

Mapping and Performance Evaluation for Heterogeneous MP-SoCs via Packing

Bastian Ristau and Gerhard Fettweis

TU Dresden, Vodafone Chair Mobile Communications Systems
01062 Dresden, Germany
{ristau, fettweis}@ifn.et.tu-dresden.de

Abstract. The computational demand of signal processing algorithms is rising continuously. Heterogeneous embedded multiprocessor systems-on-chips are one solution to tackle this demand. But to be able to take advantage of the benefits of these systems, new strategies are required how to map applications to such a system and how to evaluate the system's performance at a very early design stage. We will present a static, analytical, bottom-up methodology for temporal and spatial mapping of applications to MP-SoCs based on packing. Furthermore we will demonstrate how the result can be used for performance evaluation and system improvement without the need for simulations.

1 Introduction

The computational demand of signal processing algorithms is rising continuously. Heterogeneous embedded systems-on-chip are one solution to tackle this demand. Though ASIC centered single chip solutions are usually smaller and more energy efficient [1], flexibility and reusability of MP-SoC components is comparatively higher. This is important especially in signal processing, since it enables to react to future changes in signal processing algorithms or even to implement new algorithms without changing the hardware. But to take advantage of this flexibility new strategies are required to determine how to map applications to such a system and to evaluate the system's performance at a very early design stage.

We will present a static, analytical, bottom-up methodology for mapping applications spatial and temporal to a given architecture following the Y-Chart approach [2]. The basic idea of the Y-Chart approach is to model applications and architecture separately, to perform a mapping of application to architecture and to iterate over different mappings, architectures and application descriptions until the desired architecture and mapping is found. In our proposed methodology the iterations over different mappings is done automatically. Furthermore the resulting static mapping can be utilized for performance evaluation and system improvement without the need for simulations. For determining an initial system there already exist works (e.g. [3, 4]) using linear or multi-objective optimization (MOO). These can be used easily in interaction with our method.

In this paper we will concentrate on mapping for minimal execution time, although the methodology is not restricted to this. In a lot of signal processing applications we

have to face semi-hard real-time constraints. Thus, it is important to know in the first place, if the chosen system can meet the given timing constraints. If not, the system has to be modified. We think of this of a starting point for further optimizations, if the timing requirements are met. This can be done by modifying some constraints and the objective in our methodology, but also by applying other approaches, e.g. [5–8].

We rejected the use of MOO in the first performance evaluation step, because there are two major downsides. Firstly, MOO produces a set of pareto-optimal solutions, from which the preferred one has to be chosen manually. Secondly, the existence of more than one solution prevents solvers from efficiently making use of branch&bound or equivalent techniques. Thus, relevant details usually have to be neglected in the model to get acceptable solving times. The result of these methods has to be checked with simulations afterwards [3].

2 Methodology

In this section we will give a short overview of the setup of our methodology followed by a more detailed description, how the mapping problem can be regarded as and solved via solving packing problems.

But first we want to give a short description about the methodology we used for modeling applications and architectures. For modeling there is a variety of languages, which can roughly be divided into process-network based and control & data-flow graph (CDFG) based approaches. We have chosen a CDFG based kind of view, because in comparison to process networks the complete parallelism of the application is exposed explicitly. We use YML [9], which was designed to deal with Kahn-Process-Networks derived from Compaaan [10], but is suitable for CDFGs as well. Another advantage of YML is, that it can be used to model the architecture as well in the same way.

As mentioned, our methodology is a single objective, bottom-up, analytical approach. We favored an analytical over a simulation based approach, because in this way each corner-case does not have to be identified and simulated individually, but is considered within the model. However, this – as well as the fact that our methodology is static – limits our approach to applications that can be scheduled statically. The methodology is designed bottom-up due to fact, that implementation and performance figures are available usually at very low abstraction levels – or at least can be estimated more precise. Multiple abstraction levels are included by finding mappings on the lowest abstraction levels, inserting the gained results into higher abstraction levels and in this way reaching the top-level step-by-step.

