# PARETO FRONT GENERATION FOR A TRADEOFF BETWEEN AREA AND TIMING

*M. Holzer and B. Knerr*

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

## ABSTRACT

Different implementation possibilities of an algorithm into hardware offer a variety of design solutions. Only best solutions so called pareto optimal solutions are for design decision of interest. Heuristic methods are preferred for the generation of these pareto optimal solutions, because exhaustive search methods cannot efficiently cope with this problem. This paper describes the mapping of this problem to a genetic algorithm. Further the generation of pareto optimal solutions is presented within an example.

## 1. INTRODUCTION

The design flow of embedded systems includes several optimisation steps like for example bit-width optimisation [1] and hardware/software partitioning [2]. Those optimisations usually consider only one implementation of an algorithm with certain values for time, area, and power. A better optimisation can be achieved if more than one implementation type is provided. Different realisations of processes are achieved by applying diverse source code transformations. Especially loop transformations like loop unrolling with varying unrolling factors and loop pipelining are to be mentioned, as well as data flow transformations like tree height reduction. Some high level synthesis tools allow for automatic application of these techniques (CatapultC from Mentor Graphics [3], SPARK [4]).

Those transformation can be applied to sub parts of the algorithm and the permutation of those independent transformations results in different realisation that are called design points. Additionally requirements from the specification may set constraints on the set of feasible design points. Maybe 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. This paper describes the generation of optimal solutions for area and timing tradeoffs with a genetic algorithm.

The rest of the paper is organised as follows. Section 2 gives an overview of related work in the field of design space exploration. In Section 3 definitions of terms for graphs that are used further on are provided. Further in Section 4 a definition for the pareto front is given and the generation of a design space is described. The mapping of the optimisation problem to a genetic algorithm is presented in Section 5. Further the generation of a pareto front is shown on an example in Section 6. Finally some conclusions are drawn.

## 2. RELATED WORK

Design space exploration has been performed on different trade offs between timing, power, and area as well on different abstraction levels.

Design space exploration for multi processor architectures is presented in [5]. In this work a focus is given on the comparison of fast against accurate estimations generated by simulation traces.

A focus on system optimisation and exploration for power is presented within the work of Henkel [6]. In this work mutual effects of certain system parameters like cache size or main memory size are considered.

Haubelt and Teich [7] generate an estimated pareto front by applying an hierarchical approach. They use pareto front arithmetics [8] for combining different pareto fronts of subproblems. Further extensions of this approach utilise particle swarm optimisation [9].

## 3. GRAPH PREREQUISITES

The following list enumerates the basic definitions for graphs that are needed in the remaining part.

**Definition 1** *Graph*
*A **graph** $G(\mathcal{V}, \mathcal{E})$ is defined as an ordered pair of a set $\mathcal{V} =$*

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

$\{v_1, v_2, ..., v_{|\mathcal{V}|}\}$ *of vertices and a set* $\mathcal{E} = \{e_1, e_2, ..., e_{|\mathcal{E}|}\}$ *of edges.*

**Definition 2** *Edge*
*The set of edges* $\mathcal{E}$ *is defined as a 2-tuple of vertices* $\mathcal{E} = \{(v, w) \,|\, v, w \in \mathcal{V}\}$

**Definition 3** *begin/end*
*The operation* ***beg*** *returns the source vertex, and the operation* ***end*** *returns the sink vertex of an edge* $e$ *as follows:* $\forall e = (v, w) \in \mathcal{E} : \mathrm{beg}(e) = v, \mathrm{end}(e) = w$.

**Definition 4** *path*
*A* ***path*** $\mathcal{P}$ *from a vertex* $v_0$ *to a vertex* $v_n$ *in a directed graph is a sequence of vertices* $v_0$, $v_1$, $v_2$, ..., $v_n$ *that satisfies:* $\forall i,\ i = 0 \ldots n-1\ \exists\, (v_i, v_{i+1}) \in \mathcal{E}$. *The vertex* $v_0$ *is the initial vertex and* $v_n$ *is the terminal vertex of the path. Equivalently a* ***path*** *from a vertex* $v_0$ *to vertex* $v_n$ *can be described by a sequence of edges* $e_1$, $e_2$, ..., $e_n$.

