Evaluating Run-time Resource Management Policies for Multi-core Embedded Platforms with the EMME Evaluation Framework

Giovanni Mariani[1], Gianluca Palermo[2], Vittorio Zaccaria[2], Cristina Silvano[2]

[1] ALaRI – University of Lugano, Switzerland
[2] Politecnico di Milano, Italy
marianig@alari.ch, {gpalermo, zaccaria, silvano}@polimi.it

Abstract: Toady's embedded computing electronic products are based on multi-core platforms and they are capable to concurrently execute different applications. For these products it is of paramount importance that a Run-time Resource Management (RRM) system integrated in the Operating System (OS) arbiters about resource allocation to the active applications. The RRM should take decisions at run-time to maximize platform performance and minimizing non-functional costs such as power consumption or memory requirements. However, embedded system design covers a broad range of applications and the customer requirements are very different depending on the target device. In general there is not an unique RRM that best fits in all possible embedded scenarios.

This paper presents the **EMME Evaluation Framework**, an open source tool that provides a methodology and the accompanying infrastructure to quickly explore the effects of different RRM systems for a target use case scenario. The tool aims at the analysis of different figures of merit of the system being designed such as the applications' *response time*, system *throughput* and *power consumption*.

Different RRM modules are released with the framework. These modules implement different RRM policies that define how to allocate computing resources to the active applications while fitting in a power budget that is assumed assigned by other layers of the OS.

1 Introduction

The availability of many processors (or *computing elements*) on a single chip brought new challenges to system designers. In particular, efficient power management has become a primary factor for product success. This is evident for portable devices but has subtle yet important consequences on reliability and cooling costs for non-portable systems.

Traditional techniques for power management consider switching off or slowing down the frequency of computational elements which are underutilized [BBM00, IBC+06]. If the switching overhead is negligible and the performance is not saturated, it is possible to meet performance requirements with fewer active resources and lower power consumption.

When considering multiprogrammed multi-core scenarios where multiple applications are competing to access shared resources, traditional DVFS techniques [IBC+06, BHB+08] are not enough. The proposed *Efficient run-time resource Management for Multi-core Embedded systems* (EMME) framework shall also decide how to allocate available com-

puting resources to the active applications. In these scenarios, solving the RRM problem is a challenging task for the following reasons:

- **Dynamism of the system:** operating configurations providing high performance are typically power hungry hence they should be avoided unless strictly necessary. On the other hand, user behaviors are highly dynamic, unpredictable and unknown at design time. It is not possible to statically identify a setting of the operating parameters which maximizes system performance while fitting in a given power budget. To solve the RRM problem the operating parameters should be tuned at run-time once the system state and user requirements are known.
- **Complex system behaviors:** in a multiprogrammed multi-core scenario, the relationships between operating parameters and overall platform performance can be very complex when considering the resource distribution as a run-time tunable parameter. As shown in [SRS+95, MPSZ11], optimal processor assignment is very workload dependent (and is less than intuitive).
- **Complexity of the decision problem:** the problem of deciding which operating configuration to set for each active application (and thus for the system as a whole) belongs to the NP-hard class of problems [YCNCC06]. Hence, the problem is too complex to be solved exactly and only an approximate solution can be found. It is of crucial importance that heuristic algorithms implementing the run-time decision making should be fast in order to minimize run-time overhead.

The RRM system can significantly impact on the system performance. In order to fully exploit the potentialities of multi-core architectures, selecting the right RRM technique to apply for the target platform is of outstanding importance. Unfortunately, there is not an unique RRM system that best fits in all possible use case requirements in general.

In this paper we present the *EMME Evaluation Framework* (in short, **EMMEframework**) [Mar11]. The goal of this framework aims at providing to embedded system designers a tool for system level performance analysis of homogeneous multi-core systems. Given a run-time scenario and a RRM policy, the framework provides a fast empirical analysis of the system behavior. This enables the designer to explore the effects of different RRM policies and to validate the desired system properties at the earliest design phases. The approach is targeted to *soft real-time* applications where the RRM is responsible for *maximizing the system performance* while fitting in a *power budget constraint*.

