Fast Scenario-Based Design Space Exploration using Feature Selection

Peter van Stralen, Andy Pimentel

Insititute for Informatics, University of Amsterdam
{p.vanstralen,a.d.pimentel}@uva.nl

Abstract: This paper presents a novel approach to efficiently perform early system level design space exploration (DSE) of MultiProcessor System-on-Chip (MPSoC) based embedded systems. By modeling dynamic multi-application workloads using application scenarios, optimal designs can be quickly identified using a combination of a *scenario-based DSE* and a feature selection algorithm. The feature selection algorithm identifies a representative subset of scenarios, which is used to predict the fitness of the MPSoC design instances in the genetic algorithm of the scenario-based DSE. Results show that our scenario-based DSE provides a tradeoff between the speed and accuracy of the early DSE.

1 Introduction

A significant amount of research has been performed on system-level DSE for MPSoCs [Gri04] during the last two decades. The majority of this work is focused on the analysis of MPSoC architectures under a single, static application workload. The current trend, however, is that application workloads executing on embedded systems become more and more dynamic. This is not only the time-dependent behavior of a single application, the interaction between different applications can also be hard to predict. This dynamic behavior can be classified and captured using so-called *workload scenarios* [G+09].

In this paper, we propose a novel scenario-based DSE method that allows for capturing the dynamic behavior of multi-application workloads in the process of system-level DSE. An important problem that needs to be solved by such scenario-based DSE is the rapid evaluation of MPSoC design instances during the search. Because the number of different workload scenarios can be large, it is infeasible to rapidly evaluate a MPSoC design instance during DSE by exhaustively analyzing (e.g., via simulation) all possible workload scenarios for that particular design point. As a solution, a *representative subset of workload scenarios* can be used to make the evaluation of MPSoC design instances as fast as possible. The difficulty is that the representativeness of a subset of workload scenarios is dependent on the target MPSoC architecture. But since the evaluated MPSoC architectures are not fixed during the process of DSE, we need to *simultaneously co-explore* the MPSoC design space and the workload scenario space to find representative subsets of workload scenarios for those MPSoC design instances that need to be evaluated. To this end, we pro-

pose a scenario-based DSE method combining a multi-objective genetic algorithm (GA) and a feature selection algorithm.

The remainder of this paper is organized as follows. In the next section, we briefly summarize the concept of scenario-based DSE. Section 3 describes the proposed framework to perform scenario-based DSE. In Section 4, we present experimental results in which we evaluate our scenario-based DSE approach, and also compare our method to a coevolutionary scenario-based DSE approach. Section 5 discusses related work, after which Section 6 concludes the paper.

2 Scenario-based Design Space Exploration

With the introduction of multiple applications that can execute (possibly simultaneously) on a single embedded system, the interactions on the system become more complex. If traditional DSE is applied, there are two options: 1) isolate the different applications on the MPSoC platform or 2) design the system for the situation where all the applications are running simultaneously. In both situations, the system will typically be over-designed. In case of an isolated design, each application must run on a separate part of the platform to prevent interaction at runtime. This can take up more resources on the MPSoC than required. The same is true for running all applications simultaneously, as they must meet their deadline while assuming that all applications are running. To avoid such worst case design, and to optimize the system for cost and efficiency, we propose to model the interactions of the applications using workload scenarios. Workload scenarios allow the designer to exactly describe the usage of the system. To prevent over-design, scenarios can describe which applications can be active simultaneously.

