Strictly periodic generators are sometimes idealization of more time-noisy processes and we allow additional types of non-deterministic generators listed below (for simplicity of notation we assume here that the arrival of the first instance is $t_0 = 0$).

1. Periodic: $t_k = t_{k-1} + d = (k-1)d$;

2. Periodic with jitter (non-accumulated deviations from the period):
   $t_k \in [(k-1)d, (k-1)d+J]$
3. Periodic with uncertainty: $t_k \in [t_{k-1}+d, t_{k-1}+d+J] = [(k-1)d, (k-1)(d+J)$;
4. Bounded variability: for every interval of the from $[r, r+\Delta]$ the number of arrival events is at most M;
5. Bi-bounded variability: for every such interval the number of events ranges between m and M.

All these types of generators are translated into timed automata that realize their semantics. They play the same role that stochastic arrival processes play in queueing theory. For generators of type 2 and 3 we also implemented a probabilistic semantics drawing *uniformly* from $[t, t+J]$. Other types of generators that can choose (non-deterministically or randomly) among different types of jobs [21] or stochastic generators with other distributions can be easily added.

Technically, each instance of a job generates a new instance of the corresponding automaton and this may lead to an infinite-number of automata and global states. However, we are aiming at systems that do not accumulate an unbounded backlog of unprocessed tasks and all our input generators have a finite bound on the number of instances that can arrive in a given interval of time. Thus, we can purge the automaton associated with a job when it reaches its final states and keep the number of automata which are alive in any given moment bounded. We make extensive use of the techniques developed in [6, 7] to handle dynamic creation and deletion of timed automata, tracking the shifting denotation of clocks, etc.

Architecture

The architecture description language (extensible as well) describes the components of the execution platform. These include:

- Processors, characterized by their possible speeds which may be controlled during execution and which may be turned on and off;
- Memories defined by their access time to distinguish between slow offchip memory and fast local memory;
- Communication mechanisms to transfer data between memories, characterized by their transfer rate, initialization costs, etc.
- All architecture components can be decorated with power consumption figures. We assume simple system-level power models consisting of static consumption when the component is on but idle, and dynamic consumption which occurs when the components is busy (computing or moving data). Different frequencies of the processor lead, of course, to different consumption rates.

As an example, Figure 2 shows a model, generated by the graphic user interface of our tool, associated with a 16-core instance of the P2012 family. Each of the processors can work in several frequencies. The computation times of tasks are based on the assumption that their data resides in the local memory.[3] The DMA agent is characterized by its initialization time. A DMA call occupies the external and internal busses for durations that depend on the amount of data and the respective transfer rates of the busses.

[3] To avoid confusion, note that although physically the local memory is realized in the same way as caches in more traditional processors, the programmer has full control of its contents.

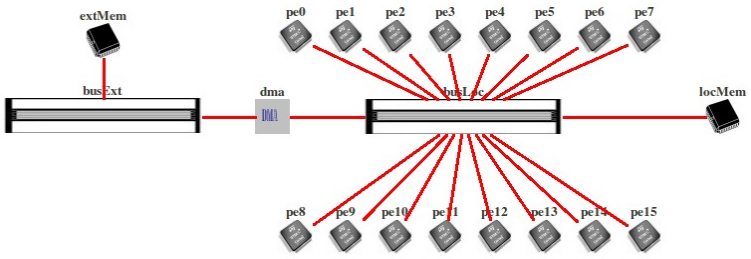


Figure 2: A model of a 16-processor instance of P2012

Deployment

Finally, given all the above descriptions, we specify the deployment policy for the application on the platform. There are many possible types of deployment decisions and we mention some of the policies that we implemented. Adding a new policy corresponds to writing the scheduler as a timed automaton and is currently a matter of few hours, depending on the complexity of the scheduler. We have implemented a FIFO scheduler, with and without priority queues and a strict priority scheduler which may hold a low-priority task waiting although there is a free processor, to wait for a higher priority tasks not yet enabled. Each of these schedulers admits a global and a local version. In the former there is a single scheduler that may assign any task to any processing element (PE), while in the local version, the mapping of tasks to PEs is determined in advance and each PE has its own scheduler and waiting queues. A more detailed explanation of timed automata models of various schedulers appears in [26]. The scheduler can also specify at which frequency to execute each task.

