Classification, Customization, and Characterization: Using MILP for Task Allocation and Scheduling

Abhijit Davare
Jike Chong
Qi Zhu
Douglas Michael Densmore
Alberto L. Sangiovanni-Vincentelli

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2006-166
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-166.html

December 11, 2006

Acknowledgement

This work is supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF award #CCR-0225610), the State of California Micro Program, and the following companies: Agilent, DGIST, General Motors, Hewlett Packard, Infineon, Microsoft, National Instruments, and Toyota. This work is also supported in part by the MARCO-sponsored Gigascale Systems Research Center (GSRC).

Classification, Customization, and Characterization: Using MILP for Task Allocation and Scheduling

Abhijit Davare, Jike Chong, Qi Zhu, Douglas Densmore, Alberto Sangiovanni-Vincentelli
$\{$davare, jike, zhuqi, densmore, alberto$\}$@eecs.berkeley.edu

Abstract

Task allocation and scheduling for heterogeneous multi-core platforms must be automated for such platforms to be successful. Techniques such as Mixed Integer Linear Programming (MILP) provide the ability to easily customize the allocation and scheduling problem to application or platform-specific peculiarities. The representation of the core problem in a MILP form has a large impact on the solution time required. In this paper, we investigate a variety of such representations and propose a taxonomy for them. A promising representation is chosen with extensive computational characterization. The MILP formulation is customized for a multimedia case study involving the deployment of a Motion JPEG encoder application onto a Xilinx Virtex II Pro FPGA platform. We demonstrate that our approach can produce solutions that are competitive with manual designs.

1 Introduction

Applications for embedded systems are becoming more complex and are increasingly being realized with software deployed on heterogeneous and highly parallel architectural platforms. Even though these platforms may provide better performance, they greatly complicate the programming effort – a significant road block for their adoption.

The design flow we consider starts with a concurrent description of the application and a profiling of the architectural platform in terms of computational capabilities and communication costs. The application description used in this paper is a statically schedulable dataflow specification, such as cyclo-static [16] dataflow. Such descriptions are commonly used for the multimedia applications we target and can be automatically transformed into acyclic data precedence graphs.

The core problem to be solved is to map this weighted directed acyclic graph (DAG) representing the application onto a set of architectural nodes. In the application DAG, nodes represent tasks while edges represent data precedence relationships. The architecture is represented with a weighted directed graph where nodes are processing elements (PEs). Edges that represent communication channels may be added explicitly to the architecture graph if connectivity between PEs is restricted, otherwise, a fully-connected graph is assumed. The execution time for each task on each processor is fixed and given. The amount of communication between each application task is given by weights on the application graph edges. The objective is to allocate and schedule the tasks onto the PEs such that the completion time – or makespan – is minimized.

This paper has four main contributions. First, a classification of existing MILP representations for this problem into a taxonomy. Second, based on the taxonomy, a core MILP formulation and useful customizations. Third, computational characterization and a comparison of our approach with a competing approach. Finally, a representative case study that illustrates the ability or our approach to produce competitive designs in terms of makespan and area.

2 Prior Work

The scheduling problem we consider is a *generalization* of $R|prec|C_{max}$ [8] and is strongly NP-hard. R refers to the usage of multiple heterogeneous PEs with unrelated processing times, $prec$ indicates that the application description includes precedence constraints, and C_{max} indicates the objective is to minimize the makespan.

For the special case of $R||C_{max}$ (no precedence constraints), there exist polynomial-time approximation algorithms that can guarantee a solution within a factor of 2 of the optimal [19]. No poly-time approximation algorithm exists that can provide a solution for $R||C_{max}$ within 1.5 times the optimal, unless $P = NP$ [12]. If precedence constraints are added, there are no known good approximation results; an overview of related work for $R|prec|C_{max}$ is provided in [10]. A comprehensive listing of known lower and upper approximation bounds for a variety of scheduling and allocation problems can be found in [4] while an overview of heuristics is given in [11].

