Using Symbolic Feasibility Tests during Design Space Exploration of Heterogeneous Multi-Processor Systems*

Thomas Schlichter, Christian Haubelt, Frank Hannig, and Jürgen Teich
Department of Computer Science 12
University of Erlangen-Nuremberg, Germany
E-mail: {schlichter, haubelt, hannig, teich}@cs.fau.de

Abstract

The task of automatic design space exploration *of heterogeneous multi-processor systems is often tackled with Evolutionary Algorithms. In this paper, we propose a novel approach in combining Evolutionary Algorithms with* symbolic techniques *in order to improve the convergence speed. The main idea is to guide the search towards the* feasible region *by utilizing symbolic techniques. We will present experimental results showing the advantages of our novel approach, especially when the search space contains only few feasible solutions, what is often the case when designing heterogeneous multi-processor systems.*

1. Introduction

Modern embedded systems often consist of many communicating processor cores. The challenge in designing such heterogeneous multi-processor systems is to find optimal implementations with regard to multiple objectives while meeting several constraints. In order to allow an unbiased search, the task of *design space exploration* is performed before selecting (*decision making*) the actual implementation. Design space exploration is a very challenging constrained multi-objective optimization task [8, 6]. The basic problem is the selection of appropriate hardware resources and the assignment of processes to the selected resources. Obviously, there is an inherent tradeoff between the number of resources used and the performance of the systems. Many successful design space exploration methodologies based on Evolutionary Algorithms are reported in literature [8, 6, 5, 3]. Due to data dependencies among the processes, nearly all possible implementations are *infeasible*, as will be shown in this paper.

Our problem is somehow different from other *constraint optimization problems* known from literature (see e.g. [4, 1]), as we can only determine the corresponding *objective values* if the solution found is feasible. Thus, the task of design space exploration is twofold: (i) *guide* the search towards the *feasible region* and (ii) *optimize* the feasible solutions. In this paper, we will focus on the first task. We will present a sophisticated feasibility test using symbolic techniques in order to guide the search. The feasibility test uses Binary Decision Diagrams (BDD) and a technique known as *functional simulation by BDDs* [10]. We will provide experimental results showing the efficiency of our novel method.

The rest of the paper is organized as follows: Section 2 contains the definition of the search space, which is represented by a so called *specification graph*. This section also defines the task of design space exploration and the actual optimization problem. In Section 3, we describe how the optimization problem can be solved using Evolutionary Algorithms. Here, the test for feasible solutions as well as our proposed improvements will be discussed in detail. We will provide experimental results showing the advantages of our novel approach in Section 4. Finally, Section 5 will conclude the paper.

2. Problem Statement

In this paper, we consider the problem of design space exploration for embedded systems. Basically, the design space exploration problem is a multi-objective selection and assignment problem. Typical problems known from real-world applications show search spaces containing only a very small fraction of feasible solutions.

2.1. Defining the Search Space

To allow a mathematical model of the search space, the concept of a so-called *specification graph* is needed. A specification graph specifies a multi-processor system by means of its applications, the architecture, and the relation

*This work was supported in part by the Fraunhofer IIS, Germany

between these two views. Here, we use a graph-based approach already proposed by Blickle et al. [3].

Definition 1 (Specification Graph) *A* specification graph *is a directed graph* $g_s(V_s, E_s)$ *that consists of a process graph* $g_p(V_p, E_p)$*, an architecture graph* $g_a(V_a, E_a)$*, and a set of* mapping edges E_m*. In particular,* $V_s = V_p \cup V_a$*,* $E_s = E_p \cup E_a \cup E_m$*, where* $E_m \subseteq V_p \times V_a$*.*

Consequently, mapping edges relate the vertices of the process graph to vertices of the architecture graph. The edges represent user-defined mapping constraints in the form of a relation: "can be implemented by".

The goal of design space exploration is to find optimal solutions which satisfy the specification given by the specification graph. Such a solution is called a *feasible implementation* of the heterogeneous multi-processor systems. Due to the multi-objective nature of this optimization problem, there is in general more than a single optimal solution.

