An Approach to the Adaptation of Estimated Cost Parameters in the COSYMA System

D. Herrmann, J. Henkel, R. Ernst

Technische Universität Braunschweig
Institut für Datenverarbeitungsanlagen
Hans-Sommer-Straße 66
D–38106 Braunschweig
Germany
Email: henkel@ida.ing.tu–bs.de

Abstract

Hardware/software partitioning is one of the key issues in hardware/software co–design. The system COSYMA uses simulated annealing based on estimated costs. Deviations between estimations and real costs seem unavoidable due to synthesis, compiler and communication effects. This paper describes an approach to adapt the estimation. The results show fast convergence of estimated to real costs.

1 Introduction

The system COSYMA (**COSY**nthesis of e**M**bedded **A**rchitectures) has been described in earlier publications (e.g. [ErHeBe93]). So far hardware/software partitioning in COSYMA was based on estimated results only. This has lead to suboptimal results. This paper presents the adaptation of the estimation to actual values in an *outer partitioning loop*.

In this section a summary of COSYMA is given. Section 2 describes a model for the partitioning process while section 3 shows how adaptation of estimated values is realized. In section 4 first results are shown and discussed. Finally, section 5 gives a conclusion.

COSYMA is an experimental system for hardware/software co-design of small embedded real–time–systems. The target architecture consists of a standard RISC processor core (we use the SPARC architecture) and an application specific co-processor. Communication is done through shared memory with a CSP type protocol (Communicating sequential processes). The input is a real–time system described in a

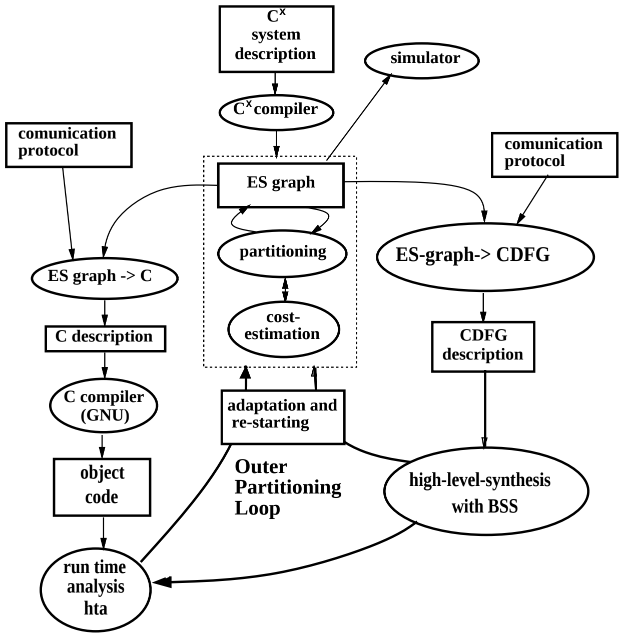

Figure 1: The design flow of the COSYMA experimental system

superset of the C language, C^x, with time constraints and processes. As shown in figure 1 this input description is translated to an Extended Syntax Graph (ESG, see [BeHeEr93]). The ESG represents a compromise for different requirements to the system:

- The hardware/software partitioning process should take place at this level of abstraction in order to be independent from the hardware architecture.
- Parts which have been selected to be realized in hardware should be easily translated to hardware and those parts which have been selected to remain in software should be easily translated to software (a C–program).
- Dataflow analysis for different co-synthesis steps like scheduling, profiling, communication estimation, translation to other target languages etc. should be supported.

The major aim of COSYMA is the automated hardware/software partitioning process. There are only very few automated partitioning approaches, so far. In contrast to our approach, the approach in [GuMi92] is hardware–oriented i.e. it starts with a hardware solutionand moves components to software until constraints are violated.

COSYMA partitions at basic block and function level (including hierarchical function calls). For partitioning we chose the simulated annealing algorithm because of the flexibility in the cost function and the possibility to trade–off computation time versus result quality. The annealing temperature control is adaptive as described in [OG89]. A characteristic feature of our partitioning problem is that we start with an all–software solution which is non feasible. The cost function has been defined such that it first forces the annealing to reach a feasible solution before the other optimization goals (area) become effective.

