Fitness Prediction Techniques for Scenario-Based Design Space Exploration

Peter van Stralen, *Student Member, IEEE* and Andy Pimentel, *Senior Member, IEEE,*

***Abstract*—Modern embedded systems are becoming increasingly multifunctional. The dynamism in multifunctional embedded systems manifests itself with more dynamic applications and the presence of multiple applications executing on a single embedded system. This dynamism in the application workload must be taken into account during the early system-level design space exploration (DSE) of multiprocessor system-on-a-chip (MPSoC)-based embedded systems. Scenario-based DSE utilizes the concept of application scenarios to search for optimal mappings of a multi-application workload onto an MPSoC. The scenario-based DSE uses a multi-objective genetic algorithm (GA) to identifying the mapping with the best average quality for all the application scenarios in the workload. In order to keep the exploration of the scenario-based DSE efficient, fitness prediction is used to obtain the quality of a mapping. This fitness prediction is performed using a representative subset of application scenarios that is obtained using co-exploration of the scenario subset space. In this paper, multiple fitness prediction techniques are presented: stochastic, deterministic, and a hybrid combination. Results show that, for our test cases, accurate fitness prediction is already provided for subsets containing only 1–4% of the application scenarios. Larger subsets will obtain a similar accuracy, but the DSE will require more time to identify promising mappings that meet the requirements of multifunctional embedded systems.**

***Index Terms*—Co-exploration, design space exploration, fitness prediction, subset selection.**

I. Introduction

THE design of modern embedded systems, especially those targeting the consumer market, is severely complicated by an increasing demand for versatile and multifunctional products that also have to meet stringent requirements in terms of performance, power consumption, and cost. In addition, these systems are typically dealing with an enormous time-to-market pressure. As a result, embedded systems are increasingly implemented using heterogeneous, software-centric multiprocessor system-on-a-chip (MPSoC) architectures. To cope with the design complexity of these systems, the concept of system-level design has been introduced, which raises the abstraction level of the design process [1]. Design space exploration (DSE) is a key ingredient of system-level design, during which a wide range of design choices are explored, especially during the early design stages. Such early DSE is of paramount importance, as early design choices heavily influence the success or failure of the final product.

Manuscript received May 29, 2012; revised October 15, 2012, January 10, 2013; accepted February 18, 2013. Date of current version July 15, 2013. This paper was recommended by Associate Editor P. R. Panda.

The authors are with the University of Amsterdam, Amsterdam 1098XH, The Netherlands (e-mail: p.vanstralen@uva.nl; a.d.pimentel@uva.nl).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2013.2252711

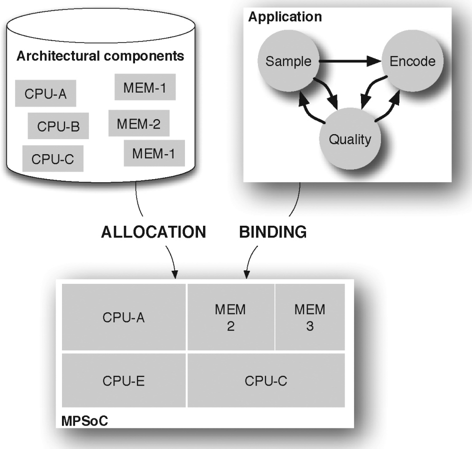

Fig. 1. Mapping involves two aspects: allocation (selection of the architectural components) and binding (assigning the application task to allocated components).

An important element of system-level DSE is the search for an optimal mapping of the application workload onto the underlying MPSoC platform architecture. Here, the mapping involves two aspects (Fig. 1): 1) allocation, and 2) binding. Allocation selects the architectural components for the MPSoC platform architecture (i.e., not all platform components need to be used). Subsequently, the binding specifies which application task or application communication is performed by which MPSoC component. The number of possible mappings is enormous, especially if there are multiple applications in the workload of the embedded system.

A significant amount of research has been performed on system-level DSE for MPSoCs [2] during the last two decades. The majority of this paper is focused on the analysis of MPSoC architectures under a single, static application workload. However, current application workloads executing on embedded systems are becoming more and more dynamic. This dynamic behavior can be classified and captured using so-called workload scenarios [3]. Workload scenarios make a distinction between two aspects. First, intra-application scenarios describe the dynamic behavior within applications (i.e., application modes). For example, a QoS mechanism within a video decoder application may dynamically lower the bitrate to meet its deadlines or to save power. Second, inter-application scenarios describe the interaction between different applications that are concurrently executing on an embedded system and contending for its system resources.

In order to capture the dynamic behavior of multi-application workloads in system-level design we have introduced scenario-based DSE [4]. An important problem that needs to be solved by such scenario-based DSE is the rapid evaluation of mappings during the search through the MPSoC design space. The number of potential interactions between different applications grows exponentially with the number of applications and application modes that can be simultaneously executed in the embedded system. As a consequence, the potential number of different application scenarios can be huge. Therefore, it is infeasible to rapidly evaluate mappings during the process of early DSE by exhaustively analyzing (e.g., via simulation) all possible workload scenarios. As a solution, fitness prediction [5] can be used to quickly obtain an approximated fitness value. In scenario-based DSE, a representative subset of application scenarios is used to predict the fitness of a mapping. If mapping m_1 is better than mapping m_2, a representative subset should be able to give a better predicted fitness to mapping m_1 that it assigns to mapping m_2. The difficulty, however, is that the representativeness of a subset of application scenarios is dependent on the target MPSoC architecture. As the evaluated MPSoC architectures are not fixed during the process of DSE, we need to simultaneously co-explore the MPSoC design space and the application scenario space to find representative subsets of application scenarios for those mappings that need to be evaluated.

Earlier work already introduced two different approaches for subset selection: a genetic algorithm [4] and a feature selection approach [6]. The contribution of this paper is a more extensive description of scenario-based DSE with additional experiments. Moreover, it will provide a new hybrid fitness prediction method. The hybrid method combines the strengths of both the genetic algorithm and the feature selection.

The rest of this paper is organized as follows. In the next section, we quickly provide an overview of scenario-based DSE. Sections 3 and 4 will describe the components of our scenario-based DSE framework in more detail. First, Section 3 describes the design explorer, which accounts for the MPSoC mapping DSE using a representative subset of application scenarios to evaluate mappings. Subsequently, Section 4 provides an overview of the subset selector that takes care of (dynamically) finding the representative subset of application scenarios during the process of DSE. Section 5 presents the experiments in which we compare the different fitness prediction techniques. In Section 6, we describe related work, while Section 7 provides a conclusion.

II. Scenario-Based DSE

Scenario-based DSE rapidly evaluates mappings for multi-application workloads during the search through the MPSoC design space. For this rapid evaluation, a co-exploration is performed of the MPSoC design space and the application scenario space. The first part of this section describes the high-level MPSoC simulation framework Sesame [7], which is used for the evaluation of mappings in our scenario-based DSE framework. Sesame is a high-level framework for quickly obtaining an estimate of nonfunctional metrics (such as execution time and energy consumption) for a mapping, allowing for separating the high-quality from the low-quality mappings. As Sesame is part of the Daedalus design methodology [8], the resulting mappings can directly be synthesized on FPGA for validation and calibrational purposes.

Subsequently, the second part of this section describes the complete exploration framework that efficiently deploys workload scenarios [3] during the process of early DSE. This framework explicitly takes the existence of multiple applications into account. If multiple applications are not explicitly taken into account during the DSE, there are two options for designing an embedded system with multiple applications: 1) isolate all the applications on the architecture, and 2) design for the case where all applications are active. The first approach completely disregards resource sharing, whereas the second approach is quite unrealistic. In both cases, more resources are used than necessary, resulting in an overdesigned system.

A. Scenario-Aware Sesame

The design flow of Sesame is depicted in Fig. 2A. It provides fast performance evaluation using a separation-of-concerns based approach with a typical accuracy of $\pm 5\%$ with respect to the real implementation [8].

Sesame is scenario aware [9], allowing it to simulate workload scenarios [3]. The workload scenarios are illustrated in Fig. 2(b). In this paper, we distinguish two types of workload scenarios: intra-application and inter-application scenarios. Intra-application scenarios describe the different behaviors, or operation modes, within an application. Fig. 2(b) shows the intra-application scenarios for three example applications. For example, the GSM application can be used in sending and receiving mode, or the MP3 decoder can play music in mono or stereo sound.

An inter-application scenario, on the other hand, describes the behavior of multiple applications. That is, an inter-application scenario specifies which applications can run concurrently. For example, the inter-application scenario in Fig. 2(b) describes the situation in which the MP3 and video applications are running concurrently while the GSM is inactive.

To fully describe what a system is doing, a complete application scenario bundles the possible intra-application scenarios of all the active applications. The set of active applications is described using an inter-application scenario, whereas each intra-application scenario specifies a particular operation mode of an individual application. The example application scenario in Fig. 2(b) specifies that the MP3 application is playing music in mono sound, while a simple profile is used to decode video. It is possible for application scenarios to overlap in their execution. During this overlap, however, the inter-application scenario constraints must always be kept. Therefore, for the example in Fig. 2(b) it can be the case that while one video frame is decoded, multiple MP3 frames are handled. However, given the inter-application scenario, it can never be the case that the GSM application starts together with the MP3 or the video application.

