Nonuniform Banking for Reducing Memory Energy Consumption*

Ozcan Ozturk and Mahmut Kandemir
Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA, 16802, USA
{ozturk, kandemir}@cse.psu.edu

Abstract

Main memories can consume a large percentage of overall energy in many data-intensive embedded applications. The past research proposed and evaluated memory banking as a possible approach for reducing memory energy consumption. One of the common characteristics/assumptions made by most of the past work on banking is that all the banks are of the same size. While this makes the formulation of the problem easy, it also restricts the potential solution space. Motivated by this observation, this paper investigates the possibility of employing nonuniform bank sizes for reducing memory energy consumption. Specifically, it proposes an integer linear programming (ILP) based approach that returns the optimal nonuniform bank sizes and accompanying data-to-bank mapping. It also studies how data migration can further improve over nonuniform banking. We implemented our approach using an ILP tool and made extensive experiments. The results show that the proposed strategy brings important energy benefits over the uniform banking scheme, and data migration across banks tends to increase these savings.

1. Introduction

Behavior of many data intensive embedded systems is dictated by their data access pattern, making memory system performance critical. In fact, the past research [1, 2] showed that a significant portion of overall execution energy (excluding input/output units) is spent in memory system. While circuit level techniques targeting energy efficiency are extremely important, high level approaches [3, 4, 5, 6] at the architectural and software levels can also play a major role in shaping the overall memory energy behavior and optimizing it.

Recently, *banking* has been used as a popular method for reducing memory energy consumption. In banking, memory space is divided into multiple banks, each of which can be controlled independently of the others. Experiments performed by several research groups [7, 8, 9, 10] reported significant reductions in memory energy due to banking. This is mainly because each read/write access in a banked memory architecture experiences a smaller capacitance, as compared to a monolithic unbanked memory system. Another advantage of the banked architecture (especially in the context of DRAMs) is low-power operating modes that can be used to place a bank into the low-power mode if it is not being actively used. Several efforts were already made to determine when and how the banks should be powered down, and also which low-power mode to employ when more than one mode are available. Broadly speaking, these methods are either hardware [7] or software [8, 9] oriented.

One of the common characteristics/assumptions of these prior banking related studies is that most of them assume all the banks are of the same size, i.e., they are uniform. While this makes the formulation of the problem easy, it also prevents some further opportunities by restricting the data mapping unnecessarily. In particular, in embedded systems that execute a single application, one can come up with a customized banking strategy where the banks can be of different sizes. Such a banking is referred to as *nonuniform banking* in this paper. The first contribution of this paper is to present an ILP (integer linear programming) based approach that decides the best (nonuniform) bank architecture and accompanying data mapping for a given array based embedded application. We show through experiments that working with customized nonuniform memory banks brings significant energy savings over an alternate strategy that works with uniform bank sizes (which is also formulated using ILP).

While our empirical evaluation of the nonuniform banking shows promising results, one can achieve further energy savings by not fixing the location of each data (i.e., its bank) for the entire execution, that is, by migrating data across the banks during the course of program execution. The second contribution of this paper is an ILP formulation of the data migration problem in the nonuniform bank architecture. Our experimental results with data migration show that it brings additional energy savings over nonuniform banking.

This paper is organized as follows. Section 2 gives a brief overview of banked memory architecture and explains our

* This work is supported in part by NSF Career Award #0093082

Operating Mode	Energy Consumption (nJ)	Resynchronization Cost (cycles)
Active	3.570	0
Standby	0.830	2
Nap	0.320	30
Power-down	0.005	9000

Table 1. Energy consumptions (per cycle) and resynchronization costs for our operating modes.

data access pattern extraction strategy. Section 3 describes our ILP variables, constraints, and overall problem formulation. Section 4 introduces the experimental platform, and presents the results. Section 5 concludes with a summary of the paper.

