

Run-Time Mapping of Applications to a Heterogeneous Reconfigurable Tiled System on Chip Architecture

Lodewijk T. Smit, Gerard J.M. Smit, Johann L. Hurink
Hajo Broersma, Daniel Paulusma and Pascal T. Wolkotte
Department of Electrical Engineering, Mathematics & Computer Science,
University of Twente, Enschede, The Netherlands,
email:L.T.Smit@utwente.nl

Abstract

This paper evaluates an algorithm that maps a number of communicating processes to a heterogeneous tiled System on Chip (SoC) architecture at run-time. The mapping algorithm minimizes the total amount of energy consumption, while still providing an adequate Quality of Service (QoS). A realistic example is mapped using this algorithm.

1 Introduction

The architecture of a portable multimedia system has to meet many conflicting requirements. For example, it has to be energy-efficient, due to the scarce energy resources and it has to be flexible. It should be flexible so that it a) can employ a lot of different standards, b) can adapt quickly to a new standard c) can run different sets of tasks concurrently and d) can adapt to the dynamically changing environment.

The designer can choose from a wide spectrum of architectures to implement such a system. This can vary from energy-efficient, high-performance but static and inflexible ASICs to flexible and easy programmable but energy hungry general purpose processors. The optimal choice depends on the application/algorithms and several other aspects, including the available energy budget, the time to market and the production volume.

No specific architecture can meet all these requirements perfectly. A heterogeneous System on Chip (SoC) with different kind of (reconfigurable) processing tiles interconnected by a Network on Chip (NoC) provides an attractive solution for this dilemma.

The best of both worlds (energy-efficient and flexible) are combined in such a heterogeneous architecture. For example, small computational intensive algorithms of an application can be

mapped to an ASIC or a coarse-grain reconfigurable tile avoiding a power hungry tile such as a general purpose processor. On the other hand, control intensive but less computational intensive parts of the application can be mapped better to a general purpose processor. In this way, the architecture can match the application instead of the other way around, as usual.

However, the use of such a heterogeneous tiled SoC architecture changes the standard development flow (e.g. code a program in C and compile or code functionality in VHDL and synthesize). The designer has to partition the application into a graph with communicating functional processes. In a process graph, a vertex represents a functional process and a directed edge represents communication between functional processes. For each functional process one or more realization for one or more different types of processing tiles have to be made. Designing more, functional equivalent, realizations of the same process for different types of tiles allows running an application even when the optimal tiles are not available. Often, the partitioning of an application into a process graph arises naturally from the application (see Section 4). Quite often, the designer knows which kind of partitionings makes sense. The designer plays an important role in this process and we assume that the partitioning and the choice of possible realizations are still made manually by the designer.

The mapping of these realizations to the heterogeneous tiled SoC architecture can best be done automatically at run-time. At design time it is unknown which applications run simultaneously and how the external environment (with regard to available services, end-user behavior, wireless link quality) behaves. Therefore, this mapping decision has to be made at run-time. This article evaluates such a run-time mapping algorithm.

2 Related Work

In the area of scheduling and optimization theory (operations research) a lot of literature exists on models which have some similarities with the considered problem (see e.g. [5, 2]). However, our application has some properties, which does not allow us the use of existing approaches without modification. Compared to traditional scheduling for parallel systems we have the following differences:

- The use of a heterogeneous architecture instead of a homogeneous architecture.
- The most important optimization parameter is minimization of the energy consumption instead of performance. The goal of most scheduling methods is to optimize for performance. In our method, the required performance is only one of the constraints, which has to be satisfied.
- The communication is an important parameter to be included in the total optimization process because the communication consumes a substantial part of the total energy budget. In conventional multiprocessor systems, the main focus is on computation costs.

Another important difference with regard to optimization in literature is that we need a lightweight algorithm. It may be better to have a reasonable good solution computed with little effort than to have an optimal solution that requires a lot of effort. Therefore, on beforehand a lot of existing optimization algorithms are not useful for us.

3 MinWeight Algorithm

In [1] the MinWeight algorithm is described that determines the weight of a minimum processor assignment for *any* weighted process graph and a set of processors. Its running time is exponential. However, in practice it can compute solutions quite fast, as long as the input graphs have only a small number of vertices with a high degree (greater than two). A proof of the correctness of the algorithm, the complexity of the algorithm and a further explanation can be found in [1].

3.1 Properties of the MinWeight Algorithm

In this part we describe and discuss the most relevant strong and weak points of the MinWeight algorithm with respect to our specific mapping problem.

Firstly, the MinWeight algorithm computes an optimal solution (see [1]) to the mapping problem

instead of an approximation. This is a strong advantage of the algorithm.

Secondly, due to the dynamic programming like approach for vertices with low degrees, the complexity is low. The exact complexity depends on the degree of the vertices in the graphs, see [1]. E.g. for the mapping of 10 processes to 16 possible processors, $16^{10} \approx 10^{12}$ solutions are possible, but the algorithm finishes within a few milliseconds. When the degree of the vertices increases, the computation time of the algorithm increases exponentially. However, the process graphs are relatively small (between 5 and 20 vertices) in our targeted application domain and in practice a process graph does not have a lot of vertices with degree ≥ 3 . Therefore, we do not expect that the computation time will be a problem in practice.

