Methodologies for Design Space Exploration

Andy D. Pimentel

Abstract In this chapter, we present an overview of techniques and methods for the design space exploration (DSE) of embedded systems. DSE is the critical design process in which system designs are modeled, evaluated and, eventually, optimized for the various extra-functional system behaviors, such as performance, power or energy consumption and cost. We organize our discussion along the lines of the two primary elements of DSE, namely the evaluation of single design points and the search strategy for covering the design space.

Key words: Design Space Exploration, Multi-objective Optimization, Performance Modeling, Genetic Algorithms

1 Introduction

Designers of modern embedded computer 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. Mission- and safety-critical embedded computer systems, like those in the avionics and space domains, usually also demand high levels of dependability, which is becoming even more important as the levels of system autonomy rise. Moreover, a great deal of these systems must, increasingly, support multiple applications and standards for which they often need to provide real-time performance. For example, mobile devices must support a variety of different standards for communication and coding of digital contents. In addition, many of these systems also need to provide a high degree of flexibility, allowing them to be easily updated

Andy D. Pimentel
Parallel Computing Systems group, University of Amsterdam, Science Park 904, 1098 XH, Amsterdam, The Netherlands, e-mail: a.d.pimentel@uva.nl

and extended with future applications and standards. This 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* multi-processor 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 Multi-Processor System-on-Chip (MPSoC) architectures [76].

To cope with the design complexity of such systems, we have witnessed the emergence of a new design methodology in the past 15 to 20 years, called system-level design (see Chapter on Electronic system-level 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 MPSoC platform architectures to facilitate re-use of IP components and the concept of high-level system modeling and simulation [28, 57]. The latter allows for capturing the behavior of platform components and their interactions at a high 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) [24, 45]. 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. It is of paramount importance to start performing such DSE as early as possible in the design process because the considered design choices may heavily influence the success or failure of the final product. However, the process of DSE also is highly challenging since 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., system performance or power consumption – exponentially grows with the number of application tasks and processors in the system, and is known to be an NP-hard problem [60]. Therefore, the development of efficient and effective DSE methods has received significant research attention in recent years. In this chapter, we will provide an overview of the various aspects involved in embedded systems DSE.

2 Design Space Exploration: the 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 multi-objective DSE [45]. 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 trade-offs between design criteria.

Given a set of m decision variables, which are the degrees of freedom (e.g., MP-SoC 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:

$$f_i : R^m \rightarrow R^1 \tag{1}$$

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 i-th 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). As illustrated in Figure 1, the combined fitness function $f(x)$ subsequently translates a point in the solution space into the objective space Y. Formally, a multi-objective 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 $1 \leq i \leq n$:

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

Here, the decision variables x_i (with $1 \leq i \leq m$) usually are constrained. These constraints make sure that the decision variables refer to valid system configurations (e.g., using not more than the available number of processors, using a valid mapping of application tasks to processing resources, etc.), i.e., x_i are part of the so-called feasible set. 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 objec-

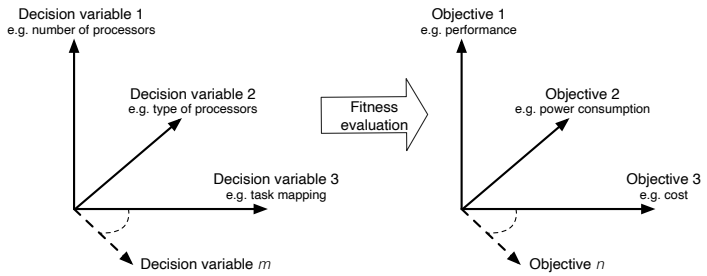

Fig. 1 The design space broken down in solution and objective space.

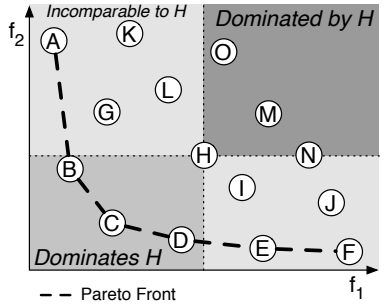

Fig. 2 A Pareto front and an example of the dominance relation.

tives, however, the comparison becomes non-trivial. Take, for example, two different MPSoC designs: a high-performance MPSoC and a slower but much cheaper one. 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 is better or the low-cost MPSoC. 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 generally used. Here, a solution $x_a \in X$ is said to dominate solution $x_b \in X$ if and only if $x_a < x_b$:

$$x_a < x_b \iff \forall i \in \{1, 2, ..., n\} : f_i(x_a) \leq f_i(x_b) \wedge \\ \exists i \in \{1, 2, ..., n\} : f_i(x_a) < f_i(x_b)$$

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

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

The Pareto dominance relation only provides a partial ordering. For example, the solutions A to F of the example in Figure 2 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 non-dominated solutions, called the *Pareto front*. A set X' is defined to be a Pareto front of the set of solutions X as follows:

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

The Pareto front of Figure 2 contains six solutions: $A - F$. Each of these solutions does not dominate the other. An improvement on objective f_1 is matched by a worse value for f_2. Generally, it is up to the designer to decide which of the solutions provides the best trade-off.

2.1 Two basic ingredients of DSE

The search for Pareto optimal design points with respect to multiple design criteria as targeted by DSE entails two distinct elements [24, 45]:

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.
2. The search strategy for covering and navigating through the design space, spanned by the decision variables x_i (with $1 \leq i \leq m$), during the DSE process.

Figure 3 shows a simple taxonomy for DSE approaches, recognizing the above two DSE elements as well as different properties of these DSE elements. We note that these properties typically cannot be considered in pure isolation as they can be interdependent and even conflicting with each other. As will be discussed in more detail later on, there usually exists a trade-off 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 aspect of DSE, the confidence property 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

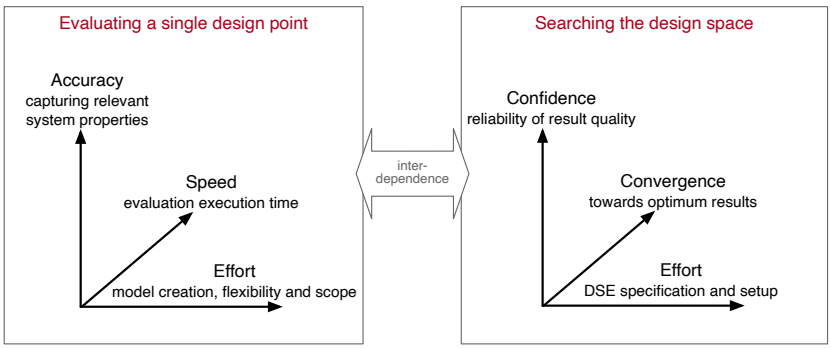

Fig. 3 A taxonomy for DSE approaches (taken from [68]).

in the design space is evaluated and compared would provide a 100% confidence. However, such exhaustive search is usually prohibited due to the sheer size of the design space. In those cases, as will be discussed later on, search techniques based on metaheuristics 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.