Analysis Methods

Once all components have been defined, their composition is equivalent to a global timed automaton whose only under-determination is related to the tasks, their durations and arrivals. We apply two types of analysis:

- *Formal*: Using the IF toolset [13] we perform on-the-fly reachability computation in the state- and clock-space. For a single job instance this type of analysis computes lower- and upper-bounds on the total termination time. For a stream of jobs, using auxiliary clocks, one can compute lower- and upper-bounds on the response time. This type of analysis manipulates timed polyhedra (zones) whose maximal dimensionality is equal to the maximal number of active system components. Moreover, with the dynamic creation and deletion of timed automata it may take more time to detect fixed-points in the reachability graph [6, 7]. For all these reasons this type of analysis is restricted to systems that may have up to 20-25 clocks (concurrently active components).

- *Statistical*: Taking the probabilistic interpretation of temporal uncertainty we draw random values uniformly and simulate the resulting behavior. This is a fairly standard discrete-event simulation whose only particularity that it is generated based on semantically rigorous models. The runs are registered as timed traces over the alphabet of all *start* and *end* events. A specification in a dedicated language defines pairs of events, for example the arrival of a job and its termination,

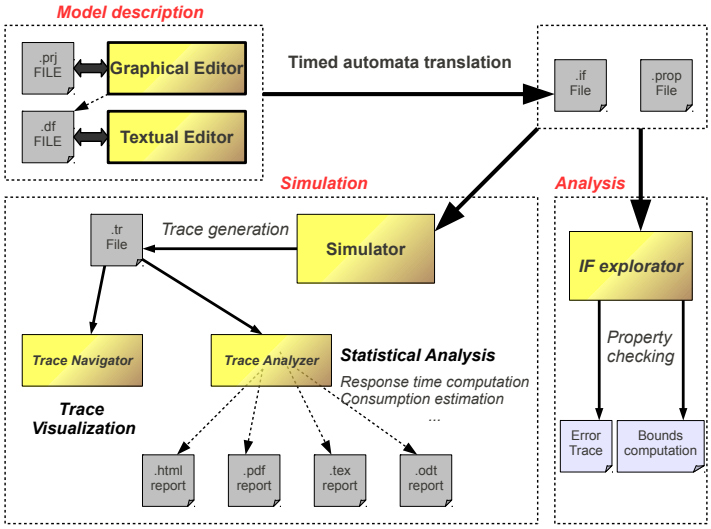


Figure 3: The architecture of the tool

whose temporal distances are extracted from the traces. For these values we compute the mean and other statistical measures that form the basis for automatically-generated reports.

Implementation Details

The tool, temporarily dubbed *The Design-Space Explorer*, consists of 25K lines of C++ code (not counting the IF analysis engine). It has a textual system description languages incorporating the abovementioned features. A graphical user interface written using Qt provides an alternative way to define systems (the illustration in this papers are produced by this interface). New types of systems components can be defined via minimal programming and are automatically propagated to the user-interface, analysis engine and the reporting system. There are two major types of outputs: raw execution traces that can be zoomed on via a dedicated *trace navigator* and statistical reports in various formats. The tool architecture is summarized in Fig. 3.

3 Case Study: Deploying a Video Application

In this section we demonstrate the applicability of our tool in exploring and comparing different deployment solutions for a data-parallel application which processes an image consisting of 16×16 blocks. The image resides initially in the offchip memory and has to be brought to local memory and dispatched to the processors for execution. This is a very typical application and similar ones exist in other domains, for example in radio-sensing, the process in which a cell phone scans the bandwidth to detect channels. The sensed array of data is split into windows each undergoing the same signal processing algorithm. We will use two variants of application and of the P2012 architecture to demonstrate the functionality of our tool. All these experiments should be taken with a grain of salt concerning their realism since the development of P2012 and its applications is still in a stage where models are very approximate. The main purpose of the exploration is to illustrate the types of analysis provided by our framework.