2.2. System Synthesis

The task of system synthesis is to determine an implementation, i.e., assigning activity values to vertices and edges of the specification graph. An implementation, consists of three parts: (1) the *allocation* that indicates which elements of the architecture graph are used in the implementation, (2) the *binding*, i.e., the set of mapping edges which define the binding of processes to components of the architecture graph, and (3) the *schedule* assigning a start time to each operation in the process graph.

Before defining the term *implementation* formally, Blickle et al. [3] introduce the so-called *activation* of vertices and edges:

Definition 2 (Activation) *The* activation *of a specification graph* $g_s(V_s, E_s)$ *is a function* $a : V_s \times E_s \mapsto \{0,1\}$ *that assigns to each edge* $e \in E_s$ *and to each vertex* $v \in V_s$ *the value* 1 *(activated) or* 0 *(not activated).*

For the sake of simplicity, it is assumed that all vertices $v \in V_p$ and all edges $e \in E_p$ of the process graph g_p are activated subsequently. So only the vertices $v \in V_a$ of the architecture graph and the edges $e \in E_a \cup E_m$ can be either activated or deactivated.

Definition 3 (Allocation) *An* allocation α *of a given specification graph* g_s *is the subset of all activated vertices and edges of the architecture graph* g_a*, i.e.,*

$$\begin{aligned} \alpha &= \alpha_v \cup \alpha_e, \text{where} \\ \alpha_v &= \{v \in V_a \mid a(v) = 1\} \\ \alpha_e &= \{e \in E_a \mid a(e) = 1\} \end{aligned}$$

Here, α_v *denotes the set of allocated vertices and* α_e *denotes the set of allocated edges.*

Definition 4 (Binding) *A* binding β *of a given specification graph* g_s *is the subset of activated mapping edges* E_m*, i.e.,*

$$\beta = \{e \in E_m \mid a(e) = 1\}$$

In order to restrict the search space, it is useful to determine the set of *feasible allocations* and *feasible bindings*. A feasible binding guarantees that communications demanded by the process graph can be established in the allocated architecture. This property makes the resulting optimization problem hard to be solved.

Definition 5 (Feasible Binding) *Given a specification graph* g_s *and an allocation* α*, a* feasible binding *is a binding* β *that satisfies the following requirements:*

1. *Each activated mapping edge* $e \in \beta$ *ends at an activated vertex, i.e.,* $\forall e = (v_p, v_a) \in \beta : v_a \in \alpha$*.*
2. *For each process graph vertex* $v \in V_p$*, exactly one outgoing mapping edge* $e \in E_m$ *is activated, i.e.,*

$$\left|\{e \in \beta \mid e = (v_p, v_a), v_a \in V_a\}\right| = 1.$$

3. *For each process graph edge* $e \in (v_i, v_j) \in E_p$*:*
   - *either both operations are mapped onto the same vertex, i.e.,* $\widetilde{v}_i = \widetilde{v}_j$ *with* $(v_i, \widetilde{v}_i), (v_j, \widetilde{v}_j) \in \beta$*,*
   - *or there exists an activated edge* $\widetilde{e} = (\widetilde{v}_i, \widetilde{v}_j) \in E_a \cap \alpha$ *in the architecture graph to handle the communication associated with edge e, i.e.,*

$$(\widetilde{v}_i, \widetilde{v}_j) \in E_a \cap \alpha \text{ with } (v_i, \widetilde{v}_i), (v_j, \widetilde{v}_j) \in \beta.$$

The last property is especially important in multi-processor systems, where communication demanded by the desired application must be handled by the allocated architecture.

Definition 6 (Feasible Allocation) *A* feasible allocation *is an allocation* α *that allows at least one feasible binding* β*.*

Finally, a *schedule* τ can be computed. With the above discussion, we can define an *implementation* by means of a feasible allocation, a feasible binding, and a schedule.