2.2 Y-chart based DSE

Many system-level fitness evaluation and DSE methods and tools in the embedded systems domain are based on the Y-chart methodology [30, 1], which is illustrated in Figure 4. This implies that these DSE methods separate application models (or workload models) and architecture models while also recognizing an explicit mapping step to map application tasks onto architecture resources (i.e., bind tasks to processing elements in space and time). In this approach, an application model – derived from a specific application domain – describes the functional behavior of an application workload in a timing and architecture independent manner. An MPSoC (platform) architecture model – which usually has been defined with the application domain in mind – defines architecture resources and captures their extra-funtional behavior, i.e. behavior in terms performance, power consumption, cost, etc. To perform quantitative analysis of the fitness of a design point, application models are mapped onto the architecture model under investigation, after which the fitness of

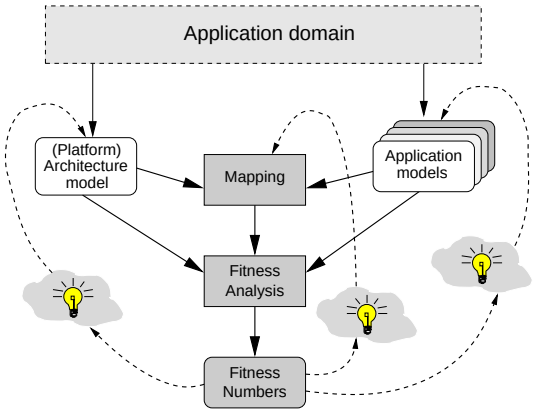

Fig. 4 Y-chart based design space exploration [30, 1].

each application-architecture combination can be evaluated. Subsequently, the resulting fitness numbers may be used by the search algorithm of a DSE process to change the architecture, restructure/adapt the application(s) or modify the mapping of the application(s). These actions are illustrated by the light bulbs in Figure 4.

Essential in this methodology is that an application model is independent from architectural specifics, assumptions on hardware/software partitioning, and timing characteristics. As a result, application models can be re-used in the exploration cycle. For example, a single application model can be used to exercise different hardware/software partitionings or can be mapped onto a range of architecture models, possibly representing different MPSoC architecture designs or modeling the same architecture design at various levels of abstraction. With the latter, we refer to the gradual refinement of architecture models (e.g., [46, 72]). As design decisions are made, a designer typically wants to descend in abstraction level by disclosing more and more implementation details in an architecture model. Eventually, such refinement can bring an initially abstract architecture model closer to the level of detail where it is possible to synthesize an implementation [40, 69, 61].

In the next two sections, we will provide a more detailed overview of the different techniques, and their properties, applied in each of the two aforementioned elements of DSE, i.e., fitness evaluation of a single design point and searching the design space.

3 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 an 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 (see also Chapter on Processor Simulation and Characterisation). This trade-off 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 usually use simulation or analytical models to evaluate single design points, the remainder of this section will focus on these methods.

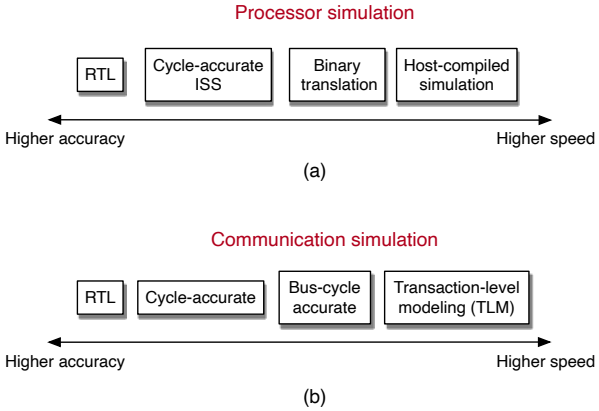

Fig. 5 Different levels of abstraction for (a) simulating processors and (b) simulating communication.

3.1 Simulative fitness evaluation

Simulating system components can 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 trade-off is shown in Figure 5. 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.

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 simulation as compared to RTL simulation at the cost of a somewhat reduced accuracy since the system state in between cycles is not accounted for. Cycle-accurate simulation is a popular technique for simulating microprocessors (see also Chapter on Processor Simulation and Characterisation): so-called *cycle-accurate instruction set simulation*

(ISS). These ISS simulators try to capture the cycle-by-cycle behavior of the micro-architectural 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 micro-architecture components based on their activity ratio. A good example is the widely-used cycle-accurate Gem5 ISS [4], which can be extended to also support area and power predictions using activity-based modeling frameworks such as CACTI [73] and McPAT [31]. Although these ISS simulators can be deployed to perform micro-architectural DSE for processor components, they are generally still too slow for performing full system-scale DSE of multi-core based embedded systems.

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 for example deployed in the widely-used QEMU simulator [3], 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. In some cases, however, ISS simulators and especially binary-translation based simulators only focus on mimicking the functional behavior and do not capture the extra-functional behavior of the target processor. In these cases, they are usually referred to as *emulators* rather than simulators.

For simulating communication between system components, one could use so-called *bus-cycle accurate simulation* [7] 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* [10, 5]. 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 this type of simulation is efficient as it directly executes target programs on the host computer, it is very suitable for system-level DSE. However, at this level of abstraction it is difficult to accurately capture intricate micro-architectural 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 communications, *transaction-level modeling (TLM)* [7] 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., [6, 9]). These trace events can focus on the evaluation of specific system elements such as memory access address traces for cache simulation [74]. 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, communications) 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 re-used 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 [46, 15]. Sesame is based on the aforementioned Y-chart methodology [30], 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 of extra-functional system behavior (performance, power consumption, etc.). Figure 6 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, which can be seen as an abstract (real-time) operating system (RTOS) model, 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 [13]. 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

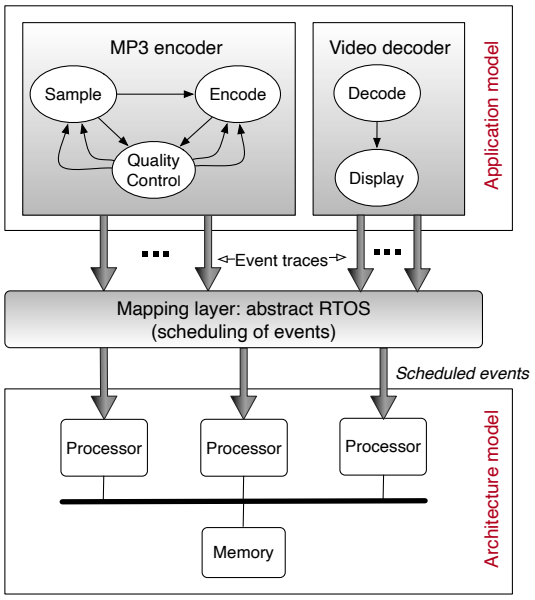

Fig. 6 The Sesame system-level MPSoC simulation infrastructure.

