PAM-SoC: A Toolchain for Predicting MPSoC Performance*

Ana Lucia Varbanescu, Henk Sips, and Arjan van Gemund

Department of Computer Science, Delft University of Technology, The Netherlands
A.L.Varbanescu@tudelft.nl

Abstract. In the past, research on Multiprocessor Systems-on-Chip (MPSoC) has focused mainly on increasing the available processing power on a chip, while less effort was put into specific system-level performance analysis, or into behavior prediction. This paper introduces PAM-SoC, a light-weight performance predictor for MPSoC system-level performance. Being based on PAMELA, a static performance predictor for parallel applications, PAM-SoC can compute its prediction in seconds for cases when cycle-accurate simulation takes tens of minutes. The paper includes a set of PAM-SoC validation experiments, as well as two sets of experiments to show how PAM-SoC can be used for either application tuning or MPSoC platform tuning in early system design phases.

1 Introduction

Systems-on-Chips (SoCs) are built to answer the increased processing power requirements of real-time embedded applications by integrating (most of) the functions of a complete electronic system on a single chip [1]. Multiprocessor SoCs (MPSoCs) are SoCs that integrate several programmable processors, adding more flexibility and programmability to these devices. Currently, consumer electronics, automotive and dedicated industrial control systems are foreseen as the main consumers of MPSoC technology.

So far, MPSoC research was focused mainly on hardware issues, allowing designers to prove their skills in squeezing as much processing power as possible on a single chip. As a result, many different platforms have emerged [2, 3]: IXP2850 Network Processor (Intel), OMAP (Texas Instruments), Nexperia^{TM}(Philips), Nomadik^{TM}(STMicroelectronics), or Cell (IBM/Sony/Toshiba).

Currently, MPSoCs face a difficult challenge: predictable programmability in terms of performance. Unfortunately, integrating more resources on the same chip does not directly increase performance. It does, however, increase the analysis complexity and the software design time. On top of this complexity, the inherent *hardware imbalance* between the almost unlimited available processing power and the severely limited on-chip memory may induce performance gaps that have not been foreseen during design. Furthermore, MPSoCs lack dedicated performance analysis methodologies. Most of the current analysis is based

* The research is part of the Scalp project http://scalp.ewi.tudelft.nl funded by STW-Progress.

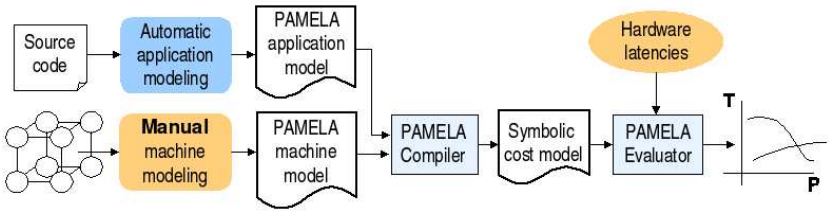


Fig. 1. The PAMELA symbolic cost estimation

on simulations, a time-consuming solution that also requires specific well-defined benchmarks (not yet available), and specific programming models (emerging [4]), in order to provide meaningful conclusions. In other words, current performance analysis is expensive and cumbersome, thus difficult to use in any system design flow feedback loops.

This paper presents PAM-SoC, a light semi-static performance prediction toolchain that computes system-level performance estimations for applications running on MPSoCs. PAM-SoC is based on PAMELA[5], a static performance prediction methodology for general purpose parallel platforms (GPPPs). By coupling an application model with the target machine model, PAMELA computes the lower bound of the execution time of the application on the target architecture. To address the specifics of MPSoCs, PAM-SoC includes new techniques for machine modeling and tools for gathering memory behavior statistics. In its prediction, PAM-SoC trades accuracy for estimation speed: in cases when cycle-accurate simulation would take tens of minutes, application behavior can be estimated in tens of seconds. Thus, PAM-SoC can be part of the MPSoC design flow, for either application or architecture tuning.

The paper is organized as follows: Section 2 briefly presents the PAMELA methodology. Section 3 introduces the PAM-SoC predictor, discussing the application and machine modeling in detail. Section 4 presents the validation process of PAM-SoC and two interesting usage scenarios. Section 5 presents related work, while Section 6 draws the conclusions and presents future work directions.

