Proc. Industrial Embedded Systems, p. 125 - 133, Lisbon, Portugal, 2007

Design Space Exploration with Evolutionary Multi-Objective Optimisation

M. Holzer, B. Knerr, and M. Rupp
Vienna University of Technology
Institute for Communications and RF Engineering
Gusshausstr. 25/389, 1040 Wien, Austria
{mholzer, bknerr, mrupp}@nt.tuwien.ac.at

***Abstract*— High level synthesis is one of the next major steps to improve the hw/sw co-design process. The advantages of high level synthesis are two-fold. At first the level of abstraction is raised and secondly this allows for exploring the design space to a higher degree than on register transfer level. The design space can be explored by applying several kinds of source code transformations like loop unrolling and tree height reduction. Performing design space exploration manually leads to inefficient and inflexible mappings. This work describes the automated design space exploration for area timing trade-offs by utilising evolutionary multi-objective optimisation. Several performance improvements for genetic algorithms are introduced, which target a fast and accurate design space exploration. Finally, the performance of the proposed algorithm is evaluated.**

I. Introduction

One of the trends for increasing design productivity in the hardware/software co-design process is to raise the level of abstraction. Nowadays, the implementation process of most designs flows is based on descriptions on register transfer level (RTL). The advances to higher abstraction levels are at the one hand provided by languages for simulation (e.g. SystemC) and on the other hand by the capability of high level synthesis (HLS). The object of high level synthesis is twofold. First, by describing a system at higher level a productivity gain can be obtained. Secondly, because the design transformations occur on a higher level there is greater potential for exploring the design space, which should lead to better designs. The importance of this evolution has a direct impact on costs for a design. Figure 1 depicts the evolution of the cost during the development time [1], where it can be seen that early design decisions have a much higher resulting cost span than design decisions taken at the end of the development time.

The full advantage of such improvements can only be exploited by additional tools that allow for discovering trade-offs automatically, which has been pointed out by the International Technology Roadmap for Semiconductors [2]. Different realisations of processes are achieved by applying diverse source code transformations. Especially loop transformations like loop unrolling with varying unrolling factors as well as data flow transformations like tree height reduction are to be mentioned. Some high level synthesis tools allow for automatic application of these techniques (CatapultC from Mentor Graphics [3], SPARK [4]). Nevertheless, these decisions have to be performed manually and are based for example on `pragma` [5], [6] statements within the code in order to control the compilation task. This leads to inflexible mappings, which have to be updated when either the target hardware or the algorithm itself changes.

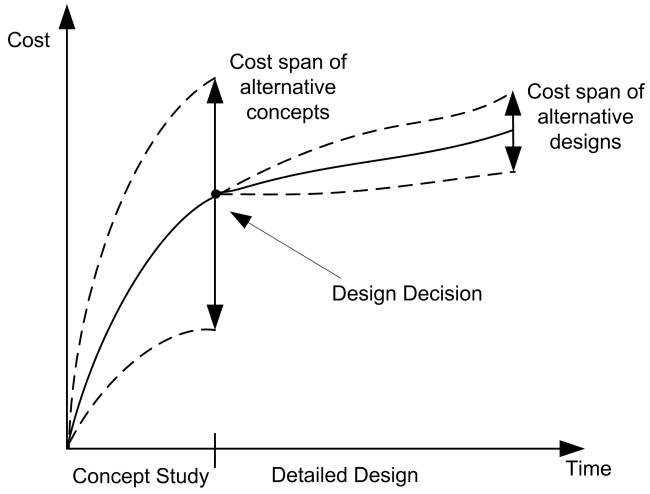


Fig. 1. Up to 90% of the costs are determined at the first part of the design.

Those transformations can be applied to code segments of the algorithm and the permutation of those independent transformations results in different realisations, that are called design points. Additional requirements from the specification translate into constraints on the set of feasible design points. For example maximum values for area as well as for timing are given. The design space grows exponentially with the size of the function under consideration and therefore cannot be exhaustively searched in reasonable time. Even heuristics, which yield a quite good approximation of the design space show their incapability for an application to real examples, because of a not negligible run time, in the magnitude of several hours, assuming a typical project with about several hundred functions.

This work has been funded by the Christian Doppler Laboratory for Design Methodology of Signal Processing Algorithms.

One of the main applications of the design space exploration manifests in the extended partitioning problem [7]. Here hw/sw partitioning is performed on systems, that are represented as task graphs and each node has several implementation options differing in area and execution time. It is of paramount importance in an industrial setting to guarantee feasible running times for the estimation of those implementation options. This paper describes the generation of Pareto optimal solutions for area and timing trade-offs with evolutionary multi-objective optimisation. One of the main targets is to identify a parameter setting combined with several improvements with respect to the fitness function and to the introduction of a problem specific elitism technique, which achieves small running times, while preserving the performance.

The rest of the paper is organised as follows. Section II gives an overview of related work in the field of design space exploration. Further, in Section III the generation of a design space for area and timing trade-offs is described. Its formulation as multi-objective optimisation problem, which leads to the definition of the Pareto optimality, is presented in Section IV. The mapping of the optimisation problem to a genetic algorithm (GA) is described in Section V. Within this section several problem specific improvements of the standard GA implementation are introduced. Further, performance indicators are defined and a performance analysis of the generated Pareto front is demonstrated on several examples in Section VI. Finally some conclusions and perspectives to future work are given.

II. Related Work

The design space exploration embraces a multitude of different optimisation scenarios on varying abstraction levels. Typical parameters of this exploration are timing, power, and area.

The work of Ahmad et al. [8] identifies trade-offs between area and control steps for data flow graphs. For this task a problem specific genetic algorithm is introduced, which optimises the number of hardware resources, the number of control steps, and the length of the clock period. A shortcoming of this approach is the neglect of control and wiring overhead for the area estimation.

