Time-Energy Design Space Exploration for Multi-Layer Memory Architectures

Radoslaw Szymanek
Dept. of Computer Science
Lund University
221 00 Lund, Sweden
radsz@cs.lth.se

Francky Catthoor
IMEC
Belgium
Also prof. at K.U.Leuven
catthoor@imec.be

Krzysztof Kuchcinski
Dept. of Computer Science
Lund University
221 00 Lund, Sweden
kris@cs.lth.se

Abstract

This paper presents an exploration algorithm which examines execution time and energy consumption of a given application, while considering a parameterized memory architecture. The input to our algorithm is an application given as an annotated task graph and a specification of a multi-layer memory architecture. The algorithm produces Pareto trade-off points representing different multi-objective execution options for the whole application. Different metrics are used to estimate parameters for application-level Pareto points obtained by merging all Pareto diagrams of the tasks composing the application. We estimate application execution time although the final scheduling is not yet known. The algorithm makes it possible to trade off the quality of the results and its runtime depending on the used metrics and the number of levels in the hierarchical composition of the tasks' Pareto points. We have evaluated our algorithm on a medical image processing application and randomly generated task graphs. We have shown that our algorithm can explore huge design space and obtain (near) optimal results in terms of Pareto diagram quality.

1. Introduction

Many memory-centric applications, such as multimedia and image processing applications, need to process huge amount of data. These data are stored in system memory, which has to provide bandwidth for processing units. In this paper, we assume that the processor architecture has enough processing power and we concentrate instead on the memory architecture and data access methods. It is now well-known that the memory organization and data storage/access scheme has become a crucial part of many embedded systems [2, 6].

Several major memory constraints need to be satisfied during application execution. The first type of constraints which need to be considered are memory size constraints. Bandwidth constraints are the second important class of memory constraints. In order to achieve higher bandwidth, memories, such as SDRAM's, with sophisticated access schemas are used. These schemas often impose bandwidth related constraints on data placement and memory access patterns. Finally, modern memories come with different low energy operation modes [11] to save energy. Therefore energy considerations can yield their own data placement and access constraints.

It takes as input timing and energy constraints, which are obtained from the application tasks and memory architecture specification, and possibly other designer constraints. The exploration finds a set of Pareto solutions in a two-dimensional space, time and energy, with memory mapping and access schedule constraints. These solutions can later be used by a scheduler to optimally adapt the application's execution to the current state of the system.

This paper is organized as follows. Section 2 outlines related work. In section 3, we present basis and assumptions of our work. Section 4 describes our algorithm for time-energy design space exploration. Section 5 presents experimental results. Finally, section 6 concludes the paper.

2. Related Work

Our methodology requires to have Pareto points for every individual task. We assume that the task graph for the application is known. Each Pareto point specifies different execution mode of the task. Much research, presented in survey [13], has concentrated on design space exploration for relatively small pieces of code (tasks). These works often optimize different parameters, such as execution time, energy consumption, and bandwidth, taking into account architecture constraints. We make a hierarchical composition of tasks' Pareto diagrams and obtain a single Pareto diagram for the whole application. A similar method was proposed in [1], but they can use monotonic quality evaluation functions as well as independent and strongly correlated metrics

1530-1591/04 $20.00 (c) 2004 IEEE

for their problem. Our problem does not have these features and we have to use composition heuristics.

The initial specification often does not fully specify the order how the data should be accessed. The remaining access scheduling freedom can be used to find the cheapest memory architectures in terms of bandwidth and size requirements. This issue was addressed in [6]. If the access scheduling is done first then the problem of data assignment is restricted so much that little optimization of memory architecture is possible. The restricted data assignment can be represented as a minimum cost network flow problem and solved optimally, as shown in [8]. The importance of the scheduling for the quality of data assignment was also indicated in [15]. In our work, during time-energy exploration, we specify the scheduling only partially to preserve good optimization possibilities during the placement exploration.