2 PAMELA Methodology

PAMELA (PerformAnce ModEling LAnguage) [6] is a performance simulation formalism that facilitates symbolic cost modeling, featuring a modeling language, a compiler, and a performance analysis technique. The PAMELA model of a Series-Parallel (SP) program[7] is a set of explicit, algebraic performance expressions in terms of program parameters (e.g., problem size), and machine parameters (e.g., number of processors). These expressions are automatically *compiled* into a *symbolic cost model*, that can be further reduced and compiled into a *time-domain cost model* and, finally, evaluated into a *time estimate*. Note that PAMELA models trade prediction accuracy for the lowest possible solution complexity. Fig. 1 presents the PAMELA methodology.

Modeling. The PAMELA modeling language is a process-oriented language designed to capture concurrency and timing behavior of parallel systems. Data

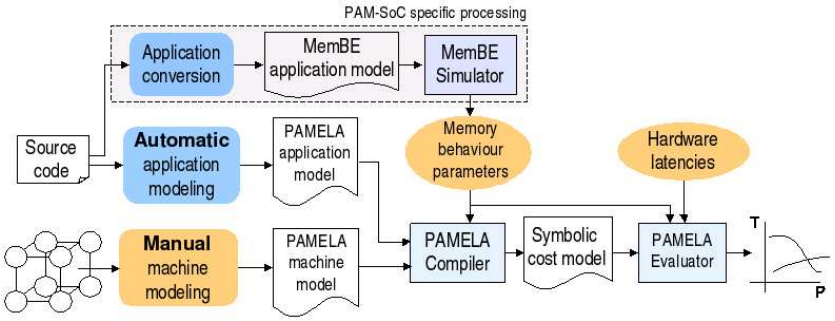


Fig. 2. The PAM-SoC toolchain

computations from the original source code are modeled into the *application model* in terms of their resource requirements and workload. The available resources and their usage policies are specified by the *machine model*.

Any PAMELA model is written as a set of process equations, composed from `use` and `delay` basic processes, using sequential, parallel, and conditional composition operators. The construct `use(Resource,`t`)` stands for exclusive acquisition of `Resource` for t units of (virtual) time. The construct `delay(`t`)` stalls program execution for t units of (virtual) time. A *machine model* is expressed in terms of available resources and an abstract instruction set (AIS) for using these resources. The *application model* of the parallel program is implemented using an (automated) translator from the source instruction set to the machine AIS. The example below illustrates the modeling of a block-wise parallel addition computation $y = \sum_{i=1}^{N} x_i$ on a machine with `P` processors and shared memory `mem`:

```
// application model:                    // machine model
   par (p=1,P) {                           load=use(mem,t_access)
       seq (i=1,N/P) { load ; add } ;      add=delay(t_add)
       store }
```

Symbolic Compilation and Evaluation. A PAMELA model is translated into a *time-domain performance model* by substituting every process equation by a numeric equation that models the execution time associated with the original process. The result is a new PAMELA model that only comprises numeric equations, as the original `process` and `resource` equations are no longer present. The PAMELA compiler can further reduce and evaluate this model for different numerical values of the parameters, computing the lower bound of the application execution time. The analytic approach underlying the translation, together with the algebraic reduction engine that drastically optimizes the evaluation time, are detailed in [8].

3 The PAM-SoC toolchain

Using PAMELA for MPSoC performance predictions is quite difficult because of the architecture and application modeling efforts required. Details that can be

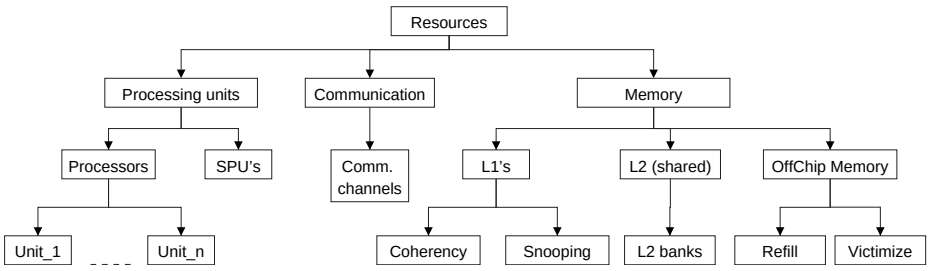


Fig. 3. The extended set of resources to be included in a PAM-SoC machine model

safely ignored for GPPP models, as they do not have a major influence on the overall performance, may have a significant influence on MPSoC behavior. As a consequence, for correct modeling of MPSoC applications and architectures, we have extended PAMELA with new techniques and additional memory behavior tools. The resulting PAM-SoC toolchain is presented in Fig. 2. In this section we will further detail its specific components.

