Identifying Data-Dependent System Scenarios in a Dynamic Embedded System

E. Hammari[1], F. Catthoor[2], P. G. Kjeldsberg[1], J. Huisken[3], K. Tsakalis[4] and L. Iasemidis[4]
[1]Dept. of Electronics and Telecom., Norwegian Univ. of Science and Technology, Trondheim, Norway
[2]imec and Katholieke Univ. Leuven, Leuven, Belgium; [3]imec/Holst Centre, Eindhoven, The Netherlands
[4]Dept. of Electrical Engineering, Arizona State Univ., Tempe, AZ, USA

Abstract— *System scenario-based design methodologies are applied to reduce the costs of dynamic embedded systems. At design-time, the system is optimized for a set of system scenarios with different costs, e.g., alternative scheduling of tasks. At run-time, certain parameters are monitored, scenario changes are detected, and mapping and scheduling are reconfigured accordingly. In this process, optimized identification of parameters and system scenarios is essential. Previously, the parameters have been limited to control variables, or variables with a limited number of distinct values. This paper presents a scenario identification approach based on polyhedral partitioning of the parameter space for systems where the parameters may have a wide range of data-dependent values. We evaluate our approach on a biomedical application. The results indicate that with 20 system scenarios the average execution time cost can be reduced with a factor 3 and brought within 15% of the theoretically best solution for the workload-adaptive designs.*

Keywords: Application-specific embedded systems, run-time reconfiguration, system scenario-based design

1. Introduction

Increasingly, modern applications are becoming dynamic resulting in input data dependent variations in system costs, e.g., execution time and energy consumption. An over dimensioned solution based on a few extreme workloads can be very costly, or even impossible to implement, and a workload-adaptive reconfigurable design will be necessary [14].

System scenario based design methodologies [5] provide a systematic way of constructing workload-adaptive embedded systems and have been successfully applied to multiple designs in multimedia and wireless domains [3], [11], [12], [13], [15], [17], [18]. Through structural analysis and profiling of the application code at design-time, a set of system scenarios with different costs is identified along with the parameters that determine the cost variations. The system is then separately optimized for each system scenario and augmented with a scenario predictor and switching mechanism. At run-time, the active system scenario is predicted up front from the actual parameter values and the system is switched to the most cost-optimal configuration for this system scenario.

Note, that system scenarios are conceptually different from the more common use-case scenarios. While both of them aim at reducing the total costs, use-case scenarios are extracted from the obvious system parameters, modes or usage pattern which can be detected without detailed knowledge of the algorithmic implementation. System scenarios are identified from the observed costs and then characterized in terms of implementation parameters. System scenarios do not depend on obvious parameters, modes or usage patterns and can hence be efficiently applied even if the application do not contain any of them.

This paper targets the scenario identification technique in system scenario based design methodologies, in particular for systems having parameters with widely varying data-dependent values. Existing techniques assume that the parameters are control variables and/or that they have a limited number of possible parameter values. They make use of enumeration and apply a bottom-up approach to cluster these values into system scenarios [6], [4].However when the parameters are data-dependent, they may have thousands or even millions of possible data values making bottom-up clustering and enumeration-based prediction impractical (see Section 3). Our method should then instead be used because it performs a scalable top-down polyhedral partitioning of the parameter space. This is our first main contribution.

Secondly, we apply our scenario identification technique to a real application and demonstrate the feasibility of our approach for different number of system scenarios.

The paper is organized as follows. Section 2 gives a motivating example for our work. In Section 3 the existing techniques for scenario identification are reviewed and the necessary terminology is introduced. Our proposed approach for scenario identification is detailed in Section 4. Experimental results are presented in Section 5, followed by our conclusions and plans for future work.