Even though heuristics can often provide good solutions in a short amount of time, they do not provide bounds on

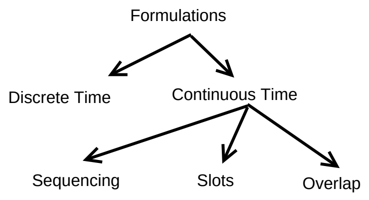

Figure 1. Taxonomy of MILP Approaches

solution quality. More importantly, they are brittle with respect to changes in the problem assumptions. For instance, partial solutions and side constraints are typically difficult to add to most heuristics without sacrifices in effectiveness. Approximation algorithms do provide bounds, but the analysis applied to produce these bounds is even less resilient to problem changes. Consequently, even though heuristics and approximation algorithms excel at clearly defined problems, their applicability is limited within a design flow where platform-specific constraints are needed.

An alternative to approximation algorithms and heuristics is to use mathematical programming techniques such as MILP. These techniques are much easier to customize, since application or platform-specific constraints can be added as required. Branch-and-bound solution techniques for MILPs which utilize linear programming (LP) relaxations also provide a lower and upper bound on the desired cost function at each step of the solution process. This allows us to trade off solution time and quality.

3 MILP Taxonomy

Solution time for MILP instances is strongly affected by the representation used for the core allocation and scheduling problem. We observe that the effective encoding of task precedence relationships is key not only for approximation algorithms as mentioned in Section 2, but also for MILP representations. Along these lines, we propose a taxonomy of known MILP representations in Figure 1.

Discrete time approaches introduce a variable for each instant of time on each PE. The resultant scheduling constraint requires that each such time instant be allocated to at most one task. The advantage of this method is that the formulation can easily be constrained to use only integer or binary variables. A rich variety of SAT and Pseudo-Boolean techniques can be utilized to solve these problems. However, this formulation has a significant drawback: the number of time variables introduced can quickly become very large, especially if diverse task execution times are present.

Continuous time approaches represent time with real-valued variables in the formulation. Vastly different execution times can easily be handled by these approaches, but the choice of variables and constraints used to specify a correct scheduling on each PE becomes critical in determining performance [9].

Sequencing: Variables are used to indicate sequencing to schedule tasks on PEs. These sequencing variables indicate whether a task is executed after another task on the same PE [1, 3]. This choice of variables can be viewed as a straightforward extension of the well-known formulations used in uniprocessor scheduling [15]. Typically, a large number of constraints or variables is required to enforce the scheduling requirements on each PE. Many of these constraints can be attributed to the linearization of bilinear terms [13].

Slots: This method uses explicit slots on each PE [14, 5] to which at most one task can be allocated. The start and finish time for each slot is not fixed a priori. With slots, the scheduling constraints between tasks on each PE become simpler to represent. However, since the exact number of slots on each PE is unknown, a conservative amount need to be used. As a result, this approach may suffer from variable blow-up if the typical number of tasks allocated to each PE is large.

Overlap: Variables are used to indicate overlap (independent of PE assignment) in the execution of tasks [17, 18, 20]. Constraints that prevent overlap on the tasks allocated to each PE are used to enforce scheduling. Since the scheduling constraints can be expressed succinctly, this type of formulation scales well with respect to variables and constraints than the formulations in the other categories.

In this paper, we focus on continuous-time MILP formulations that use overlap variables, since this category seems the most promising for generating problems with fewer constraints and variables.

4 MILP Approach

In this section, our core formulation and customizations will be described in detail.

4.1 Core Formulation

Let $\mathbf{F}$ represent the set of tasks in the application DAG while $\mathbf{E} \subset \mathbf{F} \times \mathbf{F}$ represents the set of communication edges. The set $\mathbf{A}$ indicates the set of architectural PEs. The parameter $t \in \mathbb{R}^{\mathbf{F} \times \mathbf{A}}$ specifies the execution time of each task on each PE.