The paper is organized as follows. Section 2 presents related work in the field of RRM for multi-core embedded systems. Section 3 describes in details the **EMMEframework**. Then, Section 4 reports some experimental results obtained from the analysis of different RRM techniques for an example multiprogrammed multi-core embedded scenario. Finally, in Section 5 some concluding remarks are outlined.

2 Background

The RRM scenarios considered in the present work are *multiprogrammed multi-core embedded systems* where different applications, each one composed of different parallel

threads, are competing to access the system resources. In these scenarios the relationships between run-time tunable parameters and overall platform behavior might be very complex and sometime counter-intuitive [SRS+95].

In general, analytical models of system performance are not available when RRM takes into account parameters such as resource distribution. For instance, performance values of an application when parallelized on a certain number of processors are strictly application-dependent. In fact, for some applications performance values might scale linearly with the number of computing resources allocated to their execution while other applications cannot exploit a huge number of processors.

Authors in [AW06] noted that for multiprocessor workloads, the *Instruction Per Cycle* (IPC) is a poor performance index and identified cases where IPC increases do not reflect any performance gain. For multiprogrammed multi-core scenario, *system-level performance metrics* as *throughput* measured in terms of *job/s* or other *user-oriented performance metrics* as the applications' *response time* are more suitable [EE08].

The RRM approach we consider assumes that the target application set is known at design time and application performance for different resource allocations can be measured by exploiting off-the-shelf simulation models [YCAM+11, YCNCC11, SGB+09]. Given the design-time analysis of application performance, the RRM should decide how to allocate the computing resources available on the embedded platform.

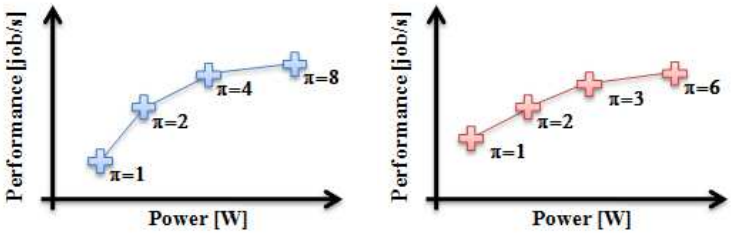
In order to maximize system performance and improving battery lifetime in portable devices, it is of outstanding importance to select the right RRM policy to deploy on the system being designed. For these reason we propose an open source framework for quickly analyzing the effects of different RRM systems targeting the optimization of an user-oriented performance index such as the applications' *response time*.

To avoid lengthy simulations, the performance evaluation considered in **EMMEframework** is carried out at a very high level. First, the applications are independently characterized in terms of their execution time. Then, to evaluate the performance of a specific RRM policy for a given run-time scenario, the applications' execution times are retrieved from the previous characterization. Thus, different RRM policies can be evaluated without the need to re-run a detailed simulation of each application. An example of power management system that considers functional and temporal independence between different applications can be found in [NMG11]. Few work has been done on tools for modeling and analyzing power management techniques for multi-core [BHB+08, TDM11]. However state of the art is manly dedicated to the modeling of DVFS and other orthogonal power management approaches rather than on the resource allocation problem.

3 The EMME Evaluation Framework

In the **EMMEframework**, the user selected RRM system takes decisions on the basis of the following information:

- The **application characterization** performed at design-time. The application characterization reports for each application the set of *operating configurations*, i.e. performance and power indices obtained when the application is executed using a cer-



(a) app0's operating configurations (b) app1's operating configurations

Figure 1: Design-time application characterization reporting the operating configurations of two applications in terms of: power consumption (x axis), performance (y axis) and resource requirement (the π value).

tain number of cores (Figure 1). In our approach we assume that this characterization can be obtained simulating each application independently.

- The **user activity**. We consider that the user activity (or the interaction with the external environment) issues the processing of some data by the active applications. From now on we will use the term *job* to refer to an unitary data chunk to be processed, e.g. a single frame in a video application. In general the user activity is unknown at design-time but we assume that it can be profiled during run-time. For example the execution of a video application might require the processing of a certain number of frames per second (e.g. 25 frames per second). This computing request can be profiled at run-time and the RRM can use the profiling data during its decision making.

- The **power budget**, which is assumed to be set by the OS. This can be set according to, among other things, the actual system state (e.g., the system is plugged into a power supply or not).