Definition 7 (Implementation) *Given a specification graph* g_s*, a* (feasible) implementation ψ *is a triple* (α, β, τ) *where* α *is a feasible allocation,* β *is a corresponding feasible binding, and* τ *is a schedule.*

2.3. The Optimization Problem

Now, the task of system synthesis can be formulated as a combinatorial *Multi-objective Optimization Problem*.

Definition 8 (System Synthesis) *The task of* system synthesis *is the following multi-objective optimization problem (MOP) where without loss of generality, only minimization problems are assumed here:*

minimize $f(x)$,
subject to:
x represents a feasible implementation ψ,
$c_i(x) \leq 0,\ \forall i \in \{1,\ldots,q\}$

where $f(x) = (f_1(x), f_2(x), \ldots, f_n(x)) \in Y$ *is the* objective function, *Y is the* objective space, *$x = (x_1, x_2, \ldots, x_m) \in X$ is the* decision vector *and X is the* decision space.

Here, x is an encoding called *decision vector* representing an implementation ψ. Moreover, there are q constraints $c_i(x)$, $i = 1,\ldots,q$ imposed on x defining the set of feasible implementations. The *objective function* f is n-dimensional, i.e., n objectives are optimized simultaneously. For example, in embedded system design it is required that the monetary cost and the power dissipation of an implementation are minimized simultaneously.

Only those *design points* $x \in X$ that represent a feasible implementation ψ and that satisfy all constraints c_i, are in the set of feasible solutions, or for short in the *feasible set* called $X_f = \{x \mid \psi(x)$ being feasible $\wedge\, c(x) \leq 0\} \subseteq X$. The objective space of X is defined as $Y = f(X) \subset \mathbb{R}^n$, where the objective function f on the set X is given by $f(X) = \{f(x) \mid x \in X\}$. Analogously, the *feasible region* in the objective space is denoted by $Y_f = f(X_f) = \{f(x) \mid x \in X_f\}$.

In single-objective optimization, the feasible set X_f is totally ordered, whereas in multi-objective optimization problems, the feasible set X_f is only partially ordered and, thus, there is generally not only one global optimum, but a set of so-called *Pareto points*. A Pareto-optimal implementation is a design point which is not worse than any other feasible solution in the design space in all objectives (cf. [9]).

3. Design Space Exploration

In this section, we will show how to solve the system synthesis problem by using Evolutionary Algorithms (EAs). The EA determines the allocation and bindings. Afterwards, the schedule of an implementation is computed by a list scheduler. The main idea is outlined in Figure 1.

3.1. The Evolutionary Algorithm

To obtain a meaningful encoding for the task of system synthesis, one has to address the question of how to handle infeasible allocations and infeasible bindings suggested by the EA. Obviously, if allocations and bindings may be randomly chosen, a lot of them can be infeasible. In general, there are two different methods to handle these infeasible

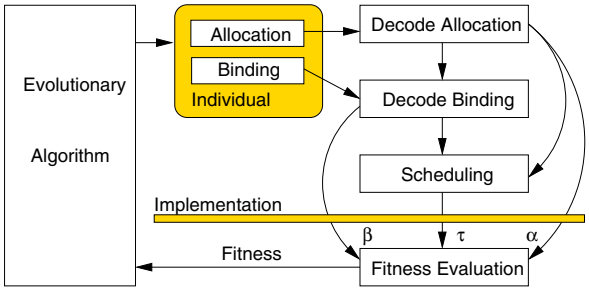

Figure 1. The decoding of an individual to an implementation [3].

implementations: Punishing and Repairing [3]. Here, repairing is used. Because of the well known properties of feasible allocations and bindings, one can "repair" infeasible individuals by incorporating domain knowledge in these repair mechanisms easily. But as the determination of a feasible allocation or binding is $\mathcal{NP}$-complete [3], this would result in solving an $\mathcal{NP}$-complete task for each individual to be repaired.