The variable $d \in \mathbb{B}^{\mathbf{F} \times \mathbf{A}}$ indicates if a task is mapped to a PE. Variables $s \in \mathbb{R}^{\mathbf{F}}$ and $f \in \mathbb{R}^{\mathbf{F}}$ indicates the start and finish times respectively for each task. $o \in \mathbb{B}^{\mathbf{F} \times \mathbf{F}}$ is a variable which is used to determine overlap in execution times between a pair of tasks.

$$
\begin{array}{llll}
min & \max_{i \in \mathbf{F}} f_i & & (1)\\
s.t. & \sum_{x \in \mathbf{A}} d_{ix} = 1 & \forall i \in \mathbf{F} & (2)\\
& f_i \le s_j & \forall (i,j) \in \mathbf{E} & (3)\\
& f_i = s_i + \sum_{x \in \mathbf{A}} (t_{ix} d_{ix}) & \forall i \in \mathbf{F} & (4)\\
& f_j - s_i \le M o_{ij} & \forall i,j \in \mathbf{F}, i \ne j & (5)\\
& o_{ij} + o_{ji} + d_{ix} + d_{jx} \le 3 & \forall i,j \in \mathbf{F}, x \in \mathbf{A}, i \ne j & (6)
\end{array}
$$

The objective function in (1) minimizes the maximum finish time over all tasks. Constraint 2 (C2) ensures that each task is mapped onto exactly one PE. C3 requires that the precedence relationships between edges in the application DAG hold. C4 relates the start times and finish times of each task based on the execution time of the task on the appropriate PE. C5 ensures that if a task j finishes after task i begins, the corresponding variable o_{ij} is set to 1. In this constraint, M represents a large constant, which can be no less than the maximum finish time. Finally, C6 is a particularly elegant means of ensuring that no two tasks mapped onto the same PE may overlap. In this constraint, the sum $o_{ij} + o_{ji}$ has a value of 2 iff the executions of the two tasks i and j overlap. It is these last two sets of constraints that define task overlap and prevent it on any single PE. Note that C6 only needs to be defined over $i, j \in \mathbf{F}$ such that neither i nor j are in each other's transitive fan-out (TFO). For all other cases, the sum $o_{ij} + o_{ji}$ must be 1.

4.2 Customizing the Formulation

The core formulation does not support communication cost, restricted architectural topologies, partially specified task allocation on the platform, and real-time requirements on portions of the application. Of these, the first three are crucial for the case study we target in Section 6.

The additional set $\mathbf{C} \subseteq \mathbf{A} \times \mathbf{A}$ represents the directed edges between PEs. The parameter $c \in \mathbb{R}^{\mathbf{E}}$ denotes the communication cost for each edge in the application DAG. The parameter $n \in \mathbb{B}^{\mathbf{F} \times \mathbf{F}}$ indicates whether two tasks are required to be mapped onto the same PE. Parameter $e \in \mathbb{B}^{\mathbf{F} \times \mathbf{A}}$ indicates if a particular task must be mapped onto a particular PE. The variables $r \in \mathbb{R}^{\mathbf{F}}$ and $w \in \mathbb{R}^{\mathbf{F}}$ represent the reading and writing time required for each task.

$$
\begin{array}{lll}
f_i = s_i + \sum_{x \in \mathbf{A}} (t_{ix} d_{ix}) + r_i + w_i & \forall i \in \mathbf{F} & (7)\\
r_i \ge \sum_{(j,i) \in \mathbf{E}} c_{ji} (d_{ix} - d_{jx}) & \forall i \in \mathbf{F}, x \in \mathbf{A} & (8)\\
w_i \ge \sum_{(i,j) \in \mathbf{E}} c_{ij} (d_{ix} - d_{jx}) & \forall i \in \mathbf{F}, x \in \mathbf{A} & (9)\\
d_{iy} + d_{jz} \le 1 & \forall (i,j) \in \mathbf{E}, (y,z) \notin \mathbf{C} & (10)\\
d_{ix} \ge e_{ix} & \forall i \in \mathbf{F}, x \in \mathbf{A} & (11)\\
n_{ij} - 1 \le d_{ix} - d_{jx} \le 1 - n_{ij} & \forall i,j \in \mathbf{F}, x \in \mathbf{A} & (12)
\end{array}
$$

C7 replaces C4 from the core formulation and considers the reading and writing time for each task. C8 charges time for reading iff the predecessor task is assigned to a different PE. Likewise, C9 charges the corresponding write time. C10 ensures that the mapping conforms to the restricted architectural topology. C11 is a forcing constraint that allows some allocations to be fixed. Finally, C12 restricts certain pairs of tasks to be allocated to the same PE. This is useful when considering applications derived from dataflow specifications, where multiple invokations of the same actor may be constrained to occur on a single PE.

