A System Level Design Methodology for the Optimization of Heterogeneous Multiprocessors

M. Schwiegershausen P. Pirsch

University of Hannover
Laboratorium für Informationstechnologie
Schneiderberg 32, 30167 Hannover, Germany

Abstract

This paper presents a system level design methodology and its implementation as CAD tool for the optimization of heterogeneous multiprocessor systems. These heterogeneous systems, consisting of dedicated as well as programmable processors, are highly suitable for performing complex schemes of image processing algorithms under real time constraints. It starts from a specification of the image processing scheme, explores the design space based on a finite set of parametrizable processor modules, and by using mixed integer linear programming as mathematical framework derives heterogeneous systems, being optimal in terms of area expense and throughput rate.

1 Introduction

Today's real–time image processing applications are characterized by an increase concerning computational requirements and algorithm complexity. Examples can be found in CCITT visual telephony [1], and JPEG, MPEG [2] image compression and coding schemes. The realization of such composite DSP schemes calls for architectures, providing extremely high computational and throughput rates, which can only be achieved by massive application of parallel processing and pipelining as provided by multiprocessor systems. These DSP schemes can be splitted into several subtasks with different requirements leading to different architectures appropriate for the single image processing tasks of the whole scheme.

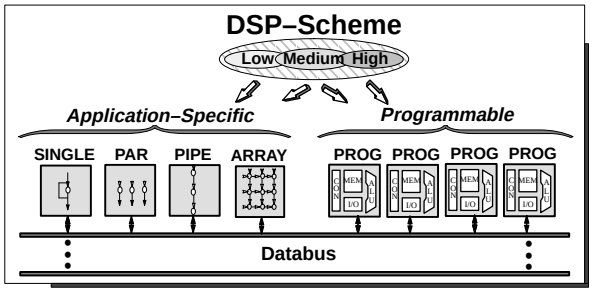

Fig. 1: Heterogeneous Multiprocessor

One efficient hardware realization is a heterogeneous multiprocessor system for the implementation of the composite scheme, see Fig. 1. Such heterogeneous systems consist of different, application–specific processors like array processors for implementing low and medium level tasks and programmable processing elements on the other hand to provide the flexibility required by medium and high level tasks. Unfortunately, for each single algorithm of the DSP scheme a large multitude of dedicated as well as programmable processor alternatives can be found. Thus, the question arises, which combination of application–specific and programmable processors leads to an optimal heterogeneous multiprocessor system for a given image processing scheme. Due to the large multitude of possible heterogeneous systems, it is impossible for the human designer to choose manually the best combination. Therefore, a systematic design methodology for the optimization of heterogeneous systems is mandatory.

The remainder of this paper is organized as follows: Section 2 reviews related research. Section 3 describes our system level design methodology for the optimization of heterogeneous systems. First results, using a video encoding scheme as application example are presented in Section 4. Section 5 gives some hints on the implementation as CAD tool. Concluding remarks are provided in Section 6.

2 Related Research

Especially, in the field of high–level synthesis there has been extensive research on mapping algorithms onto multiprocessor systems. Examples concerning methodologies for mapping one single algorithm onto a dedicated datapath or even onto an array of processing elements like systolic array processors can be found in [3]–[4]. Furthermore, DSP environments like *CATHEDRAL II, 2nd* [5]–[6] were developed in order to derive VLIW architectures consisting of synchronous DSP units with dedicated datapaths connected via a bus, addressing medium throughput applications. However, image processing applications demand for architectures providing extremely high computational and throughput rates.

Therefore, these approaches provide no mean for the derivation of heterogeneous systems, being optimal in terms of area expense and throughput rate for the execution of composite schemes of image processing algorithms. So, we present a new methodology for the optimization of these heterogeneous systems at the system level.

