Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Scenario-based run-time adaptive MPSoC systems

Wei Quan [a,b,*], Andy D. Pimentel [a]

[a] Informatics Institute, University of Amsterdam, Science Park 904, 1098XH Amsterdam, The Netherlands
[b] School of Computer Science, National University of Defence Technology, Yanwachi Main Street 47, Changsha, Hunan, China

ARTICLE INFO

Article history:
Received 3 June 2015
Revised 2 October 2015
Accepted 13 November 2015
Available online 2 December 2015

Keywords:
Embedded systems
Multiprocessor system-on-chip (MPSoC)
Design space exploration
Task mapping
Adaptive systems
Reconfiguration

ABSTRACT

The ever-increasing performance demand of modern embedded applications drives the development of multi-processor system-on-chip (MPSoC) systems in the embedded domain. Today's MPSoC-based products increasingly have to deal with multiple application execution scenarios which may change dynamically at run time. To improve the system performance, a state-of-the-art solution is to dynamically adapting the allocation of system resources at run time for each execution scenario based on pre-determined resource schemes that have been optimized at design time. However, such approaches will not work well for MPSoC systems that have a large number of execution scenarios and/or frequent run-time variations in execution scenario behavior. In this work, we therefore propose a scalable run-time self-adaptive framework for MPSoC systems that addresses these problems, thereby considerably improving the system efficiency.

1. Introduction

Due to the ever-increasing performance demands of modern embedded applications, the use of heterogeneous multi-processor system-on-chip (MPSoC) systems has become increasingly popular in the embedded systems domain. Today's MPSoC systems often require supporting an increasing number of applications and standards, where multiple applications can run simultaneously and concurrently contend for system resources. For each single application, there may also be different execution modes (or program phases) with different computational and communication 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 which 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. Here, one can distinguish two forms of dynamic application behavior: inter-application dynamism and intra-application dynamism, which are often captured using *scenarios* [1,2]. This means that there are two different kinds of scenarios: inter-application scenarios describe the simultaneously running applications in the system, while intra-application scenarios define the different execution modes for each application. The combination of these inter- and intra-application scenarios are called *workload scenarios*, and specify the application workload in terms of the different applications that are concurrently executing and the mode of each application as shown in Fig. 1. The change of workload scenarios over time on a certain MPSoC system typically depends on environmental behavior, such as initiated by a user.

The mapping of application tasks onto the underlying system resources plays a crucial role in achieving high performance in MPSoC systems. The performance of a workload scenario may vary greatly among different mappings. To enable MPSoC systems to support the application dynamism more effectively, a state-of-the-art solution would provide a light-weight resource scheduler that allows for reconfiguring the system at run time based on pre-optimized system configurations, such as task-to-resource mappings, derived at design time [3–8]. This type of methods can be divided into two stages. The first stage is the design-time preparation which determines one or multiple system configurations for each possible scenario that may appear on the target system. For example, these configurations could be different task mappings optimizing the system for e.g. performance and/or energy consumption. The second stage is the run-time stage in which a resource scheduler chooses the appropriate system configuration from the pre-optimized configurations based on the current active workload scenario on the system.

However, these state-of-the-art run-time mapping solutions typically still lack scalability and the capability to throttle adaptivity. 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

* Corresponding author at: Informatics Institute, University of Amsterdam, Science Park 904, 1098XH Amsterdam, The Netherlands. Tel.: +31205255355.
E-mail addresses: w.quan@uva.nl, quanwei02@gmail.com (W. Quan), a.d.pimentel@uva.nl (A.D. Pimentel).

http://dx.doi.org/10.1016/j.sysarc.2015.11.004
1383-7621/© 2015 Elsevier B.V. All rights reserved.

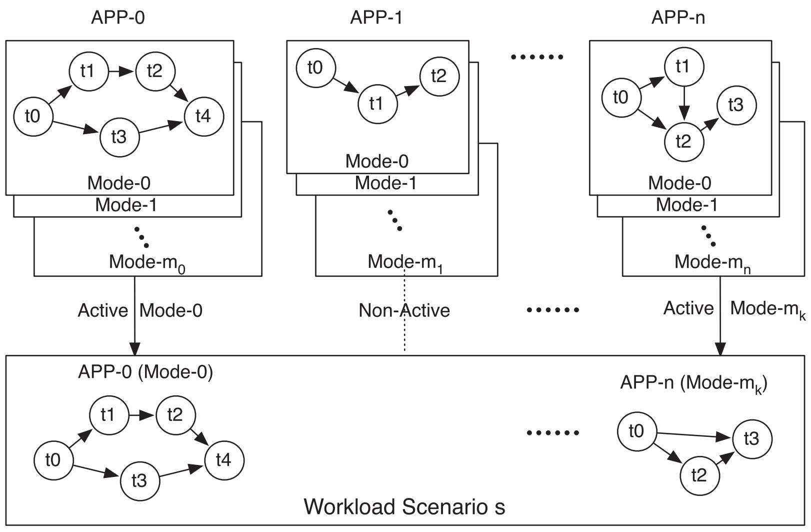

Fig. 1. Definition of a workload scenario.

pre-optimized mappings on the system. The ability of adaptivity throttling is required when the workload scenarios of a MPSoC system are relatively fine grained, which means that different workload scenarios rapidly succeed one another at run time. In the previously mentioned dynamic MPSoC solutions, the systems typically always try to reconfigure when a new workload scenario has been detected and, doing so, the *reconfiguration costs* are not explicitly taken into account. These reconfiguration costs may be substantial as they include the overhead of application tasks that may need to be migrated between different processors in the MPSoC. Especially in the case of fine-grained workload scenarios – which are workload scenarios that are only active for a short duration – such overheads may easily eliminate the benefits of reconfiguring the system: the reconfiguration itself may take longer than the performance gain that is obtained after reconfiguration. Consequently, in the case of fine-grained workload scenarios on the target MPSoC system, these dynamic MPSoC solutions may actually *degrade* the system performance, especially on heterogeneous MPSoC systems[1]. In this paper, we refer to this problem as *blind adaptivity*.

Novel Contributions and Concept Overview: In order to address the above challenges, we propose a Scenario-based run-time Adaptive Resource Allocation (SARA) framework for MPSoC systems. This framework involves:

(1) A scalable task mapping approach to solve the scalability problem. In this approach, there still exist two stages. The first stage is the design-time preparation stage during which a performance-optimized task mapping for each execution mode of each separate application (in isolation) will be derived using a static Design Space Exploration (DSE) approach. The second stage is the run-time mapping re-optimization stage. In this stage, for each detected workload scenario, the pre-optimized mappings of each isolated active application will be combined and further optimized to improve the performance of the entire scenario.

(2) A self-adaptive scheduler for adaptivity throttling. After a new mapping has been derived for a newly detected workload scenario by our task mapping approach, the scheduler tries to predict whether or not reconfiguration of the system actually is beneficial. According to this prediction, the system will either be reconfigured or not. Due to this adaptivity throttling, our scheduler is able to clearly improve the system's efficiency as compared to MPSoCs that do not provide such intelligent reconfiguration control.

Paper Organization: The remainder of this paper is organized as follows. Section 2 gives some prerequisites for this paper. Section 3 provides a more detailed description of a specific instance of SARA for the target MPSoC system in this work. Section 4 introduces the experimental environment and presents the results of our experiments. Section 5 discusses related work, after which Section 6 concludes the paper.

2. Prerequisites

2.1. Workload scenario

As introduced in the previous section, we use the concept of scenarios to capture application dynamism. A workload scenario is defined as one particular instance of the combined execution modes of a set of active target applications. Here, we denote S as the set of all possible workload scenarios for the target applications. For n target applications where each application has m execution modes, the total number of possible workload scenarios in S is $(m+1)^n - 1$. In each workload scenario $s_i \in S$, the tasks and communication between tasks are described as a directed graph $WS_i = (T_i, C_i)$ where T_i is the set of tasks in scenario s_i and C_i represents the set of communication channels between two communicating tasks. Each element in T_i and C_i, denoted as t_i^{km} and c_i^{kn}, respectively represents the mth task and the nth communication channel in application app_k which is active in workload scenario s_i.

