FlexRAM Architecture Design Parameters

Seung-Moon Yoo[‡], **Jose Renau**[†], **Michael Huang**[†], and **Josep Torrellas**[†]
[†]Department of Computer Science
[‡]Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu

Center for Supercomputing Research and Development
Technical Report 1584
October 2000

Abstract

Steady advances in VLSI technology and, in particular, Merged Logic and DRAM (MLD) technology are helping Processor-In-Memory (PIM) chips become more feasible and attractive. FlexRAM is an ongoing PIM project at the University of Illinois. In this report, we summarize some architectural aspects of the system and provide a detailed study of the energy consumption of the chip. In our analysis, we break down the system into different components and we establish energy consumption models accordingly. We verify the numbers produced in our model against other studies of existing chips and find that the results agree.

This report also describes several techniques for dynamic energy management that we include in our PIM system.

1 Introduction

Continuous advances in VLSI technology are fueling the trend toward processor and memory integration in a single chip. Moore's law predicts sustained dramatic increases in the number of transistors that can be integrated on a chip [22]. In addition, recent advances in Merged Logic DRAM (MLD) technology seem able to integrate on-chip logic that cycles as fast as in a logic-only chip, with DRAM that is only about 10% less dense than in a memory-only chip [7, 8]. This means, for example, that a chip in 0.18 μm can integrate 64 Mbytes of DRAM and 64 simple processors cycling at 800MHz.

This integration trend can potentially have an important and lasting impact on computer architecture. Specifically, by placing the memory so close to the processor, we reduce the latency of memory accesses dramatically and can decrease the processor stall time due to memory accesses. In addition, by including multiple processors on the chip, we can allow for fine-grain parallel execution, therefore speeding up parallelizable applications.

Many on-chip high-frequency processors, all of them potentially accessing the memory system concurrently, may consume much energy. In addition, these chips are likely to be used in non-traditional places like the memory of a server, which may not have heavy-duty cooling support. Consequently, it is important to target energy efficiency for these chips.

Different architectures based on what has been called Processor-In-Memory (PIM) architectures, intelligent memory, or Intelligent RAM (IRAM) have been proposed [19]. Roughly speaking, these architectures can be classified based on the role of the PIM chip: main processor (or processors) in the system, special-purpose processor or coprocessor, and intelligent memory system of a workstation or server. The first class of architectures includes Berkeley IRAM [17], Shamrock [13], Raw [24], and SmartMems [14] among others. Examples of special-purpose processors include, among others, Imagine [21] and engines to run vector applications [11], process data at the disk [18] or control ATM switches [2]. Finally, examples of intelligent memory systems include FlexRAM [10], Active Pages [16], and DIVA [5]. This document includes a detailed description of the FlexRAM architecture [10].

The FlexRAM architecture is implemented on MLD chips that include many, relatively simple processors, each of which is associated with a memory bank. Such a design, which is often used in the intelligent memory systems class of architectures above, represents an attractive design point. Indeed, including many processors on a PIM chip is a good way to extract high bandwidth from the DRAM. Past work indicated that the inability to extract high bandwidth from the DRAM was the main reason why adding a single, powerful processor to a DRAM chip delivered disappointing performance [1]. Furthermore, a banked organization is a natural configuration for DRAM chips. In our design, we associate a high-frequency, simple processor with each of the banks.

Such a chip organization has recently become more interesting thanks to dramatic advances in the speed of logic in MLD technology [7, 8]. In each bank, there is a mismatch between the high-speed processor and the slow, high-latency DRAM. For example, in our design, the former is a 800-MHz two-issue processor, while the latter has a 15ns access time. Consequently, we need an efficient on-chip memory hierarchy in every bank. The hierarchy includes a cache, data buffers, row buffers, and a sub-banked DRAM bank. Untuned memory hierarchies may significantly harm the performance of the intelligent memory.

Complicating the problem is the fact that the on-chip memory hierarchy in a bank has to be tuned not only for performance, but also for energy-efficiency and, to a lesser extent, for area-efficiency as well.