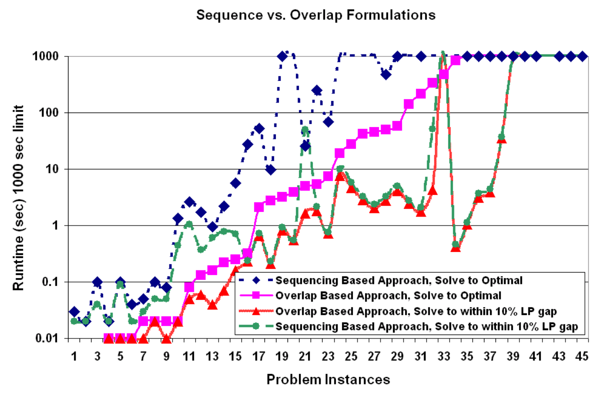

Figure 2. Sequence vs. Overlap Runtime

5 Characterizing the Formulation

In this section, we compare our formulation against the sequence-based formulation and identify characteristics of problem instances that affect the runtime of the MILP formulation.

The experimental setup involves coding the sequence-based formulation from [3] and our core formulation in AMPL [7] and evaluating them with a set of 45 test cases. The test cases were generated with the TGFF [6] tool with three random seeds. Five problem sizes, ranging from 10 to 50 tasks, were generated from each seed. Each task graph was allocated to different numbers of PEs to keep the average task/PE ratio the same. The test cases were solved using CPLEX 9.1.2 on 2.8GHz Linux machines with 2GB of memory under a time limit of 1000 seconds.

5.1 Comparison: Sequencing vs. Overlap

For the 45 test cases, on average, our overlap-based formulation has 30% more variables than the sequence-based formulation. However, our formulation also has 63% fewer constraints, which substantially reduces overall problem size.

For solving problems to optimality in a balanced branch-and-bound exploration of the solution space, the sequence-based approach is an *order of magnitude* slower than our approach, as shown in Figure 2. LP relaxations of the problems are usually quite tight, often within 10-15% of the optimal value. This means that good lower bounds can be obtained in polynomial time for these problem instances. For instances that could not be solved to optimality within the time limit, feasible solutions within 14% of optimal were obtained on average.

If a solution within 10% of the optimal is sufficient, we can bias the branch-and-bound exploration to find feasible solutions. These results are also plotted in Figure 2 and show that solution time can be decreased by 1-2 orders of magnitude with biasing and a 10% optimality gap. For very few cases, feasibility biasing may increase solution time.

5.2 Factors Influencing Solution Time

Solution time is typically analyzed with respect to the number of tasks, the number of constraints or the number of PEs for a given problem instance. None of these factors is a good indicator of solution time for this formulation. For a problem with same number of tasks, as the number of PEs available decreases, we discover a counterintuitive trend: the number of constraints (and variables) drops, but the runtime increases.

The rising solution time for test cases with fewer PEs and constraints can be explained with three observations. First, when there are relatively fewer PEs, more unrelated tasks (tasks not in each other's TFOs) have to be sequentialized onto each PE. A formulation that relies on binary variables and big M constants to enforce non-overlapping of tasks (C5) has a weaker LP lower bound with more tasks/PE. Secondly, when many unrelated tasks have similar processing times, many feasible solutions have similar makespans, this prevents effective pruning of the branch and bound tree based on known feasible solution upper bounds. Thirdly, the number of feasible permutations of task ordering explodes with more tasks/PE. If we have k unrelated tasks allocated on the same processor, many of the $k!$ permutations must be considered in the branch and bound tree. The total number of permutations increases as a function of the maximum clique (fully connected component) of the inverse application graph. Making more PEs available disperses unrelated tasks - fragmenting the cliques and improving the LP lower bound.