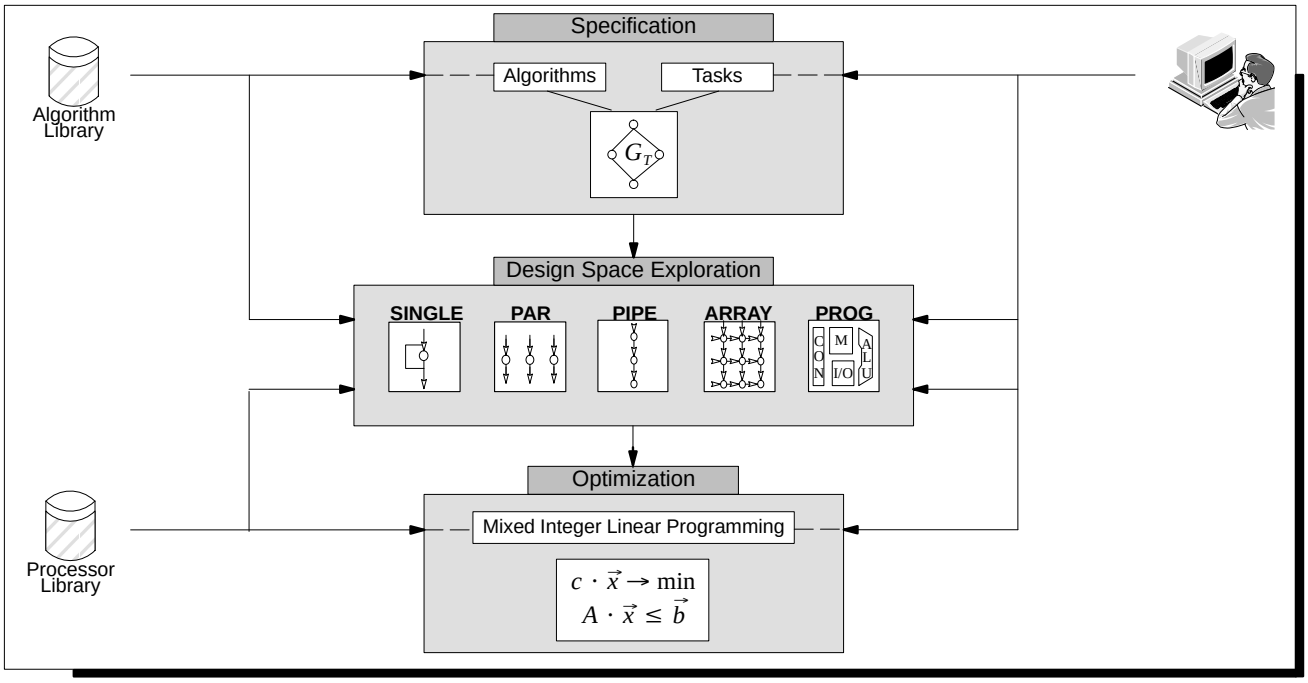

Fig. 2: Design flow of the system level design methodology

3 Design Flow

The design flow for the optimization of heterogeneous multiprocessor systems can be decomposed into three main steps, i) the *specification* of the composite DSP scheme with all single image processing tasks or algorithms and their dependencies, ii) the *design space exploration* with respect to a set of suitable processor alternatives, and iii) the *optimization* of the overall multiprocessor system, see Fig. 2.

3.1 Specification

The first step comprises the specification of all tasks or single algorithms for the composite DSP scheme as well as the dependencies between the tasks. A suitable form for representing a DSP scheme at this level is a task graph $G_T = (V, E)$ with each vertex $v_i \in V$ designating a single task m_i or algorithm $\mathcal{ALGO}(m_i)$, and each directed arc $e_{i,k} \in E$ designating a data dependency between two adjacent tasks m_i, m_k.

3.2 Design Space Exploration

After all single tasks have been specified, it is necessary to explore the possible design space for the composite DSP scheme, whereas for each single image processing algorithm of the scheme a large set of suitable architectures can be derived. Therefore, an investigation of algorithms and processor alternatives is mandatory. Our approach is to supply two libraries:

- Algorithm Library
- Processor Library

The algorithm library is intended to have the user select all single image processing algorithms $\mathcal{ALGO}(m_i)$ of the DSP scheme as well as their parameters Θ_i, whereas the processor library is intended to provide different processor alternatives suitable for each algorithm and adapted to the algorithm's parameter set Θ_i. This leads to an assignment f of a task's algorithm $\mathcal{ALGO}(m_i)$ to a set of parametrizable datapaths or processor alternatives $\mathcal{DP}_j$. Thus, in the sequel, the *algorithm library*, the *processor library*, as well as the parametrizable *assignment* is described.

