

A Mapping Framework Based on Packing for Design Space Exploration of Heterogeneous MPSoCs *

Bastian Ristau, Torsten Limberg, and Gerhard Fettweis
TU Dresden, Vodafone Chair Mobile Communications Systems
01062 Dresden, Germany
{ristau, limberg, fettweis}@ifn.et.tu-dresden.de

The computational demand of signal processing algorithms is rising continuously. Heterogeneous embedded multiprocessor systems-on-chips are one solution to satisfy this demand. But to be able to take advantage of these systems, new strategies are required to map applications to such a system and to evaluate the systems performance at a very early design stage. We will present a framework for static, analytical, bottom-up temporal and spatial mapping of applications to MPSoCs based on packing. This mapping framework allows easy performance evaluation and design space exploration of heterogeneous systems on chip.

1 Introduction

The computational demand of signal processing algorithms is rising continuously. Heterogeneous embedded systems-on-chip are one solution to satisfy this demand. Though ASIC centered single chip solutions are usually smaller and more energy efficient [1], flexibility and reusability of MPSoC components is comparatively higher. This is important especially in signal processing, since it enables to react to future changes in signal processing algorithms or to implement new algorithms without changing the hardware.

Future MPSoCs will mainly consist of well known processing elements (PE) which have proven their worth for a certain application domain. Since these PEs have been used in other applications, for most tasks key implementation parameters like execution time and power consumption are already known at design time. If not so, an experienced system designer will probably be able to estimate these parameters. Each processor can

*The original publication is available at www.springerlink.com.

be integrated into the system multiple times. PEs in general have local memories and are connected by some kind of network or bus.

Today MPSoCs are mainly evaluated with simulation models written in languages such as Simulink or SystemC. But simulation models require a lot of implementation effort. Furthermore explicit spatial mapping of tasks to processors or at least mapping to a certain processor kind is required. As a result only one of the available implementations is covered within a single simulation and each corner-case of the application has to be determined and simulated separately. Therefore identifying worst-case execution times is time-consuming and error prone.

But execution time as well as power consumption for a given application strongly depends on the way the application is mapped to the system temporally and spatially. Hence to be able to evaluate an MPSoC for a given set of applications it is important to exploit the available parallelism of the architecture and the application. Automatic iteration over different mappings allows efficient analysis of this parallelism. In the optimization process not only task mapping has to be considered. The mapping of data to memories and data transfers to communication resources also has significant influence on latency, power consumption, etc. This is especially true for streaming applications where the data rate is increasing continuously.

We will present a static, analytical, bottom-up methodology for spatial and temporal mapping of applications to a given architecture following the Y-Chart approach [2] without the need for simulations. 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. Our framework automates these iterations over different mappings and allows fast performance analysis of a given system. In addition detailed mapping result analysis facilitates identifying performance bottlenecks and possibilities for system improvement enabling a guided walk through the design space.

Our framework is extensible for reading various modeling languages and provides the possibility to test different mapping strategies. It supports temporal and spatial mapping not only to processors, but also to processor types. The framework is not intended to be a replacement for the mentioned modeling languages. In fact it is intended to be used for finding a candidate system that can be simulated more detailed with existing modeling tools and languages.

The rest of this paper is organized as follows: Sec. 2 puts this work into the context of existing works, Sec. 3 gives an overview over the framework structure followed by a detailed description about the packing based mapping methodology in Sec. 4 which is the center part of our framework. Sec. 5 gives some results for applying our methodology to a set of signal processing algorithms and Sec. 6 shows how analysis of the mapping result can be utilized for system refinement. Finally the paper is concluded in Sec. 7.

2 Related Work

There are a lot of frameworks for modeling and simulating MPSoCs such as Simulink, SystemC, Sesame [3], Ptolemy [4], Casse [5], MILAN [6], just to mention a few. Since these frameworks are focusing on functional simulation they require a model described in their own modeling languages and in most cases a manual mapping of tasks to processors. We see these tools as a refinement after design space exploration has led to a promising system, but not for finding an initial architecture because of the downsides of simulation models for exploring the design space mentioned in Sec. 1.

In [7] a methodology using multi-objective optimization (MOO) is presented, that is producing an initial mapping for the Sesame framework, but to be able to handle the optimization problem details are neglected in the model. Thus the results have to be refined and checked with simulations. In [8] an automatic mapping approach for the Cell BE architecture is introduced requiring a priori selection of processor kinds for each task.

