# Linear Programming in Embedded Systems

Bastian Ristau, Gerhard Fettweis
Vodafone Chair
Mobile Communications Systems
TU Dresden

Guntram Scheithauer, Andreas Fischer
Institute for
Numerical Mathematics
TU Dresden

## 1 Introduction

In the past signal processing algorithms for mobile communications systems were implemented in hardware using a single chip solution, usually a Digital Signal Processor (DSP). But due to the increased and still increasing complexity of these signal processing algorithms, the computational power of single chip solutions is not sufficient any more. In addition the number of algorithms to be processed simultaneously is rising. For example owners of mobile phones nowadays want to use WLAN, WiMAX and DVB-H at the same time.

To tackle this increasing computational demand, there are two competing approaches in chip design. On the one hand dedicated hardware design like application specific integrated circuits (ASICs), on the other hand so-called systems-on-chip (SoC) design. SoCs can consist of several general purpose processors as well as DSPs and ASICs. In general terms ASIC-based solutions are smaller and more energy efficient [1] than SoC-based solutions, but not as flexible, because once the chip is developed, the possibility to modify the behavior of the system is very limited. Another advantage of SoCs is, that components can be reused from existing processor designs. Thus, from now on we will concentrate on SoCs.

There are several contradicting objectives for designing SoCs. These objectives can be area (die size), performance, energy consumption as well as rather soft defined goals like programmability or flexibility. Since chip design and verification is very time consuming, the designer wants to know at a very early design stage, how the system should look like, how the application behaves on the determined architecture and what the performance figures would be. An idea presented in [2] is to determine a first system at a very high abstraction level neglecting some details and in this way reducing the set of remaining candidate systems. With the gained solution the designer can proceed with the next lower abstraction level and refine the system step by step. Figure 1 illustrates this method.

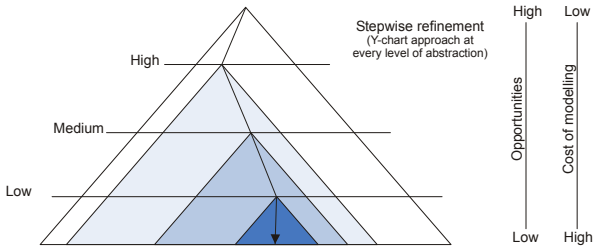


Figure 1: Using multiple abstraction levels in SoC design

## 2 Applications of Linear Programming

One methodology to determine a suitable SoC-architecture is the Y-Chart approach [3]. It is based on the idea of modeling architecture and applications separately, performing a mapping step and iterating over different architectures, applications and mappings until a suitable architecture is found (Fig. 2)

Since manual iteration over different architectures and mappings would be very time consuming, there is a need for systematic exploration of the design space. Automating the iterations by solving it via linear programming can be one solution for this problem.

As there are two nested iteration loops for a given

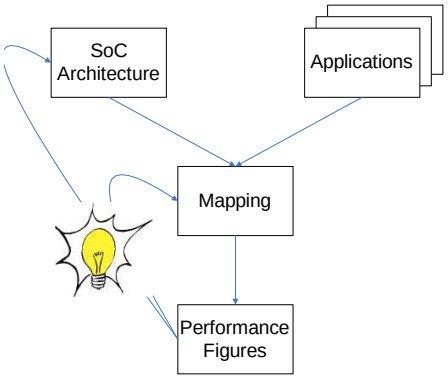


Figure 2: Illustration of the Y-Chart approach

set of applications, this leads to nested optimization problems. The outer loop iterates over architectures, the inner loop optimizes the temporal and spatial mapping for each architecture.

When optimizing the architecture, several aspects have to be considered. For example not only the number and kinds of processors have to be chosen. Also memory hierarchy (local, shared, global memories) and the interconnection network have to be selected.

Optimization of the mapping includes the spatial and temporal assignment of the sub-functions of the application to processors of the given system (resource allocation and scheduling) as well as the allocation of data in memories and the scheduling of data transfers between memories and processors. Thus, the whole mapping process can be considered again as a set of nested optimization problems.

An example for a formulation as mixed-integer linear programs (MILP) for both aspects can be found in [4].

# 3 Using Packing for Optimizing Mappings of Applications to Architectures

Mapping of applications to architectures deals with resource constraints. Thus, packing problems are very related to the subproblems arising. As an example, we want to introduce in this section, how allocation of data in memory is related to a packing problem. We will show, that solutions for this special kind of packing problem can be retrieved fast by existing solvers. This subproblem can be expanded by including other aspects of the mapping process and is described in [5]. For applying it to other aspects of the mapping process, see [4].

If a temporal and spatial mapping of tasks to processors is selected and the data is assigned to memories, the remaining question is, how much memory resources are needed. Minimizing these needed resources can be formulated as strip-packing problem [6].