2.2. Architecture model

In this work, we restrict ourselves to heterogeneous MPSoC architectures with shared memory[2]. An architecture can be modeled as a graph $MPSoC = (PE, M)$, where PE is the set of processing elements used in the architecture and M is a multiset of pairs $m_{ij} = (pe_i, pe_j) \in PE \times PE$ representing a buffered communication medium, composed of a network channel (like a Bus, NoC, etc.) and a buffer located in shared memory, between processors pe_i and pe_j. However, our proposed approach is not limited to the architecture we assumed here.

[1] Usually, the reconfiguration overhead on a heterogeneous MPSoC is higher than the on a homogeneous MPSoC.

[2] In our work, a basic assumption is that the target MPSoC system is over designed for all the target applications. It means that the system can handle all the application constraints as a basic requirement.

It can be applied to any other architecture if the corresponding predictor needed by the run time scheduler, which will be introduced in the next section, is changed accordingly.

2.3. Task mapping

The task mapping defines the binding of the components in a workload scenario (including the tasks and the communication channels) to the underlying architecture resources. Given a workload scenario and a target MPSoC, a correct mapping is a pair of unique assignments (μ: $T \rightarrow PE$, η: $C \rightarrow M$) such that it satisfies $\forall c \in C, source(\eta(c)) = \mu(source(c)) \wedge destination(\eta(c)) = \mu(destination(c))$. For each workload scenario $s_i \in S$, the possible task mappings are denoted as TM_i with each single mapping $tm_i^j \in TM_i$ complying with the mapping constraint. In this work, we assume that the task mapping of applications on the target system only can be changed by task migration. Under these definitions, the computation cost of task $t_i^{km} \in T_i$ and the communication cost of $c_i^{kn} \in C_i$ in workload scenario s_i under the task mapping of tm_i^j is represented as et_{ij}^{km} and ec_{ij}^{kn} respectively.

2.4. Objective formulation

Our target applications belong to the domain of streaming applications (like multimedia applications) that continuously process an incoming stream of data elements. To capture the duration of a workload scenario in this case, we use the concept of *scenario frames* to define the workload of active applications. Here, we define one *scenario frame* as the time it takes for each active application within a specific workload scenario to process at least one unit (frame) of data (e.g., processing a single MP3 frame, an H264 frame, etc.). Therefore, the frame execution time of a workload scenario is defined as the maximum frame execution time among active applications (each processing its own unit of workload). Using this definition, the frame execution time of workload scenario s_i under mapping tm_i^j is formulated as p_i^j which can be calculated by Eq. (1).

$$p_i^j = max(p_{ij}^k) \tag{1}$$

where p_{ij}^k represents the frame execution time of app_k which is active in s_i under mapping tm_i^j. Consequently, the execution duration of workload scenario s_i under mapping tm_i^j is calculated as $p_i^j * n_i$ where n_i is the number of scenario frames executed for s_i on the system.

In our work, we assume that when a new application is started, it is added to the system using a pre-determined, default task mapping. Given a newly detected scenario, the complete task mapping of those applications that persist in the new scenario and any newly added applications needs to be reconsidered. Remapping of the application tasks in the newly detected workload scenario can be beneficial performance wise (i.e., every workload scenario has an optimal task mapping) but this depends on both the actual performance gain of reconfiguring the system and the reconfiguration costs. The reconfiguration costs include two parts: (1) the overhead of finding a new mapping and making a reconfiguration decision, and (2) the task migration cost that may occur during system reconfiguration. Here, we denote the reconfiguration costs for scenario s_i to change from mapping tm_i^j to $tm_i^{j'}$ as $c_i^{jj'}$, which includes the time of finding the new mapping $tm_i^{j'}$, making the reconfiguration decision for the new mapping and task migration during reconfiguration. This implies that the system reconfiguration benefit B can now be expressed as:

$$B = (p_i^j - p_i^{j'}) * n_i - c_i^{jj'} \tag{2}$$

Our objective is to maximize the system performance for a sequence of workload scenarios S^*. This means that we want to maximize the total system reconfiguration benefit $\sum_{s_k \in S^*} b_k * B_k$, where $b_k \in \{0, 1\}$ is the migration decision made by the run time scheduler and B_k is the reconfiguration benefit of workload scenario $s_k \in S^*$. Obviously, the solution to this problem is $b_k = 0$ if $B_k <= 0$ and $b_k = 1$ if $B_k > 0$. Consequently, to achieve our objective, the system needs to correctly predict the system reconfiguration benefit B for each workload scenario.

3. Scenario-based run-time adaptive resource allocation framework

As mentioned in the introduction, our SARA framework [9] is proposed to address the scalability and blind adaptivity issues in self-adaptive MPSoC systems. 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 collecting system statistics. To detect and identify workload scenarios in our Sesame simulation framework, we need to instrument the source code of target applications with scenario related events such as *STARTSCENARIO* and *ENDSCENARIO*. When a processing element in a Seseme system model encounters a *STARTSCENARIO* event of an application, it will register this application with its execution mode information in the RSM to notify that a new application has started execution on the system. Similarly, for an *ENDSCENARIO* event of an application, the processing element will unregister the application in the RSM. According to the registered application information in the RSM, the active workload scenario on the target system can be identified. The RMG and ARS are responsible for the system adaptation and address the scalability and blind adaptivity issues as explained above. Fig. 2 shows the high-level system 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. In this section, 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. decide whether or not to adjust the task mapping. The details of our proposed SARA framework will be described in details in the following subsections.

3.1. Design time preparation

For our SARA framework, we need to prepare several important information at design time and store these information on the system for run-time usage. The first one is the mappings that will be considered for run-time mapping optimizations. In our framework, we consider a hybrid task mapping approach where the mapping optimization process include a design time application-level mapping exploration and a run time scenario-level mapping optimization. It means that, at design time, we need do several mapping explorations to provide the SARA framework with the required mapping information for the target applications. This hybrid task mapping approach will be introduced in the next subsection.

Besides the pre-optimized mappings, the execution time of each task on each processor, the communication times between tasks on different communication channels of the target system and the migrating data size between processors for each task should also be analyzed at design time and stored on the target system for the purpose of mapping performance prediction and system reconfiguration cost

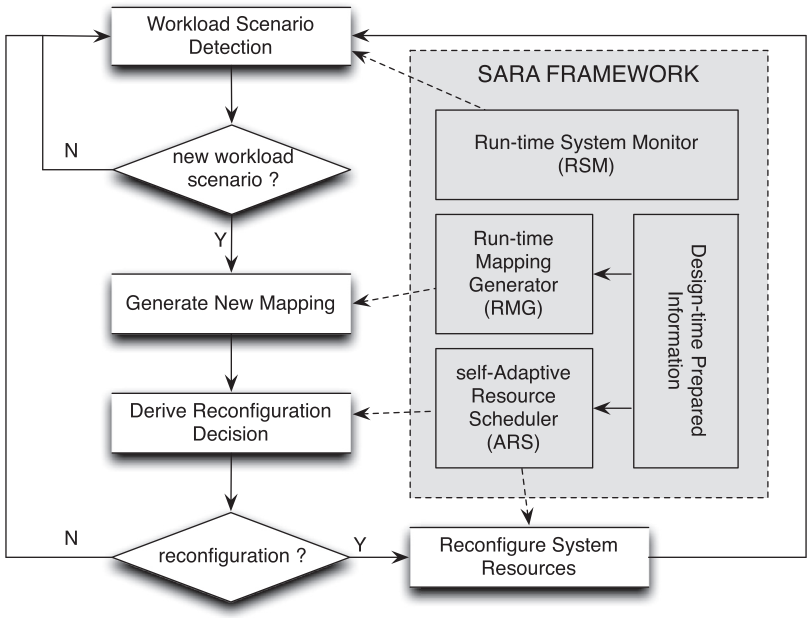
Workload Scenario Detection
new workload scenario ?
N
Y
Generate New Mapping
Derive Reconfiguration Decision
reconfiguration ?
N
Y
Reconfigure System Resources
SARA FRAMEWORK
Run-time System Monitor (RSM)
Run-time Mapping Generator (RMG)
self-Adaptive Resource Scheduler (ARS)
Design-time Prepared Information

Fig. 2. The structure and workflow of SARA framework.

prediction. These design-time prepared information will be used by the RMG for run-time dynamic mapping optimization and ARS for adaptivity throttling for the target MPSoC system.