Our framework in contrast supports simultaneous temporal and spatial mapping of all tasks not only to processors, but also to processor types. Since we abstract from the functional behavior of the application we are able to abstract from the modeling language as well. As long as a given modeling language has certain properties described in Sec. 3.1 it is possible to implement a front-end for this language.

The framework is not intended to be a replacement for the mentioned frameworks. The idea is to explore and reduce the design space with the presented method to find promising candidate systems that can be simulated more detailed with existing modeling tools and languages.

3 The Mapping Framework

The idea of our framework basically follows the Y-Chart approach but extends it by an additional guidance step (Fig. 1). We start with separate graph based descriptions of application and architecture. For the architecture we assume a given MPSoC as described in Sec. 1, which is determined by an experienced system designer. The outlines of both models are passed to the guidance step where information about possible implementations of application parts on architecture parts is provided. This data along with the application and architecture models are read in by the mapping engine which performs a temporal and spatial mapping based on packing. The generated mapping is analyzed in the evaluation step where different analysis such as load analysis, critical path analysis, etc. can be performed. Each step will be explained in detail in the following sections.

The results of the evaluation step are used to identify possibilities for system improvement. For instance if load analysis unveils a very high or low load on some kinds of processors, one solution would be to add another or respectively remove one of these processors and evaluate the modified system again.

Another use-case scenario would be that there already exists an MPSoC and an up-

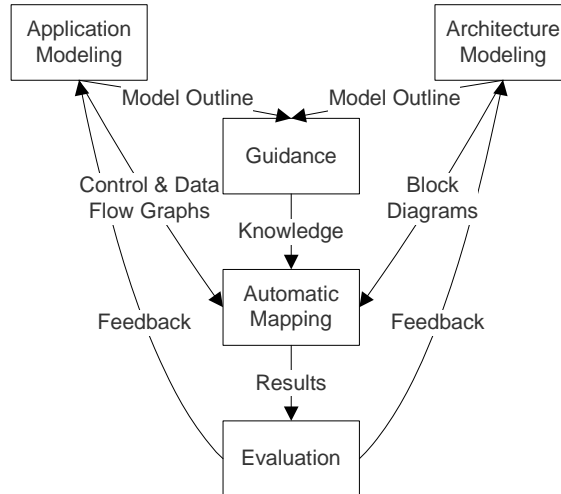


Figure 1: Methodology overview following the Y-Chart approach

graded or new option for certain processor types becomes available. The framework can be used to check, if a replacement of these processors with the new processor type will result in significant speed-up. Within this scenario it is also possible to evaluate a fictitious processor that is under development using estimates for cycle counts and power consumption.

As there is a feedback loop not only from evaluation to architecture but also to the application in the Y-Chart a critical path analysis can help to identify paths in the application where some more parallelism would help to speed up run-time.

The framework is implemented as a set of C++ libraries. The strict separation of each step allows the exchange of stages and to plug in different front-ends, mapping engines, etc. To disburden the user from the details we also implemented a GUI that calls the library functions for reading models and implementation data, performing the mapping and writing the mapping report. Convenient control over basic optimization parameters and used file formats is nevertheless accessible to the user. Since we utilize existing tools for modeling of application and architecture as well as for visualization of the mapping result, we were able to keep the GUI minimalist and easy to use.

Our mapping framework currently uses a single objective, bottom-up, analytical approach. The analytical approach ensures that every possible branch the application can take is considered within the model. However, 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 the 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.

3.1 Application and Architecture Modeling

For modeling applications 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.

The internal representation of the CDFGs is similar to the structure of YML [9]. Although YML was designed to deal with Kahn-Process-Networks derived from Compaan [10], it is suitable for CDFGs as well. The concept is based on nodes communicating over edges via ports. This concept is also used in other modeling languages such as UML or SystemC. We currently have implemented front-ends for YML, UML activity diagrams and the intermediate representation used by the MOUSE compiler [11]. Front-ends for other languages are under development.

A node in this context is a task in the application. The ports of the nodes represent input and output data of these tasks. Edges from node to node describe control flow and edges going from port to port data flow. As tasks can contain sub-tasks a node at a given abstraction level can also represent a graph in a lower level of abstraction.

To handle loops and branches efficiently we make two restrictions on the CDFGs during the modeling phase. Loops are wrapped into a loop node resulting in an almost acyclic graph containing loops only from a node to itself. Each branch caused by an if-then or switch-case statement is also wrapped into a separate node enabling fast detection of two mutually exclusive tasks. As a result static single assignment form (SSA)

[12] known from compiler construction can be generated easily if needed. These two restrictions allow us to use results and algorithms from graph theory that require directed acyclic graphs.