These considerations have led to the following compromise: The randomly generated allocations of the EA are partially repaired using a heuristic. Possible complications detected later on during the calculation of the binding will be considered by a penalty. Hence, the mapping task can be divided into three steps: (1) the allocation of resources $v \in V_a$ is decoded from the individual and repaired with a simple heuristic, (2) next the binding of the edges $e \in E_m$ is performed, and (3) finally, the allocation is updated in order to eliminate unnecessary vertices $v \in V_a$ from the allocation and all necessary edges $e \in E_a$ are added to the allocation.

Thus, this algorithm results in a feasible allocation and binding of the vertices and edges of the process graph g_p to the vertices and edges of the architecture graph g_a. If no feasible binding could be found, the whole decoding of the individual is aborted.

The allocation of vertices is directly encoded in the so-called *chromosome*, i.e., the elements in a vector *alloc* encode for each vertex $v \in V_a$ if it is activated or not, i.e., $a(v) = alloc(v)$. This simple encoding might result in many infeasible allocations. Hence, a simple repair heuristic is applied. This heuristic only adds new vertices $v \in V_a$ to the allocation and reflects the simplest case of in-feasibility that may arise from non-executable functional vertices: Consider the set $V_B \subseteq V_p$ that contains all vertices that can not be executed, because not a single corresponding resource vertex is allocated, i.e., $V_B = \{v \in V_p \mid \forall (v, \widetilde{v}) \in E_m : a(\widetilde{v}) = 0\}$. To make the allocation feasible (in this sense), we add for each $v \in V_B$, at most one $\widetilde{v} \in V_a$, until feasibility in the sense above is achieved.

In this paper, the binding is of special interest and will be discussed in more detail in the next section. Calculating

a feasible binding for a given allocation of multiple processors is equivalent to solve the underlying satisfiability problem. To solve this problem, we can use fast heuristics that try to find a feasible binding. If no feasible binding is found, then β is the empty set, and the individual will be given a penalty to its fitness value.

3.2. Feasibility Test

As stated above, finding a feasible binding for a given allocation is a complex task. A binding is obtained by activating exactly one incident edge $e \in E_\mathrm{m}$ for each vertex $v \in V_\mathrm{p}$. To encode the binding *independent* from the actual allocation we use the following way:

All processes are bound in the order they appear in the binding order list L_O. For each process $p \in V_\mathrm{p}$, a list L_B is encoded as allele that contains all incident edges $e \in E_\mathrm{m}$ to p. This list is seen as a priority list and the first edge e_k with $e_k = (v, \widetilde{v})$ that gives a feasible binding is included in the binding, i.e., $\mathrm{a}(e_k) := 1$. The test of feasibility is directly related to the definition of a feasible binding (see Definition 5). As the priority lists contain all incident edges $e \in E_\mathrm{m}$, each individual contains a feasible binding iff it contains a feasible allocation.

The feasibility test can be performed in different ways. The first possibility is to solve the problem if there still exists a valid completion for all the unbound processes if the currently tested mapping edge is chosen. Of course, this will always find the correct binding for each process, and each feasible allocation results in a feasible binding. This problem can be expressed as a boolean satisfiability problem which is satisfiable iff there exists such a completion [7].

The boolean equation for this satisfiability problem can be generated by interpreting the three requirements from Definition 5. The first requirement states that each activated mapping edge ends at an activated resource. The corresponding boolean equation is:

$$\bigwedge_{p \in V_\mathrm{p}} \left[\bigvee_{m=(p,r) \in E_\mathrm{m}} \mathrm{a}(m) \wedge \mathrm{a}(r) \right] \tag{1}$$

The second requirement states that exactly one mapping edge is activated for each process. The boolean equation from above already ensures that at least one mapping edge for each process is activated. The following equation additionally ensures that at most one mapping edge is activated for each process.

$$\bigwedge_{m_i=(p,r_i), m_j=(p,r_j) \in E_\mathrm{m}: r_i \neq r_j} \overline{\mathrm{a}(m_i)} \vee \overline{\mathrm{a}(m_j)} \tag{2}$$

The third and last requirement states that communicating processes have to be mapped to the same or to an adjacent resource. This can be expressed by the following equation.

