

Minimizing Energy Consumption of Banked Memories Using Data Recomputation

H. Koc

O. Ozturk, M. Kandemir,
S.H.K. Narayanan

E. Ercanli

Department of EECS
Syracuse University
hkoc@ecs.syr.edu

Department of CSE
The Pennsylvania State University
{ozturk, kandemir, snarayan}
@cse.psu.edu

Department of EECS
Syracuse University
eercanli@ecs.syr.edu

Abstract

Banking has been identified as one of the effective methods using which memory energy can be reduced. We propose a novel approach that improves the energy effectiveness of a banked memory architecture by performing extra computations if doing so makes it unnecessary to reactivate a bank which is in the low-power operating mode. More specifically, when an access to a bank, which is in the low-power mode, is to be made, our approach first checks whether the data required from that bank can be recomputed by using the data that are currently stored in already active banks. If this is the case, we do not turn on the bank in question, and instead, recalculate the value of the requested data using the values of the data stored in the active banks. Given the fact that the contribution of the leakage consumption to overall energy budget keeps increasing, the proposed approach has the potential of being even more attractive in the future. Our experimental results collected so far clearly show that this recomputation based approach can reduce energy consumption significantly.

©ACM, 2006. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in the proceedings of ISLPED'06. DOI Bookmark: <http://doi.acm.org/10.1145/1165573.1165658>.

This work is supported in part by NSF career award #0093082 and a grant from the GSRC.

Minimizing Energy Consumption of Banked Memories Using Data Recomputation *

H. Koc
 Department of EECS
 Syracuse University
 hkoc@ecs.syr.edu

O. Ozturk, M. Kandemir,
 S.H.K. Narayanan
 The Pennsylvania State
 University
 {ozturk, kandemir,
 snarayan}@cse.psu.edu

E. Ercanli
 Department of EECS
 Syracuse University
 eercanli@ecs.syr.edu

ABSTRACT

Banking has been identified as one of the effective methods using which memory energy can be reduced. We propose a novel approach that improves the energy effectiveness of a banked memory architecture by performing extra computations if doing so makes it unnecessary to reactivate a bank which is in the low-power operating mode. More specifically, when an access to a bank, which is in the low-power mode, is to be made, our approach first checks whether the data required from that bank can be recomputed by using the data that are currently stored in already active banks. If this is the case, we do not turn on the bank in question, and instead, recalculate the value of the requested data using the values of the data stored in the active banks. Given the fact that the contribution of the leakage consumption to overall energy budget keeps increasing, the proposed approach has the potential of being even more attractive in the future. Our experimental results collected so far clearly show that this recomputation based approach can reduce energy consumption significantly.

Categories and Subject Descriptors:B.3 [Memory Structures]: Miscellaneous

General Terms: Experimentation, Management, Algorithms.

Keywords: Energy, memory bank, multiple operating modes.

1. INTRODUCTION

Banking has been identified as one of the effective methods using which memory energy can be reduced. Two factors contribute to this: reduced effective capacitance brought by banking as compared to a single monolithic memory and existence of multiple low-power operating modes that work with banked memories. As a result, prior research has shown that banking can reduce memory energy for different types of applications in the context of both embedded and high end computing platforms. Several studies have also proposed techniques - in hardware and/or software - that help

*This work is supported in part by NSF career award #0093082 and a grant from GSRC.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ISLPED'06, October 4–6, 2006, Tegernsee, Germany.
 Copyright 2006 ACM 1-59593-462-6/06/0010 ...\$5.00.

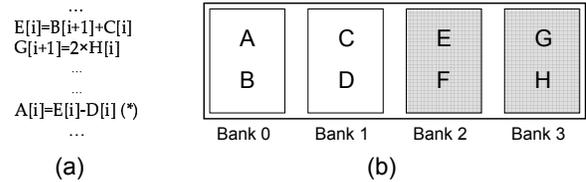


Figure 1: An example scenario with four banks. a) Sample code fragment, b) Data mapping into arrays

increase the effectiveness of low-power operating modes.