3.1 The run-time methodology

The **EMMEframework** targets homogeneous multi-core computing platform. We consider that at run-time different applications are executed and compete to access the available processing elements. The RRM introduces a run-time processor assignment policy to maximize the user-perceived performance (in terms of applications' response time) while fitting in the **power budget**. We define as application response time the average time an application job spend in the system from its arrival to its completion. The processor assignment depends on the **user activity** that dynamically issues the processing of different applications' jobs (e.g. the elaboration of different frames for a video application) and on the design-time **application characterization**.

We consider code versioning [YCNM+06] as the main enabling technology in order to change the task-level parallelization of an application. However, other mechanisms for

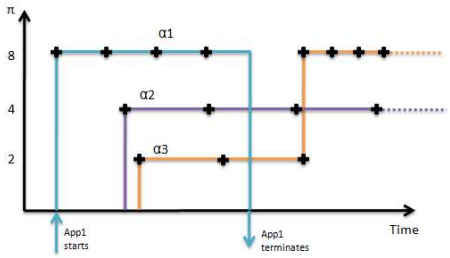


Figure 2: Run-time behavior. The parallelization π changes only between the execution of two jobs.

manipulating the program representation to exploit available processors can be considered with their additional overhead (e.g. stream program fusion [GTA06, GTK+02]).

We consider that application parallelization cannot be changed during the execution of a single job but only between the execution of two different jobs. We also consider that some jobs might be temporarily stored in memory while waiting to be processed. This might happen for bursty applications where many jobs are issued during a short time interval. Figure 2 shows an example behavior of the task-level parallelization chosen by a RRM for a scenario with 3 applications running concurrently (job starting times are indicated with '+'). When α_1 starts, 8 cores are allocated to it. Then applications α_2 and α_3 enter the system. The RRM allocates to these applications 4 and 2 cores respectively. When α_1 exits the system, the parallelization of α_3 is increased to 8 cores. The run-time decisions are taken following the RRM policy and influence the overall system performance.

3.2 Evaluation of system performance

Following an application specific design approach we consider that the target application set is known at design time. However, we consider that the mix of applications to be concurrently executed is not known a priori and, at any moment, the user is free to launch the execution of any application selected between the ones known at design time. We consider that at run-time the user activity issues the elaboration of a sequence of applications' jobs. We assume that at run-time the user activity can be profiled in terms of number of jobs issued for each application in a time unit (the applications' *arrival rates*).

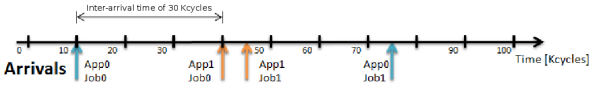
Using a traditional approach to evaluate the run-time performance of the target multiprogrammed multi-core system one would have to simulate the concurrent execution of the whole application mix with a detailed architectural model. To give a practical idea of the computational cost of simulating a realistic scenario, the simulation of a single MPEG2 application processing 2 frames using the SESC simulator [RFT+05] might take few minutes. Extending this simulation to a significant number of frames and considering the concurrent execution of other applications might lead to a simulation time of several hours or even few days.

To reduce this computational cost and to produce results for complex scenarios in few seconds, the **EMMEframework** takes some assumptions on the underlying computing

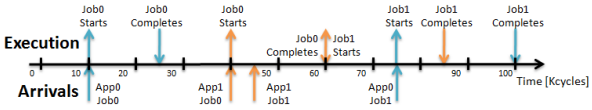
platform and on the predictability of the jobs' execution times. We assume that:

- At run-time, the set of computing resources is partitioned into disjoint subsets and each subset is allocated to a different application.
- The execution time of a specific job depends only on the input dataset and on the resources allocated to its elaboration. Thus, there are no interferences between different applications during the concurrent execution. It is worth to notice that this assumption might require specific communication infrastructure in order to keep a predictable communication time. In future works we envision to extend the framework to consider a less conservative assumption on the job execution time.
- For a given application, before starting the execution of a new job, all previous jobs should be completed. If a new job arrives while another job is under elaboration, the new job is temporary stored in the on-chip memory and it waits to be scheduled.
- The switching time required to change the operating configuration for a given application is negligible in reference to the execution time of a single job.