This document is organized as follows: Section 2 defines the main objectives in the FlexRAM architecture; Section 3 describes the architecture, defining all the elements present. The technology parameters are described in Section 4. Sections 5, 6, 7 and 8 provide detailed information about the processor core, the caches, the clock distribution, and the DRAM bank respectively. Some energy-management techniques applied in the FlexRAM architecture are explained in Section 9. Just before the summary, the simulation enviroment is explained in Section 10.

2 Design Objectives

The design of our intelligent memory is guided by the following principles.

Extract high DRAM bandwidth. Simply including a conventional wide-issue superscalar in a DRAM chip has delivered disappointing performance [1]. For higher performance, we need to extract more DRAM bandwidth. Consequently, we embed in the DRAM chip many simple processing elements, all of which can access memory concurrently.

Run legacy codes. Since many existing programs cannot be recompiled, PIM chips cannot generally replace the main commodity microprocessor of the workstation. Instead, we propose that these chips take the place of memory chips and appear as plain DRAM to applications that are not enabled for intelligent memory.

Minimize DRAM cost increase. The processing engines embedded in the DRAM must be simple to minimize losses in memory density and therefore cost.

Be general purpose. The PIM chips should not hard-wire a few algorithms and become special-purpose co-processors. Instead, the in-memory processing should be usable in as wide a range of algorithms as possible.

3 FlexRAM Architecture

This section gives a detailed description of the *FlexRAM* architecture. We envision a chip that is connected as a plain DRAM chip and can appear as one to the applications. We call it *FlexRAM*. It contains many simple compute engines called *P.Arrays* that are finely interleaved with DRAM macro-cells. To avoid incorporating extensive interconnection among P.Arrays, we restrict each P.Array to see only a portion of the on-chip memory. To increase the usability of P.Arrays, we also include a low-issue superscalar RISC core on chip. This processor, called *P.Mem*, coordinates the P.Arrays and executes serial tasks. Without the *P.Mem*, these tasks would need to be performed by the commodity microprocessor in the workstation or server (*P.Host*) at a much higher cost. Many *FlexRAM* chips can be connected to the commodity memory interconnect of a workstation or server. A general view of the envisioned architecture is shown in Figure 1. We now describe the components of the *FlexRAM* chip. Their parameters have changed a bit since [10].

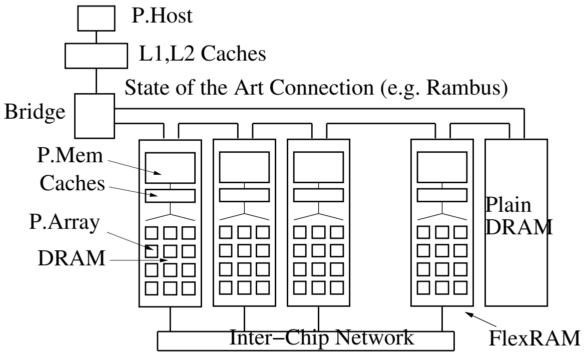

Figure 1: Overall organization of a *FlexRAM*-based memory system.

3.1 P.Array

A FlexRAM chip has 64 P.Arrays. Each P.Array is a simple, 32-bit fixed-point 2-issue RISC engine cycling at 800MHz. Initially, it was envisioned as having a 4-stage pipeline with 16 general-purpose registers, but we are now designing it as a 5-stage 32-register RISC processor. Each P.Array has a 8Kbyte cache, a 1-entry store buffer, and a 1-entry load buffer.

Each P.Array shares a multiplier with 3 other P.Arrays. P.Arrays cycle at 800 MHz and have 28 16-bit instructions. Each P.Array is associated with 1 Mbyte of DRAM and can also access the 1 Mbyte of its two neighbors, forming a logical ring. While neighbor P.Arrays communicate through shared-memory, non-neighbor communication requires the involvement of P.Mem, which can access all memory. Figure 2 shows the P.Array datapath.

