MAPPING AND SCHEDULING WITH TASK CLUSTERING FOR HETEROGENEOUS COMPUTING SYSTEMS

Y. M. Lam, J.G.F. Coutinho, W. Luk

Dept. of Computing,
Imperial College London.
{ymlam, jgfc, wl}@doc.ic.ac.uk

P. H. W. Leong

Dept. of Comp. Sci. and Eng.,
The Chinese University of Hong Kong.
phwl@cse.cuhk.edu.hk

ABSTRACT

This paper presents a new approach for mapping task graphs to heterogeneous hardware/software computing systems using heuristic search techniques. Two techniques: (1) integration of clustering, mapping, and scheduling in a single step and (2) multiple neighborhood functions strategy are proposed to enhance quality of mapping/scheduling solutions. Our approach is demonstrated by case studies involving 40 randomly generated task graphs, as well as four real applications including signal processing and pattern recognition. Experimental results show that the proposed integrated approach outperforms a separate approach in terms of quality of the mapping/scheduling solution by up to 18.3% for a heterogeneous system which includes a microprocessor, a floating-point digital signal processor, and an FPGA.

1. INTRODUCTION

Hardware/software codesign takes an application specification as input, and generates an implementation for a heterogeneous computing system that fulfils given performance criteria. Codesign involves task mapping and scheduling: the mapping process assigning tasks to processing elements and the scheduling process determining the execution sequence of tasks. This problem is known to be NP-hard in its general form and some of the works proposed to address this issue are shown in Table 1.

Constructive heuristic, such as list scheduling, build partial valid solutions until a complete solution is formed. In list scheduling, each task is assigned a priority based on heuristics, available tasks are scheduled in each iteration based on the assigned priority. Examples of such heuristics are critical path [1] [2], job length [3]. Another approach is deterministic method, which searches the solution space exhaustively. This method guarantees optimal solution but is not scalable. An example of such method is integer linear programming [4]. Heuristic search techniques are also applied to tackle the mapping and scheduling problem. In [5], the mapping process uses a genetic algorithm without considering scheduling. Tabu search has been applied to find the best mapping while a list scheduling method estimates the total execution time [6]. Similarly, in [7], genetic algorithm is applied. The impact of different heuristic search methods on mapping/scheduling quality and search time have also been analyzed [8]. In all of these approaches, mapping and scheduling are considered independently in two steps, and heuristic search techniques are only applied to the mapping process. Since mapping results affect scheduling results and vice versa, separate schemes lead to sub-optimal solutions. In the integrated approach, each processor is assigned an ordered list of tasks to be executed. Mapping and scheduling are combined as a single process that assigns tasks to specific locations of lists. One possibility is to assume that a hardware processor always executes a task faster than a software processor, so all tasks are initially allocated in software processor and then moved from software to a hardware processor to minimize the total execution time [9]. Heuristic search techniques [10] are also used to move tasks between lists iteratively and the best mapping/scheduling solution is found via a search.

Compared with a separate approach with independent mapping and scheduling, an integrated approach covers a larger search space that results in a higher chance of covering good solutions. However, the extended search space can also increase the difficulty of finding good solutions, especially in problems that involve complex applications and heterogeneous computing systems. Most of previous heuristic search based approaches only apply a standard model, effects of different configuration of such techniques are less extensively addressed. Moreover, only mapping and scheduling are integrated [10] [14]. Our main contributions are as follows:

- Integrating clustering, mapping, and scheduling in a single step.
- A strategy with multiple neighborhood functions to enhance the quality of mapping/scheduling solutions.
- An analysis of different neighborhood functions and two heuristic search techniques, simulated annealing

978-1-4244-1961-6/08/$25.00 ©2008 IEEE.

Table 1. Some approaches to address mapping/scheduling.

references	approach	examples of applications	comments
[1] [2]	list scheduling	random graphs tracking algorithm	critical path based, for homogeneous architecture
[11]	list scheduling	FFT	scheduling based on depth of task and number of successors, for MONTIUM architecture
[3]	list scheduling	navigation system	shortest job first
[9]	iteratively moving	random graph	iteratively move critical task from software processor to hardware processor
[5] [12]	genetic algorithm simulated annealing	random graphs	address mapping only
[6] [7] [13]	heuristic search + lish scheduling	random graphs, FFT, JPEG	divide mapping/scheduling into two steps
[10] [14]	heuristic search	mean value analysis, FFT	address software processor only mapping + scheduling
[4]	integer linear programming	filtering	for VLIW architecture only
this work	heuristic search	40 random graphs 5 applications	clustering + mapping + scheduling

and tabu search.

