Design Space Exploration for Real-Time Reconfigurable Computing

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*— Run-time reconfigurable computing extends the classic role of FPGAs towards processing elements which feature multitasking similar to a micro processor. The main advantage of this usage is based on the granularity of the reprogrammable device which can be optimally adapted to each task that has to be executed. Hence, scheduling for such a usage scenario becomes apparent, where a set of tasks with their specific area demands and execution time has to be executed. Obviously, the provision of a set of design alternatives for each task will allow for a higher utilization of the FPGA. Nevertheless, it is computational impossible to facilitate all the theoretically achievable design alternatives of one task. Thus, this paper presents an scheduling algorithm that reduces the number of design alternatives that are utilized for a schedule. Furthermore, a scheduling algorithm is presented which achieves optimization in feasible execution time.**

I. Introduction

The traditional application of reconfigurable devices like FPGAs is mainly characterized by prototyping systems and as hardware accelerator. Here, one specific task is implemented once on such a dedicated device which can be described as compile time reconfiguration (CRC). Whereas run-time reconfigurable computing (RRC) is featured by SRAM-based FPGAs [1] like for example the Virtex II and the Virtex II Pro from XILINX with a configuration memory divided into frames which can be reconfigured independently. Thus, a partially reconfiguration is supported, that allows for executing one task, while other tasks are removed or reloaded onto the FPGA. Hence, scheduling for such a usage scenario becomes apparent, where a set of tasks with their specific area demands, execution time, and deadline has to be scheduled on the FPGA, similar to the behavior of a real time operating system.

The reconfiguration model of the reconfigurable device can be abstracted by a 1D or 2D area model as shown in Figure 1. In the 1D model the device is divided into several slots which could be separately reconfigured. This model simplifies the scheduling mechanism and trades this simplification for a sub-optimal utilization of the given hardware area. The more complex 2D area models allows for placing the tasks at any free position on the reconfigurable device [2].

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

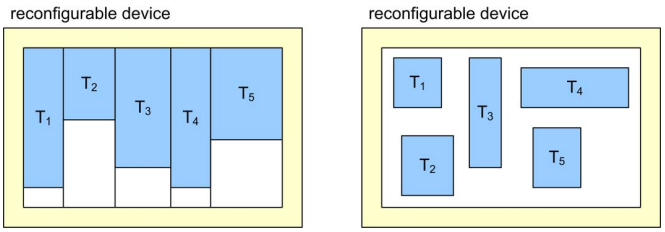


Fig. 1. 1D and 2D area models for the partial run-time reconfiguration.

Partial reconfiguration is currently provided by FPGAs from XILINX. The Vertex II family provides reconfiguration of slots, thus a 1D area model is supported. The procedure of reconfiguration is either accomplished by providing the configuration bitstream over the JTAG interface or is also achieved internally over the so called ICAP interface.

Nevertheless, both models expose themselves as NP hard scheduling problems, thus heuristic scheduling policies are to be considered like for example earliest deadline first (EDF).

This paper presents a design space exploration technique which computes a set of different design alternatives and features a time efficient scheduling algorithm.

The rest of the paper is structured as follows: In Section II an overview of existing run-time reconfiguration techniques is given. Section III refers on the deduction of area-time trade-offs. In Section IV the scheduling problem is formulated and mapped to a level strip packing problem and in Section V a scheduling algorithm is presented. Furthermore, in Section VI on results regarding execution time and optimization efficiency is reported. Finally, a conclusion ends the paper.

II. Related Work

In the work of Panainte et al. [3] two scheduling algorithms are proposed with the target to minimize the FPGA-area. Those scheduling algorithms take also into account the time that is needed for re-configuration. Two scenarios are discussed: all operations have to executed on the FPGA as well as some operation could also be executed in software.

Steiger et al. [4] present an online real-time scheduling problem and two heuristic approaches namely the horizon and stuffing technique. Both techniques are integrated in an operating system for reconfigurable devices.

978-1-4244-2110-7/08/$25.00 ©2007 IEEE

The benefits of the utilization of the FPGA by using three and five implementation variants, thus offering more flexibility to schedule the tasks has been recently shown by Danne and Platzner [5]. This work gives a theoretical gain on the utilization of an FPGA and is based on artificially generated tasks and their design alternatives which might not be achievable for real tasks sets. The authors have extended their work towards a reconfigurable operating system for a 1D area model [6].

