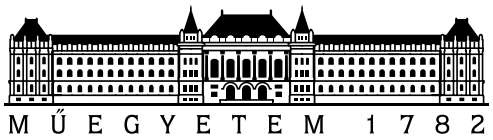

Budapest University of Technology and Economics
Department of Control Engineering and Information Technology

PARTITIONING ALGORITHMS FOR HARDWARE/SOFTWARE CO-DESIGN

Ph.D. thesis summary

Zoltán Ádám Mann

Supervisor: Dr. Péter Arató
Corresponding member of the Hungarian Academy of Sciences

Budapest
2004

1 Introduction

The requirements towards today's computer systems are tougher than ever. Parallel to the growth in complexity of the systems to be designed, the time-to-market pressure is also increasing. In most applications, it is not enough for the product to be functionally correct, but it has to be cheap, fast, and reliable as well. With the wide spread of mobile systems, size, heat dissipation, and energy consumption are also becoming crucial aspects for a wide range of computer systems [15], especially embedded systems.

Embedded systems have become a part of our lives in the form of consumer electronics, cell phones, smart cards, car electronics etc. These computer systems consist of both hardware and software; they together determine the operation of the system. In this context, hardware means application-specific hardware units, i.e., hardware designed and implemented specifically for the given system, whereas software means a program running on a general-purpose hardware unit, such as a microprocessor.

For much of the functionality of embedded systems, both a hardware and a software implementation is possible. Both possibilities have advantages and disadvantages: software is typically cheaper, but slower; moreover, general-purpose processors consume more power than application-specific hardware solutions. Hardware on the other hand is fast and energy-efficient, but significantly more expensive.

Consequently, it is beneficial to implement performance-critical or power-critical functionality in hardware, and the rest in software. This way, an optimal trade-off can be found between performance, power, and costs [C6]. Unfortunately, finding such an optimal trade-off is by no means easy, especially because of the large number and different characteristics of the components that have to be considered. Moreover, the differences between hardware and software as well as their interaction also contribute to the complexity of the design process.

To address specifically the problem of designing complex mixed hardware/software systems, a considerable amount of research has taken place in the field of *hardware/software co-design* (HSCD) since the early 1990s [8, 10, 11, 16, 19, 20, 23, 24]. The main steps of a HSCD framework are outlined in Figure 1.

The central task of HSCD is *partitioning*, i.e., deciding which components of the system should be implemented in hardware and which ones in software. This is clearly the step in which the above-mentioned optimal trade-off between the conflicting requirements should be found, also taking into account the communication overhead between hardware and software [C5].

In order to represent communication, the system to be partitioned is usually modelled with a graph, the nodes of which are the components of the system, and the edges the communication between them. The nodes and edges are usually assigned different cost values, which can mean for instance running time, occupied chip area, or heat dissipation. As can be seen in Figure 1, it is an important task of HSCD to determine the graph and the cost values. However, during partitioning, these are assumed to be given.

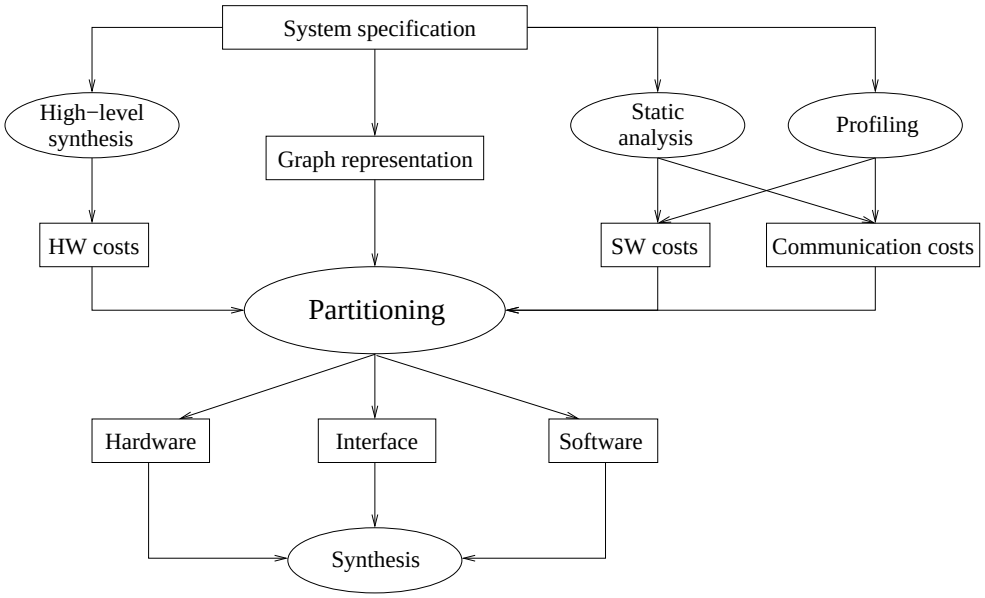

Figure 1: Block diagram of a possible hardware/software co-design framework

2 Aims of the research

The goal of the research was to investigate the hardware/software partitioning problem from an algorithmic point of view. This consists of two main tasks:

- Investigation of the algorithmic complexity of the problem. It was claimed by several researchers that partitioning is $\mathcal{NP}$-complete [3, 7, 16, 22], but only one particular formulation of the problem has been proven to be really $\mathcal{NP}$-hard [16]. However, that formulation also included the scheduling problem, and the proof also depended on the hardness of scheduling. On the other hand, the literature also includes several other formulations of the partitioning problem that do not include scheduling, and the complexity of these problem formulations is not proven. For this reason, one of my aims was to investigate the complexity of different formulations of the partitioning problem, possibly finding polynomially solvable cases, or to formally prove $\mathcal{NP}$-hardness.

