Exploring Exploration: A Tutorial Introduction to Embedded Systems Design Space Exploration

Andy D. Pimentel
University of Amsterdam

Editor's note:
As embedded systems grow more complex and as new applications such as IoT require many design constraints, sophisticated design space exploration techniques are essential in order to find the best compromise between different design goals and their tradeoff. This tutorial gives a structured insight into the field of design space exploration for embedded systems.
—Jörg Henkel, Karlsruhe Institute of Technology

Designers of modern embedded systems face several daunting challenges since these systems typically have to meet a range of stringent, and often conflicting, design requirements. As many embedded systems target mass production and battery-based devices or devices that cannot use active cooling, they should be cheap and power efficient. At the same time, a great deal of these systems must, increasingly, support multiple applications and standards for which they need to provide real-time performance. For example, mobile devices must support different standards for communication and coding of digital contents. Furthermore, modern embedded systems also need to be reliable as well as flexible such that they can easily be updated and extended with future applications and standards. The latter calls for a high degree of programmability of these systems, whereas performance, power consumption, and cost constraints require implementing substantial parts of these systems in dedicated hardware blocks. As a result, modern embedded systems often have a heterogeneous multiprocessor system architecture. They consist of processors that range from fully programmable cores to fully dedicated hardware blocks for time-critical application tasks. Increasingly, the components in such systems are integrated onto a single chip, yielding heterogeneous multiprocessor system-on-chip (MPSoC) architectures [1].

To cope with the design complexity of such systems, we have witnessed the emergence of a new design methodology in the past 15–20 years, called electronic system-level (ESL) design. It aims at raising the level of abstraction of the design process to improve the design productivity. Key enablers to this end are the use of architectural MPSoC platforms to facilitate reuse of IP components and the concept of high-level system modeling and simulation [2], [3]. The latter allows for capturing the behavior of platform components and their interactions at a high

Digital Object Identifier 10.1109/MDAT.2016.2626445
Date of publication: 8 November 2016; date of current version: 10 January 2017.

level of abstraction. As such, these high-level models minimize the modeling effort and are optimized for execution speed, and can therefore be applied during the very early design stages to perform design space exploration (DSE) [4]. During such DSE, a large variety of different design alternatives can be explored, such as the number and type of processors deployed in the platform architecture, the type of interconnection network used to connect system components, or the spatial binding and temporal binding (i.e., scheduling) of application tasks to processor cores. This process of early DSE is of paramount importance as the considered design choices may heavily influence the success or failure of the final product. However, the process of DSE also is highly challenging because the design space that needs to be explored typically is vast, especially during the early stages of design. For instance, the design space for exploring different mappings of application tasks to processing resources and trying to optimize the mapping for, e.g., performance or power consumption exponentially grows with the number of application tasks and processors, and is generally considered to be an NP-hard problem [5]. Therefore, the development of efficient and effective DSE methods has received significant research attention in recent years. In this article, we will provide a tutorial introduction to the topic of embedded systems DSE.

DSE: Basic concepts

During the DSE of embedded systems, multiple optimization objectives, such as performance, power/energy consumption, and cost, should be considered simultaneously. This is called multiobjective DSE. Since the objectives are often in conflict, there cannot be a single optimal solution that simultaneously optimizes all objectives. Therefore, optimal decisions need to be taken in the presence of tradeoffs between design criteria.

Given a set of m decision variables, which are the degrees of freedom (e.g., MPSoC system parameters like the number and type of processors, application mapping, etc.) that are explored during DSE, a so-called fitness function must optimize the n objective values. The fitness function is defined as

$$f_i : R^m \rightarrow R^1. \quad (1)$$

A potential solution $x \in R^m$ is an assignment of the m decision variables. The fitness function f_i translates a point in the solution space X into the ith objective value (where $1 \leq i \leq n$). For example, a particular fitness function f_i could assess the performance or energy efficiency of a certain solution x (representing a specific design instance). The combined fitness function $f(x)$ subsequently translates a point in the solution space into the objective space Y. Formally, a multiobjective optimization problem (MOP) that tries to identify a solution x for the m decision variables that minimizes the n objective values using objective functions f_i with $1 \leq i \leq n$

$$\begin{aligned} &\text{Minimize } y = f(x) = (f_1(x), f_2(x), \ldots, f_n(x)) \\ &\text{where } x = (x_1, x_2, \ldots, x_m) \in X \\ &\qquad y = (y_1, y_2, \ldots, y_n) \in Y. \end{aligned}$$

In the remainder of this discussion, we assume a minimization procedure, but without loss of generality, this minimization procedure can be converted into a maximization problem by multiplying the fitness values y_i with –1.

With an optimization of a single objective, the comparison of solutions is trivial. A better fitness (i.e., objective value) means a better solution. With multiple objectives, however, the comparison becomes nontrivial. Take, for example, two different MPSoC designs: a high-performance MPSoC and a slower but much cheaper MPSoC. In case there is no preference defined with respect to the objectives and there are also no restrictions for the objectives, one cannot say if the high-performance MPSoC or the low-cost MPSoC is better. A MOP can have even more different objectives, like the performance, energy consumption, cost, and reliability of an MPSoC-based embedded system. To compare different solutions in the case of multiple objectives, the Pareto dominance relation is typically used. Here, a solution $x_1 \in X$ is said to dominate solution $x_2 \in X$ if and only if $x_1 < x_2$

$$\begin{aligned} x_1 < x_2 \Leftrightarrow\ & \forall i \in \{1,2,\ldots,n\} : f_i(x_1) \leq f_i(x_2) \wedge \\ & \exists i \in \{1,2,\ldots,n\} : f_i(x_1) < f_i(x_2). \end{aligned}$$

Hence, a solution x_1 dominates x_2 if its objective values are superior to the objective values of x_2. For all of the objectives, x_1 must not have a worse objective value than solution x_2. Additionally, there must be at least one objective in which solution x_1 is better (otherwise they are equal).

An example of the dominance relation is given in Figure 1, which illustrates a 2-D MOP. For solution H, the dominance relations are shown. Solution H is

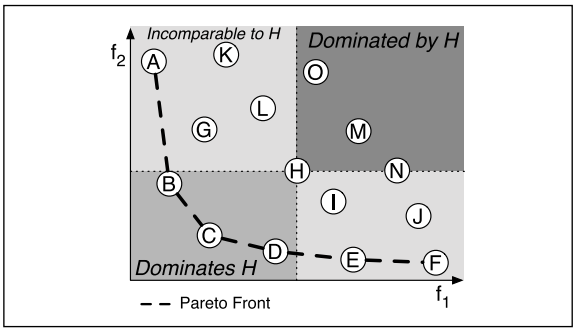

Figure 1. A Pareto front and an example of the dominance relation.