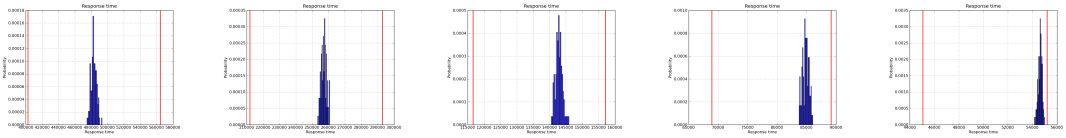


Figure 4: The distribution of total termination times using 1, 2, 4, 8 and 16 processors. The red vertical lines indicate the lower- and upper-bounds. Note that the vertical scaling is divided by two as we double the number of processors.

Worst-Case vs. Statistics

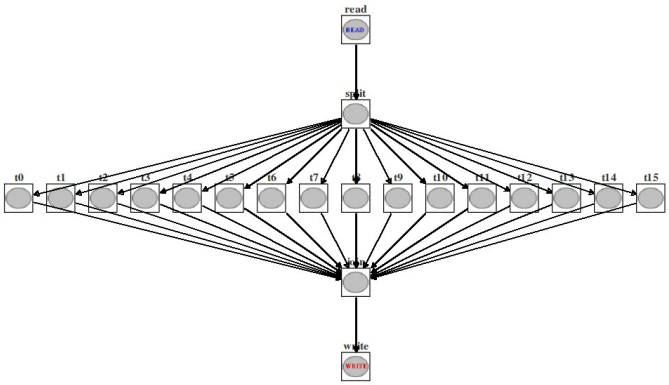
Consider the task-graph of Fig. 5-(a) which represents the treatment of a horizontal band (16 blocks) of the image. All the blocks are fetched by a single *read* command and the data is split onto 16 tasks whose output is merged and written back to the offchip memory. Execution times for processing a single block admit up 18% deviation from their average. We first run a TA-based analysis of the execution of this job on architecture instances with various numbers of processors to obtain the respective lower- and upper-bounds on execution times. Then we apply statistical analysis, based on 100 random simulation runs Fig. 4 shows a histogram of these runs for different number of processors. Note that when there is one processor per task, the average is close to the worst-case (for that configuration) because the total termination time is defined as the max of individual task termination times. On the other hand, when the number of processors is smaller and some tasks are executed sequentially, the convolution effect renders the distribution more normal-like.

Reading Granularity

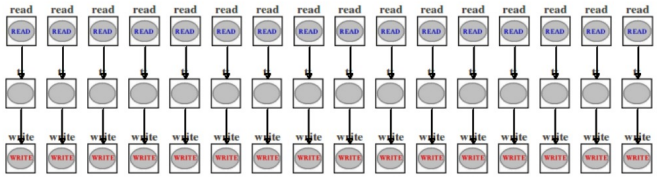
We make a comparison between two strategies for fetching the data. Fig. 5-(b) shows an alternative specification of the 16-block computation where each block is read *separately*. The whole job for 256 blocks is represented by sequential concatenation of 16 copies of the basic task-graphs (Fig. 5-(a,b)). Fig. 6 shows the speedup obtained by the second, more flexible policy, as the number of processors grow. Note that the speed-up in the average-case is much more significant.

Fixed vs. Flexible Mapping

Next we move to a situation where there is a very large variability in the execution time of the tasks, namely $[150, 2100]$, and compare a fixed mapping with a local FIFO scheduler for each PE against a flexible mapping by a global scheduler on an instance of P2012 with 4 processors. We take the task graph of Fig. 5-(b) and use a periodic event generators with jitter. Using 4 processors, each PE is assigned 4 tasks (exactly for the fixed mapping policy and approximately for the flexible policy) and hence the worst-case execution time for a job instance is around 8400. For arrival periods which are smaller than the worst-case execution time, a worst-case analysis naturally shows the possibility of an unbounded accumulated backlog and, hence, unbounded latency. We perform simulations with arrival periods 7000, 6000, 5000, and 4500. Not surprisingly, the global strategy yields a much better average performance and its advantage increases with the arrival rate. Decreasing the period to 4000 (below the average execution time) leads to frequent overflows. Fig. 7 illustrates the processor occupancy patterns following the two strategies and Fig 8 shows how the relative advantage of the flexible mapping strategy depends on the arrival period.