3.2. Scalable run-time task mapping

The problem of optimally mapping a set of tasks onto a set of given heterogeneous processors for maximal performance (e.g. 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. Solutions for this problem can be divided into three categories: static [10–13], dynamic [14,15] and hybrid task mapping algorithms [16,17] which, respectively, work at design time, run time and both design time and run time.

Traditionally, the task mapping problem is solved statically at design time for which there are many known task mapping algorithms targeting different application domains and different hardware architectures. These algorithms typically use computationally intensive search methods to find the optimal mapping or near optimal mapping for the applications that may run on the system. Dynamic task mapping techniques, on the other hand, cannot be computationally intensive as they have to efficiently make task mapping decisions at run time. Therefore, these techniques typically use heuristics to find good task mappings. Evidently, static task mapping techniques usually obtain mappings of higher quality compared to those derived from dynamic algorithms as the former allow for exploring a larger design space for the underlying architecture. This, of course, at the cost of consuming more time. Another drawback of static mapping techniques is that they cannot cope with dynamic application behavior in which different combinations of applications can be executing concurrently over time that are contending for system resources. To overcome the shortcomings of pure static and dynamic task mapping algorithms, hybrid (semi-static) approaches have become increasingly popular in recent years. Usually, in this kind of approaches, multiple mapping solutions are found at design time and applied at run time based on the current state of the system. However, 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 (to be used at run time). One solution to address this problem is by reducing the number of workload scenarios by means of clustering [2,6]. The clustering based approaches divide the target workload scenarios into different clusters, and only explore the best mapping solution or several good ones for each cluster. Consequently, the number of mappings need to be explored and stored can be greatly reduced. In the extreme case, one can put all the target workload scenarios in a cluster and only explore a single mapping which shows on average the best performance for all scenarios for this cluster. In this case, only one mapping needs to be found at design time and it will be used for all the workload scenarios. However, these methods still suffer from an additional problem of searching for optimal mappings of (clustered) workload scenarios at design time: 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 the EIM algorithm proposed in [7] 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. The middle grey box of Fig. 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 [18]. This DSE approach uses an genetic algorithm with an domain knowledge optimised mutation operator to speed up the exploration process. The optimised mutation operator can guide the genetic algorithm to search the good mapping candidates (smaller makespan and well balanced) that have a higher probability to be the best mapping solution[3]. By using this DSE approach, we can 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

[3] We refer the interested readers to [18] for more details

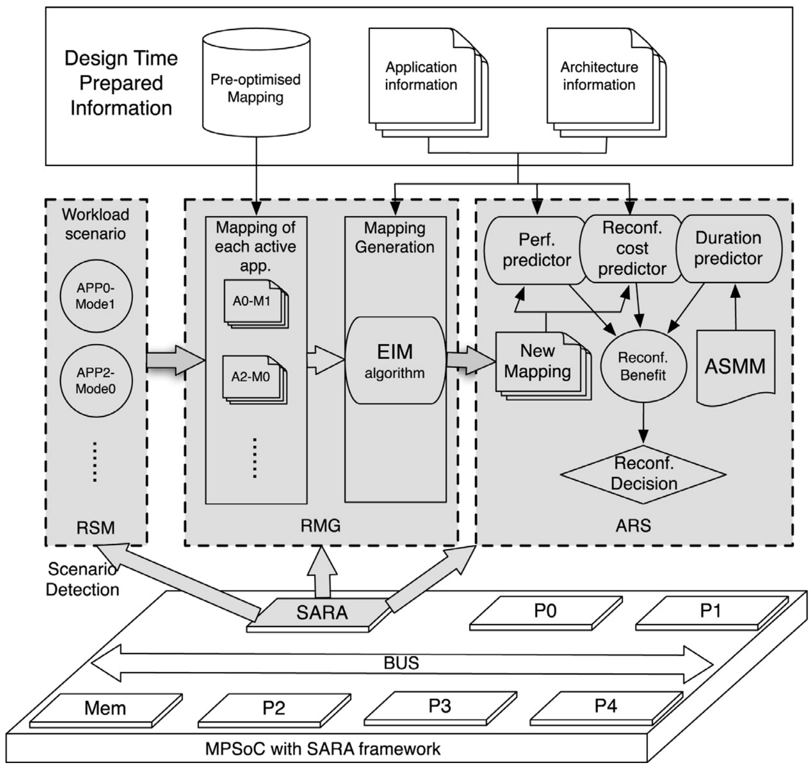

Fig. 3. An instance of SARA for the target MPSoC system.

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 [6,7]. 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 of the merged mapping will already 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 of which the rationale can be explained as follows. 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 more effectively supports the addition of new applications to the target MPSoC system. The disadvantages of this approach mainly exist in two aspects. Firstly, the quality of the mappings derived from our approach is slightly worse than the mappings generated by static task mapping approaches. As the static mapping approaches can explore a larger or even the entire mapping solution space to find the exactly optimal mapping solution. Compared with dynamic on-the-fly task mapping approaches, the complexity of our approach is higher as we need to prepare the required information at design time for run-time usage.

3.3. Adaptivity throttling

In current state-of-the-art run-time task mapping approaches for MPSoCs, the system will typically be reconfigured (i.e., the task mapping will be adapted) when a new workload scenario appears on the system, irrespective of the trade-off between reconfiguration costs and benefits. This implies that task migration might occur during this reconfiguration process of which its cost cannot be ignored, especially for heterogeneous MPSoC systems and those cases where the duration of workload scenarios are relatively short.

According to the derivation of Eq. (2) in Section 2.4, it is evident that only if $(p_i^j - p_i^{j'}) * n_i$ is larger than $c_i^{jj'}$ – 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 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 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 $(p - p')$, the scenario execution duration in *scenario frames* (n), 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. The right grey box of Fig. 3 illustrates how the ARS component conditionally reconfigures the target system based on the outcome of the prediction models (i.e, $(p - p') * u > c$).

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 [19] can therefore be applied. The prediction of scenario execution duration is the most important part of the ARS. It is 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 the history-based predictor such as a last value predictor, table-based predictor and the Statistical Metric Model (SMM) [20]. 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. According to the target architecture, we consider the following prediction models for the above three parameters [21].

3.3.1. Mapping performance prediction

For our target heterogeneous MPSoC system with shared memory, at design time, the execution time of each task and the communication time between tasks for one unit of workload of each application have been analyzed and stored on the target system. Using this information, the performance of a certain workload scenario s_i under a target task mapping tm_i^j is derived by Eq. (1) where the performance of each active application app_k is predicted by Eq. (3).

$$p_{ij}^k = CC_{ij}^k + BK_{ij}^k \tag{3a}$$

$$CC_{ij}^k = \sum_{0 \le m < t} et_{ij}^{km} + \sum_{0 \le n < l} ec_{ij}^{kn} \tag{3b}$$

$$BK_{ij}^k = \sum_{q \ne k} \sum_{t_i^{qs} \in T_{ij}^{qk}} et_{ij}^{qs} + \sum_{q \ne k} \sum_{c_i^{qs} \in C_{ij}^{qk}} ec_{ij}^{qs} \tag{3c}$$

where CC_{ij}^k represents a conservative estimate (no concurrency is taken into account) of the total execution time of app_k in scenario s_i under mapping tm_i^j of all t tasks and l communications in app_k. We assume all the tasks and communications are executed in sequential when calculating CC_{ij}^k. Under this assumption, we can simply add together the overhead of tasks and communications under the target mapping. BK_{ij}^k is the total time of tasks $t_i^{qs} \in T_{ij}^{qk}$ and communications $c_i^{qs} \in C_{ij}^{qk}$ from other active applications that contend resources with app_k. Here, T_{ij}^{qk} and C_{ij}^{qk} is the set of tasks and communications from app_q that are mapped onto the same resource (under tm_i^j) with any task and communication of app_k respectively. BK_{ij}^k represents the possible blocking time for the target application because of the resource contention between applications. Combining CC_{ij}^k and BK_{ij}^k together, we can predict a worst case execution time for the target application.