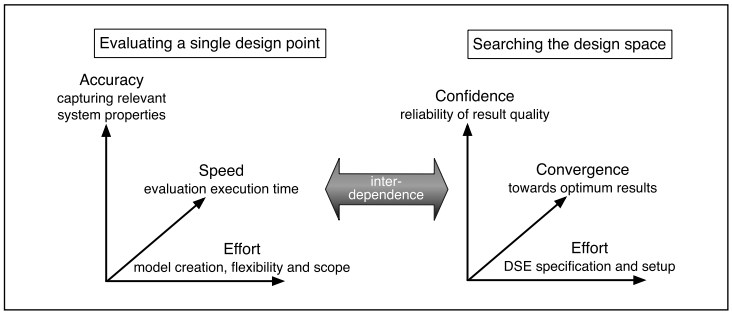

Figure 2. A taxonomy for DSE approaches (taken from [6]).

dominated by solutions *B*, *C*, and *D* as all of them have a lower value for both f_1 and f_2. On the other hand, solution *H* is superior to solutions *M*, *N*, and *O*. Finally, some of the solutions are not comparable to *H*. These solutions are better for one objective but worse for the other.

The Pareto dominance relation only provides a partial ordering. For example, the solutions *A* to *F* of the example in Figure 1 cannot be ordered using the ordering relation. Since not all solutions $x \in X$ can be ordered, the result of a MOP is not a single solution, but a front of nondominated solutions, called the Pareto front. A set X' is defined to be a Pareto front of the set of solutions X as follows:

$$\{x \in X' \mid \nexists\, x_i \in X : x_i < x\}$$

The Pareto front of Figure 1 contains six solutions: *A* – *F*. Each of these solutions does not dominate the other. An improvement on objective f_1 is matched by the worse value for f_2. Generally, it is up to the designer to decide which of the solutions provides the best tradeoff.

The search for Pareto optimal design points with respect to multiple design criteria entails two distinct elements [4]: 1) the evaluation of a single design point using the fitness function(s) regarding all the objectives in question like system performance, power/energy consumption, and so on; and 2) the search strategy for covering the design space during the DSE process. Figure 2 shows a simple taxonomy for DSE approaches, recognizing the two DSE elements as well as different properties of these DSE elements. As will be discussed in more detail later on, there usually exists a tradeoff between the accuracy and speed with which the fitness of single design points can be evaluated. In addition to this, the various fitness evaluation techniques also differ with respect to the implementation effort and the capability of evaluating the fitness for a wide range of systems, involving issues such as modularity, reusability of models etc.

Regarding the search strategy element of DSE, the confidence characteristic denotes how certain we are that the design points returned by the DSE include the true optimum, or alternatively, how close they are to the true optimum. In many search algorithms, confidence is obtained by avoiding local optima and ensuring sufficient design space coverage. Clearly, an exhaustive search in which every single point in the design space is evaluated and compared would provide a 100% confidence. However, such exhaustive search is usually prohibitive due to the sheer size of the design space. In those cases, heuristic search techniques can be used to search the design space for optimal solutions using only a finite number of design point evaluations. The convergence property denotes the speed of evaluating a range of design points, and, more specifically, the rate at which the DSE search algorithm manages to converge to an optimum. Finally, analogous with the effort property in the case of evaluating a single design point, the effort for searching the design space refers to the implementation of the search method and setting its parameters, as well as setting up, running, and evaluating the results of the exploration experiment. In the two subsequent sections, we will provide a more detailed overview of the different techniques, and their properties, applied in each of the two elements of DSE.

Evaluation of a single design point

Methods for evaluating the fitness of a single design point in the design space roughly fall into one of three categories: 1) measurements on a (prototype) implementation; 2) simulation-based

evaluations; and 3) estimations based on some kind of analytical model. Each of these methods has different properties with regard to evaluation time and accuracy. Evaluation of prototype implementations provides the highest accuracy, but long development times prohibit evaluation of many design options. Analytical estimations are considered the fastest, but accuracy is limited since they are typically unable to sufficiently capture particular intricate system behavior. Simulation-based evaluation fills up the range in between these two extremes: both highly accurate (but slower) and fast (but less accurate) simulation techniques are available. This tradeoff between accuracy and speed is very important, since successful DSE depends both on the ability to evaluate a single design point as well as being able to efficiently search the entire design space. As current DSE efforts in the domain of embedded systems design typically use simulation or analytical models to evaluate single design points, the remainder of this section will focus on these methods.

Simulative fitness evaluation

Simulating system components can, as was already mentioned above, be performed at different levels of abstraction. The higher the abstraction level, the less intricately the system components are modeled and, therefore, the higher the simulation speed is. Evidently, such efficiency improvements come at the cost of a less accurate fitness estimation because of the fact that particular system details are not taken into account. This simulation speed-accuracy tradeoff is shown in Figure 3. This figure depicts several widely used simulation abstraction levels, and it does so for both the simulation of processor components as well as the simulation of communication between system components.

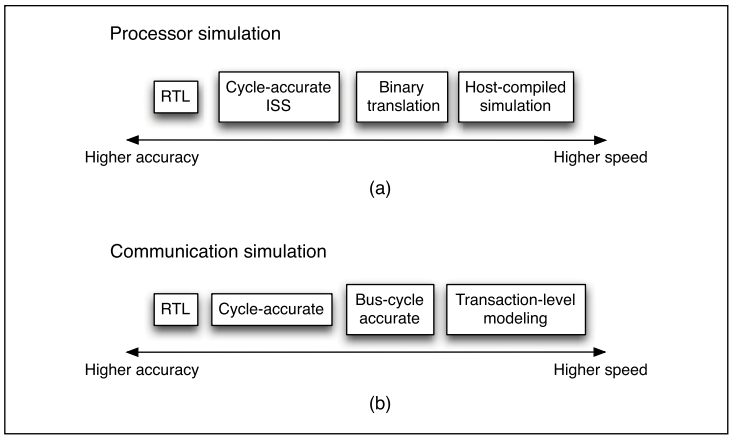

Figure 3. Different levels of abstraction for (a) simulating processors and (b) simulating communication.

For both the simulation of processor and communication components, the lowest level of abstraction for simulating a digital system is the register-transfer level (RTL). At this level of abstraction, the flow of digital signals between registers and combinational logic is explicitly simulated. This yields a highly accurate but also very slow simulation. As a result, the use of RTL simulation in the process of DSE is confined to only relatively small and narrow design spaces focusing on, for example, the design of one specific system component. Performing system-level DSE is infeasible using RTL simulation.