The importance of a systematic approach to explore and tradeoff different Pareto solutions was indicated in [5]. The authors trade energy and time budget of application tasks to achieve good solutions for the application itself. We have explicitly added to the Pareto points additional constraints for bandwidth and data placement. This addition helps to address memories with high latencies.

Synchronous DRAM adds another dimension to memory access patterns. The memory is divided into multiple banks which can be accessed through page buffers. Multiple pages can be active at the same time providing possibility to access different data without page swapping [14]. Minimization of page misses by ordering accesses while respecting data dependencies was presented in [12]. This problem was solved by forbidding sharing of the same bank by data with overlapping lifetimes. In our approach, data can share a bank if they have overlapping lifetimes but not overlapping accesses. We also allow data to share bandwidth if they are using different page buffers which makes it possible to use interleaved burst memory access. The problem of modeling (interleaved) burst read and writes modes was also presented in [9].

Data assignment is important since it influences how many parallel accesses to different memories can be performed. Parallel accesses to many memories is cheaper than parallel accesses to one large memory, at least at the SRAM layer [16]. This problem was modeled as graph coloring in [3]. In our work, we take a step further. We do not consider single read or write accesses and their parallelism but we think in terms of the bandwidth we have to sustain to satisfy time and energy constraints.

Finally, some research has concentrated on reducing the energy consumption at the level of the operating system [7]. Unfortunately, this approach can be hindered if the application does not perform a good data assignment. It will be much harder to use the low power modes of memory components if the application scatters its data too much, as can happen when only bandwidth optimization takes place. In our work, each task execution option explicitly states both placement and bandwidth constraints, therefore we make the application execution more friendly to such approaches.

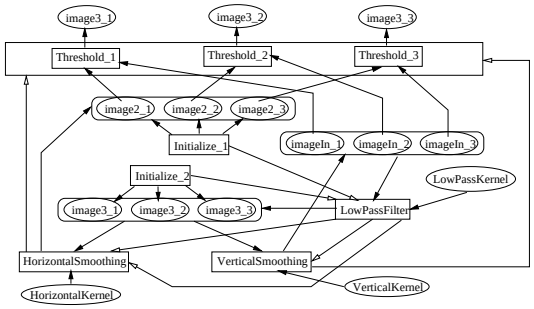


Figure 1. Medical application example

3. Basis and assumptions

The data of embedded applications usually differ in size and access patterns. The access pattern is defined by the frequency the data is accessed and duration of burst access. Data heterogeneity suggests usage of heterogeneous memory architecture to achieve better overall performance of an embedded system.

To be able to capture all memory architecture and application constraints in a single framework and make consistent decisions we use a Constraint Programming approach [10]. An important entity in this approach is a constraint store which checks and enforces consistency of constraints. All characteristics of the system are represented using finite domain variables and constraints. For example, application energy consumption is represented as a finite domain variable. This variable has its value at the beginning within an initial range (domain). Each time the decision on how to execute a task is made, the constraint store will call consistency methods which will narrow the domain of the energy consumption variable. During the search for application Pareto points, the constraint store helps us to indicate non-valid solutions which are immediately recognized since some constraints are not satisfied.

3.1. Application Model

The application is modeled as a task graph. The rectangles represent tasks and ellipses represent data which are used by tasks. A realistic application example, which needs to be assigned and scheduled, is depicted in Figure 1. This application consists of 8 tasks and 21 image variables (single assignment). Figure 1 shows less variables since some are already in-placed, making them multiple assignment variables. The arcs represent data and control precedence

Point	Execution Time [μs]	Energy [pJ] consumption	Bandwidth for Kernel [# slots]	Conflict Graph
1	11.34	395	2	CG1
2	11.83	223	1	CG1
3	10.86	380	2	CG2
4	11.34	213	1	CG2
5	13.34	457	2	CG3
6	13.82	260	1	CG3

Table 1. Pareto points for task example

constraints. The control precedence relation, represented by arc with white edge, imposes order between tasks which operate on a multiple assignment variable. We have represented some variables, such as *image3_1* (*i31*), by two different ellipses to increase picture readability.