To find a mapping and evaluate the performance of the given system, we propose the methodology depicted in Fig. 1. First we do a fast optimization to check, if the chosen system can meet the required timing constraints without resource constraints. This problem can be solved exact in polynomial time for example via shortest path algorithms from the field of graph theory. Since the algorithms are well-known, we neglect the details in this paper.

If the timing constraints can be met disregarding resource constraints, we perform a 3-step mapping algorithm, which is described in detail in the following subsections. The three steps can be seen as a stepwise refinement of the mapping. After each step

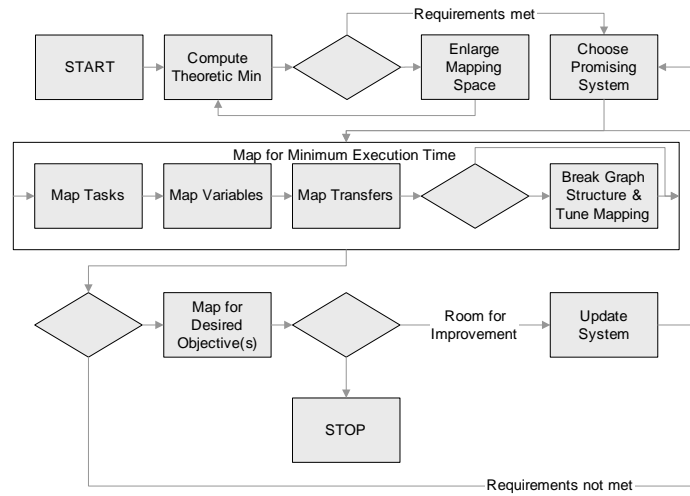


Fig. 1. Block diagram of the presented methodology and the possible steps beyond

the mapping process is stopped, if the timing constraints cannot be met. In the first step the tasks are mapped onto the system. Second, the variables are mapped to suitable memories of the system and memory addresses. This step also determines, if the chosen memory capacity is sufficient. After that transfers are taken care of.

We are aware of the fact, that there have been efforts treating individual phases – just to name [11, 12] as prominent examples. Our approach differs in putting all of these problems down to the class of packing problems. Another advantage esp. in memory allocation is the fact, that not two, but an arbitrary number of memories can be handled simultaneously. In addition, we keep the graph structure of the application throughout the whole mapping process, which can be useful for further optimizations.

After the 3-step mapping algorithm an optional refinement step can be performed, which breaks up the graph structure. If the timing constraints can be met, further optimizations can be executed utilizing the results of the mapping result.

2.1 Mapping Tasks & Transfers

The problem of mapping tasks temporal and spatial can be interpreted as 2-dimensional strip-packing problem [13]. In strip-packing, boxes of fixed length and width have to be arranged into a strip of fixed width in such a manner, that total height is minimized. Figure 2 illustrates the application of packing to mapping tasks.

Applied to mapping tasks the boxes to be packed represent the tasks. The length of these boxes is defined by the execution time of the tasks. The width of the strip is the total execution time and height represents the available processing elements in discrete unified steps. Thus, not the height but the width of the strip w^{\min} is minimized, when optimization of total execution time is desired.

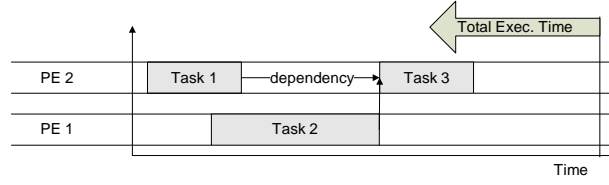


Fig. 2. Application of packing to mapping tasks

Let P_i be the set of processors capable of processing task i and $y_{i,k} := 1$, if task i is mapped to processor k . First, we have to ensure that each task i is mapped onto a processor k capable of processing this task. This is done by (1).

$$\sum_{k \in P_i} y_{i,k} = 1, \quad \sum_{k \notin P_i} y_{i,k} = 0 \quad \forall i \quad (1)$$

Let x_i be the starting time of task i . Since the run-time of task i is dependent on the processor it is mapped to, execution time can be expressed as $\sum_k W_{i,k} y_{i,k}$ (with $W_{i,k} :=$ execution time of task i on processor k). Thus, (2) guarantees that all tasks are finished before total execution time w^{\min} and (3) that precedence constraints are met. Note that (3) is a restriction of valid positions for the boxes in packing terms, which is not given in regular packing problems. This property reduces the solution space and speeds up solution time. The solution space can be reduced even more by considering ALAP and ASAP times for the tasks, which we did not implement yet.