III. Design Space Exploration

Basically, the process of generating design alternatives regarding run time and area of a given tasks is based on 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 [7]. 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 $a = \mathrm{GC}(BB_{\text{body}})$. 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 [8].

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.

Those transformations can be applied to the control flow graph (CFG) of the algorithm and the permutation of those independent transformations results in different realizations, that are called design points.

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 $DP = \prod_{i=1}^{N} |I(BB_i)|$. Hence, it is obvious that for large N an exhaustive search for all possible design points is an infeasible technique.

The problem of identifying optimal implementations can be generally described as multi-objective optimization problem [9] 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 realizations 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:

$$\begin{gathered} \mathbf{x}_1 \succ \mathbf{x}_2 \text{ (dominates) if } \mathbf{f}(\mathbf{x}_1) < \mathbf{f}(\mathbf{x}_2), \\ \mathbf{x}_1 \succeq \mathbf{x}_2 \text{ (weakly dominates) if } \mathbf{f}(\mathbf{x}_1) \leq \mathbf{f}(\mathbf{x}_2), \\ \mathbf{x}_1 \sim \mathbf{x}_2 \text{ (is indifferent to) if } \mathbf{f}(\mathbf{x}_1) \npreceq \mathbf{f}(\mathbf{x}_2) \wedge \mathbf{f}(\mathbf{x}_1) \nsucceq \mathbf{f}(\mathbf{x}_2). \end{gathered}$$

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 optimize all the objectives simultaneously. Therefore, we use the most common description of optimality for a multi-objective optimization given by Vilfredo Pareto [10].

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 $\mathcal{X}_p$ or short Pareto front. Further, we call a quality set $\mathcal{X}_q$ the approximation of the Pareto set $\mathcal{X}_p$.

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

Example 3.1: In Fig. 2 a design space for area and timing is depicted. The design point $\mathbf{x}_1$ is Pareto optimal and it dominates $\mathbf{x}_2$, $\mathbf{x}_3$, and $\mathbf{x}_4$ ($\mathbf{x}_1 \succ \mathbf{x}_2$, $\mathbf{x}_1 \succ \mathbf{x}_3$, $\mathbf{x}_1 \succ \mathbf{x}_4$), 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 $\mathbf{x}_4$($\mathbf{x}_2 \succeq \mathbf{x}_4$). 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.

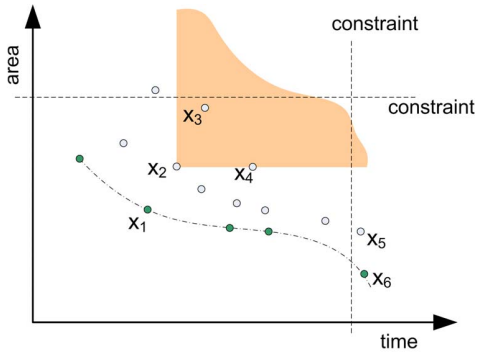


Fig. 2. Design space for area and timing trade-off.

Several heuristics are applicable in order to solve this optimization problem like for example simulated annealing

and tabu search. Usually, in this techniques the optimization result is only one design point. In order to identify a Pareto front with these 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 algorithms that maintain a set of solutions and iteratively converge towards the Pareto optimal set of solutions. Algorithms that are best suited to this behavior are genetic algorithms [11] or particle swarm optimization [12]. For example the Figures 3(a), 3(b), and 3(c) depict some examples of Pareto fronts that have been derived by problem specific genetic algorithm [13]. In this examples different control flow graphs properties are shown like N denotes the number of basic blocks, L the number of loops, DP the number of overall design points, and $|\mathcal{X}_q|$ the approximated number of Pareto optimal points. Figure 3(a) depicts a Pareto front of a CFG without any loop structures which results in a rather equally spaced distribution of design points. In Figure 3(b) and 3(c) loop structures are involved thus generating an accumulation of design points in certain areas.