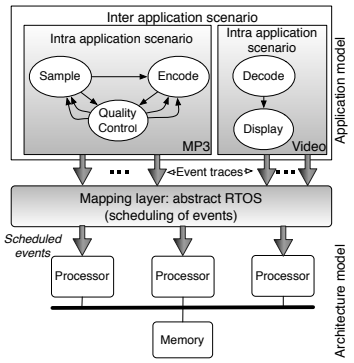

Figure 1: High level scenario-based MPSoC simulation

More precisely, scenario-based DSE bundles two concepts. The first concept is the early DSE of MPSoC based embedded systems. During such early DSE, MPSoC platform instances are quickly evaluated in order to find a set of promising candidate designs. For evaluation purposes, we are using the high-level MPSoC simulation framework Sesame [PEP06]. This framework, which is illustrated in Figure 1, enables fast performance evaluation using separate application and architectural models. An application model describes an application using a Kahn Process Network (KPN), while the architecture model models the MPSoC architecture in a cycle-approximate fashion. Subsequently, there is an explicit mapping of the application model(s) onto the architecture model, implemented using trace-driven co-simulation of the two aforementioned models. Mapping solves two aspects concurrently: 1) *allocation* and 2) *binding*. Allocation selects the architectural components used on the MPSoC platform, whereas the binding defines on which architectural component the application tasks

and communications are executed. During the evaluation of a mapping, each process in an application model generates a trace of application events, representing the application workload at a high level of abstraction. These event traces are simulated by the architecture model to obtain non-functional metrics like execution time and energy consumption.

The other concept that is used in scenario-based DSE is the deployment of workload scenarios [G+09]. In this work, we distinguish two types of workload scenarios: intra and inter application scenarios. Intra-application scenarios describe the different behaviors, or operation modes, within an application. For example, an MP3 application can play music in mono or stereo sound. An inter-application scenario describes the behavior of multiple applications. In our case, it specifies which applications can run concurrently. In the example of Figure 1, the inter-application scenario describes simultaneous execution of the MP3 application and the video application. In order to fully describe what a system is doing, a complete *application scenario* bundles the possible intra-application scenarios of all the active applications. The set of active applications is described using an inter-application scenario, whereas each intra-application scenario specifies a particular operation mode of an individual application. An example application scenario in Figure 1 is that the MP3 application is playing music in mono sound, while the video decodes at a low bitrate.

Incorporating the concept of application scenarios in the process of early DSE of MPSoCs should pave the road for exploring modern embedded systems that are executing dynamic multi-application workloads. Earlier work in this direction [vSP10a] already showed that a random pick of a set of application scenarios does not result in the best set of candidate design instances. Therefore, scenario-based DSE aims at efficiently searching for a set of candidate MPSoC design instances that perform well when considering all the potential situations that can occur in a dynamic multi-application workload.

3 Proposed Framework

In this paper, we propose a new framework for scenario-based DSE. The framework is illustrated in Figure 2. The left-hand side of the diagram shows the general flow of our scenario-based DSE. As input, the description of the target multi-application workload and a MPSoC platform are supplied, as well as search parameters for the GA.

The multi-application workload is described using KPN application models for each separate application in the multi-application workload and a scenario database in which the inter and intra application scenarios are made explicit. For the latter, we use the work from [vSP10b], in which a scenario-aware extension of the Sesame simulation framework is presented. This means that the input KPN application models specify the structure of each individual (concurrent) application allowing for mapping exploration of the application tasks, while the different possibilities for multi-application workload behavior (e.g., which applications or application modes are active at the same time) are characterized in the scenario database.

As was mentioned earlier, the number of different application scenarios can be very large. Consequently, it is infeasible to rapidly evaluate MPSoC design instances during early

DSE by exhaustively analyzing all possible workload scenarios for these particular design instances. As a solution, a *representative subset of application scenarios* can be used. The representativeness of a subset of application scenarios is, however, dependent on the target MPSoC architecture, which is in our case not fixed as we are performing MPSoC DSE. The solution is to co-explore the MPSoC design instances and the representative subset. The right-hand side of Figure 2, which zooms in on our scenario-based DSE, shows the two co-exploration processes. One process, called the *design explorer*, is similar to classical MPSoC DSE as it uses a metaheuristic, which is in our case a GA, to search for optimal mappings (i.e., optimal MPSoC design instances). The second process, referred to as the *subset selector*, tries to identify the best representative subset of application scenarios. As will be explained later on, the subset selector is implemented using feature selection. Both processes are asynchronous, and inter-process communication is performed via shared memory.