- The other main objective of my research concerned the $\mathcal{NP}$-hard versions of the problem, because—as revealed by my investigations—several formulations of the problem are indeed $\mathcal{NP}$-hard even without scheduling. Nonetheless, these problems have to be solved somehow in practice. Therefore, I worked on efficient heuristics for these problems. In contrast to most previous work in the field, the primary aim was not to include as many details of partitioning as possible in the algorithms, but rather to enhance efficiency for a more limited problem.

A related task is to test, tune, and compare the heuristics on benchmark problems. It is important to determine the effectiveness and speed characteristics of the algorithms, because this is the basis for deciding which one to use in a given situation.

The most important features of the used problem formulations are as follows. Each node of the graph is assigned two nonnegative values called hardware cost and software cost, respectively. Each edge is assigned a so-called communication cost. The hardware cost of a given partition is defined as the sum of the hardware costs of the nodes that are in hardware. Software cost of a partition is defined similarly. The communication cost of a partition is the sum of the communication costs of the cut edges. In practice, both software costs and communication costs are often time-dimensional, and their sum is approximately equal to the running time of the whole system. For this reason it makes sense to consider these two costs either separately or summed.

P1: Decide whether there exists a partition for which both the hardware cost and the sum of the software cost and the communication cost are below given bounds.

P2: Find a partition for which the hardware cost is below a given bound, and the sum of the software cost and the communication cost is minimal.

P3: Find a partition for which the sum of the software cost and the communication cost is below a given bound, and the hardware cost is minimal.

P4: Find a partition for which both the hardware cost and the software cost are below a given bound, and the communication cost is minimal.

P5: Find a partition for which the weighted sum (with given nonnegative weights α, β, γ) of the hardware cost, the software cost, and the communication cost is minimal.

Figure 2: Problem formulations

Based on these notions, the considered problem formulations are shown in Figure 2.

3 Methodology

In the used problem formulations, hardware/software partitioning is a graph bipartitioning problem. Therefore the used methods stem from the field of graph theory, algorithm theory, and heuristic optimization.

For the proof of the $\mathcal{NP}$-hardness results I reduced well-known $\mathcal{NP}$-hard problems—specifically the KNAPSACK and MINIMAL BISECTION problems [4]—to the problems at hand. For the polinomiality proofs I used the fact that both the minimum cut and the minimum $s-t$ cut of a graph can be found in polynomial time [1].

In the heuristic algorithms, I made use of general-purpose heuristics (deterministic and randomized greedy algorithms, genetic algorithms [6]), as well as methods that had been developed for graph partitioning in general (Kernighan–Lin algorithm [17], Fiduccia–Mattheyses algorithm [9]). Furthermore, I used again a reduction to a minimum cut problem. In some cases, the efficient implementation and the appropriate data structure were highly non-trivial. I also addressed these questions; for example, I used an advanced data structure from the field of geometric algorithms for one of my heuristics.

I also implemented the new algorithms, and tested them empirically on numerous industrial benchmark problems and large random graphs. Some algorithms had tuneable parameters: these were also tuned based on the test runs on the benchmark problems. Moreover, the running time and effectiveness of the algorithms could be compared this way empirically.

4 New results

Thesis 1: I formally proved the following complexity results concerning the different formulations of the hardware/software partitioning problem [C5, J3].

Theorem 1 *The **P1** problem is $\mathcal{NP}$-complete, even in the special case when only graphs with no edges are considered.*

The proof shows that the **P1** problem for graphs with no edges is actually equivalent to the KNAPSACK problem. The next theorem is a consequence:

Theorem 2 ***P2** and **P3** are $\mathcal{NP}$-hard, even in the special case when only graphs with no edges are considered.*

The KNAPSACK problem, although $\mathcal{NP}$-complete, can be handled relatively efficiently, that is, there are so-called pseudo-polynomial algorithms for it [4]. The next two theorems show that, for general graphs, the hardware/software partitioning problem is harder than this:

Theorem 3 *The **P1** problem is $\mathcal{NP}$-complete in the strong sense.*

Theorem 4 ***P2** and **P3** are $\mathcal{NP}$-hard in the strong sense.*

Because of the equivalence to the KNAPSACK problem, hardware/software partitioning becomes relatively easy when communication costs can be neglected. According to the next theorem, this is especially true for the other extreme:

Theorem 5 *If hardware and software costs can be neglected, i.e., only communication costs have to be taken into account, then all of the defined problem formulations can be solved in polynomial time. This remains true even for the generalization in which some nodes can be prescribed to be in hardware and some others to be in software.*

The last theorem is about polynomially solvable special cases. The next theorem shows that the **P5** problem can be solved in polynomial time even in general:

Theorem 6 *The **P5** problem can be solved in polynomial time.*

Finally, an inapproximability result:

Theorem 7 *The **P4** problem is $\mathcal{NP}$-hard. What is more, if $\mathcal{P} \neq \mathcal{NP}$, then not even an approximation algorithm can exist for it.*

Thesis 2: I developed a new genetic algorithm for the hardware/software partitioning problem [C7].

In the dissertation, the algorithm is presented for the **P3** problem, but it can be easily adapted to the other problem formulations as well.