Consider a task depicted in Algorithm 1, which is a part of the bigger task called *LowPassFilter* as depicted by Figure 1. This task has seven different data members. Consider a memory architecture which consists of three memories organized in two layers. The first layer contains two register files. The second layer contains a shared multi-banked SDRAM memory. Data are preassigned to a memory layer since it is known beforehand what is the best match for a given type of data. Mature solutions exist for partitioning of data into multiple memory layers (e.g. [4] where energy is incorporated in the cost model). The size of data must be known in advance. The data placement within a memory layer is modeled by memory and bank variable. In our example, the biggest data *Kernel* is pre-assigned to SDRAM and all other variables are preassigned to the memory layer with register files.

When an assignment of the data to different memory layers is done, the analysis of the task execution possibilities is performed. This analysis will yield Pareto points. Each row in Table 1 represents Pareto point for the normalization task. Each point, which represents task execution option, will specify execution time, energy consumption, required bandwidth, and data placement. Moreover access patterns are already fixed for each execution option. This helps to limit the application scheduling freedom and complexity but keeps optimization possibility since we use Pareto points for tasks execution.

The bandwidth required by a task to access a given data is represented by the number of time slots in the time window of the memory used to store this data (see column 4 in Table 1). Each time slot represents a "portion of the bandwidth" which can be accessed periodically. In this way, a task obtains sustainable bandwidth to access a specific data. The data placement is represented by a conflict graph where

Algorithm 1 Normalization of the kernel - task example

$factor = 0, r = 0, c = 0;$
for all $r < K$ **do**
  for all $c < K$ **do**
    $factor += abs((kernel + r * K)[c])$

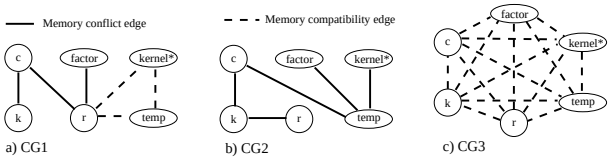


Figure 2. Conflict Graphs for task example

vertexes specify data and edges specify different kind of relations (constraints), as presented in Figure 2. We consider memory conflict edge, page conflict edge, memory compatibility edge, and bank compatibility edge. In Figure 2 only memory conflict and memory compatibility edges for register layer variables are depicted. The page conflict edge gives a possibility to constrain data so they are loaded in different pages thus giving a chance for concurrent accesses. The bank compatibility edge enforces that two data are stored in the same bank to enable page sharing.

In this particular example, after manual analysis, six different Pareto-optimal execution options are found. Option 2 in comparison to option 1 requires less bandwidth to access *Kernel* variable. Therefore it allows another parallel interleaved burst. It has lower energy consumption since a chance is present that other tasks will access memory in parallel. Therefore the overhead of memory static energy can be shared among tasks. Execution option 4 gives faster execution and lower energy than option 2 but it differs in constraints on data placement. Options 5 and 6 differ from the others since they enforce storage of all data from the register layer in one register file.

Our goal is to find Pareto points for the task graph of the whole application, represented as black circles in the right part of Figure 3. Each point specifies each task execution choice, and data assignment/access constraints.

3.2. Memory Model

A parameterized memory architecture with sufficient heterogeneity and freedom for assignment is assumed to be predefined. Our memory model reflects important constraints which influence the usage of memories. Three classes of constraints are associated with the memory structure namely *size*, *page*, and *bandwidth* constraints. A memory model for a memory architecture, which is specified in subsection 3.1, is depicted in Figure 4. Each mem-

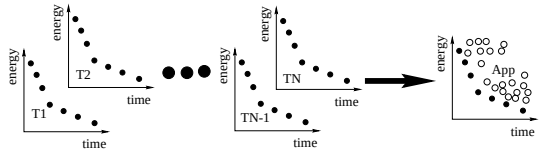
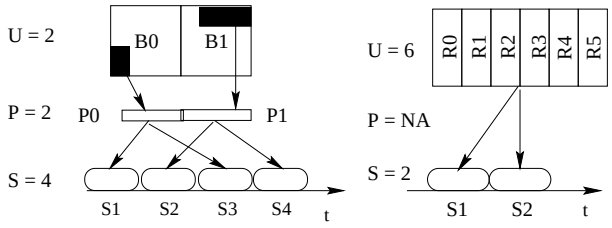