As the concept of nodes communicating over edges via ports can be applied to architecture as well we also use the described structure for modeling MPSoCs. On the architecture side nodes are used to describe processors, memories, interconnect elements, etc. Ports are representing the ports of these elements, edges the interconnection between elements. Multiple abstraction levels are present also in the architecture models. A processor modeled by a node in the MPSoC usually contains cores and local memories modeled by nodes in the processor node that is considered as a graph in this context, for instance.

3.2 Guidance

MPSoCs usually consist of existing components. For each component performance figures such as latencies or execution times of single tasks on a specific processor are known or can be estimated well. This is combined with the knowledge which mappings make sense and which do not, e.g. implementing a Viterbi decoding algorithm on a RISC processor.

In the guidance step we collect this knowledge and perform the mapping based on the given performance figures. In this way the mapping can be performed without the need for simulations. Undesired mappings are excluded by simply not providing

implementation data for these specific cases. This reduces the solution space already in the modeling phase, which results in faster solution times.

The methodology is comparable to the rules for instruction selection in compiler construction. Currently we use tables in XML format to describe these mapping options. In the future this concept can be realized with implementation data stored and maintained in a database, for instance.

3.3 Mapping

To find a mapping and evaluate the performance of the given system, we divided the mapping process in several stages (Fig. 2).

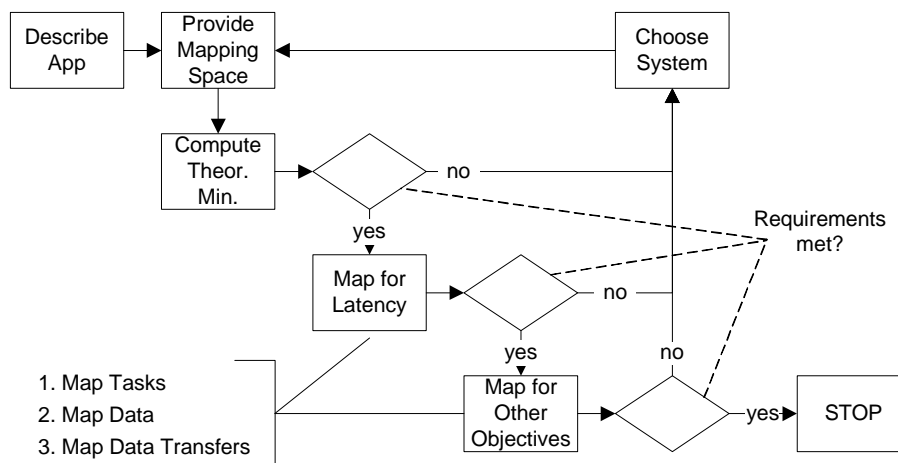


Figure 2: Structure of the mapping process

As mentioned there are several objectives such as performance or power consumption when optimizing a mapping. Against today’s trend toward multi-objective optimization (MOO) techniques we selected a lexicographic optimization. During the solution process in MOO all objectives are treated simultaneously. Given n objectives MOO generates a n -dim. pareto optimal front. But since people tend to think sequentially, selection of one of the solution usually is done by looking successively at one dimension at a time. A step-wise lexicographic optimization which reduces the feasible set of solutions successively exactly reflects this behavior.

In the first stage we run a fast optimization to check, if the chosen system can meet the required timing constraints without resource constraints. This theoretic minimum is assuming an MPSoC with the selected kinds of processors, but unlimited number of each processor kind. The problem is solved exact in polynomial time with a modified version of Dijkstra’s shortest path algorithm [13]. Since the algorithm is well-known, we neglect the details in this paper.

After that we take resource constraints into account and map for the desired objectives. We identified mapping for minimum execution time as primary objective. 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.

But there is a more important reason for latency as primary objective. Mapping for minimum latency allows for analysis of crucial parameters such as system load and critical path. When mapped for other objectives these evaluations can still be performed, but do not allow identifying performance bottlenecks and overhead since the mapping is different. But exactly these performance overhead or bottlenecks can show, if the number of a certain kind of processor in the MPSoC can be reduced or should be increased. Reducing the number of processors can reduce power consumption more than a low power optimized mapping to the original system would do. Hence the objective of the mapping problem corresponds only indirectly to the design goals.