There had already been genetic partitioning algorithms, but my algorithm is more efficient. The main novelty is the consistent handling of so-called invalid individuals (i.e., partitions whose cost exceeds the bound). The most important features of the algorithm in this respect are the following:

- Although invalid individuals do not represent correct solutions of the problem, and it would hence seem logical not to allow them, I showed that allowing such individuals in the population improves the effectiveness of the algorithm. This is because often the 'almost valid' individuals carry the most valuable genetic patterns.

- Of course, valid individuals should be favoured; therefore, the fitness function has to be chosen in such a way that it punishes invalidity. I tested different functions for this purpose in order to choose the best one, because the empirical tests showed that this choice has great impact on the effectiveness of the algorithm. A function of the following form proved best:

$$f(P) = \begin{cases} \Theta' - H_P & \text{if } P \text{ is valid} \\ \Theta' - H_P - c * exc(P) - M & \text{if } P \text{ is invalid.} \end{cases}$$

  Here, $f(P)$ is the fitness value of partition P, H_P is its hardware cost, and $exc(P)$ is the measure of its invalidity, i.e., the percentage by which P exceeds the bound on the running time; finally, Θ', c, and M are appropriate constants. As can be seen, the conflicting goals of partitioning are unified in a single function in such a way that maximizing the fitness leads to validity and decreased hardware costs. On the other hand, a small exceeding of the bound can be compensated by a sufficiently low hardware cost.

- In order to return a valid partition at the end of the algorithm, it has to be ensured that the population contains at least one valid individual at all times. For this reason, I tested several methods for generating valid individuals into the initial population, without having to fear degeneration of the population. The ratio of valid versus randomly generated individuals in the initial population is also an important parameter of the algorithm.

- I showed that the effectiveness of the selection (elitism) operator can be enhanced significantly if not only the best individuals of the population are copied into the next one, but also the best invalid ones.

Moreover, I tested several different possibilities for the recombination operator, as well as different recombination and mutation ratios. The tuning of the parameters required a huge number of empirical tests, but choosing the best parameter configuration significantly improved the effectiveness of the algorithm.

Thesis 3: I developed a new Kernighan–Lin-type heuristic algorithm for the hardware/software partitioning problem [J3, O6].

The original Kernighan–Lin algorithm addressed the MINIMUM BISECTION problem [17], which is in several respects simpler than hardware/software partitioning. Therefore, the adaptation of the algorithm was far from obvious. Furthermore, since the publication of the original Kernighan–Lin algorithm, several improvements have been suggested [5, 9, 13, 14, 18, 21, 25], so it was also necessary to investigate how these improvements can be transferred to the case of the hardware/software partitioning problem. Specifically, I had to cope with the following problems during the adaptation of the algorithm:

- Specifying the gain function of the algorithm based on the different cost functions of hardware/software partitioning. At this point, the dilemma of invalid partitions was again an issue, and it again turned out that it is beneficial to allow a small exceeding of the bound, but of course with an appropriate mechanism to punish invalidity.

- The choice of the initial partition. Again, it is beneficial to start from a valid partition. Moreover, the effectiveness of the whole algorithm depends significantly on the quality of the initial partition; therefore, the dissertation investigates different possibilities for rapidly generating a high-quality initial partition.

- Tie-breaking strategy for the case when the best gain is shared by more than one node. I showed that it is advantageous to also regard nodes with similar gain values as if they had the same gain, and to use the LIFO strategy [13] to choose the winner from them.

- The method of locking moved nodes. As it turns out, the dynamic locking scheme [14] is better for hardware/software partitioning as well than the original locking scheme suggested by Kernighan and Lin.

The biggest challenge in the development of the algorithm was the choice of the data structure that would enable an efficient implementation. I proved that, for any data structure, the algorithm needs at least $\Omega(n \log n)$ time per phase. My implementation—which uses the range tree data structure [2]—reaches this lower bound for sparse graphs.

Finally, I also investigated how the used hardware/software partitioning model influences the applicability of the Kernighan–Lin heuristic. I showed that the algorithm can be extended without a significant runtime penalty to handle more cost functions. I also showed that scheduling can also be integrated into the algorithm; however, this makes it considerably slower.

Thesis 4: I developed a new heuristic, based on finding the minimum cut in graphs, which works by constructing auxiliary graphs—using different cost factors—from the original graph, finding the minimum cut in the auxiliary graphs, and selecting the best valid partition from them [J3, O6].

The main novelty of the algorithm is that it is—in contrast with previous works—not the adaptation of a well-known, general-purpose heuristic, but it was developed specifically for the hardware/software partitioning problem, and is based on the combinatorial properties of this problem.

The basis of the heuristic is the algorithm mentioned in Theorem 6, which solves the **P5** problem optimally in polynomial time. Based on this algorithm, a heuristic for the **P3** problem can be constructed in the following way. From the original graph, several **P5** instances are constructed using different α, β, γ factors. These problem instances are solved optimally using the polynomial-time algorithm for **P5.** From the resulting partitions, the best with respect to the original **P3** problem is taken, i.e., one that does not violate the constraint and has minimum hardware cost.

This way, the algorithm investigates only partitions that have a high quality in some sense: specifically, they are optimal solutions of a related problem (for appropriate cost factors). This makes it possible to achieve good results although only a small fraction of the search space is scanned.