Note that the P.Array has no I-cache. The reason is that the *FlexRAM* chip also contains SRAM instruction memory to hold the P.Array code. Each group of 4 P.Arrays shares an 8-Kbyte instruction memory. While sharing instruction memory increases port requirements, it saves overall area. 8 Kbytes can store a sizable program of 16-bit wide instructions. These instruction memories are loaded by the P.Mem.

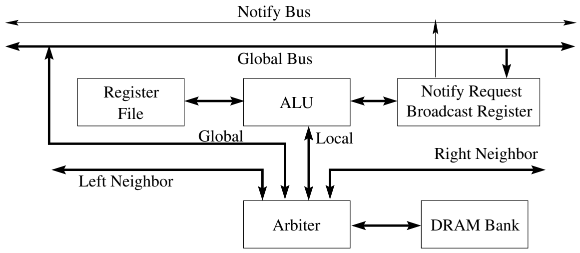

Figure 2: P.Array datapath.

Synchronization occurs with a lock-based primitive similar to MIPS' load-linked and store-conditional. In addition, There is a construct for a global P.Array barrier that uses two primitives: a *notification* from each P.Array to the P.Mem, and a *broadcast* from the P.Mem to all P.Arrays. Notification uses lines that go from each individual P.Array to one bit in P.Mem's Notify Register. Each P.Array can set one bit. The P.Mem can poll the register or be interrupted when certain bit patterns occur. The P.Mem can broadcast a 32-bit word to all P.Arrays. In each P.Array, a register receives the word and a broadcast flag is set. The P.Array can poll the flag to detect when the data has arrived. Broadcasting into memory instead of into a register is not supported because the memories of some P.Arrays could be busy, forcing the broadcast operation to wait.

3.2 P.Mem, Network, and Interface

The P.Mem is a two-issue superscalar with floating-point support like the IBM PowerPC 603 [3] with 16 Kbyte I- and D-caches. We expect that it can cycle at 800 MHz. The processor interface is modified to support the broadcast and notification primitives via memory-mapped locations. Note that this chip is only slightly more powerful than the P.Array. The instructions are 32 bits instead of 16 bits and, as previously stated, the caches are bigger. The P.Mem can support up to 4 pending loads, and 8 pending stores.

P.Mems communicate with each other via an inter-chip network (Figure 1). We minimize the network logic included on chip to make the network topology more flexible. Flexibility is important because different topologies are best with different numbers of chips. Consequently, each chip only includes an In and an Out SRAM queue and simple message packaging logic. Routing is performed by an off-chip router IC. The In and Out ports have 16 data pins each and cycle at 800 MHz. Each of the on-chip queues is 32-bit wide and can hold two 64-byte cache lines.

In a multichip *FlexRAM* memory, all memory is shared and visible to all P.Mems. Requests between chips are transferred through the inter-chip network. When a P.Mem references a location, it caches the memory line. However, there is no hardware to enforce cache coherence between P.Mems or between P.Mems and P.Host. It is up to the programmer to flush the data from the cache before it is used by another processor. A higher-end design could provide coherence support.

Finally, communication between P.Host and P.Mems is implemented by using special features and reserved code words from the Rambus definition. We use Rambus because it allows two-way control signals. We program the Rambus memory controller to support the following protocol. The P.Host starts each P.Mem by writing on a predefined register in the Rambus interface of each chip. The value written is the address of the code to execute. When the P.Host needs to wait for the P.Mems, it reads another predefined register of the Rambus interface in the master P.Mem chip. If the master P.Mem is not finished, a special acknowledgment is returned to the memory controller. The latter buffers the message and keeps retrying the read at regular intervals. This retry operation is transparent to the P.Host. When a retry finds that the master P.Mem is finished, the controller informs the P.Host. This message constitutes the reply to the initial P.Host request.

4 Technology

The *FlexRAM* floorplan design and implementation are based not only on currently available 0.18 μm logic and memory technology, but also on expected future design improvements. For the DRAM macros, which occupy most of the chip area ($\sim$ 70 %), we assume physical parameters such as memory cell size and bit line capacitance not much different from conventional DRAM. We expect that the MLD process will improve due to the advent of high dielectric material for memory cells. We also expect the effective capacitance of the future cell structure to increase to compensate for the small cell dimension. For the instruction and data caches, the cell size is obtained by general scaling-down with transistor level verification. For the P.Array design, each functional blocks is estimated by the general scaling-down theory and designed to achieve the target operating frequency of 800MHz. Table 1 summarizes the parameters used for the FlexRAM implementation.