3.1 MPSoC Machine Modeling

The successfull modeling of a machine architecture starts with accurate detection of its important *contention points*, i.e., system resources that may limit performance when used concurrently. Such resources can be modeled at various degrees of detail, i.e., *granularities*, by modeling more or less from their internal sub-resources. The model granularity is an essential parameter in the speed-to-accuracy balance of the prediction: a finer model leads to a more accurate prediction (due to better specification of its contention points), but it is evaluated slower (due to its increased complexity). Thus, a model *granularity boundary* should be established for any architecture, so that the prediction is is still fast and sufficiently accurate. This boundary is usually set empirically and/or based on a set of validation experiments.

Previous GPPPs experiments with PAMELA typically used coarse models, based on three types of system resources: the processing units, the communication channels and the memories. For MPSoC platforms, we have established a new, extended set of resources to be included in the machine model, as seen in Fig. 3. The new granularity boundaries (the leaf-level in the resource tree in Fig. 3) preserve a good speed-to-accuracy balance, as proved by various experiments we did [9], while allowing drastic simplification of the MPSoC machine modeling procedure. Some additional comments with respect to the machine modeling are the following:

- When on-chip programmable processors have subunits able to work in parallel, they should be modeled separately, especially when analyzing applications that specifically stress them.
- The communication channels require no further detailing for shared-bus architectures. For more complex communication systems, involving networks-on-chips or switch boxes, several channels may be acting in parallel. In this case, they have to be detected and modeled separately.

- The memory system is usually based on individual L1's, an L2 shared cache (maybe banked) and off-chip memory (eventually accessed by dedicated L2 Refill and L2 Victimize engines). If hardware snooping coherency is enforced, two more dedicated modules become of importance: the Snooping and the Coherency engines. Any of these components present in the architecture must be also included in the model.

After identifying model resources, the AIS has to be specified as a set of rules for using these resources, requiring (1) in-depth knowledge on the functionality of the architecture, for detecting the resource accesses an instruction performs, and (2) resource latencies. As an example, Table 1 presents a snippet from a (possible) AIS, considering an architecture with: several identical programmable processors (`Procs(p)`), each one having parallel arithmetic (`ALU(p)`) and multiplication (`MUL(p)`) units and its own `L1(p)` cache; several Specialized Functional Units (`SFU(s)`); a shared `L2` cache, banked, with dedicated `Refill` and `Victimize` engines; virtually infinite off-chip memory (`mem`).

Table 1. Snippet from an AIS for a generic MPSoC

Operation	**Model**
p: ADD	`use(ALU(p),` t_{ADD}`)`
p: MUL	`use(MUL(p),` t_{MUL}`)`
s: EXEC	`use(SFU(s),` t_{SFU}`)`
p: accessL1(addr)	`use(L1(p),` $t_{L1}^{hit} * h_{L1}^{ratio} + t_{L1}^{miss} * (1 - h_{L1}^{ratio})$`)`
p: accessL2(addr)	`use(L2(bank(addr)),` $t_{L2}^{hit} * h_{L2}^{ratio} + t_{miss} * (1 - h_{L2}^{ratio})$`)`
p: refillL2(addr)	`use(Refill,` t_{Mem}^{RD} `)`
p: victimizeL2(addr)	`use(Victimize,` $victimization^{ratio} * t_{Mem}^{WR}$ `)`

p: READ(addr)	`accessL1(addr);`
	`if (missL1) { accessL2(addr);`
	`if (missL2) {`
	`if (victimize) victimizeL2(addr);`
	`refillL2(addr)}}`

The cache hit/miss behavior cannot be evaluated using the cache (directory) state, because Pamela, being algebraic, is a state-less formalism. Thus, we compute a probabilistic average cache latency, depending on the cache hit ratio, h^{ratio}, and on the hit/miss latencies, t^{hit} and t^{miss}. Also the `if` branches in the `READ(addr)` model are addressed in a probabilistic manner. For example, `if(missL1)` is replaced by a quantification with $(1 - h_{L1}^{ratio})$, which is the probability that this condition is true. All these probabilistic models are based on *memory behavior parameters* which are both application- and architecture-dependent. PAM-SoC uses an additional tool for computing these parameters, which is presented in Section 3.3.