The cost calculation contains components for expected hardware execution times (estimated by a scheduling on the ES-graph), software execution times (determined by a timing analysis tool, see [YErBeHe93]), communication (estimation through data flow analysis), hardware costs (implicitly by number of basic blocks moved to hardware) and is based on profiling data.

For the sake of fast turnaround times these values are estimated and the cost function is updated in each step of the annealing algorithm. This iterative partitioning based on estimation is called *inner partitioning loop* and has already been described in detail in [ErHeBe93].

After partitioning those parts which have been selected to be realized in software are translated to a C-program thereby inserting code for the communication with the co–processor. The rest is translated to the input description of the high–level synthesis system BSS (see [Ho93]), also inserting code for communication — here communication with the standard processor core — and an application specific co-processor is synthesized. The co-processor is able to execute a number of unconnected code segments of the original system description. A code that is transmitted at each co-processor call tells which segment has to be executed.

Last step in the design flow is a fast timing analysis of the whole hardware/software system ([YErBeHe93]). It reveals whether the real-time constraints in the input description could be fulfilled or not. The experiments with COSYMA showed (expected) discrepancies between estimated and actual execution times of the partitioned hardware/software system. There are several reasons:

- Compiler effects are hard to predict and depend on the selected compiler. Furthermore the partitioning itself influences the optimization potential because it changes the code when it moves some blocks to hardware and replaces them by communication operations
- In order to synthesize an application specific co-processor hardware with a high speedup compared to a standard RISC processor complex hardware optimization as used in BSS is necessary ([HoEr93]) which is even hard to predict.

These are the reasons why an *outer partitioning loop* is important i.e. the re-starting of the partitioning process with an adaptation of the costs of those segments which have been estimated too inaccurately.

The aim of this paper is to describe how the costs can be adapted and how this iterative procedure can be managed. In figure 1 this part is described as the *outer partitioning loop, adaptation and re-starting.*

2 A model for nested inner–outer loop partitioning

To become independent of cost function, individual tool performance and input example, a general-

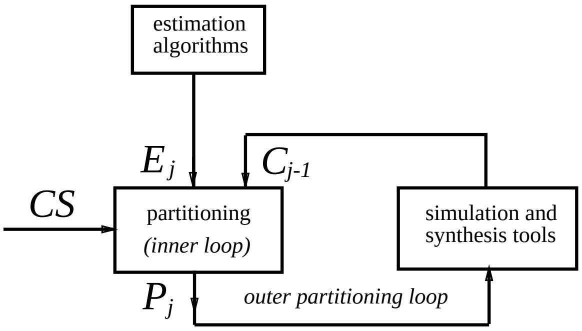

Figure 2: Model of the nested partitioning loops

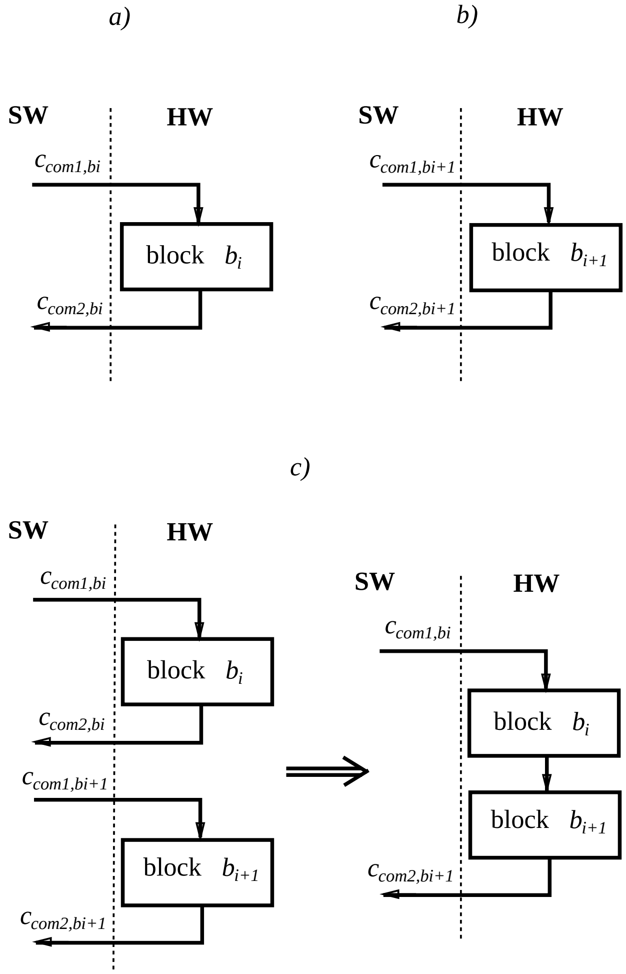

Figure 3: Costs for 3 different cases of a partition

ized model of the nested loop partitioning process is derived.