Parameter	Value
Feature Size	0.18 μm
Metal Layers	6
Operating Voltage	1.8 V
Gate Thickness	7 nm
DRAM Cell Size	0.48 μm^2
D-Cache Cell Size	1.5 μm^2
I-Cache Cell Size	1.5 μm^2

Table 1: Technology parameters.

5 P.Array Core Analysis

Each P.Array core is a 32-bit, 2-issue, in-order processor with a DLX-like pipeline. It supports a simplified version of the MIPS ISA with only 28 16-bit instructions [10]. Functional blocks are placed to minimize data communication. As shown in Figure 3, The ALU is close to the register file and the D-cache. Since the I-memory is shared by 4 adjacent P.Arrays, it is located close by. We use data from [26] and, by applying general scaling theory and considering technology trends, we estimate the average energy consumed in the register file, branch unit, ALU, and the other modules of the processor. Table 2 shows the P.Array functional blocks estimated energy. We assume perfect clock gating inside the processor core.

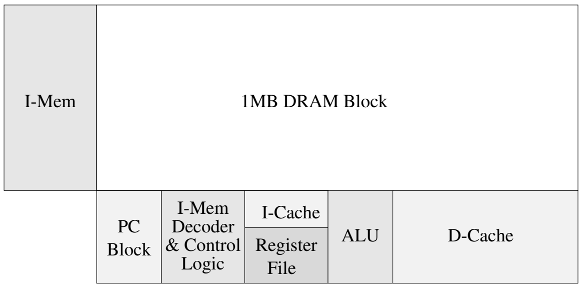

Figure 3: P.Array floorplan.

We estimate the energy consumed by each type of instruction by adding up the energy of all the modules used by that particular instruction type. For example, a branch instruction with a read from the register file consumes 34.8pJ $(2 + 14.9 + 13.9 + 4)$. If the instruction adds two registers, the total is 56.1pJ $(2 + 13.9 + 13.9 + 7.4 + 14.9 + 4)$. In the same way, if the instruction does a multiplication, the total energy is 251.2pJ $(2 + 13.9 + 13.9 + 7.4 + 210 + 4)$.

Note that these energy values include neither the data cache energy nor the instruction cache energy. The reason is that caches are modeled as external structures. Section 6 computes the energy charged for cache accesses.

Functional Block	Value (pJ)
Program Counter Block	2.0
Branch Unit	14.9
Register File (Rd)	13.9
Register File (Wr)	7.4
Multiplication ALU	210.0
ALU	14.9
Control Block	4.0

Table 2: Estimated energy consumption of the different P.Array functional blocks.

5.1 P.Array Validation

To validate the P.Array energy consumption, we compare the estimated consumption to a StrongARM processor [15]. Their simulations show the consumption of the different functional blocks in the StrongARM processor. For validating the numbers, we computed the energy consumed by four basic elements: clock, I-cache, D-cache and processor core. Table 3 shows the percentage of total energy consumed by each element in our P.Arrays and in the StrongARM.

Block	StrongARM	P.Array
I-Cache	36%	34%
D-Cache	24%	23%
Core	29%	35%
Clock	11%	8%

Table 3: P.Array and StrongARM energy distribution.

We can see that the distributions are very similar. The main difference is that our core spends more energy than the StrongARM. The main reason is that the P.Array is slightly more complicated. It is a two issue processor.

6 Cache Analysis

This section provides detailed information about the caches used in each P.Array. As previously indicated, a *FlexRAM* chip has 64 P.Arrays, 64 8-Kbyte data caches (one per P.Array), and 16 8-Kbyte instruction memories (one shared by 4 P.Arrays). This means that each P.Array sees a private 8-Kbyte data cache, and a shared 8-Kbyte instruction memory.