Raising the level of abstraction, one can simulate system components at the cycle accurate level. This means that the system components are simulated on a cycle-by-cycle basis and, as such, that the simulated system state conforms to the cycle-by-cycle behavior of the target design. This results in more efficient simulations as compared to RTL simulation at the cost of a somewhat reduced accuracy since the system state between cycles is not accounted for. Cycle-accurate simulation is a popular technique for simulating microprocessors: so-called cycle-accurate instruction set simulation (ISS). These ISS simulators try to capture the cycle-by-cycle behavior of the microarchitectural components of a microprocessor, such as the pipeline logic, out-of-order processing, branch predictors, caches, and so on. To account for power consumption behavior, ISS simulators often use activity-based power models that accumulate the power consumption of the relevant microarchitecture components based on their activity ratio. A good example is the widely used cycle-accurate Gem5 ISS [7], which can be extended to also support area and power predictions using activity-based modeling frameworks such CACTI [8] and McPAT [9]. Although these ISS simulators can be deployed to perform microarchitectural DSE for processor components, they are typically still too slow for performing full system-scale DSE.

In cycle-accurate ISS simulators, the fetching, decoding, and execution of instructions are explicitly simulated. To further optimize the speed of such simulators, one could translate the instructions from the target binary to be simulated to an equivalent sequence of instructions (using static or dynamic

just-in-time translation) that can be executed on the simulation host computer. This so-called binary translation technique, which is, e.g., deployed in the widely used QEMU simulator [10], aims at reducing the overhead of explicitly simulating the instruction fetch and decode stages. The translated instruction sequences are often instrumented with additional code to keep track of the extra-functional behavior, such as timing and power consumption, of the original code as it would have been executed on the target processor.

For simulating communication between system components, one could use so-called bus-cycle accurate simulation [11] to speed up the simulation process. In this type of simulation, all signals of the communication bus are modeled explicitly in a cycle accurate fashion, but this accuracy is only maintained for the signals on the communication bus and not for the logic around it. The surrounding components can thus use more abstract timing models.

Raising the abstraction level even further for processor simulation yields so-called host-compiled simulation [12]. In this technique, the source code of the target program is directly compiled into a binary program that can run on the host computer. In addition, and similar to the binary translation technique, the source code can be instrumented with a timing and power consumption model based on the target architecture. Since these simulations are efficient as they directly execute target programs on the host computer, they are very suitable for system-level DSE. However, at this level of abstraction, it is difficult to accurately capture intricate microarchitectural behavior, like pipeline and cacheing behavior. Another drawback of this simulation approach is that one needs to have access to the source code of a target program.

For simulating communication, transaction-level modeling (TLM) [11] provides the highest level of abstraction. In TLM, communication details at the level of signals and protocols are abstracted away by means of encapsulation into entire transactions between system components. At this level, the emphasis is more on the functionality of the data transfers, i.e., what data are transferred to and from what locations, rather than on their actual implementation. Evidently, the extra-functional behavior in TLM simulation models is also captured at the level of entire transactions.

The above processor simulation techniques are all execution-driven simulation methods as they are directly driven by the execution of a program. Alternatively, there are also trace-driven simulation techniques in which the simulation is driven by event traces that have been collected through the execution of a program (e.g., [13] and [14]). These trace events can focus on the evaluation of specific system elements such as memory address traces for cache simulation. However, an event trace may also consist of the full sequence of executed instructions, thereby allowing full, trace-driven microprocessor simulation for the purpose of performance and/or power estimation. To optimize for simulation speed, the trace events may also represent computations (and, if needed, communication) at a higher level of abstraction than the level of machine instructions, like at the level of the execution of basic blocks or even entire functions. Another advantage of trace-driven simulation is the fact that the event traces often only need to be generated once (i.e., executing the program to collect the traces once), after which they can be reused in the DSE process. Drawbacks of trace-driven simulation evidently are the need for storing the event traces which can become extremely large in size, and the fact that trace-driven simulation does not allow for simulating all intricate system behavior, such as the effects of speculative instruction execution in microprocessors.

An example of a high-level, trace-driven MPSoC simulation environment is the Sesame system-level modeling and simulation framework [15]. Sesame is based on the Y-chart methodology [16], and accordingly it recognizes separate application and architecture models. The application models are explicitly mapped onto the architecture models by means of trace-driven simulation. The workload of an application is captured by instrumenting the application model, which is a parallel specification of the application, with annotations that describe the application's computational and communication actions at a coarse-grained level (typically at the level of the execution of entire functions). By executing this instrumented application model, these annotations cause the generation of traces of application events that subsequently drive the underlying architecture model. This architecture model, capturing the system resources and their constraints, then simulates the consequences of the consumed computation and communication events in terms

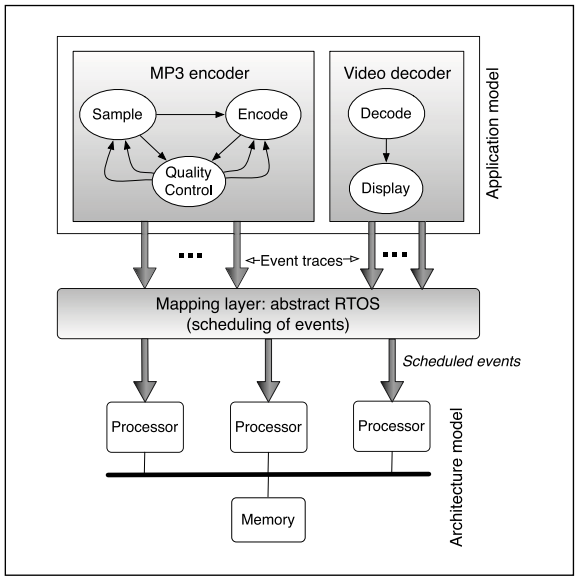

Figure 4. The Sesame system-level MPSoC simulation infrastructure.

of extra-functional system behavior (performance, power consumption, etc.). Figure 4 depicts Sesame's layered organization, illustrating the mapping of two multimedia applications (an MP3 encoder and video decoder) onto a bus-based MPSoC platform. A special mapping layer in Sesame provides the scheduling of application events in the case multiple application processes are mapped onto a single processing resource.

Orthogonal to most of the (processor) simulation methods described above, there are additional techniques to further improve the simulation speed [17]. In sampled simulation, for example, the simulation does not cover the execution of an entire program but only simulates relatively small samples of the program's execution. Here, the challenge is to select the samples in such a manner that they sufficiently represent the behavior as if the entire program was simulated. Another technique for speeding up simulation is statistical simulation. Rather than using real (benchmark) programs for simulation, it uses a statistical program profile. This profile captures the distributions of important program characteristics, and is used for generating a synthetic instruction trace that drives a simple trace-driven simulator. As the synthetic trace is randomly generated from a statistical profile, this type of simulations can converge to a set of performance predictions fairly quickly.

Analytical fitness evaluation

In comparison to simulation, analytical models allow for much more efficient prediction of the extra-functional system behavior at the expense of a reduced accuracy. This makes analytical models very suitable for exploring large design spaces to rapidly identify regions of interest that can be later explored in more detail using simulation. Another advantage of analytical models is that they can provide direct insight into the relationship between model parameters (representing design choices) and the predicted extra-functional behavior. For simulative methods, such understanding would require a large number of simulations.