Nevertheless, the number of design alternatives that are generated by this approach is far to high in order to be considered for scheduling. Therefore, a decision maker has to extract a small subset of design alternatives which supports the scheduling process optimally.

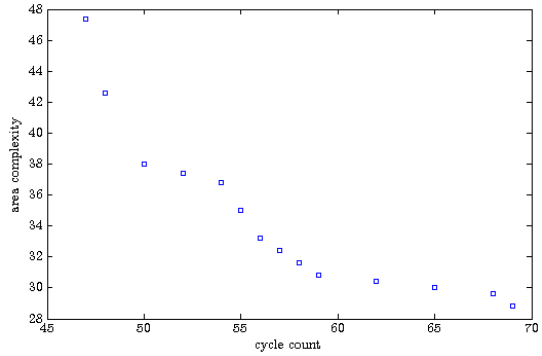
IV. Scheduling Problem

A set of tasks $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n\}$ is considered that has to be scheduled on a reconfigurable device. For each task $\mathcal{T}_i$ one or more implementation variants $\mathcal{T}_i = \{T_i^1, T_i^2, \ldots, T_i^m\}$ may exist. Hence, the cardinality $|\mathcal{T}|$ accounts for the number of tasks and the cardinality $|\mathcal{T}_i|$ denotes the number of task variants. Furthermore, a function $\mathrm{CC}(T_i^j)$ returns the number of needed execution cycles of the task variant T_i^j. Another function $\mathrm{AC}(T_i^j)$ returns the area consumption of the task variant T_i^j. It is assumed that the re-programmable device is divided into a set of slots $\mathcal{S} = \{S_1, S_2, \ldots, S_l\}$ where each slot S_i has a corresponding area A_i. Finally, a decision variable $x_{i,j}^k$ is introduced that indicates whether a task variant T_i^j is scheduled into one slot S_k or not

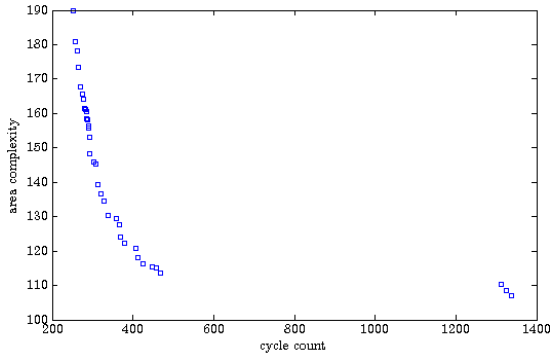
$$x_{i,j}^k = \begin{cases} 1 & \text{Task variant } T_i^j \text{ is scheduled in slot } S_k \\ 0 & \text{Task variant } T_i^j \text{ is not scheduled in slot } S_k \end{cases}. \tag{3}$$

Thus, the optimization targets the minimization of the slot with the highest utilization. With the decision variable $x_{i,j}^k$ this problem is formulated as binary programming problem

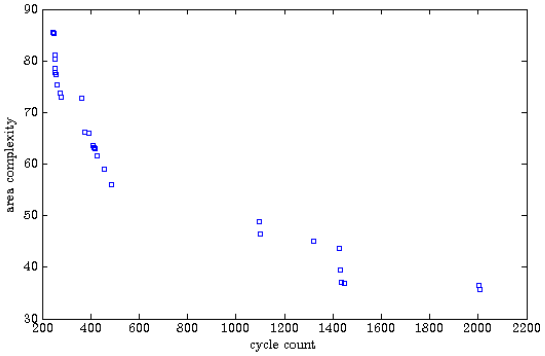
$$\min(\max\{\sum_{m=1}^{|\mathcal{T}|}\sum_{n=1}^{|\mathcal{T}_m|}\mathrm{CC}(T_m^n)x_{m,n}^1, \sum_{m=1}^{|\mathcal{T}|}\sum_{n=1}^{|\mathcal{T}_m|}\mathrm{CC}(T_m^n)x_{m,n}^2, \sum_{m=1}^{|\mathcal{T}|}\sum_{n=1}^{|\mathcal{T}_m|}\mathrm{CC}(T_m^n)x_{m,n}^3, \ldots, \sum_{m=1}^{|\mathcal{T}|}\sum_{n=1}^{|\mathcal{T}_m|}\mathrm{CC}(T_m^n)x_{m,n}^{|\mathcal{S}|}\}). \tag{4}$$



(a) $N = 10$, $L = 0$, $DP = 10^{10}$, $|\mathcal{X}_q| = 14$.