As stated in section 1 partitioning in COSYMA takes places at basic block[1] or function level. As for the partitioning there is no difference in moving a single basic block bb_i or a sequence of basic blocks $b_i = \{bb_1, \ldots, bb_k\}$ with $k \geq 1$, $i = \{1, \ldots, n\}$ and n the number of all sequences of basic blocks b_i which can potentially moved from software to hardware and vice versa. So in the following we will use the term *block* for both, bb and b. Furthermore let $P = \{b_i | b_i \ realized \ in \ hardware\}$ be the hardware part of a partition.

Figure 2 shows the nested partitioning loops from a different perspective. The partitioning process is controlled by a set CS of constraints (here time) and optimization goals (e.g. minimum additional hardware effort). For the j-th outer loop iteration, the partitioning process estimates the costs E_j[2].

Together with the previous partitions $P_1 \ldots P_{j-1}$ and their real costs $C_1 \ldots C_{j-1}$ the partitioning process determines the next hardware set P_j. In the next step all blocks of P_j are synthesized in hardware. Tools for verification and simulation in COSYMA (see figure 1) calculate the real costs C_j and repeats inner loop partitioning. It stops when

$$|E_j - C_j| < \Delta_{max} \tag{1}$$

and all time constraints are fulfilled where the user determines the precision by choosing Δ_{max}. We use a linear cost model. CS, E_j and C_{j-1} consist of several parts.

The values of E_j and C_j are composed of the costs caused by all blocks that have been implemented in hardware. It is not correct to sum the costs c_{b_i} for each individual block b_i assuming only this block would have moved to hardware:

$$C_j \neq \sum \{c_{b_i} | b_i \in P_j\}.$$

For explanation let us have a look at a simple example: assuming the single block b_i is moved to hardware, the costs c_{b_i} with $c_{b_i} = c_{b_i,0} + c_{b_i,com1} + c_{b_i,com2}$ arise, where $c_{b_i,com1}$ and $c_{b_i,com2}$ are those parts of the costs which are responsible for communication from software to hardware and from hardware to software and where $c_{b_i,0}$ represents all other kinds of costs caused by block b_i (figure 3a). The same is for block b_{i+1} where $c_{b_{i+1}} = c_{b_{i+1},0} + c_{b_{i+1},com1} + c_{b_{i+1},com2}$ (figure 3b). Picture c shows the case where b_i *and* b_{i+1} are moved to hardware. Assuming that the output data stream of

[1] *basic block* has the same meaning as described in [ASU].
[2] Cost estimation is described in [ErHeBe93]

b_i is equal to the input data stream of b_{i+1} no communication from hardware to software and vice versa at this point is necessary because the communication is internally without memory access. Therefore:

$$C = c_{b_i} + c_{b_{i+1}} - (c_{b_i,com2} + c_{b_{i+1},com1}).$$

Besides communication costs, there are compiler and synergetic effects. In general, we define c_{p_i} to be the additional costs when a set of blocks $p_i \in \mathcal{P}(P_j)$ is moved to hardware. Let $C(P_j) = C_j$. More precisely we can write ($\mathcal{P}$: power set):

$$C(P_j) = \sum_{p_i \in \mathcal{P}(P_j)} C_{p_i} \tag{2}$$

Example: If there are three blocks b_1, b_2 and b_3 which could be moved to hardware we could get a set of 8 terms including one term for the all software solution ($c_{\{\}}$):

$$\{c_{\{\}}, c_{\{b_1\}}, c_{\{b_2\}}, c_{\{b_3\}}, c_{\{b_1,b_2\}},$$
$$c_{\{b_1,b_3\}}, c_{\{b_2,b_3\}}, c_{\{b_1,b_2,b_3\}}\}$$

The term $c_{\{b_1,b_2,b_3\}}$ states that there may arise additional costs when *all* 3 blocks are moved to hardware. Obviously, the number of terms is $2^{card(P_j)}$.