$$x_i + \sum_k W_{i,k} y_{i,k} \leq w^{\min} \quad \forall i \quad (2)$$

$$x_i + \sum_k W_{i,k} y_{i,k} \leq x_j \quad \forall i, j : j \text{ depends on } i \quad (3)$$

Next, we have to obey machine restrictions. Let be i, j tasks and k, l the indices of the assigned processors. Let be $u_{i,j} := 1$, if $k \leq l$, and $b_{i,j} := 1$, if i is executed before task j (0 otherwise). Since DSPs and ASICs usually do not support multi-threading, tasks cannot be executed in parallel on the same processor. Equations (4) – (8) provide this non-overlapping in packing terms and are stated for each interfering task-pair (i, j) with $P_i \cap P_j \neq \emptyset$. In this context two tasks do not interfere, if there exists a path between i and j in the CDFG or if they are belonging to different case-blocks of a switch-case-construct. In the latter case, the tasks may be mapped onto the same processor at the same time, because only one of the tasks will be executed in reality. By including this property into our methodology, each possible branch that can be realized in the CDFG is considered automatically.

$$x_i + \sum_k W'_{i,k} y_{i,k} - W^{\max} + W^{\max} b_{i,j} \leq x_j \quad (4)$$

$$\sum_k k y_{i,k} + 1 - H^{\max} + H^{\max} u_{i,j} \leq \sum_k k y_{j,k} \quad (5)$$

$$b_{j,i} + b_{i,j} \leq 1 \quad (6)$$

$$u_{j,i} + u_{i,j} \leq 1 \quad (7)$$

$$u_{j,i} + u_{i,j} + b_{j,i} + b_{i,j} \geq 1 \quad \forall \text{ interfering } i, j \quad (8)$$

Note, that not run-time but the earliest starting time of the next task on the same processor is relevant to ensure a valid temporal mapping. The earliest starting time for subsequent tasks after the start of task i on processor k is denoted by $W'_{i,k}$. Constants $W^{\max} := \sum_{i,k} W_{i,k}$ and $H^{\max} := |\bigcup_i P_i|$ make sure, that (4) and (5) are redundant with $x_j \geq 0$ and $\sum_k ky_{j,k} \geq 0$ in case of $b_{i,j} = 0$ and $u_{i,j} = 0$, respectively.

Finally we need an objective that minimizes total execution time w^{\min} . The sum term in (9) (with $A :=$ adjacency matrix) additionally minimizes the time between two adjacent tasks and therefore liveness of variables. The objective can be refined by adding additional terms, e.g. trying to put adjacent tasks onto the same processor for reducing needed transfers.

$$W^{\min} + \left(\sum_{i,j} \frac{A_{i,j}(x_j - x_i - \sum_k W_{i,k}y_{i,k})}{W^{\max}} \right) \rightarrow \min \quad (9)$$

For mapping transfers, the same methodology can be applied, because the transfers have to be performed by some components of the system. But since the components involved in these transfers are determined by the memory allocation, this has to be done after the mapping of variables and cannot be integrated into the mapping of tasks.

2.2 Mapping Variables

The results of the task mapping are now used for mapping variables. Since the set of valid memories depends on the processor, the related task is mapped to, the mapping of variables has to take place after the tasks are mapped.

Mapping variables in heterogeneous MP-SoCs is similar to register allocation in compiler construction. A well known solution for this problem is Chaitin's graph coloring algorithm [12]. But in opposite to register allocation there are usually more than the two memories (register and global memory) in MP-SoCs, like local, shared and global memories or vector and scalar memories. Thus not each memory is suitable for storing a variable. But the concept of interference can be utilized and extended.

In our methodology two variables interfere, if they *can* share the same memory and *can* be live simultaneously. Since we keep the graph structure throughout the whole mapping process instead of looking at sequential code, the possibility rather than the fact of being live simultaneously is important.