Design space exploration for multi processor architectures is presented by Zivkovic [9]. This work focuses on the comparison of fast estimations against accurate estimations generated by simulation traces.

System optimisation and exploration with respect to power consumption is presented within the work of Henkel [10]. In their work effects of certain system parameters like cache size or main memory size are considered.

Haubelt and Teich [11] estimate a Pareto front by applying a hierarchical approach. They use Pareto front arithmetics [12] for combining different Pareto fronts, that are computed for sub parts of the architecture. Further extensions of this approach utilise particle swarm optimisation [13].

The work of Bilavarn et al. [14] considers time and area trade-offs for different implementation types regarding loop transformations like loop unrolling or pipelining. Two main steps compose the flow. In a first step structural exploration defines several RTL implementations. As a second step characteristics are estimated for the physical mapping on FPGAs. Here, an exhaustive search of all possible design points is performed in order to find the optimal solutions.

So et al. [15], [16] examine trade-offs between area and timing by loop unrolling. A search strategy is introduced to identify an optimal design point for an FPGA implementation. This search strategy utilises a balance metric, which guides the algorithm. It tries to equally use memory and computation resources targeting a balanced utilisation of the available resources. The compiler within this technology offers several optimisations like loop unroll and jam, scalar replacement, peeling, and invariant code motions.

III. Trade-Off between Area and Timing

In general an algorithm within a function, which is written in form of sequential code, can be decomposed into a control flow graph (CFG). In a CFG representation the interconnected vertices are called basic blocks (BB). Each basic block contains a sequence of data operations ended by a control flow statement as its last instruction. This means that the operations inside a BB are completely data oriented and can be represented as data flow graph (DFG) as depicted in Fig. 2.

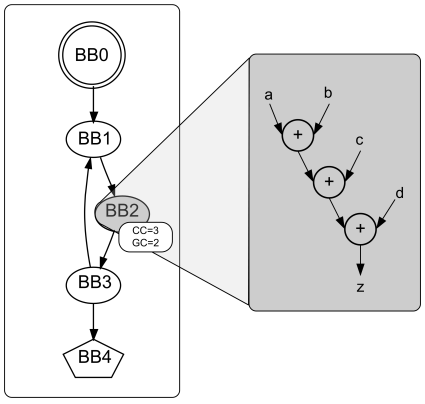


Fig. 2. CFG and a DFG for BB_2 with annotations for the corresponding cycle count (CC) and gate count (GC).

For each basic block BB an execution time $\mathrm{CC}(BB)$ and an area $\mathrm{GC}(BB)$ is assumed (e.g. BB_2 in Fig. 2).

The basic ideas of area and execution time estimation are mentioned in the following, a complete discussion of this topic would be beyond the scope of this paper. Execution time measured in cycles is based on the analysis of the depth of the DFG of one basic block [17]. Area measured in gates is estimated based on the number of required resources like ADD, SUB, MUL, or DIV and the number of control states. Further, in order to achieve a more reliable estimation the reuse capabilities within different CFG paths is accounted with a parallelism metric for each type of operation [18].

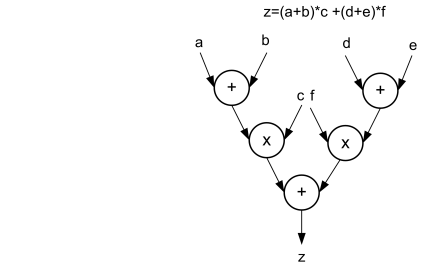
Different design points are obtained by source code transformations like loop unrolling or tree height reduction:

Loop unrolling describes the reduction of execution time of a loop by implementing several instances of the loop body, which can be executed in parallel [19]. We assume reducible loops with statically known iteration space. The unrolling factor U counts the number of duplicated loop bodies. We allow for a loop unrolling factor U for iteration counts M such that M/U is an integer. The original cycle count of the loop is defined by the cycle count of the longest path of the loop body multiplied with the number of loop iterations, $t = M\,\mathrm{CC}(LP)$. The gate count of the loop corresponds to the gate count of the loop body $a = \mathrm{GC}(BB_{\mathrm{body}})$. After loop unrolling with the factor U the new cycle count computes to $t_u = t/U$ and the new amount for the required area is computed with $a_u = aU$. For the simplification of the model we neglect an additional increase of the gate count caused by the added complexity of the controller implementation [4].

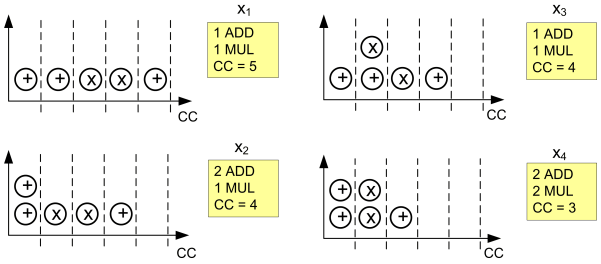
Tree height reduction deals with different scheduling of the data operations inside of a basic block. Assume for instance a complete sequential scheduling opposing to an implementation featuring full parallelism.

Example 3.1: In Fig. 3(a) a DFG is shown, which corresponds to the algebraic expression $z = (a+b)c+(d+e)f$. Different schedules allow for four realisations (x_1, x_2, x_3, and x_4) as shown in Fig. 3(b). Here, a cycle count of five is achieved by sequentially performing the additions ($\mathbf{x}_1$), whereas a completely parallel implementation achieves a cycle count of three ($\mathbf{x}_4$). Finally the design space of the four solutions is shown in Fig. 3(c)). It is assumed that the implementation cost for a multiplication and additions is equal. In this case only the design points $\mathbf{x_4}$ and $\mathbf{x}_3$ achieve the best trade-off between area and timing and therefore could be considered as most promising candidates for an implementation.