3 Iterative partitioning procedure

Aim of the *outer partitioning loop* is to minimize the difference between the estimated costs and real costs (gained by simulation and/or analysis) of P_j according to equation 2. We will show how to apply the *Gaussian normal equations* technique to achieve convergence.

The inner partitioning loop corresponds to a simulated annealing run based on estimated values as mentioned in section 1. The approach for the outer partitioning loop described here does not depend on simulated annealing — it may be used for any other optimization algorithm.

The estimated cost of a set P_j is defined according to equation 2:

$$E(P_j) = \sum_{p_i \in \mathcal{P}(P_j)} e_{p_i,j} \tag{3}$$

Hereby the term $e_{\{\},j}$ stands for the costs which arise if no block is moved to hardware i.e. an all software solution. As a simple example, there may be a partition consisting of three blocks, $P(\{b_{23}, b_7, b_{15}\})$. According to equation 3 we are expecting $2^3 = 8$ terms in the sum:

$$\begin{aligned} E(\{b_{23}, b_7, b_{15}\}) \quad = \quad & e_{\{\}} + e_{\{b_7\}} + e_{\{b_{15}\}} + e_{\{b_{23}\}} \\ & + e_{\{b_7,b_{15}\}} + e_{\{b_7,b_{23}\}} \\ & + e_{\{b_{15},b_{23}\}} + e_{\{b_7,b_{15},b_{23}\}} \end{aligned} \tag{4}$$

All the terms in 3 and 4 are pre-computed once before the iterative partitioning process starts, such that cost function estimation is fast during partitioning. The real costs $C(P_j)$ are only gained when hardware synthesis, simulation etc. have been executed. For reasons explained in section 1 and in [HeBeEr93] estimated and real costs differ:

$$E(P_j) \neq C(P_j) \tag{5}$$

Or in general we can write:

$$\vec{x}^T_{P_j} \cdot \vec{e}_j \neq C(P_j), \tag{6}$$

where $\vec{x}^T_{P_j} \in \Re^{2^n}$ is the partition vector encoding the partition P_j, with n the number of blocks b in the input description (independent whether put to hardware or not). x_i corresponds to p_i and is set to "1", if $p_i \in \mathcal{P}(P_j)$, and to "0" otherwise. The vector $\vec{e}_j \in \Re^{2^n}$ consists of the corresponding estimated costs $e_{p_i,j}$.

For the example from equation 4 we get:

$$\begin{aligned} & (1\ 0_1\ \cdots\ 0_6\ 1_7\ 0_8\ \cdots\ 0_{7,14}\ 1_{7,15}\ 0_{7,16}\ \cdots \\ & \quad 0_{7,15,22}\ 1_{7,15,23}\ 0_{7,15,24}\ \cdots\ 0_{1,\ldots,n}) \cdot \vec{e_P} \\ = \quad & 1 \cdot e_{\{\}} + 1 \cdot e_{\{b_7\}} + 1 \cdot e_{\{b_{15}\}} + 1 \cdot e_{\{b_{23}\}} \\ & +1 \cdot e_{\{b_7,b_{15}\}} + \cdots \\ \neq \quad & C(P) \end{aligned}$$

Now 6 is corrected such that

$$\vec{x}_{P_j} \cdot \vec{e'}_j \stackrel{!}{=} C(P_j), \tag{7}$$

where (with $N = \{b_1, \ldots, b_n\}$)

$$\begin{aligned} \vec{e'}_j = \vec{e}_j + \vec{\Delta e}_j &= \begin{pmatrix} e'_{\{\},j} \\ e'_{\{b_1\},j} \\ \vdots \\ e'_{N,j} \end{pmatrix} \\ &= \begin{pmatrix} e_{\{\},j} \\ e_{\{b_1\},j} \\ \vdots \\ e_{N,j} \end{pmatrix} + \begin{pmatrix} \Delta e_{\{\},j} \\ \Delta e_{\{b_1\},j} \\ \vdots \\ \Delta e_{N,j} \end{pmatrix} \end{aligned} \tag{8}$$

Vector $\vec{\Delta e_j}$ is a correction vector of minimal length. Using the Gaussian normal equations (least squares method) $\vec{\Delta e_j}$ can be determined.