3.2.1 Algorithm Library

The algorithm library can be thought of as a hierarchical tree, distinguishing between regular low–level algorithms on one side and non-regular, data dependent medium–level algorithms on the other side. Typical examples of low– and medium–level algorithms are given below:

- Low–Level Algorithms
  - filtering (FIR, IIR, etc.)
  - transform (DCT, DFT, etc.)
  - motion estimation (BMA, etc.)
- Medium–Level Algorithms
  - adaptive quantization (Q, etc.)
  - run length coding (RLC, etc.)
  - variable length coding (VLC, etc.)

Since for composite DSP schemes all single image processing algorithms are known in advance, there is no need to represent each algorithm by means of a signal processing language like *SILAGE* [5] and synthesize it by means of a datapath compiler. On the contrary, with respect to the design space exploration of heterogeneous systems, it is sufficient to characterize each algorithm class by a predefined set of parameters.

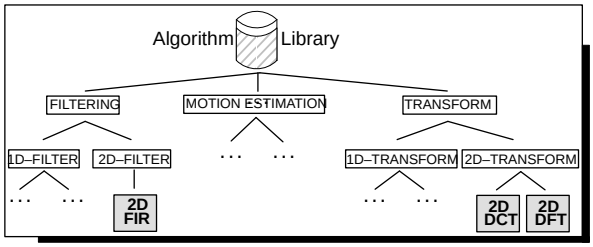

Fig. 3: Algorithm Library as hierarchical Tree

Fig. 3 shows the algorithm library as hierarchical tree, each node representing an algorithm class, and inheriting its properties, or parameters Θ_i, to all its successor nodes, algorithm subclasses respectively. Here, each leaf node corresponds to exactly one single image processing algorithm of the DSP scheme.

ALG_i	T_W	T_H	T_L	K_W	K_H	K_L	R_W	R_H	R_L	S_W	S_H	S_L	D_W	D_H	D_L	Example	Θ_i
Transform			x													1D–DCT	$[T_L]$
	x	x														2D–DCT	$[T_W, T_H]$
Filtering			x			x										1D–FIR	$[T_L, K_L]$
	x	x		x	x											2D–FIR	$[T_W, T_H, K_W, K_H]$
Motion Estimation									x			x			x	1D–BMA	$[R_L, S_L, D_L]$
							x	x		x	x		x	x		2D–BMA	$[R_W, R_H, S_W, S_H, D_W, D_H]$

Table 1: Parameter set Θ_i for some low-level algorithms

Table 1 shows the different parameter sets Θ_i for some low–level algorithms, i.e. filtering, transform, and motion estimation. Each row of Table 1 corresponds to one low–level algorithm class with a distinction between 1– and 2–dimensional algorithms. The columns indicate which parameters are necessary to characterize one specific algorithm.