Analytical models can roughly be divided into three classes [17]: 1) mechanistic (or whitebox) models; 2) empirical (or blackbox) models; and 3) a hybrid combination of mechanistic and empirical modeling. Mechanistic models are based on first principles, which implies that they are built in a bottom-up fashion starting from a basic understanding of the mechanics of the modeled system. For example, in a mechanistic microprocessor performance model, penalties due to cache misses, branch mispredictions, the execution of instructions with different latencies, etc., are explicitly captured in the model.

In empirical models, statistical inference and machine learning techniques, like regression models or neural networks, are used to automatically synthesize a model through the process of learning from training data. For example, using a set of microarchitectural parameters such as pipeline depth, issue width, caches sizes, etc., one could train a model that predicts the Instructions Per Cycle (IPC) or Cycles Per Instruction (CPI) of a microprocessor. Inferring a model by means of automatic training typically is easier than developing a mechanistic model because it does not require intimate understanding of the mechanics of the modeled system. Evidently, the latter is also an immediate drawback as empirical models also tend to provide less insight than mechanistic models.

In hybrid mechanistic-empirical modeling, which is sometimes referred to as greybox modeling, extra-functional system aspects are captured using a formula that has been derived from insights in the underlying system. However, this formula includes a number of unknown parameters, which are then inferred through fitting (e.g., using regression), similarly to empirical modeling. Such hybrid

mechanistic-empirical modeling is motivated by the fact that it provides insight (like mechanistic modeling) while easing the construction of the model (like empirical modeling).

Searching the design space

As explained before, searching a design space is a multiobjective optimization process. This process will evidently benefit from a good tradeoff between speed, accuracy, and effort of the method for evaluating the fitness of a single design point, as discussed in the previous section. But, even if this tradeoff is ideal, we still have to make sure that each evaluation of a design point contributes as much as possible to an effective and efficient search of the design space. A crucial component toward this goal is the search algorithm that navigates the design space toward areas of interest by proposing which design points to evaluate next. Regardless of the specific type of search method that is used for such a design space traversal, its success depends on three major concerns, as was shown in Figure 2: confidence, convergence, and effort. These concerns typically cannot be considered in isolation, as they are highly interdependent, contradictory, and sometimes overlapping. The state of the art in DSE can be summarized as finding a good tradeoff between these concerns.

DSE search algorithms can be divided into exact and heuristic methods. In exact DSE methods, like those implemented using integer linear programming (ILP) solutions (e.g., [18] and [19]) or branch & bound algorithms (e.g., [20]), the optimum is guaranteed to be found. As such methods generally are compute intensive, they typically use design space pruning (i.e., discarding unsuitable design points) to optimize the efficiency of the search, thereby allowing to handle larger design spaces. However, for realistic design problems with design spaces that are vast, these methods may still be less suited. Alternatively, in heuristic methods, metaheuristics are used to find a design point in the known design space that meets the design requirements as best as possible. To this end, these methods search the design space for optimal solutions using only a finite number of design point evaluations, and can thus handle larger design spaces. However, there is no guarantee that the global optimum will be found using metaheuristics, and therefore the result can be a local optimum within the design space. Examples of metaheuristics are hill climbing, tabu search, simulated annealing, ant colony optimization, particle swarm optimization, and genetic algorithms. In this tutorial, we will focus on methods to navigate the design space that are based on genetic algorithms (GA). GA-based DSE has been widely studied in the domain of system-level embedded design (e.g., [21] and [22]) and has been demonstrated to yield good results. Moreover, GAs can be used in their basic (domain-independent) form or, as will also be explained later on, with custom extensions that incorporate domain-dependent knowledge in order to improve search performance even further.

GA-based DSE

GAs operate by searching through the solution space (spanned by the design variables/decisions being explored) where each possible solution is encoded as a string-like representation, often referred to as the chromosome [23]. A (randomly initialized) population of these chromosomes is then iteratively modified by performing a fixed sequence of actions that are inspired by their counterparts from biology: fitness evaluation and selection, crossover, and mutation. A fundamental design choice of a GA is the genetic representation of the solution space, because each of the crossover and mutation steps depends on it. To illustrate how such a genetic representation could look like, let us use a widely studied DSE problem in the domain of system-level embedded system design as an example: optimizing the mapping of a (set of) concurrent application(s) onto an underlying (heterogeneous) MPSoC platform architecture [5]. As a convenient mapping description for an application with n tasks, we use a vector of size n with processor identifiers p_i, where p_i indicates the mapping target of task i

$$[p_0, \ldots, p_i, \ldots, p_{n-1}].$$

This commonly used description is very suitable to serve as the chromosome representation for a GA.A valid mapping specification is a feasible partitioning of all n tasks. With feasible, we mean that tasks are mapped onto processing elements that can execute those tasks (i.e., there are no functional restrictions of the processing element in question, like an ASIC component that only allows the execution of one particular piece of functionality), and that communicating tasks are mapped onto processing elements that can actually communicate with each other (i.e., there are no topological communication