Figure 3. Application Pareto Diagram



a) dual SDRAM memory model b) register file - dual port

Figure 4. Memory Model

ory can be divided into U units. In our example, $U = 2$ for SDRAM memory. At any time of the application execution, the sum of the data stored in each unit cannot exceed its size.

For some memories, e.g. SDRAMs, a fixed number of page buffers is used to access data. The data itself has to be first in the page buffer to be accessible. In our model, we allow those values of U and P that U `mod` P = 0. We assume that the each page can be loaded from U `div` P consecutive banks. The page buffer is reserved by a task during its execution so other tasks do not interfere.

Each memory has fixed maximum bandwidth. The time axis is divided into an infinite number of time windows, consisting of a given number of time slots, denoted by *S*. Each slot of the time window is assigned an equal portion of the bandwidth. For proper execution, each task may require different bandwidth to access a specific data. Therefore one or more time slots within time window will be reserved by a task during its execution. Our requirement on memory technology makes it possible to model memories, such as SDRAM, DRAM, on-chip, which have good energy estimates based on required bandwidth and possibility to divide bandwidth into smaller portions. Each portion can be used to access different data under well specified constraints.

4. Application Pareto Points Creation

The number of possible ways of executing an application is equal to the product of number of possible ways to execute tasks. This number grows exponentially with respect to the number of tasks. Algorithm 2 performs branch and bound search to find Pareto points among all possible compositions of task executions. Pareto points are depicted as black circles in the right part of Figure 3.

Algorithm 2 Branch-and-bound search algorithm

exhaustiveComposition($\mathcal{S}$, $\mathcal{T}$, $\mathcal{R}$, $\mathcal{SP}$)
{$\mathcal{S}$ denotes constraint store with input constraints}
if $\mathcal{T} \neq \emptyset$ **then**
  nextTask $\in \mathcal{T}$
  for all Pareto Point $\mathcal{P}$ of *nextTask* **do**
    Add constraints to $\mathcal{S}$ to enforce $\mathcal{P}$
    if consistent($\mathcal{S}$) **then**
      exhaustiveComposition ($\mathcal{S}$, $\mathcal{T} \setminus$ *nextTask*, $\mathcal{R} \cup \mathcal{P}$, $\mathcal{SP}$)
    Remove constraints which enforced $\mathcal{P}$ from $\mathcal{S}$
else
  Add current solution $\mathcal{R}$ to $\mathcal{SP}$
  Add constraint to $\mathcal{S}$ so all following solutions will be better

This algorithm selects task by task recursively and assigns it to execute in one of possible ways. Each time the decision on how to execute a task is made, additional constraints are added to store *S* and an evaluation of current partial solution follows. If constraint store *S* is not consistent then either a resource violation has occurred or the current partial solution has no chance to become a Pareto solution. In this case, another execution point for current and/or previous tasks is chosen. If the algorithm succeeds in assigning execution option to all tasks then new Pareto solution was found.

The branch and bound algorithm works well for small examples, but it has excessively long runtime for larger problems since it has exponential complexity. Therefore we have also developed a heuristic, presented in algorithm 3, to solve larger problems in reasonable time. In algorithm 3, the first while loop uses branch and bound search to compute optimal Pareto points for small task subsets. The merge part of the algorithm, represented by the second while loop, recursively calls the exhaustive search but for two closest neighbors task subsets only. Each time the inner while loop finishes, a Pareto diagrams for the next level are created. Eventually, the heuristic obtains one Pareto diagram on the highest level as a result of merging smaller Pareto diagrams.