Apart from the computation of the theoretic minimum all mapping stages are divided into three steps, which are described in detail in the next section. The three steps can be seen as a stepwise refinement of the mapping. After each step the mapping process is stopped, if the timing constraints cannot be met. In the first step tasks are mapped to processors of the MPSoC. Second, the data are mapped to suitable memories and memory addresses. This step also determines, if the chosen memory capacity is sufficient. After that transfers are taken care of. These mappings can be described as functions $f(1)$, $g(2)$ and $h(3)$.

$$f : \text{task} \rightarrow (\text{processor, starting time}) \quad (1)$$

$$g : \text{data} \rightarrow (\text{memory, address}) \quad (2)$$

$$h : \text{data transfer} \rightarrow (\text{network resource, starting time}) \quad (3)$$

We are aware of the fact, that there have been efforts treating individual phases – just to name [14, 15] as prominent examples. Our approach differs in reducing all of these problems to the class of packing problems. Another advantage esp. in memory allocation is the fact, that in comparison to [15] we are able to deal with not only two, but an arbitrary number of memories simultaneously. In addition, we keep the graph structure of the application throughout the whole mapping process, which can be useful for further optimizations.

3.4 Evaluation

As the result of the mapping has the property of a project schedule we decided to use existing project management software to visualize the mapping results (Fig. 3). Using project management software also enables us to make use of built-in analysis such as resource utilization, critical path identification, etc. for architecture refinement. The result is currently written in MS Project XML format.

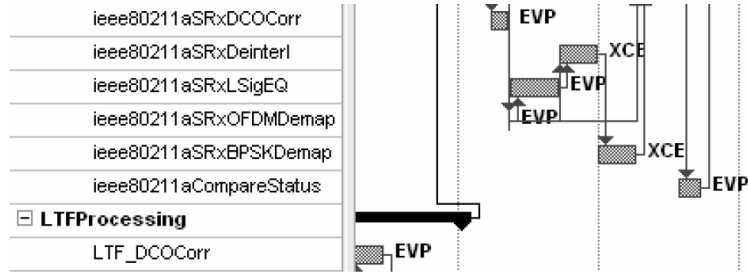


Figure 3: Visualization of the mapping result in MS Project

4 Mapping via Packing

In this section we will show that mapping is closely related to packing problems. We will give a detailed description of similarities and necessary modifications in the linear programs for mapping tasks, data and data transfers. As mentioned there is more than one objective for the mapping. We will state the linear programs for minimizing latency as we identified this as primary objective. A short outline of the necessary modifications for other objectives in subsequent optimizations will be stated at the end of each description.

4.1 Mapping Tasks

The problem of mapping tasks temporal and spatial can be interpreted as 2-dimensional strip-packing problem [16]. 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 4 illustrates the application of packing to mapping tasks.

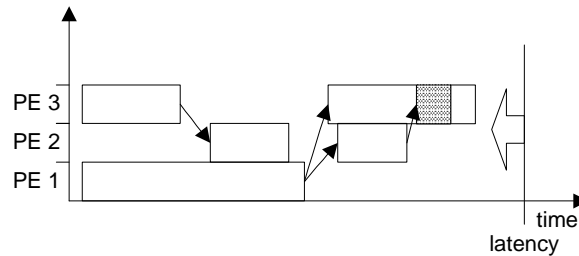


Figure 4: Application of packing to mapping tasks

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.

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 (4).

$$\sum_{k \in P_i} y_{i,k} = 1, \quad \sum_{k \notin P_i} y_{i,k} = 0 \quad \forall i \quad (4)$$

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, 5 guarantees that all tasks are finished before total execution time w_{\min} and 6 that precedence constraints are met. 6 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.

$$x_i + \sum_k W_{i,k} y_{i,k} \leq w^{\min} \quad \forall i \quad (5)$$

$$x_i + \sum_k W_{i,k} y_{i,k} \leq x_j \quad \forall i, j : j \text{ depends on } i \quad (6)$$

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 $k = l$, 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 (7) – (11) 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-caseconstruct. 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. This possible overlap is illustrated in Fig. 4 by the gray area. By including this property into our methodology, each possible branch that can be realized in the CDFG is considered automatically.

$$x_i + \sum_k W'_{i,k} y_{i,k} - W^{\max} + W^{\max} b_{i,j} \leq x_j \quad (7)$$

$$\sum_k k y_{i,k} + 1 - H^{\max} + H^{\max} u_{i,j} \leq \sum_k k y_{j,k} \quad (8)$$

$$b_{j,i} + b_{i,j} \leq 1 \quad (9)$$

$$u_{j,i} + u_{i,j} \leq 1 \quad (10)$$

$$u_{j,i} + u_{i,j} + b_{j,i} + b_{i,j} \geq 1 \quad \forall \text{ interfering } i, j \quad (11)$$