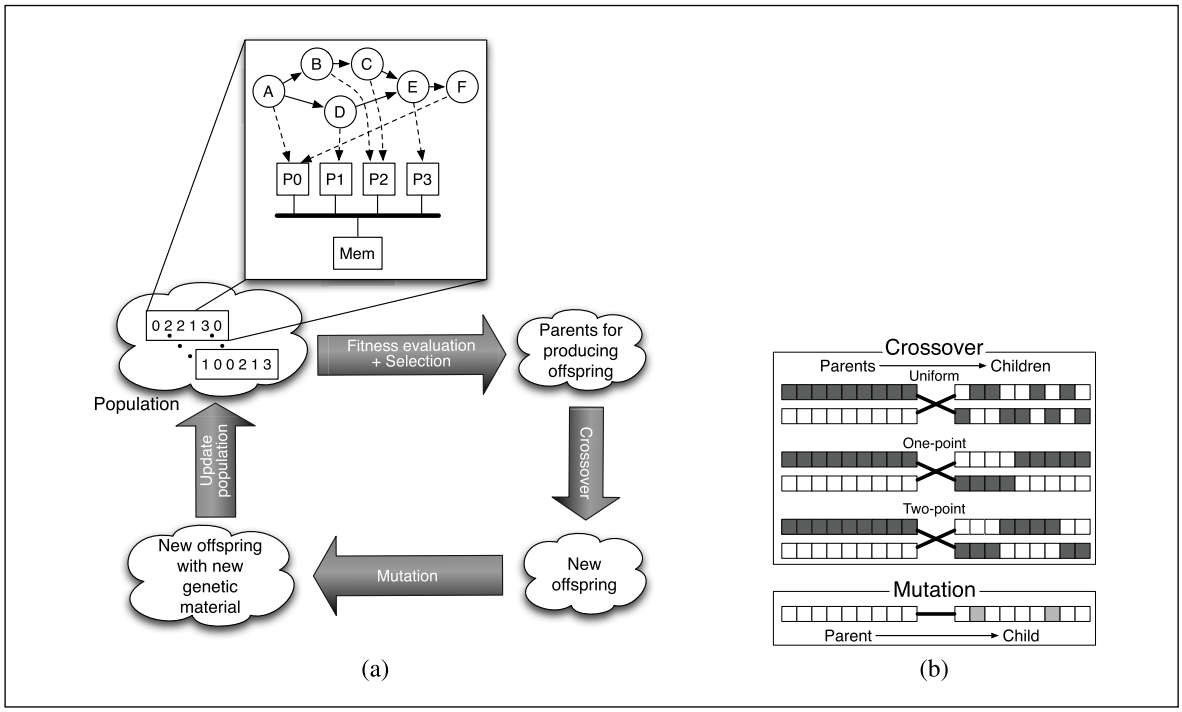

Figure 5. GA-based mapping DSE: (a) general overview of the GA steps; and (b) crossover and mutation operators.

restrictions). In case an infeasible mapping is created by the genetic operators of a GA (crossover and mutation), a mechanism is required that either discards or repairs such a chromosome. Repairing a chromosome implies that it is transformed into a valid chromosome (mapping) that is as close as possible to the original, invalid one. Moreover, note that task partitions specifying a mapping may also be empty [particular processor(s) not in use] or contain all n tasks (a single processor system). A processor that is not assigned any tasks (having an empty task partition) can be considered idle or nonexistent.

In Figure 5a, the different steps of a GA are shown. This figure also illustrates the mapping representation of a chromosome for an application with six tasks and a 4-processor bus-based MPSoC platform. Starting from a (randomly initialized) population of chromosomes, representing the different mapping design instances, the fitness of the mapping solutions in the population is first evaluated. To this end, any of the previously discussed analytical or simulative techniques can be used. Subsequently, based on the fitness evaluation, a selection of chromosomes is made that will be used to create offspring. This offspring is created by combining genetic material from two parents using a crossover operation, as illustrated in the top part of in Figure 5b. There exist various forms of this crossover operator, of which the uniform, onepoint, and two-point crossovers are the most popular. Next, new genetic material is introduced in the offspring by means of a mutation operator as illustrated at the bottom of Figure 5b. Such a mutation randomly changes one or more genes within chromosomes. Finally, the newly created offspring is used to update the population by either replacing it or by deploying so-called elitism. Such elitism involves the combination of the new offspring with a small number of the best solutions from the original population to avoid loosing strong solutions.

To provide a small example of the results a GA-based DSE could obtain, we present some results of a small-scale case study where the design space consists of an application with 11 tasks that is to be mapped onto a 4-processor MPSoC architecture with a crossbar interconnect [6]. The mapping design space contains more than four million design points, of which 175 000 are unique ones (as the target platform is a homogeneous, symmetric MPSoC). Because of the relatively small design space, in this particular case, we were also able to perform an

exhaustive search, allowing us to evaluate the quality of the GA-based search results. To account for the stochastic behavior of GAs, all results are averages over 300 GA runs. The fitness of mapping solutions has been evaluated using the Sesame MPSoC simulation framework [15] (see also the Simulative fitness evaluation section). Figure 6 shows the results of the GA-based DSE with different population sizes (10, 15, 40, or 80 chromosomes), a constant mutation rate (0.1) and crossover probability (0.9), and a uniform crossover in a so-called probability-quality (P-Q) plot. Regarding the top part of this plot, the horizontal axis (top *x*-axis) represents the quality of the result as a percentile toward the true optimum (a lower percentile indicates a result closer to the optimum) and the vertical axis represents the probability of achieving a result with that quality. The straight lines in the graph represent the theoretically derived probabilities of finding results using a simple, uniform random search. We have also computed the 80%–95% confidence intervals of the mean fitness value (execution time in cycles, in this case) of mapping solutions found by the GA, averaged over the 300 runs of each GA search. These confidence intervals, shown at the bottom of the graph in Figure 6, indicate how certain (as specified by the confidence level) we are that the real mean lies within the confidence interval. The more the confidence intervals for different experiments are nonoverlapping, the more significant the difference of the mean behavior (which is clearly the case in the example of Figure 6). The results from this particular case study show that the GA-based DSE with the largest population size can find mapping solutions that are always very close to the real optimum: within the 0.1 percentile, implying that they belong to the best 175 000/1000 = 175 solutions. A larger population size, however, comes with a higher number of fitness evaluations during the search and thus requires a longer search time (assuming the number of search iterations remains constant). According to Figure 6, a population size of 40 may therefore provide a good compromise.

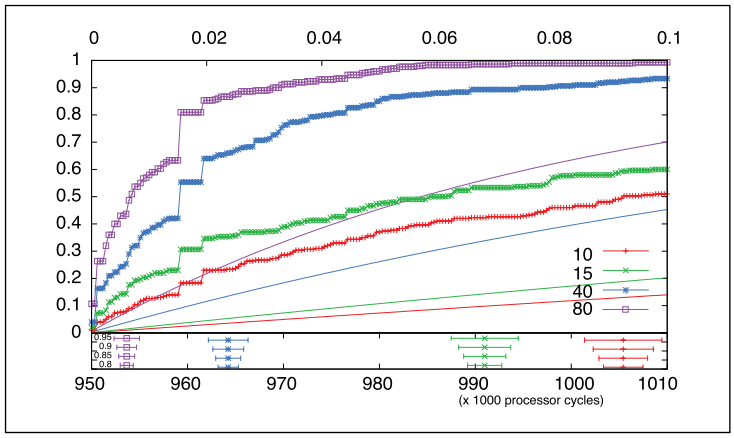

Figure 6. P-Q plot for GA-based DSE with different population sizes.

Optimizing GA-based DSE

There are various methods for making the search process of a GA-based DSE more efficient. This allows the DSE process to either find the design candidates quicker (i.e., improve the convergence behavior of the DSE) or to spend the same amount of time to evaluate more design points. The latter can be used to enable the search of larger design spaces or to improve the chance of finding better design candidates (i.e., improve the confidence property of the DSE). One approach for optimizing the GA-based search is to enrich the genetic operators of the GA with domain knowledge such that they produce more diverse offspring or offspring with a higher probability of being closer to the optimum. For example, in [24], new GA operators have been proposed that optimize the search performance by 1) reducing the redundancy present in chromosome representations (e.g., mapping symmetries in the case of homogeneous, symmetrical MPSoC platforms); or 2) using a new crossover operator that is based on a mapping distance metric that provides a measure of similarity between mappings. Using this mapping distance information, the new crossover operator aims at retaining the strong chromosome parts of both of the parents. In [25], a new mutation operator has been proposed that considers the affinity of tasks with respect to processors, the communication cost between tasks, and the differences of processor workloads to steer the mutation in such a way that offspring is produced with a higher probability of being (near) optimal.