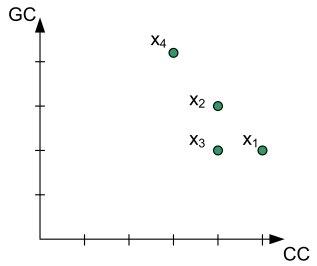
Based on the described code variants each BB is annotated with a set of k possible implementation types $I(BB) = \{(a_1, t_1), (a_2, t_2), \ldots, (a_k, t_k)\}$. Thus, the number of possible design points for a CFG, that consists of N basic blocks, is computed with $\prod_{i=1}^{N} |I(BB_i)|$. Hence, it is obvious that for large N an exhaustive search for all possible design points is an infeasible technique.



(a) Data flow graph.



(b) Four different implementations $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, and $\mathbf{x}_4$.



(c) Design space.

Fig. 3. Trade offs and design space for one data flow graph.

IV. Multi-objective Optimisation

As shown in the section before area and time are competing resources for the implementation. The problem of identifying optimal implementations can be generally described as multi-objective optimisation problem [20] and has the following formulation

$$\begin{aligned} &\min\{\mathbf{f}(\mathbf{x})\} \\ &\text{subject to}\, \mathbf{x} \,\epsilon\, S, \end{aligned} \tag{1}$$

where $\mathbf{f}(\mathbf{x})$ defines a vector involving $k(\geq 2)$ conflicting objective functions $f_i : \mathbb{R}^n \rightarrow \mathbb{R}, i = 1..k$. The decision (variable) vector $\mathbf{x} = (x_1, \ldots, x_n)^T$ belongs to the feasible region $X_f \subset \mathbb{R}^n$. This region is determined by a set of constraints b_i like maximum area, maximum response time, or maximum power consumption given by the requirements of the system. Those constraints, which can be grouped into a vector $\mathbf{b} = (b_1, \ldots, b_n)^T$, define a set of inequalities

$$\mathbf{x} \leq \mathbf{b}. \tag{2}$$

All possible realisations span the design space $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$. The following relations between two design points $\mathbf{x}_1$ and $\mathbf{x}_2$ can be identified:

$$\begin{gathered} \mathbf{x}_1 \succ \mathbf{x}_2 \text{ (dominates) if } \mathbf{f}(\mathbf{x}_1) < \mathbf{f}(\mathbf{x}_2), \\ \mathbf{x}_1 \succeq \mathbf{x}_2 \text{ (weakly dominates) if } \mathbf{f}(\mathbf{x}_1) \leq \mathbf{f}(\mathbf{x}_2), \\ \mathbf{x}_1 \sim \mathbf{x}_2 \text{ (is indifferent to) if } \mathbf{f}(\mathbf{x}_1) \not\leq \mathbf{f}(\mathbf{x}_2) \wedge \mathbf{f}(\mathbf{x}_1) \not\geq \mathbf{f}(\mathbf{x}_2). \end{gathered}$$

We use the relation for vectors in the following way.

$$\begin{gathered} \mathbf{u} = \mathbf{v} \text{ if for all } i = 1, \ldots, k : u_i = v_i, \\ \mathbf{u} \leq \mathbf{v} \text{ if for all } i = 1, \ldots, k : u_i \leq v_i, \\ \text{and } \mathbf{u} < \mathbf{v} \text{ if for all } i = 1, \ldots, k : u_i < v_i. \end{gathered}$$

The relations greater and greater equal are defined symmetrically. As stated before it is not possible to find a single solution that would optimise all the objectives simultaneously.

Therefore, we use the most commonly description of optimality for a multi-objective optimisation given by Vilfredo Pareto [21].

Definition 1 (Pareto-optimality): A vector $\mathbf{x}_1$ is Pareto optimal if there does not exist another vector $\mathbf{x}_2$ such that $\mathbf{x}_2 \succ \mathbf{x}_1$. The set of Pareto optimal points is called Pareto optimal set X_p or short Pareto front. Further, we call a quality set X_q the approximation of the Pareto set X_p.

Mathematically, all the Pareto optimal points are equally acceptable solutions. However, is is generally desirable to obtain one or a sub set of those points as solution.

Example 4.1: In Fig. 4 a design space for area and timing is depicted. The design point $\mathbf{x}_1$ is Pareto optimal and it dominates $\mathbf{x}_2$, $\mathbf{x}_3$, $\mathbf{x}_4$ and $\mathbf{x}_5$, ($\mathbf{x}_1 \succ \mathbf{x}_2$, $\mathbf{x}_1 \succ \mathbf{x}_3$, $\mathbf{x}_1 \succ \mathbf{x}_4$, $\mathbf{x}_1 \succ \mathbf{x}_5$), whereas $\mathbf{x}_3$ is indifferent to $\mathbf{x}_4$ ($\mathbf{x}_3 \sim \mathbf{x}_4$). The design point $\mathbf{x}_2$ weakly dominates $\mathbf{x}_3$($\mathbf{x}_2 \succeq \mathbf{x}_3$). Also some constraints for maximum timing and area are shown, so that the design points $\mathbf{x}_5$ and $\mathbf{x}_6$ become not valid.

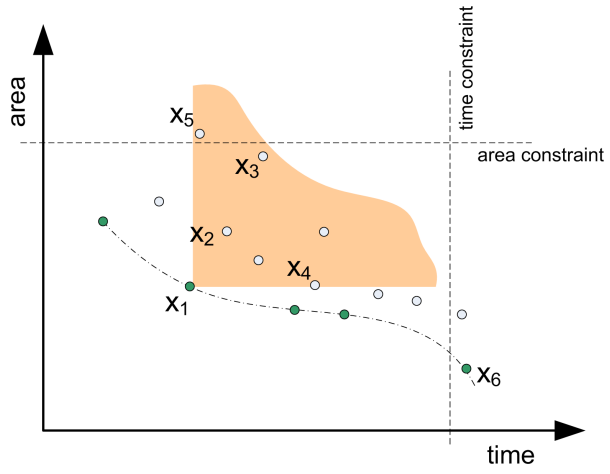


Fig. 4. Design space for area and timing trade-off.