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 (7) and (8) are redundant with $x_j = 0$ and $\sum_k ky_{j,k} = 0$ 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 (12) (with $A :=$ adjacency matrix) additionally minimizes the time between two adjacent tasks and therefore liveness of data. The objective can be refined by adding additional terms, e.g. trying to put adjacent tasks onto the same processor for reducing needed transfers.

$$W^{\min} + \left(\sum_{i,j} \frac{A_{i,j}(x_j - x_i - \sum_k W_{i,k}y_{i,k})}{W^{\max}} \right) \rightarrow \min \quad (12)$$

For mapping transfers, almost the same methodology can be applied, because the transfers have to be performed by some components of the system. The details are given in Sec. 4.3 But since the components involved in these transfers are determined by the memory allocation, this has to be done after the mapping of data and cannot be integrated into the mapping of tasks.

When optimization for power consumption is desired also, we propose to perform this in a second optimization run. Let us assume that we have computed a minimum latency with the method mentioned before. Then the width of the strip denoting that latency is given for the problem of minimizing power consumption. As an effect the variable W becomes a constant. This constant can be enlarged if some additional latency is permitted to reduce power consumption in return. Apart from the transformation of W from variable to constant all constraints in the linear program stay untouched. Only the objective has to be modified. One possibility is to introduce a measure $C_{i,k}$ for power consumption of task i on processor k and use (13) as objective function. This objective can be refined by introducing terms for minimizing used resources, for instance.

$$\sum_{i,k} C_{i,k}y_{i,k} \rightarrow \min \quad (13)$$

4.2 Mapping Data

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

Mapping data in heterogeneous MPSoCs is similar to register allocation in compiler construction. A well known solution for this problem is Chaitin's graph coloring algorithm [15]. But in opposite to register allocation there are usually more than the two memories (register and global memory) in MPSoCs, such as 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 data 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 data 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 data i to be mapped represent the boxes to be packed into the shelves, whose heights h_i matches the size of the data.

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 (14).

$$\sum_{k \in M_i} x_{i,k} = 1, \quad \sum_{k \notin M_i} x_{i,k} = 0 \quad \forall i \quad (14)$$

Let y_i be the starting address of variable i in the memory. Eq. (15) takes care, that the variable is stored in a valid memory address.

$$y_i + h_i \leq \sum_k C_k x_{i,k} \forall i \quad (15)$$

Let H^{offset} be the shelf height of memory k and $H^{\text{max}} \geq \max_k \{H^{\text{offset}} + C_k\}$ a constant denoting the height of the rack. To make certain that two data are not stored in the same memory at the same address, we introduce non-overlapping constraints (refe16) & (17) for all interfering data i, j .

$$y_i + h_i + \sum_k H_k^{\text{offset}} x_{i,k} - H^{\text{max}} + H^{\text{max}} u_{j,i} \leq y_j + \sum_k H_k^{\text{offset}} x_{j,k} \quad (16)$$

$$u_{j,i} + u_{i,j} = 1 \quad \forall \text{ interfering } i, j \quad (17)$$

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 data into fast memories. This is done by (19), where the required resource amounts for memory k are denoted with y_k^{min} and constraint by (18).

$$y_i + h_i + H^{\text{max}} x_{i,k} - H^{\text{max}} \leq y_k^{\text{mem}} \quad \forall i, k \quad (18)$$

$$\sum_{i,k} W_k x_{i,k} + \sum_k y_k^{\text{mem}} C_k^{-1} \rightarrow \min \quad (19)$$

If power consumption should be taken into account, we can introduce the constant C_k denoting the costs of memory k in terms of power consumption. Then the objective can be modified to (20), for instance.

$$\sum_{i,k} C_k x_{i,k} \rightarrow \min \quad (20)$$

As mentioned in the beginning of this section we assume interference if two data can interfere. This is due to the fact that we keep the graph structure throughout the whole optimization process. If this is not needed, we can also try to reduce interference by integrating the temporal scheduling of tasks to achieve better results in terms of power consumption. The details are presented in [17].

4.3 Mapping Data Transfers

After the tasks and data are mapped to processors and memories data transfers have to be mapped. This contains determining starting times and – in case of more than one unit capable of performing a transfer – the unit that should perform a transfer.