**Definition 5** *Control Flow Graph*
*A* ***control flow graph (CFG)*** *is a directed graph* $G(\mathcal{V}, \mathcal{E}, root, exit)$ *with the set* $\mathcal{V}$ *of vertices and the set* $\mathcal{E}$ *of edges. The vertices* $root$ *and* $exit$ *are special vertices because* $root$ *does not have any incoming and* $exit$ *does not have any outgoing edge (*$\mathrm{indegree}(root) = \mathrm{outdegree}(exit) = 0$*).*

## 4. DESIGN SPACE

A design space $S = \{D_1, D_2, ..., D_n\}$ is a set of design points $D$. Each design point specifies a point in the two dimensional design space with two corresponding metrics $\{a, b\}$. A feasible implementation $D_1 = \{a_1, b_1\}$ is said to be pareto optimal, if there is no other design point $D_2 = \{a_2, b_2\}$, which dominates it, i.e. all properties are better ($a_1 < a_2$ and $b_1 < b_2$). The set of pareto optimal points is called pareto optimal set or short pareto front. In Fig. 1 a design space for area and timing is depicted. The design point $D_1$ is pareto optimal and it dominates $D_2$. Also some constraints for maximum timing and area are shown, so that some producible design points get unfeasible.

In general an algorithm within a function, which is written in form of sequential code, can be decomposed into a 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 (Fig. 2).

For each basic block $BB$ an execution time $\mathrm{CC}(BB)$ and an area $\mathrm{A}(BB)$ is assumed (e.g. $BB_2$ in Fig. 2). The execution time of one path $\vec{p}_j$ can be expressed with

$$t(\vec{p}_j) \;=\; \vec{p}_j \vec{C}_{beg} + \mathrm{CC}(exit). \tag{1}$$

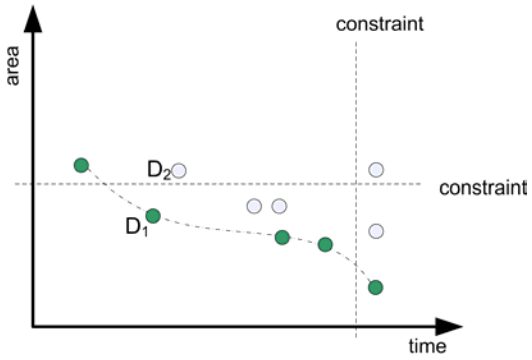


**Fig. 1**. Design space for area and timing trade off.

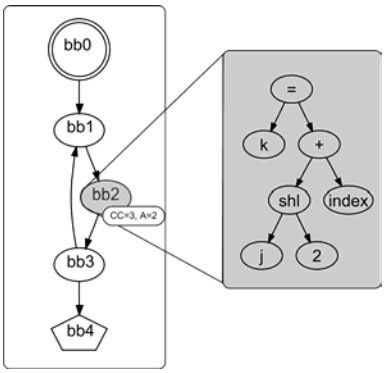


**Fig. 2**. CFG and a DFG for $BB_2$ with annotations for the corresponding cycle count (CC) and area (A).

Here the vector $\vec{C}_{beg}$ represents the execution times of the basic blocks of the CFG,

$$\vec{C}_{beg} \;=\; \begin{pmatrix} \mathrm{CC}(\mathrm{beg}(e_1)) \\ \mathrm{CC}(\mathrm{beg}(e_2)) \\ \mathrm{CC}(\mathrm{beg}(e_3)) \\ \vdots \\ \mathrm{CC}(\mathrm{beg}(e_{|\mathcal{E}|})) \end{pmatrix}. \tag{2}$$

The entries are built by assigning each entry of the vector the execution time of the corresponding basic block.