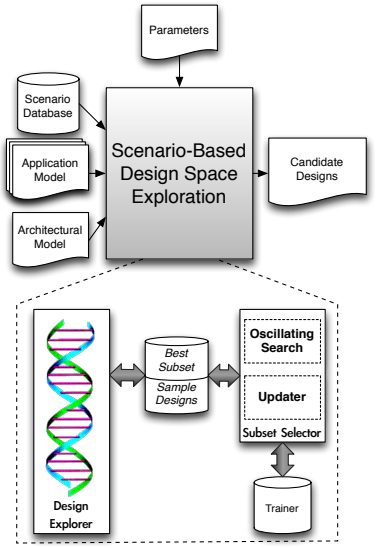

Figure 2: The framework for scenario-based DSE using feature selection

The primary reason for asynchronous execution is that both processes are independent. To provide a trade-off between DSE and subset training effort, the user can manually assign a fixed number of worker threads to the design explorer and subset selector. After termination of the design explorer, the scenario-based DSE finishes by returning a Pareto front of MPSoC design instances (in the case of this paper a trading off performance, energy consumption and cost).

3.1 Design Explorer

The design explorer is responsible for searching and identifying optimal MPSoC design instances by exploring different mappings of the multi-application workload on the MPSoC platform architecture model. To actually perform the search, a NSGA-II [D$^+$02] based multi-objective GA is used. NSGA-II is an elitist selection algorithm that uses non-dominated sorting to select the offspring individuals. An example of non-dominated sorting is given in Figure 4. Non-dominated sorting ranks all the design points based on their dominance depth [C$^+$07]. Conceptually, this dominance depth can be obtained by iteratively obtaining and removing the set of non-dominated individuals. The main reason of choosing NSGA-II is its use of the dominance depth for the optimization. In the subset selector, the scalar value of the dominance depth can easily be used for rating the subset quality independent of the number of objectives. This will be explained in the next subsection.

In order to couple the GA to the feature selection algorithm inside the subset selector, some communication is required. Figure 3 shows the extended algorithm of the GA. Before any evaluation can be done, the current representative subset must be retrieved from the shared memory. Using the obtained subset of application scenarios, the candidate mappings can be quickly evaluated using Sesame [PEP06]. The design explorer has a pool of worker threads, where each thread can simulate and evaluate a single mapping in parallel, to fully utilize multi-core systems. A sample of the simulated mappings is sent to shared memory. After the evaluation, the GA can select individuals based on their *estimated fitness*. In case of a representative scenario subset, the decisions made by the NSGA-II selector are similar to the case where the *real fitness* would have been used. The real fitness is equal to the average fitness of all possible application scenarios as stored in the scenario database. The selected individuals can be used for reproduction after which the whole procedure will repeat itself in the next generation.

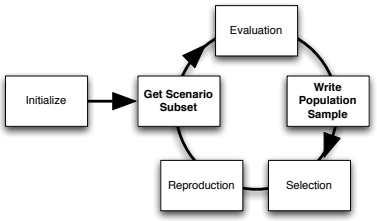

Figure 3: The GA to find the optimal mapping extended with steps to communicate data with the subset selector

3.2 Subset Selector

To obtain an estimated fitness of the MPSoC design instances in the GA population of the design explorer, a good representative subset of scenarios is required. We have chosen to select this representative subset of scenarios using feature selection. Feature selection is a technique of selecting a subset of relevant features for building classifiers. This fits our needs well since we want to select a set of application scenarios (the features) to classify the dominance depth of a mapping (i.e., a MPSoC design instance). An important consequence of this decision is that we need to have a set of training mappings of the multi-application workload from the *design explorer* to be able to score a subset of scenarios within the *subset selector*. Moreover, we need to have the real fitness in order to judge if the estimation of the dominance depth is correct.