In the following, Section 2 provides an overview of multiple neighborhood functions and their use in search techniques. Section 3 covers the formulation of the integrated mapping/scheduling problem. Section 4 describes the proposed methodology and the associated reference architecture, neighborhood functions, and cost function. Section 5 presents the experimental setup and the results. Section 6 contains concluding remarks.

2. MULTIPLE NEIGHBORHOOD FUNCTIONS

The basis of heuristic search techniques is neighborhood search, which starts with a feasible solution and attempts to improve it by searching its neighbors, i.e. solutions that can be reached directly from the current solution by an operation called a move. Simulated annealing accepts not only neighbors that improve on the previously best solutions, but also those increases the cost. The probability of accepting a neighbor is defined as: $p = exp((cost(b) - cost(c))/T)$, where b is the best neighbor and c is the current neighbor, T is called the temperature and is updated as: $T_{new} = \alpha T_{old}$, where α is a constant and typically in the range of 0.9 to 0.99.

Instead of using single neighborhood function, a strategy involving multiple neighborhood functions is proposed. Initially, a feasible solution is generated and a neighborhood function is chosen to generate a new neighbor. If the new neighbor yields lower cost than the best previous solutions, it is accepted, i.e. a move from previous solution to the new one. Otherwise, acceptance of this new neighbor depends on the probability function. If there is no improvement after iterating a given number of times, a new neighborhood function is chosen to replace the old one. In this work, two neighborhood functions are used alternately, and the search ends when neither neighborhood function produces better solutions.

Tabu search keeps a list of the searched space and uses it to guide the future search direction; it can forbid the search moving to some neighbors. In the proposed tabu search based on multiple neighborhood functions, after an initial solution is generated, two neighborhood functions are used to generate neighbors simultaneously. If there exists a neighbor of lower cost than the best solution so far and it cannot be found in the tabu list, this neighbor is recorded. Otherwise a neighbor that cannot be found in the tabu list is recorded. If all the above conditions cannot be fulfilled, a solution in the tabu list with the least degree, i.e. a solution being resident in the tabu list for the longest time, is recorded. If the recorded solution has a smaller cost than the best solution so far, it is recorded as the best solution. The searched neighbors are added to tabu list and solutions with the least degree are removed. This process is repeated until the search cannot find a better solution for a given number of iterations.

3. PROBLEM FORMULATION

3.1. Directed acyclic graph

In this work, it is assumed that an application is described by a directed acyclic task graph (DAG), which is used as input to the mapping/scheduling process. Given a set of tasks $TK = \{tk_1, tk_2, ..., tk_n\}$ to be executed, a directed acyclic graph can be defined as $G = (TK, DF)$. Each task tk_i is a node in the graph and $DF = \{df_{ij}; i = 1, 2, ..., n; j = 1, 2, ..., n\}$ is a set of directed edges corresponding to data flow dependencies between tasks. Each edge $df_{ij} \in DF$ denotes an amount of data flow from task tk_i to tk_j, so tk_i is a predecessor of tk_j, and conversely, tk_j is a successor of tk_i. The number of nodes and edges are defined as $|TK|$ and $|DF|$ respectively. Given a set of interconnected heterogeneous processing elements $PE = \{pe_1, pe_2, ..., pe_m\}$, each task node tk_i of the task graph is associated with a set $\{t_{i1}, t_{i2}, ..., t_{im}\}$ that denotes the execution times for implementing this task on each processing element of PE. The time to transfer results between tasks can be represented as $DT = \{dt_{ij}; i = 1, 2, ..., n; j = 1, 2, ..., n\}$, each dt_{ij} denoting the time to transfer results from task tk_i to tk_j. This is calculated as the amount of data flow df_{ij} divided by the data transfer rate.

3.2. Integrated mapping/scheduling

Given a set of task lists $PL = \{pl_1, pl_2, ..., pl_m\}$, where $pl_j = (as_{j1}, as_{j2}, ..., as_{jq})$ is an ordered task sequence to be executed by processing element pe_j, each task in pl_j will be processed by pe_j in sequence when it is ready for execution, when all its predecessors are finished. Task mapping and scheduling thus deal with assigning tasks to task lists. A task assignment function is defined as A: $TK \rightarrow PL$, e.g. $A(tk_i) = as_{jk}$ denotes task tk_i being assigned to as_{jk} of list pl_j. This means that tk_i is the kth task to be executed by processing element pe_j. A mapping/scheduling solution is characterized by assignments of all tasks to processing elements, i.e. for every task $tk_i \in TK$, $A(tk_i) = as_{jk}$ for a $pl_j \in PL$. It is assumed that only one task can be assigned to each as_{ij}.