The numbers have been obtained following the Kamble and Ghose model [9]. The energy consumption has been validated with CACTI v2 [25].

6.1 Data Cache

The design of the data cache for the P.Array is very important. A sizable cache not only improves the performance but also the energy efficiency. The cache has been optimized to provide an access in one cycle. Since the P.Array cycles at 800 MHz, the required access time is 1.25 ns.

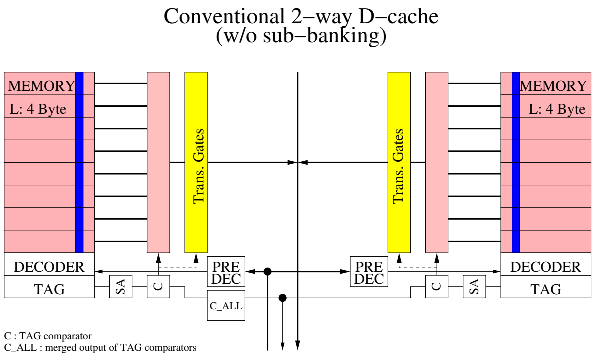

Figure 4: Data cache diagram.

After analyzing different cache sizes, namely 256 bytes, 1 Kbyte, 8 Kbytes, and 16 Kbytes, we demonstrated that an 8-Kbyte cache is a sweet point for energy-delay product [20]. We also noticed that, for the set of benchmarks used, a system with 8-Kbyte caches is approximately 50% faster than one with 256-byte caches.

The cache diagram is shown in Figure 4. An access to the cache has the following steps:

1. The address required is provided by the P.Array (ALU core in Figure 4). This activates the peripheral circuits (predecoder and decoder).
2. The *word lines* in the SRAM memory bank are enabled.
3. The data is driven by the *bit lines*. This includes the sense amplifier and the transmission gate.
4. The *tag* in the data cache is compared against the address provided by the P.Array.

The main parameters for the data cache in each P.Array are shown in Table 4.

Parameter	Value
Associativity	2
Cache line size (bits)	256
Number of sets	128
Offset bits	5
Index bits	7
Memory address bits	20
Valid & dirty bits	2
Tag bits	8

Table 4: Data cache parameters.

Cache Activity	Charge (pC)	Energy (pJ)
Word Line Activation	1.45	2.61
Bit Line Activation	33.50	60.30
Sense Amplifier	66.89	120.40
Tag Comparison	2.12	3.81
Peripheral Circuits	19.80	35.64
Total Cache Hit	**123.76**	**222.76**
Total Cache Miss	**21.92**	**39.45**

Table 5: 8-Kbyte data cache charge and energy.

A cache hit requires all the previous steps, while a cache miss does not require to activate the sense amplifiers and transmission gates. To calculate the energy required, we have to compute the total charge for each one of the previous steps. Table 5 shows the charge and the energy values. To obtain the energy values we only need to multiply the charge by V_{dd}, which is 1.8 V. This is because $Energy = C * V^2 = C * V * V = Q * V$.

6.1.1 CACTI Validation

In future FlexRAM architecture releases, we plan to validate our results with SPICE simulations. This time we validate our results with the widely used CACTI tool [25].

We model the cache with the parameters specified in Table 4. Since we assume an aggressive sense amplifier, we modify the scaling factor in CACTI for the sense amplifier.

The access time from CACTI is 0.97 ns for the cache. We have a 1.25 ns cycle time. Since we reserve 0.25 ns of the time for transferring signals from the P.Array to the cache and back, we have 1 ns for the data cache access.

CACTI provides a detailed breakdown of the energy consumed in the cache. In Table 6, we see a 10% difference between CACTI and our model. The main source of difference is the tag comparison. Specifically, the difference comes from activating the sense amplifiers. Since CACTI has not been validated with 0.18μm technology, we believe that the match is good enough.

Cache Activity	FlexRAM (pJ)	CACTI (pJ)
Word Line Activation	2.6	1.4
Bit Line Activation	60.3	63.3
Sense Amplifier	120.4	131.0
Tag Comparison	3.8	13.9
Peripheral Circuits	35.6	35.8
Total Cache Hit	**222.7**	**245.4**

Table 6: Comparing the consumptions predicted by our model and by CACTI.