3.3.2. Reconfiguration cost prediction

As mentioned in Section 2.4, the reconfiguration cost on our target MPSoC system includes two parts: the overhead of our SARA framework and the task migration cost during system reconfiguration. The overhead of SARA is determined by means of measurements and the task migration cost is calculated by Eq. (4b). The model used for the task migration overhead prediction is a simple linear analytic model. The rationale behind this model is based on the task migration mechanism we assume for the target MPSoC system in which the migrating data is transferred via the MPSoC's shared memory and the ARS sequentially controls task migrations. Here, we label the overhead of SARA as *CSara* and the task migration cost as *CMig*. Consequently, for a certain workload scenario s_i with an original task mapping tm_i^j and a newly generated mapping $tm_i^{j'}$, the reconfiguration cost can be derived by the following equation. Notice that, the overhead of this specific SARA instance (*CSara*) includes the time of deriving a new mapping in the RMG, estimating mapping performance for both the old mapping and the new mapping, calculating task migration cost *CMig* and updating the system run-time information (e.g., actual scenario execution duration) in SARA.

$$c_i^{jj'} = CSara_i^{jj'} + CMig_i^{jj'} \tag{4a}$$

$$CMig_i^{jj'} = \left(\sum_{t_k \in T_i^{jj'}} ms_k \right) \Big/ r_{mem} \tag{4b}$$

where $T_i^{jj'}$ is the set of tasks in workload scenario s_i that need to be migrated from mapping tm_i^j to $tm_i^{j'}$, ms_k represents the amount of migrating data for task t_k, and r_{mem} is the memory access speed.

3.3.3. Reconfiguration decision prediction

For the scenario execution duration n, which is a dynamic parameter that relates to the user/system behavior, we do not try to predict the exact value. As in most MPSoC systems (except for systems with periodic workload scenarios), it is hard to accurately predict the exact value of this parameter based on history information. However, we can avoid this problem by approaching our goal slightly differently. Our purpose is to derive a reconfiguration decision based on the reconfiguration benefit B. If the execution duration of the detected workload scenario is higher than a certain value (boundary), then the system can be reconfigured. Consequently, there is no need to directly predict a concrete value for the scenario execution duration but, instead, it is sufficient to predict the probability that the execution duration is higher than the given boundary. The details of how to predict a reconfiguration decision for a newly detected workload scenario will be explained in the following.

For a newly detected workload scenario s_i, according to the predicted performance improvement and the system reconfiguration cost[4], the ARS can determine a lower bound bn_i for the execution duration of this workload scenario:

$$bn_i = c_i^{jj'} / (p_i^j - p_i^{j'}) \tag{5}$$

In this work, we propose an Accumulated Statistical Metric Model (ASMM), which is based on the Statistical Metric Model (SMM) [20], to predict a reconfiguration decision based on the derived lower bound of scenario execution duration. The SMM is a probability distribution over application patterns of varying length. It models the conditional distribution on the identity of the *ith* (quantized) sample given the identities of all previous (quantized) samples in a metric sequence. The difference between our ASMM and the original SMM concerns the way of how the prediction value is generated, as will be explained below.

Using our ASMM, we build a metric model based on the probability distribution of scenario execution duration for each workload scenario, which means each workload scenario has its own ASMM. When a new workload scenario is detected, the system scheduler uses the ASMM of this workload scenario to predict its duration. Fig. 4 gives an example of the ASMM-based prediction of a reconfiguration decision when using 3 history samples. In our problem, the samples of scenario execution duration are measured in the number of *scenario frames* (F). These frame numbers are quantized using a limited number of bins (or *ranks*) to reduce the complexity of our predictor, see the upper part of Fig. 4 for an example. The lower right part of Fig. 4 shows a simple instance of our ASMM. It includes three kinds

[4] Note that the scenario duration prediction cost is negligible compared to the other costs of system reconfiguration since it only involves a table look-up. It is therefore not included in the reconfiguration cost in the SARA implementation.

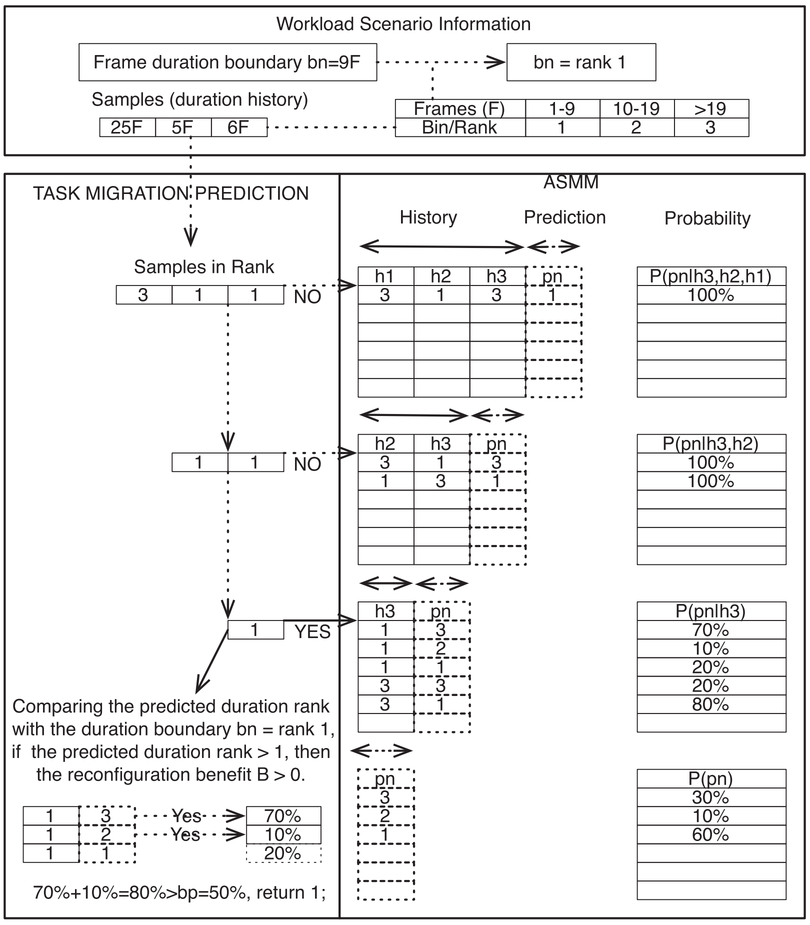
Workload Scenario Information
Frame duration boundary bn=9F
bn = rank 1
Samples (duration history)
25F 5F 6F
Frames (F) 1-9 10-19 >19
Bin/Rank 1 2 3
TASK MIGRATION PREDICTION
Samples in Rank
3 1 1 NO
1 1 NO
1 YES
Comparing the predicted duration rank with the duration boundary bn = rank 1, if the predicted duration rank > 1, then the reconfiguration benefit B > 0.
Yes
Yes
70% 10% 20%
70%+10%=80%>bp=50%, return 1;
ASMM
History
Prediction
Probability
h1 h2 h3 pn
P(pn|h3,h2,h1)
100%
P(pn|h3,h2)
100%
100%
P(pn|h3)
70% 10% 20% 20% 80%
P(pn)
30% 10% 60%

Fig. 4. The workflow of a simple ASMM with a max of 3 history samples.

of tables: the execution duration history tables, duration prediction tables and tables with probabilities for all possible duration predictions. The first two types of tables store *rank numbers*. The width and depth of the history tables usually determine the prediction ability of the ASMM and should be set based on the target problem.

To illustrate the ASMM-based reconfiguration decision, please consider the lower left part of Fig. 4. The input of our ASMM is the (quantized) execution duration sample history (top of Fig. 4) of the detected scenario and the duration bound bn_i. According to the detected workload scenario, the corresponding ASMM will be used to determine the reconfiguration decision. In our example, the ASMM first checks the history table with 3 history samples to see if there is a pattern match regarding the scenario's duration history. If there is no match, like in the case of our example, the ASMM will continue to search the history tables with a smaller width of history samples (i.e., using a shorter history). This process continues until there is a history pattern match or it ends up at the direct duration prediction without any execution duration history (the table at the bottom of the ASMM in Fig. 4). In both cases, the duration bound bn_i is compared with all the possible duration prediction values and for those prediction values bigger than bn_i their probabilities will be accumulated: hence the name *Accumulated* SMM. This accumulated probability represents the chance of $B > 0$ if the system is reconfigured for this workload scenario. Only if the accumulated probability is larger than the probability bound *bp* (set by the designer), the ASMM will return a positive reconfiguration decision. In our example, the probability bound *bp* is set to 50%.