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 simulation can converge to a set of performance predictions fairly quickly.

3.2 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 and 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 simulation runs.

Analytical models can roughly be divided into three classes [13]: 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 micro-architectural 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).

4 Searching the design space

As explained before, searching a design space is a multi-objective optimization process. This process will evidently benefit from a good trade-off 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 trade-off 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 towards this goal is the search algorithm that navigates through the design space towards 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 3: confidence, convergence and effort. As was already mentioned earlier, 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 trade-off between these concerns.

DSE search algorithms can be divided into *exact* and *non-exact* methods. In exact DSE methods, like those implemented using integer linear programming (ILP) solutions (e.g., [39, 34]) or branch & bound algorithms (e.g., [41]), the optimum is guaranteed to be found. As such methods generally are compute intensive, they typi-

cally use design space pruning (i.e., discarding unsuitable design points) to optimize the efficiency of the search, thereby allowing them to handle larger design spaces. However, for realistic design problems with design spaces that are vast, these methods may still not scale and thus be less suited. Alternatively, in non-exact methods, *metaheuristics* are typically 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 [43]. In this chapter, 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., [42, 35, 14, 50, 21]) and has 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.

4.1 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* [2]. 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 [60]. 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

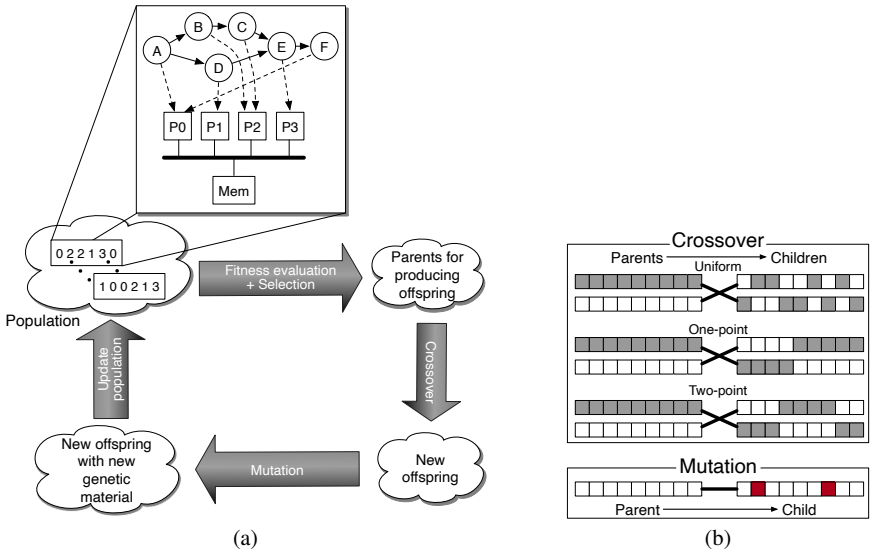

Fig. 7 GA-based mapping DSE: (a) general overview of the GA steps, and (b) crossover and mutation operators.

topological communication 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 (i.e., particular processor(s) not in use) or contain all n tasks (i.e., a single processor system). A processor that is not assigned any tasks (having an empty task partition) can be considered idle or non-existent.

In Figure 7(a), the different steps of a GA are shown. This figure also illustrates the mapping representation of a chromosome for an application with 6 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 are first evaluated. To this end, any of the analytical or simulative techniques discussed in Section 3 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 7(b). There exist various forms of this crossover operator, of which the uniform, one-point, 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 7(b). 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-core MPSoC architecture with a crossbar interconnect [68]. The mapping design space contains more than 4 million design points. Of these design points, 175K are unique ones since the target platform is a homogeneous, symmetric MPSoC, and as a consequence, exhibits mapping symmetries. 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 [46, 15] (see also Section 3.1). Figure 8 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 P-Q (Probability-Quality) plot. Regarding the top part of this plot, the horizontal axis (top x-axis) represents the quality of the result as a percentile towards 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

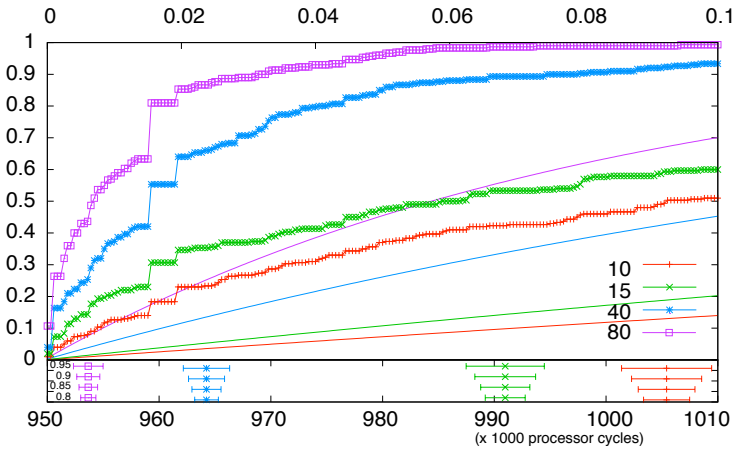

Fig. 8 P-Q plot for GA-based DSE with different population sizes.

confidence intervals, shown at the bottom of the graph in Figure 8, 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 non-overlapping, the more significant the difference of the mean behavior (which is clearly the case in the example of Figure 8). 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 $\frac{175K}{1000} = 175$ 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 8, a population size of 40 may therefore provide a good compromise.

4.2 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 [71], new GA-operators have been proposed that optimize the search performance by 1) reducing the redundancy present in chromosome representations (e.g., mapping symmetries [22] 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 [50], 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 [48], 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 simulation 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 [49]. A similar approach is taken in [36], 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 simulation. The knowledge of a few design points is used to predict the expected improvement of unknown configurations.

Alternatively, in hierarchical DSE (e.g., [38, 26, 27]), DSE is first performed using analytical or symbolic models to quickly find the interesting parts in the design space. Hereafter, simulation-based DSE is performed on the selected sweet spots in the design space to more accurately search for the optimal design points.

5 Multi-application workload models

The DSE techniques discussed so far focus on the evaluation and exploration of MPSoC architectures under static, single-application workloads. Today's 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 [70, 8]. 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. Or a video application could dynamically lower 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.