The next step is the generation of a new partition P_{j+1} using $\vec{e}_{j+1} = \vec{e'}_j$ for estimation. In most cases P_{j+1} will differ from P_j. Assumed that the estimation is sufficiently accurate then typically $P_j \cap P_{j+1} \neq \{\ \}$ i.e. that one or more blocks are moved to hardware in partition P_j as well as in partition P_{j+1}. After each new partition the least squares method is used and another equation is added.

The procedure of iterative partitioning can be described by the following equation

$$\begin{pmatrix} 1 & x_{\{b_1\},P_1} & x_{\{b_2\},P_1} & \cdots & x_{N,P_1} \\ 1 & x_{\{b_1\},P_2} & x_{\{b_2\},P_2} & \cdots & x_{N,P_2} \\ 1 & x_{\{b_1\},P_3} & x_{\{b_2\},P_3} & \cdots & x_{N,P_3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{\{b_1\},P_r} & x_{\{b_2\},P_r} & \cdots & x_{N,P_r} \end{pmatrix} \cdot$$

$$\left(\begin{pmatrix} e_{\{\},r} \\ e_{\{b_1\},r} \\ e_{\{b_2\},r} \\ \vdots \\ e_{N,r} \end{pmatrix} + \begin{pmatrix} \Delta e_{\{\},r} \\ \Delta e_{\{b_1\},r} \\ \Delta e_{\{b_2\},r} \\ \vdots \\ \Delta e_{N,r} \end{pmatrix} \right) = \begin{pmatrix} C(P_1) \\ C(P_2) \\ C(P_3) \\ \vdots \\ C(P_r) \end{pmatrix}, \tag{9}$$

where r is the number of partitioning runs (inner partitioning loop) and where each row of matrix X has the same meaning as the vector in equation (7) for a single partitioning run. So for each partitioning run X will grow by one row and $\vec{C_P}$ will grow by one element. In shorter terms we can write

$$X \cdot (\vec{e}_j + \vec{\Delta e_j}) = \vec{C} \tag{10}$$

After each partitioning run the task of the least squares method is to find an adequate $\vec{\Delta e_j}$. In practical use the method is limited by different aspects:

- Not all cost components can be taken into account because the number grows very fast: e.g. a system description that consists of only 30 basic blocks (small system) would cause 2^{30} components — the absolute limit of a today's workstations. Therefore the number of components must be limited. We select those ones which are expected (i.e. estimated) to be the largest. Nevertheless there may be some which are not taken into regard and which are very large. In such a case the algorithm described above will be disturbed.

