Applications of scenarios in early embedded system design space exploration
van Stralen, P.

Citation for published version (APA):
van Stralen, P. (2014). Applications of scenarios in early embedded system design space exploration

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
In the previous chapters it became clear that modern embedded systems become more and more dynamic. The application behavior of these, typically software-centric, MPSoC based embedded systems is described using application scenarios. As the number of application scenarios is relatively large and diverse, the diversity of the application scenarios needs to be taken into account during the embedded system design.

To cope with the design complexities of MPSoC based embedded systems, system-level design has become a promising approach for raising the abstraction level of design, and thereby increasing the design productivity. Early design space exploration (DSE) is an important ingredient of such system-level design, which has received significant research attention in recent years. However, the majority of these DSE efforts still evaluates and explores MPSoC architectures under single-application workloads. In this chapter, we therefore exploit the concept of application scenarios to introduce scenario-based DSE.

An important problem that needs to be solved by scenario-based DSE is the fact the number of possible application scenarios is too large for an exhaustive evaluation of all the design points with all the scenarios during the MPSoC DSE. Therefore, a representative subset of scenarios must be selected for the evaluation of MPSoC design points. This representative subset must compare 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. Looking back at the experiment in Section 3.4.2, the selection of such a representative subset is not trivial. Not only did the experiment illustrate that a single application scenario is not sufficient to judge the relative quality of two different mappings, but also that the representative subset is dynamic with respect to the current set of mappings that are 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 statically be selected. For the static selection one should have a large fraction of the mappings that are going to be explored during the MPSoC DSE. These mappings are only available during the DSE and, therefore, a dynamic selection method must be used. To this end, we use a coevolutionary genetic algorithm [53, 43] where during the coevolution the search for the representative subset is done simultaneously with the search for the optimal mapping. In this way, the representative subset is able to adapt to the current set of mappings.

This chapter is organized as follows. Section 4.1 will introduce the coevolutionary genetic algorithm. Next, the solution space and the problem space will be discussed in Section 4.2 and Section 4.3. The solution space is concerned with the search for the optimal mapping, whereas the problem space deals with the identification of the representative subset. Section 4.4 discusses some experiments. Finally, the last two sections will discuss some related work and provide a short conclusion.

4.1 Coevolutionary Genetic Algorithm

The goal of scenario-based DSE is to identify a set of optimal mappings of the applications onto the architecture. Such a mapping defines both the allocation of the architectural components for the MPSoC and the binding of the application tasks and communication channels onto the architecture. To boost performance, a representative subset of scenarios is used instead of evaluating each design point using all scenarios. As the representativeness of this scenario subset is dependent on the specific mapping, the DSE problem is ill-defined.

To solve the DSE problem both the representative subset of scenarios and the best set of mappings have to be found. These two elements are dependent and need to be
solved at the same time. The most suitable technique to solve an ill-defined problem [43] is to use a coevolutionary genetic algorithm. Generally, a coevolutionary GA can be split in two parts: a solution part and a problem part. The GA tries to solve the problem, even if the problem is not completely defined yet. While searching for the solution, the problem is also being fully defined. To implement a coevolutionary GA, there are two approaches [43]: a combined and a separate genotype. The combined genotype encodes the solution and the problem within a single chromosome, whereas the separate approach uses two separate chromosomes. For scenario-based DSE, the separate genotype approach is used and the GA is based on the widely used SPEA-2 GA [81].

Strictly, a coevolutionary genetic algorithm with two separate genotypes can be seen as two GAs running in parallel. As illustrated by Figure 4.1, one GA is dedicated to the solution space and one GA is dedicated to the problem space. For scenario-based DSE the solution space consists of a set of mappings that allocate the architecture and bind the application(s) onto it. The goal of the solution space is to optimize the mapping given a representative subset of scenarios. This representative subset of scenarios is searched by the problem space. The embedded system designer limits the size of such a representative subset. In this way, the required time for the early DSE can be limited.