After having derived 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 in the ARS, our adaptive MPSoC system is able to cope with fine-grained workload scenarios for which it is not beneficial to reconfigure the system.

4. Experiments

4.1. Experimental setup

To illustrate the effectivity of our SARA framework, we deploy the system-level MPSoC simulation framework from the work of [22] which is based on the open source Sesame simulator [23]. This Sesame-based modeling and simulation environment facilitates efficient performance analysis of embedded (media) system architectures. The most important feature of this simulator for this work is its ability to support the simulation of run-time system reconfiguration of MPSoC systems. This makes the modeling and simulation of our SARA instance in this simulator relatively easy.

In our experiments, we aim at showing how our SARA framework improves the system performance by applying the proposed scalable run-time task mapping approach and adaptivity throttling. To this end, it is important to assess system performance under a variety of different workload scenario behaviors. The actual functionality of the applications within these scenarios is, on the other hand, of lesser importance for this purpose. Therefore, we use synthetic streaming applications within workload scenarios to simplify the simulation process. The target hardware architecture is a bus-based heterogeneous MPSoC platform containing five different processing elements with different computational characteristics, a shared memory and a controlling processor that runs the SARA framework.

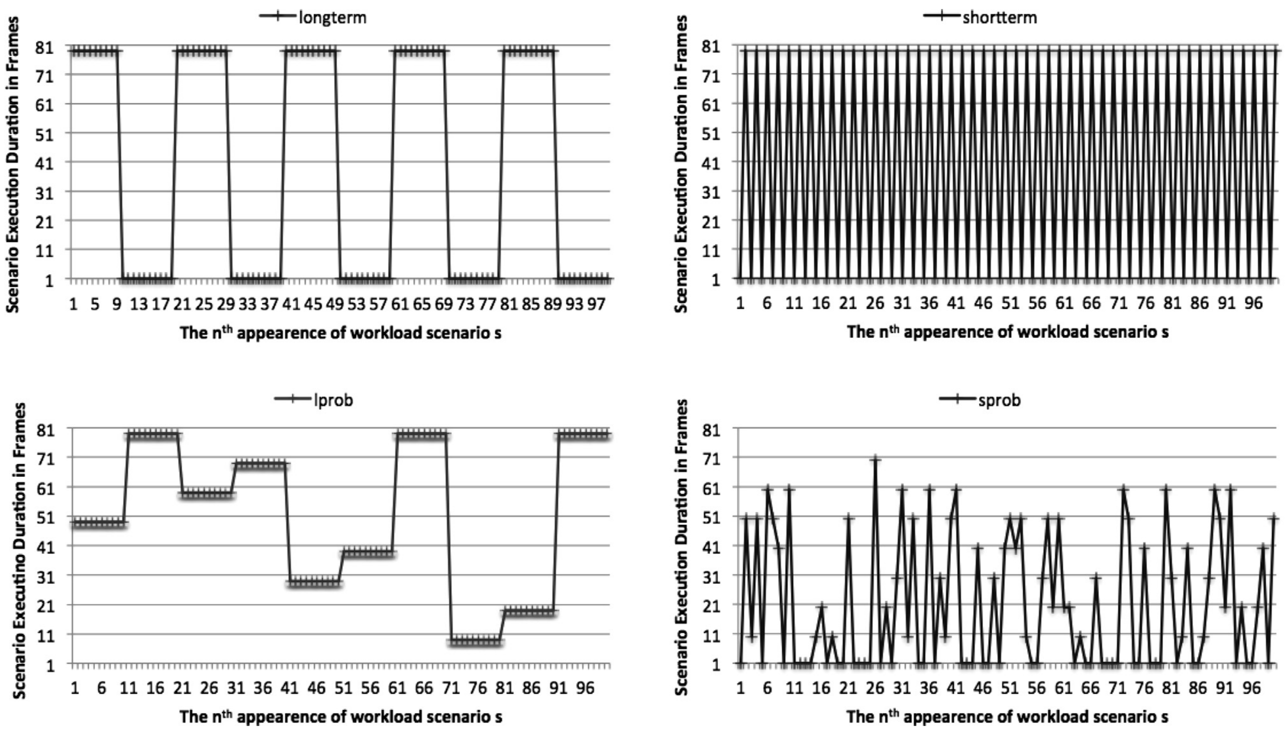

Fig. 5. The scenario duration distribution used for generating workload scenarios.

4.2. Evaluating adaptivity

In this experiment, the main goal is to evaluate the adaptivity of our approach. For this reason, a relatively small number of workload scenarios is considered. We use five synthetic streaming applications with each application containing only 1 execution mode. In this case, the total number of workload scenarios is therefore 31 ($2^5 - 1$). The number of tasks in each application ranges from 4 to 8. We assume that each task can be executed on each processor of the target MP-SoC using the corresponding pre-compiled code (stored in the shared memory). The task execution time and migration data size of each task on each processor have been randomly generated and range between 10,000 and 100,000 time units (simulation cycles) and between 50 K and 500 K Bytes respectively. Communications between tasks range from 1000 to 10,000 Bytes in size.

To model dynamic application behavior over time (e.g. due to user behavior), we generate four different workload scenario sequences. These sequences are generated in two steps. The first step is to choose a workload scenario from the total 31 workload scenarios considered in our experiment. Each workload scenario has the same probability to be selected. The second step is to generate the duration in *scenario frames* of the selected workload scenario. This process iterates until a pre-defined total frame number (100,000 frames in our case) has been achieved for the scenario sequence. As the workload scenarios considered in our test case need around 40 frames on average to neutralize the reconfiguration cost, we limit the duration of each workload scenario to a value between 1 and 80 frames. In our four scenario sequences, the duration of each workload scenario and frequency of changes to this duration are generated using different distributions as shown in Fig. 5. Here, the x-axis represents the nth appearance of one specific workload scenario in the scenario sequence and the y-axis represents the execution duration in *scenario frames* for that particular appearance of the workload scenario. In the *longterm* distribution, the duration of a workload scenario is either long or short and does not frequently change, whereas in the *shortterm* distribution the scenario execution duration does frequently change. The frequency of changes in the *lprob* and *sprob* distributions are similar to *longterm* and *shortterm* but the actual scenario execution duration now has been generated from the following probability distributions: 9 → 10%, 19 → 10%, 29 → 10%, 39 → 10%, 49 → 10%, 59 → 10%, 69 → 10%, 79 → 30% for *lprob* and 1 → 30%, 11 → 10%, 21 → 10%, 31 → 10%, 41 → 10%, 51 → 10%, 61 → 10%, 71 → 10% for *sprob* respectively. Taking the *sprob* distribution as an example, a workload scenario has a probability of 30% that it will be executed for only 1 frame and 10% for each of the other durations (11 frames, 21 frames, etc.). Where the *longterm* and *shortterm* scenario sequences more or less reflect extreme cases in workload scenario behavior, the two *prob* sequences possibly exhibit a more realistic view on dynamic behavior in application workloads.

As we limit the scenario execution duration from 1 to 80 frames, we divide the scenario execution durations into 8 *bins/ranks* in our ASMM, each containing a frame range of 10. The maximal width of the history tables in the ASMM is 2 which is large enough for our test cases. The maximal depth of the history tables depends on both the maximal width and the number of *bins* allocated for scenario execution duration and consequently is set to 64 ($8*8$) in our ASMM. The probability bound *bp* for our ASMM is set to 50%. This will not lead to either a pessimistic or aggressive decision.

In this experiment, we compared our SARA framework with three alternative approaches. First, an approach (*STATIC*) [13] 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. Second, an approach (*MIGRATE-OPT*) that always reconfigures the system whenever a new workload scenario has been detected according to a corresponding pre-optimized mapping derived at design time. 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* [7]).