3.2 Application modeling

Translating an application implemented in a high-level programming language to its Pamela application model (as well as writing a Pamela model from

scratch) implies two distinct phases: (1) modeling the application as a series-parallel graph of processes, and (2) modeling each of the processes in terms of PAMELA machine instructions. However, modeling an existing application to its PAMELA model is a translation from one instruction set to another, and it can be automated if both instruction sets are fully specified as exemplified in [5].

3.3 The memory statistics

For computing its prediction, PAM-SoC uses two types of numerical parameters: (1) the hardware latencies (measured under no-contention conditions), and (2) the memory statistics. While the former have been also required by GPPP models, the latter become of importance mainly for modeling MPSoC platforms.
The hardware latencies are fixed values for a given architecture and can be either obtained from the hardware specification itself (i.e., theoretical latencies) or by means of micro-benchmarking (i.e., measured latencies). We have based our experiments on the theoretical latencies.
The memory statistics are both machine- and application-dependent, and they have to be computed/evaluated on a per-application basis. For this, we have built MemBE, a custom light-weight *Memory system Behavior Emulator* able to obtain memory statistics like cache hit ratios, snooping success ratios, or vitimization ratios, with good speed and satisfactory accuracy. MemBE is built as a multi-threaded application that permits the (re)configuration of a custom memory hierarchy using the memory components supported by PAM-SoC. MemBE emulates the memory system of the target architecture and executes a memory-skeleton version of the analyzed application[1]. The memory skeleton is stripped of any data-processing, which allows MemBE to run faster and to focus exclusively on monitoring the application data-path.

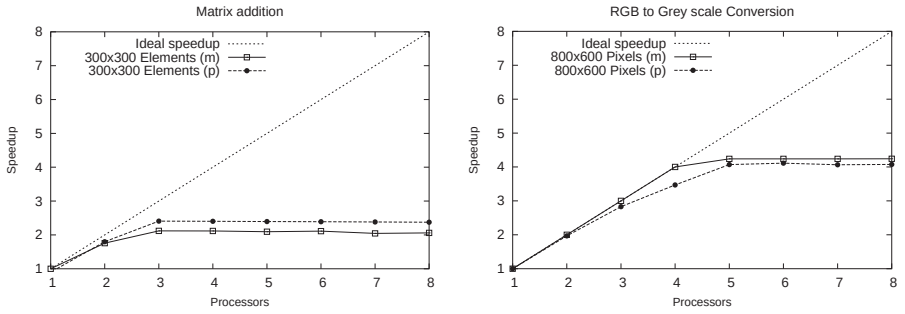
4 Experiments and results

In this section we present the validation experiments for our PAM-SoC toolchain, as well as two sets of design-space exploration experiments, one for architecture tuning and one for application tuning, respectively.

4.1 Validation experiments

The validation process of PAM-SoC aims to prove its abilities to correctly predict application behavior on a given MPSoC platform. For these experiments, we have modeled the Wasabi platform, one tile of the CAKE architecture from Philips [10, 11]. A Wasabi chip is a shared-memory MPSoC, having 1-8 programmable processors, several SFUs, and various interfaces. The tile memory hierarchy has three levels: (1) private L1's for each processor, (2) one shared on-chip L2 cache, available to all processors, and (3) one off-chip memory module. Hardware consistency between all L1's and L2 is enforced. For software

[1] Currently, the application simplification from the source code to the memory-skeleton is done by hand. In principle, we believe that PAMELA and MemBE can both start from a common, autmatically-generated application model.



(a) (b)

Fig. 4. Predicted and measured speedup for (a) matrix addition and (b) RGB-to-Grey conversion

support, Wasabi runs eCos[2], a modular open-source Real-Time Operating System (RTOS), which has embedded support for multithreading. Programming is done in C/C++, using eCos synchronization system calls and the default eCos thread scheduler.