6 Case Study

We now turn our attention to demonstrating the applicability of our customized MILP formulation on a case study. The chosen application is a Motion JPEG encoder. The architectural platform we consider contains soft-core processors and processing elements on a Xilinx Virtex II Pro FPGA fabric. For various manual and automated mappings, we compare the performance in terms of system throughput and area utilization. For applications derived from dataflow specifications, the makespan of an unrolled dataflow graph is equivalent to throughput.

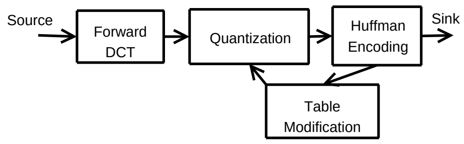

Figure 3. Motion JPEG Encoder

6.1 Motion JPEG Application

The motion JPEG encoder application, shown in Figure 3, carries out video encoding without inter-frame compression. A motion JPEG encoder is commonly implemented in consumer and security cameras as well as high-resolution video editing.

The application compresses a stream of raw image data in 4:4:4 format as per the JPEG [21] standard and emits a stream of compressed JPEG images. This application was chosen since it is relatively simple, yet representative of a wide class of multimedia applications.

The input to the JPEG encoder application is a stream of 8x8 blocks coded in the YCbCr color space in scan order. DCT, quantization, and Huffman encoding is carried out on these blocks. Finally, the actual compression ratio is compared against the desired ratio, and the quantization coefficients are changed accordingly. Figure 3 gives an overview of this flow.

6.2 Xilinx Virtex II Pro Platform

The Xilinx Virtex II Pro FPGA provides a set of hard and soft processor cores, bus and FIFO interfaces between cores, and the ability to create customized IP blocks. This case study uses a 2VP30 part on Xilinx XUP board with a maximum frequency of 100 MHz.

The uBlaze 32-bit soft processor is a standard RISC-based engine. All peripherals are interfaced using the On-chip Peripheral Bus (OPB).

Fast Simplex Links (FSLs) are a low-overhead method of communication between uBlaze cores and the fabric. FSL depth can range from 1 to 8,192 entries, each of which may be 4 bytes in width. Reads and writes to the FSL FIFOs from the uBlaze take only a single cycle. Both blocking and non-blocking read/write access to FSLs is provided.

Finally, processing elements can be directly synthesized onto the fabric. For this case study, we use a DCT-specific PE with FSL interfaces created using the XPilot [2] synthesis system.

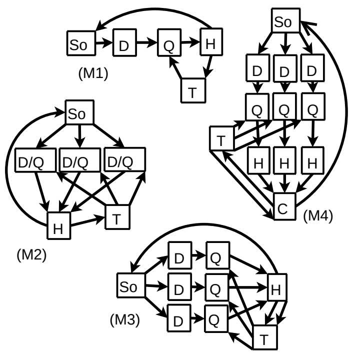

Figure 4. Topologies of Manual Designs

6.3 Manual Design Space Exploration

The goals in manual design space exploration are to utilize various numbers of uBlaze processors and DCT-specific PEs to maximize the throughput of the Motion-JPEG application. A nominal frame size of 96x72 is assumed for all implementations. We start from a baseline topology where the entire application is mapped onto a single uBlaze processor. As additional PEs are utilized, portions of the application are migrated to these PEs to improve throughput.

The PEs in the various designs are connected with FSL queues that are accessed in blocking-read, blocking-write mode. Data is fed to and retrieved from the device with a 100 Mbps Ethernet connection. An Ethernet MAC device is instantiated on the fabric to handle this communication. One of the uBlaze PEs in each design is designated as the I/O processor and connects to the Ethernet device. In addition, this uBlaze is connected to peripherals to allow for debugging and performance measurement. In all designs except the baseline topology, a single uBlaze is reserved for quantization table update.

The different manual designs obtained are shown in Figure 4. The blocks used include the data source (So), DCT (D), quantization (Q), Huffman encoder (H), and table update (T). A combiner (C) is necessary when the Huffman block is split into 3 parts. Salient characteristics of each implementation – the number of uBlaze and DCT PEs used, the frames processed per second, and the area (% slices occupied on the FPGA) – are summarized in Table 1. Designs $M1D$ and $M3D$ are obtained from $M1$ and $M3$ respectively by substituting uBlazes with DCT PEs where possible.

