**IP/SOC 2006**

Session: High Level / TLM

# Distributed software behaviour analysis through the MPSoC design flow

Julien BERNARD
INRIA/MOAIS
Montbonnot, France
julien.bernard@imag.fr

Serge DE PAOLI
STMicroelectronics
Crolles, France
serge.de-paoli@st.com

Fabrice SALPÊTRIER
IMAG/LSR
Grenoble, France
fabrice.salpetrier@imag.fr

**Abstract**: The complexity of developing Systems-on-Chip (Soc) is increasing continuously, but the productivity of hardware and software developers is not growing at a comparable pace. As a consequence, the conception of a new SoC can take a few years and software can't wait its availability. Therefore, on the one hand, software development is carried out at the simulation level. On the other hand, as the SoCs have more and more cores, we have to find innovative techniques for programming them and observing the behaviour of the programs.

This leads to introduce a programming model allowing to run a complex application and to measure its behaviour with regards to the SoC model during the design phase.

Our work is focusing on two main domains. The first one is the programming model, based on the work-stealing paradigm. The second one is a profiler technology probing the SoC model, with a knowledge of the programming model, and giving information on the behaviour of the application. Typical usage of this tool is to provide views and statistics regarding hardware resources, as well as embedded software behaviour. Our objective is to be able to target both programming model and profiling tools for a variety of target system (SMP, simulators or real SoCs). This paper is presenting this two main objectives.

## 1 Work-stealing: a programming model for parallel computing

### 1.1 Principles of work-stealing

The work-stealing scheduling is based on a greedy scheme: an idle processor can steal a task that is ready on another processor but not yet executed. Given an application, we call $T_p$ the execution time on $p$ processors, $T_1$ the sequential time of the application, $T_\infty$ the execution time of the application on an infinite number of processors. Then, according to Graham [8], the execution time on $p$ processors $T_p$ without the cost of scheduling time verifies:

$$T_p < \frac{T_1}{p} + T_\infty \quad (1)$$

Efficient work-stealing schedulers minimizes the cost of scheduling by generating parallelism only when required i.e. only when a processor becomes idle: this is known as the *work-first principle*. The number of idles is bounded by $T_\infty$ on each processor, so the scheduling overhead is bounded by $O(p.T_\infty)$, which is negligible if the application has a high level of parallelism (i.e. $T_\infty \ll T_1$).

Initially developped for SMP architectures [6], the principle has been extended to processors with different speeds [1] and then to distributed architecture [11], SMP clusters and heterogeneous grids [10].

We are focusing on a particular usage of work-stealing : adaptive applications. The idea is to use two concurrent algorithms, a fast sequential algorithm and a parallel algorithm, and to generate parallelism on-the-fly only when required. The coupling of two algorithms using work-stealing has already been studied for the iterated product problem [4] and for the prefix computation problem [12].

## 1.2 An example of simple adaptive application

If $n$ is an integer and $F$ is a function taking a single argument, the problem is to calculate $F(i)$ for $i$ from 0 to $n-1$. Moreover, we assume that the time for the computation of $F(i)$ depends on $i$.

The sequential algorithm for computing this problem is very simple, it consists in a simple loop. For our application, we use a modified version of the loop. If the range to be computed is $[a,b[$, then sequential algorithm reserves a sub-range of length $\log(b-a)$ at the beginning of the range and computes them with a loop. It does this as long as the range is not empty. If the range is empty, then, the parallel part of the algorithm takes place. It searches for a non-empty range $[a,b[$ among the others processors. When it finds one, it steals a sub-range of length $(b-a)/2$ at the end of the range and then computes this sub-range using the sequential algorithm. To explain these choices, we must calcultate $T_\infty$.

We call $\tau_F$ the maximum time for the computation of $F(i)$ and $\tau_S$ the time of a steal. For this algorithm, $T_1(n) = n.\tau_F = O(n)$ as a single processor will only compute its range using a simple loop. As for $T_\infty$, we must imagine an infinite number of processor. The first one has the total range of length $u_0 = n$ and computes a sub-range of length $\log u_0 = \log n$. Then another processor will steal half of the rest i.e $u_1 = (u_0 - \log u_0)/2$ and will compute a sub-range of length $\log u_1$, and so on recursively until computing a sub-range of length 1. After $k$ steals, the sub-range length is $u_k = (u_{k-1} - \log u_{k-1})/2$, and necesseraly $k \leq \log n$. So, $T_\infty(n) = \max_{0 \leq k \leq \log n}(\tau_F . \log u_k + \tau_S . k)$. As we know that $u_k < n/2^k$, we can say that $T_\infty(n) \leq \max_{0 \leq k \leq \log n}(\tau_F . \log n/2^k + \tau_S . k)$. In conclusion :

$$T_\infty \leq \max(\tau_F, \tau_S) \log n = O(\log n)$$

## 1.3 Why work-stealing is suitable for embedded software?

The work-stealing principle has many advantages regarding embbeded software.