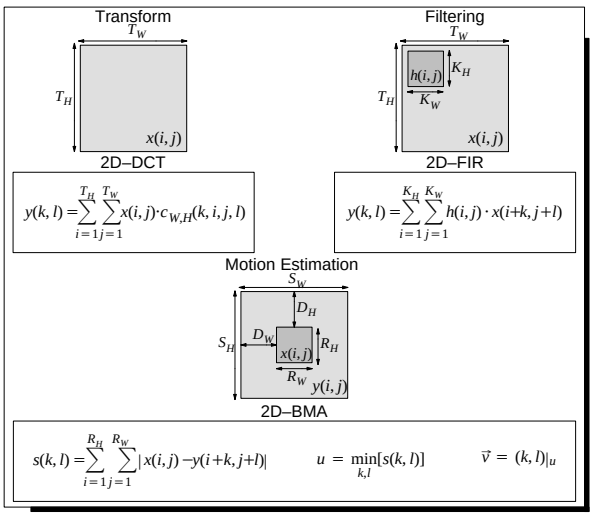

Fig. 4: Parameter sets Θ_i and their meaning

Furthermore, a typical representative for each algorithm class is given, for example a 2–dimensional discrete cosine transform (2D-DCT), belonging to the transform class. The meanings of all parameters are again visualized in Fig. 4. For example, in order to characterize a 2–dimensional transform, two parameters T_W, T_H are mandatory, whereas T_W designates the width and T_H the height for an rectangle image block of pixels to be transformed by a DCT or DFT. In case of a 2–dimensional filter algorithm like FIR, two additional parameters K_W, K_H are necessary to specify the width K_W and height K_H for the size of the kernel window, to be passed over the rectangle image block. Finally, motion estimation by blockmatching algorithms (BMA) can be described by six parameters as follows: The width R_W and height R_H for the reference block of pixels, the width S_W and height S_H for the rectangular search area, as well as D_W, D_H, designating the maximum displacement in horizontal and vertical direction to look for the best matching block within the search area region with respect to a specific blockmatching criterion, like the MAD (*mean-of-the-absolute-differences*) criterion. Concerning the 1–dimensional algorithms a similar set of parameters can be derived.

Based on these algorithm specific parameters Θ_i, our aim is to explore the permissible design space by deriving a set of parametrizable datapaths $\mathcal{DP}_j(\Theta_i)$ by means of a processor library as follows.

3.2.2 Processor Library

In order to restrict the possibly large design space to a finite set of parametrizable, architectural alternatives, a processor library with five different processor types for each single image processing algorithm was developed. The library contains different *templates* of processors, whereas the datapath $\mathcal{DP}_j$ of each processor is customized to the properties Θ_i of any image processing algorithm to be executed by one of the processors.

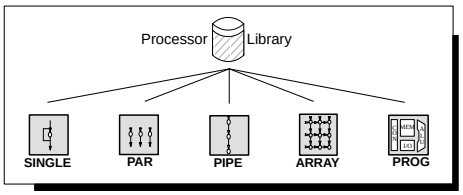

Fig. 5: Different processor types

Fig. 5 shows the different processor types, called SINGLE, PAR, PIPE, ARRAY, and PROG. For example, SINGLE denotes a processor type, performing all operations of the algorithm sequentially with one dedicated processing element (PE), thus leading to a low area expense at the cost of a large execution time. In contrast to this, the processors of type PAR and PIPE provide the possibility to execute the operations of the algorithm in parallel or in a pipelined mode. Furthermore, the processor of type ARRAY provides both, parallel and pipelined execution of operations by a two dimensional grid of processing elements. Additionally, a flexible and programmable processor type called PROG is supported, preferably suitable for performing irregular medium–level algorithms.

Since we are interested in the best combination of application–specific and programmable processors, it is necessary to characterize each single processor alternative by some performance and expense attributes. The most relevant attributes of any processor p_j or associated datapath $\mathcal{DP}_j$ with respect to an optimization of the overall multiprocessor system are 1) the ex-

ecution time $\tau_{i,j}$ for one specific algorithm $\mathcal{ALGO}(m_i)$ to be executed by the datapath, and 2) the area expense of the datapath itself. Clearly, the area expense can be estimated from the number and types of processing elements (PE) or building blocks (BB) of each datapath. This leads to the definition of a parametrizable datapath $\mathcal{DP}_j$:

$$\mathcal{DP}_j \stackrel{\text{def}}{=} \{\tau_{i,j}, A_j, \{PE_1, \ldots, PE_K\}, \{BB_1, \ldots, BB_L\}, \ldots\}$$

We experienced, that all these attributes can be calculated deterministically in advance, depending on the parameter set Θ_i of the algorithm class and on the processor type. This is true, especially in case of regular low–level algorithms. For example, the execution time τ for a filtering algorithm like FIR is given by:

$$\tau = \begin{cases} 1 + K_H \cdot K_W \cdot T_H \cdot T_W & : \text{SINGLE} \\ 1 + K_H \cdot T_W \cdot T_H & : \text{PAR} \\ 1 + K_H + K_W \cdot T_H \cdot T_W & : \text{PIPE} \\ 1 + K_H + T_W \cdot T_H + 2 \cdot (K_W - 1) & : \text{ARRAY} \end{cases}$$

The expense of the four corresponding datapaths is given in Table 2 in terms of number / type of processing elements (PE) and building blocks (BB). As can be seen, these datapath attributes, necessary to estimate the area expense, can also be calculated deterministically with respect to the parameters Θ_i of the algorithm class.