For this purpose, we added a trainer. The trainer keeps a small set of mappings which are exhaustively evaluated with Sesame (using all application scenarios in the scenario database) to obtain their real fitness. The trainer is updated at runtime, using the current sample of mappings originating from the design explorer. In this way, we train the scenario subset exactly with respect to the part of the design space where the design explorer is currently searching. To isolate the exhaustive mapping evaluation from the GA in the design explorer, we have introduced a separate pool of worker threads in the subset selector. These worker threads are managed by the ”updater thread”. This updater thread will read out the sample mappings from the shared memory. Based on their estimated fitness, it will select a small portion of interesting mappings to add to the trainer, from which the real fitness is determined. To fully parallelize the simulation of these mappings (i.e., exploiting multi-core host systems), we do not only evaluate the different mappings in parallel,

but also the total set of application scenarios is divided in multiple chunks. In this way, a single mapping can be evaluated quicker using multiple worker threads, each simulating a different scenario for the respective mapping.

Simultaneously with the updater thread, a feature selection algorithm is executed. Every time this algorithm finds a better subset, it updates the best subset in the shared memory. We have chosen to use the sequential oscillation search [SP00] as our feature selection algorithm. Oscillating search uses a kind of hill climbing technique to improve the current subset of scenarios. We have chosen to use the oscillating search for two reasons. As it iteratively improves the scenario subset, it can provide premature results that can already be used in the design explorer. Additionally, it starts from an existing subset. As a result, we can keep using the current representative subset after the trainer has been updated.

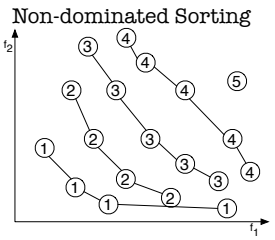

Figure 4: An example of non-dominated sorting. All design points are annotated with their dominance depth.

The *misclassification rate* is used to judge the quality of the scenario subset. The misclassification rate is equal to the percentage of incorrectly predicted dominance depths (Figure 4). The lower the misclassification rate, the better the subset is able to identify the Pareto front that consists of the non-dominated solutions. The benefits of using the dominance depth are twofold. First, the better the estimation of the dominance depth is, the better our selection algorithm NSGA-II will perform its work. Second, the quality metric of the subset is independent of the number of the objectives in the mapping space.

At the startup phase, the trainer is empty. Therefore, a random subset is created as the quality of the subset can not be judged. Once the updater thread has issued mappings to the trainer, the sequential oscillating search can be started. When a better subset is found during this search, it is communicated to the mapping space using the shared memory. Periodically, the updater thread has a new group of exhaustively evaluated mappings for the trainer. In this case, the oscillating search is temporarily halted.

4 Experiments

To verify our framework, a number of experiments have been performed with a stochastic multi-application workload with 10 different applications. The multi-application workload with 4607 application scenarios is generated using a Python tool that is loosely based on [Ors09]. We decided to use stochastic applications because it provides us full control over the workload. On top of that, stochastic applications can be instrumented to mimic the abstract behavior of real applications. The workload is mapped onto a MPSoC platform with four general purpose processors, two ASIPs and two ASICs. In addition to these processing elements, the platform contains a shared memory, a crossbar network, and four point-to-point FIFO communication channels. As our framework performs mapping DSE, handling both the allocation and the binding, not all the components in the platform are used in the final design. As a result, we consider three objectives: execution time, energy

and cost (with respect to silicon area).

The early DSE of an embedded system like this can only be done using heuristic search methods such as a GA. The number of possible mappings is exponential with respect to the number of potential architectural components. Our example multi-application workload has more than $6.8 * 10^{58}$ different mappings onto the architecture. Moreover, the exponential number of scenarios (with respect to applications) makes fitness prediction – based on a representative subset of scenarios – a necessity for a feasible early DSE. Two types of experiments have been performed. The first experiment focusses on the feature subset selection. Based on the results of this experiment, we have performed our second experiment in which we demonstrate the functionality of the scenario-based DSE.