Adaptive: Due to its dynamic scheduling, work-stealing adapts to every situation to offer the best finish time. It supports a great variation of data, and even a loss of a unit, with theoretically [1] and experimentally [6] [2] guaranteed performace, which makes it a predictable model.

Robust: The scheduling is totally decentralised, all the units play the same role, no unit is privileged. Moreover, it tries to minimize the number of communications between units. Thus, this model prevents a loss of a central unit and contention problems on communication channels.

Long term: Work-stealing suits today's platform with a few processors but also tomorrow's platform with tens of processors. The model is independent from

the underlying architecture and is scalable. Experiments on clusters and grids show that it can support a great number of processors.

So work-stealing offers a novel approach for embedded systems as they contain more and more processor cores. Such a technique allows the programmer to avoid the constraints of the architecture.

# 2 A tool for behaviour prediction and performance evalutation

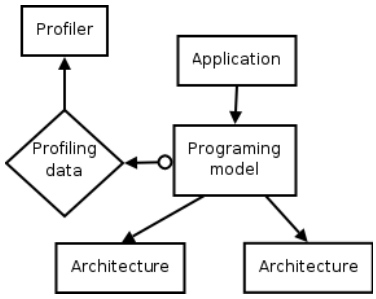


Figure 1: Profiling through a programming model

## 2.1 Profiling services through the programming model

Profiling an application running on multiple processors consist in two main parts: local profiling for improving local functions on a single processor, and global profiling for improving the overall behaviour and performance of the application. We are only interested in the second part as the first part can be achieved with traditional tools.

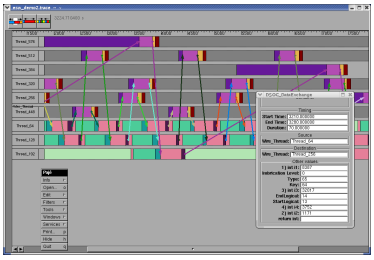

Figure 2: Pajé: an exemple of visualisation tool for parallel applications. This example has been obtained with a multi-DSP SystemC simulation platform

For large scale computations on clusters and grids, many tools already exist such as Vampir [3] or Pajé [5] (see Fig. 2). They are mainly based on MPI (Message Passing Interface) [9]. The message passing paradigm is not the same as the work-stealing paradigm, yet they share common points in the way applications using these paradigms can be profiled.

The common goal is to get the relevant information from the programming model (see Fig. 1) to see the behaviour of the application and its performance. We want to give information in the context of the user's program, thanks to a generic instrumentation of the programming model. In MPI, the scheduling is active, the communications and task creations are initiated by the programmer so that the behaviour of the application is always the same. As for work-stealing, the scheduling is passive, the communications and task creations are initiated by the middleware. The scheduling can differ from an execution to another. But the way to watch the task and the communication remains the same (see Fig. 3), the interpretation could differ somehow.

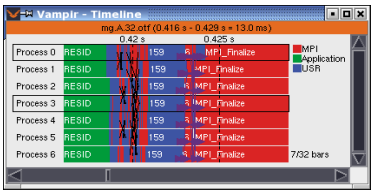


Figure 3: Vampir: a tool for the visualisation of MPI applications

## 2.2 Multi-level probing for a better profiling

Our main contribution to these techniques is to apply them in the context of embedded software development tools, more particularly focusing on SystemC/TLM simulation platforms.

As the programming model does not depend on the underlying architecture, it is possible to test the application on any architecture that the model supports. So we don't need the final SoC to have a quite precise idea of the performance of the application, we can make more and more accurate measures as we use architectures that are closer to the final SoC.

An added value of this approach is to be available across the design flow. For example, a Symetric Multi-Processor (SMP) machine can be used to observe the global behaviour of the application. Then SystemC level (not cycle accurate) can be used to refine the first observation. Then cycle-accurate simulation provides more precise performance measures. And finally we can use test boards to tune precisely the application.

We can also get information from the target architecture, as a complement of the information from the programing model. For example, in TLM platforms, it is possible to record transactions. Then we obtain multiple levels of information combining: architecture information, programming model information and application information.

The profiling tool is in fact the combination of platform probing, the programming model knowledge and the visuali-

sation tool. The programming model has an abstraction of the architecture, the platform does not know the embedded software. But the profiler can combine information from both and it gives relevant information to the programmer. In the end, the programmer can focus on its application.