The problem of mapping variables can now be seen as shelf-packing problem as follows: Each shelf represents a memory k of the system, whose height is defined by the capacity C_k . The variables i to be mapped represent the boxes to be packed into the shelves, whose heights h_i matches the size of the variables.

Let M_i be the set of memories suitable for storing variable i and $x_{i,k} := 1$, if variable i is stored in memory k (0 otherwise). First, we have to select a valid memory (shelf) for each variable (10).

$$\sum_{k \in M_i} x_{i,k} = 1, \quad \sum_{k \notin M_i} x_{i,k} = 0 \quad \forall i \quad (10)$$

Let y_i be the starting address of variable i in the memory. Eq. (11) takes care, that the variable is stored in a valid memory address.

$$y_i + h_i \leq \sum_k C_k x_{i,k} \quad \forall i \quad (11)$$

Let H_k^{offset} be the shelf height of memory k and $H^{\text{max}} \geq \max_k \{H_k^{\text{offset}} + C_k\}$ a constant denoting the height of the rack. To make certain that two variables are not stored in the same memory at the same address, we introduce non-overlapping constraints (12) & (13) for all interfering variables i, j .

$$y_i + h_i + \sum_k H_k^{\text{offset}} x_{i,k} - H^{\text{max}} + H^{\text{max}} u_{j,i} \leq y_j + \sum_k H_k^{\text{offset}} x_{j,k} \quad (12)$$

$$u_{j,i} + u_{i,j} = 1 \quad \forall \text{ interfering } i, j \quad (13)$$

Since each memory k has different access times denoted by W_k , we minimize not only needed memory resources, but also force the allocation of variables into fast memories. This is done by (15), where the required resource amounts for memory k are denoted with y_k^{min} and constraint by (14).

$$y_i + h_i + H^{\text{max}} x_{i,k} - H^{\text{max}} \leq y_k^{\text{mem}} \quad \forall i, k \quad (14)$$

$$\sum_{i,k} W_k x_{i,k} + \sum_k y_k^{\text{mem}} C_k^{-1} \rightarrow \min \quad (15)$$

3 Results

In this section we will apply the proposed method to a case study. For the case study we considered a processor core [14] based on Synchronous Transfer Architecture (STA) as a MP-SoC. The intermediate representations our test bench applications generated by the MOUSE [15] compiler front-end served as sample applications.

STA is the concept of loosely coupled functional units and different memory types connected by an interconnection network. Thus, these functional units can be considered as processors in the MP-SoC. In our case study we selected a STA core with 21 functional units. As an important property for legitimating this architecture for the MP-SoC case study there are some operations that have to be executed on a dedicated functional unit of the architecture and some operations, that can be mapped to an element of a subset of functional units of the STA core. In addition, each of these units has output registers. This characteristic is used to model the behavior of keeping data in local memory as rather than having to write it back to shared or global memory.

The advantage of this abstract example is, that we have on the one hand well-known execution times for the instructions and on the other hand a set of test benches, that suit as complex examples (up to 135 nodes per basic block) to demonstrate the potentials and limits of our methodology.

3.1 Results for Mapping Tasks

After transferring the STA core as well as the applications into our internal YAML-based data structure, we applied the methodology described in Sect. 2 to our test benches. We compared the performance of our methodology with the results from our compiler and the theoretic minimum which is defined as the minimal number of VLIWs required without resource constraints. Figure 3 shows an improvement of about 75% in average compared to the existing MOUSE compiler. But even more important is the comparison with the theoretic optimum without resource constraints on the same kind of architecture. As Fig. 3 shows, we are within a factor of 1.45 of this minimum in average.