The simulation experiments have been run on Wasabi's configurable cycle-accurate simulator, provided by Philips. For our experiments, we have chosen a fixed memory configuration (L1's are 256KB, L2 is 2MB, and the off-chip memory is 256MB) and we have used up to 8 identical Trimedia processors[3].

For validation, we have implemented a set of six simple benchmark applications, each of them being a possible component of a more complex, real-life MPSoC application. These applications are: (1) **element-wise matrix addition** - memory intensive, (2) **matrix multiplication** - computation intensive, (3) **RGB-to-YIQ conversion** - a color-space transformation filter, from the EEMBC Consumer suite[4], (4) **RGB-to-Grey conversion** - another color-space transformation, (5) **high-pass Grey filter** - an image convolution filter, from the EEMBC Digital Entertainment suite[5], and (6) **filter chain** - a chain of three filters (YIQ-to-RGB, RGB-to-Grey, high-pass Grey) successively applied on the same input data. All benchmark applications have been implemented for shared-memory (to comply with the Wasabi memory system), using the SP-programming model and exploiting data-parallelism only (no task parallelism, which is a natural choice for the case of these one-task applications).

The results of PAM-SoC prediction and Wasabi simulation for matrix addition and RGB-to-Grey transformation are presented together in Fig. 4. Due to space limitation, we have only included these two graphs, for the two applications that clearly exhibit memory contention and therefore show the prediction abilities of PAM-SoC. The complete set of graphs (for all the applications) and experiment results are presented in [9]. For all the applications, the behavior

[2] http://ecos.sourceware.org/
[3] TriMedia is a family of Philips VLIW processors optimized for multimedia processing
[4] http://www.eembc.org/benchmark/consumer.asp
[5] http://www.eembc.org/benchmark/digital_entertainment.asp

Table 2. Simulation vs. prediction times [s]

Application	**Data size**	T_{sim}	T_{MemBE}	T_{Pam}	T_{PAMSoC}	**Speed-up**
MADD	3x1024x1024 words	94	2	1	3	31.3
MMUL	3x512x512 words	8366	310	2	312	26.8
RGB-to-YIQ	6x1120x840 bytes	90	7	4	11	8.1
RGB-to-Grey	4x1120x840 bytes	62	3	1	4	15.5
Grey-Filter	2x1120x840 bytes	113	6	4	10	11.3
Filter chain	8x1120x840 bytes	347	20	12	32	10.8

trend is correctly predicted by PAM-SoC. The average error between simulation and prediction is within 19%, while the maximum is less than 25% These deviations are due to (1) the differences between the theoretical Wasabi latencies and the ones implemented in the simulator (50-70%), (2) the averaging of the memory behavior data, and (3) the PAMELA accuracy-for-speed trade. While Fig. 4 demonstrates how PAM-SoC is accurate in terms of application behavior, Table 2 emphasizes the important speed-up of PAM-SoC prediction time ($T_{PAMSoC} = T_{MemBE} + T_{Pam}$) compared to the cycle-accurate simulation time, T_{sim}, for the considered benchmark applications and the largest data sets we have measured. While a further increase of the data set size leads to a significant increase for T_{sim} (tens of minutes), it has a minor impact on T_{Pam} (seconds) and leads to a moderate increase of T_{MemBE} (tens of seconds up to minutes). Because MemBE is at its first version, there is still much room for improvement, by porting it on a parallel machine and/or by including more aggressive optimizations. In the future, alternative cache simulators or analytical methods (when/if available) may even replace MemBE for computing memory statistics.