6.2 Instruction Memory

In the FlexRAM architecture, the code for the applications resides in a different address space. For a P.Array, the instructions reside in a 8-Kbyte SRAM shared by 4 processors. Sharing the SRAM reduces the area requirements but also increases the energy consumption because the data has to be transported to the P.Array, which is not so close. In the same way as the data cache, the instruction memory has been optimized for being accessed in one cycle. Since the P.Array cycles at 800 MHz, the access time is 1.25 ns.

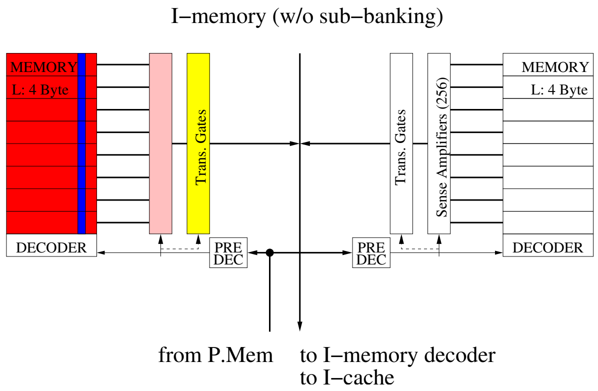

Figure 5: Instruction memory diagram.

Figure 5 is a diagram for the instruction memory. Note that it is very similar to the data cache. The main differences between the D-cache and the I-memory are the number of bits activated and the absence of tag in the I-memory. Table 7 shows the main parameters for the instruction memory. Each time that there is an access, only two instructions (32 bits) are transferred to the P.Array.

Since the P.Mem is in charge of copying the program that the P.Array is going to execute, it needs to have access to the instruction memory. An instruction memory read from the P.Array proceeds as follows:

1. The address required is provided by the P.Array (I-cache decoder in Figure 5). This activates the peripheral circuits (predecoder and decoder).
2. The word lines in the SRAM memory bank are enabled.
3. The data is driven by the bit lines. This includes the sense amplifier and the transmission gate.
4. The data is transferred to the P.Array.

Parameter	Value
Size	8 Kbytes
Number of sub-banks	2
Bank line size	256 bits
Activated line size	32 bits
Distance to the P.Array	2,120 um

Table 7: Instruction memory parameters.

Note that, in the previous steps, there is no neeed to perform a tag check as in the data cache. To calculate the total energy spent for each read, we estimated the total charge. Table 8 shows the charge and energy values for the instruction memory.

Memory Activity	Charge (pC)	Energy (pJ)
Word Line Activation	0.78	1.40
Bit Line Activation	19.61	35.30
Sense Amplifier	16.02	28.83
Peripheral Circuits	16.72	30.10
Data Transfer	4.26	7.66
Total	**57.39**	**103.29**

Table 8: I-memory charge and energy values.

An instruction memory read requires 103.29 pJ. Since the P.Array is a 2-issue processor, an I-memory read provides 32 bits (two instructions). Then for each instruction executed we approximate the energy consumed by $\frac{103.29}{2} = 51.6pJ$.

6.2.1 CACTI Validation

The results obtained have also been generated with CACTI v2. We model the cache with the parameters specified in Table 7. As previously stated, since we assume an aggressive sense amplifier, we modify the scaling factor in CACTI for the sense amplifier. Since CACTI models caches, and the I-memory is not a real cache, we suppressed the tag influence in the energy and delay computation.

The data bank access time from CACTI is 0.79 ns. In our I-memory, we have a 1.25 ns cycle time, but the I-memory is quite far from the ALU (2120 μm). We reserved 0.40 ns for transferring the signals from the core to the I-memory and back. We have, therefore, enough time.

Cache Activity	FlexRAM (pJ)	CACTI (pJ)
Word Line Activation	1.4	1.4
Bit Line Activation	35.3	63.3
Sense Amplifier	28.8	28.6
Peripheral Circuits	30.1	31.9
Data Transfer	7.7	-
Total	**103.3**	**125.2**

Table 9: Comparing the I-memory consumptions predicted by our model and by CACTI.