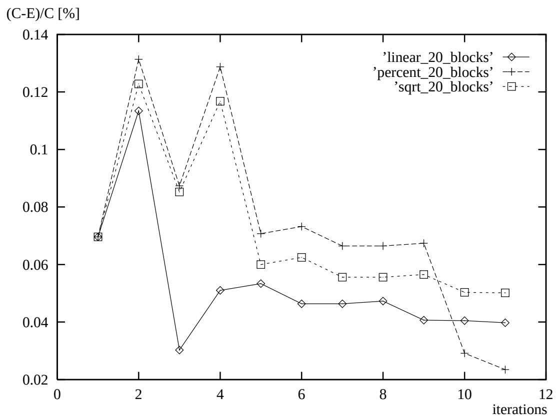

Figure 4: First outer loop partitioning run with 20 blocks

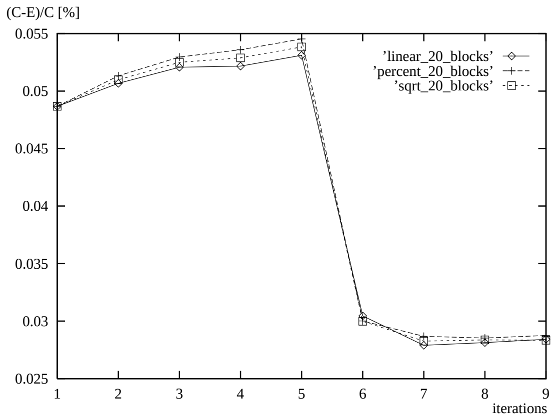

Figure 5: Second outer loop partitioning run with 20 blocks

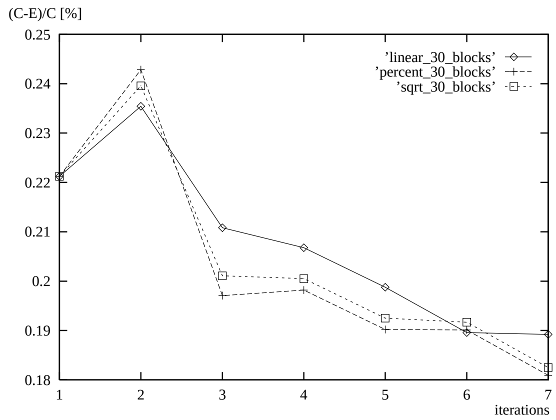

Figure 6: First outer loop partitioning run with 30 blocks

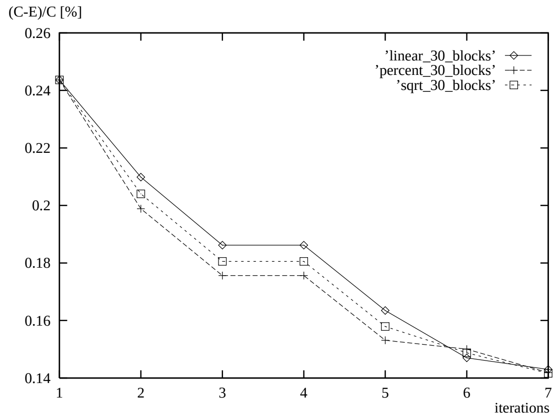

Figure 7: Second outer loop partitioning run with 30 blocks

4 Experiments

The experiments described in this section should show the feasibility of this method for usage in hardware software co–design. A crucial question is how the limitations in the number of components influence the convergence and the adaptation of the estimation. Instead of using COSYMA, which would be very time consuming an experimental tool has been developed with the structure of figure 2. Given the number of basic blocks, the tool stochastically generates the c_i and $e_{i,j}$ components according to a user defined distribution function. This makes these investigations independent of benchmark characteristics. The distributions used here are close to benchmarks which have also been used in COSYMA.

The costs of P_i are distributed in such a way that a small subset (10% to 20%) is very cheap for the case of implementing in hardware. That reflects the experiences we made with real–time systems where typically a few nested loops were executed very often at relatively low communication costs (e.g. see [ErHeBe93]). Those ones are appropriate candidates for implementation in hardware.

In order to cope with the complexity of the cost function, the number of components is limited to all components for single blocks ($e_{\{x\}}$), those between single blocks ($e_{\{x,y\}}$) and a subset of those between 3 blocks ($e_{\{x,y,z\}}$) with a value above a given threshold. Studying the cost function of real–time examples showed that the other components are mostly very small or equal to 0.

For practical use, convergence ($\leq \Delta_{max}$) has to be reached with a small number of iterations (10 to 15, see [HeErHo+94]). All the following experiments are limited to this range. The experiments in figures 4 to 5 are based on $n = 20$ blocks, those in figures 6 and 7 show the results for $n = 30$ blocks. The horizontal axis describes the number of runs i.e. number of partitioning runs in the inner partitioning loop, the vertical axis shows the relative error $\frac{E_j - C_j}{C_j}$ in percent. Each figure contains 3 solutions gained by different methods for correcting the error $E_j - C_j$: the one named *"linear..."* creates the correction vector $\Delta\vec{e}$ (see eq. 10) without taking into account the absolute value of a component in $\vec{e}$. The other one named *"percent..."* corrects the error dependent of the absolute value that must be corrected and the last one is a mixture of the previous methods, taking the square root of both.

Figure 4 shows that for all three methods the quality varies very strongly. Beginning with the 5th iteration the quality becomes continuously better. The explanation is that at the beginning the correction is insecure because of many undetermined components. Due to the fact that the starting value of the measure in figure 5 is lower than in figure 4 (about 0.05 compared to 0.07) it is hard to find an improvement at the beginning. After 5 runs for the inner loop partitioning here also a significant improvement is gained.