Vice versa the area for one path $\vec{p}_j$ can be expressed with

$$a(\vec{p}_j) \;=\; \vec{p}_j \vec{A}_{beg} + \mathrm{A}(exit). \tag{3}$$

Here the vector $\vec{A}_{beg}$ represents the needed area of the basic blocks of the CFG,

$$\vec{A}_{beg} \;=\; \begin{pmatrix} \mathrm{A}(\mathrm{beg}(e_1)) \\ \mathrm{A}(\mathrm{beg}(e_2)) \\ \mathrm{A}(\mathrm{beg}(e_3)) \\ \vdots \\ \mathrm{A}(\mathrm{beg}(e_{|\mathcal{E}|})) \end{pmatrix}. \tag{4}$$

The entries are built by assigning each entry of the vector the area of the corresponding basic block. Here no reuse of operations within one path is assumed (pipelined design). Further in order to have a more appropriate estimation the reuse capabilities within several CFG paths has to be accounted for example with an operation type dependent parallelism metric as defined in [10]. Also the size of the CFG accounts with its number of states to the whole area. Hence, for ease of computation the area of the longest path is assumed to be equal to the whole area consumption of the CFG implementation.

Different design points are achieved by source code transformations: The execution time of the DFG of one basic block can be approximated by the depth of the corresponding DFG. At one extreme there is a complete sequential implementation, whereas at the other side maximum parallelism allows for a minimal timing. For example in Fig. 3 a DFG with different realisation is shown. Here a cycle count of three is achieved by sequentially performing the additions, whereas a completely parallel implementation achieves a cycle count of two.

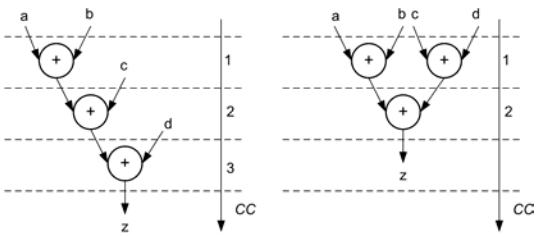


**Fig. 3**. Different implementation possibilities for a DFG.

A second potential for code transformations can be identified for loop structures. Here a complete loop unrolling tries to parallelise as many operations as possible, whereas with varying unrolling factors different performance values can be achieved.

Based on the described code variants each BB is annotated with a set of possible implementation types $I(BB) = \{(a_1, t_1), (a_2, t_2), ..., (a_N, t_N)\}$. The number of possible design points for a system grows approximately in the order of $I_{avg}^d$. Here $I_{avg}$ is the average number of available implementation types for the basic blocks and $d$ is the number of basic blocks of the longest CFG path. The CFG with all its different implementation for one BB spans a whole design space (Fig. 4).

## 5. GENETIC ALGORITHM

The problem of identifying a pareto optimal front can be treated by the application of a genetic algorithm [11]. The basis of the algorithm is the representation of the chromosome, which is vector of the length of the number of basic blocks in the considered path (Fig. 5).

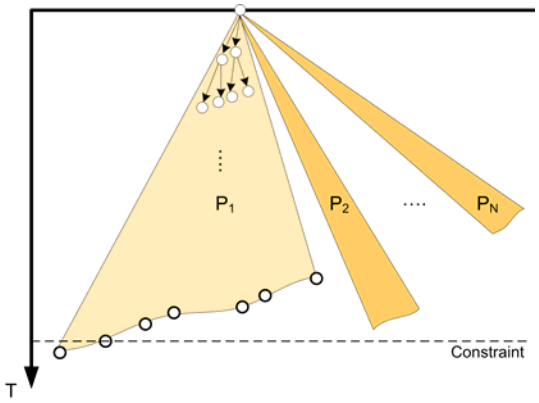


**Fig. 4**. Design space constructed for all paths of a CFG.

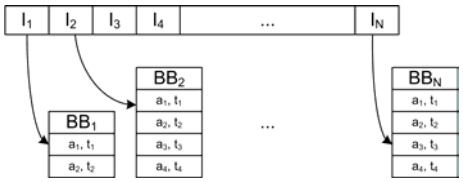


**Fig. 5**. Chromosome representation for the genetic algorithm.

Each element of this sequence selects one implementation type of its corresponding basic block. As first step of the algorithm a initial population is generated, where the implementation types are randomly generated. Now the parents for the new generation are found by binary selection. This means that two chromosomes are selected and the one, which has a shorter distance to the current pareto front is selected. After that the parents are recombined by joining sub sequences of two different chromosomes, in order to get the new population. The best solutions within one population are preserved for the next generation (elitism). Further the implementation types of certain chromosomes are probabilistically changed in order to guarantee the diversity of the population (mutation). This procedure is repeated until no new pareto optimal points found within one iteration of the algorithm.