A second consequence of this method is that the algorithm is able to infer non-trivial *lower estimates* on the optimum, which get better and better during the operation of the algorithm. This way, the algorithm always has an upper estimate on how far its best result found so far can lie from the optimum. This is a feature that none of the previously suggested partitioning algorithms possessed.

The speed of the algorithm depends heavily on the number of evaluated (α, β, γ) tuples, and how the next tuple to be evaluated is chosen. Clearly, only the ratio of the three parameters is important, and not their absolute values. Therefore, one of them, e.g. β, can be fixed, and only the other two have to be varied. This means that some points of the $\alpha - \gamma$ plane are evaluated. In the dissertation I proved that it is sufficient to deal with a bounded region of the plane because points outside this region do not yield further solutions. The shape of this region can be seen in Figure 3.

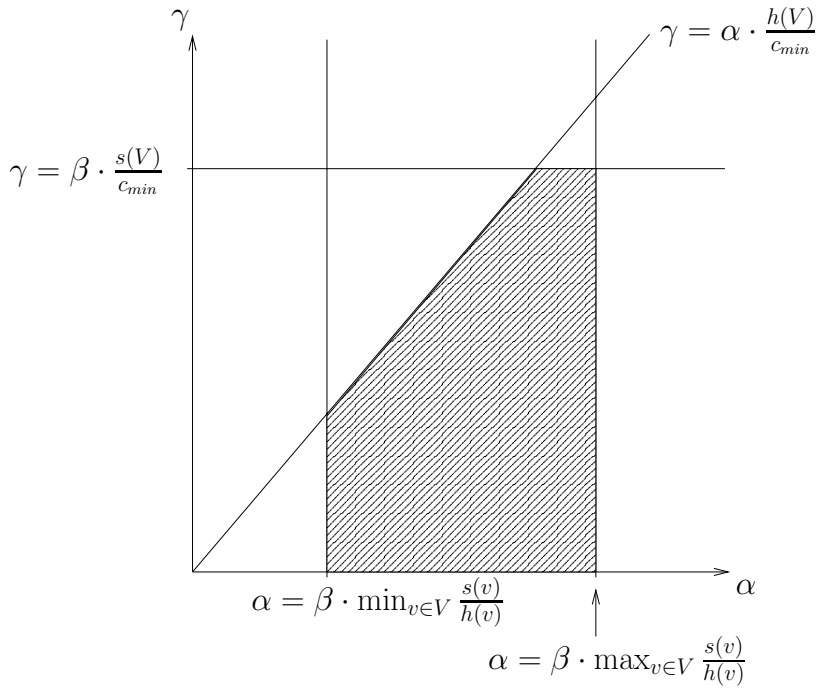

Figure 3: The region to be scanned

Inside this region, the algorithm proceeds with increments of size $d\alpha$ and $d\gamma$. In order to make the algorithm faster, $d\alpha$ and $d\gamma$ are set adaptively during the course of the algorithm: as long as no better solution is found, the size of the increments is multiplied by $(1 + \varepsilon)$ in each step, i.e., the search accelerates exponentially. However, when a better solution is found, the increment is reset and the search becomes thorough again.

This way, the best point of the plane can be found approximately, but it is possible that the algorithm jumps over the best point itself. This is corrected in a second phase, in which the neighbourhood of the previously found best point is searched again, this time more thoroughly. Of course, this method works well only if the costs of the optimal partition of **P5**, as functions of α and γ, change smoothly and according to some simple structure. Luckily, this is really the case, as can be seen in Figure 4. The figure also shows that the hardware cost of the optimal partition is monotonously decreasing in α. This is also proven formally in the dissertation.

5 Comparison of the results of Thesis 2, 3, and 4

I compared the three algorithms both analytically, based on their theoretical features, and empirically using industrial benchmark problems. The analytical comparison focused on the following aspects:

- The role and importance of initial partitions, and how they are generated

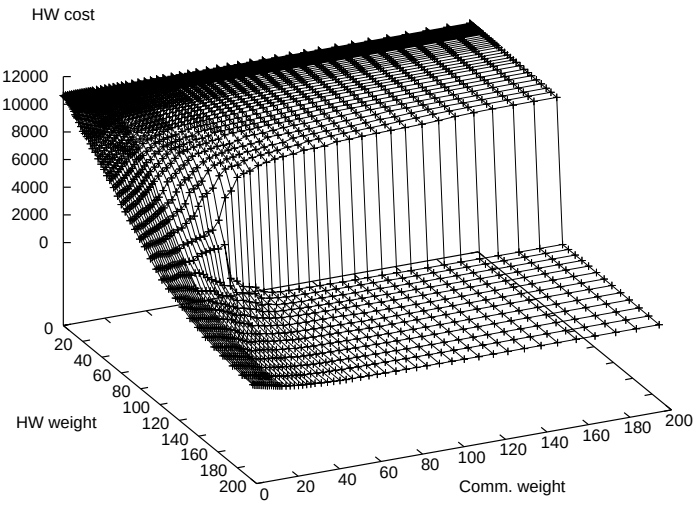

Figure 4: Hardware cost of the optimal partition of the **P5** problem as the function of the weights α and γ

- Stopping criteria
- Parallelization possibilities
- Stability, i.e., the ability of not forgetting promising areas of the search space
- Extension possibilities
- Parameters that need to be tuned, hardness of the tuning process
- Number of evaluated partitions per unit time and the complexity of generating the next partition to evaluate
- The way that the relative importance of the different cost functions is captured
- Determinism versus randomization

The empirical tests were carried out on a number of benchmark problems: members of the MiBench benchmark suite [12], the own designs of our research group, as well as large random graphs.