The manual designs exploit the task-level and data-level parallelism in the application. The designs first attempt to use task-level parallelism between the different stages and the exploit the natural data-level parallelism between the three color channels.

Design	uBlaze	DCT	fps	Area
Base	1	0	26.5	21%
M1	5	0	51.1	39%
M1D	4	1	72.0	53%
M2	6	0	85.1	47%
M3	9	0	85.3	62%
M3D	6	3	85.6	94%
M4	12	0	148.8	83%

Table 1. Manual Designs

6.4 Automated Design Space Exploration

Automated Design Space Exploration uses the MILP formulation developed in Section 4 to determine the scheduling and allocation for tasks in the case study. The aim is to show that the cost model in the formulation accurately captures the design space and can be used to implement competitive designs.

The first step is to create a representation for the application which identifies the maximal amount of available concurrency. Since we would like to compare against the manual implementations, we create a task representation which extracts no more concurrency than is utilized in the manual designs. We also reserve a separate uBlaze for the table update portion of the application, just as in the manual designs. Note that this restriction can be relaxed to obtain higher quality automated designs.

The next step is to characterize the application so that the task execution times (the t parameters) can be obtained. For the uBlaze processors, the cycle times obtained from the timer peripheral are as follows: Source - 200, DCT - 4,760, Quant - 2,572, Huffman - 3,442, Combiner - 2,542. On the DCT PE, the DCT task has a latency of 328 cycles.

If these parameters are used in our MILP formulation, any legal solution produces a static estimate for throughput. The accuracy of this estimate is important in determining the effectiveness of an automated approach. We compare the estimates for the base design and the 6 manual designs from Section 6.3 against the actual implementation results obtained from the development board.

The makespan estimated from the formulation is, on average, within 5% of the execution time measured on the development board. Most of the predictions overestimate the makespan, since the formulation does not consider simultaneity between reads and writes on each FIFO.

With this accurate model of the design space, we can automatically evaluate a number of different solutions from the MILP formulation. Based on the characteristics of the manual designs, we picked three promising MILP solutions and implemented them on the development board. These three automated solutions (A1, A2, and A3) use 3, 5, and 8 uBlazes and were obtained with a 100s time limit on the solver. The MILP solution also confirms that design M4 is optimal given the chosen granularity of the task graph.

Both the manual and automated solutions can be plotted

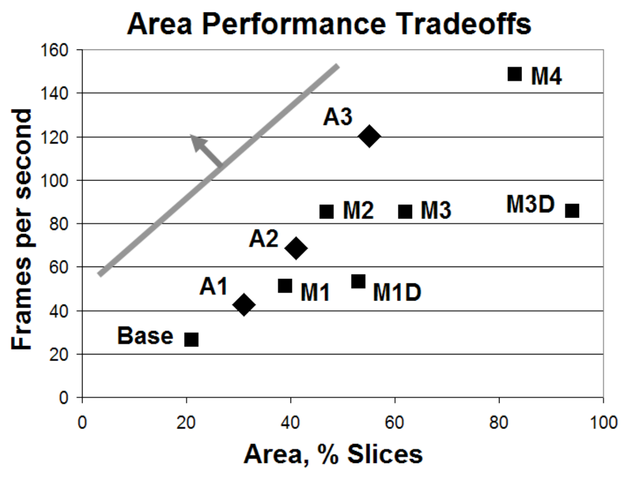

Figure 5. Manual vs. Automated Designs

in terms of frame rate vs. FPGA slices consumed - which is roughly proportional to the number of uBlazes and DCT PEs used. Figure 5 shows this tradeoff and indicates that the automated designs do indeed result in competitive implementations that lie on the Pareto curve.