2. Architecture and Access Pattern Extraction

Our architectural model is based on a multi-bank memory system, which is similar to RDRAM [11]. In this model, each memory bank can be in one of four operating modes: *active*, *standby*, *nap* or *power-down*. These memory banks can be placed into low-power modes independently. Memory access requests (read/write) are serviced only if the corresponding bank is active, that is, the bank is not in one of the low-power operating modes (standby, nap, power-down).

Each low-power mode has different energy consumption (per cycle) and resynchronization cost, since low-power operating modes are typically implemented by disabling certain parts of the DRAM chip. Bringing a low-powered memory bank back to the active mode causes a resynchronization cost (wake-up penalty), which makes a tradeoff analysis between performance and energy inevitable. Energy consumption (per cycle) and resynchronization costs for typical, RDRAM-like low-power operating modes are given in Table 1. As can be seen from this table, when choosing a low-power mode for an idle bank, a tradeoff analysis between energy savings (by using a more aggressive mode) and performance penalty (caused by this aggressive mode due to increase in cycles) has to be performed. If the bank in question is predicted to be idle for sufficiently long time, a more power saving operating mode should be selected. This follows from the fact that frequent transitions between low-power and active modes may result in non-tolerable performance penalties. Hence, a suitable low-power operating mode should be selected based on bank inter-access time (the time between successive accesses to the same memory bank).

Embedded programs constructed using loop nests (with compile time known bounds) and array accesses (with affine subscript expressions) are the main focus in this paper. Such codes frequently occur in embedded image/video processing domain [1]. An optimizing compiler can analyze these loop-intensive applications with regular data access patterns. Since frequent transitions between low-power modes and active mode can be very costly, one early design decision that we made is to perform these mode transitions at well-defined program points. In our implementation, these transition points, which delimit the boundaries of execution phases, are expressed in terms of loop iterations. In this paper, we adopted the concept of a *step* to define these transition points. Although, in theory, we have the flexibility to assign any number of iterations between two transition points, these points should be selected carefully. That is, in moving from one point to another during execution, the data access pattern should exhibit a significant change. To make these steps explicit we need to modify the input code as explained below.

An example loop nest that accesses an array *X* through two references with affine subscript expressions $X[i+2,j-1]$ and $X[i,j]$ is shown in Figure 1. The unit of data that is being stored in banks (and also the unit of data migration across banks when migration is used) is a *data block* (*data tile*). Figure 2 shows a two-dimensional array that is logically divided into data blocks. All the data blocks have the same size except possibly at the boundaries of the array. The blocked (tiled) version of the original loop nest is given on the right hand side of the Figure 1. In this code, loops ii and jj iterate over the data blocks, and are referred to as the *block (inter-tile) iterators*. Also, ii and jj are called *intra-tile iterators*, and iterate over the elements of a given data block. A main factor which effects the data access pattern is the data block size. In our experiments, we manually selected the suitable data blocks and their sizes. An optimizing compiler can be used in the future to automate the data access pattern extraction.

3. ILP Formulation

ILP provides a set of techniques that solve those optimization problems in which both the objective function and constraints are linear functions and the solution variables are restricted to be integers. The 0-1 ILP (also known as ZILP) is an ILP problem in which each (solution) variable is restricted to be either 0 or 1 [12]. It is used in this paper for determining the optimal bank sizes, data to bank mapping, and data migration pattern. Table 2 gives the constant

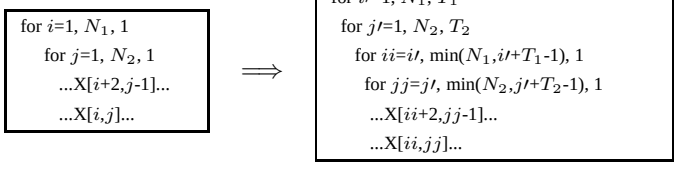

Figure 1. An example loop nest written in a pseudo high-level language (left) and its blocked (tiled) version (right). Each data block (tile) is of size $T_1 \times T_2$ array elements, and the transformed loop nest is structured based on this tile size.