The tests revealed that the effectiveness of the algorithms depends significantly—beside on the size of the input—on how strict the constraint is, as well as on the communication to computation ratio (CCR), i.e., the ratio of communication costs to software costs. The typical speed versus result quality of the three algorithms for different constraints and CCR

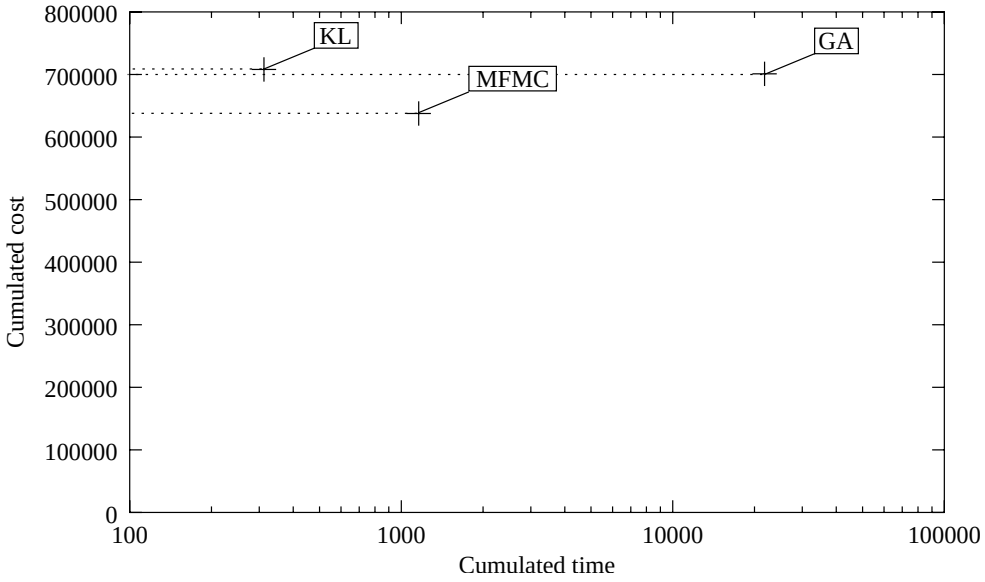

Figure 5: Cost/speed trade-off of the three algorithms for low CCR and tight constraint

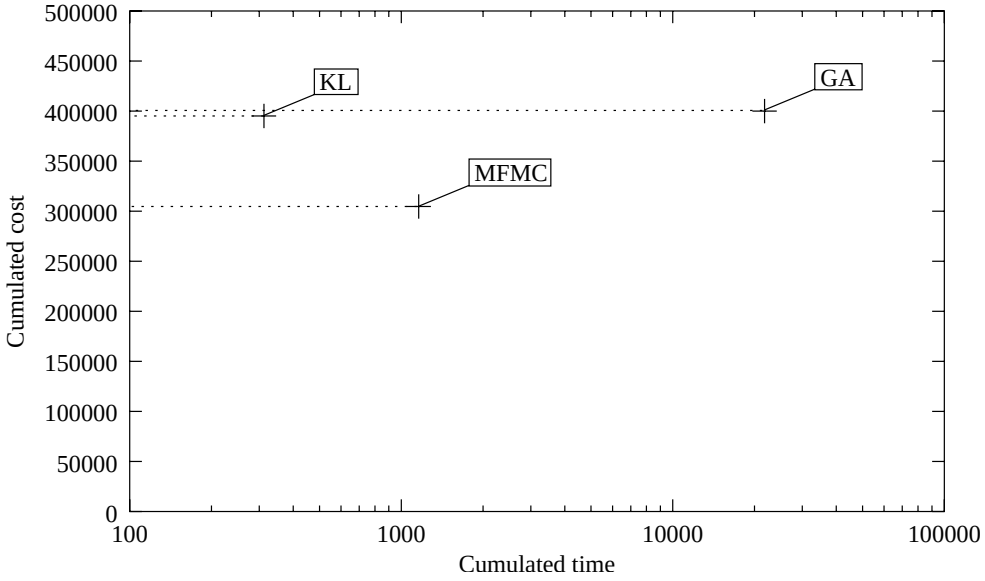

Figure 6: Cost/speed trade-off of the three algorithms for low CCR and loose constraint

can be seen in Figures 5-8. In these experiments, 'tight constraint' meant a bound chosen randomly from the lower half of the interval of meaningful bound values, whereas a 'loose constraint' meant a bound chosen randomly from the upper half. In the case of 'low CCR,' the average of the communication costs was roughly equal to the maximum software cost, in the case of 'high CCR,' it was equal to about ten times the maximum software cost.