(a)



(b)

Figure 5: Two ways to process 16 blocks of data: (a) one centralized read, split and merge; (b) 16 independent *read*s and *write*s.

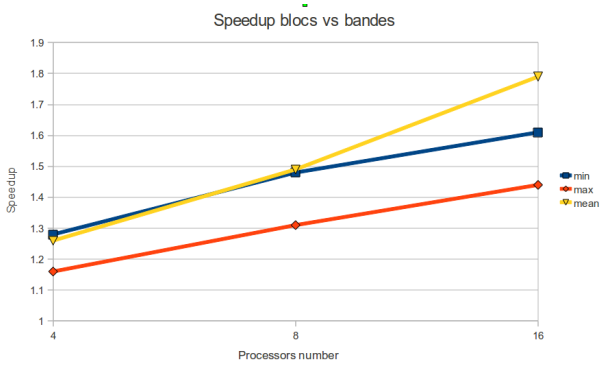


Figure 6: The speed-up obtained by reading single blocks compared to reading 16-block bands.

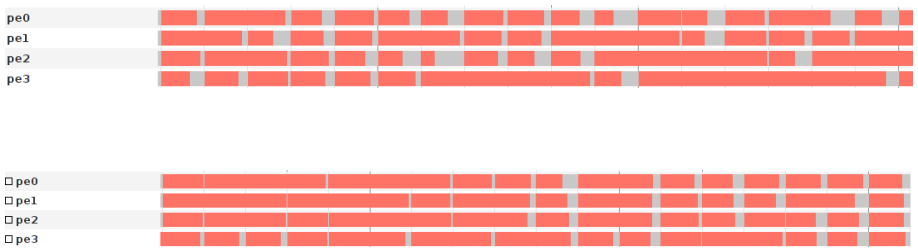


Figure 7: Processor utilization under the fixed (up) and the flexible mapping strategies.

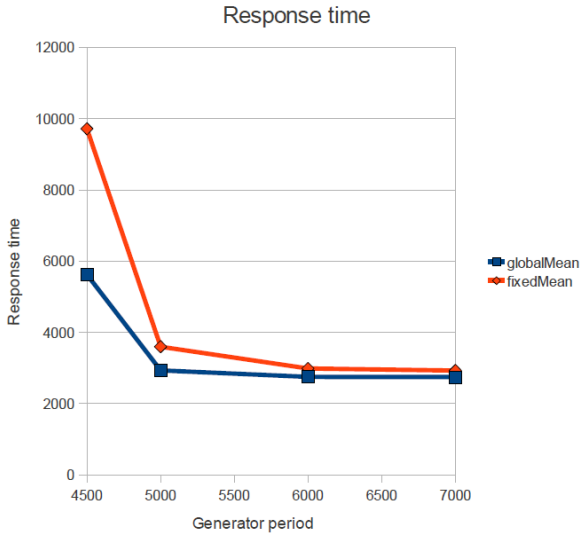


Figure 8: Comparing the average performance of the fixed and flexible mapping strategies as a function of the arrival period.