7 Conclusions and Future Work

In this work, we considered a number of MILP approaches for solving the task allocation and scheduling problem. Based on their treatment of precedence constraints, a taxonomy of known MILP representations was proposed, and the most promising core formulation was selected. Extensions were then added to customize the formulation for our needs. With extensive computational testing, we showed that our overlap-based formulation has better solution time than a competing sequence-based formulation, and demonstrated that tight lower bounds could be obtained in polynomial time. We also identified key metrics for determining the difficulty of problem instances - demonstrating that for a given application, larger platforms actually decrease solution time. Our formulation was applied to a case study that considers the deployment of an MJPEG application on a Xilinx FPGA platform. We showed that our formulation can accurately predict the performance of the system and quickly produce solutions that are competitive with manual designs.

In the future, we plan to further extend the customizations used here to enable the targeting of more complex platforms. For instance, by handling the allocation of distributed memory. Also, we would like to consider more specialized solution techniques. In particular, the addition of constraints corresponding to critical paths during the branch-and-bound process seems promising.

References

[1] A. Bender. MILP based task mapping for heterogeneous multiprocessor system. In *Proceedings of EURO-DAC*, september 1996.

[2] D. Chen, J. Cong, Y. Fan, G. Han, W. Jiang, and Z. Zhang. xPilot: A platform-based behavioral synthesis system. In *SRC TechCon'05*, November 2005.

[3] P. E. Coll, C. E. Ribeiro, and C. C. D. Souza. Multiprocessor scheduling under precedence constraints: Polyhedral results. Technical Report 752, Opt. Online, October 12, 2003.

[4] P. Crescenzi, V. Kann, M. Halldórsson, M. Karpinski, and G. Woeginger. A compendium of NP optimization problems, 20 Mar. 2000.

[5] T. Davidovic, L. Liberti, N. Maculan, and N. Mladenovic. Mathematical programming-based approach to scheduling of communicating tasks. Technical report, GERAD, December 15, 2004.

[6] R. P. Dick, D. L. Rhodes, and W. Wolf. TGFF: task graphs for free. In *CODES*, pages 97–101, 1998.

[7] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL – A Modeling Language for Mathematical Programming*. The Scientific Press, South San Francisco, 1993.

[8] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discrete Mathematics*, 5:287–326, 1979.

[9] M. Grajcar and W. Grass. Improved constraints for multiprocessor system scheduling. In *DATE*, page 1096. IEEE Computer Society, 2002.

[10] V. S. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan. Scheduling on unrelated machines under tree-like precedence constraints. In *Proceedings of APPROX-RANDOM 2005*, volume 3624 of *Lecture Notes in Computer Science*, pages 146–157. Springer, 2005.

[11] Y.-K. Kwok and I. Ahmad. Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. *ACM Comput. Surv.*, 31(4):406–471, 1999.

[12] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. In *28th Annual Symposium on Foundations of CS*, pages 217–224, Los Angeles, California, 12–14 Oct. 1987. IEEE.

[13] L. Liberti. Compact linearization for bilinear mixed-integer problems. Technical Report 1124, Opt. Online, May 6, 2005.

[14] N. Maculan, S. C. S. Porto, C. Carneiro, R. Cid, and C. Souza. A new formulation for scheduling unrelated processors under precedence constraints, Apr. 28 1997.

[15] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, New York, 1988.

[16] T. Parks, J. Pino, and E. Lee. A Comparison of Synchronous and Cyclostatic Dataflow, 1995.

[17] S. Prakash and A. C. Parker. SOS: Synthesis of application-specific heterogeneous multiprocessor systems. *J. Parallel Distrib. Comput.*, 16(4):338–351, 1992, December.

[18] N. Shenoy, P. Banerjee, and A. N. Choudhary. A system-level synthesis algorithm with guaranteed solution quality. In *DATE*, page 417. IEEE Computer Society, 2000.

[19] D. B. Shmoys and É. Tardos. Scheduling unrelated machines with costs. In *SODA*, pages 448–454, 1993.

[20] M. F. Tompkins. Optimization techniques for task allocation and scheduling in distributed multi-agent operations. Master's thesis, MIT, June 2003.

[21] G. K. Wallace. The JPEG Still Picture Compression Standard. *j-CACM*, 34(4):30–44, Apr. 1991.