A Run-time Self-Adaptive Resource Allocation Framework for MPSoC Systems

Wei Quan[†,‡]

[†]Informatics Institute
University of Amsterdam
The Netherlands
{w.quan,a.d.pimentel}@uva.nl

Andy D. Pimentel[†]

[‡]School of Computer Science
National University of Defense Technology
Hunan, China
quanwei02@gmail.com

***Abstract*—Self-adaptivity is becoming a key feature of modern embedded systems to meet performance and power constraints in increasingly common situations where embedded application workloads show highly dynamic behavior. This paper presents a scalable framework for adaptive MultiProcessor System-on-Chip (MPSoC) systems that allows for adaptivity throttling.**

I. Introduction

Due to the ever-increasing performance demands of modern embedded applications, MultiProcessor System-on-Chip (MPSoC) systems have gained considerable popularity in the field of embedded systems. These MPSoCs usually are heterogeneous systems, containing programmable processor cores for flexible application support as well as dedicated processing elements for achieving performance and/or power goals. Today's MPSoC systems often require to supporting an increasing number of applications and standards, where multiple applications can run simultaneously and therefore contend for system resources. In addition, each separate application may also consist of different execution modes with different requirements. For example, in Software Defined Radio appliances a radio may change its behavior according to resource availability, such as the Long Term Evolution (LTE) standard that uses adaptive modulation and coding to dynamically adjust modulation schemes and transport block sizes based on channel conditions. As a consequence, the behavior of application workloads executing on the embedded system can change dramatically over time. Such dynamic application behavior can be captured using the concept of *workload scenarios*, specifying the different applications (at the level of communicating tasks) that are concurrently executing and the mode of each application over time. This is illustrated in Figure 1. Clearly, the number of workload scenarios scales exponentially with the number of applications and modes within each application.

For each workload scenario, the mapping of application tasks onto the underlying MPSoC resources plays a crucial role in achieving high performance and low power consumption, especially in the case of heterogeneous MPSoC systems. The performance of each workload scenario may vary greatly among different task mappings. To allow MPSoC systems to cope with application dynamism, state-of-the-art solutions therefore try to improve system efficiency by providing a light-weight resource scheduler that is capable of dynamically re-mapping application tasks based on the detected workload scenario [2], [12], [9], [6], [10]. Generally, this kind of methods can be divided into two stages. First, a design-time preparation stage prepares one or multiple pre-optimized task mappings for each possible scenario that might appear on the target system. These task

978-1-4799-9877-7/15/$31.00

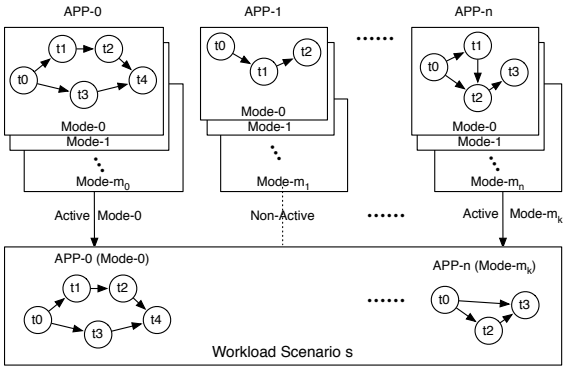

Figure 1: Definition of a workload scenario.

mappings could optimize the system for different objectives, such as performance and energy consumption. The second stage then involves a run-time resource scheduler that dynamically selects a task mapping from the pre-optimized task mappings, given the workload scenario at hand.

However, these state-of-the-art run-time mapping solutions typically still lack scalability and adaptivity throttling. Regarding scalability, future MPSoC systems may need to support a vast number of workload scenarios. This could result in a design-time mapping optimization stage that is computationally intractable and requires an unacceptable amount of memory to store the pre-optimized mappings on the system. The lack of adaptivity throttling in current run-time mapping solutions relates to the problem that workload scenarios in MPSoC systems can be fine-grained. That is, when the duration of a certain workload scenario is not long enough (i.e., different workload scenarios appear after each other with a relatively high frequency), the overhead that is involved in re-mapping application tasks may easily eliminate the benefits of adapting the system. In these cases, the system should actually decide not to adapt.