V. Genetic Algorithm

Several heuristics are applicable in order to solve this optimisation problem like for example simulated annealing and tabu search. Usually, in this techniques the optimisation result is only one design point. In order to identify a Pareto front with theses heuristics a multi start approach has to be chosen, which leads to a high running time of the algorithms. Most promising candidates for the identification of a Pareto front are genetic algorithms because of their good capability of maintaining a large set of possible solutions [22]. The fundament of the algorithm is the representation of a solution in form of a chromosome. In this case it is a vector of the length of the number of basic blocks of the CFG (Fig. 5).

Each vector entry selects one implementation type of its corresponding basic block. The structogram of a genetic algorithm is depicted in Fig. 6. Initially a set of individuals is generated with randomly chosen implementation types, which forms the starting population. Now the parents for the next generation are found by binary selection. This means that two chromosomes

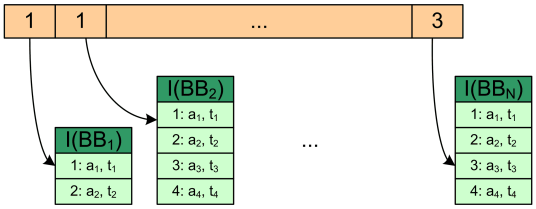


Fig. 5. Chromosome representation of a design point for the genetic algorithm.

are randomly chosen and the one, which has a better fitness is selected as parent. After that the parents are recombined by joining sub sequences of two different chromosomes, in order to get the new population. Further, the implementation types of certain chromosomes are probabilistically changed in order to guarantee the diversity of a population (mutation). This procedure is repeated until a saturation criteria is achieved, e.g. new Pareto optimal points cannot be found within one iteration of the algorithm.

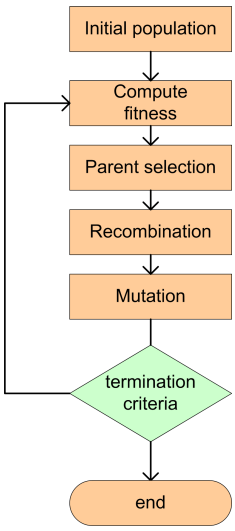


Fig. 6. Structogram of a genetic algorithm.

A. Fitness Function and Elitism

The performance of the optimisation heavily depends on the fitness computation. Also the application of an elitism technique can significantly improve the performance of the genetic algorithm.

1) Weighted Sum: The fitness of each design point is computed by a weighted sum

$$\sum_{i=1}^{k} w_i f_i(\mathbf{x}) \tag{3}$$

with the weighting coefficients w_i that represent the relative importance of the objective functions. Usually it is assumed that the weights sum up to 1, $\sum_{i=1}^{k} w_i = 1$. With this method a scalar fitness function is achieved, which allows for simple

comparison of two design points. A serious drawback lies in its incapability of generating members of the Pareto optimal set if the Pareto front is concave [23]. Therefore, an approach is proposed to solve the same problem for different values of the weighting coefficients. This approach does not require any changes to the basic mechanism of the genetic algorithm and is therefore easy to implement. We will refer to this type of implementation feature as *weighted sum* (WS).

2) Rank Ordering of Pareto Fronts: A strategy featuring higher complexity determines the fitness criteria of each design point by a consecutive identification of Pareto fronts within one population: once a population is generated, the Pareto optimal set of this population is determined and removed from the set. Afterwards the next Pareto front of the remaining set is determined. This procedure is repeated until the set is empty. Now, each point gets a fitness value according to rank number of its Pareto front. In Fig. 7 the design points $\mathbf{x}_1, \ldots, \mathbf{x}_{14}$ are shown and their split into consecutive Pareto fronts. $X_1 = \{\mathbf{x}_1, \ldots, \mathbf{x}_5\}$, $X_2 = \{\mathbf{x}_6, \ldots, \mathbf{x}_{11}\}$, and $X_3 = \{\mathbf{x}_{12}, \ldots, \mathbf{x}_{13}\}$.

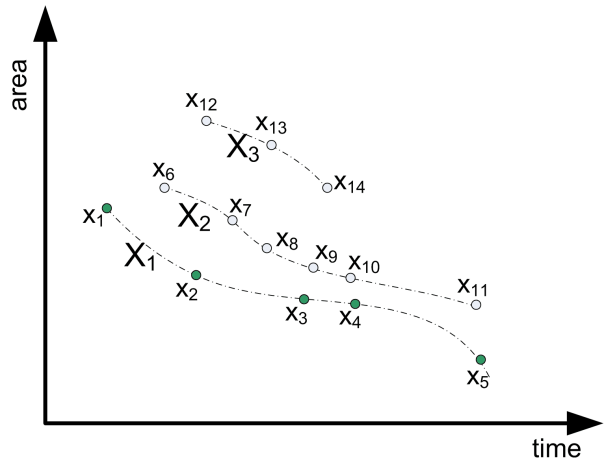


Fig. 7. Rank ordering of a population.

In general the computation of one Pareto front has a complexity of $\mathcal{O}(MN^2)$. Here M denotes the number of objectives and N is the size of the population. In the worst case this sorting has to be performed N times resulting in a complexity of $\mathcal{O}(MN^3)$ in order to determine all fronts of one set. A fast sorting approach in the order of $\mathcal{O}(MN^2)$ is shown in [24]. A GA with this fitness computation will be referred to in this work as *rank ordered* (RO).