In this work, a directed acyclic task graph is used as input to the mapping/scheduling process which adopts an integrated strategy. Simulated annealing and tabu search based on multiple neighborhood functions are then applied to modify the task assignment and generate mapping/scheduling solutions iteratively. Based on each solution, an overall processing time, the time to finish all tasks, is calculated and used as the cost to guide the heuristic search. The goal is to find a solution with minimum overall processing time.

4. METHODOLOGY

4.1. Reference architecture

The reference heterogeneous computing system contains three processing elements: one microprocessor, one FPGA, and one DSPU, which are fully connected. Each processing element has a local memory for data storage during task execution, and each communication channel between two processing elements is being assigned a weight which specifies the data transfer rate. Results of a task's predecessors must be transferred to the local memory before this task starts execution. The method proposed in this work is not limited to this architecture, this reference architecture is just used to evaluate the performance of difference approaches.

4.2. Neighborhood functions

Given that s denotes current mapping/scheduling solution, the following five basic neighborhood functions $NFx(s)$ are adopted:

```
NF1: random-relocation
  (1) Randomly select a task at list $pl_p$ position $as_{pq}$.
  (2) Randomly select another task at list $pl_j$ position $as_{jk}$.
  (3) Relocate task from ($pl_p$, $as_{pq}$) to ($pl_j$, $as_{jk}$).
  (4) Relocate largest data flow parent of task ($pl_p$, $as_{pq}$)
      to a position in $pl_j$ which yields lowest cost.
NF2: guided-relocation
  (1) Randomly select a task at list $pl_p$ position $as_{pq}$.
  (2) For all lists $pl_j \in PL$ and positions $as_{jk}$: relocate task
      ($pl_p$, $as_{pq}$) to ($pl_j$, $as_{jk}$) which yields lowest cost.
  (3) Relocate largest data flow parent of task ($pl_p$, $as_{pq}$)
      to a position in $pl_j$ which yields lowest cost.
NF3: random-swap
  (1) Randomly select a task at list $pl_p$ position $as_{pq}$.
  (2) Randomly select another task at list $pl_j$ position $as_{jk}$.
  (3) Swap task in ($pl_p$, $as_{pq}$) with task in ($pl_j$, $as_{jk}$).
NF4: guided-swap
  (1) Randomly select a task at list $pl_p$ position $as_{pq}$.
  (2) For all lists $pl_j \in PL$ and positions $as_{jk}$: swap task
      ($pl_p$, $as_{pq}$) with task ($pl_j$, $as_{jk}$) which yields
      lowest cost.
NF5: best-relocation
      For all lists $pl_p \in PL$ and positions $as_{pq}$:
      relocate task ($pl_p$, $as_{pq}$) to every list $pl_j \in PL$
      and position $as_{jk}$, choose the relocation which yields
      lowest cost.
```

Based on the observation that mapping tasks with large data flow to the same processing element can potentially reduce data transfer overhead, clustering techniques, which group tasks with large data flow together and allocate them to the same processing elements have been proposed [6]. However, clustering, mapping, and scheduling are solved separately in previous work, which may result in sub-optimal.

This work proposes two new neighborhood functions ($NF1$ and $NF2$) which integrate the clustering technique into the neighbourhood function, i.e. after relocating a task, move the largest data flow parent, which has the largest data flow among all parents of the current task, to the same processing element. If such movement yields lower cost, accept this move and generate a new neighbor, otherwise, discard it.

One more neighbourhood function $NF5$, which uses a best-relocation strategy [10], is also analysed. This neighbourhood function searches all possible relocations for all tasks, and choose the one yield lowest cost as new neighbor.

4.3. Cost function

As mentioned before, overall execution time is used as the cost to guide the heuristic search. This is the time for processing all tasks using the reference heterogeneous computing system and includes data transfer time. The processing time of a task tk_i on processing element pe_k is calculated as the execution time of tk_i on pe_k plus the time to retrieve results from all of its predecessors. The data transfer time between a task and a predecessor is assumed to be zero if they are assigned in the same processing element.