II. The SARA Resource Allocation Framework

To address the above scalability and adaptivity throttling issues in self-adaptive MPSoC systems, a Scenario-based run-time Adaptive Resource Allocation (SARA) framework is proposed. This framework consists of three components: a Run-time System Monitor (RSM), a Run-time Mapping Generator (RMG) and a self-Adaptive Resource Scheduler (ARS). The RSM is in charge of detecting the active workload scenario on the target MPSoC system and dynamically collects system statistics. The RMG and ARS are responsible for the system adaptation and address the scalability and adaptivity throttling issues as explained above. Figure 2 shows the high-level system

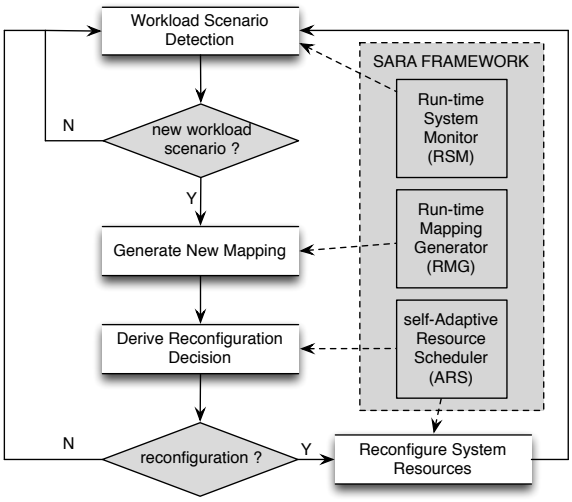

Figure 2: The high-level system workflow of the SARA framework.

workflow of the SARA framework. When the RSM detects a new workload scenario, the RMG will generate a new mapping for the detected (active) scenario. Hereafter, the ARS makes an adaptation decision by predicting the benefit of changing the current mapping into the newly proposed one (by the RMG). According to this decision, the ARS will then either reconfigure the system based on the new mapping or continue the system's execution under the current mapping.

The mechanism of detecting workload scenarios is beyond the scope of this paper. Here, we focus on how to improve the efficiency of MPSoC systems by using SARA's run-time mapping framework in which the RMG solves the scalability problem by using a hybrid mapping optimization approach and the ARS deploys a history-based scenario duration prediction mechanism to perform adaptivity throttling, i.e. deciding whether or not to adjust the task mapping.

III. Scalable Run-time Task Mapping

The problem of optimally mapping a set of tasks onto a set of given heterogeneous processors for maximal performance (like system throughput) has been known, in general, to be NP-complete. This problem is exacerbated when mapping multiple applications (i.e., bigger task sets) onto the target platform. Traditionally, the mapping of applications onto the underlying architectural components of MPSoC systems has always been done in a static fashion at design time. These methods typically use computationally intensive search methods to find the optimal mapping or near optimal mapping for all applications that may run on the system. Evidently, the drawback of such static mapping techniques is that they cannot cope with dynamic application behaviour in which different combinations of applications are concurrently executing over time and contending for system resources. As mentioned before, run-time task mapping techniques have been proposed to overcome this drawback of static mapping techniques, but these methods typically still suffer from scalability issues when the number of workload scenarios becomes very large as they need to find and store one or more optimal task mappings per scenario at design time (and to be used at run time). One solution to address this problem is by reducing the number of workload scenarios by means of clustering [1], [6]. However, these methods still suffer from an additional problem of searching for optimal mappings of (clustered) workload scenarios at design time:

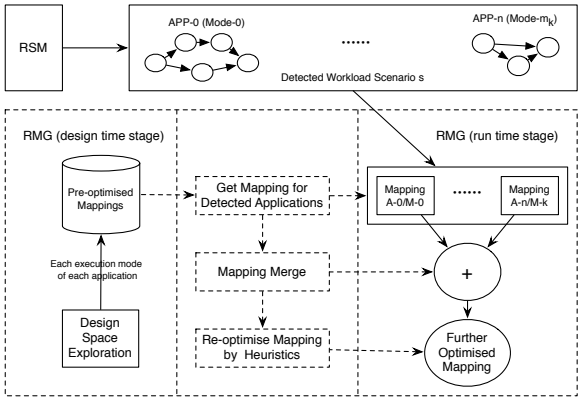

Figure 3: Mapping generation in the RMG component.

it should already be known at design time which applications can execute on the target platform. This implies that extending the system with a new application would require to redo the entire design-time mapping preparation for all (clustered) workload scenarios.

In the SARA framework, we address the above problems by using a hybrid task mapping technique which prepares *partial task mappings* for workload scenarios at design time and completes the mappings for the entire scenario at run time using the RMG component. Figure 3 gives an overview of how the RMG generates a new mapping for the workload scenario detected by the RSM. At design time, a performance-optimized task mapping (and, if needed, also a power-optimized mapping) for each execution mode of each *application in isolation* is determined by using state-of-the-art scenario-aware Design Space Exploration (DSE) techniques [11], [7]. This significantly reduces the time and memory requirements needed for, respectively, finding and storing the pre-optimized task mappings at design time. For example, when considering n target applications with each m execution modes, the number of mappings that need to be optimized and stored is $m*n$ in our case. This number is greatly reduced compared to the $(m+1)^n-1$ mappings that need to be optimized and stored in the case of performing mapping preparation for complete workload scenarios. Moreover, if a new application needs to be supported on the target MPSoC system, this would only require providing the pre-optimized mappings of this new application to the RMG without redoing the entire process of design-time mapping preparation for all possible (new) workload scenarios.

At run time, after the RSM has detected a new workload scenario, the RMG will first *merge* the pre-optimized mappings of each separate, active application in the detected workload scenario to form a first-order mapping for the entire scenario. Subsequently, the RMG will then further optimize this first-order mapping by using run-time mapping optimization heuristics, based on e.g. a load balance algorithm or a dynamic mapping optimization algorithm such as proposed in [6], [5]. This run-time mapping optimization focuses mainly on resolving resource contention problems. This is because the communication between tasks inside each application has already been optimized at design time, which saves significant run-time effort to re-optimize the task communications. As a consequence, in RMG's mapping optimization process only slight adjustments to the first-order mapping will typically lead to a good mapping for the target workload scenario.

In SARA, the scalability problem is therefore addressed by means of a divide-and-conquer approach. At design time, we divide the mapping problem of a workload scenario that may contain several

applications into several smaller application-level mapping problems. This greatly reduces the complexity of the whole scenario mapping problem and consequently increases the possibility of finding the optimal results for each separate application-level mapping problem. Consequently, the SARA framework is able to handle large numbers of workload scenarios and also effectively supports the addition of new applications to the target MPSoC system.