3) Elitism with Extreme Values: Generally the identification of all Pareto optimal design points requires an exhaustive search. Nevertheless, this specific problem allows to compute some Pareto optimal points beforehand. Generally, we define extrema as design points $\mathbf{x}_{e,i}$, that minimise or maximise at least one of the objective functions f_i,

$$\begin{aligned} \mathbf{x}_{e,i} &= \underset{\mathbf{x}\in\mathbf{X}}{\operatorname{argmin}} f_i(\mathbf{x}),\ i = 1, \ldots, k, or \\ \mathbf{x}_{e,i} &= \underset{\mathbf{x}\in\mathbf{X}}{\operatorname{argmax}} f_i(\mathbf{x}),\ i = 1, \ldots, k. \end{aligned} \tag{4}$$

The extreme values for the execution time are found by.

$$T_{\max} = \max_{p\in\mathbf{B}} \{T(p)\}. \tag{5}$$

Here B defines the set of all possible paths through the CFG and for each BB the implementation type is chosen, which yields the highest cycle count. Vice versa, if for each path the implementation type with the minimal execution time is chosen, $T_{\min}$ is computed with

$$T_{\min} = \max_{p\in\mathbf{B}} \{T(p)\}. \tag{6}$$

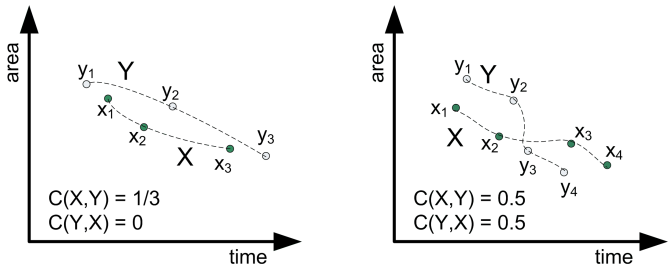
We will use this extreme points in order to implement an elitism feature within the genetic algorithm. This means that usually the overall best values are added to the population in order to preserve them. In our case we will add always the extreme design points to the population. In the remaining sections we will refer to a genetic algorithm, which includes the described feature as *extreme value elitism* (EVE).

VI. Performance Analysis

As the result of a multi-objective optimisation is not one single value, which could be easily compared to the result of another optimisation run, it is necessary to define some quality measures. In order to quantify the quality of the genetic algorithm, the actually achieved Pareto front can be compared to the Pareto optimal solutions found by the full search method. This is certainly only possible for small optimisation problems. The size of such problems is in the order of 10^{10} design points and a full search takes some hours of computation time on a standard PC (3 GHz). For larger problem sizes we will use the following two quality measures. The *coverage* reflects the dominance behaviour of two pareto fronts and the *hyper volume indicator* is a measure for the length of one pareto front.

Coverage: The coverage C is defined for two sets of design points and describes the number of points that are dominated, $\mathrm{C}(X,Y) = \frac{|\{y\,\epsilon Y \,|\, \exists\, x\epsilon X : x \succeq y\}|}{|Y|}$. A coverage of $\mathrm{C}(X,Y) = 1$ means that all elements of Y are weakly dominated by at least one element of the set X, whereas $\mathrm{C}(X,Y) = 0$ indicates that no element of the set Y is weakly dominated by any element of the set X. Note, that this function is usually non symmetric, $\mathrm{C}(X,Y) \neq \mathrm{C}(Y,X)$, and also the the sum of $\mathrm{C}(X,Y)$ and $\mathrm{C}(Y,X)$ does not equal 1, $\mathrm{C}(X,Y)+\mathrm{C}(Y,X) \neq 1$. For example in Fig. 8(a) the Pareto set X dominates $1/3$ of the set Y. Vice versa no design point of X is dominated by Y. This means that if we want to compare the two sets of the Figure we would consider X better, nevertheless the set Y covers the design space to a higher extend (higher distance between the extrema). In Fig. 8(b) a symmetric scenario is depicted.

Hyper Volume Indicator: The hyper volume indicator [25] I_H is a unary quality measure that can be applied to a single quality set. The hyper volume is spanned by the union of of cubes between the solutions x_i and a reference point $\mathbf{z}$. Certainly, it is important that the reference point $\mathbf{z}$ has to be chosen such, that all of the design points dominate this



(a) Non symmetric coverage. (b) Symmetric coverage.

Fig. 8. Different coverage scenarios.

point ($x \succcurlyeq z$) in order to have an appropriate measure of the covered area. In our problem this requirement is achievable, because the maximum extension of the Pareto front is known. Therefore, we have to chose $\mathbf{z}$ such, that it is dominated by the extreme values $\mathbf{x}_{e1}$ and $\mathbf{x}_{e2}$, which are known beforehand, $\mathbf{x}_{e1} \succ \mathbf{z}$ and $\mathbf{x}_{e2} \succ \mathbf{z}$ (Fig. 9).

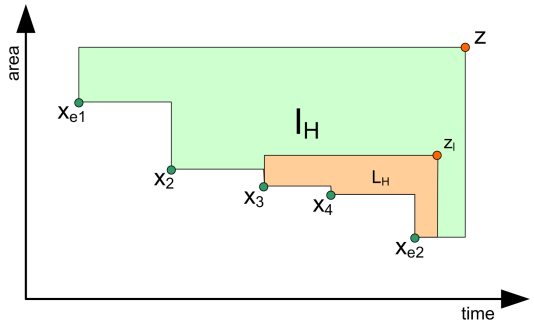


Fig. 9. Hyper volume indicator I_H and local hyper volume indicator L_H for a Pareto front.

A measure for the local performance of a Pareto front is the local hyper volume indicator L_H. For example, we apply this local measure to identify the performance of the Pareto set in the vicinity of the extreme values.