## 6. EXAMPLE

The genetic algorithm has been applied to a CFG with 24 basic blocks in its longest path. The number of implementation types varies from two up to ten different implementation. This results in a search space of about $10^{10}$ possible design points. The population consists of 128 individuals and the mutation probability is 10%. In Fig. 6 the initially generated population $P_1$ and the population $P_{48}$ is shown

where the effect of the concentration of the whole population to a front can be seen.

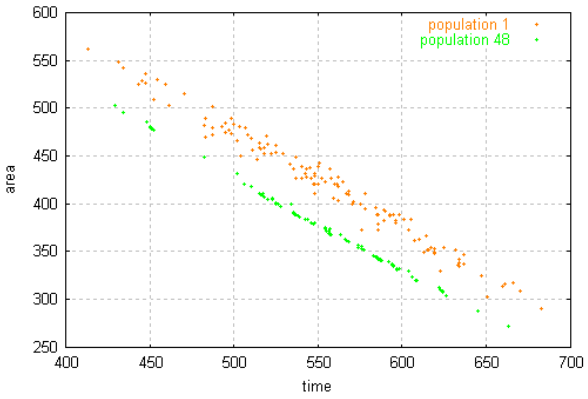


**Fig. 6**. Population $P_1$ and $P_{48}$.

In Fig. 7 the actual pareto optimal front of the population $P_1$ and $P_{48}$ is depicted. The finally found pareto front consists of 120 design points. So nearly the whole number of individual is concentrated in the pareto front. The remaining eight points that are not part of the front are caused by the mutation probability of 10%.

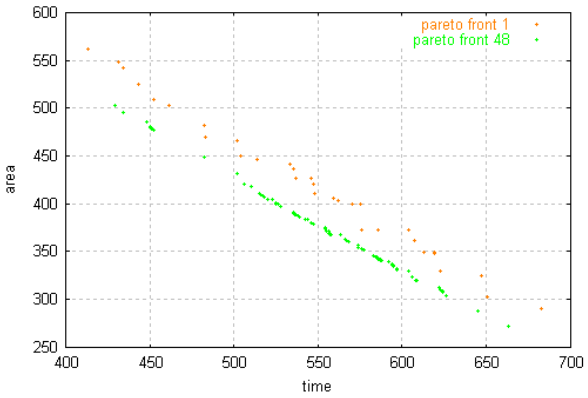


**Fig. 7**. Pareto front of population $P_1$ and $P_{48}$.

## 7. CONCLUSIONS

The design space of area and time trade offs grows exponentially with the size of the algorithm that should be implemented. This design space cannot feasibly to be searched in efficient time. Nevertheless, it is of high importance to provide the designer or other optimisation tools pareto optimal design points in order to get efficient solutions. This paper showed the application of a genetic algorithm to the problem of identifying pareto optimal solution of the time and area design space. The effectiveness of this approach has been shown on an example.

## 8. REFERENCES

[1] P. Belanović and M. Rupp. Automated Floating-point to Fixed-point Conversion with the *fixify* Environment. In *International Workshop on Rapid System Prototyping RSP'05*, pages 172–178, June 2005.

[2] B. Knerr, M. Holzer, and M. Rupp. HW/SW Partitioning Using High Level Metrics. In *International Conference on Computing, Communications and Control Technologies (CCCT)*, pages 33–38, 2004.

[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*, pages 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*, pages 461–466, 2003.

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

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

[7] Christian Haubelt and Jürgen Teich. Accelerating Design Space Exploration. In *International Conference on ASIC*, pages 79–84, 2003.

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

[9] Sanaz Mostaghim and Jürgen Teich. Covering Pareto-optimal Fronts by Subswarms in Multi-objective Particle Swarm Optimization. In *Congress on Evolutionary Computation*, volume 2, pages 1404–1404, 2004.

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

[11] Kalyanmoy Deb, Amrit Pratap, and T. Meyarivan. A Fast and Elitist Multi-Objective Genetic Algorithm: NSGA-II. In , 2003.