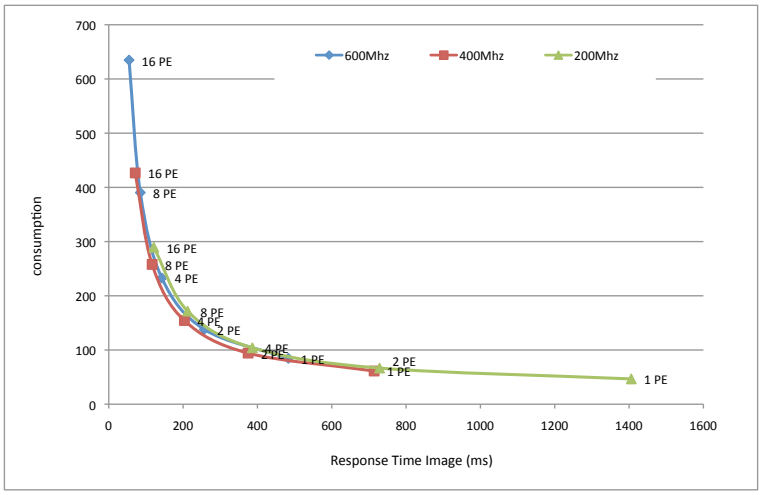


Figure 9: Power-performance tradeoffs obtained on different configurations (number of processors and frequencies).

Power Consumption

In the last experiment we compare different configurations of P2012 for the trade-offs between *response time* and *power consumption* that they provide. We consider again a job consisting of a concatenation of 16 copies of the task graph of Fig. 5-(b) and execute it on instances of the architecture with 1, 2, 4, 8 and 16 active processors, all running in either 200, 400 or 600 MHz. For each configuration we run 100 simulations and compute the average response-time and consumption. Fig. 9 shows the trade-offs obtained. Such plots are extremely useful for detecting regions where power consumption can be significantly reduced with a modest performance degradation which still meets the system requirements.

4 Discussion

We presented what we believe to be a convincing demonstration of the potential contribution of a relaxed variant of formal verification to system design. We took from formal verification the following: 1) High-level abstract models that suppress details and focus on the features important for the task in question (traditionally synchronization and concurrency and here timing); 2) Executable and semantically correct models; 3) An explicit treatment of under-determination and 4) Tool support. We augment exhaustive timed automata analysis with Monte-Carlo simulation for scalability (similar to [19] and thus can add costs such as power consumption to the model without worrying too much about decidability. Design-space exploration is a very active topic in other communities handling embedded systems [24, 10, 36] and we hope that our approach will contribute to bridging the gap between communities and easing the transition to multi-core computing. The main message that we want to convey is that *timed models* such as timed automata are exactly the kind of models needed for this type of applications. What prevents their real-life application is their association with an overly-ambitious and intractable analysis method, which on the top of that is also hard to explain to practitioners. We believe that the more lightweight approach presented in this paper will change this situation.

Among the future extensions we consider we mention tighter integration with other formalisms used

to write such applications such as synchronous data-flow (SDF) and its variants, adding a module for piecewise-analytic computation of expected performance as in [26], more sophisticated Monte-Carlo simulation, computing confidence bounds on the statistical results and more selective trace generation to reduce storage and increase speed. To promote applicability we also need to enrich the component library and define a hierarchy of models of varying granularity and precision.

References

[1] Y. Abdeddaïm, E. Asarin & O. Maler (2006): *Scheduling with timed automata*. Theoretical Computer Science 354(2), pp. 272–300. Available at http://dx.doi.org/10.1016/j.tcs.2005.11.018.

[2] R. Alur & D.L. Dill (1994): *A Theory of Timed Automata*. Theoretical Computer Science 126(2), pp. 183–235.

[3] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson & Wang Yi (2003): *TIMES: A Tool for Schedulability Analysis and Code Generation of Real-Time Systems*. In: *FORMATS*, pp. 60–72. Available at http://dx.doi.org/10.1007/978-3-540-40903-8_6.

[4] Eugene Asarin, Oded Maler & Amir Pnueli (1995): *Reachability Analysis of Dynamical Systems Having Piecewise-Constant Derivatives*. Theor. Comput. Sci. 138(1), pp. 35–65. Available at http://dx.doi.org/10.1016/0304-3975(94)00228-B.

[5] Ananda Basu, Marius Bozga & Joseph Sifakis (2006): *Modeling Heterogeneous Real-time Components in BIP*. In: *SEFM*, pp. 3–12. Available at http://doi.ieeecomputersociety.org/10.1109/SEFM.2006.27.