### 2.3 Measures and visualisation

In our programming model, work-stealing, and our main architecture target, SystemC/TLM platforms, we can combine many useful informations.

At the programming model level, we know when a steal is requested and when it succeeds, a steal always generates a communication between two nodes. Then, after a successful steal, a new task starts on the stealer processor and makes computations until the next steal, so we can determine the beginning and the end of a task life. We can also measure the time for each task. At the TLM/SystemC level, we can intercept the raw communications between components (see Fig. 2) which correspond to writes in memory or messages between two processors. Knowing we use a work-stealing model, we can correlate the raw communications with the steal request and task creation.

We are currently using a platform with a host processor (e.g. a microcontroller) and two DSPs, simulated with SystemC/TLM. The processors communicate through two mailboxes and interrupts. On top of that, a small hardware adaptation layer provides an API. The embedded software, based on the adaptive work-stealing programming model, is the same on any processor as it does not depend on the underlying architecture. As a consequence, the application is independent from the processor types (typically, a DSP is much more powerful than a microcontroller), from the number of processors (we can add or remove a processor), from the characteristic of the processors (e.g. speed). The programming model dynamically adapts the application to the platform architecture.

## 3 Conclusion

We are proposing tools and techniques to facilitate embedded software development. Our techniques are consisting in a programming model, based on the work-stealing paradigm, and a profiler technology combining target probing and programming model awareness. Our main contribution to these techniques is to apply them in the context of embedded software development tools, more particularly focusing on SystemC/TLM simulation platforms. In such an environment, the programming model can take care of load balancing according to hardware constraints. Then, thanks to the profiler wich collects data at different levels (hardware and software), we can give the programmer an overall view of the application. At the moment, we have some preliminary successful results, the next step is to build a full demonstration of this concept in order to define a future generation of tools. We also want to show that the programming model we are using is suitable for a wide range of embedded applications.

## References

[1] Michael A. Bender and Michael O. Rabin. Scheduling Cilk multithreaded parallel programs on processors of different speeds. In *ACM Symposium on Parallel Algo-*

*rithms and Architectures*, pages 13–21, 2000.

[2] Robert D. Blumofe and Dionisos Papadopoulos. HOOD: A user-level threads library for multiprogrammed multiprocessors. Technical report, The University of Texas at Austin, October 1998.

[3] Holger Brunst, Dieter Kranzlmüller, and Wolfgang E. Nagel. Tools for scalable parallel program analysis - vampir vng and dewiz. In *DAPSYS*, pages 93–102, 2004.

[4] El-Mostafa Daoudi, Thierry Gautier, Aicha Kerfali, Rémi Revire, and Jean-Louis Roch. Algorithmes parallèles à grain adaptatif et applications. *Technique et Science Informatiques*, 24:1—20, 2005.

[5] Jacques Chassin de Kergommeaux and Benhur de Oliveira Stein. Pajé: An extensible environment for visualizing multi-threaded programs executions. In *Euro-Par '00: Proceedings from the 6th International Euro-Par Conference on Parallel Processing*, pages 133–140, London, UK, 2000. Springer-Verlag.

[6] Matteo Frigo, Charles E. Leiserson, and Keith H. Randall. The implementation of the Cilk-5 multithreaded language. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'98)*, June 1998.

[7] François Galilée, Jean-Louis Roch, Gerson Cavalheiro, and Matthias Doreille. Athapascan-1: On-line Building Data Flow Graph in a Parallel Language. In IEEE, editor, *International Conference on Parallel Architectures and Compilation Techniques, PACT'98*, pages 88–95, Paris, France, October 1998.

[8] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17(2):416–426, 1969.

[9] Andreas Knüpfer, Ronny Brendel, Holger Brunst, Hartmut Mix, , and Wolfgang E. Nagel. Introducing the open trace format (otf). In *Computational Science – ICCS 2006*, volume 3992 of *LNCS*, pages 526–533. Springer, 2006.

[10] Remi Revire. *Ordonnancement de graphe dynamique de tâches sur architecture de grande taille. Régulation par dégénaration séquentielle et distribuée.* PhD thesis, Institut National Polytechnique de Grenoble, September 2004.

[11] Jean-Louis Roch, Thierry Gautier, and Rémy Revire. Athapascan: API for Asynchronous Parallel Programming. Technical Report RT-0276, INRIA Rhône-Alpes, projet APACHE, February 2003.

[12] Jean-Louis Roch, Daouda Traore, and Julien Bernard. On-line adaptive parallel prefix computation. In *Euro-Par 2006 Parallel Processing*, volume 4128 of *Lecture Notes in Computer Science*. Springer-Verlag, 2006.