The proposed heuristic needs the following number of constraint store consistency techniques executions as presented below

$$O(\frac{n}{s} \cdot p^{\log_2(\frac{n}{s})} \cdot k^2 + \frac{n}{s} \cdot k^s) \quad (1)$$

where *n* is the number of tasks, *k* is the number of execution modes, *s* is the task cluster size and *p* is the problem dependent constant. In general, a larger constant *p* means that more Pareto points exist in the intermediate Pareto diagrams. In the worst case, the number of Pareto points in the combined diagram is in the order of $O(k^n)$, but typically, like in the medical image processing example, the number of Pareto points for the application is in the order of $O(kn)$.

Algorithm 3 The Pareto points composition heuristic

heuristicCombinePD(*Size*, $\mathcal{C}$)
{$\mathcal{C}$ denotes constraint store with input constraints}
$i = 0$ {Compose Pareto diagrams for task sets of size (*Size*) }
while $i <$ (*noTasks*/*Size*)) **do**
  exhaustiveComposition ($\mathcal{C}$, T_{i*Size} to $T_{(i+1)*Size}$, $\emptyset$, $\mathcal{SP}$)
  add constraints to $\mathcal{C}$ to enforce solutions only from $\mathcal{SP}$, $i{+}{+}$
{Recursively join two previously obtained Pareto diagrams}
while *Size* $<$ *noTasks* **do**
  $i = 0$, *Size* = *Size* $* 2$
  while $i <$ (*noTasks*/*Size*)) **do**
    exhaustiveComposition ($\mathcal{C}$, T_{i*Size} to $T_{(i+1)*Size}$, $\emptyset$, $\mathcal{SP}$)
    update constraints to enforce solutions only from $\mathcal{SP}$, $i{+}{+}$
Return constraints to enforce application Pareto points from $\mathcal{SP}$

task	1	2	3	4
initialize	640 μs	1250 μs	1860 μs	1880 μs
threshold	1080 nJ	650 nJ	560 nJ	550 nJ
smoothing	14680 μs	16500 μs	20120 μs	27400 μs
lowpassfi lter	16740 nJ	15250 nJ	13800 nJ	11770 nJ

Table 2. Pareto points for medical appl. tasks

The creation of time-energy Pareto points for an application requires a multi-objective cost function. The energy requirement for an application is defined as a sum of the energy requirements for all tasks. We can estimate how much energy application will require since we have energy estimates for each task execution option. On the other hand, determining how much time application requires is harder since no assignment or scheduling decisions are made at this point. Therefore, we decided to use an estimation function presented below

$$time = w_1 \cdot \sum_{j \in \mathcal{T}} time_j + w_2 \cdot \max(\min_{j \in \mathcal{T}} ECT_j) \qquad (2)$$

where ECT_j denotes earliest completion time of task j and $time_j$ denotes the execution time of task j. If architecture has a large amount of parallelism available then the second measure, called *ECT* here, is a much better criterion to assess the execution time of application. On the other hand if most work is done sequentially then the sum of task execution times, called SUM here, is a much better measure for the overall time of an application execution time.

5. Experimental Results

Each task of the medical image processing application, which was depicted in Figure 1, has been analyzed and different code transformation techniques, such as loop splitting and loop unrolling, are applied to adapt the task to the memory structure. Different task execution alternatives are evaluated yielding Pareto points for each task as presented in Table 2. The energy values for each task are computed using the power model for DRAM memories from Micron [11]. These Pareto points have different data placement constraints, e.g. at the SDRAM layer. Despite the fact that data assignment is not performed it is still possible to check consistency constraints deduced from conflict graphs. These checks may indicate non satisfiable solutions.

Some tasks, such as *initialize* and *threshold*, perform the same amount of accesses and since they can be scheduled in the same amount of cycles, the energy model gives the same amount of consumed energy.