The results of the total execution time (including system reconfiguration time) of all workload scenarios in each generated scenario sequence are shown in Fig. 6. In this figure, we can clearly see that our *SARA* approach outperforms the three alternative approaches in

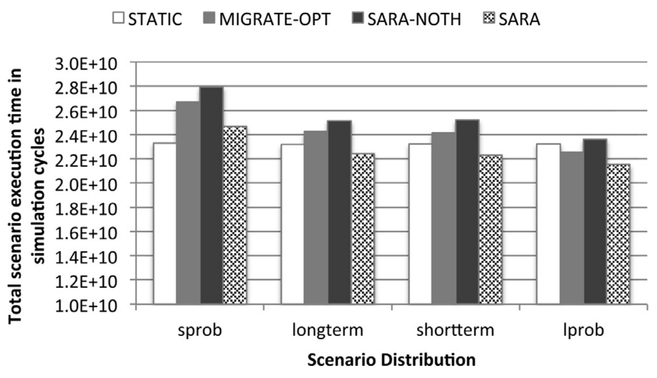

Fig. 6. Performance comparison of different resource scheduling approaches.

almost all scenario cases except the *STATIC* approach in the *sprob* case, whereas *SARA-NOTH* performs the worst. According to the above-mentioned distribution of each scenario sequence, the average scenario duration of *sprob* is less than 30 frames and *lprob* is higher than 60 frames, where *longterm* and *shortterm* both have a mean execution duration of around 40 frames. As mentioned before, the workload scenarios considered in our test cases need around 40 frames on average to neutralize the reconfiguration cost. From the perspective of average scenario duration, one can easily understand the reason of why the approaches that always reconfigure the system for a newly detected workload scenario, i.e. *MIGRATE-OPT* and *SARA-NOTH*, perform worse than *STATIC* in the scenario sequences of *sprob, longterm* and *shortterm*. However, when the average scenario duration is large enough like in *lprob*, this kind of approaches start to perform better than *STATIC*. Notice that with even larger average scenario durations the performance difference of the approaches that (always) migrate and *STATIC* wil only become larger. Comparing *MIGRATE-OPT* and *SARA-NOTH*, the former one always performs better as it uses the pre-optimized mappings for entire workload scenarios which are better in mapping quality compared to the mappings found by the EIM algorithm in *SARA-NOTH*. With regard to our SARA instance, it shows good behavior for each test case because of its ability to throttle adaptivity. It even performs relatively well in the case of *sprob* where 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.

To better understand how these adaptive techniques work, we zoom into the run-time executing behavior of a certain workload scenario in the scenario sequences for these techniques. Fig. 7 shows the results of the scenario execution time (including system reconfiguration time) of the first 100 appearances of a single selected workload scenario in our four workload scenario sequences. Clearly, the *MIGRATE-OPT* approach performs best and *STATIC* performs worst in those cases where the system should have been reconfigured (i.e., scenario execution duration is large enough such that reconfiguration is beneficial): see the top parts of the four figures. On the other hand, in the cases where the system should not be reconfigured (bottom parts of the four figures), *STATIC* is the best and *SARA-NOTH* is the worst. We can also see that the scenario execution time is influenced by the reconfiguration cost. For example, consider the bottom parts of the upper two graphs (*longterm* and shortterm), i.e., small scenario execution durations. Here, one can clearly see that *SARA* consistently outperforms *MIGRATE-OPT* and *SARA-NOTH* since the latter two approaches are negatively affected by the reconfiguration overhead whereas *SARA* is not because it accurately predicted that reconfiguration is not beneficial. However, compared with *STATIC, SARA* still suffers from its computational overhead. Moreover, erroneous predictions inducing unnecessary system reconfigurations also affect the performance of *SARA*. In the upper two graphs (*longterm* and *shortterm*), there is a clear pattern for the duration of workload scenarios where *SARA* can accurately predict. In these two cases, the performance is mainly affected by its computational overhead. However, in the case of *sprob*, a high prediction error is the main factor that

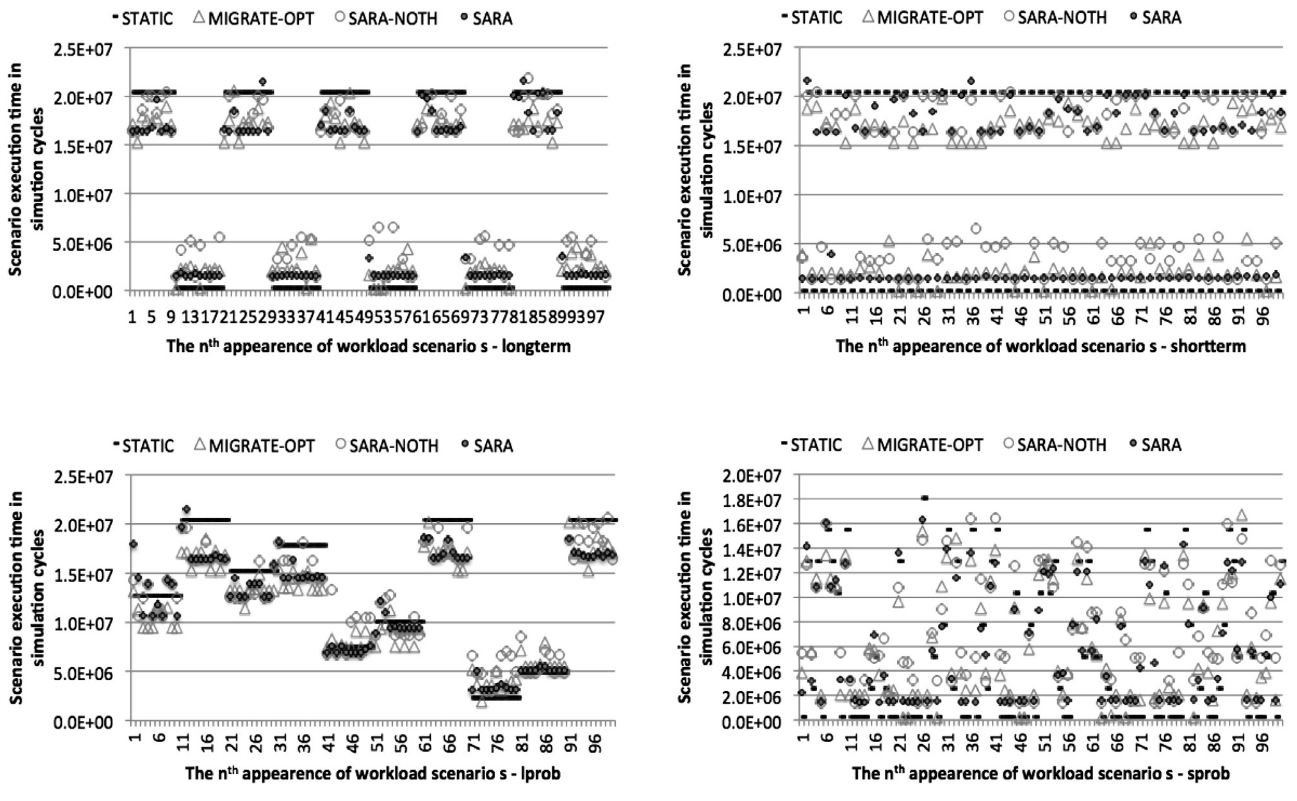

Fig. 7. execution pattern of different resource scheduling approaches.

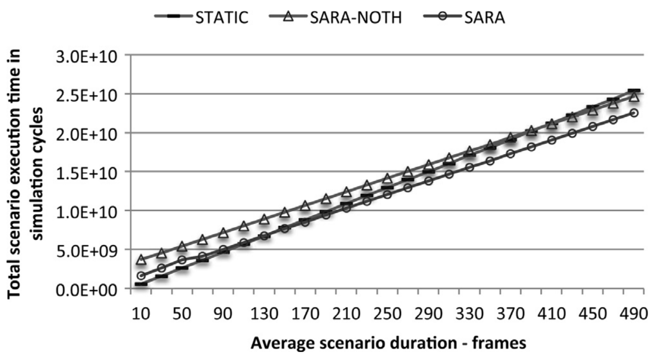

Fig. 8. Performance comparison of different resource scheduling approaches in complex workload scenarios.

influences the performance of *SARA* as it is hard to predict such a random scenario behavior with frequent changes. This implies that the prediction error mainly comes from the scenario duration predictor (ASMM) in the ARS of our SARA instance. A similar situation can be seen in the first 10 appearances of scenario *s* in the graph of *lprob* where *SARA* shows a poor behavior (filled dots randomly distributed around the straight line of *STATIC*) whereas SARA's performance in the other parts of this graph is much better. This is also caused by a high prediction error in SARA. However, in this case, the prediction error is caused by the accuracy of the mapping performance and reconfiguration cost predictors in the ARS of SARA. As the scenario duration of these first 10 appearances of scenario *s* is very close to the average frame number needed to neutralize the reconfiguration cost in our test cases, a small prediction error of these two parameters can easily lead to an erroneous system reconfiguration prediction.