IV. Adaptivity Throttling

In current state-of-the-art run-time task mapping approaches for MPSoCs, the system will typically be reconfigured (i.e., adapting the task mapping) when a new workload scenario appears on the system, irrespective of the trade-off between reconfiguration costs and benefits. However, such system reconfigurations usually involve task migrations of which the performance overhead cannot be ignored. This is especially true for heterogeneous MPSoC systems and for those cases in which the duration of workload scenarios is relatively short. For example, let us assume that a newly detected workload scenario needs to finish a certain amount of work that is equally divided into u units. Under the current/old task mapping, the target MPSoC system requires t units of time to finish a single unit of this work. However, if this new scenario would be executing under the new mapping derived by the run-time task mapping generator (and optimized for the workload scenario at hand), the execution time of a single unit work of this scenario would be reduced to t'. Consequently, for the target u units of work, the execution time saved by reallocating the system resources under the new mapping for executing this scenario is $\Delta t = (t - t') * u$. However, this time reduction comes at the cost of task migration between processors and the computational overhead of the SARA framework itself (i.e., the time needed for determining the new mapping and deriving the reconfiguration decision). Here, we assume that these costs are referred to as c. Under these assumptions, it is evident that only if Δt is larger than c – implying that the system actually benefits from the reconfiguration – then the system should re-map the application tasks to improve system efficiency, and otherwise not. This can be seen as *throttling* the adaptivity. Although a similar trade-off for costs and benefits of reconfiguration can be made in terms of optimizing for power consumption, we will focus on performance in the remainder of the discussion. To make the adaptivity support in MPSoC systems more effective, the resource scheduler should therefore be capable of explicitly making these reconfiguration decisions (i.e., provide support for adaptivity throttling) whenever workload scenarios change. In the SARA framework, this is taken care of by the ARS component.

To determine a reconfiguration decision, three parameters are required: the performance improvement of re-mapping tasks ($t - t'$), the scenario execution duration (u), and the reconfiguration cost (c). These three parameters are, however, unknown before the system reconfiguration. As a consequence, prediction models should be used to predict each of these values. Figure 4 illustrates how the ARS component conditionally reconfigures the target system based on the outcome of the prediction models (i.e, $\Delta t > c$). The information about the target applications and hardware architecture used in the performance prediction model as well as the reconfiguration cost prediction model should have be prepared at design time, and depends on the type of models used for these predictions (as discussed below). Also notice that the reconfiguration cost consists of two parts: 1) the overhead of SARA itself which includes the determination of a new mapping and making a reconfiguration decision, and 2) the task migration cost during system reconfiguration. The overhead of

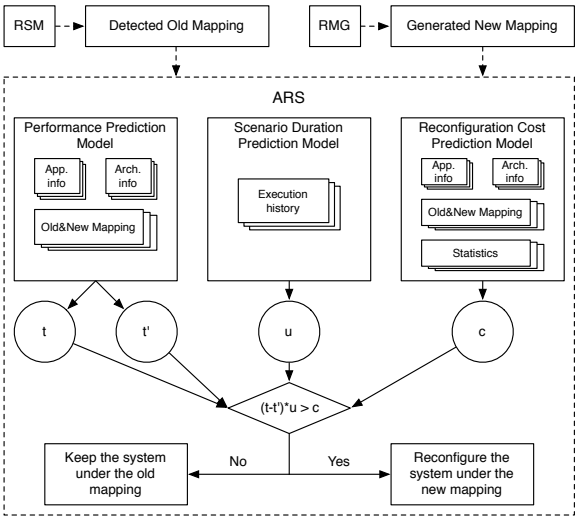

Figure 4: Overview of adaptive system reconfiguration in ARS

SARA can be determined by means of measurements. However, the overhead of the task migration should still be predicted.

The prediction models in ARS cannot be computationally intensive as they have to efficiently make a reconfiguration decision at run time. For the performance and reconfiguration cost prediction, relatively simple regression models or analytical models such as the performance model from [4] can therefore be applied. However, the prediction of scenario execution duration needs a different approach as it involves a dynamic parameter that could be heavily influenced by user behavior. A commonly used predictor for such kind of parameters, which has also been used in our ARS component, is a history-based predictor such as a last value predictor, table-based predictor or the Statistical Metric Model (SMM) [8]. They can predict the future value of a parameter – in our case the duration of a newly detected workload scenario – based on its history information.

After deriving a reconfiguration decision based on the three predictive models for performance, reconfiguration cost and scenario duration, the ARS will either reconfigure the system according to the new mapping or keep the old mapping. By applying such adaptivity throttling, our adaptive MPSoC system is able to cope with fine-grained workload scenarios for which it is not beneficial to reconfigure the system.