2. Motivational example

Recent biomedical applications for outpatient care have a dynamic nature and are at the same time subject to strict cost constraints. They continuously monitor patient's signals for

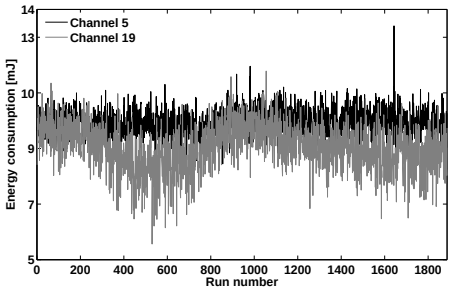

Fig. 1: Energy consumption of Lyapunov exponent calculation in 6 hours EEG recording.

an anomaly and perform specific tasks when it is detected. They may use complex signal processing on multiple channels and are required to be powered by a battery for months or even years. One such example is an epileptic seizure predictor, [7], [8], which tracks EEG signals from up to 32 channels and may warn patients of upcoming seizures hours in advance. A front-end part of this predictor performs calculations of Lyapunov exponents for each channel once every 10 seconds. Figure 1 shows the variations in the energy consumption of different runs of the Lyapunov exponent calculation of two channels over a six hour period. Due to different EEG input data, the energy consumption of one calculation can vary widely from 6 mJ to 13 mJ. The peak energy consumption for this application occurs only once in the 6 hours long EEG recording. A system designed based on this worst case energy consumption will consume 829 J/channel while processing the recording.

An ideal workload-adaptive system is able to reconfigure the system optimally in each run so that it consumes the minimum amount of energy possible. A heavily optimized thread-level workload-adaptive design for the Lyapunov exponent calculation will require only 567 J/channel for the same recording. However it can never be built in practice as the costs of reconfiguring such system, storing the different configurations and predicting them would be excessive.

A system scenario-based design methodology uses the same concept of adaptively reconfiguring the system, but allows only a limited set of possible configurations. A given system scenario has a fixed system cost corresponding to its system configuration. It contains the group of runs for which this configuration is better than any of the other configurations in the limited number of scenarios. For most runs the system will then require a small energy overhead compared with running on the optimal configuration. E.g., with 10 scenarios, the Lyapunov exponent calculator will consume 594 J/channel processing the EEG recording above. That is, somewhat more than the ideal workload-adaptive system, but far less than the system based on the worst case energy consumption. There will be an added energy consumption related to the scenario detection and reconfiguration, but this can be kept low if the guidelines for scenario based design is followed [5].

The Lyapunov exponent calculator is a good example of an applicatin were more traditional use-case scenarios can not be applied. It repeatedly performs the same calculation on equal-sized packages of input data. A system scenario approach can, however, be used to exploit the potential benefits of a reconfigurable platform.

The system scenario-based design methodology is a powerful tool that can also be used for fine grain optimizations at the task abstraction level and for simultaneous optimization of multiple system costs. The ability of handling multiple and nonlinear system costs differentiates system-based design methodologies from the dynamic run-time managers intended for DVFS type platforms [10]. DVFS methodologies concentrate on optimization of a single cost - the energy consumption of the system, that scales monotonically with frequency and voltage. They perform direct selection of the system reconfiguration from the current workload situation. This, however, cannot be generalized for costs that depend on the parameters in a nonuniform way. That makes the decision in one run-time step too complex. Scenario-based design methodologies solve this problem by a two-stage approach decided both at run-time: they first identify what scenario the working situation belongs to and then choose the best system reconfiguration for that scenario. Since the relationship between the parameters and the costs will in practice be very complex, the scenario identification is however performed at design-time.

This paper targets fine grain optimization of a single system cost.

3. Theory and related work

The term *Run-Time Situation (RTS)* is an important concept used in task level system scenario-based design methodologies [5]. Each instance of running a task has a corresponding cost (e.g., energy consumption). The run instance and its cost is treated as a unit denoted an RTS. One complete run of the application on the target platform represents the sequence of RTSs.

A scenario identification technique lies at the heart of any system scenario-based design methodology. It determines how the different observed RTSs should be divided into groups with similar costs - the system scenarios, and how the system scenarios should be represented to make their runtime prediction as simple as possible.

Two examples of techniques for task-level scenario identification are presented in [6] and [4]. Both of them split scenario identification into two steps. In the first step, the variables in the application code are analyzed, either statically, [6], or through profiling of the application with a representative data set, [4]. The variables having most impact on the runtime cost of the system are determined. These variables are called RTS parameters, denoted by $\xi_1, \xi_2, \ldots, \xi_k$, and are used to characterize system scenarios

and design the scenario prediction mechanism. Typically a small set of RTS parameters is selected to keep the runtime prediction overhead low.