As illustrated in Figure 9, there are several approaches for dealing with multi-application workloads in the context of DSE. A commonly-used approach is to consider the applications in isolation, as illustrated in Figure 9(a). This implies that each of the applications in the multi-application workload will be mapped to a different, isolated part of the system. As a consequence, the DSE for each of these applications can also be performed in isolation. However, this approach typically leads to over-designed systems since there is no or limited resource sharing between applications. Another approach, illustrated in Figure 9(b), makes the pessimistic assumption that all applications that can be executed on the system will always be active (and will thus be contending for system resources). Again, performing DSE with such an assumption may lead to highly over-designed systems, as in

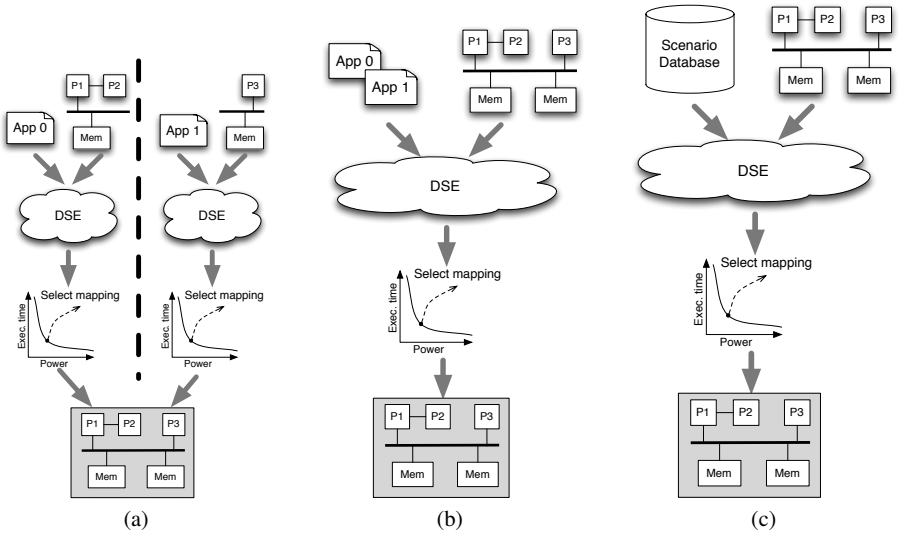

Fig. 9 DSE for multi-application workloads on a 3-core, bus-based MPSoC: (a) DSE with application isolation, (b) pessimistic DSE, and (c) scenario-based DSE.

reality the concurrent activation of all possible applications may be unlikely. To address the problem of over-designing systems and to capture the dynamism in application workload behavior during the design process, the DSE could employ the concept of application scenarios [17], leading to scenario-based DSE [63, 66, 47, 8]. This is illustrated in Figure 9(c). The remainder of this section will discuss the concepts of application scenarios and scenario-based DSE, again using the example of application mapping exploration for illustration purposes.

5.1 Scenario-based DSE

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- 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 over-design 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 concept of application scenarios, inter-application scenarios and intra-application scenarios is illustrated in Figure 10 for three multi-

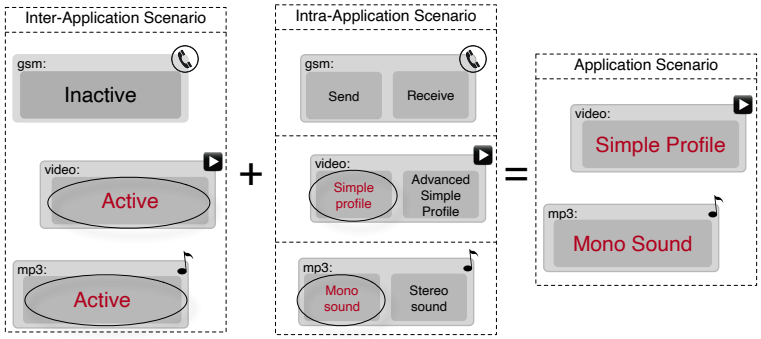

Fig. 10 Application scenarios, inter-application scenarios and intra-application scenarios.

media applications (mp3 player, video decoder, and gsm application) with each two application modes.

The number of different application scenarios grows exponentially with the number of applications involved. So, to perform DSE with these application scenarios, scenario-based DSE needs to solve the problem that the total 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 application 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 [47]. 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.

As a result, the representative subset cannot be statically 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 *co-explored simultaneously* such that the representative subset is able to adapt to the set of mappings that are currently being explored [63, 66, 47]. Figure 11 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 input, the scenario-based DSE requires a database of application scenarios, 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

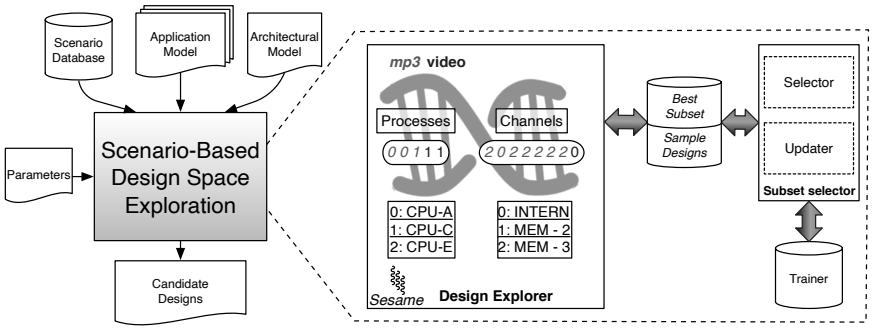

Fig. 11 The exploration framework for scenario-based DSE.

before), whereas a scenario database [64] explicitly stores all the possible multi-application workload behaviors in terms of application scenarios (i.e., intra- and inter-application scenarios). In the scenario-based DSE framework, two separate components are recognized that simultaneously perform the co-exploration 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 Section 4. 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 [66], 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) of these two. The latter aims at combining the strengths of both the GA-based and feature selection 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.

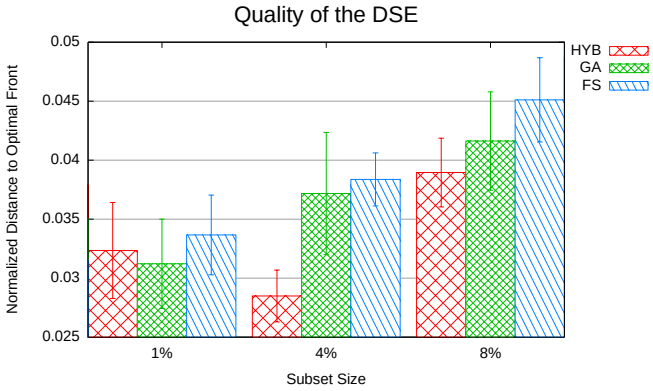

Fig. 12 Quality of the scenario-based DSE for the different subset selection approaches. The quality is determined based on the distance between the estimated Pareto front and the optimal front.