It is noted that tasks cannot be randomly moved to any location due to data dependency, i.e. a task cannot be moved to a location such that it will execute prior to its predecessor. To avoid generating infeasible solution, our approach inflicts a huge penalty cost if such move is infeasible.

5. RESULTS

5.1. Experimental setup

The neighborhood size for the tabu search is 10 and the tabu list length is 10, the search would terminate if it cannot find a better solution after 200 iterations. In simulated annealing, the cooling rate α is 0.99, and it changes neighborhood function if current function yields no improvement after 200 iterations. The overall processing time using a single CPU is divided by the overall processing time using the reference heterogeneous computing system to obtain a speed up (SU):

$$SU = \frac{overall\ processing\ time_{single\ CPU}}{overall\ processing\ time_{Reference\ system}} \quad (1)$$

Solution quality is used to measure the performance of different experimental setups, it is defined as follows:

- **Solution quality**: Each experimental setup is run for a given duration (time constraint) and the best speed up (BSU) is recorded. An average BSU (ABSU) of N runs is calculated and used as quality measure, which represents the quality of solutions a particular experimental setup can find. A higher ABSU value means the solution quality is higher.

Table 2. Solution quality using tabu search.

Combinations of	Number of task			
neighborhood functions	10	30	50	80
($NF1$ only)	3.34	5.59	5.23	4.75
($NF2$ only)	3.34	6.45	6.82	6.81
($NF3$ only)	3.07	3.81	3.08	3.05
($NF4$ only)	3.10	4.35	3.70	3.30
($NF5$ only)	3.34	5.43	5.00	3.41
$(NF1, NF3)$	3.34	5.41	4.92	4.32
$(NF1, NF4)$	3.34	6.45	6.91	7.08
$(NF2, NF3)$	3.34	6.42	6.62	6.57
$(NF2, NF4)$	3.34	6.51	7.04	7.17
$(NF1, NF2)$	3.34	6.40	6.62	6.55
$(NF3, NF4)$	3.07	4.63	3.94	3.47

Table 3. Solution quality using simulated annealing.

Combinations of	Number of task			
neighborhood functions	10	30	50	80
($NF1$ only)	3.24	5.74	5.90	5.60
($NF2$ only)	3.34	6.37	6.59	6.50
($NF3$ only)	2.98	4.56	4.51	4.22
($NF4$ only)	3.04	4.90	4.17	3.65
($NF5$ only)	3.34	5.78	5.57	5.49
$(NF1, NF3)$	3.29	6.21	6.52	6.45
$(NF1, NF4)$	3.28	6.28	6.72	6.86
$(NF2, NF3)$	3.34	6.44	6.77	6.67
$(NF2, NF4)$	3.34	6.46	6.86	7.01
$(NF1, NF2)$	3.34	6.37	6.48	6.37
$(NF3, NF4)$	2.99	4.83	4.70	4.07
$(NF3, NF1)$	3.34	6.44	6.73	6.64
$(NF4, NF1)$	3.32	6.28	6.60	6.75
$(NF2, NF1)$	3.34	6.36	6.63	6.50

5.2. Random graphs

Directed acyclic graphs with 10, 30, 50, and 80 tasks are analyzed. 10 instances of each graph size are generated randomly so there are a total of 40 graphs. Experimental results are obtained from 10 runs in each problem instance. Tables 2 and 3 show the solution quality obtained using integrated approach with tabu search and simulated annealing respectively. The results of using the strategy with two neighborhood functions are designated by the notation (NFa, NFb). For simulated annealing, this notation also denotes the order to choose neighborhood function: NFa is used first to generate neighbors, changing to NFb if there is no improvement after 200 iterations. The notation (NFa only) means using a single neighborhood function NFa.

In Table 2, among the four neighborhood functions $NF1$ to $NF4$, $NF2$ achieves the highest quality: its value is twice as much as others for most problem instances. Similar results are obtained when simulated annealing is used (Table 3). A quality comparison between different neighborhood functions is shown in Figure 1. It is found that tabu search outperforms simulated annealing, and using a strat-

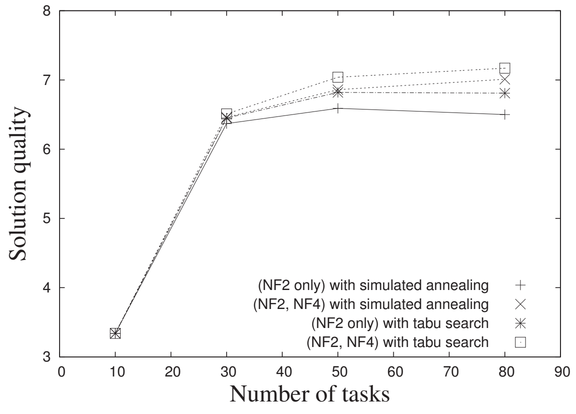

Fig. 1. Comparison of solution quality (speedup in regards to single CPU execution) between different neighborhood functions of integrated approach using random graphs.