(b) $N = 35$, $L = 1$, $DP = 10^{35}$, $|\mathcal{X}_q| = 35$.



(c) $N = 15$, $L = 2$, $DP = 10^{20}$, $|\mathcal{X}_q| = 29$.

Fig. 3. Examples for different Pareto fronts.

Here, one entry of the set corresponds to the sum of cycle counts that are needed by the task variants that are scheduled to a slot. In addition to this optimization the constraint that for each task exactly one and only one task variant has to be scheduled must be fulfilled

$$\sum_{k=1}^{|\mathcal{S}|}\sum_{j=1}^{|\mathcal{T}_i|}x_{i,j}^k = 1,\ i = 1, \ldots, |\mathcal{T}|. \tag{5}$$

Furthermore, it is required that the size of a task variant T_i^j, if the corresponding decision variable equals one, has to be

smaller or equal than the size of the slot S_k

$$\mathrm{AC}(T_i^j)x_{i,j}^k \leq A_k, \qquad (6)$$
$$i = 1, \ldots, |\mathcal{T}|, j = 1, \ldots, |\mathcal{T}_i|, k = 1, \ldots, |\mathcal{S}|.$$

Let $V = \sum_{i=1}^{|\mathcal{T}|} |\mathcal{T}_i|$ denote the accumulated number of task variants. Thus, this problem formulation contains $V|\mathcal{S}|$ decision variables. Additionally, there have to be $|\mathcal{T}|$ constraints (6), and V constraints (5) to be fulfilled. Hence, it is of paramount importance to limit the number task variants and slots in order to keep the complexity of this problem controllable.

V. Scheduling Algorithm

The algorithm starts with the determination of slots and task variants. We define $A_{\max,\mathrm{i}} = \max\{\mathrm{AC}(T_i^1), \ldots, \mathrm{AC}(T_i^{|\mathcal{T}_i|})\}$ and similarly $A_{\min,\mathrm{i}} = \min\{\mathrm{AC}(T_i^1), \ldots, \mathrm{AC}(T_i^{|\mathcal{T}_i|})\}$. The minimal $CC_{\min,\mathrm{i}}$ and maximum $CC_{\max,\mathrm{i}}$ execution time of one task $\mathcal{T}_i$ are defined analogous. Thus, three slot sizes $S_{\max} = \max\{A_{\max,1}, \ldots, A_{\max,|\mathcal{T}|}\}$, $S_{\min} = \max\{A_{\min,1}, \ldots, A_{\min,|\mathcal{T}|}\}$, and $S_{\mathrm{av}} = (S_{\max} + S_{\min})/2$ as it is depicted for example in Figure 4 are chosen. Hence, it

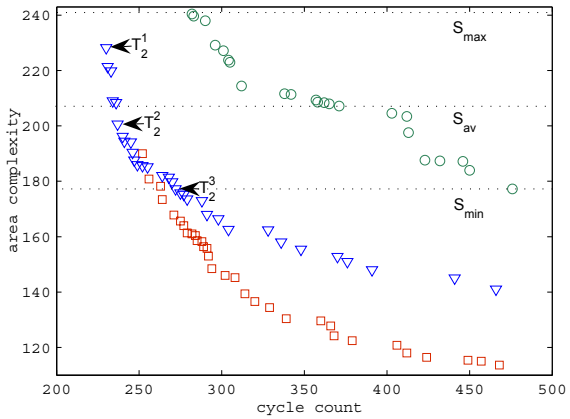


Fig. 4. Slot size and task variant determination.

is obvious that not the full range of design alternatives for each task has to be considered but only the task variants for each task that are closest to the determined slot sizes $S_{\max}$, $S_{\min}$, and S_{av}. For example like the three task variants that are chosen for task T_2 in Figure 4. The next step in the algorithm is the enumeration of solutions. The generation of solution can be accomplished with a depth first search of a decision tree (Figure 5). In this decision tree each level corresponds to one decision variable, thus the depth of tree equals the overall number of task variants V that have to be scheduled. The order of the decision variables in the tree is chosen such that tasks with less variants are at a higher position than others