Unfortunately, the algorithm does not take possible constraints into account. E.g. the capacity of processors and communication links are assumed to be infinite and the delay in communications can not be taken into account. This is a serious limitation of the MinWeight algorithm.

3.2 Adding the Processor Capacity Constraint

The MinWeight algorithm does not handle additional constraints, e.g. the constraint that a processor has a limited capacity and therefore only a limited fixed number of processes can run on a processor. Or even more advanced, it has to determine the number of processes that can run on a specific processor depending on the capacity of the processor in combination with the load for the execution of the processes. To cope with the limited capacity of a processor, we adapted the MinWeight algorithm such that it satisfies the constraint that at most one process is mapped to each processor. A similar approach can be used for other constraints, e.g. at most two processes may be mapped to one processor. It is implemented in such a way that before computing the weight of a particular solution two conditions are checked. First, it checks whether the processors involved in the mapping solution are not already occupied in an earlier mapping step for another vertex of a processor graph. Second, it checks whether the processors involved in the mapping solution are all unique. Only if both conditions are satisfied, the solution is feasible.

3.3 Improvement of the Adapted MinWeight Algorithm

The adapted MinWeight algorithm (as discussed in Section 3.2) suffers from two problems:

1. The algorithm does not find a mapping at

all.

Different processes can compete for the same resources and it may happen that all the resources for a specific process are already occupied due to mapping decisions in the past. In this case, it is not possible for the algorithm to find a solution.

- The algorithm finds a mapping that is far from optimal. This is a result of the introduced dependencies between the different assignment steps.

The first problem is the most severe one. To reduce the chance of getting no feasible solution we may improve the MinWeight algorithm by changing the order of the assignment of the vertices to the processors. If a vertex needs scarce resources, the probability is high that these resources are already taken by other processes when this vertex is mapped as one of the latest. Therefore, it is probably better to start with mapping of vertices that need scarce resources to avoid resource bottlenecks to avoid ending up with the result that the algorithm is not able to map the process graph to the SoC architecture.

When there are no (longer) resource problems, processes that have high processing costs are mapped next, because the quality of a solution is worse when a process with high processing costs is mapped inefficiently compared to when a process with low processing costs is mapped inefficiently. Therefore, we propose an ordering of the vertices that is based on 1) the scarcity of the resources and 2) the processing costs of the processes.

However, it is not so simple to estimate when the resource scarcity is no longer a threat. It is important to detect as soon as possible that a reordering based on the processing costs of the processes is possible because this improves the optimality of the final solution. Currently, we are investigating which simple metric we may use to decide how to order the processes to obtain a possible near optimal mapping.

For the NoC in general we do not expect resource problems. Most tiled SoC architectures use a mesh structure for the NoC, which means that there are several different routes possible between two processing tiles with equal length.

4 Example

Digitale Radio Mondiale (DRM) [3] is a standard for digital radio below the 30 Mhz. A concise explanation of the DRM standard can be found in [4]. This section describes the mapping of a part of a DRM receiver to a heterogeneous tiled SoC architecture with 16 processing tiles as shown below:

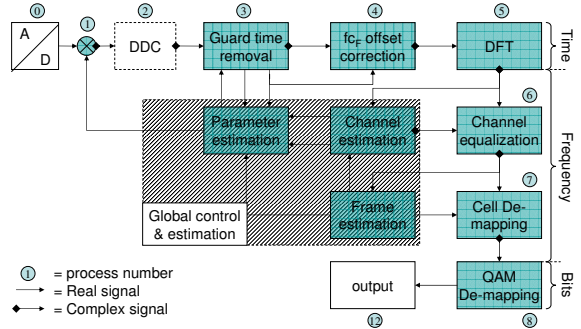


Figure 1. DRM Processes to Map on SoC

GPP (0)	DSRH (1)	DSP (2)	DSP (3)
DSP (4)	ASIC (5)	DSRH (6)	DSRH (7)
DSP (8)	ASIC (9)	ASIC (10)	DSRH (11)
FPGA (12)	DSRH (13)	DSP (14)	GPP (15)

Figure 1 shows the processes of the digital baseband part of our DRM receiver. Table 1 shows the processes that we would like to map on the SoC (for a functional description of the processes see [4]). These processes concern the data flow of the DRM application; we do not consider the processes in the "Global control & estimation" part of Figure 1. To test our algorithms, it is not important to have very accurate estimations of the processing costs. Therefore, we make a few assumptions to test our algorithms so that we do not have to realize the system to get the exact numbers:

- the number of multiplications per second is used as an indication for the costs of a process. Table 1 shows of the costs of the process in terms of multiplications per second for reception of Mode B, and the available implementations for the different type of processors.
- the ratio between processing an multiplication on an ASIC, DSRH, FPGA, DSP, GPP are 10:40:50:60:500 respectively.
- the communication costs increase linearly with the distance of the communication path on the SoC. The communication costs are equal to the throughput in kbit/s given in Table 2 multiplied by the Manhattan distance between the tiles.