To solve this problem we first apply two graph transformations. The first one adds additional control flow dependencies between the tasks in the application graph caused by the task mapping. This is required, if two tasks without a given dependency are mapped to the same processor. The task with the later starting time becomes dependent on the earlier scheduled task.

In the second graph transformation step we add tasks for performing the transfers, remove the original data dependency and introduce dependencies between the transfer and task nodes. Figure 5 shows an example transformation where data is transferred temporary into some global memory and then transferred back to some local memory.

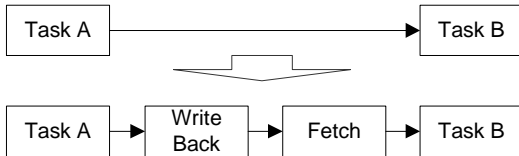


Figure 5: Adding new nodes into the application graph based on the data mapping

Note that in case of a direct transfer from local to local memory (determined by the data mapping) only one additional node is inserted between the two tasks.

After these two transformations the methodology described in Sec. 4.1 can be applied almost without changes enabling the reuse of the task mapping implementation as follows: Tasks representing transfers are executed exactly as computational tasks using hardware resources. Therefore we can apply the task mapping methodology to the transformed graph. But since the original task and data mapping determined the necessary graph transformations we have to keep the spatial mapping of already mapped tasks.

Let M be the set containing already mapped tasks and $m(i), i \in M$ a function returning the index of the processor task i is mapped to. Then replacing (4) by (21) $\forall i \in M$ ensures that all already mapped tasks are mapped to the same resource as before.

$$y_{i,m(i)} = 1, \quad \sum_{k \neq m(i)} y_{i,k} = 0 \quad \forall i \in M \quad (21)$$

We make no additional restrictions on the temporal mapping of the tasks. Tasks that are not on the critical path have some buffer time. Allowing to move the starting

time determined by the task mapping ensures the possibility to change the starting time within the buffer period to insert transfer mappings efficiently.

As the optimization problem is basically the same as for mapping tasks the same methodologies for optimizing for power consumption mentioned in Sec. 4.1 can be applied.

5 Results

In this section we will present evaluation results of applying the proposed method to a case study.

5.1 Testbench Setup

For the case study we considered the SAMIRA vector DSP [18] based on Synchronous Transfer Architecture (STA) [19] as an MPSoC. Intermediate representations of test bench applications generated for the MOUSE compiler [19] are used as sample applications.

STA is a concept of loosely coupled functional units and different memories connected by an interconnection network. The functional units can be considered as processors in the MPSoC. Each of these units has output registers. This characteristic is used to model the behavior of keeping data in local memory rather than having to write it back to shared or global memory.

The SAMIRA has 21 functional units. As an important property for legitimating this architecture for the MPSoC 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.

The advantage of this abstract example is, that we have well-known execution times for the instructions on the one hand and on the other hand a set of test bench applications with large CDFGs to demonstrate the potentials and limits of our methodology.

We also modeled and inspected mapping IEEE 802.11a and MJPEG to a model of a real MPSoC, but the CDFGs were rather small and did not contain much parallelism. Also the considered MPSoCs did not consist of a large number of processors. This resulted in obvious mappings which can be performed manually. For that reasons we neglect the results in this paper.

5.2 Evaluation of the Results

After implementing front-ends for the STA given in XML format as well as for the test-bench applications, we applied the methodology described in Sec. 4. We compared the performance of our methodology with the results from our compiler and the theoretic minimum which is defined as the minimum latency without resource constraints. For denoting the impact of transfers on the overall latency the latency for mapping tasks without transfers is denoted in Fig. 6, too.

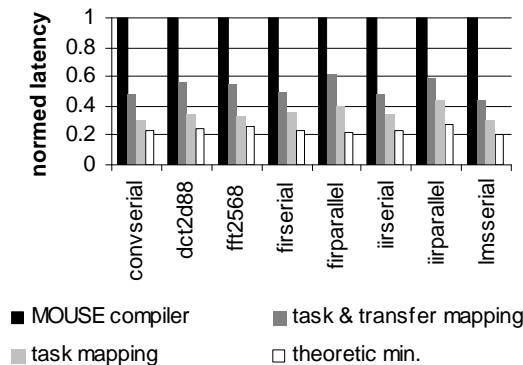


Figure 6: Comparison of the quality of the mapping methodology with the MOUSE compiler and the theoretic min without resource constraints

Figure 6 shows an improvement of almost 100% in average after transfer mapping compared to the existing MOUSE compiler. But even more important is the comparison of the task mapping result before transfer mapping with the theoretic optimum (same processor kinds, but unlimited number of processors). As Fig. 6 shows, we are within a factor of 1.5 of this minimum in average. Figure 6 also reveals that the data transfers have strong impact on the execution time although we were able to keep the majority of data in local memories rather than having to transfer them into the register file or global memory.