This paper builds upon this prior work and proposes a novel approach that further improves energy effectiveness of a banked memory architecture by performing *extra computations* if doing so makes it unnecessary to reactivate a bank which is in the low-power operating mode. More specifically, when an access to a bank which is placed into a low-power mode is to be made, our approach first checks whether the data required from that bank can be recomputed by using the data that are currently stored in already active banks. If this is the case, we do not turn on the bank in question, and instead, recalculate the value of the data using the values of the data stored in the active banks. Given the fact that the contribution of leakage consumption to overall energy budget keeps increasing, the proposed approach has the potential of being even more attractive in the future.

Figure 1 illustrates the proposed approach using an example. Consider the code fragment shown in Figure 1(a), which appears in the body of a loop whose index is i . Let us assume that the memory system has four banks and, at a particular moment in execution, only two of these banks (Bank 0 and Bank 1) are active, whereas the remaining two (Bank 2 and Bank 3) are put in the low-power operating mode. Figure 1(b) depicts how the data manipulated by the code fragment in Figure 1(a) are mapped to our banks. Assuming now that the execution is currently about the touch the statement marked using “*”, the first data element to be accessed is $E[i]$, which is stored in Bank 2. Now, we have two options: either transition Bank 2 to the active (fully operational) mode and access $E[i]$ from there or recompute the value of $E[i]$ using arrays $B[i+1]$ and $C[i]$, which are stored in Bank 0 and Bank 1, respectively. Our approach chooses the second option if it is beneficial to do so from the energy perspective.

We implemented the proposed approach and performed experiments with six array-intensive benchmark programs. Our experimental results collected so far clearly show that this recomputation based approach can reduce energy consumption by about 12.8% when averaged over all codes in our experimental suite.

Power Mode	Dynamic Energy/Access (nJ)	Leakage Energy/Bit (fJ)	Resynch. Cost (cycles)
Active	0.294	1.369	0
Power-down	-	0.218	2

Table 1: Energy consumptions and resynchronization costs for our operating modes, given for a bank size of 64KB.

The rest of this paper is organized as follows. Section 2 introduces the banked memory architecture we consider in this work and discusses the related work on exploiting low-power operating modes for memory energy saving. Section 3 presents our compiler algorithm that implements selective recomputation for energy saving and explains how it operates using an example. Section 4 presents the data we collected from our experiments. Section 5 concludes the paper and briefly discusses the planned future work.

2. BANKED MEMORY AND RELATED WORK

Our architectural model is based on a multi-bank memory system, in which SRAM banks can be placed into low-power modes independently. More specifically, each memory bank can be in one of two operating modes: *active* or *power-down* (also called *drowsy*) at any point during execution. Read/write requests are serviced only in the active mode.

Low-power operating modes are typically implemented by disabling certain parts of the memory chip. In addition to this, the different implementation techniques such as gated- V_{dd} [12], ABB-MTCMOS [10], and dynamic voltage scaling [6] have been utilized to enable low-power operating modes in SRAM circuits. Consequently, each low-power mode can have a different energy consumption and a different resynchronization cost (i.e., the penalty of reactivation) from the rest. This reactivation penalty is mainly due to bringing a low-powered memory bank back to the active mode. Table 1 shows energy consumption and resynchronization costs used in our evaluation. The energy consumed during transition from power-down to active mode is 25.6pJ. One must perform tradeoff analysis between energy savings and performance penalty when choosing a low-power mode for an idle bank. The time between successive accesses to the same memory bank is the main factor in selecting the most suitable operating mode.

Power management of banked memories have been investigated from the different angles including hardware, OS, and compiler. Delaluz et al [3] investigated software and hardware techniques to exploit the memory mode control capabilities. Using memory access patterns in embedded systems, [1] proposed an algorithm to partition on-chip SRAM into multi-banks that can be accessed independently. Flautner et al [6] used a simple technique to put cold cache lines into a state preserving low power mode to improve the leakage power consumption. Kim et al [8] extended this technique for instruction caches. Fan et al [4] presented memory controller policies for memory architectures with low-power operating modes. The impact of classical loop optimizations on energy consumption of banked memories has been evaluated in [7]. Farrahi et al [5] discussed how a sleep mode can be exploited for memory partitions. The impact of loop optimizations (loop splitting and loop distribution) and array placement strategies on a banked off-chip memory architecture are presented in [2]. Lyuh and Kim [9] used a compiler directed approach to determine the operating modes of memory banks after scheduling the memory access operations. Panda [11] addressed the problem of incorporating the application-specific customization of memory bank configuration into behavioral synthesis. In comparison, the work described in this paper shows that it is possible to further increase energy savings of a banked memory by using recomputation, on top of the

energy savings coming from exploiting the available low-power operating modes.