V. Evaluation and Conclusion

To illustrate the effectivity of our SARA framework, we present a case study using the open-source Sesame system-level MPSoC simulator [3]. Figure 5 provides an overview of the SARA implementation within the Sesame simulator. The target hardware architecture is a bus-based heterogeneous MPSoC platform containing five different processing elements, a shared memory and a controlling processor that runs the SARA framework. In this SARA implementation, the RMG adopts the Energy-aware Iterative multi-application Mapping (EIM) algorithm from [5]. However, in this evaluation we only focus on performance, measured as system throughput, as the run-time mapping optimization objective. The performance and reconfiguration cost predictors in the ARS use linear regression models to reduce the run-time computation complexity. With regard to the scenario duration predictor, a slightly modified SMM [8] predictor, referred as Accumulated SMM, has been implemented in the ARS.

For the target applications, as the actual functionality of the

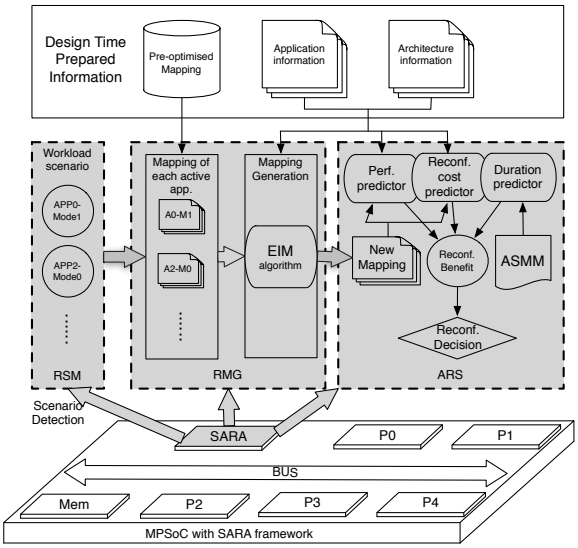

Figure 5: An implementation of the SARA framework

applications is not important in our experiment, we have used five synthetic streaming applications to simplify the simulation process. Each application contains only a single execution mode, which makes the total number of workload scenarios 31 ($2^5 - 1$). Using these workload scenarios, three different workload scenario sequences – specifying the different workload scenarios occurring over time – have been generated in which the percentage of workload scenarios with a short duration (i.e., fine-grained scenarios) is varied. Figure 6 shows the results of our experiment, where the x-axis shows the percentage of coarse-grained scenarios in the scenario sequence. In this experiment, we compared our SARA framework with three alternative approaches. Firstly, an approach in which all applications are statically mapped (i.e., no run-time mapping takes place) using a mapping which has shown to be optimal *on average* for all possible workload scenarios (*STATIC* in Figure 6). Secondly, an approach that always reconfigures the system whenever a new workload scenario has been detected according to a corresponding pre-optimized mapping derived at design time (*MIGRATE-OPT* in Figure 6). Please note that this approach, which is similar to how many state-of-the-art run-time mapping techniques operate, stores 31 mappings (one mapping for each workload scenario) in total on the system. Finally, we also compare to SARA without adaptivity throttling (*SARA-NOTH* in Figure 6). From Figure 6, we can clearly see that our SARA framework with throttling shows the best performance of all run-time mapping approaches. Because of its ability to throttle adaptivity, it even performs relatively well in the case when 70% of the workload scenarios have a short duration and for which reconfiguring the system is not beneficial. In this particular case, the static mapping approach works best as this approach does not suffer from any run-time overheads. Besides the performance improvement of SARA, it also only needs to store 5 pre-optimized mappings (one mapping for each application) on the target system instead of the 31 mappings stored by MIGRATE-OPT. These savings in memory footprint will become even more apparent when the number workload scenario increases.