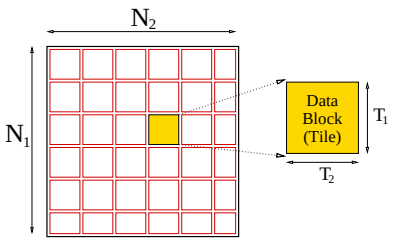

Figure 2. Dividing a two-dimensional array into data blocks (tiles).

terms used in our ILP formulation. Bank energies given in this table (i.e., AE, IE, IE_S, IE_N and IE_P) are shown as energy consumption per step.

Our presentation is in two parts. In the first part, we are partitioning the given memory space into nonuniform banks in the most energy efficient way, and determine optimal location (banks) for each data block. The second part incorporates data migration into the nonuniform bank strategy. In both cases, data-to-bank mapping is optimized.

3.1. Memory Partitioning and Data Assignment

Our objective is to partition the available memory into nonuniform (sized) banks to minimize the overall energy consumption. We determine the number of banks and their sizes based on the data access pattern and the total memory size (to be partitioned) using 0-1 variables. For each possible bank size, we define 0-1 variables. By using these 0-1 variables, we determine the partitions (banks) and their contents. Although, we restrict the possible sizes to be powers of two to simplify the problem, it can be extended to cover all possible sizes. This way we keep the problem size smaller (less variables and constraints) and by doing so, we reduce the ILP solution time. If, for example, the memory size in question is $4k$ (assuming that k is a power of two) and the minimum bank size possible is k, then we will have 0-1 variables for one $4k$-bank, two $2k$-bank and 4 $1k$-bank. If two of the four $1k$-bank variables and one $2k$-bank variable are returned as 1 from the ILP solver, then we conclude that the application in question will spend the minimum energy when the memory space is partitioned into three banks of sizes k, k and $2k$. Our formulation also gives the optimum mapping of the data blocks to these banks.

In our implementation, we identify memory banks with a pair of attributes (l, n): l is used for its size (in terms of data blocks it can hold), and n is used for distinguishing a bank from the other banks that have the same size. For example, with a memory of size $8k$, there can be two banks with a size of $4k$, where k is the minimum bank size possible. These two banks can be expressed using the pairs $(4k, 1)$ and $(4k, 2)$.

Assuming that N denotes the maximum number of banks possible if each bank holds only one data block (which occurs when the minimum possible bank size is used), S the number of steps, M the number of data

Constant	Definition
N	Number of memory banks
S	Number of steps
M	Number of data blocks
σ	The rate of energy consumption by the size of bank
$R_{m,s}$	Indicates whether block m is accessed at step s
$size_{block}$	Size of a data block
AE	Energy consumed by an accessed memory bank per step
IE	Energy consumed by a low-power memory bank per step
IE_S	Energy consumed by a *standby* memory bank per step
IE_N	Energy consumed by a *nap* memory bank per step
IE_P	Energy consumed by a *power-down* memory bank per step
RE	Energy consumed for reactivation of a memory bank
RE_S	Energy consumed for activating a *standby* memory bank
RE_N	Energy consumed for activating a *nap* memory bank
RE_P	Energy consumed for activating a *power-down* memory bank
DE	Energy consumed for deactivation of a memory bank
DE_S	Energy consumed for switching to *standby* mode
DE_N	Energy consumed for switching to *nap* mode
DE_P	Energy consumed for switching to *power-down* mode
ME	Energy consumed for migrating a data block

Table 2. The constant terms used in our ILP formulation. These are either architecture specific or program specific.

blocks, we can use 0-1 variables to specify potential location (L) of each data block. Specifically,

- $L_{m,l,n}$: indicates whether data block m is residing in bank (l, n).

There are possibly $\frac{N}{l}$ banks of size l ($\frac{N}{l}$ at most if all of the banks are of size l). We use a variable for each one of these bank candidates. If this 0-1 variable is 1 then we conclude that the corresponding bank actually exists. We specify whether bank (l, n) actually exists (returned by our ILP formulation) by using $E_{l,n}$.