Under these assumptions, the evaluation of system performance can be obtained at a very high level. Given an **input trace**, defined in terms of jobs' arrival times, the **EMME-framework** simulates the computing system by scheduling the input jobs considering the resource distribution defined by the user selected RRM policy. The framework output consists of an **execution trace** that completes job arrivals information with data about job waiting and execution time (we consider the job response time as the overall time a job spend in the system, i.e. waiting plus execution times).



(a) Example of **input trace**.



(b) Example of output **execution trace**.

Figure 3: **Input trace** and output **execution trace** for an example two application scenario. Events related to different applications are highlighted with different colors.

An example is shown in Figure 3. As the first job of app0 arrives (Figure 3(a)), its execution starts and the elaboration is completed at $25Kcycles$ (Figure 3(b)). Then a job of app1 arrives and its execution starts (at $40Kcycles$). A second job of application 1 arrives at $45Kcycles$ but its execution will start only at $60Kcycles$. The second job of app1 is subject to a waiting time since we assumed that, before starting the execution of a new job, the application must terminate the elaboration of all previous jobs. A second job of app0 arrives while app1 is elaborating. In this example we consider that the RRM did not allocate all computing resources to the execution of app1 and thus the system can schedule the concurrent execution of the new job of app0.

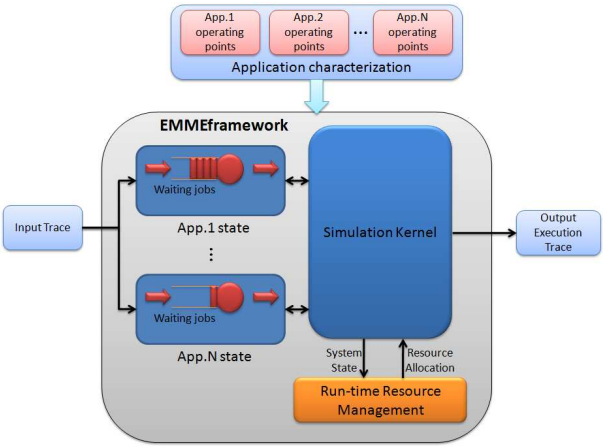


Figure 4: The **EMMEframework** structure.

To generate the output **execution trace** from the **input trace**, the **EMMEframework** does not simulate on the detailed architectural model the job execution. In fact, it uses the assumption on the job execution time predictability to compute, given the starting time of the job and the resources allocated to its execution, the completion time. Optionally, in the **input trace**, together with the job arrival times, one can specify information on the variability of each job execution time in reference to the average one. This variability is considered due to the specific input dataset.

In the **EMMEframework** (Figure 4), the application states are kept updated during the system simulation. As new jobs are arriving these are dispatched to the applications' waiting queue and the **simulation kernel** cares of iteratively identifying the event to be processed next (e.g. either a job arrival or completion or an RRM invocation). During the **EMMEframework** execution, the **simulation kernel** keeps track of the system state in terms of number of completed jobs, number of jobs currently in the system (either waiting or executing), and power consumption resulting from the current resource allocation.

When executing the **EMMEframework**, the user can select a **RRM policy** among the ones available (Section 3.3). During the simulation, the selected RRM will interact with the simulation kernel to access the current system state and to assign the resource allocation (Figure 4).

Together with the output jobs' starting and completion times, detailed information about resource distribution and power consumption are reported in the output **execution trace**.

3.3 The run-time resource management policies

A RRM policy defines how the resources should be distributed between the active applications. Three RRM policies are released with the **EMMEframework**, named *Pull High Push Low* (**`PHPL`**), *maximization of the current Throughput* (**`maxT`**) and *Application-*

specific Run-Time managEment (***ARTE***). The RRM policies can be linked to the **EMME-framework** at run-time. This linking mode enables to quickly evaluate different RRM strategies without the need of recompiling the framework for each strategy.

3.3.1 Pull High Push Low

The ***PHPL*** policy is derived from the approach presented in [IBC+06]. This policy periodically verifies the power consumption of the different applications and modifies the resource allocation to fit in the power budget (first) and to balance the power consumed by the different applications (second).