CACTI provides a detailed breakdown of the energy consumption. The comparison between CACTI and our model is shown in the Table 9. The FlexRAM cache consumes about 18% less energy than the CACTI model predicts. The main source for this difference is the bit line activation energy.

7 Clock

The clock network in FlexRAM basically assumes a H-Tree structure as shown in Figure 6. This configuration minimizes clock skew in the chip.

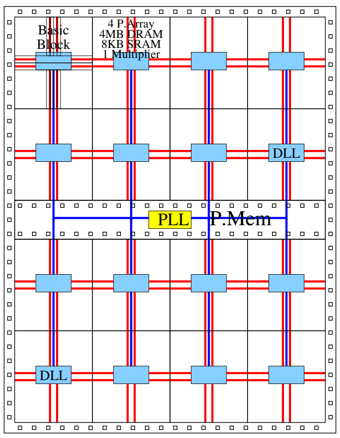

Figure 6: Overall clocking network of a FlexRAM chip.

The PLL (Phase-Lock-Loop) is placed in the center of the chip to generate the clock for the P.Mem and the P.Arrays. Clock signals with different frequency can be generated to cope with various external frequencies. A DLL (Delay-Lock-Loop) is assigned to each block of 4 P.Arrays. This replication of DLLs is done to minimize capacitive loading and

Circuit	Charge (pC)	Energy (pJ)
PLL to Buffer	13.3	24.0
Buffer to P.Mem	14.6	26.2
16 DLL	460.3	828.5
Buffer to P.Array	43.7	78.6
Total	**531.9**	**957.3**

Table 10: Clock energy consumption.

clock jitters from the PLL. Each DLL is assumed to have a simplified structure to minimize area and power cost. A DLL can also be turned-off selectively to reduce power consumption. To estimate the power consumption in the clock distribution, the PLL and all the DLLs are assumed to be active. The energy also considers drivers from each clock generator, and line load capacitance from the PLL to the DLLs. Signal and gate capacitances in the basic blocks are also estimated. Table 10 shows the energy required for the clock distribution across the chip.

8 DRAM Bank Analysis

The DRAM macro block design that we use is not very different from the one used in a traditional DRAM chip. The following design considerations therefore, are not limited to embedded systems and may also be applied to general DRAM design.

Traditionally, sizable DRAM memories have been organized in banks. In our design, each memory bank has 1 Mbyte. Figure 7 shows an overview of the data memory hierarchy associated with a single DRAM bank. In the figure, we use a traditional DRAM organization. The main blocks are the DRAM sub-banks, the row buffers, and the data buffer. This section describes the parameters used for each one of the main blocks.

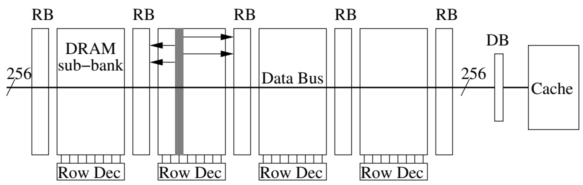

Figure 7: Example of the data memory hierarchy associated with a DRAM bank. RB, DB, and Row Dec stand for row buffer, data buffer, and row decoder, respectively.

8.1 DRAM Sub-Bank

Sub-banked DRAM offers many advantages. For example, in some organizations, each sub-bank can potentially service an independent memory request. This increases DRAM's bandwidth. In addition, smaller sub-banks also means faster access. Indeed, although accessing multiple banks may require extra pre-decoding, the savings in bit sensing outweigh this time overhead. As a result, accessing a DRAM divided into multiple small sub-banks is usually faster than accessing an undivided big bank. Finally, using smaller sub-banks often means sense amplifying fewer bit lines and with less capacitance, which implies less energy per access. For 0.18μm technology, we decided to have 1-Mbyte memory banks with 4 sub-banks per bank. Section 8.4 analyzes different organizations.