One of the crucial aspects of a coevolutionary genetic algorithm is the interaction between the two populations. Common types of behavior are symbiotic or competitive [53]. In a symbiotic situation both populations cooperate in order to maximize their fitness. An example of a symbiotic relation is the interaction between bees and flowers. Bees gain benefit from the flower by gathering nectar to use as food, whereas the flower benefits from bees as they take care of spreading the pollen of the plant to the next plant. Hence, the bee is taking care of the reproduction of the flower. The competitive behavior is the other way around. What is good for one population is bad for the other. An example is a predator prey behavior. The predator means to improve his hunting capabilities, whereas the prey tries to improve his defensive measures. The scenario-based DSE can be classified as symbiotic. The goal of the representative subset is not to find the worst-case scenarios, but the scenarios that are able to correctly predict the ordering of the mappings. Hence, if the fitness of the representative subset is high, the quality of the identified mappings also should improve.

4.2 Solution Space: MPSOC Mapping

The solution space is responsible for exploring the mapping design space. As can be seen in Figure 4.1, a traditional GA is used where a population of mappings is evaluated over time. The only addition is the exchange of information between the problem space and the solution space. Each generation a sample of the mapping population is taken and communicated to the problem space. Afterwards, the most
recent representative subset of scenarios is taken from the problem space. This representative subset of scenarios is used to evaluate the current population of mappings.

Figure 4.2 shows the chromosome design for the solution space. The mapping chromosome consists of two parts: 1) a Kahn process part and 2) a Kahn channel part. Within these parts, all of the applications are encoded consecutively. The gene values encode the architectural components on which the elements of the applications are mapped; The Kahn processes are mapped onto processors and the Kahn channels are mapped onto memories. A special memory is the internal memory. The internal memory is only available when both the reader and writer of a Kahn channel are mapped onto the same processor. If a channel is mapped onto internal memory, only the local memory of the processor is used.

The example chromosome in Figure 4.2 has 11 genes. Five genes are dedicated to the processes and six genes are dedicated to the communicational channels. As there are three potential processors, the gene value for the Kahn process part is between 0 and 2. For the memories there are three possibilities: two memories and a reserved entry for the internal memory. In this way, the bounded architectural component is encoded for each of the processes and channels. The first process gene of the MP3 application, for example, has gene value 0. Looking at the platform, gene 0 is the SAMPLE process. This process is to be mapped on the first processor: CPU-A. Similarly, the channel 4 (QUALITY → SAMPLE) is mapped on MEM-3. The complete encoded mapping is illustrated in Figure 4.2.

4.2.1 Fitness Function

As discussed earlier, it is infeasible to do an exhaustive evaluation, using all the scenarios, of all the individuals in the solution space. Therefore, we need to estimate the
fitness of the application mappings. There are several methods for estimating the fitness of a solution [34]. Examples are problem approximation, data-driven functional approximation and fitness inheritance. Data-driven functional approximation tries to train a function that, given the inputs, provides the fitness of the solution. Fitness inheritance, on the other hand, bases the fitness of an individual chromosome on its parents. For scenario-based DSE problem approximation is used. Problem approximation tries to replace the original problem statement by an alternative statement that is approximately the same, but that is cheaper to solve. Scenario-based DSE tries to statically identify the mapping that, on average, behaves the best for all the scenarios.

4.3 Problem Space: Application Scenario Subset

The problem space is responsible for identifying the representative subset of scenarios. Just as the solution space, the problem space is based on a traditional GA. As Figure 4.1 shows, the only additional step is that every generation the best representative subset of the population is sampled and communicated to the solution space. Next, a set of mappings from the solution space is obtained. This set of mappings will be used for the trainer, as we will discuss later.

A chromosome (as illustrated in Figure 4.3) for the problem space encodes a complete representative subset. Each gene encodes a single scenario of the representative subset. The scenario is encoded using the integer key of the scenario in the scenario database. A single scenario is allowed to occur more than once in the representative subset. In this case, fewer scenarios need to be evaluated and the repetitive scenario gets more weight within the average fitness of the evaluated mappings.

4.3.1 Fitness Function

The fitness of a subset of scenarios should reflect the representativeness of the scenario subset. In order to obtain the representativeness of a subset of scenarios, the behavior of the complete set of scenarios must be known. For this purpose, a trainer $T$ is used that consists of a small number of mappings that are evaluated exhaustively (i.e.,
(a) Exhaustively evaluate training mappings.