3. DETAILS OF OUR APPROACH

3.1 Energy Evaluation Model

In this section, we present the energy model used in this work. Please note that we tried to keep our model simple and at the high level for the sake of clarity. Total energy consumption E_{total} of an SRAM-based memory system can be modeled, at a high level of abstraction, as summation of dynamic, $E_{dynamic}$, (during access) and static, $E_{leakage}$, (during access and idle periods) components, in addition to the energy spent during transitions, E_{trans} , between operating modes (comprising both dynamic and static energy components).

The first component of the equation, $E_{dynamic}$ is directly proportional to the number of accesses, N_{access} , to the memory. Then, we multiply the number of accesses with the required energy per memory access, DE_{access} , to determine dynamic energy consumption, $E_{dynamic}$. In this model, the energies consumed during memory reads and that during writes are assumed to be equal.

$$E_{dynamic} = N_{access} \times DE_{access}$$

On the other hand, the leakage energy consumption, $E_{leakage}$, is proportional to the capacity of the memory system, M_{size} , and the number of cycles, N_{cycles} , required for the execution of the application. Since our memory system has two different operating modes, we need to consider the percentage of the execution time the memory is in the active mode, P_{active} , and the percentage of the time during which the memory is in the power-down mode ($1 - P_{active}$). Finally, we need to take into account the leakage energy per bit for each power mode. In our case, they are the leakage energy per bit for the active mode, LE_{active} , and the leakage energy per bit for the power-down mode, LE_{down} . Please note that this model could easily be modified to capture the systems with more than one low-power operating modes. The energy equation for leakage is shown below:

$$E_{leakage} = M_{size} \times N_{cycles} \times ((P_{active} \times LE_{active}) + (1 - P_{active}) \times LE_{down})$$

Transition energy, E_{trans} , is calculated by multiplying the number of transitions from power-down mode to active mode, N_{trans} , with the energy consumed per transition, E_{trans} . In addition to this, the number of cycles required for transition, T_{cycles} , should be accounted for when computing the leakage energy consumption (they are captured in N_{cycles}).

$$E_{trans} = N_{trans} \times E_{trans}$$

Note that since the memory model used in this paper has multiple banks, the models for $E_{leakage}$ and E_{trans} should be employed for each bank.

Let us now discuss the benefits and overheads of the recomputation considering the energy model. Recomputation brings extra computations into consideration. This increases the number of memory accesses by N_{access_extra} (number of extra memory accesses) and the execution time by N_{cycles_extra} (number of extra execution cycles). Consequently, the overall energy overhead, E_{over} , is calculated as follows:

$$E_{\text{over}} = N_{\text{access_extra}} \times DE_{\text{access}} + N_{\text{cycles_extra}} \times LE_{\text{active}}$$

On the other hand, recomputation prevents the banks switching back and forth between the active and power-down modes. This, in turn, reduces the number of cycles by $N_{\text{access_less}}$ (the number of cycle reduced); the number of transitions from the power-down to active mode by $N_{\text{trans_less}}$ (the reduced number of transitions from the power-down to active mode); and significantly increases the percentage of the execution time in which a bank is in the power-down mode by P_{gain} . As a result, the energy improvement can be determined as follows:

$$E_{\text{gain}} = N_{\text{trans_less}} \times E_{\text{trans}} + N_{\text{access_less}} \times DE_{\text{access}} + P_{\text{gain}} \times M_{\text{size}} \times (LE_{\text{active}} - LE_{\text{down}})$$

Based on the above-mentioned formulation, our approach can save energy if $E_{\text{gain}} > E_{\text{over}}$.