To give a feeling of the performance of the three different fitness prediction techniques, Figure 12 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 multi-application 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 minutes 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 Section 3.1). To determine the efficiency of the multi-objective 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 the 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 trade-off 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 12. 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 minutes). 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.

6 Application exploration

As described in Section 2.2, the Y-chart methodology is a popular approach for system-level DSE in the domain of embedded systems. This means that exploration can take place to investigate (the fitness of) different MPSoC architectures, different mappings of application tasks to the underlying architecture, but also different application implementations. With respect to the latter, one could, for example, explore the use of different algorithms for implementing a certain piece of application functionality or vary the degree of concurrency in an application (e.g., fine-grained versus more coarse-grained concurrency) that can be exploited by an underlying MPSoC platform. So far, we have focused on the first two types of exploration, i.e., exploring different MPSoC architectures and task-to-architecture mappings. In this section, we will describe an example of application exploration, and do this for the popular application domain of deep learning. More specifically, we will outline an approach for so-called Neural Architecture Search (NAS) [58, 59], which automates the discovery of an efficient neural network for a given task, such as image/video recognition, classification, natural language processing, etc.

6.1 NAS by means of Evolutionary Piecemeal Training

The NAS approach that we will describe, searches for an efficient Convolutional Neural Network (CNN) architecture. To this end, it leverages a genetic algorithm (GA), which allows a group of candidate CNNs in the GA's population to train in parallel. In most NAS techniques, training of a neural network is considered a separate task or a performance estimation strategy to perform the neural network architecture search. However, the approach described here considers NAS from a different perspective as it aims at finding optimal CNN architectures *during the training process itself* as opposed to accuracy prediction or training as a separate performance estimation strategy. The NAS approach is called Evolutionary Piecemeal Training (EPT) [58, 59], where piecemeal-training refers to training a neural network with a small 'data-piece' of size δ_k. In this technique, a traditional continuous training is interceded by an evolutionary operator at regular intervals and the interval of intervention is dictated by the value of δ_k. The evolutionary operators applied to CNNs in the GA population lead to CNN architecture modifications and hence exploration of the search space. A new CNN architecture derived like this is always partially trained already as it was modified from another CNN undergoing

training. In subsequent iterations, derived CNN architectures continue to train. Those CNN candidates that are not able to achieve high accuracy during training will be dropped from the population. This can also be seen as early training termination of candidates that are performing poorly. Towards the end of this algorithm, the best candidates are selected from the population, which can then be post processed or trained further, if needed.

The search space for the algorithm is focused on plain CNNs, which consist of convolutional, fully-connected (FC) and pooling layers without residual connections, branching etc. Batch normalization and non-linear activations are configurable and can be added to the network. CNN architectures are defined by a group of blocks, where each block is of a specific layer type and is bounded by minimum and maximum number of layers it can have. Additionally, each layer has upper and lower bounds on each of its parameters. For example, a convolutional layer will have bounds on the number of units, kernel sizes and stride sizes possible. These constraints are in place to make sure that CNN architectures do not become too big and limits the resource consumption of the final neural network. This is an important factor to consider when mapping the CNNs to resource-constrained embedded systems. The search space specifications along with its bounds are encoded as a collection of genes, also called a genotype. All possible combinations of parameters together form the gene pool from which individual neural networks are created and trained.

A population based training process is used where an initial population of neural networks is randomly created from the defined gene pool. In each iteration, all candidates of the population are piecemeal-trained and then evaluated using the validation set. Depending upon the available resources, all candidates can be trained in any combination of parallel and sequential manner. The size of the population is kept constant throughout the algorithm, though the candidates of the population keep changing as they are altered through the evolutionary operators applied in each iteration. The number of candidates in the population needs to be large enough to maintain enough diversity of CNN architectures in the population, while still satisfying the constraints applied to it.

6.1.1 Evolutionary operators

The crossover operator in the EPT-based NAS works with two neural networks and swaps all layers in a gene-block of the same type. In this replacement, the layers being swapped are roughly in the same phase of feature extraction. The input and output feature map sizes of the layer block being swapped are also identical in both of the selected networks. Figure 13 illustrates the crossover operator for swapping convolutional layers from two networks. Crossover is not a function preserving operator, but in experiments they were found to be important to introduce diversity in the population by changing the total number of layers in a candidate through swapping. To reduce the negative effect of training loss incurred due to the crossover, a cooling-down approach is used to the crossover rate. In earlier GA iterations, where

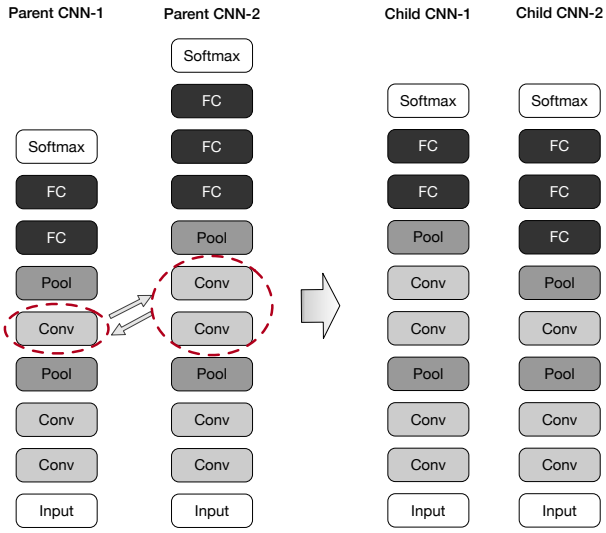

Fig. 13 Crossover operator on two CNNs swapping convolution layers.

the training loss is already high, there are more swaps happening than in the later ones, where training loss is very low.

The mutation operator changes a layer's parameters such as the number of kernels or kernel size and is designed to be function preserving. Every mutation disrupts the ongoing training of the mutated candidates and some additional loss is incurred in the training-in-process. However, due to the function preserving nature of the mutation operator, the loss incurred from this operator is as small as possible and recoverable in later piecemeal-training.