Table 4. Comparison of solution quality between different works using random graphs. SEP: separate approach, INT: integrated approach.

Number of task	10	30	50	80
SEP,TABU [6]	3.31	5.89	6.23	6.17
INT,TABU,($NF5$, only) [10]	3.34	5.43	5.00	3.41
INT,TABU,($NF2, NF4$)	3.34	6.51	7.04	7.17

Table 5. Solution quality for different search time constraints using FFT.

Time constraint (s)	10	30	50
SEP,TABU [6]	5.37	5.47	5.57
INT,TABU,($NF5$, only) [10]	2.17	2.36	2.57
INT,TABU,($NF2, NF4$)	5.6	5.97	6.13

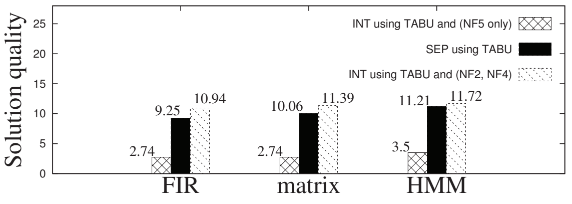

Fig. 2. Solution quality comparison for various applications

egy with two neighborhood functions yields higher quality than a single neighborhood function. The improvement is more significant for larger problem size. In Figure 1, tabu search with $(NF2, NF4)$ achieves the highest quality, which is actually the highest among all experimental setups chosen in this work; this strategy is used later in our experiments using real applications (Section 5.3).

Table 4 shows a solution quality comparison between this work and other approaches: a best-relocation based integrated approach [10], and a separate approach proposed in [6], where tabu search is found to yield best performance. Our approach outperforms both approaches in all problem instances. The improvement is more significant for larger problems: it is 16.2% higher than the separate approach and 110% higher than the best-relocation based approach for the case with 80 tasks.

5.3. Task graphs for real applications

Apart from random graphs, the commonly used butterfly structure for the fast Fourier transform (FFT) [15] is used as an additional example. A 32-point FFT containing 80 butterfly nodes with each node regarded as a single task is used. The execution time of each node is measured for processing 1.0×10^7 32-point FFTs, and the communication speed is assumed to be 1.0×10^8 data elements per second. For an Intel Pentium-4 3.2GHz microprocessor and an Atmel mAgic floating-point digital signal processor the execution times of each node are 384 ms and 100 ms respectively. A fully pipelined architecture for the butterfly node has been developed for a Xilinx Virtex-II XC2V6000 FPGA using the Haydn design-flow [16] and the execution time for this architecture is 91 ms. Table 5 illustrates the solution quality of 20 runs for each setup. One can see that using integrated approach with tabu search and $(NF2, NF4)$ achieves higher value; it shows that this setup is capable of finding better solutions. The maximum improvement is 10.0% higher than the separate approach and 138.5% higher than the best-relocation based approach for 50-second constraint.

Besides FFT, three other applications are employed to evaluate the proposed approach; these include FIR filtering, matrix multiplication, hidden Markov model (HMM) decoding for pattern recognition. Figure 2 illustrates the solution quality comparison given a 30-second search time constraint, the proposed multiple neighbourhood function strategy outperforms the other two approaches in all cases, the corresponding improvements over the separate approach are 18.3%, 13.2% and 4.5% respectively. The improvement for HMM is less significant; one reason is that the amount of data flow is smaller in this application, so the penalty of inappropriate task mapping using the separate approach is also less significant.

6. CONCLUSIONS

An integrated hardware/software codesign approach using the multiple neighborhood functions strategy is presented, where clustering, mapping, and scheduling are integrated in a single step. It is found that the guided-move based clustering approach outperforms other neighbourhood functions and the solution quality can be further improved using the multiple neighbourhood functions strategy. In particular, using tabu search with $(NF2, NF4)$ achieves the highest quality. Experimental results obtained using both randomly generated task graphs and four real applications show that the proposed codesign approach with multiple neighborhood functions is superior to previous approaches in quality by up to 18.3%.