From this experiment, we can see that our SARA framework is able to handle complex and dynamic workload scenarios and further improves the system efficiency by run-time task remapping and adaptivity throttling.

4.3. Evaluating scalability

To evaluate the scalability of our approach, we consider ten synthetic streaming applications in this experiment, where each application has the same properties as the applications used in the previous experiment. Consequently, there are 1023 workload scenarios in total. For the purpose of simulating the dynamic application behavior over time, we use a scenario sequence which is similar to *lprob* from the first experiment but without limiting the scenario execution duration. In this experiment, the impact of an increasing scenario execution duration will be studied. Furthermore, the parameters of the ASMM are the same as in the first experiment except for the maximal depth of the history tables which is set based on the actual scenario duration *bins*.

Fig. 8 shows the total execution time (including system reconfiguration time) of 100 appearances of the workload scenario with all the 10 applications active. The *x*-axis in Fig. 8 represents the average scenario duration (in scenario frames) of the 100 appearances of the workload scenario. For better visibility of the results, we have discarded the results of *MIGRATE-OPT* which is a line between *SARA-NOTH* and *SARA*, but converges earlier with *SARA* than *SARA-NOTH*. From this figure we can clearly see that our SARA framework still works well when the number and the complexity of workload scenarios increases. Note that in this experiment, as the workload scenario contains a large number of tasks and communication channels, the system reconfiguration cost is also larger compared with the first experiment. Consequently, the number of average scenario frames needed for neutralizing the reconfiguration cost is also bigger than in the first experiment. Taking our *SARA* instance for example, the

Table 1
Run-time system storage demands (*bytes*) of the different techniques.

Experiment1	Mapping	App&Arch Inf.	ASMM
STATIC	x	x	x
MIGRATE-OPT	880	x	x
SARA-NOTH	55	920	x
SARA	55	920	Dynamic
Experiment2	**Mapping**	**App&Arch Inf.**	**ASMM**
STATIC	x	x	x
MIGRATE-OPT	56320	x	x
SARA-NOTH	110	1840	x
SARA	110	1840	Dynamic

computational overhead of SARA can be neutralized when the average number of scenario frames is larger than 150 (the crosspoint of *SARA* and *STATIC* is around 150 frames).

To further improve the scalability of our SARA framework, we use a caching mechanism that aims at avoiding the run-time mapping optimization heuristic becoming a performance bottleneck of the target system when frequent scenario changes occur. To this end, the SARA framework uses a small amount of system memory like a scratchpad to cache the mappings optimized by the run-time heuristic for the workload scenarios that undergo the most frequent changes. Consequently, it is able to further save considerable computational overhead with regard to run-time mapping optimization, especially for complex workload scenarios.

4.4. System storage overhead

Regarding the run-time system storage consumption of the four studied approaches, several assumptions should be mentioned. On our target MPSoC system, we store all the design-time prepared information in the shared memory. For storing the pre-optimized mappings, we assume that the mapping information of each task and each communication channel between tasks can be stored in one *byte*. In the first experiment, there are 30 tasks and 25 communication channels in total for all the five synthetic streaming applications. Consequently, to store a pre-optimized mapping, the maximal memory usage is 55 *bytes* (all tasks and communication channels are active). The total number of tasks and communication channels in the second experiment is 60 and 50 respectively. Beside the pre-optimized mappings, in our SARA framework, we also need to store the application/system information as mentioned in Section 3.1 and the scenario execution history information if the ARS predicts scenario duration based on history information like in the ASMM-based method of our SARA instance. Here, we assume that each piece of application/system information needs one *word* of system memory and each history scenario duration is encoded using one *byte*.

Based on these assumptions, Table 1 gives the run-time system storage demands of the four approaches in the above two experiments. From this table, we can see that our SARA instance consumes the largest system memory usage in the first experiment. However, in our SARA framework, the memory usage only linearly increases with the number of applications for storing the *Pre-optimized Mappings* and *App&Arch Information* on a certain target MPSoC system. It does not have the scalability problem of *MIGRATE-OPT*. Consequently, in the second experiment, the memory usage of *MIGRATE-OPT* is much larger than *SARA-NOTH* and *SARA*. With regard to the memory usage of the ASMM in our SARA instance, it is dynamic at run-time and depends on the application behavior. Taking the memory consumption of the ASMM initialized in the first experiment as an example, in the worst case, it consumes 584 *bytes* of system memory to record all the possible execution behaviors of a workload scenario. However, during the simulation, only a small part of the worst memory usage was actually used for our test cases. Furthermore, in our SARA framework, this scenario execution duration predictor can also be implemented by other techniques like Neural Networks [24,25] if the memory usage becomes an issue.

5. Related research

In recent years, much research has been performed in the area of run-time task remapping for embedded systems to achieve better performance or save energy consumption. In these research efforts, a hybrid task mapping approach is commonly used that combines a design-time preparation with a run-time dynamic mapping policy to do task reallocation. For example, Mariani et al. [3] proposed a run-time management framework in which Pareto-fronts with system configuration points for different applications are determined during design-time DSE, after which heuristics are used to dynamically select a proper system configuration at run time. In [4], a fast and light-weight priority based heuristic is used to select near-optimal configurations explored at design time for the active applications according to the available platform resources. [8] proposes DSE strategies that perform exploration in view of optimizing throughput and energy consumption by considering a generic platform. The design points derived from the DSE will be selected efficiently at run time. In [26], Schranzhofer et al. proposed static and dynamic task mapping approaches for probabilistic applications based on static and dynamic power components. Statically pre-computed template mappings for each execution probability are stored on the system and applied at run time, allowing the system to adapt to changing environmental conditions. Based on this work, [27] presents an extension that considers only the static mapping and takes into account the communication and reconfiguration energy component.

Schor et al. [5] and Quan et al. [6] also proposed scenario-based run-time mapping approaches in which mappings derived from design-time DSE are stored for run-time mapping decisions. However, in these proposed approaches, none of them consider the problem of whether the system will benefit from the resource reconfiguration when the workload of the system changes frequently. This problem might be caused by the user behavior. In this case, it is better to consider the user behavior [28,29] or system execution history [20] to further improve the system efficiency. Sarikaya et al. [20] proposed SMM to predict the run-time application behavior, and applied this technique to an adaptive dynamic power management scheme. In our work, we modified their SMM to predict the scenario duration which is part of our adaptive run-time framework. In [28], the user behavior information is used to adapt the strategy used for resource allocation at run time. Based on the user behavior, the online machine learning model will predict which kind of communication contention should be minimized on an NoC based MPSoC system. The authors of [29] proposed a customer-aware task allocation and scheduling for MPSoCs. In their approach, an initial task allocation and scheduling (TAS) solution under the objective of minimizing the energy consumption and system lifetime for each execution mode is generated at design time. At run time, they conduct online adjustment of the TAS based on the processor usage history to guarantee the lifetime reliability and/or reduce the energy consumption. Different to these previous efforts, we use the user behavior/system execution history to control the resource allocation process.

6. Conclusion

To increase the efficiency of MPSoC systems, in this paper, we propose a scalable, run-time adaptive resource scheduler that reconfigures the system based on the system workload and user behavior. At design time, we explore performance (near) optimal task mappings for different workload scenarios. These pre-optimized mappings will then be used at run time by the resource scheduler to reconfigure the system resources. The decision of whether or not the system should be reconfigured is made explicitly based on the scenario execution history pattern. By using the proposed approach, the system is able to effectively adapt its behavior according to user behavior, as demonstrated by our experimental results.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under NSFC No. 61033008, 61272145.