- $E_{l,n}$: indicates whether data bank (l, n) exists.

In our ILP formulation, we use variable A to identify if the bank is currently active.

- $A_{l,n,s}$: indicates whether bank (l, n) is active at step s due to a read/write operation on it.

Although it is better to have a bank in low-power operating mode from the energy perspective, depending on the data access pattern, it might be better not to go to a low-power operating mode for performance reasons. As a result, frequent switchings between low-power modes and active mode in short intervals should be avoided. Our next set of 0-1 variables are used to capture the switchings between the active and low-power modes for a given bank. If a bank is activated from a low-power mode, the following variable is set to 1:

- $X_{l,n,s}$: indicates whether bank (l, n) is activated at step s.

On the other hand, if a bank is switched to a low-power mode, $Y_{l,n,s}$ will be set to 1. In other words,

- $Y_{l,n,s}$: indicates whether bank (l, n) goes to the low-power mode at step s.

After having defined our 0-1 integer variables, we can now discuss our ILP formulations. The following constraints are

needed to capture the values of $X_{l,n,s}$ and $Y_{l,n,s}$. A bank (l, n) is activated at step s, if it is not active at step $(s-1)$ and is active at step s:

$$X_{l,n,s} \geq A_{l,n,s} - A_{l,n,s-1}, \quad \forall l, n, s. \tag{1}$$

A bank (l, n) goes to a low-power mode at step s if it is active at step $(s-1)$ but it is not active at step s:

$$Y_{l,n,s} \geq A_{l,n,s-1} - A_{l,n,s}, \quad \forall l, n, s. \tag{2}$$

In our experiments, we assume that bank sizes are restricted to be powers of two. For example, if the memory in question can hold at most 8 data blocks then this memory can be partitioned into memory banks such that each bank can hold 1,2,4 or 8 data blocks. Since available memory space is partitioned among banks, the total size of the banks (determined by the ILP solver) should be equal to the available memory space:

$$\sum_{i=0}^{\log_2 N} \sum_{j=1}^{\frac{N}{2^i}} E_{2^i,j} \times 2^i = N. \tag{3}$$

A data block can be residing in a bank only if the bank in question exists:

$$E_{l,n} \geq L_{m,l,n}, \quad \forall m, l, n. \tag{4}$$

Since a data block can be residing only in a single bank at any given time, it must satisfy the following constraint:

$$\sum_{i=0}^{\log_2 N} \sum_{j=1}^{\frac{N}{2^i}} L_{m,2^i,j} = 1, \quad \forall m. \tag{5}$$

The limited bank capacity forms the basis for the next constraint that needs to be satisfied. Assuming that the size of a block is $size_{block}$, the following expression can be written for each possible bank size l:

$$size_{block} \times \sum_{i=1}^{M} L_{i,l,n} \leq l, \quad \forall l, n. \tag{6}$$

A bank is currently active (at step s) if one of its data blocks is accessed:

$$A_{l,n,s} \geq R_{m,s} \times L_{m,l,n}, \quad \forall m, l, n, s. \tag{7}$$

$R_{m,s}$ indicates whether block m is accessed at step s. Its value is extracted from the data access pattern.

Having specified the necessary constraints in our ILP formulation, we next give our objective function. In our execution model, there are four components of the total memory energy consumption:

• *active:* the energy consumed when a bank is read/written.

• *low-power:* the energy consumed when a bank is in the low-power mode.

• *activation:* the energy consumed to activate a bank from the low-power mode.

• *deactivation:* the energy consumed to switch a bank to the low-power mode.