4.1 Subset Selection

One of the parameters of our framework is the size of the scenario subset. This size involves a tradeoff with respect to quality and speed. The smaller a subset is, the faster our early DSE will be performed. On the other hand, a larger subset will result in a better fitness prediction and thus a better average outcome of the GA. The potential design space of possible subsets is huge: in our example multi-application workload there are already $6.6 * 10^{42}$ different subsets of size 15.

To investigate the exact influence of the subset size, we have applied our subset selector in isolation. For this purpose, the updater thread is disabled and replaced by a fixed trainer. The fixed trainer is selected in beforehand and contains 518 exhaustively evaluated mappings. As a result, we are able to investigate the properties of the subset selection without any external influences. We have restricted the execution time of the subset selector to 10 minutes (wallclock time). A longer execution time is likely to get better scenario subsets, but in early DSE the time available for training is limited.

4.1.1 Quality versus size

In our framework, we have decided to perform feature selection with a deterministic algorithm. To substantiate this decision, we have compared our subset selector using a deterministic feature selection (FS) algorithm to a genetic algorithm (GA) approach (comparable with the subset search method in [vSP10a]). Figure 5 shows the result of the experiment. The horizontal axis shows the wallclock time in seconds, whereas the vertical axis shows the quality of the best subset at that point in time. Quality is defined by the misclassification rate: the lower the number of incorrectly predicted dominance depths, the better the subset is. Each time a new subset is found, a marker is placed. The experiment is done for different subset sizes, where each line refers to the misclassification of a subset with a specific size.

Both approaches start with random subsets and therefore their initial quality is similar. This can be clearly seen in the figure. In general, a larger subset means a better quality. A subset with a size of 60 starts with a misclassification rate of 54 percent, whereas a subset with 15 scenarios incorrectly classifies 68 percent of the individuals. During the startup

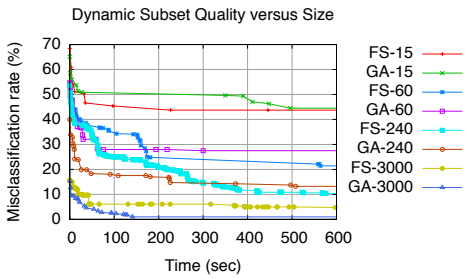

Figure 5: The relation between the quality of the subset and its size for feature selection (FS) and a genetic algorithm (GA)

phase of the search, the GA-based selector has better quality subsets than our FS-based subset selector. The oscillating search applies a hill climbing technique. As a result, the changes to the subset are minor and the subset can only be changed slowly. The GA, on the other hand, can apply crossover to quickly change large parts of the representative subset of scenarios. However, the deterministic approach of our FS-based subset selector will eventually outperform the GA. For a subset of size 15, it takes only 30 seconds. This time grows to 170 and 300 seconds for the subset size of 60 and 240 scenarios, respectively. As comparison, a large subset of 3000 scenarios is also added. Within 600 seconds, our subset selector does not outperform the GA-based approach of [vSP10a]. However, after a certain period it is expected to outperform the GA.

Since a larger subset means a larger evaluation time of the design explorer, the better eventual quality of the subset is more important than the better selection of the approach of [vSP10a] in the startup phase. In this phase, the quality improvements in the MPSoC designs (execution time, energy, etc.) are still quite significant and it is relatively easy to discriminate between different MPSoC designs. Later in the DSE, differences become smaller, but at this point in time our deterministic subset selector will have a better representative subset than is achieved with the stochastic GA-based approach. The larger the representative subset is, the more time there is to find a high quality representative subset. This large representative subset results in a longer evaluation period in the design explorer, resulting in more time for the subset selection.

Based on the above results, we have decided to use a subset with 240 scenarios. We believe that it provides a good trade-off between size (fast evaluation time in the design explorer) and quality. Compared to the large subset of 3000 that achieves a misclassification of 5%, a subset of 240 scenarios drops below 15% within 5 minutes. After five minutes, the trainer will likely be updated and the subset can quickly be adjusted for better representativity.