(b) Get real and estimated fitness.

(c) Calculate average fitness deviation.

\[
F_p([s_1, s_3]) = 0.5D([s_1, s_3]) + 0.5R([s_1, s_3]) = 0.5 \times 0.23 + 0.5 \times \frac{1}{2} = 0.37
\]

(d) Determine rank correlation and obtain the representativeness of the subset.

Figure 4.4: An example fitness evaluation of the subset \([d_1, s_3]\).

using all the scenarios). Hence, the exact fitness for each of the mappings is known and, therefore, the quality of a subset in the problem population can be obtained. The quality of a subset \(F_p(s)\) is determined for each individual objective (like execution time and energy):

\[
D(s) = \frac{1}{|T|} \sum_{i=0}^{\mid T \mid} \text{abs}(F(T[i]) - \overline{F^s(T[i])}) \tag{4.1}
\]

\[
R(s) = 1 - \rho(T, F, F^s) \tag{4.2}
\]

\[
F_p(s) = \alpha \times D(s) + (1 - \alpha) \times R(s) \tag{4.3}
\]

The problem fitness \(F_p\) is composed of two individual metrics: fitness deviation (Equation 4.1) and rank correlation (Equation 4.2). Main reason of using two metrics is that the individual flaws of the metrics make it infeasible to use them individually. Therefore, the metrics are combined using the weighting factor \(\alpha\) (Equation 4.3). These two metrics are not inherently conflicting. If the fitness deviation of a subset is zero, the subset also has an optimal rank correlation. Currently, we have only used an equal weighting \((\alpha = 0.5)\) for both metrics.

Figure 4.4 shows an example of the fitness calculation. In this example, the scenario database consists of four scenarios. First, a trainer is required. This trainer is shown in Figure 4.4a and consists of three mappings \((A, B\) and \(C)\). Each of these mappings is exhaustively evaluated for all of the four scenarios. Using this trainer, the representativeness of a scenario subset can be obtained.

To obtain the representativeness of the scenario subset (i.e., the fitness of the subset), its fitness approximations for the training mappings have to be calculated (Figure 4.4b). In this way, the real fitness of the training mappings and the estimated
fitness of the scenario subset are known. Using the real and the estimated fitness values of the training mappings, the two metrics for the representativeness of the subset can be evaluated:

1. **Fitness Deviation:** At first, the fitness deviation measures the difference between the real fitness \( F \) (based on exhaustive simulation) and the estimated fitness \( F_s \) based on the representative subset \( s \). The fitness deviations are normalized with respect to the fitness values of the specific mapping (see Definition 5 on page 28).

Figure 4.4c shows how the fitness deviation of our example subset fitness evaluation can be calculated. In the case of mapping \( B \), for example, the real fitness is 4.2 and the estimated fitness is 3.4 (as shown in Figure 4.4b). The range of fitness values of mapping \( B \) is between 3.3 and 6.4 (see Figure 4.4a). As a result, the normalized value of the real fitness is \((4.2 - 3.3)/(6.4 - 3.3) = 0.29\). Similarly, the normalized estimated fitness is \((3.4 - 3.3)/(6.4 - 3.3) = 0.03\). The absolute difference of these normalized fitness values is 0.26. As shown in the formula for \( D([s_1, s_3]) \) in Figure 4.4c, the fitness deviation for mapping \( A \) and mapping \( C \) is 0.23 and 0.19. This results in an average fitness deviation of approximately 0.23.

2. **Rank Correlation:** Secondly, the Spearman’s rank correlation is used [69] to measure the correlation between two variables. In our case, the real fitness and the estimated fitness are compared. At first, the mappings are ranked according to the real and the estimated fitness. This can be seen in Figure 4.4b. For the real fitness, the ranking is as follows: 1) Mapping \( A \), 2) Mapping \( C \) and 3) Mapping \( B \) (this means that mapping \( A \) is the best and mapping \( B \) is the worst). The estimates given by our scenario subset are: 1) Mapping \( A \), 2) Mapping \( B \) and 3) Mapping \( C \). As there are no ties, the Spearman’s rank correlation can be calculated as follows:

\[
\rho = 1 - \frac{6 \sum d_i^2}{n (n^2 - 1)} = 1 - \frac{6((1 - 1)^2 + (3 - 2)^2 + (2 - 3)^2)}{3 (3^2 - 1)} = 0.5
\]

In this equation \( n \) is the number of training mappings and \( d_i \) is the difference between the estimated and the real rank of training mapping \( i \).