In order to identify the effects of three strategies described in the previous section, we apply the the most promising combinations of features that are used for the optimiser $I \in \{\mathrm{GA}_1, \mathrm{GA}_2, \mathrm{GA}_3, \mathrm{GA}_4\}$ as shown in Table I separately. Note, that the *extreme value elitism* cannot be applied as single feature and is therefore used only in combination with other features. The genetic algorithm GA_1 utilises both features rank ordering and weighted sum. Here, the selection process inside the genetic algorithm contains two steps. At first the rank of two chromosomes is compared and the one with the minor rank is selected. If the rank of the two competing chromosomes is equal, than the one with the smaller weighted sum is selected as parent for the next generation.

Due to the randomised characteristics of GAs for each algorithm type 30 runs have been performed. The achieved quality set of the genetic algorithm GA_i will be described with $X_q^i(\mathrm{GA}_i)$. Hence, 30 approximation sets $X_{q1}^i, X_{q2}^i, \ldots, X_{q30}^i$ for every algorithm GA_i have been generated.

	WS	RO	EVE
GA_1	x	x	x
GA_2		x	x
GA_3		x	
GA_4	x		

TABLE I

FEATURES OF THE VARIOUS GENETIC ALGORITHMS $\mathrm{GA}_1, \ldots, \mathrm{GA}_4$.

The performance of the different optimisers I is evaluated on two control flow graphs (CFG_{13} and CFG_{23}). Here the CFG_{13} consists of 13 basic blocks and features a size which allows for an exhaustive search of the Pareto front. The control flow graph CFG_{23} incorporates 23 basic blocks. The parameters of the genetic algorithms are set to 20 individuals per population and a mutation probability of 10%.

In Table II the results regarding the found quality set X_q compared to the optimal front X_p is described. The actual front, which has been derived by an exhaustive search, consists of 133 design points. Only GA_4 is able to find a substantial portion (22%) of these design points of the optimal Pareto front. Nevertheless, the draw back of a pure weighted sum approach will be apparent in the next examples when the design space coverage is considered. A first indication of this behaviour can be seen if we compare the number of design points that build the quality set. Here the algorithm GA_1 contains about 50% more design points compared to GA_4. Certainly, the performance of all the different genetic algorithms can be substantially improved by increasing the number of individuals. With a population of about 100 individuals nearly 80% of the Pareto optimal design points can be found. The usage of 20 individuals aims at low computation times of about several minutes in order to be appropriate for an application in realistic setting.

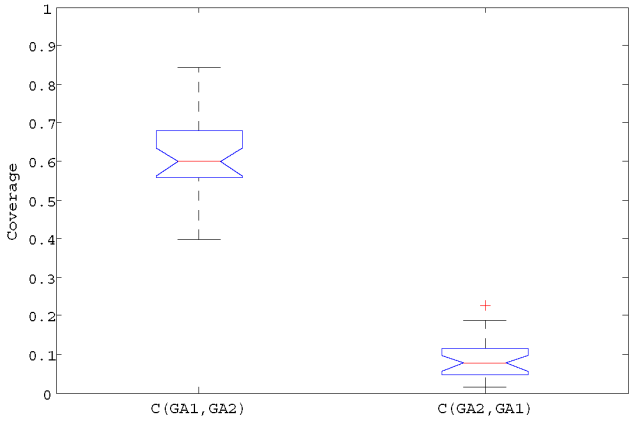
	$\lvert X_q\rvert$	$\lvert X_q \cap X_p\rvert$	$(\lvert X_q \cap X_p\rvert)/\lvert X_p\rvert\%$
Full Search	133	133	100
GA_1	67	5	3
GA_2	65	3	2
GA_3	50	0	0
GA_4	44	29	22

TABLE II

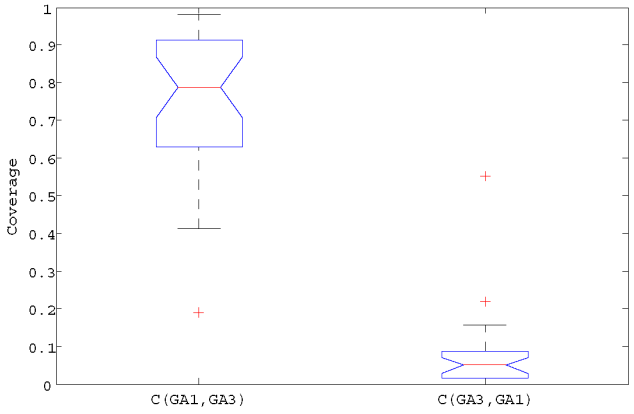
PERFORMANCE COMPARISON OF THE DIFFERENT OPTIMISERS FOR THE CONTROL FLOW GRAPH CFG_{13}.

In the following we consider a control flow graph CFG_{23} with a more realistic size. Since the statistical distribution of the results of the optimisation runs is not known, the results of the optimisations are illustrated with box plots, that show the median, the first and the third quartile, 1.5 of the interquartile range, and outliers. The significance of differences regarding the performance indicators of the algorithms has been tested with the rank based Wilcoxon-Mann-Whitney [26] test.

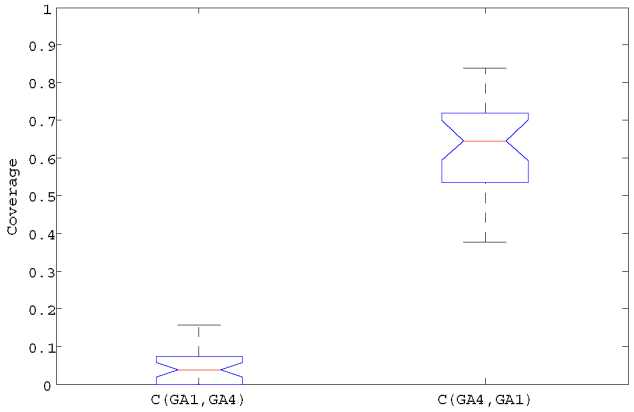
Fig. 10 shows the coverage of the algorithm GA_1 compared to GA_2, GA_3, and GA_4. Here GA_1 significantly outperforms GA_2 and GA_3 (Fig. 10(a) and Fig. 10(b)). The Pareto front of the algorithm GA_1 dominates 60% to 80% of the quality set



(a) Coverage C(X_q(GA$_1$),X_q(GA$_2$)) and C(X_q(GA$_2$),X_q(GA$_1$)).