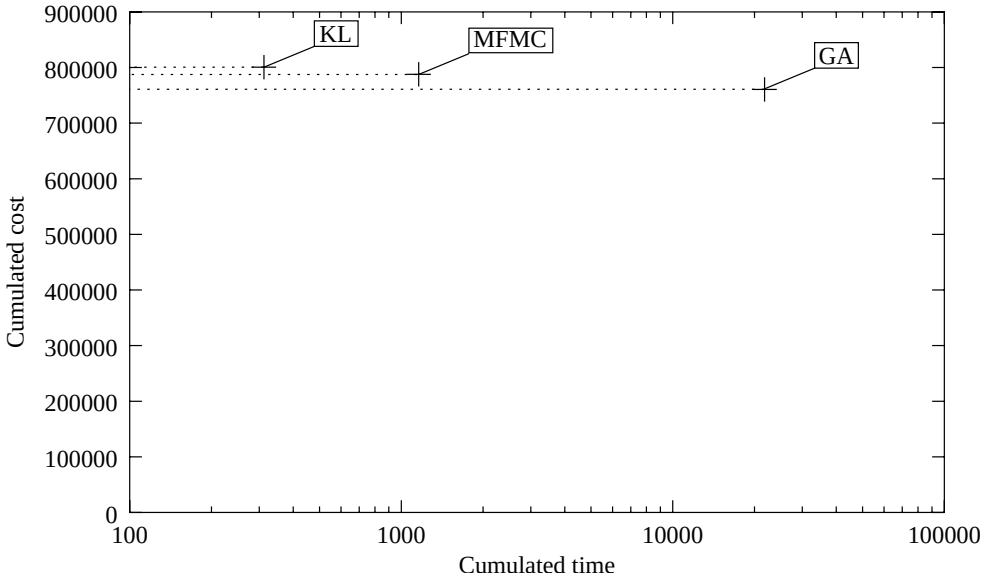

Figure 7: Cost/speed trade-off of the three algorithms for high CCR and tight constraint

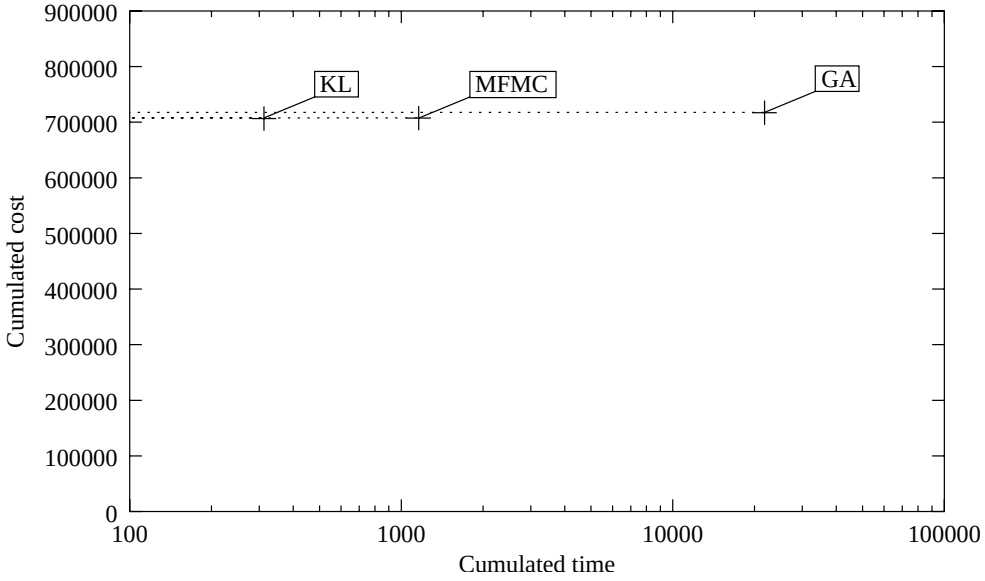

Figure 8: Cost/speed trade-off of the three algorithms for high CCR and loose constraint

As can be seen from the figures, the best results were achieved in the case of low CCR (regardless of the tightness of the constraint) by the MFMC-based algorithm of Thesis 4. In the case of high CCR and loose constraint, there is no clear winner, and in the case of high CCR and tight constraint, the GA achieved the best result. Concerning running time, the winner was in all cases the KL-type algorithm, the second was the MFMC-based

algorithm, and the third the GA.

The tests also revealed that, in most cases, the lower bounds produced by the MFMC-based algorithm are not far from the results it found, which proves the high quality of both the results and the lower bounds.

6 Applications

The results of Thesis 1 found indirect application: the polynomial-time exact algorithm for the **P5** problem became the basis for the heuristic algorithm of Thesis 4; the $\mathcal{NP}$-hardness results helped to better understand the problem and lead my attention to the presented heuristics.

The heuristic partitioning algorithms of Theses 2-4 have already found practical applications:

- In an *exact* partitioning algorithm based on branch-and-bound (BB), developed by our research group [J6]. The BB algorithm is able to optimally solve partitioning problems of several hundred nodes in reasonable time. It also significantly outperforms similar exact algorithms based on integer linear programming. In BB, both the genetic algorithm of Thesis 2 and the MFMC-based algorithm of Thesis 4 have been applied as pre- optimization heuristics: they generate initial solutions and thus *initial upper bounds* for the branch-and-bound algorithm. As demonstrated in [J6], both of these algorithms are very useful for decreasing the running time of the BB algorithm without sacrificing optimality.

  Moreover, the polynomial-time algorithm of Thesis 1 also plays an important role in the BB algorithm: it is used to determine whether a given partial solution (i.e., when it has not yet been decided for all nodes whether they should be in hardware or in software) can be extended to a valid complete solution with lower cost than the best one found so far. Thus, the polynomial-time algorithm of Thesis 1 significantly helps in cutting off unpromising branches of the search tree and so improving the performance of the BB algorithm.