Another approach for optimizing GA-based DSE concerns the reduction of the time taken to evaluate the fitness of solutions during the GA's execution. As mentioned before, DSE approaches typically use either simulation or an analytical model to evaluate the fitness of design points, where simulative approaches prohibit the evaluation of many design options due to the higher evaluation performance

costs and analytical approaches suffer from accuracy issues. Therefore, in [26], a hybrid form of mapping DSE has been proposed that combines simulation with analytical estimations to prune the design space in terms of application mappings that need to be evaluated using simulation. To this end, the DSE technique uses an analytical model that estimates the expected throughput of an application given a certain architectural configuration and application-to-architecture mapping. In the majority of the search iterations of the DSE process, this analytical throughput estimation avoids the use of simulations to evaluate the design points. However, since the analytical estimations may in some cases be less accurate, the analytical estimations still need to be interleaved with simulative evaluations in order to ensure that the DSE process is steered into the right direction. A similar approach is taken in [27], where an iterative DSE methodology is proposed exploiting the statistical properties of the design space to infer, by means of an empirical analytic model, the design points to be analyzed with low-level simulations. The knowledge of a few design points is used to predict the expected improvement of unknown configurations. Alternatively, in hierarchical DSE (e.g., [28], [29], and [30]), DSE is first performed using analytical or symbolic models to quickly find the interesting parts in the design space, after which simulation-based DSE is performed to more accurately search for the optimal design points.

Workload models: Static versus dynamic

The DSE techniques discussed so far focus on the evaluation and exploration of MPSoC architectures under static, single-application workloads. Todays MPSoC systems, however, 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. Another example would be a video application that dynamically lowers its resolution to decrease its computational demands in order to save battery life. As a consequence, the behavior of application workloads executing on the embedded system can change dramatically over time.

To capture the dynamism in application workload behavior during the design process, this section describes the concept of application scenarios [31] as well as scenario-based DSE [32], [33]. Like in the previous section, we will again use the example of application mapping exploration to illustrate the concepts. Application scenarios are able to describe the dynamism of embedded applications and the interaction between the different applications on the embedded system. An application scenario consists of two parts: an inter-application scenario and an intra-application scenario. An inter-application scenario describes the interaction between multiple applications, i.e., which applications are concurrently executing at a certain moment in time. Inter-application scenarios can be used to prevent the overdesign of a system. If some of the applications cannot run concurrently, then there is no need of reserving resources for the situation where these applications are running together. Intra-application scenarios, on the other hand, describe the different execution modes for each individual application.

The number of different application scenarios grows exponentially with the number of applications involved. So, to perform DSE with these application scenarios, this so-called scenario-based DSE needs to solve the problem that the number of possible application scenarios is too large to exhaustively evaluate the fitness of design points with all of these scenarios. Therefore, a small but representative subset of scenarios must be selected for the evaluation of MPSoC design points. This representative subset must be used for comparing mappings and should lead to the same performance ordering as would have been produced when the complete set of the application scenarios would have been used. That is, if mapping *m1* is better than mapping *m2*, the representative subset should be able to give a better predicted fitness to mapping *m1* than it assigns to mapping *m2*. However, the selection of such a representative subset is not trivial [34]. This is because the representative subset is dependent on the current set of mappings that are being explored. Depending on the set of mappings, a different subset of application scenarios may reflect the relative mapping qualities of the majority of the application scenarios.

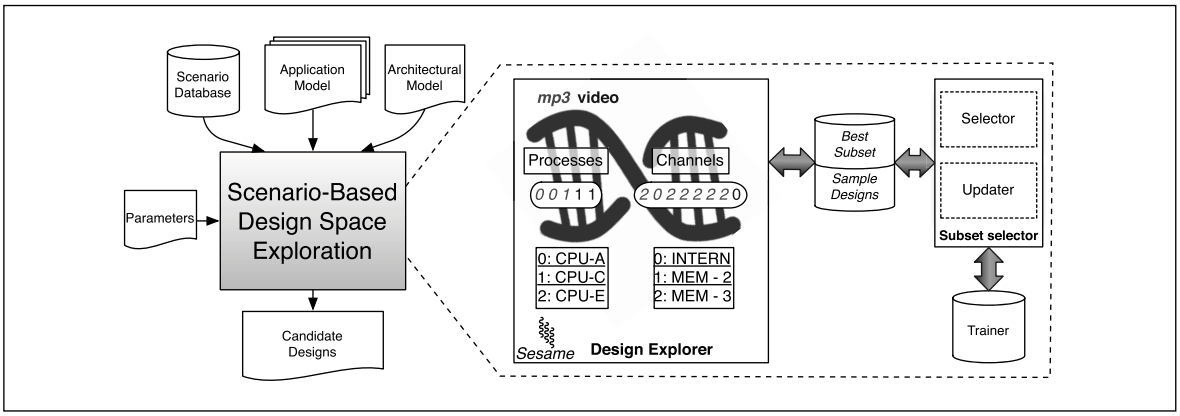

Figure 7. The exploration framework for scenario-based DSE.

As a result, the representative subset cannot statically be selected. For a static selection, one would need to have a large fraction of the mappings that are going to be explored during the MPSoC DSE. However, since these mappings are only available during DSE, a dynamic selection method must be used. Thus, both the set of optimal mappings and the representative subset of scenarios need to be coexplored simultaneously such that the representative subset is able to adapt to the set of mappings that are currently being explored. Figure 7 shows the scenario-based DSE framework. The left part of the picture provides a general overview of the exploration flow, whereas the right part illustrates the scenario-based DSE in more detail. As an input, the scenario-based DSE requires a scenario database, application models, and an MPSoC platform architecture model. The description of the application workload is split into two parts: 1) the structure and 2) the behavior. The structure of applications is described using application models (as described before), whereas a scenario database [35] explicitly stores all the possible multiapplication workload behaviors in terms of application scenarios (i.e., intra-application and inter-application scenarios). In the scenario-based DSE framework, two separate components are recognized that simultaneously perform the coexploration tasks: the design explorer searches for the set of optimal mappings while the subset selector tries to select a representative subset of scenarios. To this end, they exchange data in an asynchronous fashion after every search iteration. Here, the design explorer sends a sample of the current mapping population to the subset selector, whereas the subset selector makes the most representative subset available for the fitness prediction in the design explorer.