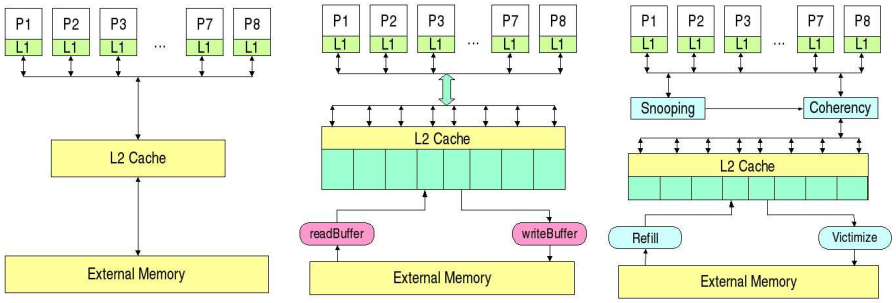
4.2 Design space exploration

PAM-SoC can be successfully used for early design space exploration, both for *architecture tuning*, where architectural choices effects can be evaluated for a given application, and for *application tuning* where application implementation choices effects can be evaluated for a given architecture.

Architecture tuning. For *architecture tuning*, we have considered a hypothetical system with up to eight identical programmable processors, each processor having its own L1 cache, a fast shared on-chip L2 cache and practically unlimited external memory (i.e., off-chip). We have modeled three variants of this hypothetical system, named **M1**, **M2**, and **M3**, and we have estimated the performance of a benchmark application (matrix addition, implemented using 1-dim block distribution) on each of them. In this experiment we have chosen to tune the memory configuration of the architecture for these different models, but different processing and/or communication configurations can be evaluated in a similar manner.

M1 is a basic model presented in Fig. 5(a), being a good starting point for modeling any MPSoC architecture. Its key abstraction is the correct approximation of the off-chip memory latencies - for both `READ` and `WRITE` operations.

M2 is an improved version of **M1**, presented in Fig. 5(b). It has multiple interleaved banks for the L2 cache (providing concurrent access to addresses that



(a) (b) (c)

Fig. 5. The hypothetical MPSoCs: (a) **M1**, (b) **M2**, (c) **M3**

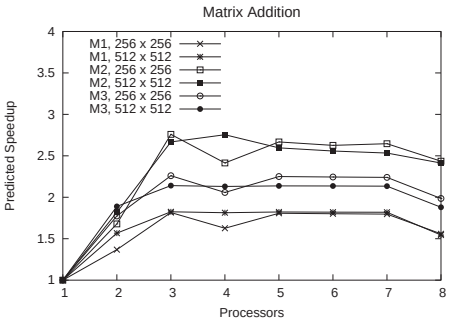


Fig. 6. Architecture tuning results (predicted)

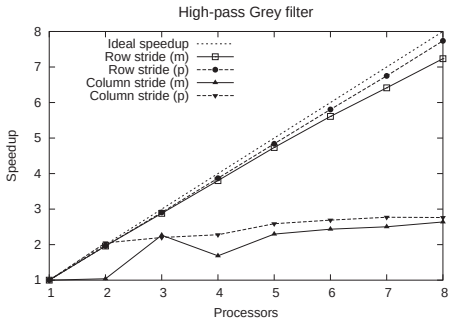


Fig. 7. Application tuning results (predicted vs. measured)

do not belong to the same bank) and buffered memory access to the external memory. Due to these changes, we expect the execution of an application to speed-up on **M2** compared to **M1**. The **M2** model can be adapted to suite any MPSoC architecture with shared banked on-chip memory, if the number of banks, the sizes of the on-chip buffers, and the banking interleaving scheme are adapted for the target machine.

M3, presented in Fig. 5(c), has hardware-enforced *cache coherency*, based on a snooping protocol. The snooping mechanism may increase performance for successful snoopings, when an eventual L1-to-L1 transfer is replacing a slower L2-to-L1 fetch. On the other hand, the L1-to-L2 coherency writes may slow down the application execution. Furthermore, due to the L2-to-Memory transfers performed by the `Victimize` (for `WRITE`) and `Refill` (for `READ`) engines, two new contention points are added. Overall, because matrix addition has almost no successfull snoopings, the application execution on **M3** is slowed down compared to its run on **M2**. **M3** covers the most complex variants of a three-level memory hierarchy to be found in shared memory MPSoCs. Fig. 6 shows that PAM-SoC is able to correctly (intuitively) predict the behavior of the given application on these three models. Unfortunately, we could not run validation experiments for the same data sets because these would require cycle-accurate simulators for

M1, M2, and M3[6], which are not available. However, previous PAMELA results [5] provide ample evidence that PAM-SoC will not predict a wrong relative order in terms of performance: it either correctly identifies the best architecture for the given application, or it cannot distinguish one single best architecture.