Based on these, we can express memory energy consumption (*B*) as follows:

$$\begin{aligned} B \quad = \quad & \sum_{i=0}^{\log_2 N} \sum_{j=1}^{\frac{N}{2^i}} \sum_{k=1}^{S} (A_{i,j,k} \times AE + (1 - A_{i,j,k}) \times IE \\ & + X_{i,j,k} \times RE + Y_{i,j,k} \times DE) \times \sigma^i. \end{aligned} \tag{8}$$

In this formulation, AE, IE, RE and DE correspond to active energy, low-power energy, activation energy, and deactivation energy, respectively. Note that, σ is used to capture the energy consumption behavior of the banks with different sizes. For example, if the size of the bank is doubled the energy spent would be σ times of the original energy consumption. Based on this formulation, our 0-1 ILP problem can be defined as one of "minimizing *B* under constraints (1) through (7)."

3.2. Data Migration

Further energy improvements can be achieved by migrating data blocks among banks during the course of execution. In particular, in some cases, instead of activating/deactivating a bank, it might be more beneficial to transfer the data block to an already active bank. To incorporate data migration into our ILP formulation, several modifications to the original problem have to be made.

Since the location of a data block is no longer restricted to be the same throughout the execution of the program (due to migration), we have to include the step parameter, s, in the location variable, L. For this purpose, $L_{m,l,n}$ is replaced with $L_{m,l,n,s}$ to indicate whether block m is residing in bank (l, n) at step s.

Equations (4),(5) and (6) should also be written for every step to capture bank existence constraint, data block location constraint (a data block can be in a single bank at any time) and bank size constraint. Equation (7), on the other hand, does not require any change except for the variable renaming, since it is already expressed for every step s.

Migration behavior of data blocks needs to be captured as well. We use $Z_{m,s}$ to serve for this purpose. A data block m migrates (at step s) if its current location (bank) is different from its previous location (i.e., the one in the previous step):

$$Z_{m,s} \geq L_{m,l,n,s} - L_{m,l,n,s-1}, \quad \forall m, l, n, s. \tag{9}$$

Although data migration can reduce the energy consumption (as it can empty banks which can be placed into the low-power mode), it also brings additional energy consumption due to migrating data blocks. This overhead has to be included in the ILP formulation:

$$T = \sum_{i=1}^{M} \sum_{j=1}^{S} (Z_{i,j} \times ME). \tag{10}$$

Then, our new 0-1 ILP problem can be defined as "minimizing $B+T$ under constraints (1) through (7) and (9)."

4. Experimental Evaluation

4.1. Setup

We performed several experiments with five array-intensive benchmarks to test the effectiveness of our ILP-based approach. Table 3 lists the benchmark codes used in this study and their important characteristics (taken from the Spec and Perfect Club benchmark suites). The second column gives the total size of the data processed by each benchmark, and the third column shows the data block size used. The last column gives the number of arrays for each benchmark. The default simulation parameters used in our experiments are given in Table 4. Note that, the energy values given in Table 4 are normalized to power-down energy. The low-power energy values given in this table reflect the actual energy consumption values from Table 1. For example, the ratio between the power-down bank energy and the active bank energy is 1:714, which corresponds to 0.005 nJ / 3.570 nJ. During activation (from a low-power mode to the active mode) and deactivation (from the active mode to a low-power mode), $\frac{3}{4}$ of the active mode energy is assumed to be spent.

We performed experiments with three different execution models for each benchmark code in our experimental suite:

• *Uniform Banking:* This in a sense is the conventional memory space management strategy which has a fixed bank size and that does not use any data migration. Data blocks are placed into uniform memory banks and they do not move during the course of execution. Even though banks are uniform, we still implement this scheme using ILP to obtain the optimum data-to-bank mapping and minimum energy under this banking. The only difference from our initial ILP formulation is in selecting the bank sizes and the number of banks. Specifically, the bank sizes and the number of banks are given as constants to the ILP tool. In this case, Equation (3) and Equation (4) become unnecessary, and the start and end values of index variables i and j in Equation (5) and Equation (8) need to be changed such that they include only the specified bank size and the given number of banks.