4.1.2 Misclassification: how does it look like

To give a feeling of how the misclassification looks like, consider Figure 6. This figure uses a density graph for the relationship between the predicted and the real values of the dominance depths. The color legend (density) shows the number of mappings that have a predicted dominance depth of x (the x-axis) and a real dominance depth of y (the y-axis). To get this relationship, we used a set of 1933 mappings which is disjunct to the set of mappings with which the scenario subset is trained. Ideally, the predicted and real dominance depth are equal. In the density graph, this would manifest itself with a straight line ($y = x$). In our experiment, this straight line is also visible. However, the line becomes wider for high dominance depths.

In our case, we are interested in good mappings. These mappings have a dominance depth that is as low as possible. Looking at the real Pareto front (the mappings with dominance depth 1), Figure 6 indicates that 84 percent is classified correctly. Looking at the other 16 percent, we see that 14 percent has a predicted dominance depth of 2. Once the population of the NSGA-II based GA is large enough, it is quite likely that also these mappings are added to the population of individuals that is carried to the next generation. The larger discrepancy between prediction and reality is mainly observed at the individuals with a high value of the real dominance depth. This is, however, not problematic for our case, as we are not interested in low quality mappings anyway.

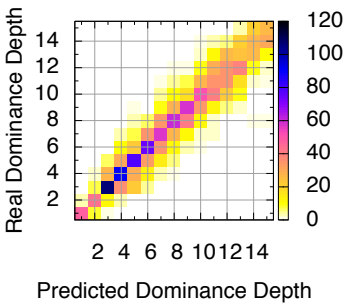

Figure 6: A density graph with the relationship between the predicted and the real dominance depth.

4.2 Design Space Exploration

Now that the subset selector has been validated, the complete framework can be validated. This is done using two experiments. First, we will show that the scenario-based DSE with feature selection can deliver results much faster than traditional early DSE. In such traditional DSE, MPSoC design points in the design explorer's GA are evaluated exhaustively using all application scenarios. Subsequently, in the second experiment, we investigate the quality of our proposed scenario-based DSE method.

The parameters of the GA in the design explorer are fixed for all experiments. The GA uses a population size of 300 and an offspring size of 100. For the fitness prediction in the experiments with scenario-based DSE, we use a subset of 240 scenarios. To compare the quality of the resulting Pareto fronts, we use the hypervolume [ZT98] metric. The hypervolume is equal to the area of the Pareto front and the larger it is, the better the Pareto front. We have normalized the hypervolume values such that the *combined* Pareto fronts of *all* performed experiments equal to a hypervolume of 1.0.

4.2.1 The evaluation time of early DSE

In the first experiment the evaluation time of traditional DSE and the proposed scenario-based DSE with feature selection (FS) is compared. The traditional DSE uses 12 threads to evaluate the individuals in the GA, whereas the scenario-based DSE with FS uses 10 threads for the design explorer (the mapping GA) and 2 threads for the subset selector. This ratio determined using calibration of time versus precision. The experiment is done on a dedicated server with 2 Intel six-core Xeon L5640 processors running at 2.26Ghz.

From Figure 7, it can be clearly seen that our framework is much faster than the traditional DSE. While the traditional DSE only performs 34 generations, the scenario-based DSE with FS can execute 300 generations. As a consequence, the resulting Pareto front of the

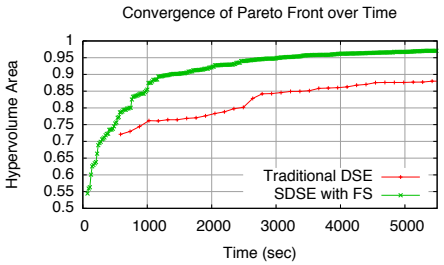

Figure 7: Time-based comparison

traditional DSE after 90 minutes is clearly outperformed by the faster scenario-based approach. We would like to note, however, that although the convergence rate of the traditional DSE looks more slowly and bumpy, this is only due to the small number of generations. The scenario-based DSE is capable of doing more generations, resulting in a smoother and faster convergence. However, it is as expected that the final result of the exact traditional DSE is better as can be seen in Figure 8.