The output of the first step is the set of the selected RTS parameters and, for each RTS i, its RTS *signature* is given by Equation 1 below:

$$r(i) = \xi_1(i), \xi_2(i), \ldots, \xi_k(i); c(i), \quad (1)$$

containing parameter values $\xi_1(i)$, $\xi_2(i)$, ..., $\xi_k(i)$ and the corresponding task costs $c(i)$. I.e., each run instance of each task will have its own RTS signature. The number, N, of RTS signatures will hence be very large. Depending on the number of RTS parameters and how many different values each of them can take, there will be a small or large number of *different* RTS signatures. This is important for the complexity of step 2 of the scenario identification.

In the second step, the RTS signatures are divided into groups with similar costs - the system scenarios. In [6] and [4] this is done by a bottom-up clustering of RTS signatures with a resulting multi-valued decision diagram (MDD) that is used as a predictor for the upcoming system scenario. The limitation for this technique is that the size of an MDD explodes for many-valued parameters making it infeasible for the runtime prediction.

The two techniques differ in how they evaluate the impact of RTS parameters. The first one [6] is based on pure static analysis of the code and do not take into consideration the frequencies of occurrence of different RTS parameter values. It may therefore produce system scenarios that almost never occur. The second one [4] extends the first one [6] with profiling information and forms a system scenario set that exploits runtime statistics. Our scenario identification technique uses the same approach for the selection of RTS parameters as the second technique [4]. This approach typically leads to only a limited amount of parameters being labeled as important enough to incorporate in the identification step. That is crucial to limit the complexity.

Identification of thread-level system scenarios has been studied in [16]. This is fully complementary to the focus of our paper which considers intra-thread-level system scenarios.

As we have seen the existing scenario identification approaches cannot be used in a system with many-valued RTS parameters, causing an explosion of states in the MDD and the associated runtime/implemenation costs. In the next section we will discuss a possible solution to this problem.

4. Proposed method

4.1 General overview

Figure 2 illustrates the theoretical concepts of our scenario identification technique. Given k RTS parameters and N profiled RTS signatures from Equation 1. If we assign one dimension to each RTS parameter, the resulting k-dimensional

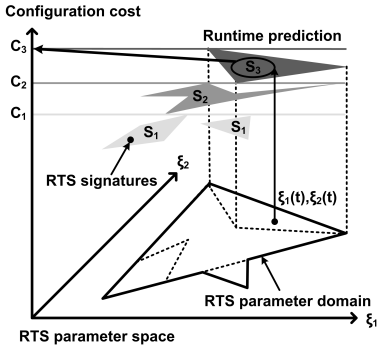

Fig. 2: System scenarios and runtime prediction.

space will define all theoretically possible values for the RTS parameters in the application. We will call such space an *RTS parameter space*. When static max and min constraints on the values are added, the space reduces to one or several k-dimensional domain(s).

Assuming a $(k+1)$-dimensional cost representation for each RTS, all signatures can then be plotted as points in a $(k+1)$-dimensional space. In the profiling sequence, several identical signatures may exist, giving coinciding points in the space. The number of times a point repeats itself is useful information as it quantifies the probabilities of occurence of each RTS.

With the representation above, the scenario identification task can be viewed as a distribution of points into S different groups, representing system scenarios, according to which the overall configuration cost is minimized. An RTS point i is assigned to scenario j whenever its cost $c(i)$ falls into that scenario's cost range $\{C(j)_{min}, C(j)_{max}\}$. The scenario cost ranges are determined by a balancing function that ensures that all scenarios have a near-equal probability to occur at run-time. In this way, rare system scenarios are avoided since their storage cost will exceed the gains of adding them. We measure this probability by the number of points, including the repeating ones, that each scenario contains and call it *scenario size*. The maximum scenario size equals the number of all RTS signatures N, divided by the number of system scenarios S. Given a list r of RTS signatures sorted by descending cost, the scenario cost ranges are given by the indices corresponding to the integral number of the scenario size:

$$C(j)_{max} = r\left((j-1) \cdot N/S + 1\right) \quad (2)$$
$$C(j)_{min} = r\left((j) \cdot N/S\right) \quad (3)$$

The *cost* of scenario j is defined as the maximum cost of any of the RTS signatures that it includes: $C_j = C(j)_{max}$.