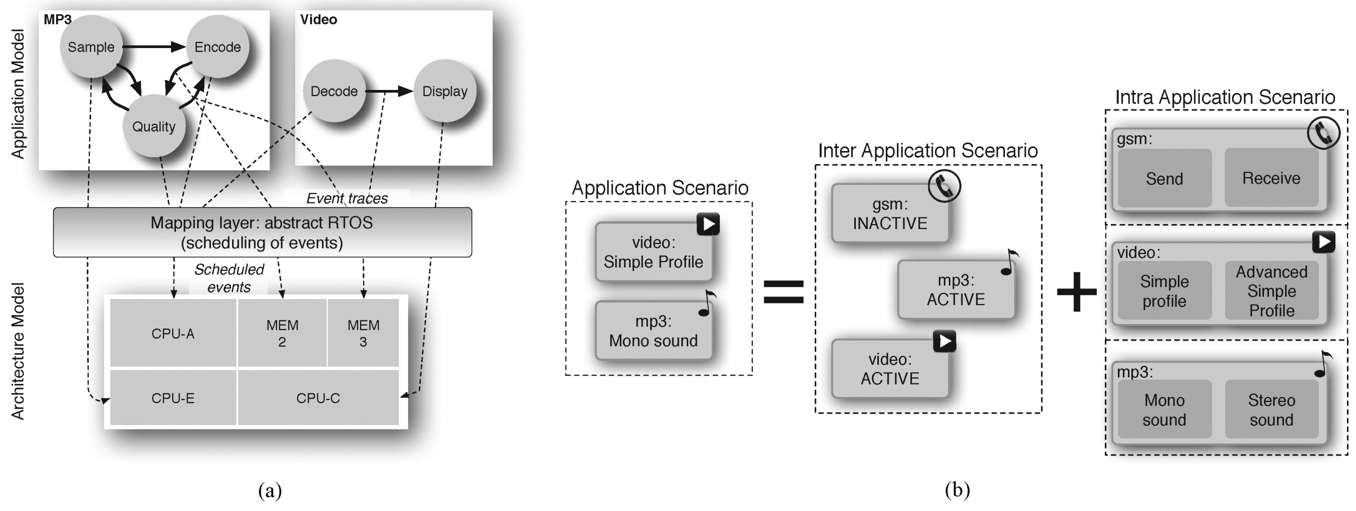

Fig. 2. High level scenario-based MPSoC simulation. (a) Sesame. (b) Application scenarios.

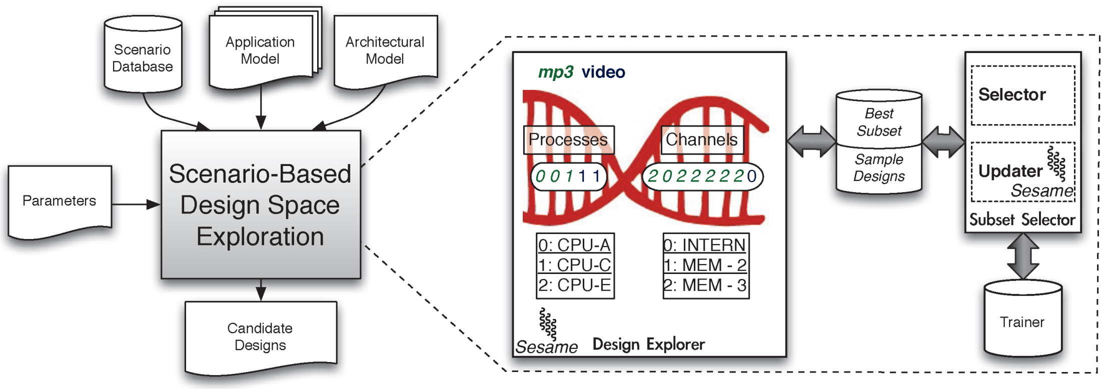

Fig. 3. Exploration framework for scenario-based DSE. The GA chromosome within the design explorer is shown for our MP3-video example.

The complete set of workload scenarios is kept in the scenario database. The scenario database is identified using a profiling method. More details can be found in [9].

B. The Exploration Framework

The objective of the exploration framework is to provide a static mapping of the multi-application workload onto the MPSoC. This static mapping is used throughout the system's entire lifetime. Therefore, the average behavior of the system for all different possible application workload scenarios must be as good as possible. In this paper, we assume an equal likelihood of each intra-application scenario. However, it is perfectly possible to use a different probability distribution.

Despite the fact that Sesame provides fast MPSoC simulation, it is infeasible to exhaustively simulate all the possible application scenarios for each individual mapping. This is why a representative subset of scenarios is used to estimate the quality of each mapping. Fig. 3 shows the exploration framework that is capable of performing such scenario-based DSE. The general flow of scenario-based DSE is given on the left side of Fig. 3, whereas the right side zooms in on the actual scenario-based DSE part.

The scenario-based DSE explores the mapping of multi-application workloads onto an MPSoC platform. To this end, a couple of inputs need to be made explicit. Not only the architectural model needs to be given, but also the multi-application workload.

The architectural model describes the complete set of available architectural components (including the available interconnections). This architecture typically will not fit on the final MPSoC. Therefore, the DSE will reduce this architecture by only using a subset of the architectural resources. An example architectural model can be seen in Fig. 2A. The mapping chromosome in Fig. 3 (shown in the design explorer) allocates the components CPU-A, CPU-C, and MEM-3. The component INTERN is a virtual component that specifies that the internal memory of a CPU is used.

From the multi-application workload two characteristics are required. First, a model that describes the structure of the applications. Secondly, the possible workload behavior is described explicitly using a scenario database [9] in which the inter- and intra-application scenarios are stored and made explicit. Whereas the input application models specify the structure of each individual (concurrent) application enabling mapping exploration of the application tasks, the scenario database characterizes the different possibilities for multi-application workload behavior (e.g., which applications or application modes are active at the same time).

A representative subset of application scenarios from the scenario database is used to rapidly evaluate mappings during MPSoC DSE. As the quality of the fitness prediction of a subset is dependent on the mappings that are under evaluation, we have already pointed out that the MPSoC design space needs to be simultaneously co-explored with the application scenario space. This co-exploration is illustrated in the right-hand side of Fig. 3, which zooms in on our scenario-based DSE. One of the two key elements, called the design explorer, is similar to classical MPSoC DSE as it uses a metaheuristic, which is in our case a genetic algorithm (GA), to search for optimal mappings. The second key element, referred to as the subset selector, tries to identify the best representative subset of application scenarios that is used to evaluate the mappings in the design explorer. The subset selector can be implemented with multiple techniques. Both processes are running asynchronously, and communication between them is performed via shared memory. In the next two sections, we provide a detailed description of the design explorer and subset selector of our scenario-based DSE.

III. The Design Explorer

In this section, a description is given of how the design explorer identifies optimal mappings. The first subsection specifies the Sesame system model. Next, the mapping procedure is described.

A. System Model

Fig. 2A shows the Sesame system model. There are three conceptual layers in Sesame: 1) the application model; 2) the mapping layer; and 3) the architecture model. The application model describes each individual application using a Kahn process network (KPN) [10], whereas the architecture model describes the MPSoC platform architecture in a cycle-approximate fashion. Furthermore, the mapping layer explicitly maps the tasks and communications from the application model(s) onto the components in the architecture model. This is implemented using a trace-driven co-simulation of the application and the architecture model[7].

1) *Application Layer:* Applications are represented by a directed graph $G_K(V, E_k)$. Vertices represent the Kahn process nodes. Directed edges $E_k = V \times V$ represent FIFO communications links to pass messages between process nodes.

2) *Architecture Layer:* The architecture is described by the undirected graph $G_R(R, E_R)$. R represents architectural resources, such as processors, communication buses, crossbars, FIFOs, and memories. There are three types of architectural elements. Architectural processors $R_P \subset R$ are the elements capable of running processes. $R_B \subset R$ are FIFO buffers used for communication. In case, two communication processes are mapped onto the same processor, the communication could potentially be done internally. Internal communication of a processor $p \in R_P$ is modeled by connecting a buffer $b \in R_B$ from and to the processor $((p, b), (b, p) \in E_R)$.

All remaining architectural resources are only for interconnecting purposes. The edges in $E_R = R \times R$ describe the communication links in the architecture.

3) *Mapping Layer:* Computation mapping edges E_X assign architectural resources to the Kahn process nodes. More precisely, the edge $(v, p) \in E_X$ assigns Kahn process $v \in V$ to processor $p \in R_P$. A Kahn process can only be mapped onto a processing element that is feasible of running the task:

$$(v, p) \in E_X \iff \text{Feasible}(p, v).$$

An example of a processing element that supports only a limited number of application tasks is an application specific integrated circuit (ASIC).

Next, communication mapping edges (E_C) bind FIFO channels. An edge (c, b) assigns the channel $c \in E_K$ to a buffer $b \in R_B$ in the architecture.