The results of design space exploration for the medical application is presented in Table 3. Different time metrics are applied to estimate the execution time of an application. In case of a two-dimensional design space exploration we have used different values for weights w_1 and w_2 in (2). It

Exp	Size	Time (w_1, w_2)	# Tasks	Cost [%]	# Pareto Points	Heuristic Runtime
1	4	(0,1)	8	100.00	50	5 s
2	8	(0,1)	8	100.00	50	200 s
3	4	(1,0)	8	100.00	161	30 s
4	8	(1,0)	8	100.00	161	130 s
5	4	(1,1)	8	100.06	190	60 s
6	8	(1,1)	8	100.00	219	200 s
7	4	3D	8	100.00	384	150 s
8	8	3D	8	100.00	384	520 s
9	2	(0,1)	16	103.00	48	33 s
10	4	(0,1)	16	103.00	68	50 s
11	8	(0,1)	16	100.00	61	2 min
12	2	(1,0)	16	100.00	427	11 min
13	4	(1,0)	16	100.00	427	18 min
14	8	(1,0)	16	100.00	427	22 min
15	2	(1,1)	16	100.08	590	66 min
16	4	(1,1)	16	100.06	455	29 min
17	8	(1,1)	16	100.00	536	120 min
18	2	3D	16	100.00	2316	23 hours

Table 3. Results for medical application

is also possible to perform 3-dimensional space exploration where energy, *ECT*, and *SUM* are the cost functions. *ECT* without any index denotes the maximum value of ECT_j for any task j. Figure 5 depicts the result of such an exploration for the pipelined version of the medical application.

The cost of the created 2D Pareto diagram is computed by the formula:

$$cost = \sum_{t_0}^{t_n} energy(t_{i-1}) \cdot (t_i - t_{i-1}) \qquad (3)$$

where smaller cost represents a better solution, *energy(t_x)* gives the required energy for execution time t_x, t_0 (t_n) denotes the fastest (slowest) application execution mode. Experiments 2,4,6, and 8 show optimal solutions since we have used a *size* equal to the number of tasks.

We have normalized the cost of the solutions, where 100% denotes the optimal solution in case of experiments 1 to 8, and in other cases 100% denotes the best obtained results. We have used a two-dimensional projection, energy and sum dimension, for experiments 7,8, and 18 to be able to compute the cost for them. Experiments 9 to 18 present results for the pipelined version, where two instances of the

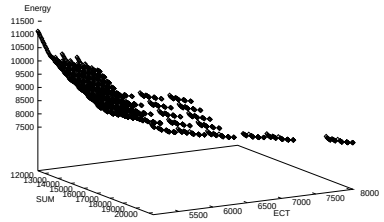


Figure 5. 3D Pareto diagram

Exp	Size	Task Order	# Tasks	known sub-optimal	# Pareto Points	Heuristic Runtime
1	2	t	32	1	310	10 min
2	4	t	32	1	310	25 min
3	8	t	32	0	310	2 hours
4	2	r	32	0	310	10 min
5	4	r	32	0	310	25 min
6	8	r	32	0	310	2 hours

Table 4. Results for rtg - SUM

medical application are executed concurrently with 20 ms delay. For this case, it is not possible to prove, in reasonable time, that results are optimal. However, increasing the scope of the checked design space gives only minor gains. So this indicates that our heuristic can obtain very good results even when the design space is not fully explored.

We have also evaluated our approach on larger randomly generated task graphs (rtg). Each task has between four and six different execution options. In this case task graphs are more diverse and irregular than in the medical application case. Each experimental result is computed as an average value of results obtained for 20 task graphs.

The experimental results for random task graphs and time metric SUM are presented in Table 4. The heuristic can be influenced by two parameters: the group size and task order. Two orders of tasks are considered. They are used to group tasks, where *t* stands for the topological order and *r* stands for random order. It has been possible to find one better solution, for 1 out of 20 task graphs, by changing the task order or increasing the size parameter. In this case, we obtained marginal improvement of less than 1% since only one Pareto point is slightly improved. The experiments are run on Sun Sparc with a 300 MHz processor.