The projection of scenarios onto the RTS parameter space (see Figure 2) will produce $M \geq S$ regions that characterize the system scenarios in terms of RTS parameter values. Each region can be described as a polyhedron, and the runtime scenario prediction can be done by checking which polyhedron contains the RTS parameter values of the next

RTS. Since we know which scenario the region belongs to, we can foresee that the next running cost will be no more than the cost of that scenario. Checking if a point lies inside a polyhedra is the classical point location problem from the computational geometry domain, and the advantage of using it for prediction instead of MDD is that it operates on/stores only the vertices of polyhedral regions, not all possible parameter values.

This top-down approach can handle arbitrary large domains, provided that the number of regions stays reasonably low. Otherwise prediction overhead will grow. The number of regions depends on the number of system scenarios and the underlying structure of the system - the relationship between the cost locality of RTS points and the value locality of their RTS prameters.

The desired number of system scenarios is best defined by the user according to the characteristics of the application domain. Typically this is limited to a few tens because beyond that the potential gains in better following the system dynamics are counterbalanced with the additional cost complexity of detecting and exploiting the (too) large set of possible system scenarios.

For the biomedical application that we are investigating, a strong correlation is present in the locality of the RTS parameter values and the locality of the corresponding costs on the target DSP platform, resulting in a single region per scenario (see Figure 5). The locality of parameters and the corresponding costs is an important prerequisite for the efficiency of the current scenario identification technique. Moreover, we assume that that there is a one-to-one correspondence between the cost of a scenario and the system configuration. In other words, we target the systems were scenarios are mostly defined by the application characteristics and not by the details of the platform.

4.2 Detailed algorithm

Our scenario identification algorithm, GENERATESCENARIOSET, is presented in Figure 3. Line 2 is a preprocessing step, where profiled RTS signatures are sorted by their costs starting from the maximum cost. In line 4 the worst case system scenario is created. In lines 6 to 8, the system scenario is filled in with signatures having the next costs in the sorted sequence. When the size of the system scenario exceeds the maximum allowable size, a new system scenario is created (line 17).

Finally, in lines 10 to 15, each completed system scenario is checked for overlap with previously calculated higher cost system scenarios. An overlap means that the scenario regions in the RTS parameter space are not disjoint (see example on Figure 6), and equals the intersection of the regions: *overlap* $\leftarrow$ *scenario1.paramRegion* $\cap$ *scenario2.paramRegion*.The intersections make prediction of scenarios ambiguous and have to be eliminated. This is done by subtracting the overlap region from the lower cost system scenario and moving all signatures in the overlap region to the higher cost system scenario. Given $C_1 > C_2$, these operation can be written as:
scenario2.paramRegion $\leftarrow$ *scenario2.paramRegion - overlap*
scenario1.paramRegion $\leftarrow$ *scenario1.paramRegion* $\cup$ *overlap*

The complexity of this algorithm is calculated below:

$$\mathrm{O}(N,S) = N\mathrm{O}(\textsc{AddSignature}) + (1/2)S(S-1)(\mathrm{O}(\textsc{Overlap}) + \mathrm{O}(\textsc{AdjustBorder})) + S\mathrm{O}(\textsc{NewScenario}) \quad (4)$$

Thus, it depends on the complexity of the underlying geometric algorithms in the labelled functions.

For the fixed number d of RTS parameters, the function NEWSCENARIO has a constant complexity O(1) as it only copies the value of each parameter in a single RTS signature to a scenario region.

The function ADDSIGNATURE performs a CONVEXHULL operation on the existing border of the scenario and the projection point of the new RTS signature onto the RTS parameter space. For a 2 or 3-dimensional RTS parameter space an incremental convex hull algorithm has the complexity O($n \log n$) [9], where n is the final number of processed points, which here equals to the scenario size, N/S. The convex hull of a polygon has the expected number of $v = \mathrm{O}(\log n)$ vertices and many of them may lie very close to each other. To limit the number of vertices in the hull for faster run-time prediction, we modify the algorithm, such that it calculates the distance between the points on the hull and removes those that are closer than L/v_{max}, where L is the perimeter of the hull, and v_{max} is a user defined constraint of the maximum number of vertices in the prediction polyhedra. For the application that we investigate a reasonable value of this parameter could be 10 (see Figure 5).