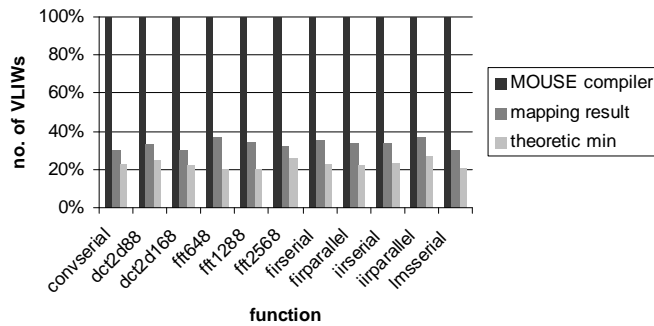


Fig. 3. Comparison of the quality of the task mapping methodology with the MOUSE Compiler and the theoretic min without resource constraints

Solving was done by passing the problem to CPLEX 10.1. Although it was not possible to compute the optimal mapping in acceptable time for applications with a large number of nodes, first solutions were found fast. Therefore we implemented a time limit for finding a better solution of 10 seconds after an improvement of a solution was found. To get an indicator about the quality of these solutions, we ran the CPLEX solver for one block of an application for more than six hours, which showed improvement of about 10% over the solution found after less than a minute.

Figure 4 shows the times required for mapping tasks. We think the denoted running times up to 250 sec. are worth to be spend at an early design stage, since no additional simulations are required. Moreover, note that the solving time can be accelerated by reducing the time specified for finding a better solution during progress.

3.2 Results for Mapping Variables

We applied the shelf-packing methodology for mapping variables to the test benches. As mentioned, the functional units can hold data in their output registers, which is used in the case study for modeling the behavior of keeping data in local memory as rather than having to write it back to shared or global memory. The results of our methodology

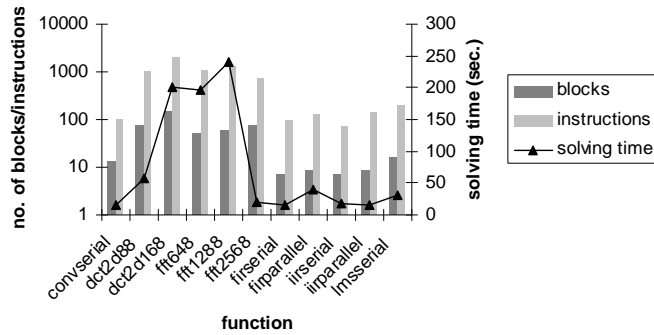


Fig. 4. Number of blocks and instructions to be mapped and the resulting solving times

applied to the test benches showed that this is an important property to be considered in the mapping process. It significantly reduced not only required memory resources to 10% in average but also communication overhead.

3.3 Utilizing Results for Performance Evaluation and System Refinement

As mentioned in Sect. 1, the methodology is designed for performance analysis of an existing MP-SoC in the first place. But the results can be utilized to guide the designer in the process of modifying the system to improve metrics as performance, for example. Since we have a static mapping, we can derive some performance figures like processor loads or memory requirements very easily. Figure 5 shows, that the load of the decoder is very high in comparison to the other functional units.

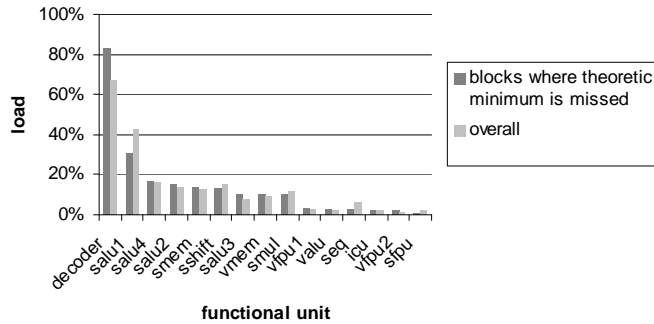


Fig. 5. Load analysis of the task mapping result for all test benches

Thus, we modeled a system with two decoders and did the mapping for this modified system. Figure 6 shows the improvement compared to the original system. With this quite simple analysis of the loads of the functional units and the derived modification of the system we were able to reduce the gap between the theoretic minimum without resource constraints and our mapping result from about 45% to under 20%.