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. Applying this to the problem of minimizing total memory requirements, we face a set of packing problems (one for each memory in the system), which can be solved simultaneously. The width of all strips is the total execution time determined by the mapping of tasks and transfers. The heights are the amount of resources needed for the respective memory. The boxes to be packed are the data. The width of such a box is defined by the time the data is alive. The height is given by the required size of the data in the selected memories.

Note, that due to the given temporal mapping, the horizontal positions of the boxes are fixed. Another important property is, that some boxes can overlap. Since the application can take different branches during execution caused by switch/case of if/then statements, there are some data, that cannot exist simultaneously. Thus, these data can overlap in packing terms.

Let $y_i^k$ be the starting address of variables $i$ in memory $k$ and $h_i^k$ its storage requirements. For non-overlapping we introduce $u_{i,j}^k := 1$, if variable $i$ is placed below variable $j$ (0 otherwise).

Memory usually is available in discrete sizes, e.g. 64K, 128K, etc. Therefore, not total height but memory capacity steps have to be optimized. Let $z_l^k \in \{0,1\}$ be the memory capacity step and $C_l^k$ the respective capacity of this capacity step.

The MILP can now be stated as follows:

$$\sum_{l,k} l z_l^k \quad \rightarrow \quad \min \tag{1}$$

subject to

$$\sum_{l} z_l^k \quad = \quad 1 \tag{2}$$

$$y_i^k + h_i^k \quad \leq \quad \sum_{l} z_l^k C_l^k \tag{3}$$

$$y_i^k + h_i^k - H u_{j,i}^k \quad \leq \quad y_j^k \tag{4}$$

$$y_i^k + h_j^k - H u_{i,j}^k \quad \leq \quad y_i^k \tag{5}$$

$$u_{j,i}^k + u_{i,j}^k \quad = \quad 1 \tag{6}$$

with $y_i^k \in \mathbb{R}_{\geq 0}$ and $u_{i,j}^k$, $z_l^k \in \mathbb{B}$. (2) assures that exactly one capacity step is chosen, and (3) that all variables are placed within the memory. Thus, both equations are stated for all variables. Remaining constraints (4)–(6) ensure non-overlapping, if needed, and are stated only for a appropriate subset of variables. $H$ is a constant big enough to make one of the constraints (4) and (5) redundant with $y_j^k \geq 0$ and $y_i^k \geq 0$, respectively (dependent on the values of $u_{i,j}^k$ and $u_{j,i}^k$).

For this problem we generated a set of 100 randomly generated test-instances and solved them using CPLEX 9.1. The results showed the interesting property, that a first solution was found comparably fast. In addition, the quality of the solution (defined by $\frac{\text{current objective value}}{\text{optimal objective value}}$) was improved mainly at the very beginning of the solution process. For our test cases the exact solution was always found in less than a second, the proof of optimality took in worst case more than 3000 seconds. Although an exact solution could be found for rather small problems, this property can be utilized to apply the methodology on bigger problems in favor of a heuristic approach. The solution process can in this way be individually controlled with a time limit.

# 4 Conclusion

We have shown by an example, how linear optimization can be applied in embedded systems design. Although these problems are rather complex, they can be split into subproblems in a very natural way. In most cases these still complex subproblems cannot be solved in acceptable time. But during the solution progress, first solutions and quick improvements allow us to employ mixed-integer linear optimization as a heuristic in some cases.

# References


[1] H. Blume, H. T. Feldkaemper, and T. G. Noll. Model-Based Exploration of the Design Space for Heterogeneous Systems on Chip. *J. VLSI Signal Process. Syst.*, 40(1):19–34, 2005.

[2] Bart Kienhuis, Ed F. Deprettere, Pieter van der Wolf, and Kees A. Vissers. A Methodology to Design Programmable Embedded Systems - The Y-Chart Approach. In *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation - SAMOS*, volume 2268/2002, pages 18–37, Samos, Greece, 2002. Springer Berlin/Heidelberg.

[3] Bart Kienhuis, Ed Deprettere, Kees Vissers, and Pieter van der Wolf. 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*, page 338, Washington, DC, USA, 1997. IEEE Computer Society.

[4] Bastian Ristau. *Optimization of System-on-Chip-Design.* Diploma thesis, TU Dresden, Institute for Numerical Mathematics, 2005.

[5] B. Ristau and G. Fettweis. 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*, volume 4017/2006, pages 89–98, Samos, Greece, July 2006. Springer Berlin/Heidelberg.

[6] G. Belov, A.V. Chiglintsev, A.S. Filippova, E.A. Mukhacheva, G. Scheithauer, and R.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, January 2005.