The functions OVERLAP and ADJUST BORDER apply boolean set operations for intersection, difference and union of two d-polytopes. For the case $d = 2$ and $d = 3$ these operations can be done in O($v_{max} \log v_{max}$) time [2], giving the total complexity of the algorithm:

$$\mathrm{O}(N,S) = S\mathrm{O}((N/S)\log N/S) + (3/2)S(S-1)\mathrm{O}(1) + S\mathrm{O}(1)$$
$$\mathrm{O}(N) = \mathrm{O}(N \log N) \quad (5)$$

Recall that this scenario identification algorithm is run only in the design phase of the embedded system. The run-time prediction of the next scenario is equivalent to a point location problem in the polyhedral partitioning of the parameter space. The time complexity of the point location problem is O($\log v_{tot}$), where $v_{tot} = S \cdot v_{max}$ is the total number of vertices in the partitioning. The required memory space is O($v_{tot} \log v_{tot}$). To compare, an MDD for d parameters with l distinct values has l^d states and a query time of O($d \cdot l$), where $l \gg v$.

For $d > 3$, i.e., for systems with more than 3 parameters, the complexity of convex hull, boolean set operations and the point location algorithm, increases exponentially in d [2], similar to MDD. It remains an open research area to find

```
GENERATESCENARIOSET(SET rtsSignatures, INT S)
 1  SCENARIO solutions ← ∅
 2  SORTBYCOST(rtsSignatures)
 3  wcSignature ← rtsSignature(1)
 4  currentScenario ← NEWSCENARIO(wcSignature)
 5  MAXSCENARIOSIZE := N/S
 6  for signature in rtsSignatures do
 7     if (currentScenario.size < MAXSCENARIOSIZE) then
 8        currentScenario.ADDSIGNATURE(signature)
 9     else
10        for scenario in solutions do
11           overlap ← OVERLAP(scenario, currentScenario)
12           if (overlap ≠ ∅) then
13              ADJUSTBORDER(currentScenario, overlap)
14           end if
15        end for
16        solutions.INSERT(currentScenario)
17        currentScenario ← NEWSCENARIO(signature)
18     end if
19  end for
20  return solutions

NEWSCENARIO(RTSSIGNATURE signature)
   SCENARIO new
   new.size ← 1
   new.cost ← signature.cost
   new.paramRegion ← signature.paramValues
   return new

ADDSIGNATURE(RTSSIGNATURE signature)
   thisScenario.size ← thisScenario.size + 1
   thisScenario.paramRegion ←
                  CONVEXHULL(thisScenario.paramRegion,
                             signature.paramValues)
   return
```

Fig. 3: Scenario identification algorithm

an efficient representation of polytopes in higher dimensions that will decrease the complexity of the algorithms operating on them. The complexity does not increase exponentially when the number of possible parameter values increase, however, the way it does for MDD. Thus, our algorithm is efficient for the systems with upto three important data-variables that determine the data-dependent behaviour of the system and have a significant number possible values.

4.3 Refining system scenarios

The function ADDSIGNATURE in Figure 3 produces convex scenario projections in the RTS parameter space. An example of convex scenarios is shown on Figure 5. However, in some situations concave scenario projections, representing a geometrically tighter envelop of a set of points, are preferrable. This is the case when: a) the inherent correlation between the RTS parameter values and the corresponding costs has a concave shape, b) the system scenarios overlap in the RTS parameter space and complete migration of the signatures to the higher cost system scenarios results in considerable reduction of run-time gain. An example of concave scenarios is presented in Figure 6.

A concave scenario projection reduces the overlap and can potentially improve the run-time gain. However, large overhead may incur since algorithms processing concave polyhedra are much more complex. A possible solution is to split the concave projection into a set of convex polyhedra at design-time and apply convex hull algorithms. The separate polyhedra require still additional storage and processing time that should be kept low. To achieve that, a restriction must be made on the number of reflex angles v_r in the concave projection, and also a careful consideration of the cost tradeoffs is required. The refinement step is performed before the OVERLAP and ADJUST BORDER functions in Figure 3 and currently includes only a geometric refinement of the border by manual selection of additional vertices. The cost tradeoff considerations are the goals of our future work.

It should be noted that scenario overlaps may also be produced by variables affecting the costs, but not selected as RTS parameters, or by nondeterministic properties of the underlying platform, resulting in different costs for the same RTS parameters. Such overlaps indicate either a faulty RTS parameter selection or the use of hardware components unsutable for scenario-based design.