B. Mapping Procedure

The mapping procedure maps the application onto the architecture. This is split into two steps: allocation and binding. Allocation reduces the resource-usage of the architecture. Next, the binding maps all processes and channels onto the architecture. The procedure is as follows:

1) *Allocation:* The first step during mapping is allocation α, where $\alpha_P = R_P \cap \alpha$ and $\alpha_B = R_B \cap \alpha$. The allocation α contains a subset of architectural resources such that $\alpha \subseteq R$:

$$\left(\sum_{r\in\alpha} \text{area}(r)\right) < \text{MAX\_AREA}.$$

Given an allocation α, a set of potential communication paths $\psi = (\alpha_P \times \alpha_B \times \alpha_P)$ can be defined:

$$\psi = \{(p_1, b, p_2) : \text{PATH}_\alpha(p_1, b) \wedge \text{PATH}_\alpha(p_2, b)\}.$$

ψ is the set of paths (p_1, b, p_2) such that (:) there is a path $p_1 \to b$ and a path $p_2 \to b$. The function PATH determines if there is a path between two resources using a subset of the allocated resources α

$$\begin{aligned}\text{PATH}_\alpha(r, b) :=& (r, b) \in E_R \vee \\ & \exists r_i \in \alpha | (r, r_i) \in E_R \wedge \text{PATH}_\alpha(r_i, b).\end{aligned}$$

An allocation is only valid if each combination of processors have at least one buffer to communicate:

$$\forall p_1, p_2 \in \alpha_P : \exists (p_1, b, p_2) \in \psi.$$

2) *Binding:* There are two steps during binding. The first step is computational binding and the second step is the communicational binding. The computational binding β_X maps the processes onto processors such that $\beta_X \in E_X$:

$$\forall v \in V : |\{p : (v, p) \in \beta_X \wedge p \in \alpha_P\}| = 1.$$

Each process must be mapped on exactly one processor within the set of allocated components α_P. The communicational binding β_C is done after the computational binding. It maps the communication channels onto architectural buffers such that $\beta_C \in E_C$:

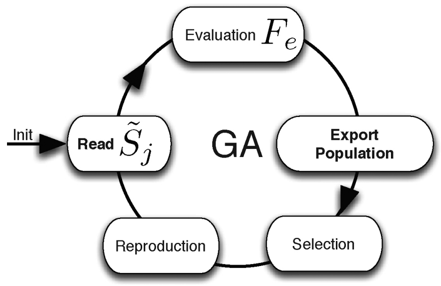

Fig. 4. GA to find the optimal mapping extended with steps to communicate data with the subset selector.

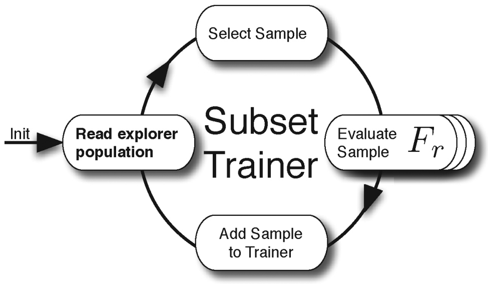

Fig. 5. Update procedure of the training set.

$$\begin{aligned}\forall(v_1, v_2), b \in \beta_C :&(v_1, p_1) \in \beta_X \wedge (v_2, p_2) \in \beta_X \\ &\wedge (p_1, b, p_2) \in \psi \\ \forall c \in E_k :&|\{b : (c, b) \in \beta_C\}| = 1.\end{aligned}$$

Each mapped channel c must be mapped onto a buffer b. Moreover, there must be a communication path between the processors and the buffer.

A mapping m is the combination of an allocation α, and the bindings β_X and β_C. It is only valid if all the preceding constraints are fulfilled.

C. Genetic Algorithm

To actually perform the search for good mappings, a NSGA-II [11] based multi-objective GA is used. NSGA-II is an elitist selection algorithm that uses nondominated sorting to select the offspring individuals. Nondominated sorting follows the approach of Goldberg's Pareto ranking [12] and is illustrated in Fig. 6. All the nondominated individuals get rank one and are removed from the set of mappings. The nondominated individuals in the remaining set get the second rank and this is repeated until all individuals are ranked. In this way, each of the individuals get their dominance depth [13].

Let S be the total set of scenarios and $\tilde{S}_j$ be the representative subset at time step j. Our objectives for a mapping m are defined as follows:

$$F(m) = \sum_{s\in S} \frac{(\text{perf}(m), \text{energy}(m), \text{cost(m)})}{|S|}$$
$$\tilde{F}_{\tilde{S}_j}(m) = \sum_{s\in \tilde{S}_j} \frac{(\text{perf}(m), \text{energy}(m), \text{cost(m)})}{|\tilde{S}_j|}$$

In this case, the performance and energy are obtained by invoking Sesame [7], whereas cost is defined as the sum of the individual costs of the elements in allocation α. There is a difference between the real fitness F and the estimated fitness $\tilde{F}_{\tilde{S}_j}$. The real fitness uses all possible scenarios to determine the objectives. Therefore, this fitness is independent of the current generation. The estimated fitness $\tilde{F}_{\tilde{S}_j}$, however, is only valid during generation j as in the next generation the most representative scenario subset $\tilde{S}_{j+1}$, which is used to estimate the fitness may be changed.

Fig. 4 shows the extended algorithm of the GA that couples the design explorer and the subset selector. Before the design explorer can perform any evaluation, the currently most representative scenario subset $\tilde{S}_j$ must be retrieved. Using the obtained subset of application scenarios $\tilde{S}_j$, the design explorer can quickly evaluate the fitness $\tilde{F}_{\tilde{S}_j}$ for all mappings in the current population. As a result, mapping individuals in the parent population may need to be partially reevaluated. Afterwards, the current population is exported to the subset selector. Next, the GA can select individuals based on their estimated fitness $\tilde{F}_{\tilde{S}_j}$. In case, the scenario subset S_j is representative, this means that the decisions made by the NSGA-II selector are similar to the case where the real fitness F would have been used. The selected individuals can be used for reproduction. The reproduction procedure creates a new population of individuals that can be used in the next generation.

IV. The Subset Selector

The subset selector is responsible for obtaining a representative subset of scenarios $\tilde{S}_j$ to predict the fitness of mappings in the design explorer. Due to the potentially large number of scenarios, this selection is not trivial. Ideally, this selection is done statically, before the MPSoC DSE process starts. However, the problem is that mappings need to be available to select the best subset of scenarios. Unfortunately, these mappings are unavailable until the DSE process is running. On top of this, the type of mappings are also changing during the DSE search.

Therefore, the subset selection needs to be performed dynamically using a training set T_i of application mappings m. The training set T_i is dynamically updated during the complete process of scenario-based DSE. Logically, the subset selector is split into two threads: the selector thread and the updater thread. The selector thread is responsible for selecting the representative subset of scenarios, whereas the updater thread is only concerned with updating the training set.

In the remainder of this section, we provide a detailed description of the various elements of the subset selector. First, we will look into the updater thread and describe the update procedure of the training set. Next, the metric that is used to judge the quality of a representative subset will be described. The final subsection subsequently describes how the selector thread obtains the best representative subset of scenarios.

A. Training Set Update Procedure

During the search for the representative subset of application scenarios, it is crucial to have a set of training mappings T_i available to evaluate a certain subset of scenarios. The

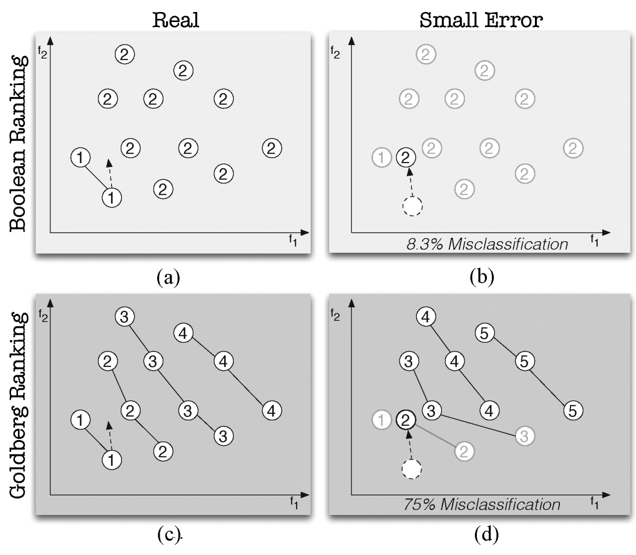

Fig. 6. Set of Pareto fronts showing the effect of a small misprediction (as shown with the dashed arrow) on the misclassification rate using the Boolean (A + B) and the Goldberg (C + D) ranking scheme. In A + C the real fitness is used, whereas B + D use a predicted fitness.

real fitness of this set of mappings must be known. That is, each of these mappings should have been evaluated using all applications scenarios stored in the scenario database. Hence, the training set of mappings must be small, but still correctly represent the population of mappings that is judged in the design explorer. Fig. 5 illustrates the training set update procedure from set T_i to set T_{i+1}. The steps are as follows:

1) *Read Explorer Population:* To keep the training set T_i up to date with the mapping population in the design explorer, the current design explorer population g_j is imported. The list of candidate mappings C_{i+1} is updated as follows:

$$\begin{aligned}\underset{C_{i+1}}{\text{maximize}} \quad & \sum_{m\in C_{i+1}} \text{last\_gen}(m)\\ \text{subject to } & (1)\ C_{i+1} \subseteq C_i \cup g_j\\ & (2)\ C_{i+1} \cap T_i = \emptyset\\ & (3)\ |C_{i+1}| = min(|C_i|, \text{C\_SIZE})\end{aligned}$$

Condition 1 states that g_j is added to the set of candidates C_{i+1}. Condition 2 takes care that none of the candidates are already part of the trainer. Finally, the maximization criteria and condition 3 take care that the number of candidates is limited to C_SIZE. In case of truncation, the oldest mappings are removed (the function last_gen(m) returns the last generation that mapping m was part of the population).