The Spearman’s rank correlation is always in the range between \(-1\) and 1. A correlation of 1.0 corresponds to a perfect match between the ranking of the real and estimated fitness, whereas a correlation of 0.0 denotes that there is absolutely no relation between both rankings. Correlation can also be negative. In this case, the estimated fitness gives exactly the opposite mapping ordering.
than the real fitness. As this is undesirable, the rank correlation must be as close to 1.0 as possible. Therefore, we subtract the Spearman’s rank correlation from 1.0 in order to transform it into a minimization problem.

The final fitness of our example subset is the weighted average of the two aforementioned metrics: 0.37.

### 4.3.2 Trainer Selection

In the previous subsection, we discussed how the fitness of an individual subset is obtained. The fitness calculation of a scenario subset, however, is completely dependent on the trainer that is used. If the trainer is not representative, the obtained subset cannot be representative either. Therefore, it is crucial to obtain a representative trainer.

While selecting a trainer, there are two conflicting requirements: 1) size and 2) quality. On one hand, the trainer must be as compact as possible. The fewer mappings that there are in the trainer, the less exhaustive evaluations that are required. On the other hand, the quality of the trainer must be as high as possible. The more mappings there are in the trainer, the higher the probability that the trainer reflects the current population of the solution space (i.e., the set of mappings that is currently used to find the optimal mappings).

In order to keep a modestly sized trainer that reflects the current state of the solution space, we have chosen to dynamically update the trainer during the coevolution. Periodically, the trainer is updated with a small number of mappings that is sampled from the current population in the solution space. In this way, the trainer can reflect the part of the solution space that is searched at that specific point in time.

The procedure is illustrated in Figure 4.5. The core of the procedure is the coevolutionary process. During this process both spaces (solution and problem space) are coevolved. After each generation, some of the individuals are exchanged. For the solution part, this means that a new representative subset is read to predict the fitness of a new set of mappings. The problem part, on the other hand, will collect the
sampled mappings for later use. Periodically, the coevolutionary process is halted. At this halting point the continuation depends on the user-defined exploration time (i.e., the total time the DSE is running). In case all the exploration time has passed, the DSE is finished after the exchange of individuals. Otherwise, the trainer is updated before the coevolutionary process is continued.

For the trainer extension, the collected mappings in the problem space are used. First, these mappings are filtered such that no duplicate mappings will be added to the trainer. Next, a quick evaluation is done for the potential new trainer mappings. For this quick evaluation, a subset of scenarios is used that is obtained by selecting a limited number of the mostly used scenarios in the complete set of scenario subsets. Finally, the new trainer mappings are selected based on the standard deviation of the mapping fitness on the different scenarios that are evaluated. The mappings with the highest diversity are added to the trainer. As the diversity is high, the real average fitness is harder to predict. Therefore, the trainer uses these mappings to improve the fitness prediction in the solution space. Before the selected mappings are added to the trainer, they are evaluated exhaustively.

As the size of the trainer effects to execution time of the DSE, we have built in several measures to control the execution time and how this time is spent. As the Sesame simulation calls account for the majority of the execution time, the designer can specify the maximal number of Sesame invocations. This is done both for mapping evaluation in the solution space and the trainer extension for the problem space. Given the number of remaining Sesame invocations, it is decided how many training mappings can be added to the trainer. The more Sesame invocations are assigned to the problem space, the higher the potential quality of the trainer.