$$\bigwedge_{\substack{m_i=(p_i,r_i) \in E_\mathrm{m}: \\ (p_i,p_j) \in E_\mathrm{p}}} \left[\overline{\mathrm{a}(m_i)} \bigvee_{\substack{m_j=(p_j,r_j) \in E_\mathrm{m}: \\ r_i = r_j \vee (r_i,r_j) \in E_\mathrm{r}}} \mathrm{a}(m_j) \right] \tag{3}$$

As the Equations (1) - (3) have to be satisfied to allow a feasible binding, the complete boolean equation is the conjunction of these. The BDD for this equation can be built once and be used for each individual. When this BDD is built, the test if a mapping edge m still allows a feasible binding is just as simple as combining $\mathrm{a}(m)$ with the BDD with a logic AND-function. So the feasibility test can be done in $O(|E_\mathrm{m}| + |V_\mathrm{a}|)$ once the BDD is built [10]. Unfortunately, for real-world problems, the BDD is prohibitively large and thus this approach is not viable.

A second possibility is to check if the tested mapping edge is not feasible with the binding performed up to this test. Thus, the mapping edges are activated sequentially and the outcome of this *sequential feasibility test* strongly depends on both, the binding order list L_O and the binding priority list L_B. This test is easy and fast to perform. But as only the yet bound processes are considered, the feasibility test might be trapped, even if the allocation is feasible. This case is sketched in Figure 2(a). If process $p1$ is bound to the resource $r1$ before binding process $p2$, the feasibility test will be passed even though the only feasible binding is $\beta = \{m2, m3\}$.

To overcome this drawback, a relaxed version of the first feasibility test is proposed here. It is tested if the selection of a certain mapping edge for a process p prohibits a feasible binding of any direct predecessor or successor of p. It has already been mentioned that encoding the complete boolean equation in one BDD is not feasible for real-world problems. As compromise, we constructed a small BDD for each process. Equations (1) and (2) can be splitted into parts that contain only mapping edges incident to a single process. These boolean equations are encoded in the BDD

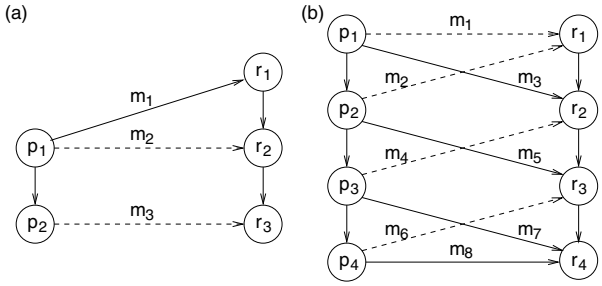

Figure 2. Examples that may trap (a) the sequential feasibility test and (b) even the BDD feasibility test.

of the corresponding process. The Equation (3) always contains mapping edges of different processes. Hence, the implementation given by Equation (3) must be considered in the BDDs associated with p_i and p_j.

With these BDDs, we can test if the activation of mapping edge m_i prohibits the feasibility of the adjacent processes. Therefore, we have to set $a(m_i) = 1$ for the BDD associated with process p_i belonging to m_i, and for all the BDDs associated with adjacent processes p_j. If one of the BDDs collapses to a logic 0 value, the mapping edge m_i would not allow to find a feasible binding and thus will be rejected.

This method solves the problem shown in Figure 2(a). Of course, this test generally is not able to find a feasible binding for each feasible allocation. This is illustrated in Figure 2(b) where only these bindings are feasible: $\{m1, m2, m4, m6\}$ and $\{m3, m5, m7, m8\}$. But as $p1$ and $p4$ do not have common neighbors, even the BDD feasibility test may select the mapping edges $m1$ and $m8$ what prohibits a feasible binding.

In this paper, we compare the sequential and BDD feasibility test. Our results show that the BDD feasibility test can improve the convergence even though the test itself is slower.