2) *Select Mappings:* The next step is to select k new candidate mappings M_c for the trainer T_{i+1}:

$$\begin{aligned}\underset{M_c}{\text{minimize}} \quad & \sum_{m\in M_c} \text{dist}(\tilde{P}_j, \tilde{F}_{\tilde{S}_j}(m))\\ \text{subject to } & (1)\ M_c \subseteq C_{i+1}\\ & (2)\ |M_c| = k\end{aligned}$$

In order to select the interesting mappings, the candidate mappings are ordered based on their Euclidean distance dist(...) to the closest mapping on the estimated Pareto front $\tilde{P}_j$ of generation j. As the design explorer needs to correctly identify the real Pareto front, these are the most interesting mappings. The closer the estimated fitness to an estimated nondominated Pareto point, the higher the probability that the mapping is part of the real Pareto front.

3) *Evaluate Real Fitness:* Once the mappings are selected, a separate pool of Sesame worker threads is used to evaluate all the selected mappings.

4) *Add Mappings to Trainer:* At this point, the fitness of each of the selected mappings is exactly known. Therefore, trainer T_{i+1} can be generated. T_{i+1}:

$$\begin{aligned}\underset{T_{i+1}}{\text{minimize}} \quad & \sum_{m\in T_{i+1}} \text{dist}(P, F(m))\\ \text{subject to } & (1)\ T_{i+1} \subseteq T_i \cup M_c\\ & (2)\ |T_{i+1}| = \min(|T_i \cup M_c|, \text{T\_SIZE})\end{aligned}$$

The candidate mapping are first added to T_i (condition 1), after which the trainer is truncated to a maximum size T_SIZE (condition 2). During truncation the mappings with the highest Euclidean distance from the real Pareto front P are removed.

B. Subset Quality Metric

With the training set in place, the metric of judging a subset of application scenarios needs to be defined. To be independent from the number of optimization objectives, the misclassification rate of the Pareto ranking of the individual training mappings is used to judge the quality of the representative subset.

There are several approaches for Pareto ranking [14]. In this paper, we focus on two: Boolean ranking and Goldberg ranking. Both ranking schemes are visualized in Fig. 6. Goldberg's ranking approach uses the dominance depth of individuals as explained in Section III-C. Boolean ranking follows a more simplistic approach: if the solution is nondominated the rank is one and otherwise the rank is two.

Since the NSGA-II selector in the design explorer uses Goldberg's ranking, it may be obvious to use Goldberg's ranking to judge the quality of the representative subset. However, the computation complexity of obtaining Goldberg's ranking is rather high, $O(MN^2)$, where M is the number of objectives and N the population size, while the misclassification rate on the ranking may also be somewhat deceiving. An example is given in Fig. 6. In this example, the fitness of one of the individuals is slightly off. As a consequence, it becomes dominated by the left most individual and in both ranking schemes its rank becomes two. Additionally, in the Goldberg's ranking all the individuals that are dominated by this individual also increase in Pareto rank. This domino effect results in a 75% of misclassified Pareto ranks, whereas the Boolean ranking only has a misclassification rate of 8.3% (the individual with the wrong fitness). It turns out that it is complicated to correctly classify the lowly ranked individuals. Therefore, we have chosen to use Boolean ranking. The elitist selection algorithm only needs to be capable of identifying the Pareto front. The order of the rest of the individuals is irrelevant.

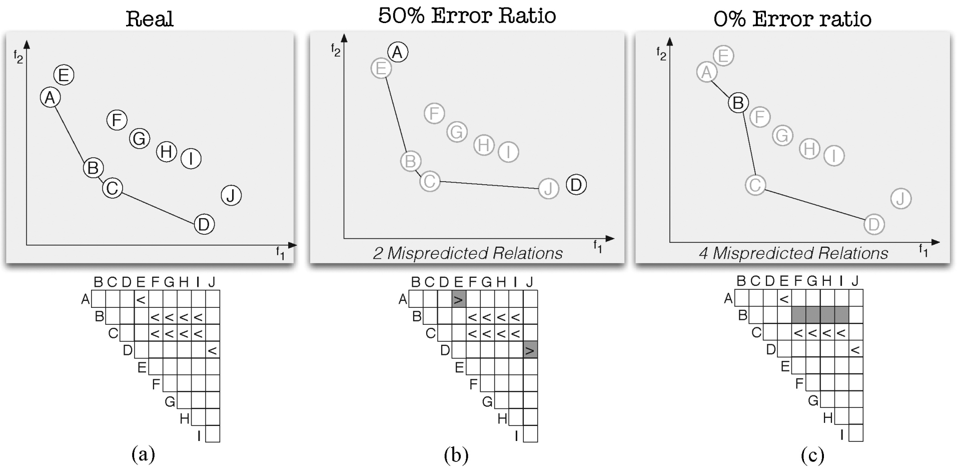

Fig. 7. Scenario subset with a larger number of misclassified relations do not necessarily result in a worse Pareto front. Pareto front B has only two mispredicted relations (mapping [A, E] and mapping [D, J]), but only two of four Pareto points (mapping B and C) match the real Pareto front A. Pareto front C may have four mispredicted relations, but all of its identified Pareto points are correct. Therefore, this metric is subordinate to misclassification.

During the subset evaluation two functions are used:

$$\mathrm{rel}_{F'}(m_1, m_2) := \begin{cases} 1 & F'(m_1) \text{ dominates } F'(m_2) \\ -1 & F'(m_2) \text{ dominates } F'(m_1) \\ 0 & \textit{else} \end{cases}$$

$$\mathrm{rank}_{F'}(m, T) := \begin{cases} 2 & \exists m' \in T \left(\mathrm{rel}_{F'}(m', m) = 1\right) \\ 1 & \textit{else} \end{cases}$$

The function $\mathrm{rel}_{F'}(m_1, m_2)$ determines the Pareto dominance relation between mapping m_1 and m_2 given the fitness function F'. This fitness function can be equal to F (and thus provide the real fitness) or equal to $\tilde{F}_{\tilde{S}}$. In this case, the scenario subset $\tilde{S}$ is used to estimate the fitness. Next, the function $\mathrm{rank}_{F'}(m_1, T)$ provides the ranking of a mapping given trainer T.

The misclassification rate of a scenario subset $\tilde{S}$ is the ratio of ranks that are estimated incorrectly:

$$\mathrm{r_{rank}}(\tilde{S}, T) := \frac{|\{m \in T : \mathrm{rank}_F(m, T) \neq \mathrm{rank}_{\tilde{F}_{\tilde{S}}}(m, T)\}|}{|T|}$$

A second and subordinate metric is the number of misclassified relations:

$$\mathrm{r_{rel}}(\tilde{S}, T) := \frac{|\{m_1, m_2 \in T : \mathrm{rel}_F(m_1, m_2) \neq \mathrm{rel}_{\tilde{F}_{\tilde{S}}}(m_1, m_2)\}|}{|T|^2}$$

By definition, when there are no misclassified relations, the misclassification rate is also zero. However, less misclassified relations does not imply that the Pareto ranking is better. This is mainly due to the fact that the number of misclassified relations is only loosely coupled to the ranking of the individual mappings. A clear example can be seen in Fig. 7. Compared to the Pareto front based on the real fitness (7A), the estimation in Fig. 7C is the best. The total Pareto front is predicted correctly. However, it has more misclassified relations than the situation in Fig. 7B, where 50 percent of the estimated Pareto front is incorrect (E and J).

C. Subset Selection

The final part of this section discusses the core part for selecting the representative subset: the selector thread. The selector thread dynamically searches for the best representative subset:

$$\underset{\tilde{S}}{\text{minimize}}\ \mathrm{r_{rank}}(\tilde{S}, T) : \underset{\tilde{S}}{\text{minimize}}\ \mathrm{r_{rel}}(\tilde{S}, T)$$

The main objective to optimize on is the misclassification rate ($\mathrm{r_{rank}}$). In case, two or more scenario subsets have the same misclassification rate, the number of misclassified relations ($\mathrm{r_{rank}}$) will determine which is the best.

Every time the selector thread finds a new best subset, it immediately updates the scenario subset that is used in the design explorer. The subset may be of any size, as long as it does not exceed the user-defined maximum size. In earlier work [6], we have used a fixed size. This requirement is however illogical: if there is a smaller subset of a better quality, it has only benefits as the same quality is obtained using fewer evaluations. Initially, the subset(s) in the population of the subset selector are generated randomly. The selector thread will start once the updater thread has added the first mappings to the trainer.

A random pick of application scenarios does not result in a representative subset (see [4]). Therefore, the representative subset must be searched during the DSE. In earlier work, we have introduced two approaches for the search of the representative subset: a genetic algorithm [4] and a feature selection algorithm [6]. In this paper, we add a third approach for comparison purposes. It is a hybrid method that makes use of both methods. Below, we describe each of the methods.

1) *Genetic Algorithm (GA):* In the first approach, the subset is searched in a straightforward fashion. Similar to the design explorer, a GA is used to identify the best representative subset. A pool of individual subsets follows an evolutionary process with crossover and mutation to obtain a subset of scenarios that is as good as possible.

The benefits of this approach are: 1) it is fast; 2) the search can quickly move through the total space of possible subsets;

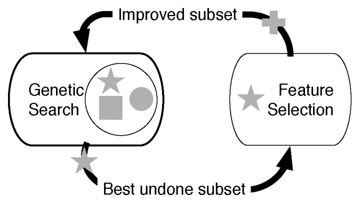

Fig. 8. Hybrid approach for subset selection. The approach alternates between two subset selection techniques: genetic search and feature selection. On convergence the GA will send the best undone subset to the FS. The FS will improve the subset until the search converges or the local search is done (in this case the subset will no longer be sent to the FS).

and 3) local optima can easily be circumvented. The downside, however, is the coarse grained nature of the search. If an individual is found that is close the optimal solution, it is quite likely that mutation or crossover moves away from it. As a consequence, it takes longer to identify the optimal subset.