6.1.2 NAS Results

To illustrate the competence and versatility of the EPT-based NAS concept, a range of experiments were performed with datasets from two different domains: CIFAR-10 for image classification [12] and PAMAP2 for human activity recognition [56]. For CIFAR-10, the search took 2-GPU days and the best prediction accuracy was found to be 92.5% on the test set. Table 1 shows comparisons with other evolutionary NAS approaches, where EPT refers to Evolutionary Piecemeal Training. We know that 92.5% is relatively low compared to other published works, but this is on a very simple and plain CNN without any architectural enhancements or advance data augmentation. Other approaches use a hybrid search space where different architecture blocks or cell modules as well as arbitrary skip connections are used. Instead of stacking conventional layers, these stack different blocks. The best model found in the EPT experiments has 13 convolutional layers followed by 2 fully connected layers. For the PAMAP2 dataset, the EPT search took only 10 GPU-hours and the

Table 1 CIFAR-10 Accuracy Comparisons with Evolutionary Approaches

Model	Search Space	GPU-days	Accuracy(%)
CoDeepNeat [37]	hybrid	-	92.7
GeneticCNN [77]	hybrid	17	92.9
EANN-Net [11]	hybrid	-	92.95
AmoebaNet [54]	cell	3150	**96.6**
NSGANet [33]	hybrid	8	96.15
Evolution [55]	hybrid	1000+	94.6
EPT	plain CNN	**2**	92.5

best prediction accuracy was 94.36%. Compared to state of the art neural network solutions for this particular dataset, EPT outperforms all other efforts of which we are aware. The best performance was found on a neural network that has 7 convolutional layers followed by 3 fully connected layers. The interested reader is referred to [58] for a more detailed analysis of EPT's experimental results.

7 Conclusion and outlook

In this chapter, we presented an overview of techniques and methods for DSE of embedded systems. We 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. The overview is certainly not meant to be exhaustive. For example, we focused on popular GA-based DSE, optimizing system performance and, to some extent, power/energy consumption. The optimization of other important design objectives, such as system reliability (e.g., addressed in [25, 19, 18, 65]), has not been covered.

There are still many open research challenges for this domain. For example, embedded systems more and more need to become adaptive systems due to increasingly dynamic application workload behavior (as was previously discussed), the need for Quality-of-Service 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 reconfigure and customize itself at run time according to the application workload at hand and the state of the system (e.g., [60, 51, 53, 52, 20, 29]), into account in the process of DSE. In the case of adaptive systems, a DSE process cannot easily compare different design choices by, e.g., simply evaluating the performance or power/energy consumption of an application workload executing on a specific platform architecture. That is, the reconfiguration behavior (i.e., when and how the system reacts to 'disruptive events' that trigger system reconfigurations) of the system and the performance / power consumption

consequences of such system adaptivity actions, must be taken into account when comparing different design instances. This calls for efficient and effective methods that allow for evaluating and optimizing adaptive embedded system designs such that the way the system instances and their extra-functional behavior evolve over time are also captured.

Another research direction we would like to mention involves the introduction of new design objectives in the process of (early) DSE, in addition to the traditional objectives such as system performance, power/energy consumption, system cost, and reliability. Arguably, a good example is the need for taking system security into account as an optimization objective [44]. As embedded systems are becoming increasingly ubiquitous and interconnected, they attract a world-wide 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 is typically 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 trade-offs 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.

At this moment, the integration of security aspects in the process of system-level DSE of embedded systems is still a largely uncharted research ground. Only a few efforts exist that address this problem but they typically provide only partial solutions or solutions to very specific security problems (e.g., [32, 75, 62, 67]). Moreover, in most of these works, security is modelled as a requirement in the DSE process, which does not allow for studying actual trade-offs between performance, power consumption or cost in relationship to secureness of a design. Only a handful of research efforts, such as [16, 23], seem to have been aiming at incorporating security as an objective that can be traded off with other objectives during early DSE.

References

1. Balarin, F., Sentovich, E., Chiodo, M., Giusto, P., Hsieh, H., Tabbara, B., Jurecska, A., Lavagno, L., Passerone, C., Suzuki, K., Sangiovanni-Vincentelli, A.: Hardware-Software Co-design of Embedded Systems – The POLIS approach. Kluwer Academic Publishers (1997)
2. Beasley, D., Bull, D.R., Martin, R.R.: An overview of genetic algorithms: Part I-fundamentals. University Computing **15**(2), 58–69 (1993)
3. Bellard, F.: Qemu, a fast and portable dynamic translator. In: Proc. of the USENIX Annual Technical Conference, pp. 41–46 (2005)
4. Binkert, N., et al.: The gem5 simulator. SIGARCH Computer Architecture News **39**(2), 1–7 (2011)

5. Bringmann, O., Ecker, W., Gerstlauer, A., Goyal, A., Mueller-Gritschneder, D., Sasidharan, P., Singh, S.: The next generation of virtual prototyping: Ultra-fast yet accurate simulation of hw/sw systems. In: Proc. of the Int. Conference on Design, Automation & Test in Europe (DATE), pp. 1698–1707 (2015)
6. Butko, A., Garibotti, R., Ost, L., Lapotre, V., Gamatie, A., Sassatelli, G., Adeniyi-Jones, C.: A trace-driven approach for fast and accurate simulation of manycore architectures. In: Proc. of the Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 707–712 (2015)
7. Cai, L., Gajski, D.: Transaction level modeling: An overview. In: Proc. of the Int. Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 19–24 (2003)
8. Castrillon, J., Leupers, R., Ascheid, G.: MAPS: Mapping concurrent dataflow applications to heterogeneous MPSoCs. IEEE Transactions on Industrial Informatics **9**(1), 527–545 (2013)
9. Castrillon, J., Velasquez, R., Stulova, A., Sheng, W., Ceng, J., Leupers, R., Ascheid, G., Meyr, H.: Trace-based KPN composability analysis for mapping simultaneous applications to MPSoC platforms. In: Proc. of the Conference on Design, Automation Test in Europe (DATE), pp. 753–758 (2010)
10. Ceng, J., Sheng, W., Castrillon, J., Stulova, A., Leupers, R., Ascheid, G., Meyr, H.: A high-level virtual platform for early MPSoC software development. In: Proc. of the 7th IEEE/ACM international conference on Hardware/software Codesign and System Synthesis (CODES+ISSS) (2009)
11. Chen, Z., Zhou, Y., Huang, Z.: Auto-creation of effective neural network architecture by evolutionary algorithm and resnet for image classification. In: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), pp. 3895–3900. IEEE (2019)
12. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition, pp. 248–255. Ieee (2009)
13. Eeckhout, L.: Computer Architecture Performance Evaluation Methods. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers (2010)
14. Erbas, C., Cerav-Erbas, S., Pimentel, A.D.: Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design. IEEE Trans. on Evolutionary Computation **10**(3), 358–374 (2006)
15. Erbas, C., Pimentel, A.D., Thompson, M., Polstra, S.: A framework for system-level modeling and simulation of embedded systems architectures. EURASIP Journal on Embedded Systems (2007)
16. Ferrante, A., Milosevic, J., Janjšević, M.: A security-enhanced design methodology for embedded systems. In: Proc. of the International Conference on Security and Cryptography (SECRYPT), pp. 1–12 (2013)
17. Gheorghita, S.V., et al.: System-scenario-based design of dynamic embedded systems. ACM Trans. on Design Automation of Electronic Systems **14**(1), 1–45 (2009)
18. Glaß, M., Lukasiewycz, M., Reimann, F., Haubelt, C., Teich, J.: Symbolic reliability analysis and optimization of ecu networks. In: Proc. of the Conference on Design, Automation and Test in Europe, pp. 158–163 (2008)
19. Glaß, M., Lukasiewycz, M., Streichert, T., Haubelt, C., Teich, J.: Reliability-aware system synthesis. In: Proc. of the Conference on Design, Automation Test in Europe, pp. 1–6 (2007)
20. Goens, A., Khasanov, R., Castrillon, J., Hähnel, M., Smejkal, T., Härtig, H.: Tetris: A multi-application run-time system for predictable execution of static mappings. In: Proc. of the 20th International Workshop on Software and Compilers for Embedded Systems (SCOPES), pp. 11–20 (2017)
21. Goens, A., Khasanov, R., Castrillon, J., Polstra, S., Pimentel, A.D.: Why comparing system-level MPSoC mapping approaches is difficult: a case study. In: Proc. of the IEEE 10th Int. Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC) (2016)
22. Goens, A., Siccha, S., Castrillon, J.: Symmetry in software synthesis. ACM Trans. on Architecture and Code Optimimization **14**(2) (2017)
23. Gressl, L., Steger, C., Neffe, U.: Security driven design space exploration for embedded systems. In: Forum for Specification and Design Languages (FDL), pp. 1–8 (2019)