The measures for 30 blocks behave almost in the same way but start at much worse qualities (about 0.23 as opposed to about 0.05 for the 20 block run; that is a factor of 4 !) The effect is due to the increasing inaccuracy: because of the complexity of 2^n for the 30 block problem much more components had to be neglected. The solution becomes the more imprecise the larger the number n of blocks becomes. But in all cases a significant relative improvement is reached through estimation adaptation.

Another effect that is common to all measures is that the correction method based on the square root (as a mixture between relative and absolut correction) aims at results between the two other methods, whereas sometimes (in figures 6 and 7) the *"linear..."* method is the best and in other cases (in figures 4 and 5) the *"percent..."* method aims the best results.

There is a great potential for improvement, we are currently working on:

- assumed there are costs that are hard to estimate and the according blocks dominate the total costs. Those ones could be recognized and the costs could be measured individually by a test run.

- we made the experience that in case equation 10 becomes over-determined, the improvement varies very strongly or becomes worse (this the case when the inner partitioning loop delivers "similar" partitions). A recognition of this case would be very helpful.

There are many other heuristics potentially providing good improvements which we are currently implementing and investigating.

5 Conclusion

Partitioning in the COSYMA system for hardware/software co–design is done with simulated annealing based on estimated values, as described in [ErHeBe93]. This is what we call the *inner partitioning loop.*

Here we introduced an approach for an *outer partitioning loop* where the cost estimations are adapted when they deviate from the real costs above a given limit. This leads to an outer loop. It is only useful if the number of iterations (correction steps) is small, say ≤ 15, because each correction step causes an execution of the inner partitioning loop.

The experiments show that an approach based on the least squares method can quickly correct the output of the inner partitioning. So the nested loop approach in COSYMA is feasible.

The experiments have also shown that heuristics are necessary for gaining acceptable results. Current and future work will be to improve the method in order to reach better convergence.

References

[ASU] A.W. Aho, R. Sethi and J.D. Ullmann, *COMPILERS Principles, Techniques and Tools,*

[BeHeEr93] Th. Benner, J. Henkel and R. Ernst, *Internal Representation of Embedded Hardware–/Software–Systems,* Innsbruck, Codes/CASHE '93, May 1993.

[ErHe92] R. Ernst and J. Henkel, *Hardware-Software Codesign of Embedded Controllers Based on Hardware Extraction,* Estes Park, Colorado, Handout from Int'l Workshop on Hardware-Software Co-Design, Oct. 1992

[ErHeBe93] R. Ernst, J. Henkel and Th. Benner, *Hardware/Software Co-Synthesis for Microcontrollers,* IEEE Design & Test of Computers, pp. 64–75, Dec. 1993.

[GuCoMi92] R.K. Gupta, C.N. Coelho, G.D. Micheli, *Synthesis and Simulation of Digital Systems Containing Interacting Hardware and Software Components,* Proc. of DAC'92, pp. 225–230, 1992.

[GuMi92] R.K. Gupta and G.D. Micheli, *System-level Synthesis using Re-programmable Components,* EDAC'92 IEEE Computer Society Press, pp. 2–7, 1992.

[HeBeEr93] J. Henkel, Th. Benner and R. Ernst, *Hardware generation and partitioning effects in the COSYMA system,* Handouts from IEEE/ACM International Workshop on Hardware–Software Codesign, Cambridge, Massachusetts, 7.–8. Oct. 1993.

[HeErHo+94] J. Henkel, R. Ernst, U. Holtmann, Th. Benner, *Adaptation of Partitioning and High-Level Synthesis in Hardware/Software Co–Synthesis,* ICCAD'94 (accepted).

[Ho93] U. Holtmann,
High–Level Synthese mit BSS für den Einsatz im Hardware/Software Codesign (High–Level Synthesis with BSS for Usage in Hardware/Software Codesign),
Handouts from 6th. E.I.S. Workshop, Tübingen, Germany, 1993.

[HoEr93] U. Holtmann and R. Ernst,
Speculative computation for coprocessor synthesis,
Cambridge, Massachusetts, ICCD 1993, Oct. 1993.

[OG89] R. Otten, P. van Ginneken,
The Annealing Algorithm,
Kluwer, 1989.

[YErBeHe93] W. Ye, R. Ernst, Th. Benner and J. Henkel,
Fast Timing Analysis for Hardware-Software Co-Syntheis,
Cambridge, Massachusetts, ICCD 1993, Oct. 1993.