(b) Coverage C(X_q(GA$_1$),X_q(GA$_3$)) and C(X_q(GA$_3$),X_q(GA$_1$)).



(c) Coverage C(X_q(GA$_1$),X_q(GA$_4$)) and C(X_q(GA$_4$),X_q(GA$_1$)).

Fig. 10. Box plot of the coverage of X_q(GA$_1$) compared to X_q(GA$_2$), X_q(GA$_3$), and X_q(GA$_4$) for the control flow graph CFG$_{23}$.

of GA$_2$ and GA$_3$, and only about 10% of the design points of GA$_1$ are dominated. Fig. 10(c) demonstrates that the weighted sum approach dominates about 60% of the design points of the combined rank order and weighted sum approach of the algorithm GA$_1$. Vice versa only 5% of the design points are dominated by GA$_1$.

In Fig. 11 the quality sets of the GA$_1$ and GA$_4$ are presented. Here the better coverage performance of GA$_4$ becomes visible caused by a grouping of the resulting design points in the middle of the design space region. Naturally, the elitism of extreme values generates also solution in the regions of the extreme values (higher local hyper volume indicator in the vicinity of the extrema). Therefore, much more solution in this region are seen, which causes a better convergence in this region of the Pareto front.

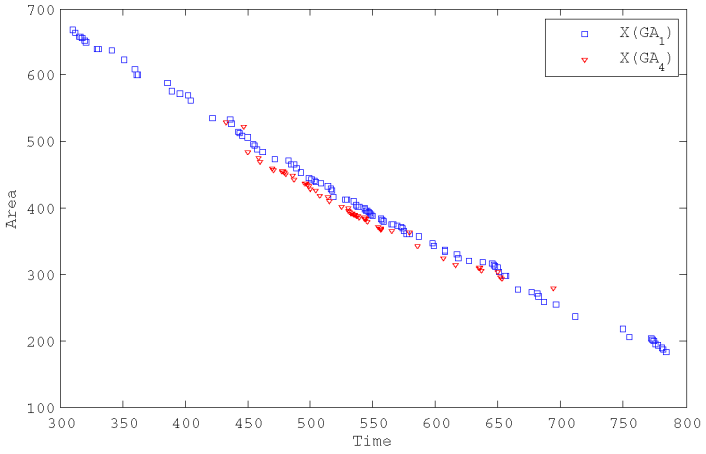


Fig. 11. Quality sets derived by the genetic algorithms GA$_1$ and GA$_4$.

Nevertheless, a quite high coverage of the possible design space is preferable. This is shown by the hyper volume indicator for the different optimisers (Fig. 12). The application of the extreme value elitism causes a significantly better performance of GA$_1$ and GA$_2$ compared to GA$_3$ and GA$_4$.

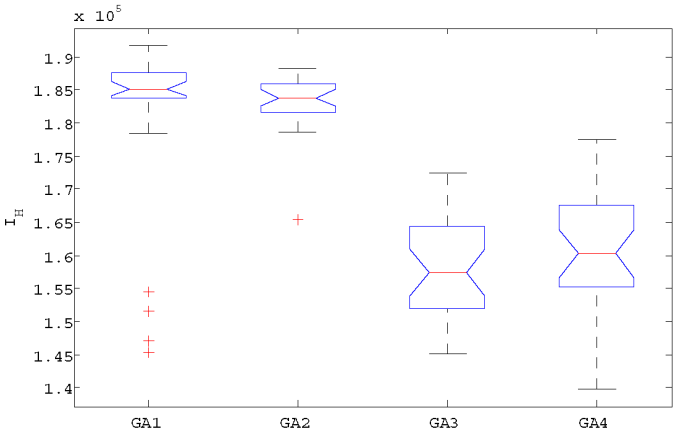


Fig. 12. Boxplot of the hyper volume indicator.

The convergency of the hyper volume indicator while the GA proceeds is presented in Fig. 13. Already more than 90% of the hyper volume indicator are achieved after 20 generations.

Finally it can be stated that the combination of improvements in GA$_1$ outperforms all the individually applied features in the other GAs and exposes good convergency behaviour. With increased population size and run time (generations)

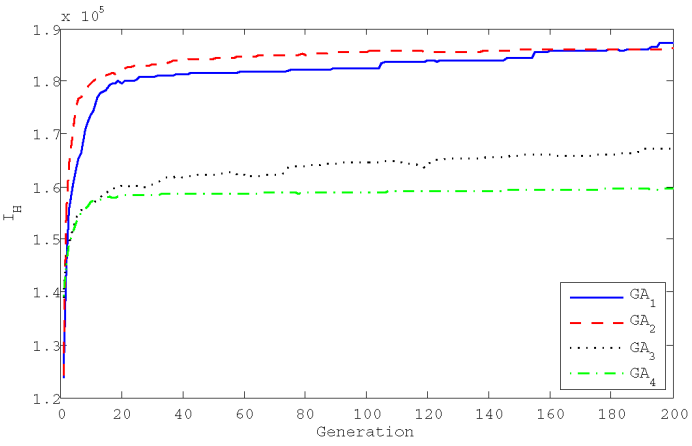


Fig. 13. Convergency of the Hyper volume indicator.

the effects of weighting and elitism vanish. Nevertheless, the design space exploration task has to be accomplished in a time that is still affordable in an industrial setting, therefore the number of individuals and generation should be chosen as small as possible.