	PE		BB		
TYPE	MULADD	ADD	MUL	ADD	REG
SINGLE	1		1	1	1
PAR	K_W		K_W		K_W
PIPE	K_H	1	K_H	$1 + K_H$	$1 + K_H$
ARRAY	$K_W K_H$	K_W	$K_W K_H$	$K_W(1 + K_H)$	$2K_W K_H + 2K_W - K_H$

Table 2: Number of PEs and BBs for a FIR datapath

Consider, for example a filtering algorithm FIR with an image block size of 8*8 pixels and a kernel window of 3*3 filter coefficients, i.e. $\Theta = \{8, 8, 3, 3\}$.

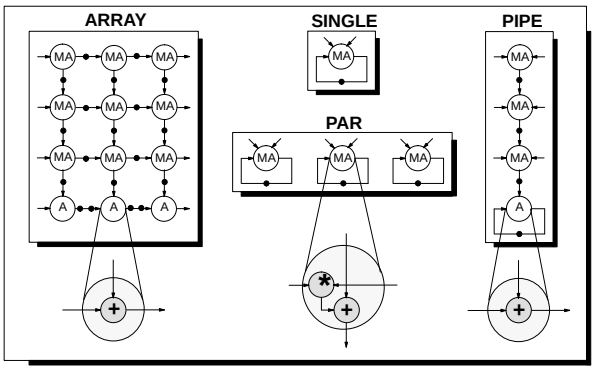

Fig. 6: Processor architectures for FIR filtering

Then, the alternative processor architectures of type SINGLE, PAR, PIPE, and ARRAY are sketched as schematic on a register transfer level in Fig. 6 with the two different PEs (MA, A) emphasized. For the other algorithm classes, similar architectures can be derived; but they will not be presented here.

3.2.3 Assignment

The last step of the design space exploration is denoted *assignment*, since, from a formal point of view, it can be regarded as a one–to–many mapping f from a task's algorithm $\mathcal{ALGO}(m_i)$ to a finite set of parametrizable datapaths $\mathcal{DP}_j$ as follows:

$$f_{assign} : \mathcal{ALGO}(m_i) \stackrel{\text{type}}{\longrightarrow} \mathcal{DP}_j(\theta_i, \texttt{type})$$

This mapping is performed with respect to the parameter set Θ_i of the algorithm class and the different processor types (SINGLE, PAR, PIPE, ARRAY) of the supported processor library.

3.3 Optimization

After the possible design space for a given image processing scheme has been explored based on the processor library, it is necessary to select from the large amount of feasible assignments of single tasks or algorithms to processor types the one which leads to an optimal overall multiprocessor system. Here, optimality means to derive a heterogeneous systems, meeting the real–time constraint and achieving the highest possible computational and throughput rate with an area expense as low as possible. So, for each task it has to be determined, which is the most suitable processor type, what is the best temporal order concerning task execution and data transfer, taking into consideration the data precedence between the tasks, the different availibility of input / output data depending on the processor type, and the non overlapping usage of processors and buses. It can be shown, that this leads to a combinatorial optimization problem. One efficient way to solve this kind of optimization problems is derive a formulation using mixed integer linear programming (MILP). That is the reason why we chose MILP as mathematical framework. Nevertheless, it is necessary to transform the given combinatorial optimization problem to a MILP, as described next.