Every time the ***PHPL*** is invoked, the RRM verifies if the power budget has been exceeded. If this is the case, ***PHPL*** reduces the parallelism of the application consuming the most power. Otherwise, the power budget not yet in use is allocated to the application consuming the least power by increasing its parallelization.

3.3.2 Maximization of the current Throughput

The ***maxT*** policy is presented in [MPSZ09]. ***maxT*** is invoked every time an application switches between the idle and the ready states. ***maxT*** exhaustively explores the possible allocations of computing resources to the set of applications currently running. The resource allocation providing the maximum throughput sum (measured in Job/s) is selected.

3.3.3 Application-specific Run-Time managEment

The ***ARTE*** policy is presented in [MPSZ11]. ***ARTE*** takes some additional assumptions on the computing system to model it using queuing theory. In particular, it is assumed that job inter-arrival times are exponentially distributed with a certain mean (derived from the profiled data). Moreover, it is assumed that the job execution time for different datasets is approximately constant. Given these assumptions, each application is modeled as a M/D/1 (Markov arrival, deterministic execution, single server) queuing system and the job response time is given analytically [Tri02].

ARTE is invoked periodically. Within a RRM period resources are reserved to the applications. Resource reservation allows to manage constraints on the individual application throughput on a window-based, periodic basis. In ***ARTE***, the run-time exploration targets the minimization of the expected average job response time which metric is derived analytically at run-time on the basis of queuing theory.

4 A practical example

As example case study we analyze an embedded Chip Multi Processor (CMP) composed of an **host processor** and a **computing fabric** consisting of 16 computing elements (Figure 5). The OS, including the RRM system, runs on the host processor. The computing fabric

is used to process the input workload. The **host processor** takes decision about resource assignment and, in this sense, it acts as **fabric controller**.

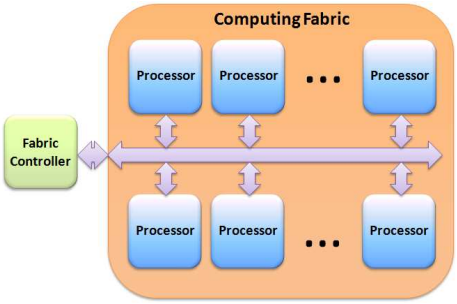
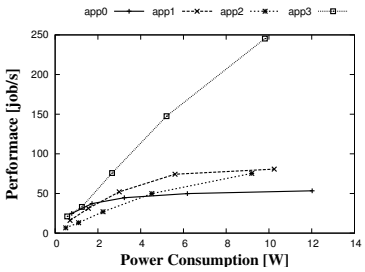


Figure 5: Overview of the target Chip Multi Processor.

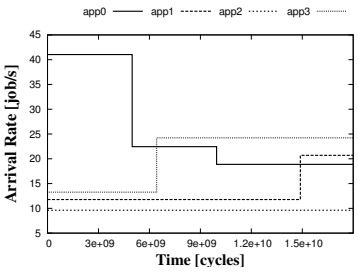
We considered MIPS-like processors with a shared memory architecture. The inter-processor communication is based on a high-bandwidth split transaction bus supporting a write-invalidate snoop-based MESI coherence protocol acting directly between L2 caches.

In the considered case study the application mix populating the computing system is taken from the SPLASH-2 parallel benchmark suite [WOT+95]. We focus our attention on the following application kernels: Complex 1D FFT (*fft*) Integer Radix Sort (*radix*), Ocean Simulation (*ocean*) and Blocked LU Decomposition (*lu*). In the following text we will call these applications: {app0 ... app3}.

To generate the input **application characterization** (Figure 6(a)), we model the computing fabric using the SESC simulation tool [RFT+05], a fast MIPS instruction set simulator for CMPs. To generate the **input trace** we considered exponentially distributed jobs' inter-arrival times (Poisson process). We also considered that the unpredictable user behavior generates variations in the average job arrival rates (Figure 6(b)) and that some variability in the job execution time is introduced due to the input datasets. In particular, we considered a job execution time uniformly distributed in the range of 10% around the average.



(a) Application characterization.