### 4.4 Case Studies

In order to evaluate our coevolutionary DSE approach, the DSE of two types of multi-application workloads are studied. The first workload is composed of two real world applications, whereas the second workload only contains synthetic applica-
Table 4.1: General settings for the (coevolutionary) GA

<table>
<thead>
<tr>
<th></th>
<th>Solution Space</th>
<th>Problem Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>200</td>
<td>150</td>
</tr>
<tr>
<td>Offspring size</td>
<td>30</td>
<td>25</td>
</tr>
<tr>
<td>$P_{\text{crossover}}$</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>$P_{\text{mutate}}$</td>
<td>0.01</td>
<td>0.02</td>
</tr>
<tr>
<td>Generations</td>
<td>600</td>
<td>Gen-train</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>

By also using synthetic applications, the experiment has more control on the characteristics of the multi-application workload. This control is used to create a more workload that is more dynamic than our real world multi-application workload. For each of the two multi-application workloads a DSE is performed that entails the search for optimal design instances with respect to cost and execution time. The used heterogeneous MPSoC platform that is shown in Figure 4.6 contains eight processing elements: three MIPS processors, three ARM processors and two dedicated ASICs for DCT and VLE operations. For communication the platform architecture contains a crossbar with private memory buffers, four dedicated point-to-point FIFOs and a bus connected to a shared memory. As we are exploring mappings, not all of these components need to be allocated in the final design.

To illustrate the added value of scenario-based DSE, we compare it against two other approaches: 1) a static approach and 2) an exhaustive approach. The static approach randomly selects the representative subset of scenarios before the DSE is started. This scenario subset will be subsequently be used to evaluate the mappings in a single GA for searching the design space. The exhaustive approach, on the other hand, uses the complete scenario database to evaluate each individual mapping.

General settings of the GAs of the solution and problem spaces are given in Table 4.1. The settings of the solution space are used for each of the approaches, whereas the settings of the problem space are only used for the coevolutionary approach. For both spaces, the first four parameters are traditional GA parameters: the size of the population, the number of offspring per generation and the probability for mutation and crossover. Based on the convergence of the initial experiments, the parameters are chosen and there is decided to run the exploration for 600 generations. During the exploration, the trainer is updated every ten generations. This gives a clear view of the complete convergence of the DSE. To show the variance of the different runs of the DSEs (the GA has stochastic nature), each experiment of the realistic workload is repeated twelve times, whereas each experiment of the stochastic workload is repeated six times.

Finally, each experiment will result in a Pareto front. These Pareto fronts can
be compared using the hypervolume indicator (see Section 2.3). Before the hypervolume can be calculated, the resulting mapping fitness values must be exhaustively evaluated. Remember that both the coevolutionary and the static approaches use a representative subset of scenarios to estimate the fitness. Most likely both approaches will use different subsets and thus the mappings must be evaluated using the same scenarios before they can be compared. Moreover, for each 15th generation the current Pareto front is logged such that it can be evaluated afterwards. In this way, the convergence can be observed.

4.4.1 Real Workload

For the real multi-application workload, two applications are used: a MJPEG encoder and a MPEG4 decoder (only for the simple profile, so without any frame reordering). The workload has a total of 131 scenarios: the MJPEG encoder has eleven intra-application scenarios, whereas the MPEG4 decoder has ten scenarios. As all the possible inter-application scenarios are valid (the applications can both run individually and simultaneously) the total number of scenarios is $10 + 11 + 10 \times 11 = 131$.

The statistically selected subset contains five scenarios, whereas the dynamically selected subset, on the other hand, may maximally contain four scenarios. This distinction is done deliberately as in this way the average number of Sesame invocations per generations is exactly the same for the static and dynamic selection. The additional effort that the coevolutionary approach requires for evaluating the trainer mappings is compensated in the static approach by using a larger subset of scenarios. In this manner, a fair comparison can be made as both approaches have a similar running time.

Figure 4.7 shows the results of the experiments. One of the important concerns of a GA is the time it takes until the resulting Pareto front stabilizes (i.e., convergence). For this purpose, we have analyzed the hypervolume over time. The results of this analysis are shown in Figure 4.7a. For both the static and the coevolutionary approach the average hypervolume is shown (averaged over the different repetitive experiments). Both of the approaches converge relatively quickly. Within 100 generations the Pareto front is more or less stabilized. The coevolutionary approach, however, converges to a better Pareto front. On average, the hypervolume is 0.02 larger. With respect to the found mappings, this means that the mappings found by the coevolutionary approach have an execution time that is on average 2 percent smaller.