For early DSE, it is key that a large number of candidate designs can be rapidly evaluated. Hence, the traditional DSE method takes too much time (12 hours and 40 minutes) for a quick feedback loop in which the designer is able to manipulate the application(s) and the MPSoC components based on the candidate designs. Moreover, as the number of applications grows, the larger the gap between the evaluation time of the scenario-based DSE and the traditional DSE.

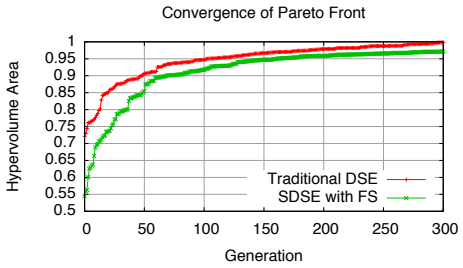

Figure 8: Generation-based comparison

4.2.2 The quality of the scenario-based DSE

In the previous experiment, we have shown that it is necessary to speed up the early DSE of a multi-application workload by using scenario-based DSE. As a next step, we show that the deterministic approach of selecting a subset of scenarios improves our result of the early DSE. To this end, we compare our approach to an alternative scenario-based DSE approach which has been proposed in [vSP10a]. In this alternative approach, a co-evolutionary GA is used to perform scenario-based DSE. The experiment is performed on a sixteen core machine and thus we have used 12 threads for the design explorer and 4 threads for the subset selector. For both the FS and the GA approach ([vSP10a]), the experiment is repeated five times.

Figure 9 shows the results of the experiments. The hypervolume is averaged over all the runs and the standard error over the different experiments is shown with an error bar. During the first 50 generations, the FS and the GA approaches are comparable. In this initial phase, the differences between the fitness of the different platform instances is still large enough to tolerate inaccurate predictions of a subset with a moderate misclassification rate.

As of approximately 50 generations, our FS method starts to outperform the GA approach, and the gap between both approaches steadily increases. After a startup period, the FS approach is able to find a better scenario subset with a higher accuracy. The higher accuracy results in a good and valid distinction between mappings which have fitness values that

are relatively close. As a consequence, the FS approach can find a higher number of better candidate designs. The GA approach is less capable of making a distinction between the highly ranked mappings and its hypervolume starts to flatten earlier.

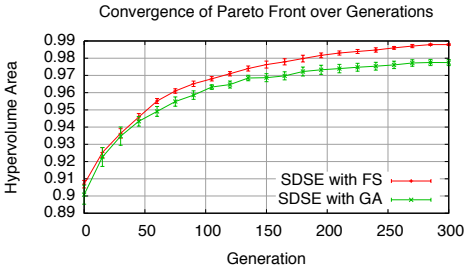

Figure 9: The effect of the deterministic subset selection on the quality of the DSE. The feature selection (FS) approach presented in this paper is compared with an approach where a GA is used to find the scenario subset.

Moreover, it can be seen that the FS approach is more deterministic than the GA approach. The standard error of the different experiments is significantly smaller after 250 generations have passed. On the final result, the standard error of the mean is 3.6 times as small as compared to the GA approach. This is because the GA approach is dependent on the stochastic nature of the GA to find good scenario subsets. A stochastic approach will eventually find a good subset, but the larger the subset is, the tougher it becomes.

5 Related work

There is a large body of research dedicated to early DSE of MPSoC based embedded systems [Gri04]. However, the majority of this work still evaluates the embedded systems with a single (fixed) application workload. Recently, research has begun to apply workload scenarios [G+09] for making the design phase scenario aware [PSZ08, P+06, S+10a]. In [vSP10a] a scenario-based DSE technique using a coevolutionary GA is described. The scenario-based DSE approach proposed in this paper has been inspired by the work of [vSP10a], but as shown in Section 4.2.2, the scenario subset selection is significantly improved and thereby the quality of the entire DSE process.