24. Gries, M.: Methods for evaluating and covering the design space during early design development. Integration, the VLSI Journal **38**(2), 131–183 (2004)
25. Jhumka, A., Klaus, S., Huss, S.A.: A dependability-driven system-level design approach for embedded systems. In: Proc. of the Conference on Design, Automation and Test in Europe, pp. 372–377 (2005)
26. Jia, Z.J., Bautista, T., Núñez, A., Thompson, M., Pimentel, A.D.: A system-level infrastructure for multi-dimensional MP-SoC design space co-exploration. ACM Trans. on Embedded Computing Systems **13**(1s) (2013)
27. Jia, Z.J., Núñez, A., Bautista, T., Pimentel, A.D.: A two-phase design space exploration strategy for system-level real-time application mapping onto MPSoC. Microprocessors and Microsystems **38**(1), 9–21 (2014)
28. Keutzer, K., Newton, A., Rabaey, J., Sangiovanni-Vincentelli, A.: System-level design: orthogonalization of concerns and platform-based design. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems **19**(12), 1523–1543 (2000)
29. Khasanov, R., Castrillon, J.: Energy-efficient runtime resource management for adaptable multi-application mapping. In: Proc. of the Design, Automation and Test in Europe Conference (DATE) (2020)
30. Kienhuis, B., Deprettere, F.E., van der Wolf, P., Vissers, K.: A methodology to design programmble embedded systems: the y-chart approach. Lecture Notes in Computer Science - Embedded Processor Design Challenges **2268**, 18–37 (2002)
31. Li, S., et al.: The mcpat framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing. ACM Trans. on Architecture and Code Optimization **10**(1), 5 (2013)
32. Lin, C.W., Zheng, B., Zhu, Q., Sangiovanni-Vincentelli, A.: Security-aware design methodology and optimization for automotive systems. ACM Transactions on Design Automation of Electronic Systems **21**(1) (2015)
33. Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., Banzhaf, W.: Nsga-net: a multi-objective genetic algorithm for neural architecture search. arXiv preprint arXiv:1810.03522 (2018)
34. Lukasiewycz, M., Glass, M., Haubelt, C., Teich, J.: Efficient symbolic multi-objective design space exploration. In: Proc. of the Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 691–696 (2008)
35. Madsen, J., Stidsen, T.K., Kjaerulf, P., Mahadevan, S.: Multi-objective design space exploration of embedded system platforms. In: B. Kleinjohann, L. Kleinjohann, R.J. Machado, C.E. Pereira, P.S. Thiagarajan (eds.) From Model-Driven Design to Resource Management for Distributed Embedded Systems, pp. 185–194. Springer US, Boston, MA (2006)
36. Mariani, G., Brankovic, A., Palermo, G., Jovic, J., Zaccaria, V., Silvano, C.: A correlation-based design space exploration methodology for multi-processor systems-on-chip. In: Proc. of the Design Automation Conference (DAC), pp. 120–125 (2010)
37. Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., et al.: Evolving deep neural networks. In: Artificial Intelligence in the Age of Neural Networks and Brain Computing, pp. 293–312. Elsevier (2019)
38. Mohanty, S., Prasanna, V.K., Neema, S., Davis, J.: Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. In: Proc. of LCTES+SCOPES, pp. 18–27 (2002)
39. Niemann, R., Marwedel, P.: An algorithm for hardware/software partitioning using mixed integer linear programming. Design Automation for Embedded Systems **2**(2), 165–193 (1997)
40. Nikolov, H., Thompson, M., Stefanov, T., Pimentel, A.D., Polstra, S., Bose, R., Zissulescu, C., Deprettere, E.: Daedalus: toward composable multimedia MP-SoC design. In: Proceedings of the 45th annual Design Automation Conference, DAC '08, pp. 574–579 (2008)
41. Padmanabhan, S., Chen, Y., Chamberlain, R.D.: Optimal design-space exploration of streaming applications. In: Proc. of the IEEE Int. Conference on Application-specific Systems, Architectures and Processors (ASAP), pp. 227–230 (2011)
42. Palesi, M., Givargis, T.: Multi-objective design space exploration using genetic algorithms. In: Proc. of the Int. Symposium on Hardware/Software Codesign (CODES), pp. 67–72 (2002)