Application tuning The aim of *application tuning* experiments is to try evaluate several possible implementations of the same application and choose the best one. To prove the use of PAM-SoC for application tuning, we have used the model of the Wasabi platform and we have implemented the high-pass Grey filter mentioned in Section 4.1 using row and column stride distribution. For this example, based on the PAM-SoC predictions (which are validated by the simulation results), we can decide that row-stride distribution is the best for the Wasabi architecture. Similarly, the experiments like, for example, matrix addition (see Fig. 4(a)), can detect the maximum number of processors to be used for the computation. Fig. 7 shows how PAM-SoC correctly detects the application implementation effects on the given architecture. The simulation results that validate the predictions are also included in the graph.

5 Related Work

Because performance prediction and analysis specifically targeted to MPSoCs is still young, we relate our work to more mature adjacent fields, namely (1) performance prediction for parallel applications, (2) MPSoC performance evaluation and design space exploration techniques, and (3) methods for estimating embedded systems performance.

For static performance prediction of (scientific) parallel applications, we direct the reader to PAMELA and its related work, to be found in [5, 12, 13]. Most of these methodologies are difficult to adapt for MPSoCs and their applications, mainly because of the sheer complexity of the hardware platforms. On the practical side, we name the PERC framework [14], close to PAM-SoC in both objectives and realization, as it aims to estimate parallel application behavior by combining machine and application models with the aid of behavior statistics; however, because PERC is intended for scientific applications running on high-performance computers, its analysis granularity is too coarse for direct applicability to MPSoCs.

MPSoC performance analysis is still relying heavily on simulation. MPSoC designers and producers deliver proprietary toolchains, while generic frameworks, like [15, 16], provide complex solutions for hardware/software co-simulation and integrated performance estimation. Although very accurate, these simulations are still expensive in terms of computation time. Many hybrid performance estimation techniques have been developed in the context of design space exploration of both MPSoC-specific applications and architectures [17, 18]. For example, an interesting hybrid co-simulation solution, similar to PAM-SoC in combining performance estimation with simulation techniques, is presented in [19]. The reduction in complexity is obtained by simulating the architecture

[6] Wasabi is a variant of M3, but its simulator is implemented as a combination of M2 and M3

at the functional level, with approximate timing behavior which, compared to PAM-SoC, is much coarser. Other MPSoC performance analysis methods are dedicated to estimate the performance of MPSoC components, such as on-chip communication [20, 21] or memory systems [22]. An available solution for formal system-level performance verification of MPSoCs is presented in [23], but their approach aims to verify the performance of the hardware system, not to estimate its application-specific behavior.

Although there is no clear border between embedded systems and MPSoC platforms, most of the existing performance evaluation methods for embedded systems, like those presented in [24, 25], have not been tested/adapted for MPSoCs, so there are no clear results in this direction.

6 Conclusions and Future Work

In this paper we have presented PAM-SoC, the first toolchain (to the best of our knowledge) for MPSoCs semi-static performance prediction.

Static performance prediction for MPSoCs is motivated by its reduced cost, which allows it to be a part of the design loop. Even though static performance predictors trade accuracy for estimation cost, a behavior estimation within minutes is more valuable, in the early design phases, than an hours-long simulation with very precise results. We have shown how PAM-SoC is used to predict the performance of MPSoC platforms. To validate the methodology, we have modeled a real MPSoC platform and compared the PAM-SoC prediction results for a set of six benchmark applications with the simulation results. Furthermore, we have presented the successful results of PAM-SoC in two early design exploration use-cases, namely application tuning and architecture tuning. For the future, we plan to enhance PAM-SoC by exploring three directions: (1) to model and test more complex applications, for further validation/improvement of PAM-SoC, (2) to make the modeling process as automatic as possible, and (3) to investigate how can PAM-SoC become a truly *static* performance predictor.

Acknowledgements. We would like to thank Paul Stravers and Philips Research for providing the Wasabi simulator, and for the help and support we got for understanding the details of the architecture, support that allowed us to properly model the system.