4. Experimental Results

In this section, we present experimental results from using the new BDD feasibility test. The three objectives used during the optimization are technical properties of embedded systems, namely *implementation cost*, *power dissipation*, and *latency*. In the following, we provide quantitative results from the comparison between the sequential feasibility test and the BDD feasibility test. The PISA (Platform independent Interface for Search Algorithms) [2] framework was chosen for optimization purposes. In the present work, the SPEA2 selection procedure [11] was applied.

The experiments are performed as follows: A generator program is used to construct MOP instances (specification graphs). Due to different random values, the generated problem instances are similar in structure, but not equal. Each MOP instance is optimized using the sequential feasibility test and the BDD feasibility test. After the optimization of each problem instance, the non-dominated solutions X_a found by both methods are combined in a single *reference set* X_R. This reference set is Pareto-filtered and is used to quantitatively assess the performance of both methods.

4.1. Problem Instances and Parameters

The MOP instances (specification graphs) are generated from following parameters: (i) The number of available resources in the architecture graph is 15. (ii) The number of processes in the process graph g_p is 45. (iii) Each process has $2 \ldots 4$ random mapping edges. (iv) The number of edges in the process graph is given by a probability value. This value determines the probability that two processes are connected by an edge and is 30%. (v) Four classes of problem instances are created with the feasibility probabilities 20%, 30%, 40% and 50%. This probability is used when the edges E_a of the architecture graph are created. For each possible mapping edge $m_i = (p_i, r_i), m_j = (p_j, r_j) \in E_m$ of two adjacent processes p_i and p_j, the resources r_i and r_j are connected with this probability to satisfy the data dependency. Smaller probability values result in less created edges, and less feasible solutions exist.

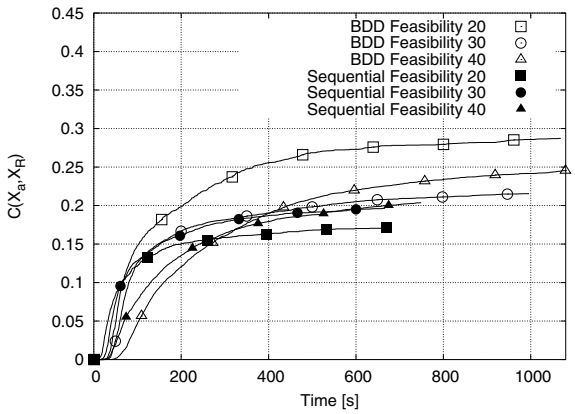

Figure 3. The mean coverage over the time.

For each of these problem classes we created 10 different problem instances. The genetic algorithm was run 10 times for each problem instance with both feasibility tests.

The parameter for the EA were chose as follows: The population size was set to 150. For recombination, 50 children were created from 50 parents by single-point crossover. The mutation rate was set to $|decision\ variables|^{-1}$. The mutation operation is either single bit flip or order-based mutation.

4.2. Qualitative Results

The used performance indicator is the *coverage* which measures the fraction of non-dominated points in the reference set X_R found by a particular optimization run (X_a). A detailed discussion on performance indicators can be found in [12]. The approximation sets X_a obtained from both optimization methods are compared to the reference set X_R by using this performance indicator. Moreover, the average time needed for a fix number of generations is calculated as well.

To show the speed difference, the mean coverage is drawn over the average computation time in Figure 3. As the sequential feasibility test is faster than the BDD feasibility test, its calculation of 1000 generations finishes earlier.

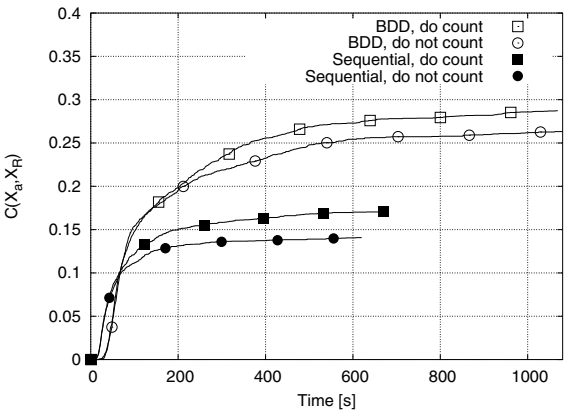

Figure 4. Comparison of the feasibility tests with and without error counting.