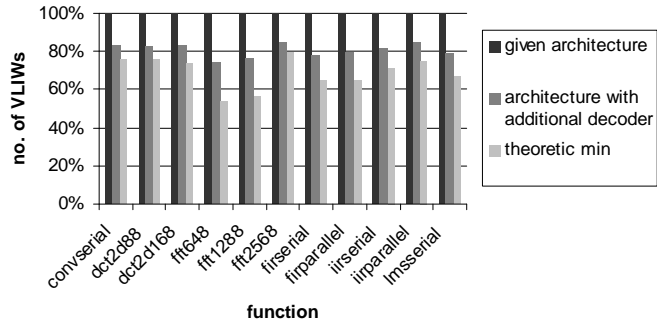


Fig. 6. Improvement gained by adding a second decoder unit to the given architecture as result of the load analysis

4 Conclusion

We have shown that mapping of applications to a given heterogeneous MP-SoC can be regarded as packing problem. These packing problems can be solved efficiently by existing optimization software. Furthermore the results of our static, analytical methodology can be utilized to guide the system designer, how to refine the given system.

5 Acknowledgement

This research is supported by NXP Semiconductors Dresden within the project MxMobile Multi-Standard Mobile Platform of the German Federal Ministry of Education and Research (BMBF).

References

1. Blume, H., Feldkämper, H.T., Noll, T.G.: Model-based exploration of the design space for heterogeneous systems on chip. *J. VLSI Signal Process. Syst.* **40**(1) (2005) 19–34
2. Kienhuis, B., Deprettere, E., Vissers, K., van der Wolf, P.: An approach for quantitative analysis of application-specific dataflow architectures. In: *ASAP '97: Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors.* (1997) 338–349

3. Erbas, C., Erbas, S.C., Pimentel, A.D.: A multiobjective optimization model for exploring multiprocessor mappings of process networks. In: CODES+ISSS '03: Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis. (2003) 182–187
4. Schwiegershausen, M., Pirsch, P.: A formal approach for the optimization of heterogeneous multiprocessors for complex image processing schemes. In: EURO-DAC '95/EURO-VHDL '95: Proceedings of the Conference on European Design Automation. (1995) 8–13
5. Pimentel, A.D., Erbas, C., Polstra, S.: A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Transactions on Computers* **55**(2) (2006) 99–112
6. Bakshi, A., Prasanna, V.K., Ledeczi, A.: MILAN: A model based integrated simulation framework for design of embedded systems. In: LCTES '01: Proceedings of the ACM SIG-PLAN Workshop on Languages, Compilers and Tools for Embedded Systems. (2001) 82–93
7. Govindarajan, R., Gao, G., Desai, P.: Minimizing memory requirements in rate-optimal schedules. In: ASAP '94: Proceedings of the International Conference on Application Specific Array Processors. (1994) 75–86
8. Ristau, B., Fettweis, G.: An optimization methodology for memory allocation and task scheduling in SoCs via linear programming. In: SAMOS '06: Proceedings of the International Workshop on Systems, Architectures, Modeling, and Simulation. (2006) 89–98
9. Coffland, J.E., Pimentel, A.D.: A software framework for efficient system-level performance evaluation of embedded systems. In: SAC '03: Proceedings of the 2003 ACM Symposium on Applied Computing. (2003) 666–671
10. Turjan, A., Kienhuis, B., Deprettere, E.: Translating affine nested-loop programs to process networks. In: CASES '04: Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems. (2004) 220–229
11. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM* **20**(1) (1973) 46–61
12. Chaitin, G.J., Auslander, M.A., Chandra, A.K., Cocke, J., Hopkins, M.E., Markstein, P.W.: Register allocation via coloring. *Computer Languages* **6**(1) (1981) 47–57
13. Belov, G., Chiglintsev, A.V., Filippova, A.S., Mukhacheva, E., Scheithauer, G., Shirgazin, R.: The two-dimensional strip packing problem: A numerical experiment with waste-free instances using algorithms with block structure. Preprint MATH-NM-01-2005 TU Dresden (2005)
14. Matus, E., Seidel, H., Limberg, T., Robelly, P., Fettweis, G.: A GFLOPS Vector-DSP for broadband wireless applications. In: CICC '06: Proceedings of the IEEE Custom Integrated Circuits Conference. (2006) 543–546
15. Cichon, G., Fettweis, G.: MOUSE: A shortcut from matlab source to SIMD DSP assembly code. In: SAMOS '03: Proceedings of the International Workshop on Systems, Architectures, Modeling, and Simulation. (2003) 159–167