5. Experiments and Conclusions

We have evaluated our scenario identification algorithm on two versions of the Lyapunov exponent calculator described in Section 2. Energy numbers in Section 2 are obtained using the CoolBio DSP platform, presented in [1], and have been extracted through layout back-annotated power simulations. In our case we use the high performance mode, running the DSP on 1.1V. For this voltage we reach 80MHz speed which is required for this application in the worst-case condition.

In this section we present results for execution time optimization using system scenarios. The improved execution time can be exploited for reconfiguration in several ways. DVFS can be applied, possibly in combination with rescheduling to allow other tasks to run in the idle time. On run-time reconfigurable multi-processor platforms, remapping of tasks is possible to achieve an overall optimized execution.

We have run tests on three different setups, displayed in Table 1. Throughout the tests we have varied: a) the version of the application, i.e. different settings for the Lyapunov exponent calculation, b) the platform, on which the execution time was measured, and c) the input database for application profiling. This results in different characteristics which represent distinct benchmarks to test our algorithm performance. Figures 5 and 6 show the results of the experiments. Prior to scenario identification, an RTS parameter selection step, similar to [4], was performed, and two internal application variables with the greatest impact on the execution time were identified - *Falsecount* and *Seqlength*. The same variables were identified in both application versions and they have upto a few thousands of distinct values. They were selected as RTS parameters, giving a two-dimensional RTS parameter space for our scenario identification algorithm.

Table 1: Experimental setup.

No	Application version	Platform	Database
1	I, settings: nsize=2048, dimm=7, evolv=12, idist=20, tau=4	CoolBio DSP	6 hrs continuous EEG w/seizures
2	II, settings: nsize=1000, dimm=4, evolv=6, idist=12, tau=4	CoolBio DSP	6 hrs continuous EEG w/seizures
3	I, settings: nsize=2048, dimm=7, evolv=12, idist=20, tau=4	General purpose	200 EEG samples from epileptogenic zone, no seizures

The results of the first two experiments are presented on Figure 5. Here both application versions are investigated on the potential target embedded platform, CoolBio DSP. A profiling of the application with a 6 hrs continuous EEG recording shows that there is a clear correlation between the RTS parameters and the execution workload (in clock cycles), and our scenario identification algorithm produces a set of non-overlapping system scenarios. The two application versions have different size of the RTS parameter domain, but the identified system scenarios appear to be very similar, approximately a scaled version of each other. A relatively small number of 5 to 20 system scenarios is generated as preferred by users in scenario-based systems.

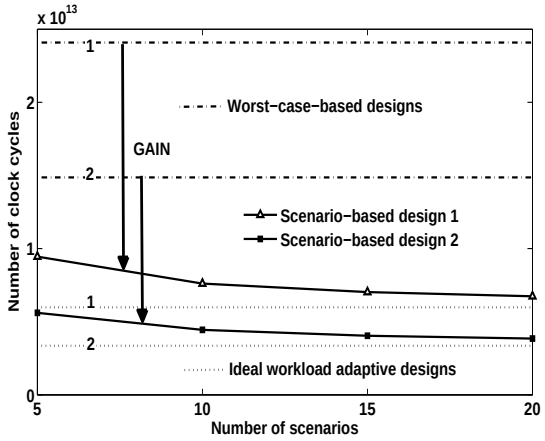

Fig. 4: Total execution time of the largest Lyapunov exponent calculation (versions I and II) with different number of system scenarios in the 6 hours continuous EEG recording.

The last experiment is run on a nondeterministic general-purpose desktop PC to demonstrate the scenario refinement step. Figure 6 presents the results. The top left subfigure shows the overlapping convex scenario projections produced by function ADDSIGNATURE on this nondeterministic platform. The remaining subfigures show the projection of each scenario separately along with its respective RTS point distribution. Concave refinement is performed on scenarios 2-5 in order to reduce the overlaps between the scenarios before the functions OVERLAP and ADJUSTBORDER are applied. Although a much smaller database is used for this experiment, the real scenario borders from Figure 5 can be discerned in the point distributions of Figure 6. The distortion in the scenario borders is caused by the noise from the nondeterministic platform. The dashed line indicates the convex hull of each scenario. It is determined by the extreme points in the distribution and causes strong overlap between the scenarios. In fact, the overlap is so big, that after application of OVERLAP and ADJUSTBORDER functions some of the scenarios would disappear totally. The solid line is the concave hull of the scenarios and comes significantly closer to the real scenario borders, reducing the overlap between them. If the point distributions here were the inherent point distributions for this application (i.e. not caused by the platform noise), the identified concave scenarios would improve the execution time of this system scenario-based design.