[6] Ramzi Ben Salah (2007): *On Timing Analysis of Large Systems*. Ph.D. thesis, INP Grenoble.

[7] Ramzi Ben Salah, Marius Bozga & Oded Maler (2009): *Compositional Timing Analysis*. In: *EMSOFT*. Available at http://www-verimag.imag.fr/~maler/Papers/tabst-new.pdf.

[8] Albert Benveniste, Paul Caspi, Paul Le Guernic & Nicolas Halbwachs (1993): *Data-Flow Synchronous Languages*. In: *REX School/Symposium*, pp. 1–45. Available at http://dx.doi.org/10.1007/3-540-58043-3_16.

[9] Dirk Beyer, Claus Lewerentz & Andreas Noack (2003): *Rabbit: A Tool for BDD-Based Verification of Real-Time Systems*. In: *CAV*, pp. 122–125. Available at http://springerlink.metapress.com/openurl.asp?genre=article{&}issn=0302-9743{&}volume=2725{&}spage=122.

[10] Tobias Blickle, Jürgen Teich & Lothar Thiele (1998): *System-Level Synthesis Using Evolutionary Algorithms*. Design Autom. for Emb. Sys. 3(1), pp. 23–58. Available at http://dx.doi.org/10.1023/A:1008899229802.

[11] Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger & Barbara Jobstmann (2009): *Better Quality in Synthesis through Quantitative Objectives*. In: *CAV*, pp. 140–156. Available at http://dx.doi.org/10.1007/978-3-642-02658-4_14.

[12] P. Bouyer, U. Fahrenberg, K.G. Larsen & N. Markey (2011): *Quantitative analysis of real-time systems using priced timed automata*. Commun. ACM 54(9), pp. 78–87. Available at http://doi.acm.org/10.1145/1995376.1995396.

[13] M. Bozga, S. Graf & L. Mounier (2002): *IF-2.0: A Validation Environment for Component-Based Real-Time Systems*. In: *CAV*, pp. 343–348.

[14] E. Brinksma, H. Hermanns & J.-P. Katoen, editors: *Lectures on Formal Methods and Performance Analysis*.

[15] G. Buttazzo (2005): *Hard Real-Time Computing Systems: Predictable Schedulding Algorithms and Applications*. Springer.

[16] Pavol Cerný & Thomas A. Henzinger (2011): *From Boolean to quantitative synthesis*. In: *EMSOFT*, pp. 149–154. Available at http://doi.acm.org/10.1145/2038642.2038666.

[17] Edmund M. Clarke, Alexandre Donzé & Axel Legay (2010): *On simulation-based probabilistic model checking of mixed-analog circuits*. Formal Methods in System Design 36(2), pp. 97–113.

[18] E.G. Coffman (1976): *Computer and Job-shop Scheduling Theory*. Wiley.

[19] A. David, K.G. Larsen, A. Legay, M. Mikucionis, D.B. Poulsen, J. van Vliet & Z. Wang (2011): *Statistical Model Checking for Networks of Priced Timed Automata*. In: FORMATS, pp. 80–96. Available at http://dx.doi.org/10.1007/978-3-642-24310-3_7.

[20] Conrado Daws, Alfredo Olivero, Stavros Tripakis & Sergio Yovine (1995): *The Tool KRONOS*. In: Hybrid Systems, pp. 208–219.

[21] Aldric Degorre & Oded Maler (2008): *On Scheduling Policies for Streams of Structured Jobs*. In: FORMATS, pp. 141–154. Available at http://dx.doi.org/10.1007/978-3-540-85778-5_11.

[22] Stephen A. Edwards & Edward A. Lee (2003): *The semantics and execution of a synchronous block-diagram language*. Sci. Comput. Program. 48(1), pp. 21–42. Available at http://dx.doi.org/10.1016/S0167-6423(02)00096-5.