Benchmark Name	Total Data Size (KB)	Data Block Size (KB)	Number of Arrays
adi	468.8	39.1	6
apsi	78.2	19.5	17
bmcm	234.4	19.5	11
btrix	75.0	11.7	29
wss	70.8	17.7	10

Table 3. The benchmark codes used in this study.

Parameter	Value
Number of Banks	4
Data Block Size : Memory Bank Capacity	1:1,1:2,1:4,1:8
σ	1.3
Bank Energy (Power-down)	1
Bank Energy (Nap)	64
Bank Energy (Standby)	166
Bank Energy (Active)	714
Bank Activation Energy	535.5
Bank Deactivation Energy	535.5
Data Migration Energy	142.8

Table 4. The default simulation parameters.

• *Nonuniform Banking:* This is the integer linear programming based strategy discussed in this paper wherein banks with different sizes are employed. The data blocks frequently accessed together can be placed in a larger bank, whereas a single data block (without any access pattern relationship with other data blocks) can be placed in a smaller sized bank to optimize the energy consumption in active banks and during bank activations/deactivations.

• *Migration:* This is an extension to the previous strategy (see Section 3.2) where data migration is employed to decrease the energy consumption further. An accessed data block is migrated to an active bank if doing so is profitable from the energy consumption point of view. The rationale behind this scheme is to transfer the data block being accessed to one of the active memory banks, thereby cutting the number of active memory banks as much as possible.

4.2. Results

Table 5 gives the results based on power-down operating mode (i.e., when, as a low-power mode, we use only the power-down mode). Note that, all the values are relative to the ratios given in Table 4. Energy improvement brought by the Nonuniform Banking version over the Uniform Banking version for different bank sizes are given in columns two through five. Bank sizes are given as ratios to a data block size as given in Table 4. For example, in the third column (titled '2'), we compare our approach with the Uniform Banking strategy with banks that have fixed sizes of $2k$ (assuming that a data block size is k). In our approach, a total of $8k$ memory is partitioned nonuniformly. However, in uniform banking scheme, memory is composed of four different banks where all of the banks are of size $2k$. The sixth column gives the average improvement brought by our approach over the uniform banking scheme. We see that the overall average reduction in energy consumption is 10.4%, showing the benefits of nonuniform banking. ILP solution times for our approach range from 163 sec (btrix) to 64 min (bmcm). As it was stated earlier, we used also ILP to obtain the results for the Uniform Banking approach in order to make a fair comparison against our approach. Performance results are given in Figure 3. These values are given as improvements brought by the Nonuniform Banking approach over the Uniform Banking scheme with different bank sizes. Negative values indicate that there is a performance overhead if our approach is used. The average performance im-

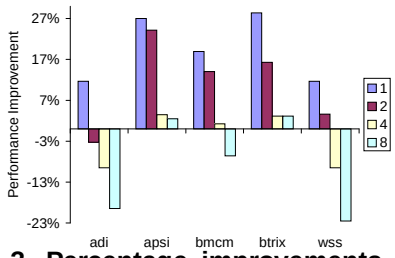

Figure 3. Percentage improvements in performance brought by the Nonuniform Banking approach over the Uniform Banking approach.

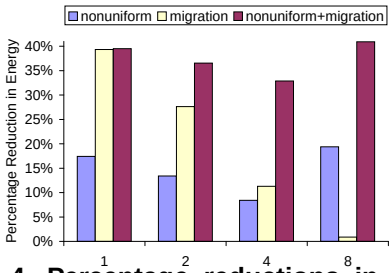

Figure 4. Percentage reductions in energy brought by different schemes.

provement values are -5%, 14%, 7%, 13% and -4% for adi, apsi, bmcm, btrix and wss, respectively. One can observe from this figure that the maximum overhead brought by our approach is 22% for wss with a bank of size 8. On the other hand, maximum improvement brought by our approach is 28% for btrix with a bank of size 1. In general, the larger the bank size, the better the performance is. This follows from the fact that the number of banks to activate (if the corresponding bank is in a low-power operating mode) decreases if a larger bank size is preferred. Figure 4 shows the reduction in energy consumption for `btrix` benchmark when only Nonuniform Banking is used, when Uniform Banking is used with data migration, and when Nonuniform Banking is used along with data migration. Each bar represents percentage reduction in energy over the Uniform Banking scheme without data migration. We see that Nonuniform Banking reduces energy by 15% and data migration (on a uniform bank) reduces by 20% when used separately. If these two approaches are combined to form the Nonuniform Banking scheme with data migration strategy, then the energy reduction reaches 38% on average.