VII. Conclusions

A new step in the automatisation of the hw/sw co-design process is introduced with high level synthesis tools. Here it is of high importance for a design flow improvement to provide the designer or other optimisation tools with Pareto optimal design points.

The design space of area and time trade-offs grows exponentially with the size of the underlying algorithm. The complete design space cannot be searched with a reasonable time effort.

This paper shows the application of a genetic algorithm to the problem of identifying Pareto optimal solutions of the time and area design space. The effectiveness of a two-stage fitness function and an extreme value elitism feature has been shown on an example. Future work will focuss on the selection of an subset of Pareto optimal design points. Also, the incorporation of power trade-offs is intended.

References

[1] J. Axelsson, "Cost Model for Electronic Architectures Trade Studies," in *Sixth International Conference on Engineering of Complex Computer Systems*, 2000.

[2] International SEMATECH, "International Technology Roadmap for Semiconductors," 2005. www.sematech.org.

[3] M. Raulet, F. Urban, J.-F. Nezan, C. Moy, O. Deforges, and Y. Sorel, "Rapid Prototyping for Heterogeneous Multicomponent Systems: An MPEG-4 Stream over a UMTS Communication Link," *Journal on Applied Signal Processing, Special Issue Design Methods for DSP Systems*, pp. 1–13, 2006.

[4] S. Gupta, N. Dutt, R. Gupta, and A. Nciolau, "Spark: A high-level synthesis framework for applying parallelizing compiler transformations," in *Intl. Conf. on VLSI Design*, pp. 461–466, 2003.

[5] A. L. Rosa, L. Lavagno, and C. Passorone, "Hardware/Software Design Space Exploration for a Reconfigurable Processor," in *Design, Automation and Test in Europe*, pp. 570–575, 2003.

[6] W. Böhm, J. Hammes, B. Draper, M. Chatawathe, C. Ross, R. Rinker, and W. Najjar, "Mapping a Single Assignment Programming Language to Reconfigurable Systems," *The Journal of Supercomputing*, vol. 21, pp. 117–130, February 2002.

[7] A. Kalavade and E. Lee, "The extended partitioning problem: Hardware -software mapping and implementation-bin selection," in *Proceedings of International Workshop on Rapid Systems Prototyping*, pp. 12–18, 1995.

[8] I. Ahmad, M. Dhodi, and F. Hielscher, "Design-Space Exploration for High-Level Synthesis," in *Computers and Communications*, pp. 491–496, 1994.

[9] V. D. Zivkovic, E. D. P. van der Wolf, and E. de Kock, "Design Space Exploration of Streaming Multiprocessor Architectures," in *IEEE Workshop on Signal Processing Systems*, pp. 228–234, 2002.

[10] J. Henkel and L. Yanbing, "Avalanche: An Environment for Design Space Exploration and Optimization of Low-Power Embedded Systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, pp. 454–468, 2002.

[11] C. Haubelt and J. Teich, "Accelerating Design Space Exploration," in *International Conference on ASIC*, pp. 79–84, 2003.

[12] C. A. Coello and M. S. Lechuga, "Mopso: A proposal for multiple objective particle swarm optimization," in *World Congress on Computational Intelligence*, pp. 1051–1056, 2003.

[13] S. Mostaghim and J. Teich, "Covering Pareto-optimal Fronts by Subswarms in Multi-objective Particle Swarm Optimization," in *Congress on Evolutionary Computation*, vol. 2, pp. 1404–1404, 2004.

[14] S. Bilavarn, G. Gogniat, J.-L. Philippe, and L. Bossuet, "Design Space Pruning Through Early Estimation of Area/Dealy Tradeoffs for FPGA Implementations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, pp. 1950–1968, October 2006.

[15] B. So, M. W. Hall, and P. C. Diniz, "A Compiler Approach to Fast Hardware Design Space Exploration in FPGA-based Systems," in *International Conference on Programming Language Design and Implementation (PLDI)*, pp. 165–176, June 2002.

[16] B. So, P. C. Diniz, and M. W. Hall, "Using Estimates from Behavioral Synthesis Tools in Compiler-Directed Design Space Exploration," in *Design Automation Conference*, pp. 514–519, 2003.

[17] M. Holzer and M. Rupp, "Static Estimation of the Execution Time for Hardware Accelerators in System-on-Chips," in *International Symposium on System-on-Chip*, November 2005.

[18] M. Holzer and M. Rupp, "Static Code Analysis of Functional Descriptions in SystemC," in *IEEE International Workshop on Electronic Design, Test and Applications*, pp. 243–248, January 2006.

[19] S. S. Muhnick, *Advanced Compiler Design and Implementation*. Morgan Kaufmann, 2004.

[20] Y. Collete and P. Siarry, *Multiobjective Optimization. Principles and Case Studies (Decision Engineering)*. Springer-Verlag Berlin Heidelberg New York, 2003.

[21] V. Pareto, *Cours D'Economie Politique*, vol. I and II. 1896.

[22] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[23] I. Das and J. Dennis, "A Closer Look at Drawbacks of Minimizing Weighted Sums of Objectives for Pareto Set Generation in Multicriteria Optimization Problems," *Structural Optimization*, vol. 14, no. 1, pp. 63–69, 1997.

[24] K. Deb, A. Pratap, S. Agrawal, and T. Meyarivan, "A fast and elitist multi-objective genetic algorithm: NSGA-II," tech. rep., Indian Institute of Technology Kanpur, Kanpur, India, 2000.

[25] E. Zitzler and L. Thiele, "Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 3, pp. 257–271, November 1999.

[26] H. B. Mann and D. Whitney, "On a test of whether one of 2 random variables is stochastically larger than the other," *Annals of Mathematical Statistics*, vol. 18, pp. 50–60, 1947.