Current and future work includes extending our approach to cover task graphs which are not acyclic, and exploring opportunities for carrying out mapping and scheduling at run time.

Acknowledgement. The support of FP6 hArtes (Holistic Approach to Reconfigurable Real Time Embedded Systems) Project, the UK Engineering and Physical Sciences Research Council, Celoxica and Xilinx is gratefully acknowledged.

7. REFERENCES

[1] K. R. Pattipati, T. Kurien, R. T. Lee, and P. B. Luh, "On Mapping a Tracking Algorithm Onto Parallel Processors," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 26, no. 5, pp. 774–791, 1990.

[2] T. Pop, P. Eles, and Z. Peng, "Holistic Scheduling and Analysis of Mixed Time/Event-Triggered Distributed Embedded Systems," in *Proceedings of the tenth International Symposium on Hardware/software Codesign*, 2002, pp. 187–192.

[3] Y. Shin and K. Choi, "Enforcing schedulability of multi-task systems by hardware-software codesign," in *Proceedings of the Fifth International Workshop on Hardware/Software Codesign*, 1997, pp. 3–7.

[4] A. Bednarski and C. Kessler, "Integer Linear Programming versus Dynamic Programming for Optimal Integrated VLIW Code Generation," in *Proceedings of the 12th International Workshop on Compilers for Parallel Computers*, 2006, pp. 73–85.

[5] J. I. Hidalgo and J. Lanchares, "Functional Partitioning for Hardware-Software Codesign Using Genetic Algorithms," in *Proceedings of the 23rd EUROMICRO Conference on New Frontiers of Information Technology*, 1997, pp. 631–638.

[6] T. Wiangtong, P. Y. K. Cheung, and W. Luk, "Hardware/Software Codesign: A Systematic Approach Targeting Data-intesive Applications," *IEEE Signal Processing Magazine*, vol. 22, no. 3, pp. 14–22, May 2005.

[7] L. Shang, R. P. Dick, and N. K. Jha, "SLOPES: Hardware-Software Cosynthesis of Low-Power Real-Time Distributed Embedded Systems With Dynamically Reconfigurable FPGAs," *IEEE Trans on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 3, pp. 508–526, 2007.

[8] T. Wiangtong, P. Y. K. Cheung, and W. Luk, "Comparing Three Heuristic Search Methods for Functional Partitioning in Hardware-Software Codesign," *Journal on Design Automation for Embedded Systems*, vol. 6, no. 4, pp. 425–449, 2002.

[9] H. Liu and D. F. Wong, "Integrated Partitioning and Scheduling for Hardware/Software Co-design," in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 1998, pp. 609–614.

[10] S. C. S. Porto and C. C. Ribeiro, "A Tabu Search Approach to Task Scheduling on Heterogeneous Processors under Precedence Constraints," *International Journal of High-Speed Computing*, vol. 7, pp. 45–71, 1995.

[11] Y. Guo, C. Hoede, and G. Smit, "A pattern selection algorithm for multi-pattern scheduling," in *Proceedings of the Parallel and Distributed Processing Symposium*, 2006, pp. 25–29.

[12] S. Banerjee and N. Dutt, "Efficient Search Space Exploration for HW-SW Partitioning," in *Proceedings of the International Symposium on Hardware/software Codesign and System Synthesis*, 2004, pp. 122–127.

[13] F. Dittmann, M. Gotz, and A. Rettberg, "Model and Methodology for the Synthesis of Heterogeneous and Partially Reconfigurable Systems," in *Proceedings of the 14th Reconfigurable Architectures Workshop*, 2007, pp. 1–8.

[14] Y. M. Lam, J. G. F. Coutinho, W. Luk, and P. H. W. Leong, "Integrated Hardware/Software Codesign for Heterogeneous Computing Systems," in *Proceedings of the IEEE IV Southern Programmable Logic Conference*, 2008, pp. 217–220.

[15] S. K. Mitra, *Digital Signal Processing: A Computer-Based Approach.* The McGraw-Hill Companies, Inc, 2002.

[16] J. G. F. Coutinho, J. Jiang, and W. Luk, "Interleaving Behavioral and Cycle-Accurate Descriptions for Reconfigurable Hardware Compilation," in *Proc. of the IEEE Symposium on FCCM*, 2005, pp. 245–254.