3.2 Recomputation

This section briefly discusses our recomputation-based approach and presents the algorithm used. Our approach targets embedded applications consisting of loops that manipulate array elements. Particularly, we assume that all scalar variables are stored in registers and hence do not play a part in our analysis. Let code C consist of a series of L loops in which a total of M arrays are manipulated. Let the loops be denoted by L_1, L_2, \dots, L_N , where a loop can be nested at an arbitrary depth. For example, consider the loop L_i to have a nesting depth of k . The l^{th} loop in this nest is denoted by L_{i_l} and the lower and upper bounds of this loop are given by L_{i_S} and L_{i_E} , respectively. Finally, an array reference, as part of an expression, in this loop nest is denoted by $A_{i_l}^R(a_1, a_2 \dots a_l)$ and $A_{i_l}^L(a_1, a_2 \dots a_l)$ for the right hand side and the left hand side, respectively, of the expression. The subscript i_l denotes the loop number (i) and the depth of the nest l . Let us consider a memory architecture which has N banks and let the arrays of the given code be partitioned among these N banks.

Recomputation in our work is defined as the computation of a previously computed value for an array reference instead of looking up the computed value that is stored in a memory bank. This process is performed in three main steps. First, possible recomputation opportunities are found in the given application code. Then, suitable arrays are selected for recomputation based on their performance cost. Total cost of all recomputations must be less than the total allowable Cost-Overhead \mathcal{T}_C that is the maximum number of extra execution cycles allowed. Finally, the code is restructured by replacing the selected array references with the expression that computes its value.

Algorithm 1 gives a sketch of our approach. The input to this algorithm is the original code, the mapping of arrays to the memory banks, and \mathcal{T}_C . The output is the restructured code. First, the set of selected elements, \mathcal{S} , is set to ϕ . Each element of this set is a tuple, formed by the array reference, $A_{i_l}^L(a_1, a_2 \dots a_l)$, that has been selected for recomputation and the set of array references, \mathcal{L} , that compute the values of the same data space as the array reference. After that, in lines [5-18], the set \mathcal{S} is calculated. To calculate it; first, for each array reference in the input code, $A_{i_l}^L(a_1, a_2 \dots a_l)$, the set \mathcal{L} is determined. Then, if the data space accessed by the reference is a subset of that accessed by \mathcal{L} and if the cost is not prohibitive, and finally, if the banks that the reference $A_{i_l}^L(a_1, a_2 \dots a_l)$ accesses are OFF and those that \mathcal{L} accesses are ON, the new tuple $\{(A_{i_l}^L(a_1, a_2 \dots a_l), \mathcal{L})\}$ is added to \mathcal{S} . Otherwise, the bank accessed by $A_{i_l}^L(a_1, a_2 \dots a_l)$ is considered to be ON. Finally, in the

Algorithm 1 *Recomputation()*

```

1: Input : Source code, array mapping to banks, and  $\mathcal{T}_C$ 
2: Output : Restructured code
3:  $\mathcal{S} = \phi$ 
4: Initialize all banks as OFF
5: for all array references  $A_{i_l}^L(a_1, a_2 \dots a_l)$  do
6:    $\mathcal{L} = \phi$ 
7:   for all  $j < i$  and  $\exists A_{i_l}^R(a_1, a_2 \dots a_l)$  do
8:      $\mathcal{L} = \mathcal{L} \cup A_{i_l}^R(a_1, a_2 \dots a_l)$ 
9:   end for
10:  if  $\mathcal{D}_S(A_{i_l}^L(a_1, a_2 \dots a_l)) \subset \mathcal{D}_S(\mathcal{L})$  and
    Bank( $A_{i_l}^L(a_1, a_2 \dots a_l)$ ) = OFF and Bank( $\mathcal{L}$ ) = ON then
11:    if  $\mathcal{T}_C > 0$  then
12:       $\mathcal{S} = \mathcal{S} \cup \{(A_{i_l}^L(a_1, a_2 \dots a_l), \mathcal{L})\}$ 
13:       $\mathcal{T}_C = \mathcal{T}_C - \mathcal{C}(T(A_{i_l}^R(a_1, a_2 \dots a_l)), N(A_{i_l}^R(a_1, a_2 \dots a_l)))$ 
14:    end if
15:  else
16:    Bank( $A_{i_l}^L(a_1, a_2 \dots a_l)$ ) = ON
17:  end if
18: end for
19: for each element  $\{(A_{i_l}^R(a_1, a_2 \dots a_l), \mathcal{L})\}$  of  $\mathcal{S}$  do
20:  for each element  $\mathcal{L}_j \in \mathcal{L}$  do
21:    Calculate the loops bounds  $L_{i_S}$  and  $L_{i_E}$  for
22:     $\mathcal{D}_S(A_{i_l}^R(a_1, a_2 \dots a_l)) \cap \mathcal{D}_S(\mathcal{L}_j)$ 
23:    Restructure code using  $L_{i_S}$  and  $L_{i_E}$ 
24:  end for
25: end for