5. Conclusion

Memory system can be a major energy consumer in embedded systems. Therefore, recent years have witnessed several studies that target at reducing memory energy consumption. Banking is such a scheme and has received a lot of attention recently from both hardware and software communities. One of the common characteristics/assumptions made by most of the past work on banking is that all the banks are of the same size. While this makes the formulation of the problem easy, it also restricts the potential solution space. This paper investigates the possibility of employing nonuniform bank sizes for reducing memory energy consumption. We show through experiments that working with customized nonuniform memory banks brings important savings over the uniform banking strategy. The second contribution of this paper is a formulation of the data migration problem. Our experiments with data migration show that it brings additional energy savings over nonuniform banking.

Benchmark	*Bank Size*				*Average*
	1	**2**	**4**	**8**	
`adi`	6.0%	1.4%	7.2%	8.2%	5.7%
`apsi`	8.8%	16.2%	9.2%	19.0%	13.3%
`bmcm`	6.7%	9.9%	9.2%	11.1%	9.2%
`btrix`	17.4%	13.4%	8.4%	19.4%	14.7%
`wss`	10.3%	8.0%	8.9%	9.2%	9.1%

Table 5. Percentage reductions in energy brought by Nonuniform Banking approach over the Uniform Banking approach with varying (fixed) bank sizes.

References

[1] F. Catthoor et al., *Custom Memory Management Methodology,* Kluwer Academic Publishers, 1998.

[2] K. I. Farkas, J. Flinn, G. Back, D. Grunwald, J. M. Anderson, "Quantifying the energy consumption of a pocket computer and a Java virtual machine," In Proc. *2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2000.

[3] Y. Cao, H. Tomiyama, T. Okuma, H. Yasuura, "Data memory design considering effective bitwidth for low-energy embedded systems," In Proc. *the 15th international symposium on System Synthesis*, 2002.

[4] A. Sudarsanam, S. Malik, "Simultaneous Reference Allocation in Code Generation for Dual Data Memory Bank ASIPs," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 5, pp. 242-264, 2000.

[5] M. A. R. Saghir, P. Chow, C. G. Lee, "Exploiting dual data-memory banks in digital signal processors," In Proc. *International conference on Architectural Support for Programming Languages and Operating Systems*, 1996.

[6] P. R. Panda, "Memory Bank Customization and Assignment in Behavioral Synthesis," In Proc. *IEEE/ACM international conference on Computer-aided design*, 1999.

[7] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, M. J. Irwin, "DRAM energy management using software and hardware directed power mode control," In Proc. *the 7th International Conference on High Performance Computer Architecture*, 2001.

[8] A. R. Lebeck, X.Fan, H. Zeng, C. S. Ellis, "Power aware page allocation," In Proc. *the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[9] V. Delaluz, M. Kandemir, U. Sezer, "Improving Off-Chip Memory Energy Behavior in a Multi-processor, Multi-bank Environment," In Proc. *the 14th Annual Workshop on Languages and Compilers for Parallel Computing*, 2001.

[10] A. H. Farrahi, G. E. Tellez, M. Sarrafzadeh, "Memory Segmentation to Exploit Sleep Mode Operation," In Proc. *the 32nd conference on Design automation*, 1995.

[11] 128/144-MBit Direct RDRAM Data Sheet, Rambus Inc., 1999.

[12] G. Nemhauser, L. Wolsey. *Integer and Combinatorial Optimization*, Wiley-Interscience Publications, 1988.