5.3 Limits

Solving was done by passing the problem to the LP solver 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 used a time limit of 30 seconds for each graph and mapping step. To get an indicator about the quality of these solutions, we ran the CPLEX solver for one large (sub-) graph of an application for more than six hours, which showed an improvement of about 10% over the solution found after less than a minute. The framework also has an option to use the open source solver GLPK which performs well compared to other open source solutions according to [20], but we experienced some significant performance differences esp. when dealing with large mixed integer linear programs (MILP).

Table 1 shows the times required for the mapping stages. We think the denoted running times are acceptable at an early design stage, since no additional simulations are required. Moreover, note that the solving time can be accelerated by reducing the time limit.

The upper bound for the problem size that can be handled strongly depends on the characteristics of the sub-problems. For example in embarrassingly parallel CDFGs the number of constraints given by (7) – (11) is rather large whereas the number is low

Table 1: Solution times for mapping tasks of various signal processing algorithms using CPLEX 10.1 and a 30 sec. time limit

application	graphs	nodes	task mapping (sec.)	data mapping (sec.)	transfer mapping (sec.)
convserial	13	102	35	31	8
dct2d88	74	1018	205	170	198
fft2568	78	703	102	59	100
firserial	7	95	32	33	1
firparallel	9	131	33	34	68
iirserial	7	71	34	32	0
iirparallel	9	140	40	33	96
lmsserial	16	194	34	65	41

in embarrassingly serial CDFGs. This results in a large (respectively small) number of binary variables which have strong impact on the solution time. Same property holds for the data and data transfer mapping methods. Thus we can only give bounds for the example applications and architectures examined in this section, where we were able to handle CDFGs with up to 150 nodes and 120 data flow edges per sub-graph.

In general the exponential growing complexity of the MILP results in rather hard bounds which can be identified easily when the methodology is applied. If a certain sub-graph cannot be solved we suggest having either a fallback method using appropriate heuristics such as list scheduling for the task mapping or dividing the sub-graph into sequential sub-graphs. In the latter case the methodology can be applied without changes if the resulting subgraphs have the property that edges between two of these sub-graphs are originating in one of these sub-graphs.

6 Example for Using Mapping Results as a Guide to System Refinement

As mentioned in Sec. 1, the key feature of the framework is the possibility to map an application to an architecture automatically. In this section we will show, how mapping result analysis can lead to system refinement and in this way enabling a guided walk through the design space.

Since we are able to export the task mapping result in MS Project XML format we can derive some performance figures such as processor loads or memory requirements easily. For this example we perform a load analysis for the signal processing algorithms mapped to the functional units of the SAMIRA vector DSP (Sec. 5). Table 2 shows the result of this analysis. Since we were able to reach the theoretic minimum in some graphs, we excluded this graphs in one analysis to identify the bottlenecks only in suboptimal blocks as well.

Table 2 reveals a very high load in the immediate unit compared to the loads of the other functional units. Especially in the sub-optimal blocks the immediate unit turned

Table 2: Load analysis result for the signal processing algorithms mapped to the functional units of the SAMIRA vector DSP. Functional units with loads < 0.1 are omitted.

functional unit	avg. load (sub-opt. graphs)	avg. load (all graphs)
immediate	0.83	0.67
salu1	0.31	0.42
salu4	0.17	0.16
salu2	0.16	0.14
smem	0.14	0.12
sshift	0.14	0.15
salu3	0.11	0.70
smul	0.10	0.12
vmem	0.10	0.09

out to be the performance bottleneck. Thus, we modeled a system with two immediates and performed the mapping for this modified system. Figure 7 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%.