3.3.1 Transformation to MILP

Mixed integer linear programming problems either minimize or maximize an objective function z of variables $x_j \in \vec{x}$, subject to a set of linear equality and inequality constraints $\sum a_{i,j} \cdot x_j = b_i$. MILPs differ from general LPs, because some of the variables are restricted to be of integer range. This can be written:

$$z = \min\{\vec{c}_I \cdot \vec{x}_I + \vec{c}_R \cdot \vec{x}_R \mid \vec{x}_I \in \mathcal{P}_I, \vec{x}_R \in \mathcal{P}_R\}$$

where

$$\mathcal{P}_I = \{A_I \cdot \vec{x}_I = \vec{b}_I, \vec{x}_I \in \mathbb{N}_+^m\}$$
$$\mathcal{P}_R = \{A_R \cdot \vec{x}_R = \vec{b}_R, \vec{x}_R \in \mathbb{R}_+^n\}$$

The use of MILP in order to tackle the scheduling, allocation and binding problem in the domain of multiprocessor synthesis can already be fond in [7]. But, we extended these models by two main aspects: First, periodic and overlapped processing of tasks are taken into consideration, which is of great importance, especially for real–time image processing. Furthermore, by using a parametrizable library with five different processor types, it becomes possible to consider a large set of alternative architectures. Since we aim at transforming the combinatorial optimization problem to a MILP, we are concerned with finding i) a set of variables, ii) a system of linear constraints or restrictions, and iii) an appropriate cost function for the MILP.

Concerning the variables $\vec{x}$ three main groups can be found:

- binary-valued decision variables
  $\vec{x}_B = [\ldots, x_{i,j}, \ldots]$
- integer-valued variables
  $\vec{x}_I = [\ldots, Y, B, \ldots]$
- real-valued timing/area variables
  $\vec{x}_R = [\ldots, s_i, e_i, \ldots]$

For example, the binary decision variable $x_{i,j}$ models the assignment of task m_i to processor p_j, the integer valued variable Y represents the number of different processors, and so on. Concerning the restrictions $A \cdot \vec{x} = \vec{b}$, the following groups can be distinguished:

- task–processor restrictions
  task execution start/end, etc.
- transfer–bus restrictions
  transfer execution start/end, etc.
- general restrictions
  area expense, number of processors, etc.

For example, the *transfer execution end* time $ce_{i,k}$ can be expressed as a linear constraint by means of the number of data items $D_{i,k}$ to be transmitted from task m_i to task m_k, the transfer rate B_R of the bus, and whether it is a remote ($\gamma^R_{i,k} = 1$) or a local ($\gamma^L_{i,k} = 1$) transfer ($ce_{i,k} = cs_{i,k} + \gamma^R_{i,k} \cdot \frac{D_{i,k}}{B_R} + \gamma^L_{i,k} \cdot \frac{D_{i,k}}{B_L}$). Finally, an appropriate cost function z has to be determined: It takes into consideration the computation time (latency) T, the computation period P in case of overlapped and periodic execution, the number of processors Y and of buses B as well as the area expense A:

$$z = c_T \cdot T + c_P \cdot P + c_A \cdot A + c_Y \cdot Y + c_B \cdot B$$

By means of individual weights $c_T, c_P, \ldots, c_B$ different goals of the designer can be considered. A detailled description of our MILP model, containing all variables, restrictions, and their meanings can be found in [8], and thus, will not be discussed here.

4 Case Study: Video Encoding

To demonstrate the feasibility of the proposed system level methodology, the hybrid video codec scheme H.261 [1] was chosen a as case study. The H.261 scheme is used for video telephone or video conferencing. It consists of several regular low level tasks like motion estimation by block matching algorithm (BMA), discrete cosine transform and its inverse (DCT, IDCT), and finite impulse response filtering (FIR). On the other hand, irregular medium level tasks like quantization and its inverse (Q, IQ), and variable / runlength coding (COD) belong to the H.261 hybrid coding scheme. The corresponding task graph is shown in Fig. 7a, the possible assignments of the $M = 12$ tasks to $N = 20$ processors, based on the processor library with four different, application–specific processor types (SINGLE, PAR, PIPE, ARRAY) and two programmable processors (PROG) can be seen in Fig. 7b. Based on these assignments the MILP model was derived, leading to 117 variables and 208 restrictions. We experienced, that by use of *branch-and-bound* the LP solver of the NAG [1] Fortran Library delivers the optimal solution within a few cpu minutes.