In our scenario-based DSE, fitness prediction[JB05] is used to deal with the large number of application scenarios. For this purpose feature subset selection[S+10b] is used. An example of a case where feature selection is used for ranking can be found in [L+04]. This work uses a hybrid method, which is a combination of a so called kNN classifier and a NSGA-II based GA, to estimate the Pareto front of a hydrological model. These parts are running in lockstep and not independent as in our approach.

6 Conclusions

In this paper, we have presented a framework for scenario-based DSE with feature selection. Scenarios enable the modeling of the dynamism in multi-application workloads. Our framework performs simultaneous co-exploration using a design explorer and a subset selector. The design explorer uses the current scenario subset to predict the fitness of the MPSoC design instances. The subset selector, on the other hand, uses a sample of design

instances to train a scenario subset using feature selection. Our experiments show that, based on the size of the subset of scenarios, we are able to make a tradeoff between quality and the evaluation time of the early DSE.

References

[C+07] C. Coello et al. *Evolutionary Algorithms for Solving Multi-Objective Problems Second Edition*. Springer US, 2 edition, 2007.

[D+02] K. Deb et al. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Trans. on Evolutionary Computation*, 6(2):182–197, April 2002.

[G+09] S. V. Gheorghita et al. System-scenario-based design of dynamic embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, 14(1):1–45, 2009.

[Gri04] M. Gries. Methods for evaluating and covering the design space during early design development. *Integration, the VLSI Journal*, 38(2):131–183, 2004.

[JB05] Y. Jin and J. Branke. Evolutionary Optimization in Uncertain Environments-a Survey. *IEEE Transactions on evolutionary computation*, 9(3):303–317, 2005.

[L+04] Y. Liu et al. A Hybrid Optimization Method of Multi-objective Genetic Algorithm (MOGA) and K-Nearest Neighbor (KNN) Classifier for Hydrological Model Calibration. In *Intelligent Data Engineering and Automated Learning*, volume 3177, pages 546–551. 2004.

[Ors09] H. Orsila. `http://zakalwe.fi/~shd/foss/kpn-generator/`, 2009.

[P+06] J. M. Paul et al. Scenario-oriented design for single-chip heterogeneous multiprocessors. *IEEE Trans. on VLSI Syst.*, 14(8):868–880, August 2006.

[PEP06] A. D. Pimentel, C. Erbas, and S. Polstra. A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels. *IEEE Trans. on Computers*, 55(2):99–112, 2006.

[PSZ08] G. Palermo, C. Silvano, and V. Zaccaria. Robust Optimization of SoC Architectures: A Multi-Scenario Approach. In *Proceedings of ESTIMedia 2008*, October 2008.

[S+10a] A. Schranzhofer et al. Dynamic and adaptive allocation of applications on MPSoC platforms. In *Proceedings of the ASP-DAC*, pages 885–890, 2010.

[S+10b] P. Somol et al. *Pattern Recognition Recent Advances*, chapter Efficient Feature Subset Selection and Subset Size Optimization, pages 75–97. In-Tech, February 2010.

[SP00] P. Somol and P. Pudil. Oscillating Search Algorithms for Feature Selection. *International Conference on Pattern Recognition*, 2:406–409, 2000.

[vSP10a] P. van Stralen and A. D. Pimentel. Scenario-Based Design Space Exploration of MPSoCs. In *Proceedings of IEEE Int. Conference on Computer Design (ICCD '10)*, October 2010.

[vSP10b] P. van Stralen and A. D. Pimentel. 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 '10)*, July 2010.

[ZT98] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms — A comparative case study. In *Parallel Problem Solving from Nature*, volume 1498, pages 292–301. 1998.