```

lines [19-25], the code for the each selected array reference is generated. First, the loop bounds are calculated and then, these bounds are used to generate the output code.

4. EXPERIMENTAL EVALUATION

This section presents the experimental evaluation of our proposed approach. We used six array-intensive benchmarks from the Spec and Perfect Club benchmark suites. Each benchmark is written in C language and comprises the manipulations of one- to three-dimensional data arrays. The data set sizes of these benchmarks range from 170KB to 256KB. In our experiments, we used a single level of SRAM with four data banks, each 64KB, as our memory architecture. Execution time is a significant factor during the calculation of leakage energy consumption. Execution time of each application with/without recomputation is evaluated using an UltraSPARC-III based system with 750 MHz of clock frequency. In our experiments with baseline configuration, we found that the benchmarks incurred around 7% performance degradation on the average when recomputation is introduced.

Our first set of results is given in Table 2, under the performance bound of 20% (i.e., we allow up to 20% increase in original execution cycles). The first column of this table gives the name of the benchmark. The next two columns are the energy consumptions without and with the proposed recomputation based approach. The last column indicates the percentage energy improvements. The average energy improvement obtained through the recomputation based approach is about 12.8%. As can be seen in Table 2, the improvement is small for the first benchmark *bmcm*. This is because the execution time increases almost 20% when recomputation is introduced. This, in turn, increases not only dynamic energy but also leakage.

In our next set of experiments we kept the total memory size (capacity) fixed at 256KB and changed the number of banks. Recall that our experiments shown in Table 2 assume that 4 data banks are available. Figure 2 shows the normalized energy savings with the different bank counts. In this case, we focus only on benchmarks *wss* and *tomcatv*. As can be seen from the figure, an increase in the

Benchmark	Energy w/o Recomp. (mJ)	Energy w/ Recomp. (mJ)	Improvement (%)
bmcmm	212	210	0.6
eflux	786	625	20.5
mxm	133	100	24.4
tomcatv	1240	1120	9.8
interp	3050	2830	7.4
wss	1980	1700	13.8

Table 2: Energy consumptions with and without recomputation for different benchmarks.

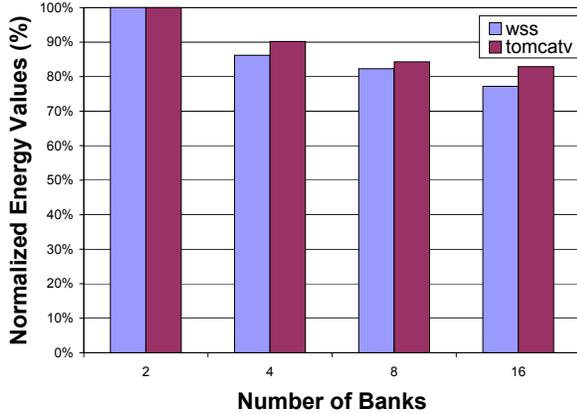


Figure 2: Normalized energy consumptions with different number of banks for wss and tomcatv.

number of banks improves the normalized energy savings archived by our recomputation based multibank memory system. This is due to two main reasons: a) our approach provides an opportunity to power down a bank, which cannot be powered down in the original case due to lifetime conflicts between the data arrays that reside in the bank, and b) our approach makes it possible to keep a bank in the power down mode for a longer period of time. As expected, there is no energy improvement for the benchmarks when only two data banks are available.