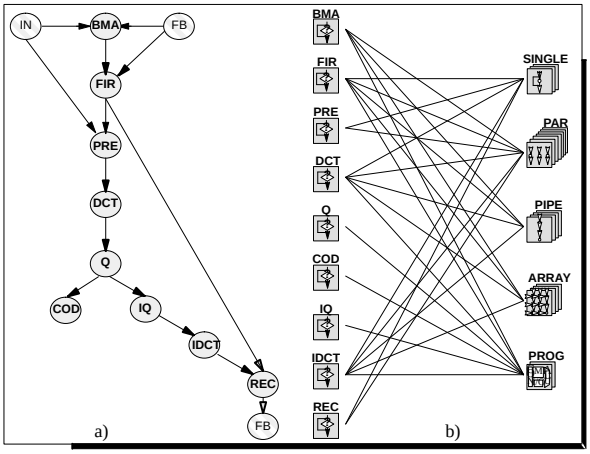

Fig. 7: Task Graph for H.261 and Possible Assignments

Four heterogeneous multiprocessor systems, resulting from the optimization, are sketched in Table 3, assuming different design priorities with respect to the weights of the system parameters A, T, P, Y.

Priority		A≻T,P≻Y	P≻A≻T≻Y	P≻T≻A≻Y	Y≻A≻T,P
Area Expense	A [#trans.]	172902	726930	605820	184190
Computation Time	T [#cycles]	7100	4514	4994	6134
Computation Period	P [#cycles]	6324	1073	1073	5581
Processors	Y [#proc.]	3	7	8	3

Table 3: Performance and area expense

In case the designers primary goal is for example, to derive a heterogeneous system with minimum area expense A and additionally small latency T and computation period P, i.e. $A \succ T, P \succ Y$, the optimal assignment is shown in Fig. 8a:

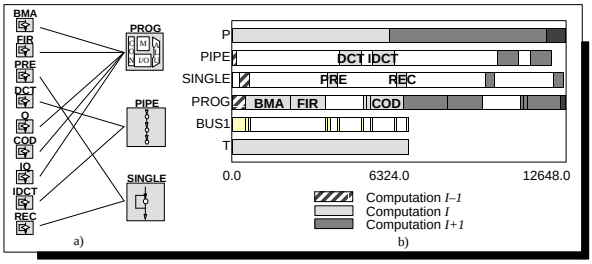

Fig. 8: Optimal assignment, periodic-overlapped schedule

The corresponding architecture consists of $Y = 3$ processors: one of type SINGLE, PIPE, and PROG, leading to an area expense of $A = 172902$ transistors with an achievable throughput rate or computation period of $P = 6324$ clock cycles. The periodic and overlapped multiprocessor schedule is sketched in Fig. 8b, whereas the computation of each macro block (16*16 pixels) is shaded in a different color. There is an overlapping computation of two macro blocks for the whole multiprocessor system, with white areas indicating processor idle times. The four optimal heterogeneous systems given in Table 3, derived by solving

[1] The Numerical Algorithm Group, Limited

the corresponding MILP under different design priorities, are again visualized in the area–time plane:

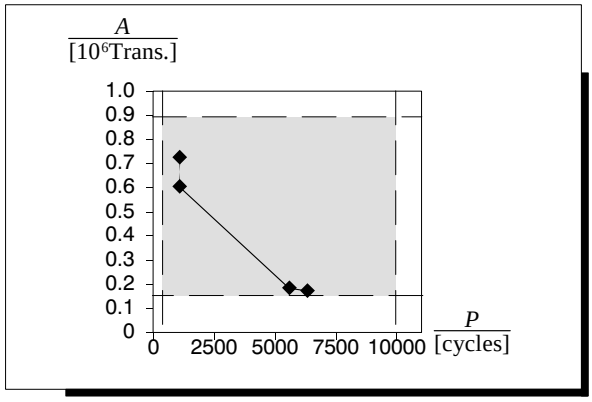

Fig. 9: Design space concerning A, P

Fig. 9 shows the four solutions in terms of area expense A and and achievable throughput rate P. The dashed rectangle indicates the possible design space with respect to the supported processor library and the real–time constraint of the video encoder example. By giving different weights to the parameters A, T, P, Y of the cost function, either time or area optimal solutions can be derived.