[23] U. Fahrenberg & S. Tripakis, editors (2011): *Formal Modeling and Analysis of Timed Systems - 9th International Conference, FORMATS*. LNCS 6919, Springer. Available at http://dx.doi.org/10.1007/978-3-642-24310-3.

[24] M. Gries (2004): *Methods for Evaluating and Covering the Design Space during Early Design Development*. Integration, the VLSI Journal 38(2), pp. 131–183.

[25] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri & Pravin Varaiya (1998): *What's Decidable about Hybrid Automata?* J. Comput. Syst. Sci. 57(1), pp. 94–124. Available at http://dx.doi.org/10.1006/jcss.1998.1581.

[26] Jean-Francois Kempf, Marius Bozga & Oded Maler (2011): *Performance Evaluation of Schedulers in a Probabilistic Setting*. In Fahrenberg & Tripakis [23], pp. 1–17. Available at http://dx.doi.org/10.1007/978-3-642-24310-3_1.

[27] Yonit Kesten, Amir Pnueli, Joseph Sifakis & Sergio Yovine (1992): *Integration Graphs: A Class of Decidable Hybrid Systems*. In: Hybrid Systems, pp. 179–208. Available at http://dx.doi.org/10.1007/3-540-57318-6_29.

[28] H. Kopetz (2011): *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer.

[29] K.G. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, T. Hune, P. Pettersson & J. Romijn (2001): *As Cheap as Possible: Efficient Cost-Optimal Reachability for Priced Timed Automata*. In: CAV.

[30] K.G Larsen, P. Pettersson & W. Yi (1997): *UPPAAL in a nutshell*. International Journal on Software Tools for Technology Transfer (STTT) 1(1), pp. 134–152.

[31] C.L. Liu & James Layland (1973): *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*.

[32] J. W. S. Liu (2000): *Real-Time Systems*. Prentice-Hall.

[33] Oded Maler (2011): *On Under-Determined Dynamical Systems*. In: EMSOFT, pp. 89–96.

[34] M. Massink & G. Norman, editors (2011): *Proceedings Ninth Workshop on Quantitative Aspects of Programming Languages*. EPTCS 57. Available at http://dx.doi.org/10.4204/EPTCS.57.

[35] C. Palamidessi & A. Riska, editors (2011): *Eighth International Conference on Quantitative Evaluation of Systems, QEST 2011*. Available at http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6041098.

[36] Andy D. Pimentel, Cagkan Erbas & Simon Polstra (2006): *A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels*. IEEE Trans. Computers 55(2), pp. 99–112. Available at http://doi.ieeecomputersociety.org/10.1109/TC.2006.16.

[37] STMicroelectonics & CEA (2010): *Platform 2012: A Many-core programmable accelerator for Ultra-Efficient Embedded Computing in Nanometer Technology*. Technical Report. Available at http://www.2parma.eu/images/stories/p2012_whitepaper.pdf.

[38] Sander Stuijk, Marc Geilen & Twan Basten (2006): *SDF^3: SDF For Free*. In: *ACSD*, pp. 276–278. Available at http://doi.ieeecomputersociety.org/10.1109/ACSD.2006.23.

[39] William Thies, Michal Karczmarek & Saman P. Amarasinghe (2002): *StreamIt: A Language for Streaming Applications*. In: *CC*, pp. 179–196. Available at http://dx.doi.org/10.1007/3-540-45937-5_14.

[40] Farn Wang (2004): *Efficient verification of timed automata with BDD-like data structures*. *STTT* 6(1), pp. 77–97. Available at http://dx.doi.org/10.1007/s10009-003-0135-4.

[41] Håkan L. S. Younes & Reid G. Simmons (2002): *Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling*. In: *CAV*, pp. 223–235. Available at http://dx.doi.org/10.1007/3-540-45657-0_17.

[42] S. Yovine (1997): *KRONOS: A Verification Tool for Real-Time Systems*. *STTT* 1(1-2), pp. 123–133. Available at http://link.springer.de/link/service/journals/10009/bibs/7001001/70010123.htm.