2) *Feature Selection (FS):* A more intelligent approach to guide the search can be achieved using feature subset selection. Feature subset selection tries to find a subset of features (application scenarios) to classify (obtain the Pareto ranks) individuals as good as possible. In our case, we decided to use dynamic sequential oscillation search [15] as our feature subset selection algorithm. The benefit of this algorithm is that it improves a subset instead of constructing one. Moreover, at any given time it can be halted to return its currently best representative subset. This algorithm uses a kind of hill-climbing technique to improve the subset of scenarios. It will oscillate the size of the subset by adding or removing the most optimal scenario (e.g., the remaining subset is as good as possible). The margin of oscillation is limited to a fixed number [1]. For a detailed description of the procedure of the feature selection technique, we refer to [15].

As the search method of FS is more directed, in time it will only move closer to the optimal subset. In contrast to a GA, it will not move away from the optimal subset. This strength is also its weakness. As it not moves away from the local optimum, it is more sensitive to finding local optima.

3) *Hybrid Approach (HYB):* To combine the strengths of both approaches, we introduce a hybrid method. Fig. 8 visualizes this approach. The GA can quickly prune the complete space of possible subsets, whereas the feature selection is good at searching thoroughly in a small part of the total space of the possible subsets. The hybrid method alternates between the GA and the FS. Starting with the GA approach, it will switch between the methods once the search is converged (no improvement in a given amount of time).

The feature selection that is applied in the hybrid approach is not just a custom variation operator. In contrast to a variation operator, it keeps state over the different invocations. The same subset can be sent to the feature selection more than once, given that it is undone. For an undone subset, the oscillation search has not exceeded the predefined maximal margin yet. An undone subset that has been sent earlier will continue the oscillating search at the margin, where it stopped during the previous invocation.

[1]In the experiments we have used 20 as a margin

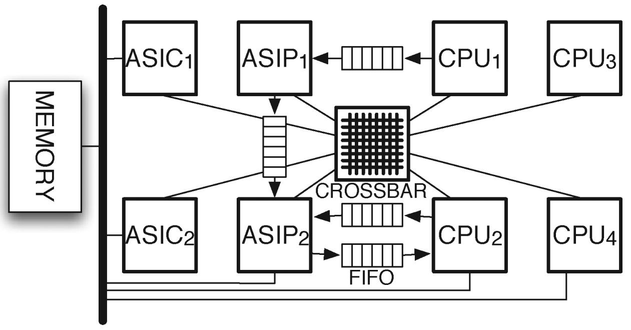

Fig. 9. MPSoC architecture components on which the application can be mapped.

D. Final Pareto Front

The outcome of the DSE is a final set of candidate mappings. This set of mappings is taken from the trainer within the subset selector. Most importantly, the real fitness of these mappings is known. Additionally, the selection procedure of the trainer ensures that the nondominated solutions that are encountered will be kept in the list during the complete design space exploration.

V. Experiments

In order to verify the scenario-based DSE, a couple of experiments have been performed. For these experiments, both the multi-application workload and the potential set of architectural components remain fixed. The multi-application workload is generated stochastically with a Python tool based on [16] in such a way that the behavior of ten embedded applications is resembled. During the stochastic generation, a set of constraints is supplied such that different types of intra application scenarios are generated (computation/communication intensive). Additionally, the number of simultaneously active applications is fixed. The ten applications have a total of 58 processes and 75 communication channels. The multi-application workload consists of 4607 different application scenarios. We have chosen to use stochastic applications as they provide a wider range of possibilities than real applications do. They can easily be instrumented to mimic real applications, but they also allow for small changes on parameters, such as communication fraction, for a thorough study of the properties of the DSE process.

The platform architecture is visualized in Fig. 9. It contains four different processors (CPU), which are able to execute all the application processes. The ASIPs are limited to the execution of 11 out of the 58 processes, whereas the ASICs can only run four out of the 58 processes. For communication, all the processing elements are connected to a crossbar. The crossbar contains buffers to temporarily store the messages[17]. Additionally, a fraction of the processors are also connected to a shared memory via a bus. There are four dedicated FIFO communication channels. As each of the components has a specific cost (in terms of area), it is not necessarily the case that all components are used in a mapping (i.e., not all components need to be allocated).

Therefore, one of the objectives during DSE is the cost of the mapping (the sum of the chip area of the used components). Next to that, execution time and energy are also objectives during the optimization of the mapping of the multi-application workload onto the architecture. For energy estimation, a simple activity-based power model is used. Each component has two power numbers: idle power and busy power. After simulation, the active time of each component is known and the energy usage can be calculated. The result of the DSE will be a Pareto front that shows the trade-off between cost, time and energy.

All the experiments have been performed on a dedicated node of the LISA cluster [18] that is part of the Dutch SARA Computing and Networking Services. The selected nodes contain two Intel quad-core Xeon L5520 processors with eight cores running at 2.26 Ghz. As job management is present on the LISA cluster, no other processes are running on the processor. Therefore, the real exploration time of the DSE can be used for comparison in the experimental results.

Fig. 10 shows an example multi-objective Pareto front that results from our scenario-based DSE. A part of the non-dominated solutions is annotated with the allocated resources. Clearly, visible is the cheap single core solution: low energy, low cost, but a high execution time. Adding more processors improves the execution time. The fastest solution uses all the available resources and communication structures (M is shared memory, X is crossbar). Interesting to notice is the difference between 2CPU+X (cost 80) and 2ASIC+2CPU+X. Adding two ASICs results in an increase in execution time (due to communication), but a lower power usage. However, the 2CPU+X may use more chip area (cost 120), its CPU is much faster and more power efficient.

In the following subsections, a couple of experiments will be discussed. For these experiments we will focus on two aspects: 1) scenario subset selection, and 2) DSE results. For data visualization and discussion, we have chosen not to focus on the quality of the complete Pareto front, but to look for mappings that minimize execution time given a certain maximal architectural cost.

The first subsection will compare the different subset selection techniques in isolation. In this experiment, the subset selector is decoupled from the design explorer and fed with a static set of training mappings. Next, the effect of the subset size during a DSE is investigated. The third subsection describes the quality of the identified subsets during a real DSE, where the training set is dynamic. Finally, the last subsection will compare the required effort to identify satisfying mappings with the different subset selection techniques.

A. Subset Selection

The first experiment focuses on the scenario subset selection procedure. Using two different training populations, the ability to identify the subset with the optimal ranking (based on cost, time, and energy) is identified for each of the procedures. For this purpose, the subset selector is run in isolation using a fixed training set. Two different training sets are used: the first training set consists of 518 mapping individuals, whereas the second training set has 498 mapping individuals. The maximal subset size is varied from 20 to 200 (0.4–4.3% of the total number of scenarios). A subset of 0.4% is the smallest

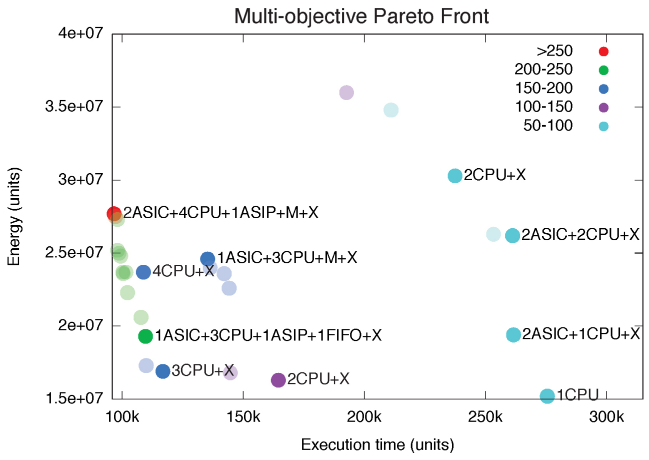

Fig. 10. Pareto front of the stochastical multi-application workload. Three objectives are used: time (x-axis), energy (y-axis) and cost (point type). The nontransparent points are annotated by the allocated resources.

subset size, where an optimal subset is available that is able to predict the ranking of both of the trainers perfectly. The subset of 4.3%, on the other hand, is large enough for a quick identification of an optimal subset. For each of the 180 different runs, the seed and subset size is chosen randomly. A subset selector approach will be run until the optimal subset is found. The search time is limited to 15 min of wall-clock time. After this period, the improvements in the quality of mappings found by the design explorer are diminishing. Therefore, a representative subset of application scenarios is crucial to predict the ranking of the newly encountered mapping individuals. In case the optimal scenario subset was not found within 15 min, the search has failed. The properties used in the genetic algorithm are listed in Table I(a).

Fig. 11C shows that the second trainer is harder to predict than the first trainer. Trainer 2 has an average crowding distance [11] which is 22.6% smaller than trainer 1. A smaller crowding distance means that the mapping individuals are closer to eachother and thus harder to rank.

Fig. 11 shows the results of the experiments. Fig. 11A shows the success rate of the search. The success rate is equal to the percentage of times that an optimal subset is found. The optimal subsets are those that perfectly predict the Pareto rank. As a consequence, there may be multiple optimal subsets. Next, Fig. 11B shows the average evaluation time in case an optimal subset was found. Generally speaking, it can be seen that it is harder to find an optimal subset with the second training set. For all the approaches, the success rate of finding an optimal subset is much lower. Additionally, the time to find an optimal subset is larger.