Figure 4 compares execution workload for running the system with and without system scenarios for the first two experiments. It also shows the execution workload of the theoretically optimal workload-adaptive design which is not realizable in practice. The results are presented for both application versions. Between 61% and 72% gain can be achieved with 5 to 20 system scenarios. With 5 system scenarios the total execution workload of the systems is still situated well above the theoretically best solution - 58% for system 1 and 67% for system 2. When the number of scenarios is increased, however, the total execution time of both systems reduces towards the theoretical limit and becomes at 20 scenarios less than 13% and 15% above the theoretically best solution for system 1 and system 2 respectively.

The results presented here demonstrate the feasibility of the proposed technique and show that it is possible to reach near optimal execution time with a limited number of scenarios. Future work includes optimization of scenario borders to realize a trade-off between overestimation and run-time prediction / switching complexity.

References

[1] M. Ashouei et al., "A voltage-scalable biomedical signal processor running ecg using 13pj/cycle at 1mhz and 0.4v," in *Proc. ISSCC'11*, 2011, pp. 332–334.

[2] M. J. Atallah and M. Blanton, Eds., *Algorithms and theory of computation handbook: special topics and techniques*, 2nd ed., NY, USA: Chapman & Hall/CRC, 2010.

[3] B. Geelen., "Low-power, Wavelet-based Applications in Dynamic Environments", PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, Dec. 2008.

[4] S. V. Gheorghita, T. Basten, and H. Corporaal., "Scenario selection and prediction for dvs-aware scheduling of multimedia applications", *J. Signal Process. Syst.*, vol. 50, pp. 137–161, Feb. 2008.

[5] S. V. Gheorghita et al., "System-scenario-based design of dynamic embedded systems", *ACM Trans. Des. Autom. Electron. Syst.*, vol.14, paper 3, pp. 1–45, Jan. 2009.

[6] S. V. Gheorghita, S. Stuijk, T. Basten, and H. Corporaal, "Automatic scenario detection for improved wcet estimation", in *Proc. DAC'05*, 2005, pp. 101–104.

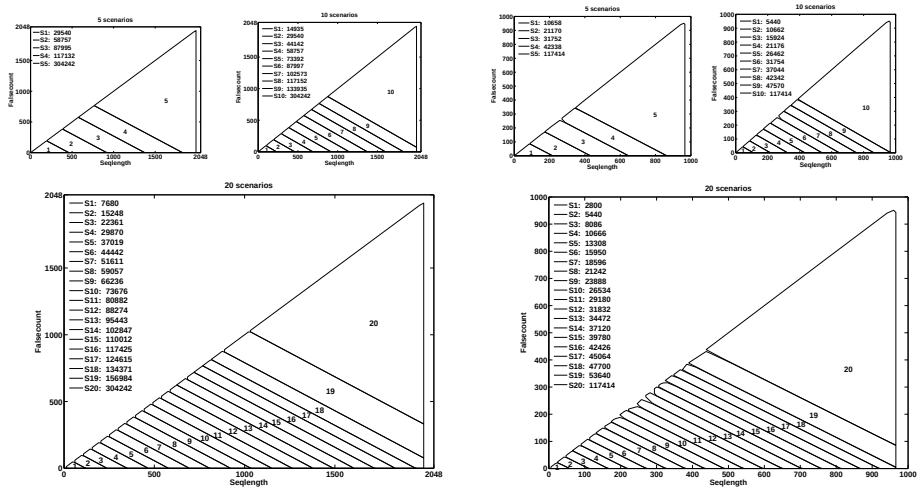

Fig. 5: System scenarios for the application version I(left) and II(right). Numbers in the top left corners are the execution times of scenarios in clock cycles