Note that processes that have an ASIC realization need a specific ASIC. It is not possible to assign a process with an ASIC realization to an arbitrary ASIC processor.

4.1 Results

Table 3 shows the solutions (the assignments and the total costs) of the different algorithms. In each mapping vector, index i (starting at zero)

gives processor number of the mapping of the i th process. So, e.g. for all mappings, process 3 is assigned to processor 12.

The optimal mapping without constraints is given by the MinWeight algorithm. Note that the MinWeight algorithm maps different processes to the same DSRH tiles (6 and 13). If we assume that a tile may be used for at most one process there is a resource problem. Even by swapping some of these processes to other tiles of the same type, no feasible solution can be obtained, since 5 processes are assigned to the DSRH tiles, but only 4 instances of this type of tile are available.

Taking into account that every processor may execute at most 1 process, another mapping is determined by the adapted MinWeight algorithm of Section 3.2. Note that the initial processor mappings are the same and that the first difference occurs when tile 13 is used a second time. This gives a mapping that is 8% more expensive compared to the solution of the MinWeight algorithm. The remaining question is how much the solution of the adapted MinWeight differs from the optimal solution, which is expected to be higher than the lower bound given by the MinWeight algorithm due to the additional constraint. Therefore, the optimal solution is determined using a quadratic programming solution. It took several hours of computation on a Pentium 4 processor to evaluate all the possibilities with the brute force enumeration. This solution is about 3% more expensive than the MinWeight solution due to the additional processor capacity constraint. Therefore, we can conclude that we lose about 5% performance due to non optimality for the adapted MinWeight algorithm.

5 Conclusion

The MinWeight algorithm computes an optimal solution very fast. However, the algorithm does not take into account all relevant constraints and therefore the practical use of the algorithm is limited. Adaptation of the MinWeight algo-

Block	No.	Mul.	Processors
A/D converter	0	0	ASIC
Mixer	1	24k	DSP, DSRH, GPP
DDC	2	0	ASIC
Guard time corr.	3	144k	DSP, FPGA, GPP
Frequency Corr.	4	96k	DSP, DSRH, GPP
FFT	5	346k	ASIC, DSP, DSRH, GPP
Channel equal.	6	38k	GPP, DSP, DSRH
Demapping	7	0	GPP, DSP, DSRH
Bit decoding	8	0	GPP, DSP, DSRH
Output	12	0	GPP

Table 1. Multiplic. Costs for DRM Mode B

Edge	kbit/s	Edge	kbit/s	Edge	kbit/s
0 → 1	375k	3 → 4	600k	6 → 7	241k
1 → 2	750k	4 → 5	600k	7 → 8	201k
2 → 3	755k	5 → 6	300k	8 → 12	47k

Table 2. Comm. Costs for DRM Mode B

algorithm	mapping	costs
MinWeight	5, 13, 9, 12, 13, 10, 6, 6, 6, 0	22231
Adapted MinW.	5, 13, 9, 12, 6, 10, 7, 3, 2, 0	24126
Quadratic progr.	5, 1, 9, 12, 13, 10, 6, 7, 11, 15	22954

Table 3. Different Mappings

gorithm in order to fulfill the additional constraints gives a method which leads to a non-optimal solution. A realistic case shows that the adapted MinWeight algorithm gives a near optimal solution in a reasonable short computation time.

In future, we focus on three issues. First, additional constraints and heuristics will be added to the MinWeight algorithm to cope with more real life restrictions and to improve the solutions respectively. Second, we expect that adding heuristics to change the order in which the processes are mapped to processors improves the optimality of the solution. We are currently investigating how to determine a better ordering based on simple criteria. Third, another approach may be used so that in the first step an optimal solution is computed using the MinWeight algorithm and in the second step the constraint violations are solved.

Acknowledgement

This research is conducted within the FP6 Smart ChipS for Smart Surroundings (4S) project supported by the European Commission.

6. References

- [1] H. J. Broersma, D. Paulusma, G. J. M. Smit, F. Vlaardingerbroek, and G. J. Woeginger. The computational complexity of the minimum weight processor assignment problem. In *Proceedings of 30th international workshop on graph-theoretic concepts in computer science (WG2004)*, pages 271–282, June 2004.
- [2] P. Brucker. *Scheduling Algorithms*. Springer, 3 edition, 2001. ISBN:3-540-41510-6.
- [3] E. T. S. I. (ETSI). Digital radio mondial (drm); system specification, Apr. 2003. ETSI ES 201 980 v1.2.2 (2003-04).
- [4] F. Hofmann, C. Hansen, and W. Schäfer. Digital radio mondiale (drm) digital sound broadcasting in the AM bands. *IEEE Transactions on Broadcasting*, 49(3):319–328, Sept. 2003.
- [5] W. L. Winston. *Operations Research; Applications and Algorithms*. International Thomson Publishing, 3 edition, 1993. ISBN: 0-534-20971-8.