Comparing the GA and the feature selection (FS) approach, we can conclude that the FS approach has a better success rate. With respect to speed, it depends on the training set. In case an optimal subset is easy to find (training set 1), the FS approach is faster than the GA. The main reason for this is that fewer modifications are required with respect to an initial subset in order to converge to an optimal subset. Therefore, the FS can easily perform these changes. Whenever an optimal subset is more complex to find, the GA benefits from its quicker

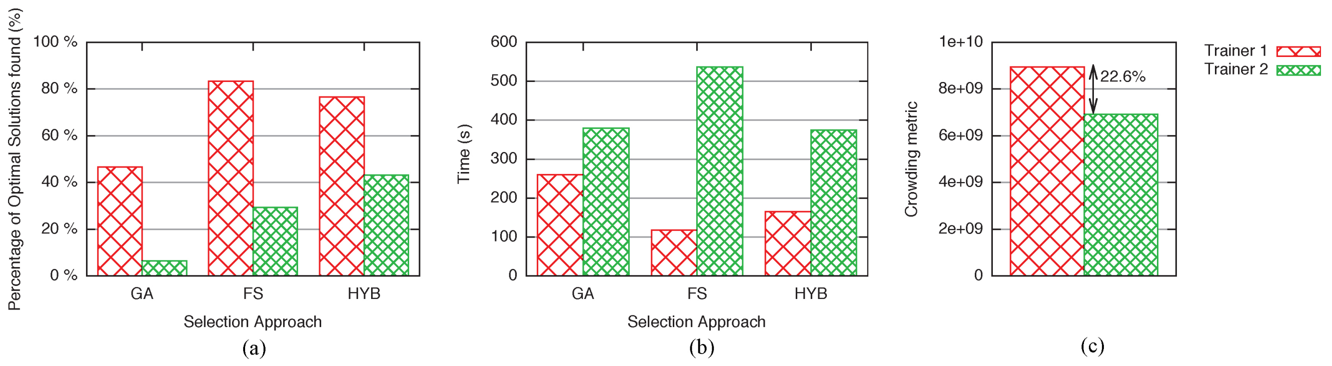

Fig. 11. Comparison of the three approaches to scenario subset selection. For each of the three approaches (genetic algorithm, feature selection, and hybrid), the subset with the optimal ranking is searched. Graph A show the success rate and graph B displays the time it takes to find this optimal subset. Graph C the difference between both trainers is visualized using the average crowding distance.

adaption of the subset (crossover can cause a change of half of the subset, whereas FS changes one scenario at the time). As a consequence, in the few cases where the GA finds an optimal subset, it achieves this in half the time.

Still, the success rate of the GA approach is insufficient. Therefore, we introduced the hybrid method. The hybrid method clearly outperforms the GA and the FS methods. For the first training set, the hybrid method is only slightly worse than the FS approach (77% versus 83%). As discussed before, in this training set an optimal subset is easy to find and the FS circumvents the computational needs of maintaining a pool of subsets. In the second training set, an optimal subset is harder to find and the hybrid approach can benefit from the pool with different potential subsets: 43% versus 29%. The fact that 57% of the optimal subsets are not found in the second training set does not imply that these subsets cannot be found. With a larger time limit, the GA can potentially still find them. The advantage of the hybrid method is that it can combine the thorough search of a specific part of the subset space of the FS approach with the GA approach that has less probability to end up in a local optimum. As a consequence, more optimal solutions are found. When an optimal subset is easy to find, this is paid for by a larger evaluation time. For more complicated searches, the evaluation time decreases due to the larger versatility of the genetic search (536 s versus 375 s).

A common technique to circumvent local optima in a feature selection technique is to run the algorithm with different initial subsets. Even if the FS approach is only run twice, it is clearly slower than the hybrid approach. Also, what the graphs in Fig. 11 do not show is the archive population at the end of the GA and the hybrid approach. In contrast to the FS approach, the GA and the hybrid approach will provide a diverse population of good subsets. This is beneficial in the case when the training set is updated. The ranking of the archive population may change and one of the other subsets in the archive may be the best subset that is used for improvement. The FS approach is only limited to one subset and has a lower possibility to immediately adapt itself to a new training set.

B. The Effect of Subset Size

After showing the success rate of the search for the optimal scenario subset, the next step is to show the effect of the subset size on the quality of the identified mapping. Therefore, we performed a DSE of 8 h with the hybrid approach to select a scenario subset. During this DSE, two threads were assigned to the subset selector and six threads were assigned to the design explorer. The GA parameters are given in Table I. The settings for the subset GA are the same as in the previous experiment.

TABLE I
EXPERIMENTAL SETTINGS

(a) Subset GA Settings

Size of Archive	20
Size of Offspring	40
Crossover Probability	0.7
Gene Mutation Prob.	0.02

(b) Design Explorer Settings

Size of Archive	100
Size of Offspring	200
Crossover Probability	0.9
Gene Mutation Prob.	0.01

In the experiment, four different subset sizes are used: 0.1%, 1%, 4%, and 16% of the total number of application scenarios. For each individual subset size, the result is averaged over nine DSE runs to take into account the stochastic nature of the GA in the design explorer. Fig. 12 shows the results of the experiments. At six points in wall-clock time, the current estimated Pareto front is logged. After the DSE, the estimated Pareto fronts are exhaustively evaluated to obtain the real fitness values of the individuals in the estimated Pareto front. From this set of mappings, the real execution time (cycle time) of the fastest mappings found by the DSE are given. For experiment A our objective is:

$$\begin{aligned} &\underset{m}{\text{minimize}}\,\text{time}(m)\\ &\text{subject to}\,\text{cost}(m) \le 100\end{aligned}$$

Experiment B allows a higher cost for the mappings:

$$\begin{aligned} &\underset{m}{\text{minimize}}\,\text{time}(m)\\ &\text{subject to}\,\text{cost}(m) \le 250\end{aligned}$$

For each time step, four bars are shown—the average minimal execution time for the DSE using a subset with 0.1%, 1%, 4%, and 16% of the scenarios.

When varying the number of scenarios in the subset there are two conflicting trends:

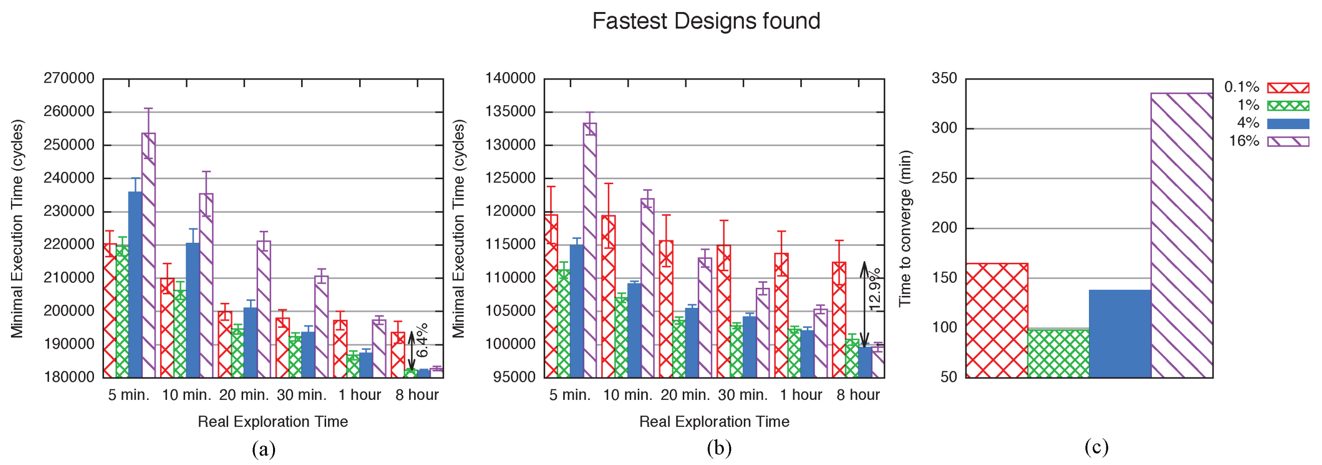

Fig. 12. Effect of the subset size on the quality perceived result of the design space exploration. In this graph the fastest designs are shown (with respect to total execution time) for designs within a certain cost.

1) *Accuracy*: The larger the subset, the more accurate the fitness prediction in the design explorer is. As a result, it can make better decisions about which individuals must be chosen for the next generation. In general, larger subsets are more likely to be able to identify the fast mapping within a certain time.
2) *Overhead*: The larger the subset, the longer it takes to obtain the fitness of a single mapping. A longer evaluation time of a single mapping not only means that less individuals can be evaluated in a certain time frame, but also that the GA performs less generations. As a consequence, the search has less benefit from the evolutionary process and it becomes harder to identify good mappings within a certain time frame.

After a short period (5 min), the evaluation overhead is the most significant effect when looking at the different subset sizes. For the 1%, 4%, and 16% subsets, the minimal execution time is larger as the size of the used scenario subset increases. An exception is the subset with 0.1% of the scenarios. As it is too small, it becomes too inaccurate to predict the fitness of a mapping. Consequently, its result after 5 min is already worse than the 1% subset. Still, the 0.1% subset is initially significantly better than the 16% subset size. This is completely caused by the overhead effect.