The design explorer performs a traditional mapping DSE using a GA, as discussed in the previous section. As explained above, it uses a representative subset of scenarios to evaluate the fitness of mapping solutions. At every iteration of the GA, the design explorer reads in the most recent representative scenario subset from the subset selector and submits the current population of mapping solutions to the subset selector in order to allow the latter to select the appropriate representative subset. This subset selection is not trivial as there are many scenarios to pick from, leading to a huge number of possible scenario subsets. Therefore, the subset selector uses the set of mappings it regularly receives from the design explorer to train the scenario subset such that it is representative for the current population in the design explorer. As the population of the design explorer slowly changes over time, the representative subset will change accordingly. In [33], three different techniques for selecting a representative scenario subset are presented and evaluated: a GA-based scenario space search (which means that two GAs are running concurrently, one for the design explorer and one for the subset selector), a feature selection (FS)-based search algorithm, and a hybrid combination (HYB) between these two. The latter aims at combining the strengths of both the GA-based and FS-based searches. That is, a GA is capable of quickly exploring the space of potential scenario subsets, but due to its stochastic nature, it is susceptible to missing the optimal scenario subsets. This is not the case with the feature selection algorithm as it more systematically explores the local neighborhood of a scenario subset.

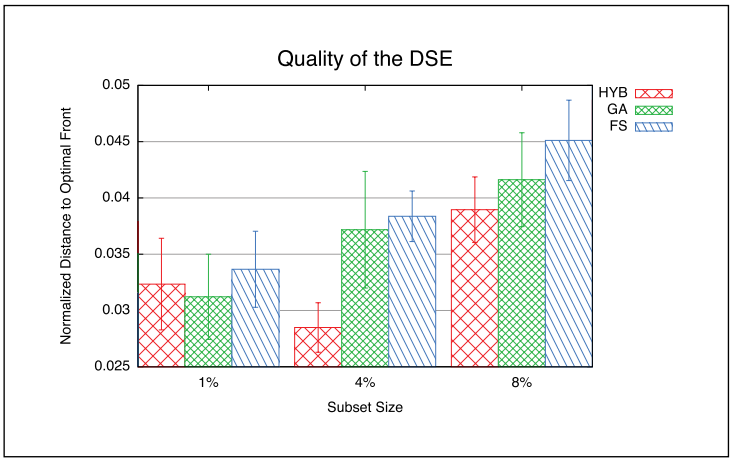

Figure 8. Quality of the DSE for the different subset selection approaches. The quality is determined based on the distance between the estimated Pareto front and the optimal front.

To give a feeling of the performance of the three different fitness prediction techniques, Figure 8 shows the results of a scenario-based DSE experiment in which the three techniques are compared for three different scenario subset sizes: 1%, 4%, and 8% of the total number of application scenarios. In this experiment, the mapping of ten applications with a total of 58 tasks and 75 communication channels is explored. The multiapplication workload consists of 4607 different application scenarios in total. The target platform is a heterogeneous MPSoC with four general-purpose processors, two ASIPs and two ASICs, all connected using a crossbar network. In this experiment, a DSE with a fixed duration of 100 min is performed for all three subset selector approaches. The results have been averaged over nine runs. To evaluate the fitness of mapping solutions, we have again deployed the Sesame MPSoC simulation framework (see the Simulative fitness evaluation section). To determine the efficiency of the multiobjective DSE, we obtain the distance of the estimated Pareto front (execution time versus energy consumption of mapping solutions) to the optimal Pareto front. For this purpose, we normalized execution time and energy consumption to a range from 0 to 1. As the optimal Pareto front is not exactly known since the design space is too large to exhaustively search it, we have used the combined Pareto front of all our experiments for this.

The size of the scenario subset provides a tradeoff between accuracy and convergence of the search. That is, a larger scenario subset will lead to a more accurate fitness prediction of mappings in the design explorer at the cost of a larger computational overhead to obtain the fitness of a single mapping causing a slower convergence of the search. This can be seen in Figure 8. The GA and the FS subset selection methods have worse results when the subset becomes larger (remember that we use a fixed DSE duration of 100 min). For a subset size of 4%, the hybrid selector is, however, still able to benefit from a subset with a higher accuracy. The slower convergence only starts to effect the efficiency for the 8% subset. Comparing the different methods, the hybrid method shows 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 GA.

In this article, we have presented various aspects of the state of the art in embedded systems DSE. Here, we have organized our discussion along the lines of the two primary elements of DSE: the evaluation of single design points and the search strategy for covering the design space. For the coming years, there are still many open research challenges for this domain. Just to give a few examples, first, embedded systems more and more need to become adaptive systems due to increasingly dynamic application workload behavior (as was previously discussed), the need for QoS management to dynamically trade off different system qualities such as performance, precision, and power consumption, and the fact that we have reached a technology level where our circuits are no longer fully reliable, increasing the chances of transient and permanent faults. This calls for research to take system adaptivity, in which a system can continuously customize itself at runtime according to the application workload at hand and the state of the system (e.g., [5] and [36]), into account in the process of DSE.

Second, the trend toward cyberphysical systems and the IoT makes the process of DSE even more complicated since DSE in this context requires taking the behavior of the physical environment (including user behavior) into account. This calls for renewed research into the speed-accuracy tradeoff for the different models and their possible co-simulation applied in DSE for this domain.

A FINAL RESEARCH direction involves the introduction of new design objectives in the process of (early) DSE, in addition to the traditional objectives such as system performance and power/energy consumption. Arguably, a good example is the need for taking system security into account as an optimization objective. As embedded systems are becoming increasingly ubiquitous and interconnected, they attract a worldwide attention of attackers, which makes the security aspect more important than ever during the design of those systems. Currently, system security is still mostly considered as an afterthought and typically is not taken into account during the very early design stages. However, any security measure that may eventually be taken later in the design process does affect the already established tradeoffs with respect to the other system objectives such as performance, power/energy consumption, cost, etc. Thus, covering the security aspect in the earliest phases of design is necessary to design systems that are, in the end, optimal with regard to all system objectives. However, this poses great difficulties because unlike the earlier mentioned conventional system objectives like performance and power consumption, security is hard to quantify. This necessitates research on techniques that make it possible to incorporate security as an objective in early DSE. ■

References

[1] W. Wolf, A. A. Jerraya, and G. Martin, "Multiprocessor system-on-chip (MPSoC) technology," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 10, pp. 1701–1713, Oct. 2008.

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

[3] A. Sangiovanni-Vincentelli and G. Martin, "Platform-based design and software design methodology for embedded systems," *IEEE Design Test Comput.*, vol. 18, no. 6, pp. 23–33, Nov./Dec. 2001.

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

[5] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *Proc. Design Autom. Conf.*, Jun. 2013, pp. 1–10.

[6] M. Thompson, "Tools and techniques for efficient system-level design space exploration," Ph.D. dissertation, Univ. Amsterdam, Amsterdam, The Netherlands, Jan. 2012.