This is not such a large difference. Most likely, the statically selected subset is good enough to correctly predict the ranking of the solution individuals. In order to verify this, we have calculated the Spearman’s rank correlation of all the representative subsets that are obtained. For this purpose, a large set (more than 14K) of exhaustively evaluated solution individuals is used. This set is unequal to the trainer and, therefore, the representative subset is not specifically trained on this set of solution individuals. Recall that when the Spearman’s rank correlation is closer
Figure 4.7: *Experimental result of DSE using a real multi-application workload*

to 1.0, the ranking of the representative subset provides a better resemblance of the real ranking. Results in Figure 4.7b shows the average ranking correlation over the different runs. It is indeed clearly visible that the quality of the subsets for the static and coevolutionary approach is comparable. As the static approach selects the representative subset before the DSE is performed, the quality of the subset does not change over time. Therefore, the correlation of the static approach (the horizontal red line) remains fixed at approximately 0.97. The scenarios in the coevolutionary
subset change over time, but its quality still fluctuates around 0.97 (the blue dashed line). The range of values for the coevolutionary approach is slightly better as shown by the blue shaded region. The best subset found by the coevolutionary approach has a ranking correlation of 0.98, whereas the best statically selected subset has a ranking correlation of 0.97.

A conclusion that can be drawn from this experiment that with this real multi-application workload the coevolutionary approach does not significantly benefit with respect to the static approach. The reason is the lack of diversity in the application workload. As the diversity is relatively small, it is quite likely that a randomly selected subset is already representative for the complete scenario database.

4.4.2 Synthetic Workload

To assess the merit of the coevolutionary approach for multi-application workloads with a higher diversity, we also preformed our previous experiment using a synthetic workload. The synthetic workload contains 18 applications with a total of 72 application tasks. With 27 inter-application scenarios and 51 intra-application scenarios, the workload now has a total of 514 scenarios. The static scenario subset uses a subset of ten scenarios and the coevolutionary approach has a maximal subset size of six scenarios. As discussed in the previous section, this makes sure that the number of Sesame invocations is the same for the static and the coevolutionary approaches. The consequence is that the coevolutionary approach must put additional effort to let its representative subset outweigh the random scenario subset.

In contrast to the previous section, we also included a DSE where the mappings are evaluated exhaustively. The exhaustive approach is only run once, as its running time is an order of magnitude larger than the static and coevolutionary approaches that use fitness prediction. This is clearly visible in Figure 4.9. The 420 generations of the exhaustive approach already require 43 million Sesame invocations, whereas the static and coevolutionary approaches only require a million Sesame invocations for 600 generations. For a Sun Fire X4440 with four quad-core AMD Opteron 8356 processors running at 2.3GHz, this means that the exhaustive approach has a running time of 23 minutes per generation. The static and the coevolutionary approaches, on the other hand, only take 36 seconds per generation.

Figure 4.8 shows the results of the experiment. In Figure 4.8a the Pareto front convergence for the different approaches is shown. Evidently, the exhaustive approach outperforms the other approaches. At each generation the average hypervolume of the exhaustive approach is higher (i.e., better) than the other approaches. As the SPEA2 based GA uses elitism and the exhaustive approach evaluates the mapping using the real fitness, the strong individuals that are found will always be kept in the population. This leads to a hypervolume that is monotonically increasing. However, the exhaustive approach is much more expensive. As discussed earlier, it is more than 30 times slower than the static and coevolutionary approaches.

The static and the coevolutionary approaches have a similar running time. With
Figure 4.8: Experimental result of DSE using a real multi-application workload

respect to convergence, the coevolutionary approach has better results than the static approach. More precisely, the average hypervolume (shown with a line within Figure 4.8a) of the coevolutionary approach is better starting from the first measuring point. Where the hypervolume of the coevolutionary approach increases almost monotonically, the average hypervolume of the static approach even drops between the 200th and the 300th generation. In this case, the fitness prediction fails to predict the correct ordering. For the coevolutionary approach, however, the representative subset
is adequate for keeping the good individuals in the population. Hence, the selection manages to get a good representative subset with only 1.2 percent of the total number of scenarios.