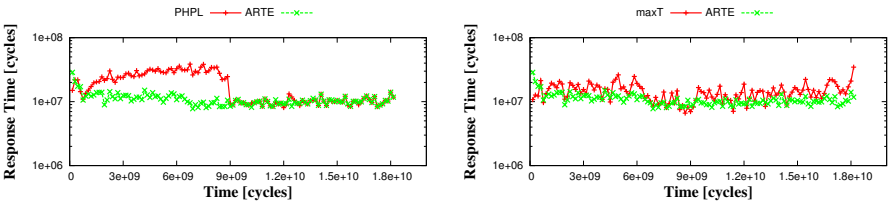
(b) Average job arrival rate times.

Figure 6: Application characterization and average job arrival rates for the specific case study.

The operating frequency of the computing elements is $300MHz$ and the input trace is of 60 seconds length (i.e. $18Gcycles$). For our analysis we consider a power budget of $8.6W$,

that is 70% of the power consumed when all applications are concurrently executing on the 16 cores composing the computing fabric.

Evaluating the system performance. To evaluate the efficiency of the different RRM policies we simulate the system using the **EMMEframework**. A system simulation runs in less than 1 second on our host machine (Intel CoreTM Duo T6570, 2.1Ghtz). The output execution traces are then post-processed using the routines released with the same framework. These routines return (among other data), a graphical comparison of the geometric average of applications' response time (Figure 7).



(a) Comparison between ***ARTE*** and ***PHPL***

(b) Comparison between ***ARTE*** and ***maxT***

Figure 7: Response time comparisons between the considered RRM policies.

For the specific case study, ***ARTE*** provides better performance. To select a RRM policy for final deployment on the target system one might be interested in verifying that job response time for a certain application never exceeds a certain value or verifying that the storage requirements do not exceed the on-chip memory.

The **EMMEframework** produced detailed data on the system behavior that can be inspected. For example the post-processing routines return also the plot of the number of jobs resident in the system that is directly correlated with the on chip memory requirements (this happens because waiting jobs must be temporary stored in memory).

When using ***PHPL*** for our case study the number of jobs resident in the system reaches 160 instances (Figure 8). This might generate implementation problems in our design. The described phenomena happens since, during the initial $5Gcycles$, the arrival rate of app0 is very high. To adequately serve this computing request the most of computing resources should be allocated to app0. ***PHPL*** equally distributes the computing resources between the active applications, thus the resources allocated to app0 are not enough to serve such a high arrival rate. Consequently during the initial period some arriving jobs of app0 should be buffered increasing the storage requirements.

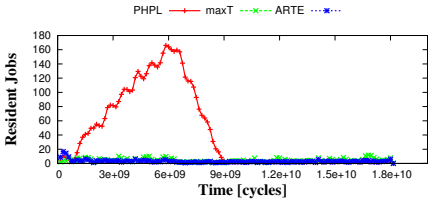


Figure 8: Number of jobs resident in the system when considering different RRM policies

5 Summary

In this paper we presented the **EMMEframework** that enables to quickly explore the effects of different RRM policies for a target multiprogrammed multi-core scenario.

The framework takes as input the **application characterization** obtained from the simulation of the target applications on a detailed architectural model. However, the concurrent simulation of the different applications on the detailed architectural model is avoided. This reduces significantly the computational costs related with the analysis of different RRM policies for the target use case scenario.

Together with the **application characterization** the framework takes an **input trace** defining jobs' arrival times. The **execution trace** output of the **EMMEframework** completes the **input trace** with data related to jobs' starting and completion time. Post-processing routines released with the **EMMEframework** enables to compare the different RRM policies at a very high level of abstraction.

References

[AW06] A.R. Alameldeen and D.A. Wood. IPC Considered Harmful for Multiprocessor Workloads. *Micro, IEEE*, 26(4):8 –17, jul. 2006.

[BBM00] L. Benini, R. Bogliolo, and G. De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on VLSI Systems*, 8:299–316, 2000.

[BHB+08] R. Bergamaschi, Guoling Han, A. Buyuktosunoglu, H. Patel, I. Nair, G. Dittmann, G. Janssen, N. Dhanwada, Zhigang Hu, P. Bose, and J. Darringer. Exploring power management in multi-core systems. In *Proc. Asia and South Pacific Design Automation Conference ASPDAC 2008*, pages 708–713, 2008.