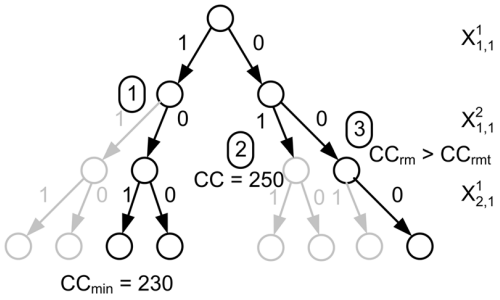


Fig. 5. Decision tree for the branch and cut algorithm.

in order to cause an early violation of the constraint (5), thus improving performance of the algorithm.

Cutting of the tree can be performed due to several stopping criteria:

- Violation of the constraints (6) and (5) (Figure 5 Label ①).
- Current cycle count is less than an already determined optimum (Figure 5 Label ②).
- While traversing the tree a remaining cycle count CC_{rm} and the cycle count that is needed to schedule the remaining tasks CC_{rmt} (time minimal implementation) is maintained. If $CC_{\mathrm{rmt}} \leq CC_{\mathrm{rm}}$ than cutting of the tree is performed (Figure 5 Label ③). For example see Figure 6, here the tasks T_1 to T_5 are already scheduled (T_6, T_7, and T_8 are not yet scheduled) and the overall remaining time to the current minimum is $CC_{\mathrm{rm}} = t_1 + t_2 + t_3$ and the time for the not yet scheduled tasks $CC_{\mathrm{rmt}} = CC_{\min,6} + CC_{\min,7} + CC_{\min,8}$.

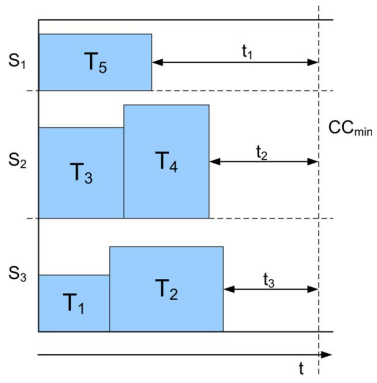


Fig. 6. Remaining cycle count.

As already mentioned the complexity of this scheduling problem (size of the decision tree) increases exponentially with the number of decision variables. Therefore, the aforementioned stopping criteria is extended to a heuristic. A factor $a \in]0, 1]$ is introduced $CC_{\mathrm{rmt}} \leq a\,CC_{\mathrm{rm}}$ that forces an earlier cutting of the decision tree.

VI. Results

The performance of the proposed algorithms (branchcut and the extension towards a heuristic with $a = 0.8$ (hbranchcut0.8) and $a = 0.6$ (hbranchcut0.6) are evaluated on twelve tasks sets $\mathrm{ts}_1, \ldots, \mathrm{ts}_{12}$. The sets differ in the number of tasks variants, number of basic blocks, number of loops, and number of loop nests. The proposed algorithms are compared to a basic level strip packing algorithm (levelpacking) which tries to optimize also the level size. The draw back of this algorithm gets apparent in Figure 7, due to the fact that the number of decision variables in this algorithm grows with the square of the overall design variants V. Thus, already the optimization of a task set with $V = 20$ already exceeds several hours of computation time on a standard PC. Whereas, the execution time of the branch and cut algorithm stays beneath one hour. Further decrease of run time is achieved with the heuristic approaches.

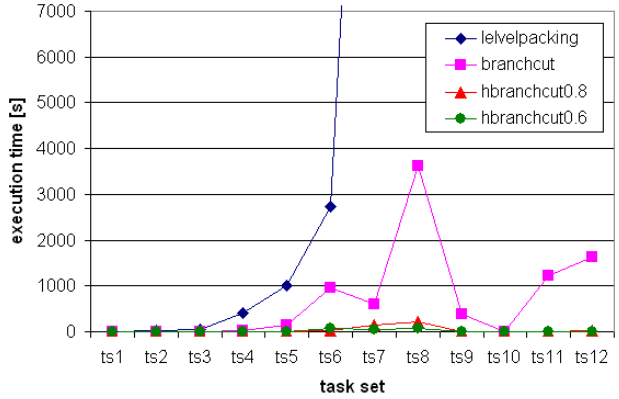


Fig. 7. Execution time of the scheduling algorithms.