More importantly, the coevolutionary approach does not only obtain better results, but the outcome of the DSE is also more stable. In Figure 4.8a the variance of the static and coevolutionary approaches is highlighted using a shaded region. The variance of the coevolutionary approach is shown with blue and for the static approach it is shown using a red region. These regions can also overlap, which is shown using the somewhat darker region. In the initial phases, not only the hypervolume of the coevolutionary and the static approaches is similar, but also the variance. Over time, the coevolutionary approach is improving the subset. Therefore, the subset prediction can be improved. As a result, the hypervolume improves and the variance becomes smaller. For the static approach, the subset remains the same. In some cases, the initial subset is adequate for identifying decent Pareto fronts. Due to the randomness of the static approach, however, some poor subset will also lead to poor Pareto fronts. A poor subset will be uncertain about the mapping ordering and will make incorrect conclusions with respect to Pareto dominance. On top of that, the longer the exploration takes, the more variance there will be between the quality of the obtained Pareto fronts of a decent and a poor scenario subset.

The poor subsets can also clearly be seen when the subset quality is measured. Just as in the previous experiment, a large set of exhaustively evaluated individuals is used to obtain the ranking correlation of the used scenario subset with the real ranking. For the results of the static and the coevolutionary approach, as shown in Figure 4.8b, different visualizations are used. As the static subset does not change over time, the subset quality is also constant over time. Hence, a straight line is used

Figure 4.9: A comparison of the number of Sesame invocations for the different approaches. The horizontal axis shows the number of generations, while the logarithmic vertical axis shows the total number of Sesame invocations made during the earlier generations.
with for error line (shown at generation 0, 300 and 600) that shows the extremes of
the subset quality. On average, the ranking correlation is 0.88. The quality variance,
however, is quite large: the subset of the best run has a ranking correlation of 0.95,
whereas the subset of the worst run has a ranking correlation of 0.75. In contrast to
the static approach, the quality of the coevolutionary approach changes over time.
That is why a line is used for the average quality and a shaded region to show the
variance. Initially, the subset quality of the coevolutionary approach is similar to
the static approach (between the 0.85 and 0.95). However, the quality of the subset
quickly becomes better. Eventually, the quality of the scenario subset stabilizes
between 0.96 and 0.97.

So, for a scenario database with more diversity the coevolutionary approach
clearly outperforms the static approach. Even the subset of the worst run of the
coevolutionary approach is better than the subset of the best run of the static ap-
proach. The higher diversity in the scenario subset makes it hard to quickly select a
good scenario subset.

4.4.3 Trainers

As a final experiment, we have visualized both a trainer from the real and the syn-
thetic workload. The trainers are shown in Figure 4.7c and 4.8c. The line shows the
Pareto front that is found in the specific DSE. For embedded system designers this is
the most important outcome of the DSE that shows the trade-off between perform-
ance and cost. Within the graph also all the final trainer mappings are shown using
markers. It can be seen that most of the selected training mappings are close to the
Pareto front. The GA in the solution space mostly focuses on this area as it tries
to improve the Pareto front. In general, the differences between these mappings are
small and, therefore, the mappings are hard to differentiate. As a result, imprecise
fitness prediction is more likely to give a better fitness to the inferior mappings. Due
to this incorrect fitness, the SPEA2 based genetic algorithm may create offspring
mappings of a lower quality as the wrong parents are selected.

Also observe that in Figure 4.8c some of the training mappings dominate the
mappings that are found in the final Pareto front of the solution space. This issue
will be resolved in the next chapter, where the final outcome of the DSE will also
take the training mappings into account.

4.5 Related Work

High-level modeling, simulation for MPSoC performance evaluation and GA based
DSE [25, 15] are relatively mature research areas. The majority of the research used
to focus on the system exploration under a single (fixed) workload. Recently, research
has been initiated on making the DSE scenario aware [67, 59, 52, 16].

One of the techniques is by using the scenario-aware dataflow model [71, 72].
The scenario-aware dataflow model allows the modeling of dynamic applications using application scenarios. Multiple analysis tools are available to obtain metrics like throughput and latency. However, the disadvantage of analysis in comparison to simulation is that analytical models quickly become very complex. Hence, there are two options: 1) the architecture is modeled in an abstract manner (e.g., properties like resource contention are not taken into account) or 2) the architecture is completely modeled. In the first case analysis is fast but it may be imprecise. For the second case, the analytical model quickly becomes too complicated or too computationally expensive.