In conclusion, adaptivity becomes an increasingly important feature for improving the efficiency of future MPSoC systems executing highly dynamic application workloads. To design such adaptive systems, the problems of scalability and adaptivity throttling should

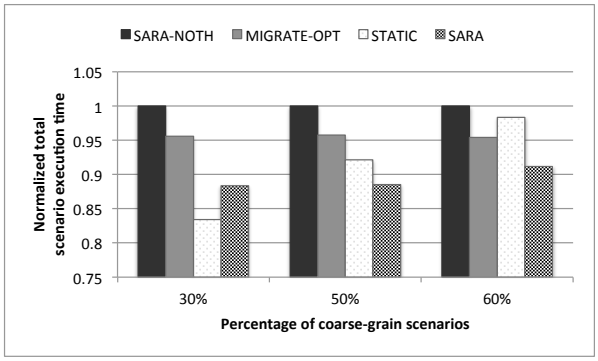

Figure 6: Performance comparison of different mapping approaches for scenario sequences with varying percentages of coarse-grained scenarios (coarse-grained scenarios/all scenarios).

be carefully considered. In this work, we have proposed the SARA framework for adaptive MPSoCs that addresses these issues. Experimental results demonstrate that this framework provides a promising approach for improving the performance of MPSoC systems under a variety of dynamic application workloads while trying to reduce the required design-time preparation effort in terms of both space and time.

REFERENCES

[1] S. V. Gheorghita, M. Palkovic, J. Hamers, A. Vandecappelle, S. Mamagkakis, T. Basten, L. Eeckhout, H. Corporaal, F. Catthoor, F. Vandeputte, and K. D. Bosschere. System-scenario-based design of dynamic embedded systems. *ACM Trans. Design Autom. Electr. Syst.*, 14(1), 2009.

[2] G. Mariani, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, G. Palermo, C. Silvano, and V. Zaccaria. An industrial design space exploration framework for supporting run-time resource management on multi-core systems. In *Proc. of DATE'10*, pages 196 –201, march 2010.

[3] A. D. Pimentel, C. Erbas, and S. Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Trans. Computers*, 55(2):99–112, 2006.

[4] R. Piscitelli and A. D. Pimentel. Design space pruning through hybrid analysis in system-level design space exploration. In *Proceedings of the Int. Conference on Design, Automation, and Test in Europe (DATE '12)*, pages 781–786, 2012.

[5] W. Quan and A. D. Pimentel. An iterative multi-application mapping algorithm for heterogeneous mpsocs. In *Embedded Systems for Real-time Multimedia (ESTIMedia), 2013 IEEE 11th Symposium on*, pages 115–124. IEEE, 2013.

[6] W. Quan and A. D. Pimentel. A scenario-based run-time task mapping algorithm for mpsocs. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 131:1–131:6, New York, NY, USA, 2013. ACM.

[7] W. Quan and A. D. Pimentel. Towards exploring vast mpsoc mapping design spaces using a bias-elitist evolutionary approach. In *Proceedings of the 17th Euromicro Conference on Digital System Design*, 2014.

[8] R. Sarikaya, C. Isci, and A. Buyuktosunoglu. Runtime application behavior prediction using a statistical metric model. *Computers, IEEE Transactions on*, 62(3):575–588, March 2013.

[9] L. Schor, I. Bacivarov, D. Rai, H. Yang, S.-H. Kang, and L. Thiele. Scenario-based design flow for mapping streaming applications onto on-chip many-core systems. In *Proc. of CASES'12*, pages 71–80, 2012.

[10] A. K. Singh, A. Kumar, and T. Srikanthan. Accelerating throughput-aware runtime mapping for heterogeneous mpsocs. *ACM Trans. Des. Autom. Electron. Syst.*, 18(1):9:1–9:29, Jan. 2013.

[11] P. van Stralen and A. D. Pimentel. Scenario-based design space exploration of mpsocs. In *Proc. of IEEE ICCD'10*, pages 305–312, October 2010.

[12] 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. *Computers Digital Techniques, IET*, 5(2):123–135, 2011.