- In a project funded by Akita Industrial Development Center (Japan), our research group participated in the development of some medical applications. Our task was to create efficient implementations for several neural algorithms using hardware/software co-design. The neural algorithms included Kohonen's self-organizing map, an image segmentation algorithm, and a clustering algorithm based on fuzzy logic. The most important step in the implementation of these algorithms was to find a good trade-off between the conflicting requirements on chip size, cost, and performance [C6]. In the first phase of the project, this was done manually, but as later the presented partitioning algorithms became ready, this work was substantially eased.

- In the EasyComp project, funded by the European Union, our research group participated in the development of new component-based system design methodologies. In particular, we developed a component-based process for hardware/software codesign [C8, J2]. The main idea of this methodology is to compose complex systems out of abstract building blocks in an intuitive way using a graphical tool. In order to cope with design complexity, implementation-specific details of the components are hidden from the designer. In particular, the components may have more than one implementation, for instance a hardware implementation and a software implementation; our tool shields the designer from such low-level issues. As a consequence, partitioning decisions have to be made automatically. This is where the heuristic partitioning algorithms presented in this work are used.

I would like to thank the following organizations for their financial support: OTKA (T030178, T043329, T042559), European Union (IST 1999-14191), Akita Industrial Development Center, Timber Hill LLC.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, 1993.

[2] M. de Berg, O. Schwarzkopf, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2nd edition, 2000.

[3] N. N. Binh, M. Imai, A. Shiomi, and N. Hikichi. A hardware/software partitioning algorithm for designing pipelined ASIPs with least gate counts. In *Proceedings of the 33rd Design Automation Conference*, 1996.

[4] Th. H. Cormen, Ch. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, 2nd edition, 2001.

[5] A. Dasdan and C. Aykanat. Two novel multiway circuit partitioning algorithms using relaxed locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(2):169–177, February 1997.

[6] L. Davis. *Handbook of genetic algorithms*. Van Nostran Reinhold, 1991.

[7] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli. Hardware/software partitioning of VHDL system specifications. In *Proceedings of EURO-DAC '96*, 1996.

[8] R. Ernst, J. Henkel, and T. Benner. Hardware/software cosynthesis for microcontrollers. *IEEE Design and Test of Computers*, 10(4):64–75, 1993.

[9] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, 1982.

[10] R. K. Gupta. *Co-Synthesis of Hardware and Software for Digital Embedded Systems.* PhD thesis, Stanford University, December 1993.

[11] R. K. Gupta and G. de Micheli. Hardware-software cosynthesis for digital systems. *IEEE Design & Test of Computers*, 10(3):29–41, 1993.

[12] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the IEEE 4th Annual Workshop on Workload Characterization*, 1997.

[13] L. Hagen, J. H. Huang, and A. B. Kahng. On implementation choices for iterative improvement partitioning algorithms. *IEEE Transactions on CAD*, 16(10):1199–1205, 1997.

[14] A. G. Hoffmann. The dynamic locking heuristic – a new graph partitioning algorithm. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 173–176, 1994.

[15] E. Hwang, F. Vahid, and Y. C. Hsu. FSMD functional partitioning for low power. In *Proceedings of the Design Automation and Test in Europe Conference*, 1999.

[16] A. Kalavade. *System-level codesign of mixed hardware-software systems.* PhD thesis, University of California, Berkeley, CA, 1995.

[17] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

[18] B. Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Transactions on Computers*, 33(5):438–446, 1984.

[19] J. Madsen, J. Grode, P. V. Knudsen, M. E. Petersen, and A. Haxthausen. LYCOS: The Lyngby co-synthesis system. *Design Automation of Embedded Systems*, 2(2):195–236, 1997.

[20] R. Niemann. *Hardware/Software Co-Design for Data Flow Dominated Embedded Systems.* Kluwer Academic Publishers, 1998.

[21] L. A. Sanchis. Multiple-way network partitioning. *IEEE Transactions on Computers*, 38(1):62–81, 1989.

[22] F. Vahid and D. Gajski. Clustering for improved system-level functional partitioning. In *Proceedings of the 8th International Symposium on System Synthesis*, 1995.

[23] W. Wolf. An architectural co-synthesis algorithm for distributed embedded computing systems. *IEEE Transactions on VLSI Systems*, 5(2):218–229, June 1997.

[24] W. Wolf. A decade of hardware/software codesign. *IEEE Computer*, 36(4):38–43, 2003.

[25] C.-W. Yeh, C.-K. Cheng, and T.-T. Y. Lin. A general purpose, multiple-way partitioning algorithm. *IEEE Transactions on CAD*, 13(12):1480–1487, 1994.

Publications

Journal publications published abroad

[J1] Z. Á. Mann and K. Kondorosi. Tracing system-level communication in distributed systems. *Software: Practice & Experience*, 34:727–755, 2004.

[J2] P. Arató, Z. Á. Mann, and A. Orbán. Extending component-based design with hardware components. *Elsevier Journal of Science of Computer Programming, Special Issue on New Software Composition Concepts*, accepted.

[J3] P. Arató, Z. Á. Mann, and A. Orbán. Algorithmic aspects of hardware/software partitioning. *ACM Transactions on Design Automation of Electronic Systems*, accepted.

[J4] P. Arató, Z. Á. Mann, and A. Orbán. Time-constrained scheduling of large pipelined datapaths. *Elsevier Journal of Systems Architecture*, submitted.

[J5] G. Ziegler, Z. Á. Mann, A. Orbán, Zs. Palotai, L. Grad, and A. Lőrincz. Three-Level Memory for Optimization. *Journal of Applied Soft Computing*, submitted.