The optimization performance of the algorithm is depicted in Figure 8. For the task sets ts_1 to ts_7 all algorithm achieve equal minimization result. For the task sets ts_7 to ts_{12} the algorithm hbranchcut0.8 deviates only up to 6% from optimization results that have been obtained with branchcut, whereas hbranchcut0.8 shows already deviations up to 20%.

As it has been mentioned earlier the decision variables within the tree are ordered in the way that decision variables that correspond to tasks with less variants are ordered at first. A comparison between ordered and not ordered decision variables has been performed where a performance gain of up to 20% has been observed, due to fact that the constraint (5) is violated earlier.

VII. Conclusions

The run time reconfiguration capability of FPGAs offers the possibility for the application as multi tasking device. The reconfigurable device can be utilized even more efficiently by offering several design alternatives. Such trade-offs for time and area can be effectively generated by evolutionary algorithms. Nevertheless, it is computational impossible to consider all the theoretically achievable design points for an exhaustive search for the schedule. An scheduling algorithm has been presented that reduces the number of design alternatives that are used for the scheduling. A depth first search algorithm is applied that constructs solutions in feasible time compared to a classical level strip packing formulation with comparable performance. With the extension to a heuristic algorithm execution time is further reduced to several seconds.

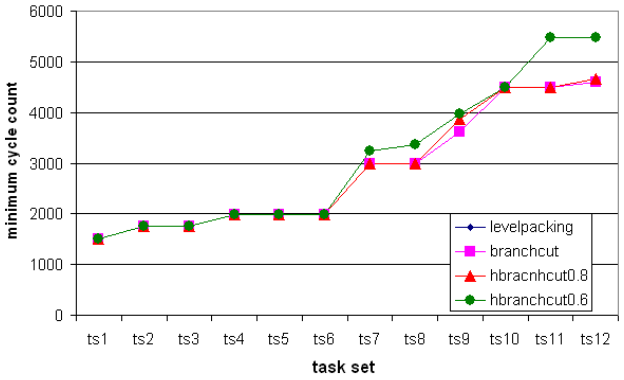


Fig. 8. Optimization results of the scheduling algorithms.

References

[1] K. Compton and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," *ACM Computing Surveys*, vol. 34, pp. 171–210, 2000.

[2] M. Hübner, C. Schuck, M. Kühnle, and J. Becker, "New 2-Dimensional Partial Dynamic Reconfiguration Techniques for Real-time Adaptive Microelectronic Circuits," in *Symposium on Emerging VLSI Technologies and Architectures*, pp. 6–11, 2006.

[3] E. M. Paniante, K. Bertels, and S. Vassiliadis, "FPGA-area Allocation for Partial Run-Time Reconfiguration," in *ProRISC Workshop on Circuits, Systems, and Signal Processing*, pp. 415–420, 2005.

[4] C. Steiger, H. Waldner, and M. Platzner, "Heuristics for Online Scheduling Real-time Tasks to Partially Reconfigurable Devices," in *International Conference on Field-Programmable Logic and Applications*, 2003.

[5] K. Danne and M. Platzner, "Partitioned Scheduling of Periodic Real-Time Tasks onto Reconfigurable Hardware," in *International Parallel and Distributed Processing Symposium (IPDPS'06), Reconfigurable Architecture Workshop (RAW'06)*, pp. 8–15, 2006.

[6] H. Waldner and M. Platzner, "Online Scheduling for Block-partitioned Reconfigurable Devices," in *Design Automation and Test in Europe Conference and Exhibition*, pp. 290–295, 2003.

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

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

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

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

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

[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] M. Holzer, B. Knerr, and M. Rupp, "Design Space Exploration with Evolutionary Multi-objective Optimisation," in *IEEE Int. Symposium on Industrial Embedded Systems (SIES)*, (Lisbon), pp. 126–133, July 2007.