[EE08] S. Eyerman and L. Eeckhout. System-Level Performance Metrics for Multiprogram Workloads. *Micro, IEEE*, 28(3):42 –53, may. 2008.

[GTA06] Michael I. Gordon, William Thies, and Saman Amarasinghe. Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. In *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, ASPLOS-XII, pages 151–162, New York, NY, USA, 2006. ACM.

[GTK+02] Michael I. Gordon, William Thies, Michal Karczmarek, Jasper Lin, Ali S. Meli, Andrew A. Lamb, Chris Leger, Jeremy Wong, Henry Hoffmann, David Maze, and Saman Amarasinghe. A stream compiler for communication-exposed architectures. In *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, ASPLOS-X, pages 291–303, New York, NY, USA, 2002. ACM.

[IBC+06] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 347–358, 2006.

[Mar11] Giovanni Mariani. EMME: Efficient run-time resource Management for Multi-core Embedded platforms, 2011. http://www.alari.ch/emme.

[MPSZ09] G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria. A design space exploration methodology supporting run-time resource management for multi-processor Systems-on-chip. In *Proc. IEEE 7th Symp. Application Specific Processors SASP '09*, pages 21–28, 2009.

[MPSZ11] G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria. ARTE: An Application-specific Run-Time management framework for multi-core systems. In *Application Specific Processors (SASP), 2011 IEEE 9th Symposium on*, pages 86 –93, june 2011.

[NMG11] A. Nelson, A. Molnos, and K. Goossens. Composable power management with energy and power budgets per application. In *Embedded Computer Systems (SAMOS), 2011 International Conference on*, pages 396 –403, july 2011.

[RFT+05] Jose Renau, Basilio Fraguela, James Tuck, Wei Liu, Milos Prvulovic, Luis Ceze, Smruti Sarangi, Paul Sack, Karin Strauss, and Pablo Montesinos. SESC simulator, January 2005. http://sesc.sourceforge.net.

[SGB+09] H. Shojaei, A. Ghamarian, T. Basten, M. Geilen, S. Stuijk, and R. Hoes. A Parameterized Compositional Multi-dimensional Multiple-choice Knapsack Heuristic for CMP Run-time Management. In *DAC '09: Proceedings of the 46th conference on Design automation*, New York, NY, USA, 2009. ACM.

[SRS+95] E. Smirni, E. Rosti, G. Serazzi, L. W. Dowdy, and K. C. Sevcik. Performance Gains From Leaving Idle Processors in Multiprocessor Systems. In *in International Conference on Parallel Processing*, pages 203–210, 1995.

[TDM11] Ibrahim Takouna, Wesam Dawoud, and Christoph Meinel. Accurate Mutlicore Processor Power Models for Power-Aware Resource Management. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 419 –426, dec. 2011.

[Tri02] Kishor S. Trivedi. *Probability and statistics with reliability, queuing and computer science applications*. John Wiley and Sons Ltd., Chichester, UK, 2002.

[WOT+95] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: characterization and methodological considerations. In *Proc. 22nd Annual Int Computer Architecture Symp*, pages 24–36, 1995.

[YCAM+11] C. Ykman-Couvreur, P. Avasare, G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria. Linking run-time resource management of embedded multi-core platforms with automated design-time exploration. *IET Computers & Digital Techniques*, 5(2):123–135, 2011.

[YCNCC06] Ch. Ykman-Couvreur, V. Nollet, Fr. Catthoor, and H. Corporaal. Fast Multi-Dimension Multi-Choice Knapsack Heuristic for MP-SoC Run-Time Management. In *Proc. International Symposium on System-on-Chip*, pages 1–4, 2006.

[YCNCC11] Ch. Ykman-Couvreur, V. Nollet, F. Catthoor, and H. Corporaal. Fast multidimension multichoice knapsack heuristic for MP-SoC runtime management. *ACM Trans. Embed. Comput. Syst.*, 10:35:1–35:16, May 2011.

[YCNM+06] Ch. Ykman-Couvreur, V. Nollet, Th. Marescaux, E. Brockmeyer, Fr. Catthoor, and H. Corporaal. Pareto-Based Application Specification for MP-SoC Customized Run-Time Management. In *Proc. International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation IC-SAMOS 2006*, pages 78–84, 2006.