The results for random task graphs (rtg) and weighted time function are presented in Table 5. Since the graphs are irregular and the metric is complicated, many more intermediate Pareto points exist. Therefore, it is beneficial from runtime and cost point of view to increase the size from two to four. Further increase of *size* to 8 will, however, increase the runtimes while giving little improvement as shown already in other cases. Even for this difficult time metric composed of two terms, we are able to find a set of Pareto points of the order O(nk) out of O(n^k) possible application executions.

6. Conclusions

We have presented a heuristic that finds executions modes for a given application. These modes are characterized by Pareto points in time-energy space. They are created by our heuristic which hierarchically composes Pareto diagrams of single tasks in a given task graph. Our heuristic achieves adjustable complexity reduction while keeping the quality of the results. We have shown

Exp	Size	# Tasks	Cost [%]	# Pareto Points	Heuristic Runtime
1	2	32	100.96	345	53 min
2	4	32	100.42	340	45 min
3	8	32	100.00	333	113 min

Table 5. Results for rtg - SUM and ECT.

that small *size* values give (near) optimal or good quality solutions and increasing the size parameter to explore larger design space gives little improvement at a high run-time cost.

References

[1] S. G. Abraham, B. R. Rau, and R. Schreiber. Fast design space exploration through validity and quality fi ltering of subsystems designs. Technical report, Hewlett Packard Laboratories, 2000.

[2] L. Benini and G. De Micheli. System-level power optimization techniques and tools. *ACM Transactions on Design Automation for Embedded Systems*, 5(2):115–192, April 2000.

[3] M. Breternitz and J. Shen. Organization of array data for concurrent memory access. In *Proc. of 21st Workshop on Microprogramming and Microarchitecture*, pages 97–98, 1988.

[4] E. Brockmeyer, M. Miranda, and H. Corporaal. Layer assignment techniques for low energy in multi-layered memory organizations. In *Proc. of 6th ACM/IEEE Design, Automation and Test in Europe Conference*, pages 1070–1075, 2003.

[5] E. Brockmeyer, A. Vandecappelle, and F. Catthoor. Systematic cycle budget versus system power trade-off: a new perspective on system exploration of real-time data dominated applications. In *Proc. of IEEE International Symposium on Low Power Design*, 2000.

[6] F. Catthoor et al. *Custom Memory Management Methodology: Exploration of Memory Organisation*. Kluwer, 1998. ISBN 0-7923-8288-9.

[7] V. Delaluz, A. Sivasubramaniam, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Scheduler based DRAM energy management. In *Proc. of 39th Design Automation Conf.*, 2002.

[8] C. Gebotys. Low energy memory and register allocation using network fbw. In *Proc. of Design Automation Conference*, pages 435–440, 1997.

[9] A. Khare, P. Panda, N. D. Dutt, and A. Nicolau. High-level synthesis with SDRAMs and RAMBUS DRAMs. *IEICE Trans. on Fundam. Electron. Commun. Comput. Sci.*, 6(2):149–206, April 2001.

[10] K. Kuchcinski. Constraints-driven scheduling and resource assignment. *ACM Transactions on Design Automation of Electronic Systems*, 8(3):355–383, July 2003.

[11] Micron Technology Inc. *SDRAM power model*. www.micron.com.

[12] P. Panda. Memory bank customization and assignment in behavioral synthesis. In *Proc. of IEEE/ACM International Conference on Computer-Aided Design*, pages 477–481, 1999.

[13] P. Panda, F. Catthoor, N. D. Dutt, et al. Data and memory optimization techniques for embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, 6(2):149–206, April 2001.

[14] P. Ranjan Panda, N. Dutt, and A. Nicolau. Incorporating dram access modes into high-level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(2):96–109, February 1998.

[15] J. Seo, T. Kim, and P. Panda. An integrated algorithm for memory allocation and assignment in high-level synthesis. In *Proc. of 39th Design Autom. Conf.*, pages 608–611, 2002.

[16] P. Slock, S. Wuytack, F. Catthoor, and G. D. Jon. Fast and extensive system-level memory exploration for atn applications. In *Proc. of Tenth International Symposium on System Synthesis*, pages 74–81, 1997.