References

[1] J.M. Paul, D.E. Thomas, A. Bobrek, Scenario-oriented design for single-chip heterogeneous multiprocessors, IEEE Trans. VLSI Syst. 14 (8) (2006) 868–880, doi:10.1109/TVLSI.2006.878474.
[2] S.V. Gheorghita, M. Palkovic, J. Hamers, A. Vandecappelle, S. Mamagkakis, T. Basten, L. Eeckhout, H. Corporaal, F. Catthoor, F. Vandeputte, K.D. Bosschere, System-scenario-based design of dynamic embedded systems, ACM Trans. Des. Autom. Electr. Syst. 14 (1) (2009).
[3] G. Mariani, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, G. Palermo, C. Silvano, V. Zaccaria, An industrial design space exploration framework for supporting run-time resource management on multi-core systems, in: Proceedings of the DATE'10, 2010, pp. 196–201.
[4] C. Ykman-Couvreur, P. Avasare, G. Mariani, G. Palermo, C. Silvano, V. Zaccaria, Linking run-time resource management of embedded multi-core platforms with automated design-time exploration, Comput. Digital Techn., IET 5 (2) (2011) 123–135, doi:10.1049/iet-cdt.2010.0030.
[5] L. Schor, I. Bacivarov, D. Rai, H. Yang, S.-H. Kang, L. Thiele, Scenario-based design flow for mapping streaming applications onto on-chip many-core systems, in: Proceedings of the CASES'12, 2012, pp. 71–80, doi:10.1145/2380403.2380422.
[6] W. Quan, A.D. Pimentel, A scenario-based run-time task mapping algorithm for mpsocs, in: Proceedings of the 50th Annual Design Automation Conference, in: DAC '13, ACM, New York, NY, USA, 2013, pp. 131:1–131:6, doi:10.1145/2463209.2488895.
[7] W. Quan, A. Pimentel, An iterative multi-application mapping algorithm for heterogeneous mpsocs, in: Proceedings of the IEEE 11th Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia), 2013, 2013, pp. 115–124, doi:10.1109/ESTIMedia.2013.6704510.
[8] A.K. Singh, A. Kumar, T. Srikanthan, Accelerating throughput-aware runtime mapping for heterogeneous mpsocs, ACM Trans. Des. Autom. Electron. Syst. 18 (1) (2013) 9:1–9:29, doi:10.1145/2390191.2390200.
[9] W. Quan, A. Pimentel, A run-time self-adaptive resource allocation framework for mpsoc systems, in: Proceedings of the 22nd European Conference on Circuit Theory and Design (ECCTD '15), 2015.
[10] S. Manolache, P. Eles, Z. Peng, Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints, ACM Trans. Embed. Comput. Syst. 7 (2) (2008) 19:1–19:35, doi:10.1145/1331331.1331343.
[11] H. Javaid, S. Parameswaran, A design flow for application specific heterogeneous pipelined multiprocessor systems, in: Proceedings of the 46th Annual Design Automation Conference, in: DAC '09, ACM, New York, NY, USA, 2009, pp. 250–253, doi:10.1145/1629911.1629979.
[12] J. Castrillon, A. Tretter, R. Leupers, G. Ascheid, Communication-aware mapping of kpn applications onto heterogeneous mpsocs, in: Proceedings of the 49th Annual Design Automation Conference, in: DAC '12, ACM, New York, NY, USA, 2012, pp. 1266–1271, doi:10.1145/2228360.2228597.
[13] P. van Stralen, A.D. Pimentel, Scenario-based design space exploration of mpsocs, in: Proceedings of the IEEE ICCD'10, 2010, pp. 305–312.
[14] V. Nollet, P. Avasare, H. Eeckhaut, D. Verkest, H. Corporaal, Run-time management of a mpsoc containing fpga fabric tiles, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 16 (1) (2008) 24–33, doi:10.1109/TVLSI.2007.912097.
[15] H. Shojaei, A.-H. Ghamarian, T. Basten, M. Geilen, S. Stuijk, R. Hoes, A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for cmp run-time management, in: Proceedings of the 46th ACM/IEEE Design Automation Conference, 2009. DAC '09., 2009, pp. 917–922.
[16] W. Liu, M. Yuan, X. He, Z. Gu, X. Liu, Efficient sat-based mapping and scheduling of homogeneous synchronous dataflow graphs for throughput optimization, in: Real-Time Systems Symposium, 2008, pp. 492–504, doi:10.1109/RTSS.2008.49.
[17] A. Schranzhofer, J.-J. Chen, L. Thiele, Dynamic power-aware mapping of applications onto heterogeneous mpsoc platforms, IEEE Trans. Ind. Inf. 6 (4) (2010) 692–707, doi:10.1109/TII.2010.2062192.
[18] W. Quan, A. 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 (DSD), 2014, pp. 655–658, doi:10.1109/DSD.2014.46.
[19] R. Piscitelli, A.D. Pimentel, Design space pruning through hybrid analysis in system-level design space exploration, in: Proceedings of the International Conference on Design, Automation, and Test in Europe (DATE '12), 2012, pp. 781–786.
[20] R. Sarikaya, C. Isci, A. Buyuktosunoglu, Runtime application behavior prediction using a statistical metric model, IEEE Trans. Comput. 62 (3) (2013) 575–588, doi:10.1109/TC.2012.25.

[21] W. Quan, A. Pimentel, Towards self-adaptive mpsoc systems with adaptivity throttling, in: Proceedings of the 15th Int. Conference on Embedded Computer Systems: Architectures, MOdeling and Simulation (SAMOS '15), 2015.

[22] W. Quan, A.D. Pimentel, A system-level simulation framework for evaluating task migration in mpsocs, in: Proceedings of the 2014 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, in: CASES '14, ACM, New York, NY, USA, 2014, pp. 13:1–13:9, doi:10.1145/2656106.2656111.

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

[24] K. Lee, Y. Cha, J. Park, Short-term load forecasting using an artificial neural network, Power Syst., IEEE Trans. 7 (1) (1992) 124–132, doi:10.1109/59.141695.

[25] N. Doulamis, A. Doulamis, A. Panagakis, K. Dolkas, T. Varvarigou, E. Varvarigos, A combined fuzzy-neural network model for non-linear prediction of 3-d rendering workload in grid computing, IEEE Trans. Syst., Man, Cybern., Part B: Cybern. 34 (2) (2004) 1235–1247, doi:10.1109/TSMCB.2003.822282.

[26] A. Schranzhofer, J.-J. Chen, L. Thiele, Dynamic power-aware mapping of applications onto heterogeneous mpsoc platforms, IEEE Trans. Ind. Inf. 6 (4) (2010) 692–707, doi:10.1109/TII.2010.2062192.

[27] A. Hussien, A. Eltawil, R. Amin, J. Martin, Energy aware task mapping algorithm for heterogeneous mpsoc based architectures, in: Proceedings of the IEEE 29th International Conference on Computer Design (ICCD), 2011, 2011, pp. 449–450, doi:10.1109/ICCD.2011.6081444.

[28] C.-L. Chou, R. Marculescu, Run-time task allocation considering user behavior in embedded multiprocessor networks-on-chip, Trans. Comp.-Aided Des. Integ. Cir. Sys. 29 (1) (2010) 78–91, doi:10.1109/TCAD.2009.2034348.

[29] J. Huang, A. Raabe, C. Buckl, A. Knoll, A workflow for runtime adaptive task allocation on heterogeneous mpsocs, in: Design, Automation Test in Europe Conference Exhibition (DATE), 2011, 2011, pp. 1–6, doi:10.1109/DATE.2011.5763189.

Wei Quanis a Ph.D. candidate in the Institute of Informatics at the University of Amsterdam. His research interests include design space exploration, design and optimisation of multi-/many-core systems (in particular Multiprocessor System-on-Chip systems), computer architecture, embedded systems. He is a student member of the IEEE Computer Society.

Andy D. Pimentel is an associate professor in the Institute of Informatics at the University of Amsterdam. His research interests include system-level design of embedded systems, computer architecture modeling and simulation, performance analysis, design space exploration, and parallel computing. He received a PhD in computer science from the University of Amsterdam, and is a senior member of the IEEE.