Another type of scenario is the use-case scenario. A use case can be compared with what we call inter-application scenario: it describes which applications can run concurrently. Examples of frameworks deploying such use cases are MAMPS [40] and a framework of Benini [4]. MAMPS is a system level synthesis tool for mapping multiple applications on an FPGA. The framework of Benini, on the other hand, reconfigures the embedded system whenever a new use case is detected. A notion of multiple applications can also be incorporated using a multimode multimedia terminal [26] that captures the inter-application scenario behavior in a single, heterogeneous model of computation. In this model an application only has fixed behavior that is described using a dataflow model, whereas a finite state machine describes the transition between the different inter-application scenarios. Where scenario-aware dataflow uses the finite state machine to switch between the operation modes of a single application, the multimode multimedia terminal switches between different sets of active tasks. The work of [64] also uses a finite state machine to model the dynamism within a multi-application workload. In this case, each application can be running or is paused. A scenario transition is explicitly done by starting, stopping, pausing or resuming an application.

None of these approaches describe both the intra-application scenarios and the inter-application scenarios. To the best of our knowledge, our scenario-based DSE is the first to address both inter- and intra-application scenarios during the DSE of MPSoCs with multi-application workloads.

On top of that, to the best of our knowledge, coevolution is also not yet used in the field of DSE of MPSoCs. Other research areas, however, have applied coevolution to reduce the number of scenarios or test cases for the efficiency improvements of GAs. Branke and Rosenbusch [8] use coevolution to improve the efficiency of their worst-case optimization problem. This problem tries to optimize the variable $s$ of the mathematical function $F(s, t)$ such that the worst-case value for all possible test cases $t$ is as high as possible. The interesting property for the worst case value is that as long as the worst-case test is in the population, the predicted fitness matches the real fitness. Therefore, no exhaustively evaluated trainer is required (in contrast to the average case optimization in our DSE problem).

Schmidt and Lipson [63] apply coevolution using an exhaustively evaluated trainer. Their optimization problem is to increase the efficiency of a GA by using a fitness
predictor. The fitness predictor is coevolved with the GA and consists of a subset of all the samples in the problem definition. A set of exhaustively evaluated individuals is used to judge the quality of the fitness predictor based on the absolute prediction error. This is one of the major differences with our work. Not only do we normalize the prediction error (in this way each objective is weighted equally), but also the relative ordering is taken into account. After all, the fitness predictor is only meant to predict which individual is better than the other. In this way, the DSE ends with best set of individuals. The fitness values of these individuals may have a systematic (absolute) error, but that does not affect the identified Pareto front.

4.6 Conclusion

This chapter introduced the use of application scenarios during the system-level DSE of MPSoC based embedded systems. More specifically, a novel scenario-based DSE is proposed based on a coevolutionary genetic algorithm. This approach searches simultaneously for optimal design instances and representative subsets of workload scenarios to efficiently evaluate these instances.

Our use cases showed that, as long as the diversity in the scenario database is large enough, the coevolutionary DSE clearly outperforms the DSE where a representative subset is selected statically. Additionally, it is an order of magnitude faster than a traditional DSE where the design instances are evaluated exhaustively using the complete scenario database. In case the diversity in the scenario database is low, the approach still can provide similar results than the static subset selection. As the static subset can compensate the overhead of the dynamic subset selection by using a larger scenario subset, this means that the computational overhead of the dynamic subset selection pays off.

The improvement of scenario-based DSE is twofold. First, the obtained Pareto fronts have a higher hypervolume. This means that the design instances that are found are faster and cheaper. Secondly, the dynamic subset selection of the scenario-based DSE also has a higher probability on a good outcome than a static subset selection. The variance of the hypervolume over the different runs for the scenario-based DSE decreases over time, whereas the variance in hypervolume for the different runs only increases over time. This means that the scenario-based DSE is less dependent on the stochastic nature of the GA leading to a more reliable exploration framework.

In the next chapter we will refine the scenario-based DSE by formalizing the solution and the problem space. Additionally, the problem space is completely revised.