One can see that the BDD feasibility test is worse in the beginning, this is due to the time needed for constructing the BDDs. But, the harder the problem is, and if only few feasible solutions exists, the BDD feasibility test clearly outperforms the sequential feasibility test after a small amount of time.

We also want to show how counting the number of processes that cannot be bound influences the convergence. Therefore we chose the example with a feasibility of 20%. In Figure 4, the different coverage behaviors are drawn over the time. Here, one can see that the EA converges faster if not feasible bindings are differed by the count of processes that cannot be bound.

5. Conclusions

We have introduced a novel approach to improve the convergence of EA-based automatic design space exploration algorithms for heterogeneous multi-processor systems. This approach makes use of symbolic techniques to guide the search of the EA towards the feasible region. Here, feasibility regards to the problem of establishing communication demanded by an application on a multiprocessor architecture. We have presented experimental results which clearly show the benefits of our approach, especially for problems with a search space containing many infeasible solutions, as is often the case in heterogeneous multi-processor system design. Also, we briefly showed the coverage gain that can be reached by counting the number of processes that cannot be bound.

Although we had the focus on automatic design space exploration of heterogeneous multi-processor systems, there is however the potential to generalize our approach to other constrained combinatorial optimization problems.

References

[1] A. H. Aguirre, S. B. Rionda, C. A. C. Coello, G. L. Lizárraga, and E. M. Montes. Handling Constraints using Multiobjective Optimization Concepts. *International Journal for Numerical Methods in Engineering*, 59(15):1989–2017, Apr. 2004.

[2] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA - A Platform and Programming Language Independent Interface for Search Algorithms. In *Lecture Notes in Computer Science (LNCS)*, volume 2632, pages 494–508, Faro, Protugal, Apr. 2003.

[3] T. Blickle, J. Teich, and L. Thiele. System-Level Synthesis Using Evolutionary Algorithms. In R. Gupta, editor, *Design Automation for Embedded Systems*, 3, pages 23–62. Kluwer Academic Publishers, Boston, Jan. 1998.

[4] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Ltd., Chichester, New York, Weinheim, Brisbane, Singapore, Toronto, 2001.

[5] R. P. Dick and N. K. Jha. MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Co-Synthesis of Distributed Embedded Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(10):920–935, Oct. 1998.

[6] C. Haubelt, S. Mostaghim, F. Slomka, J. Teich, and A. Tyagi. Hierarchical Synthesis of Embedded Systems Using Evolutionary Algorithms. In R. Drechsler and N. Drechsler, editors, *Evolutionary Algorithms for Embedded System Design*, Genetic Algorithms and Evolutionary Computation (GENA), pages 63–104. Kluwer Academic Publishers, Boston, Dordrecht, London, 2003.

[7] C. Haubelt, J. Teich, R. Feldmann, and B. Monien. SAT-Based Techniques in System Design. In *Proceedings of Design, Automation and Test in Europe*, pages 1168–1169, Munich, Germany, Mar. 2003.

[8] V. Kianzad and S. S. Bhattacharyya. CHARMED: A Multi-Objective Co-Synthesis Framework for Multi-Mode Embedded Systems. In *Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'04)*, pages 28–40, Galveston, U.S.A., Sept. 2004.

[9] M. Laumanns. *Analysis and Applications of Evolutionary Multiobjective Optimization Algorithms*. PhD thesis, Eidgenössische Technische Hochschule Zürich, Aug. 2003.

[10] C. Scholl, R. Drechsler, and B. Becker. Functional Simulation Using Binary Decision Diagrams. In *Proceedings of the 1997 IEEE/ACM Int. Conference on Computer-Aided Design*, pages 8–12, San Jose, USA, Nov. 1997.

[11] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *Evolutionary Methods for Design, Optimisation, and Control*, pages 19–26, Barcelona, Spain, 2002.

[12] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117—132, Apr. 2003.