References

1. Jerraya, A., Wolf, W.: Multiprocessor Systems-on-Chips. Morgan Kaufmann Publishers (2004)
2. Wolf, W.: The future of multiprocessor systems-on-chips. In: Proc. DAC'04, ACM Press (2004) 681–685
3. Kahle, J.A., Day, M.N., Hofstee, H.P., Johns, C.R., Maeurer, T.R., Shippy, D.: Introduction to the Cell multiprocessor. IBM Journal of Research and Development **49**(4/5) (2005)
4. Nijhuis, M., Bos, H., Bal, H.: Supporting reconfigurable parallel multimedia applications. In: EuroPAR'06. (2006)
5. van Gemund, A.: Symbolic performance modeling of parallel systems. IEEE TPDS (2003)

6. van Gemund, A.: Performance Modeling of Parallel Systems. PhD thesis, Delft University of Technology (1996)
7. Gonzalez-Escribano, A.: Synchronization Architecture in Parallel Programming Models. PhD thesis, Dpto. Informatica, University of Valladolid (2003)
8. Gautama, H., van Gemund, A.: Static performance prediction of data-dependent programs. In: Proc. WOSP'00, ACM (2000) 216–226
9. Varbanescu, A.L.: PAM-SoC experiments and results. Technical Report PDS-2006-001, (Delft University of Technology, http://www.pds.twi.tudelft.nl/reports/)
10. Stravers, P., Hoogerbrugge, J.: Homogeneous multiprocessing and the future of silicon design paradigms. In: Proc. VLSI-TSA'01. (2001)
11. Borodin, D.: Optimisation of multimedia applications for the Philips Wasabi multiprocessor system. Master's thesis, TU Delft (2005)
12. Adve, V.S.: Analyzing the behavior and performance of parallel programs. PhD thesis, Dept. of Computer Sciences, University of Wisconsin-Madison (1993)
13. Adve, V.S., Vernon, M.K.: A deterministic model for parallel program performance evaluation. Technical Report TR98-333 (1998)
14. Snavely, A., Carrington, L., Wolter, N., Labarta, J., Badia, R., Purkayastha, A.: A framework for performance modeling and prediction. In: Proc. Supercomputing '02, IEEE Computer Society Press (2002) 1–17
15. Mahadevan, S., Storgaard, M., Madsen, J., Virk, K.: ARTS: A system-level framework for modeling MPSoC components and analysis of their causality. In: Proc. MASCOTS '05, IEEE Computer Society (2005) 480–483
16. Pimentel, A.D.: The Artemis workbench for system-level performance evaluation of embedded systems. Int. Journal of Embedded Systems **1**(7) (2005)
17. Gries, M.: Methods for evaluating and covering the design space during early design development. Technical Report UCB/ERL M03/32, Electronics Research Lab, University of California at Berkeley (2003)
18. Kienhuis, B.: Design Space Exploration of Stream-based Dataflow Architectures. PhD thesis, Delft University of Technology (1999)
19. Baghdadi, A., Zergainoh, N.E., Cesario, W.O., Jerraya, A.A.: Combining a performance estimation methodology with a hardware/software codesign flow supporting multiprocessor systems. IEEE TSE **28**(9) (2002) 822–831
20. Loghi, M., Angiolini, F., Bertozzi, D., Benini, L., Zafalon, R.: Analyzing on-chip communication in a MPSoC environment. In: Proc. DATE '04, IEEE Computer Society (2004) 20752
21. Pande, P.P., Grecu, C., Jones, M., Ivanov, A., Saleh, R.: Performance evaluation and design trade-offs for Network-on-Chip interconnect architectures. IEEE TC **54**(8) (2005) 1025–1040
22. Loghi, M., Poncino, M.: Exploring energy/performance tradeoffs in shared memory MPSoCs: Snoop-based cache coherence vs. software solutions. In: Proc. DATE'05, IEEE Computer Society (2005)
23. Richter, K., Jersak, M., Ernst, R.: A formal approach to MpSoC performance verification. IEEE Computer **36**(4) (2003) 60–67
24. Lazarescu, M., Bammi, J., Harcourt, E., Lavagno, L., Lajolo, M.: Compilation-based software performance estimation for system level design. In: Proc. IEEE HLDVT'00. (2000)
25. Thiele, L., Wandeler, E.: Performance Analysis of Embedded Systems. In: The Embedded Systems Handbook. CRC Press (2004)