43. Panerati, J., Sciuto, D., Beltrame, G.: Optimization strategies in design space exploration. In: S. Ha, J. Teich (eds.) Handbook of Hardware/Software Codesign. Springer, Dordrecht (2017)
44. Pimentel, A.: A case for security-aware design-space exploration of embedded systems. Journal of Low Power Electronics and Applications **10**(3) (2020)
45. Pimentel, A.D.: Exploring exploration: A tutorial introduction to embedded systems design space exploration. IEEE Design & Test **34**(1) (2017)
46. Pimentel, A.D., Erbas, C., Polstra, S.: A systematic approach to exploring embedded system architectures at multiple abstraction levels. IEEE Trans. on Computers **55**(2), 99–112 (2006)
47. Pimentel, A.D., van Stralen, P.: Scenario-based design space exploration. In: S. Ha, J. Teich (eds.) Handbook of Hardware/Software Codesign. Springer, Dordrecht (2017)
48. Piscitelli, R., Pimentel, A.D.: Design space pruning through hybrid analysis in system-level design space exploration. In: Proc. of the Int. Conference on Design, Automation, and Test in Europe (DATE), pp. 781–786 (2012)
49. Piscitelli, R., Pimentel, A.D.: Interleaving methods for hybrid system-level MPSoC design space exploration. In: Proc. of the Int. Conference on Embedded Computer Systems (SAMOS), pp. 7–14 (2012)
50. Quan, W., Pimentel, A.D.: Towards exploring vast MPSoC mapping design spaces using a bias-elitist evolutionary approach. In: Proc. of the Euromicro Digital System Design Conference (DSD), pp. 655–658 (2014)
51. Quan, W., Pimentel, A.D.: A hybrid task mapping algorithm for heterogeneous MPSoCs. ACM Trans. on Embedded Computing Systems **14**(1) (2015)
52. Quan, W., Pimentel, A.D.: A hierarchical run-time adaptive resource allocation framework for large-scale MPSoC systems. Design Automation for Embedded Systems **20**(4), 311–339 (2016)
53. Quan, W., Pimentel, A.D.: Scenario-based run-time adaptive MPSoC systems. Journal of Systems Architecture **62**, 12 – 23 (2016)
54. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: Proc. of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 4780–4789 (2019)
55. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V., Kurakin, A.: Large-scale evolution of image classifiers. In: Proc. of the 34th International Conference on Machine Learning-Volume 70, pp. 2902–2911. JMLR. org (2017)
56. Reiss, A., Stricker, D.: Introducing a new benchmarked dataset for activity monitoring. In: 2012 16th International Symposium on Wearable Computers, pp. 108–109. IEEE (2012)
57. Sangiovanni-Vincentelli, A., Martin, G.: Platform-based design and software design methodology for embedded systems. IEEE Design and Test of Computers **18**, 23–33 (2001)
58. Sapra, D., Pimentel, A.D.: Constrained evolutionary piecemeal training to design efficient neural networks. In: Proc. of the 33rd Int. Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems (IEA/AIE 2020) (2020)
59. Sapra, D., Pimentel, A.D.: An evolutionary optimization algorithm for gradually saturating objective functions. In: Proc. of the ACM Int. Genetic and Evolutionary Computation Conference (GECCO 2020) (2020)
60. Singh, A.K., Shafique, M., Kumar, A., Henkel, J.: Mapping on multi/many-core systems: survey of current and emerging trends. In: Proc. of the Design Automation Conference (DAC), pp. 1–10 (2013)
61. Stefanov, T., Pimentel, A.D., Nikolov, H.: Daedalus: System-level design methodology for streaming multi-processor embedded systems-on-chip. In: S. Ha, J. Teich (eds.) Handbook of Hardware/Software Codesign. Springer, Dordrecht (2017)
62. Stierand, I., Malipatlolla, S., Fröschle, S., Stühring, A., Henkler, S.: Integrating the security aspect into design space exploration of embedded systems. In: Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops, pp. 371–376 (2014)
63. van Stralen, P., Pimentel, A.D.: Scenario-based design space exploration of MPSoCs. In: Proc. of IEEE Int. Conference on Computer Design (ICCD), pp. 305–312 (2010)

64. van Stralen, P., Pimentel, A.D.: A trace-based scenario database for high-level simulation of multimedia MP-SoCs. In: Proc. of the Int. Conference on Embedded Computer Systems: Architectures, MOdeling and Simulation (SAMOS), pp. 11–19 (2010)
65. van Stralen, P., Pimentel, A.D.: A SAFE approach towards early design space exploration of fault-tolerant multimedia MPSoCs. In: Proc. of Int. conference on Hardware/software codesign and system synthesis (CODES+ISSS), pp. 393–402 (2012)
66. van Stralen, P., Pimentel, A.D.: Fitness prediction techniques for scenario-based design space exploration. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems **32**(8), 1240–1253 (2013)
67. Tan, B., Biglari-Abhari, M., Salcic, Z.: An automated security-aware approach for design of embedded systems on MPSoC. ACM Transactions on Embedded Computing Systems **16**(5s) (2017)
68. Thompson, M.: Tools and techniques for efficient system-level design space exploration. Ph.D. thesis, Universiteit van Amsterdam (2012)
69. Thompson, M., Nikolov, H., Stefanov, T., Pimentel, A.D., Erbas, C., Polstra, S., Deprettere, E.: A framework for rapid system-level exploration, synthesis and programming of multimedia MP-SoCs. In: CODES+ISSS '07: Proceedings of the 5th IEEE/ACM International Conference on Hardware / Software Codesign and System Synthesis, pp. 9–14 (2007)
70. Thompson, M., Pimentel, A.D.: Towards multi-application workload modeling in Sesame for system-level design space exploration. In: S. Vassiliadis, M. Bereković, T.D. Hämäläinen (eds.) Embedded Computer Systems: Architectures, Modeling, and Simulation, pp. 222–232. Springer Berlin Heidelberg (2007)
71. Thompson, M., Pimentel, A.D.: Exploiting domain knowledge in system-level MPSoC design space exploration. Journal of Systems Architecture **59**(7), 351–360 (2013)
72. Thompson, M., Pimentel, A.D., Polstra, S., Erbas, C.: A mixed-level co-simulation method for system-level design space exploration. In: Proc. of the IEEE/ACM Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia '06), pp. 27–32 (2006)
73. Thoziyoor, S., Ahn, J.H., Monchiero, M., Brockman, J.B., Jouppi, N.P.: A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. In: Proc. of the Int. Symposium on Computer Architecture (ISCA), pp. 51–62 (2008)
74. Uhlig, R.A., Mudge, T.N.: Trace-driven memory simulation: A survey. ACM Computing Surveys **29**(2), 128–170 (1997)
75. Weichslgartner, A., Wildermann, S., Götzfried, J., Freiling, F., Glaundefined, M., Teich, J.: Design-time/run-time mapping of security-critical applications in heterogeneous MPSoCs. In: Proceedings of the 19th International Workshop on Software and Compilers for Embedded Systems (SCOPES), pp. 153–162 (2016)
76. Wolf, W., Jerraya, A.A., Martin, G.: Multiprocessor system-on-chip (MPSoC) technology. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems **27**(10), 1701–1713 (2008)
77. Xie, L., Yuille, A.: Genetic cnn. In: Proc. of the IEEE International Conference on Computer Vision, pp. 1379–1388 (2017)

Index