Next, we studied the tradeoff between performance and energy. In other words, we evaluated energy consumptions under different recomputation scenarios. Figure 3 plots the normalized energy consumption values for each scenario for the benchmark *tomcatv*. The first bar with 0% performance degradation represents the case with no recomputation. Each bar after that introduces extra recomputation over the previous one (i.e., it increases the tolerable increase in the execution cycles). As can be observed from the figure, when the amount of recomputation is increased, it is possible to achieve better energy savings (the cases with 10% and 20% performance degradation bound). However, the case with 40% performance degradation bound has a negative effect in energy consumption. The main reason for this is the increase in leakage energy consumption due to excessive execution time overhead.

5. CONCLUSIONS AND FUTURE WORK

Low-power operating modes have been employed to improve the energy consumption of banked memory architectures. Reduction in energy consumption is achieved by putting inactive memory banks into a low-power operating mode. In this paper, we propose a novel energy saving scheme based on recomputation to further increase the energy savings that could be obtained from in multi-bank memory systems. Basically, when an access to a powered-down bank is to be made, our approach first checks if it is possible to obtain the

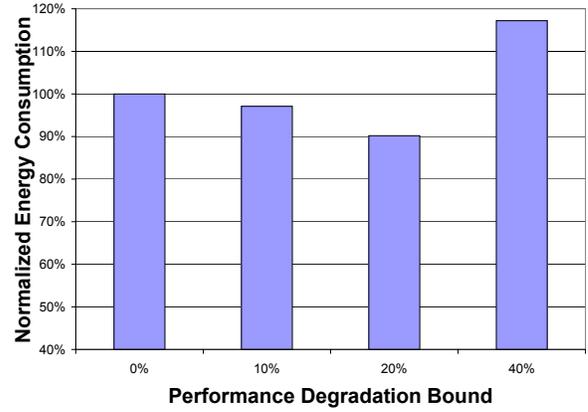


Figure 3: Normalized energy consumptions with different recomputation scenarios for tomcatv.

required data by recomputing the data placed in the active banks, instead of reactivating the powered-down bank and fetching the data from it. We performed experiments with several benchmarks and the results collected show the effectiveness of the proposed recomputation based approach. Our future work involves extending this idea for multiple low-power operating modes with compiler-directed recomputation decisions.

6. REFERENCES

- [1] L. Benini, A. Macii, and M. Poncino. A recursive algorithm for low-power memory partitioning. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 78–83, New York, NY, USA, 2000. ACM Press.
- [2] V. Delaluz, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Energy-oriented compiler optimizations for partitioned memory architectures. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 138–147, 2000.
- [3] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin. Dram energy management using software and hardware directed power mode control. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, page 159, 2001.
- [4] X. Fan, C. Ellis, and A. Lebeck. Memory controller policies for dram power management. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 129–134, 2001.
- [5] A. Farrahi, G. Tellez, and M. Sarrafzadeh. Exploiting sleep mode for memory partitions and other applications. In *Proceedings of the VLSI Design*, pages 271–287, 1998.
- [6] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. In *Proceedings of the Annual International Symposium on Computer Architecture*, pages 148–157, Washington, DC, USA, 2002.
- [7] M. Kandemir, I. Kolcu, and I. Kadayif. Influence of loop optimizations on energy consumption of multi-bank memory systems. In *Proceedings of the International Conference on Compiler Construction*, pages 276–292, 2002.
- [8] N. S. Kim, K. Flautner, D. Blaauw, and T. Mudge. Drowsy instruction caches: leakage power reduction using dynamic voltage scaling and cache sub-bank prediction. In *Proceedings of the Annual International Symposium on Microarchitecture*, pages 219–230, Los Alamitos, CA, USA, 2002.
- [9] C.-G. Lyuh and T. Kim. Memory access scheduling and binding considering energy minimization in multi-bank memory systems. In *Proceedings of the Annual Conference on Design Automation*, pages 81–86, New York, NY, USA, 2004.
- [10] K. Nii, H. Makino, Y. Tujihashi, C. Morishima, Y. Hayakawa, H. Nunogami, T. Arakawa, and H. Hamano. A low power SRAM using auto-backgate-controlled MT-CMOS. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 293–298, New York, NY, USA, 1998.
- [11] P. R. Panda. Memory bank customization and assignment in behavioral synthesis. In *Proceedings of the International Conference on Computer-Aided Design*, pages 477–481, 1999.
- [12] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar. Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 90–95, New York, NY, USA, 2000.