[7] N. Binkert et al., "The gem5 simulator," *ACM SIGARCH Comput. Architecture News*," vol. 39, no. 2, pp. 17, May 2011.

[8] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, "A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies," in *Proc. Int. Symp. Comput. Architect.*, Jun. 2008, pp. 51–62.

[9] S. Li et al., "The mcpat framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing," *ACM Trans. Architect. Code Optim.*, vol. 10, no. 1, p. 5, 2013.

[10] F. Bellard, "Qemu, a fast and portable dynamic translator," in *Proc. USENIX Annu. Tech. Conf.*, Apr. 2005, pp. 41–46.

[11] L. Cai and D. Gajski, "Transaction level modeling: An overview," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synthesis*, Oct. 2003, pp. 19–24.

[12] O. Bringmann et al., "The next generation of virtual prototyping: Ultra-fast yet accurate simulation of hw/sw systems," in *Proc. Int. Conf. Design Autom. Test Eur.*, Mar. 2015, pp. 1698–1707.

[13] A. Butko et al., "A trace-driven approach for fast and accurate simulation of manycore architectures," in *Proc. Asia South Pacific Design Autom. Conf.*, Jan. 2015, pp. 707–712.

[14] J. Castrillon et al., "Trace-based KPN composability analysis for mapping simultaneous applications to MPSoC platforms," in *Proc. Conf. Design Autom. Test Eur.*, Mar. 2010, pp. 753–758.

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

[16] B. Kienhuis, F. Deprettere, P. van der Wolf, and K. Vissers, "A methodology to design programmble embedded systems: The y-chart approach," *Embedded Processor Design Challenges*, ser. Lecture Notes in Computer Science, Berlin, Germany: Springer-Verlag, 2002, vol. 2268, pp. 18–37.

[17] L. Eeckhout, *Computer Architecture Performance Evaluation Methods* (Synthesis Lectures on Computer Architecture). San Rafael, CA, USA: Morgan Claypool Publishers, 2010.

[18] R. Niemann and P. Marwedel, "An algorithm for hardware/software partitioning using mixed integer linear programming," *Design Autom. Embedded Syst.*, vol. 2, no. 2, pp. 165–193, 1997.

[19] M. Lukasiewycz, M. Glass, C. Haubelt, and J. Teich, "Efficient symbolic multi-objective design space exploration," in *Proc. Asia South Pacific Design Autom. Conf.*, Mar. 2008, pp. 691–696.

[20] S. Padmanabhan, Y. Chen, and R. D. Chamberlain, "Optimal design space exploration of streaming applications," in *Proc. IEEE Int. Conf. Appl.-Specific Syst. Architect. Process.*, Sep. 2011, pp. 227–230.

[21] M. Palesi and T. Givargis, "Multi-objective design space exploration using genetic algorithms," in *Proc. Int. Symp. Hardw./Softw. Codesign*, 2002, pp. 67–72.

[22] C. Erbas, S. Cerav-Erbas, and A. D. Pimentel, "Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design," *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 358–374, Jun. 2006.

[23] D. Beasley, D. R. Bull, and R. R. Martin, "An overview of genetic algorithms: Part I—Fundamentals," *Univ. Comput.*, vol. 15, no. 2, pp. 58–69, 1993.

[24] M. Thompson and A. D. Pimentel, "Exploiting domain knowledge in system-level MPSoC design space exploration," *J. Syst. Architect.*, vol. 59, no. 7, pp. 351–360, Aug. 2013.

[25] W. Quan and A. D. Pimentel, "Towards exploring vast mpsoc mapping design spaces using a bias-elitist evolutionary approach," in *Proc. Euromicro Digital Syst. Design Conf.*, Aug. 2014, pp. 655–658.

[26] R. Piscitelli and A. D. Pimentel, "Design space pruning through hybrid analysis in system-level design space exploration," in *Proc. Int. Conf. Design Autom. Test Eur.*, Mar. 2012, pp. 781–786.

[27] G. Mariani, A. Brankovic, G. Palermo, J. Jovic, V. Zaccaria, and C. Silvano, "A correlation-based design space exploration methodology for multi-processor systems-on-chip," in *Proc. Design Autom. Conf.*, 2010, pp. 120–125.

[28] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis, "Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation," in *Proc. LCTES+SCOPES*, 2002, pp. 18–27.

[29] Z. J. Jia, T. Bautista, A. Núñez, M. Thompson, and A. D. Pimentel, "A system-level infrastructure for multidimensional MP-SoC design space co-exploration," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 15, p. 27, 2013.

[30] Z. J. Jia, A. Núñez, T. Bautista, and A. D. Pimentel, "A two-phase design space exploration strategy for system-level real-time application mapping onto MPSoC," *Microprocess. Microsyst.*, vol. 38, no. 1, pp. 9–21, 2014.

[31] S. V. Gheorghita et al., "System-scenario-based design of dynamic embedded systems," *ACM Trans. Design Autom. Electron. Syst.*, vol. 14, no. 1, pp. 1–45, 2009.

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

[33] P. van Stralen and A. Pimentel, "Fitness prediction techniques for scenario-based design space exploration," *IEEE Trans. Comput.-Aided Design Integr.*, vol. 32, no. 8, pp. 1240–1253, Aug. 2013.

[34] P. van Stralen, "Applications of scenarios in early embedded system design space exploration", Ph.D. dissertation, Univ. Amsterdam, Amsterdam, The Netherlands, Jan. 2014.

[35] 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 Comput. Syst. Architect. Model. Simul.*, Jul. 2010, pp. 11–19.

[36] W. Quan and A. D. Pimentel, "A hybrid task mapping algorithm for heterogeneous MPSoCs", *ACM Trans. Embedded Comput. Syst.*, vol. 14, no. 1, p. 14, Jan. 2015.

Andy D. Pimentel is an Associate Professor at the System and Network Engineering Lab, University of Amsterdam, Amsterdam, The Netherlands. His research centers around system-level modeling, simulation, and exploration of (embedded) multicore and manycore computer systems with the purpose of effectively designing and programming these systems. Pimentel has a PhD in computer science from the University of Amsterdam. He is a cofounder of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS). He has (co)authored more than 100 scientific publications and is an Associate Editor of Elseviers *Simulation Modelling Practice and Theory* as well as Springers *Journal of Signal Processing Systems*. He served as the General Chair of HIPEAC'15, as Local Organization Co-Chair of ESWeek'15, and he serves as Program (Vice-)Chair of CODES+ISSS in 2016 and 2017. Furthermore, he has served on the TPC of many leading (embedded) computer systems design conferences, such as DAC, DATE, CODES+ISSS, ICCD, ICCAD, FPL, SAMOS, and ESTIMedia.

■ Direct questions and comments about this article to Andy D. Pimentel, Institute of Informatics, University of Amsterdam, 1098XH Amsterdam, The Netherlands; a.d.pimentel@uva.nl.