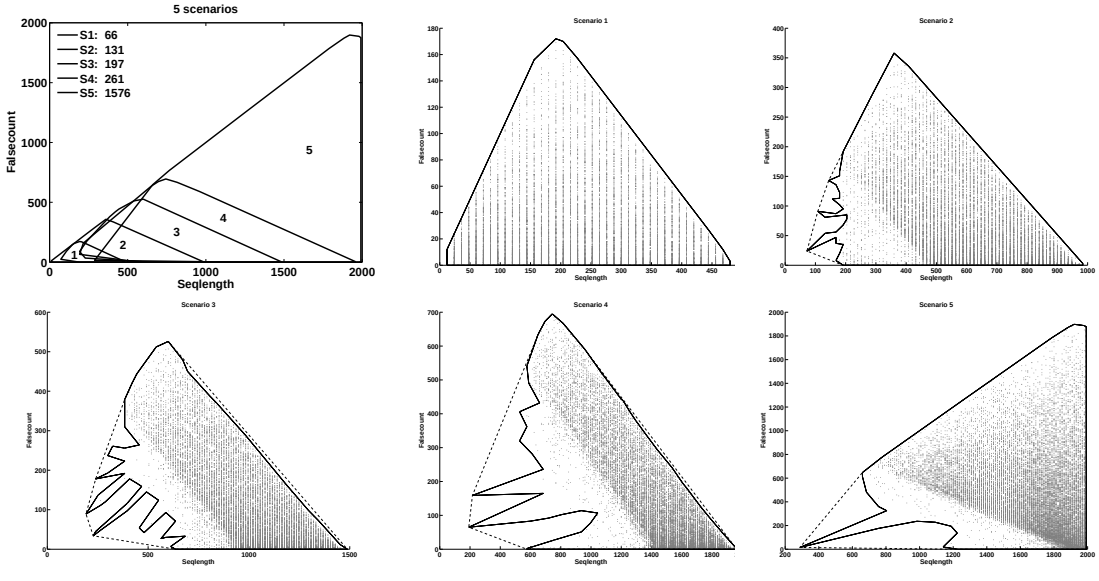

Fig. 6: Overlapping convex scenario projections and their concave refinement. Numbers in the top left corner are the execution times of scenario in μs.

[7] L. Iasemidis, "Seizure prediction and its applications", *Neurosurg. Clin. N. Am.*, vol. 22, pp. 489–506, Oct. 2011.

[8] L. Iasemidis et al., "Long-term prospective on-line real-time seizure prediction", *Clinical Neurophysiology*, vol. 116, pp. 532–544, Mar. 2005.

[9] M. Kallay., "The complexity of incremental convex hull algorithms in rd", *Information Processing Letters*, vol. 19, paper 4, p. 197, Nov. 1984.

[10] Y. Liu and H. Zhu, "A survey of the research on power management techniques for high-performance systems", *Softw. Pract. Exper.*, vol. 40, pp. 943–964, Oct. 2010.

[11] Z. Ma, "Interleaved Subtask Scheduling on Multiprocessor SoCs", PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, Jan. 2006.

[12] S. Mamagkakis, D. Soudris, and F. Catthoor, "Middleware design optimization of wireless protocols based on the exploitation of dynamic input patterns", in *Proc. DATE'07*, 2007, pp. 1–6.

[13] N. R. Miniskar et al., "Scenario based mapping of dynamic applications on mpsoc: A 3d graphics case study", in *Proc. SAMOS'09*, 2009, pp. 48–57.

[14] D. Raskovic and D. Giessel,"Dynamic voltage and frequency scaling for on-demand performance and availability of biomedical embedded systems", *IEEE Trans. Inf. Technol. Biomed.*, vol. 13, pp.903–909, Nov 2009.

[15] M. Steine et al., "Proactive reconfiguration of wireless sensor networks", in *Proc. MSWiM'11*, 2011, pp. 31–40.

[16] P. v. Stralen, "Scenario Based Design Space Exploration", PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, Sep. 2009.

[17] D. Stroobandt and K. Bruneel, "How parameterizable run-time fpga-reconfiguration can benefit adaptive embedded systems", in *Proc. ERSA'11*, 2011, pp. 184–194.

[18] N. Zompakis et al., "Enabling efficient system configurations for dynamic wireless baseband engines using system scenarios", in *Proc. SIPS'11*, 2011.