5 Implementation

The presented system level design methodology is currently under development as a prototype CAD tool in COMMON LISP/CLOS [2] on Sparc Stations. Main parts of the CAD system are shown in Fig. 10:

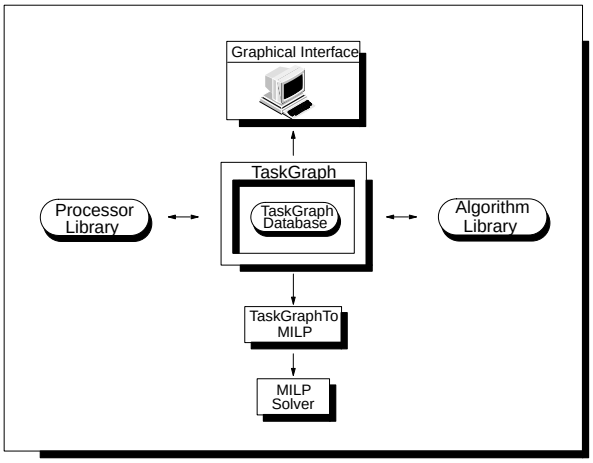

Fig. 10: Overview of the CAD tool

It starts from a user's specification of the composite DSP scheme. With respect to the algorithm and processor library, all feasible assignments are derived and transfered to a specialized task graph, serving as central database. The transformation to a MILP formulation is performed automatically, taking the designer's goal into consideration. Finally, the MILP solver is invoked. The results of the optimization are again transfered to the task graph data base. A graphical user interface (GUI) developed in LISP and based on CLUE/CLX [3] provides the possibility to visualize the results by means of diagrams and gannt charts.

6 Conclusion

In this paper, a system level design methodology for the optimization of heterogeneous multiprocessors is presented. The main task is to find the best combination of application–specific and programmable processors for composite DSP schemes of image processing algorithms. The approach chosen to solve this combinatorial optimization problem is based upon mixed integer linear programming, mainly extended to handle periodic and overlapped execution of tasks, taking into consideration the availibility of input and output data. First encouraging results could already be derived for an application example taken from visual telephony, proving the general feasibility of the developed system level methodology.

Acknowledgements

This work is supported by the Deutsche Forschungsgemeinschaft, contract number Pi–169/5.

References

[1] CCITT Study Group XV: Recommendation H.261, "Video Codec for Audiovisual Services at p x 64 kbit/s, Report R37, Geneva, July 1990.

[2] ISO–IEC IS 11172, "Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s", 1993.

[3] W. Rosenstiel, H. Krämer, "Scheduling and Assignment in High Level Synthesis", in *High Level VLSI Synthesis*, R. Composano, W. Wolf, editors, Kluwer Academic Publishers, pp. 355-382, 1991.

[4] S.Y. Kung, S.N. Jean, "A VLSI Array Compiler System (VACS) for Array Design", R.W. Brodersen, H.S. Moscovitz, editors, *VLSI Signal Processing III*, Chapter 45, pp. 495 - 508, IEEE Press, New York, 1988.

[5] J. Rabbaey, H. de Man, J. Vanhoof, G. Goosens, F. Cathoor, "CATHEDRAL II: A Synthesis System for Multiprocessor DSP Systems", in Silicon Compilation, Addison-Wesley, pp. 311-360, 1988.

[6] J. Vanhoof, I. Bolsens, G. Goosens, H. de Man, K. Rompaey, "High level synthesis for real-time digital signal processing", Kluwer Academic Press, 1993.

[7] S. Prakash, A.C. Parker, "Synthesis of Application Specific Heterogeneous Multiprocessor Systems", *Journal of Parallel and Distributed Computing*, no. 16, pp. 338 - 351, December 1992.

[8] M. Schwiegershausen, M. Schönfeld, P. Pirsch, "Mapping complex image processing algorithms onto heterogeneous multiprocessors regarding architecture dependent parameters", in *Algorithms and Parallel VLSI Architectures III*, M. Moonen, F. Catthoor, editors, Chapter 30, pp. 353 - 364, Elsevier, Amsterdam, 1995.

[2] CLOS Specification, ANSI, X3J13

[3] Common Lisp User Interface Environment/X