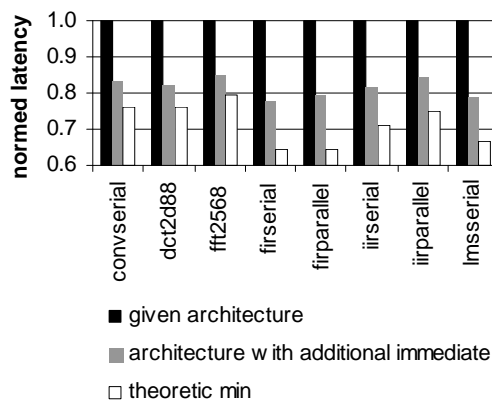


Figure 7: Improvement gained by adding a second immediate unit to the given architecture as result of the load analysis

7 Conclusion

We showed that automating mapping of applications to a given heterogeneous MPSoC enables not only performance analysis but can also help to refine the system. Therefore our approach can be used for design space exploration. We outlined the structure of our framework for automatic mapping and showed that the mapping problem can be treated as packing problem which can be solved efficiently by existing optimization software. Tests have shown that the framework is able to deal with rather large CDFGs and MPSoCs.

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] H. Blume, H.T. Feldkämper, and T.G. Noll. Model-based exploration of the design space for heterogeneous systems on chip. *Journal of VLSI Signal Processing Systems*, 40(1):19–34, 2005.
- [2] B. Kienhuis, E. Deprettere, K. Vissers, and P. van der Wolf. An approach for quantitative analysis of application-specific dataflow architectures. In *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'97)*, pages 338–349, 1997.
- [3] A.D. Pimentel, C. Erbas, and S. Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Transactions on Computers*, 55(2):99–112, 2006.
- [4] E.A. Lee. Overview of the Ptolemy project. Technical memorandum UCB/ERL M03/25, University of California, Berkeley, CA, 94720, USA, 2003.
- [5] V. Reyes, W. Kruijtzter, T. Bautista, G. Alkadi, and A. Nunez. A unified system-level modeling and simulation environment for MPSoC design: MPEG-4 decoder case study. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'06)*, pages 474–479, 2006.
- [6] A. Bakshi, V.K. Prasanna, and A. Ledeczi. MILAN: A model based integrated simulation framework for design of embedded systems. In *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES'01)*, pages 82–93, 2001.

- [7] C. Erbas, S.C. Erbas, and A.D. Pimentel. A multiobjective optimization model for exploring multiprocessor mappings of process networks. In *Proceedings of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'03)*, pages 182–187, 2003.
- [8] P. Bellens, J.M. Perez, R.M. Badia, and J. Labarta. CellSs: a programming model for the Cell BE architecture. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'06)*, pages 86–96, 2006.
- [9] J.E. Coffland and A.D. Pimentel. A software framework for efficient system-level performance evaluation of embedded systems. In M. Matsui and R.J. Zuccherato, editors, *SAC 2003*, volume 4006 of *LNCS*, pages 666–671. Springer, 2003.
- [10] A. Turjan, B. Kienhuis, and E. Deprettere. Translating affine nested-loop programs to process networks. In *Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'04)*, pages 220–229, 2004.
- [11] G. Cichon and G. Fettweis. MOUSE: A shortcut from matlab source to SIMD DSP assembly code. In *Proceedings of the International Workshop on Systems, Architectures, MOdeling, and Simulation (SAMOS'03)*, pages 159–167, 2003.
- [12] R. Cytron, J. Ferrante, B.K. Rosen, M.N. Wegman, and F.K. Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, 13(4):451–490, 1991.
- [13] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.
- [14] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.
- [15] G.J. Chaitin, M.A. Auslander, A.K. Chandra, J. Cocke, M.E. Hopkins, and P.W. Markstein. Register allocation via coloring. *Computer Languages*, 6(1):47–57, 1981.
- [16] G. Belov, A.V. Chiglintsev, A.S. Filippova, E. Mukhacheva, G. Scheithauer, and R. Shirgazin. 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.
- [17] B. Ristau and G. Fettweis. An optimization methodology for memory allocation and task scheduling in SoCs via linear programming. In S. Vassiliadis, S. Wong, and T. Hämäläinen, editors, *SAMOS 2006*, volume 4017 of *LNCS*, pages 89–98. Springer, 2006.
- [18] E. Matus, H. Seidel, T. Limberg, P. Robelly, and G. Fettweis. A GFLOPS vector-DSP for broadband wireless applications. In *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC'06)*, pages 543–546, 2006.

- [19] G. Cichon, P. Robelly, H. Seidel, E. Matus, M. Bronzel, and G. Fettweis. Synchronous transfer architecture (STA). In A. Pimentel and S. Vassiliadis, editors, *SAMOS IV*, volume 3133 of *LNCS*, pages 343–352. Springer, 2004.
- [20] J.T. Linderoth and T.K. Ralphs. Noncommercial software for mixed-integer linear programming. In J. Karlof, editor, *Integer Programming: Theory and Practice*. CRC Press, 2005.