The (in)accuracy effect of 0.1% can be observed in two ways. At first the final result is 6.4–12.9% worse than the larger subsets (Fig. 12A). Secondly, the standard deviation over the different experiments is larger. In Figs. 12A and B, the deviation over the experiments is also shown using error bars. In case of the 0.1% subset, this deviation barely decreases over time. This shows that the DSE is far from accurate. For the other subsets, the prediction is accurate enough to result in a very small deviation at the end of 8 h of DSE.

To make the overhead effect more clear, Table II shows the average number of search generations that are performed in the total time frame of 8 h. Obviously, the smaller the subset size, the higher the number of performed generations. Less obvious is the nonlinear relation between the subset size and the generation count. At first, a lower evaluation time per generation means a larger sequential portion of the DSE application. This leads to less efficient usage of the multicore machine (Amdahl's law). Both the mappings and the set of scenarios can be evaluated in parallel. However, the selection and variance in between the generations of the GA still needs to be done sequentially.

TABLE II
NUMBER OF GENERATIONS IN 8 H

Subset Size	Generations
0.1%	2098
1%	1585
4%	856
16%	316

The overhead effect is not visible any more once the GA is converged. Fig. 12C shows the convergence time (within 1% of final result). In general, a larger subset means a larger convergence time. An exception is the 0.1% subset. The 0.1% subset is not able to provide good mappings as the fitness prediction is not accurate enough. The increased convergence time of the design explorer is also seen in the minimal execution time in Fig. 12A. In the first hour, the minimal execution time of the 4% subset larger than the 1% subset. The same holds for the 16% subset. Provided that the subset is accurate enough, the smaller the subset is, the earlier it gets close to the optimal execution time.

Therefore, we can speak of an accuracy threshold. Once the accuracy of the subset is above the accuracy threshold, the final GA results are not significantly effected by the subset size. However, due to the overhead effect, the convergence time will increase with a larger subset.

C. Subset Quality During DSE

The first experiment in Subsection V-A showed the subset quality on a fixed training set. This allows us to compare the approaches for the quality of subset selection, but it does not show us what actually happens during a DSE. Therefore, we performed a scenario-based DSE using a hybrid selection method. During the DSE, the subset quality is monitored over time. More precisely, before each update of the training set, the

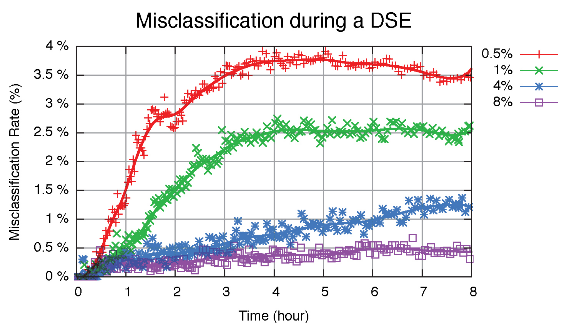

Fig. 13. Subset quality during the DSE. The quality is expressed in the misclassification rate on the current training set. Four different subset sizes are used: 0.5%, 1%, 4%, and 8% of the total number of application scenarios. The averaged measurements are shown using points, whereas the line shows the smoothed trend.

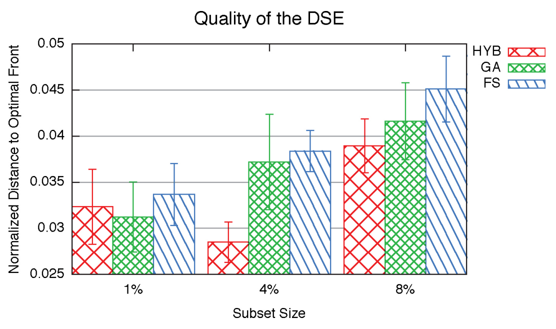

Fig. 14. Quality of the DSE for the different subset selection approaches. The quality is determined based on the distance between the estimated front and the optimal front.

quality of the subset is logged. The settings for the experiment are exactly the same as in Subsection V-B.

Fig. 13 shows the results of the experiment. For each curve, nine experiments are performed. During these experiments roughly 210 measurements are taken. The results of these experiments are averaged (as displayed with the points) after which they are smoothed. A couple of observations can be made based on this experiment. First, the smaller the subset, the higher the misclassification rate. This is as expected; the more application scenarios in the representative subset, the easier it is to predict the correct fitness (i.e. cost, cycle time and energy) of a mapping.

In this experiment, there are two opposing forces. First, the subset selector improves the subset over time. As a consequence, the quality becomes better. At the same time, the set of training mappings (provided by the design explorer and exhaustively evaluated by the updater) is changing. This changing training set makes it more complicated to correctly classify the training mappings. As a result, the quality of the subset decreases.

For all the experiments, at first the misclassification rate becomes higher (e.g., the quality is decreasing). Next, the subset quality stabilizes or even improves. The smaller subsets (0.5% and 1%) stabilize after 4 h. After that point, the misclassification rate remains stable (the 1% subset remains at a misclassification rate of around 2.5%) or it improves (the 0.5% subset misclassification rate drops from almost 4% to 3.5%). With the larger subsets, it is easier to provide a good classification (due to the larger number of application scenarios). This can especially be seen for the 8% subset. It hardly suffers from the warming up of the training set.

To conclude, the training set requires a short warm-up before it is challenging enough to train decent subsets with it. The initial set of training mappings has a higher crowding distance as the mappings are still well spread and the difference between rank 1 mappings and rank 2 mappings is easy to predict. Without any exception, all the performed experiments provide a perfect classification in the first 10 min. At this stage, the training set is still small and immature. Over time, the GA of the design explorer slowly starts to converge and the trainer becomes more crowded (i.e., mappings become more similar). Therefore, it becomes harder to correctly classify the training mappings. This improved training set is beneficial for the design explorer that constantly encounters unknown mappings. However, the perceived quality in the subset selector is decreasing. The decreased quality does not mean that the subset is worse. It is more a sign of a more crowded training set. At a certain point in time, this becomes visible: the training set complexity is stabilizing and the perceived misclassification rate of the subset stabilizes or becomes lower.

D. Subset Selection Method and its Effect on the Efficiency

Our final experiment shows a comparison between the different subset selection methods and its effect on the efficiency of the DSE. Therefore, the required exploration time for the scenario-based DSE to identify a satisfying mapping is measured. After all, the faster the DSE can provide results that match the requirement of the user, the better it is. For this purpose, a DSE of 100 min is performed with all the subset selector approaches. Each experiment is performed for three different subset sizes (1%, 4%, and 8%). The results are averaged over nine runs.

In order to obtain the efficiency of the multi-objective DSE, we obtain the distance of the estimated Pareto front (time versus energy) to the optimal front. For this purpose, we normalized the time and energy to a range from 0 to 1. As the optimal front is not exactly known, we took the combined Pareto front of all our experiments.

Fig. 14 shows the final results. In experiment V-B, we observed that when increasing the subset two effects will occur: 1) higher accuracy, and 2) slower convergence. This can be seen in Fig. 14. The GA and the FS subset selection have worse results when the subset becomes larger (the smaller the distance, the better).

The hybrid, however, shows a somewhat different effect. With 4% it is able to benefit from a subset with a higher accuracy. The slower convergence starts to effect the efficiency from the 8% subset.

Comparing the different methods, the hybrid method has the best results. The only exception is for the 1% subset. In this case the GA is still able to search the smaller design space of possible subsets. Still, the result of the hybrid method at 4% is better than the result of the GA at 1%. With the larger subset sizes, the hybrid method can exploit both the benefits of the feature selection and the genetic algorithm.

VI. Related Work

Scenario-based DSE addresses the design of MPSoC-based embedded systems with multiple applications. For a long time, only single fixed applications were addressed in the early DSE of embedded systems [2]. Recently, an increasing amount of literature is addressing the increasing dynamism of the nature and number of applications on embedded systems. A useful way of describing this application dynamism is by means of scenario-oriented design [19]. Scenario-oriented design is a benchmark-based design strategy where a notion of scenario programs is used to describe which applications can run concurrently during the usage of the embedded system.

With the introduction of workload scenarios [3], a new problem is arising in the early DSE. The increasing number of applications and their increasing dynamism translates into an exponential growth of scenarios that needs to be taken into account during scenario-oriented design. This exponential relation between the number of scenarios and the number of applications makes the early DSE of embedded systems even harder. It is not enough to extend existing single-application frameworks in such a way that multiple applications are supported, but the number of evaluated scenarios during the DSE must be limited in order to identify promising designs in a short-time frame.

Some of the early DSE approaches of embedded systems take multiple applications into account. Widely used are the use-case scenarios. Use-case scenarios are comparable to our notion of inter-application scenarios, and they describe which applications can be active simultaneously. CA-MPSoC [20] and MAMPS [21] are two design methodologies to generate multiprocessor systems for multiple use-cases. Both approaches try to find optimal mappings for each use-case individually after which the design of the different use-cases are merged. In case, the merged design of all the use-cases does not fit on the architecture, MAMPS also provides partitioning to divide the use-cases over different hardware designs. The main difference of these approaches is that in principle the mapping of a single application can change any time a new use-case is encountered. In our approach, the design of the final embedded system is less complex as the mapping of a single application is independent of the use-case. This may result in a nonoptimal application mapping for a specific use-case, but there is no overhead when a new use-case is encountered. In addition, both approaches use homogeneous architectures in order to make it possible to merge the design instances of different use-cases. In our tool, this limitation is not present. Not only are heterogeneous architectures allowed, but also the DSE is capable of allocating architectural components on the MPSoC in such a way that the architecture is optimized for all the use-cases.