[J6] P. Arató, Z. Á. Mann, and A. Orbán. Finding optimal hardware/software partitions. *IEEE Transactions on Computers*, submitted.

Journal publications published in Hungary, in foreign languages

[J7] I. Zs. Berta and Z. Á. Mann. Smart cards – present and future. *Híradástechnika, Journal on C5*, 2000/12, 24-29.

[J8] I. Zs. Berta and Z. Á. Mann. Implementing elliptic curve cryptography on PC and smart card. *Periodica Polytechnica, Series Electrical Engineering*, 46(1-2):47-73, 2002.

[J9] A. Orbán, Z. Á. Mann, and P. Arató. Time-constrained design of pipelined control-intensive systems. *Periodica Polytechnica, Series Electrical Engineering*, accepted.

Journal publications in Hungarian

[J10] Berta I. Zs. és Mann Z. Á. Programozható chipkártyák és azok biztonsága. *Magyar Távközlés*, 2000/4, 8-13.

Articles in edited books

[C1] P. Arató, Z. Á. Mann, and A. Orbán. Genetic scheduling algorithm for high-level synthesis. *Intelligent Systems at the Service of Mankind, Volume 1, Selected publications of INES 2002 Conference*, Editors: Wilfried Elmenreich, J. Tenreiro Machado, Imre J. Rudas, 311-322, 2003. (Originally: Proceedings of the IEEE 6th International Conference on Intelligent Engineering Systems, 2002.)

[C2] B. Golda, B. Laczay, Z. Á. Mann, Cs. Megyeri, and A. Recski. Implementation of VLSI routing algorithms. *Intelligent Systems at the Service of Mankind, Volume 1, Selected publications of INES 2002 Conference*, Editors: Wilfried Elmenreich, J. Tenreiro Machado, Imre J. Rudas, 349-360, 2003. (Originally: Proceedings of the IEEE 6th International Conference on Intelligent Engineering Systems, 2002.)

International conference publications

[C3] Z. Á. Mann and A. Orbán. Integrating formal, soft and diagrammatic approaches in high-level synthesis and hardware-software co-design. In *Proceedings of Informatik 2001, Workshop on Integrating Diagrammatic and Formal Specification Techniques*, volume I, 649–654, 2001.

[C4] Z. Á. Mann, J. Calmet, and P. Kullmann. Testing access to external information sources in a mediator environment. *Proceedings of the 14th IFIP International Conference on Testing Communicating Systems*, Kluwer Academic Publishers, 111-126, 2002.

[C5] Z. Á. Mann and A. Orbán. Optimization problems in system-level synthesis. In *Proceedings of the 3rd Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*, 222-231, 2003.

[C6] P. Arató, Z. Á. Mann, and A. Orbán. Hardware-software co-design for Kohonen's self-organizing map. In *Proceedings of the IEEE 7th International Conference on Intelligent Engineering Systems*, 173-178, 2003.

[C7] P. Arató, S. Juhász, Z. Á. Mann, A. Orbán, and D. Papp. Hardware/software partitioning in embedded system design. In *Proceedings of the IEEE International Symposium on Intelligent Signal Processing*, IEEE Press, 197-202, 2003.

[C8] P. Arató, Z. Á. Mann, and A. Orbán. Component-based hardware-software co-design. In *Proceedings of the 17th International Conference on Architecture of Computing Systems, Lecture Notes in Computer Science (LNCS 2981)*. Springer, 169-183, 2004.

Conference publications in Hungarian

[C9] Berta I. Zs. és Mann Z. Á. Programozható chipkártyák kriptográfiai alkalmazása. *Networkshop*, 2001.

University lecture notes

[U1] Z. Á. Mann, A. Orbán, A. Recski. Aufgaben zur theoretischen Informatik. *Számítástudományi és Információelméleti Tanszék, Budapesti Műszaki és Gazdaságtudományi Egyetem*, 85 o., 2001.

Unqualified works

Students' scientific conference

[O1] Berta I. Zs. és Mann Z. Á. A hitelesség biztosításának lehetőségei intelligens smartcard segítségével. *Tudományos diákköri dolgozat* (TDK 1999: 1. díj, OTDK 2001: 2. díj), 48 o., 1999.

[O2] Berta I. Zs. és Mann Z. Á. Elliptikus görbéken alapuló nyilvános kulcsú kriptográfia elemzése chipkártyás és PC-s környezetben. *Tudományos diákköri dolgozat* (TDK 2000: 1. díj, OTDK 2001: különdíj), 41 o., 2000.

[O3] Mann Z. Á. CORBA alkalmazások nyomkövetése interceptor-ok segítségével. *Tudományos diákköri dolgozat* (TDK 2000: 1. díj, OTDK 2001: 3. díj), 43 o., 2000.

[O4] Mann Z. Á. és Orbán A. Új ütemező algoritmusok a magas szintű szintézisben. *Tudományos diákköri dolgozat* (TDK 2000: 1. díj, OTDK 2001: 1. díj), 56 o., 2000.

Posters at international conferences

[O5] Z. Á. Mann, A. Orbán, and P. Arató. Component-based Hardware/Software Co-Design. *Microsoft Research Academic Conference*, 2003.

[O6] Z. Á. Mann and A. Orbán. Hardware/software partitioning. *Design Automation and Test in Europe*, PhD Forum, 2004.

Talks at conferences without proceedings

[O7] Z. Á. Mann. Tracing CORBA applications using interceptors. *OMG Information Day*, 2001.