An approach that does not merge design instances of different use-cases, but takes the interaction between multiple use-cases into account during design time, can be found in [22]. The authors use a Logic-Based Benders Decomposition approach using constraint programming to allocate the application tasks onto the MPSoC. The benefit of the decomposition approach is that the migration cost between the use-cases is taken into account during the design. Still, this approach does not take the intra-application dynamism into account.

An approach that takes both the inter- and intra-application dynamism into account can be found in [23]. The approach consists of static and dynamic parts. The static part will find an optimal multi-application mapping for each sequence of scenarios. This will result into templates that are used by the online dynamic part to adapt the system to changing circumstances. Although the approach is capable to adapt the system to the full dynamism in a multi-application workload, it does not address the increasing evaluation time due to the increasing number of scenarios in future multi-application workloads.

We address this issue by using fitness prediction. A survey on fitness prediction can be found in [5]. Our fitness predictors consist of a subset of the total set of application scenarios. This subset of application scenarios is evaluated to estimate the fitness of the mapping (i.e., a design instance). To train the fitness predictors, we are using co-exploration. This is slightly different than the co-evolutionary approach as presented in [24]. In their approach, the evolution of the designs (or solutions as it is called in [24]) is done in lockstep with the evolution of the fitness predictors. As a result, the training of the fitness predictors will always influence the search speed. In our approach, the training of fitness predictors can completely be separated from the search to high-quality solutions.

VII. Conclusion

In this paper, we presented scenario-based DSE for MPSoC-based embedded systems. Scenario-based DSE provided an efficient early design space exploration of dynamic multi-application workloads by co-exploring the design space of multi-application mappings onto an MPSoC with the design space of representative scenario subsets. The scenario subsets were used to quickly estimate the quality of a mapping, whereas a small dynamic set of mappings was used to train the representative scenario subset.

Three different approaches for scenario subset selection were described: 1) a genetic algorithm; 2) a feature selection algorithm; and 3) a hybrid approach. A genetic algorithm was fast and was quickly capable of covering the complete space of possible scenario subsets, but due to its stochastic nature it could easily miss the optimal scenario subsets with low-misclassification rates. The feature selection was more deterministic and its directed search can eventually found better subsets. However, feature selection was relatively slow and could suffer from local optima in the scenario subset space. The hybrid approach, as presented in this paper, combined the two techniques. Therefore, it could offer a quick coverage of the scenario subset space, a directed search and less sensitivity to local optima. By combining both methods (genetic algorithm and feature selection), it outperformed both individual methods.

The experiments in this paper showed that during the DSE there were two effects: 1) the overhead effect, and 2) the accuracy effect. A larger scenario subset results in a more expensive mapping evaluation. As a result, the DSE required more effort to identify the Pareto front. Additionally, a large subset also meant a higher accuracy. The higher accuracy resulted in a better prediction that made the genetic search to the Pareto front more efficient. Still, there was a certain accuracy threshold with respect to the size of the subset. If the size was below this threshold, the scenario-based DSE was not capable of identifying a good Pareto front. Once the subset was larger than the threshold, the mappings were classified precise enough to result in a suboptimal Pareto front.

Future work will focus on the run-time exploitation of application scenarios. Based on the dynamism in the multi-application workload and the architecture, the embedded system can be turned into an adaptive system that will dynamically change the mapping based on the current circumstances.

References

[1] K. Keutzer, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: Orthogonalization of concerns and platform-based design," *IEEE Trans. Computer-Aided Des. Integr. Circuits Syst.*, vol. 19, no. 12, pp. 1523–1543, Dec. 2000.

[2] M. Gries, "Methods for evaluating and covering the design space during early design development," *Integr. VLSI J.*, vol. 38, no. 2, pp. 131–183, 2004.

[3] S. V. Gheorghita, M. Hamers, J. Vandecappelle, A. Mamagkakis, S. Basten, T. Eeckhout, L. Corporaal, H. Catthoor, F. Vandeputte, F. Bosschere, and K. De, "System-scenario-based design of dynamic embedded systems," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 14, no. 1, pp. 1–45, 2009.

[4] P. van Stralen and A. D. Pimentel, "Scenario-based design space exploration of MPSoCs," in *Proc. IEEE Int. Conf. Comp. Des.*, Oct. 2010, pp. 305–312 .

[5] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments—A survey," *IEEE Trans. Evol. Comput.*, vol. 9, no. 3, pp. 303–317, Jun. 2005.

[6] P. van Stralen and A. D. Pimentel, "Fast scenario-based design space exploration using feature selection," in *Proc. Int. Workshop Parallel Programming Run-Time Manage. Tech. Many-core Architectures*, Feb. 2012, pp. 1–7.

[7] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Trans. Comp.*, vol. 55, no. 2, pp. 99–112, 2006.

[8] H. Nikolov, M. Thompson, T. Stefanov, A. Pimentel, S. Polstra, R. Bose, C. Zissulescu, and E. Deprettere, "Daedalus: Toward composable multimedia MP-SoC design," in *Proc. 45th Ann. Des. Automat. Conf.*, Jun. 2008, pp. 574–579.

[9] P. van Stralen and A. D. Pimentel, "A trace-based scenario database for high-level simulation of multimedia MP-SoCS," in *Proc. Int. Conf. Embedded Comp. Syst. Architectures MOdel. Simulat.*, Jul. 2010, pp. 11–19.

[10] G. Kahn, "The semantics of simple language for parallel programming," in *Proc. IFIP Congr.*, 1974, pp. 471–475.

[11] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[12] D. E. Goldberg. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1st ed. Addison-Wesley Professional [Online]. Available: http://www.worldcat.org/isbn/0201157675

[13] C. A. Coello Coello, G. B. Lamont, and D. A. Veldhuizen. (2007). "MOP evolutionary algorithm approaches," in *Evol. Algorithms Solving Multiobjective Problems* (Genetic and Evolutionary Computation, 2nd ed.). New York, NY, USA: Springer USA [Online]. Available: http://www.springerlink.com/content/978-0-387-33254-3

[14] D. A. van Veldhuizen and G. B. Lamont, "Multiobjective evolutionary algorithms: Analyzing the state-of-the-art," *Evol. Comput.*, vol. 8, no. 2, pp. 125–147, 2000.

[15] P. Somol, J. Novovicova, J. Grim, and P. Pudil, "Dynamic oscillating search algorithm for feature selection," in *Proc. Int. Conf. Pattern Recog.*, Dec. 2008, pp. 1–4.

[16] H. Orsila. (2009, Feb.). *kpn-Generator* [Online]. Avialble: http://zakalwe.fi/~shd/foss/kpn-generator/

[17] J. Hur, T. Stefanov, S. Wong, and S. Vassiliadis, "Systematic customization of on-chip crossbar interconnects," in *Reconfigurable Computing: Architectures, Tools and Applications* (Lecture Notes in Computer Science, vol. 4419). Berlin/Heidelberg, Germany: Springer, 2007, pp. 61–72.

[18] Sara. (2011). *Lisa Cluster* [Online]. Avialble: http://www.sara.nl/systems/lisa

[19] J. M. Paul, D. E. Thomas, and A. Bobrek, "Scenario-oriented design for single-chip heterogeneous multiprocessors," *IEEE Trans. vERY lARGE sCALE iNTEGRATION (VLSI) Syst.*, vol. 14, no. 8, pp. 868–880, Aug. 2006.

[20] A. Shabbir, A. Kumar, S. Stuijk, B. Mesman, and H. Corporaal, "CA-MPSoC: An automated design flow for predictable multi-processor architectures for multiple applications," *J. Syst. Architecture*, vol. 56, no. 7, pp. 265–277, Jul. 2010.

[21] A. Kumar, S. Fernando, Y. Ha, B. Mesman, and H. Corporaal, "Multiprocessor systems synthesis for multiple use-cases of multiple applications on FPGA," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 13, no. 3, pp. 1–27, Jul. 2008.

[22] L. Benini, D. Bertozzi, and M. Milano, "Resource management policy handling multiple use-cases in MPSoC platforms using constraint programming," in *Logic Programming* (Lecture Notes in Computer Science, vol. 5366). Berlin, Germany: Springer, Dec. 2008, pp. 470–484.

[23] A. Schranzhofer, J.-J. Chen, L. Santinelli, and L. Thiele, "Dynamic and adaptive allocation of applications on MPSoC platforms," in *Proc. Asia South Pacific Des. Automat. Conf.*, 2010, pp. 885–890.

[24] M. Schmidt and H. Lipson, "Coevolution of fitness predictors," *IEEE Trans. Evol. Comput.*, vol. 12, no. 6, pp. 736–749, Nov. 2008.

Peter van Stralen received the B.Sc. degree in computer science and the M.Sc. degree in grid computing from the University of Amsterdam, Amsterdam, The Netherlands, where he is currently pursuing the Ph.D. degree.

His current research interests include embedded systems, design space exploration, and concurrency.

Andy D. Pimentel received the M.Sc. and Ph.D. degrees in computer science from the University of Amsterdam, Amsterdam, The Netherlands.

He is currently an Associate Professor at the Computer Systems Architecture Group, Informatics Institute, University of Amsterdam. His current research interests include computer architecture modeling and simulation, system-level design, design space exploration, performance and power analysis, embedded systems, and parallel computing.

Dr. Pimentel serves on the editorial boards of Elsevier's *Simulation Modelling Practice and Theory* and *Springer's Journal of Signal Processing Systems*. He serves on organizational committees for a range of leading conferences, such as DAC, DATE, ICCAD, FPL, SAMOS, and ESTIMedia.