8.2 Row Buffer

A row buffer is essentially a sense amplifier array augmented with a simple row address buffer. The row buffer behaves like a special write through cache. The address of a new access is compared with the address stored in the buffer. If it is a match, the word line enabling is not activated and the data is driven directly from the sense amplifier.

A write always goes through to the DRAM cell array. An enhanced design could make it possible to have a write-back policy in the row buffer. However, it complicates the design significantly. In addition, according to our simulation, supporting a write-back policy does not increase performance noticeably.

Generally, to reduce DRAM cell size, the capacitor is fairly limited in size and hence capacitance. The sense amplifier is relatively large and can not be fitted in the bit-line pair pitch. As shown in Figure 7, the common approach is to put sense amplifiers for odd and even lines on different sides of the sub-bank. As a result, very often, the DRAM is organized as sub-banks with sense amplifiers physically lying between two sub-banks. Though not necessary, the sense amplifiers are often shared between the two sub-banks. Dedicated or even multiple sense amplifiers for a single DRAM cell array are possible. Extra sense amplifiers can serve as caches.

Our simulation results show that a dedicated SRAM cache outperforms multiple sense amplifiers. Given that these sense amplifiers are quite significant in size, it can be argued that the most cost-effective scheme is to use one row buffer shared between consecutive sub-banks, and then add a dedicated SRAM cache for the whole system (Figure 7).

8.3 Data Buffer

Although sense amplifiers drive the signal strong enough to read from the cell array, we still need larger transistors for driving long lines. These extra transistors are referred to as *Data Buffers*. Usually, data buffers tend to be narrower in width than row buffers. Data buffers consist of very wide metal lines routed on a different chip layer. These lines are not only wide, but also long. For a sub-banked DRAM, multiple sub-banks share the same data buffer. Multiplexors take care of sharing a single data buffer line among several bit lines. We choose a data buffer of 256 bits to match the cache line size.

8.4 Organization

Having several sub-banks improves bandwidth, latency, and energy. These improvements come at the price of extra area. Having shorter bit-lines means that fewer cells are sharing the same sense amplifier and other peripheral logic, thus reducing overall DRAM density.

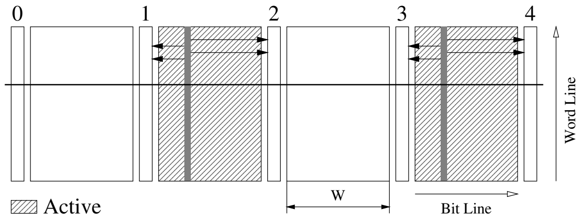

Figure 8: Traditional sub-banked DRAM bank (Trad).

There are different ways of organizing a DRAM bank. In the most basic approach, which we call the *traditional* (Trad) organization, only one access can be processed at a time. When the memory is accessed, every other memory sub-bank is activated. We cannot activate all the sub-banks because two consecutive sub-banks share a row buffer. Figure 8 shows a 4-sub-bank *traditional* organization. This organization is largely equivalent to a single large sub-bank. The division is more for timing, routing and placement of circuitry.

Based on this Trad organization we have three optimizations: Segmentation, Interleaving, and Pipelining.

8.4.1 Segmentation

As opposed to *traditional*, *segmentation* (S) only activates one sub-bank at a time. This requires a little extra control lines. Figure 9 shows a 4 sub-bank segmented organization. Segmentation could improve the performance because the contents of the row buffers are more decoupled. In the example, row buffers 0-1 and 2-3 are loaded with data from different addresses. In many benchmarks, this increase in the number of localities increases the row buffer hit rate. Although only seen in one benchmark, it is also possible that segmentation decreases the row buffer hit rate due to a reduction of spatial locality.

Energy consumption is the main segmentation benefit. The energy savings result from the fact that an access activates fewer sub-banks. Bit line sensing constitutes about 91% of the total energy in this particular configuration. Accessing one sub-bank instead of two reduces the energy by 41%. Note that this is only the energy for one access to the DRAM cell array. The total energy for many memory system accesses also depends on other factors like the spatial locality of memory references.

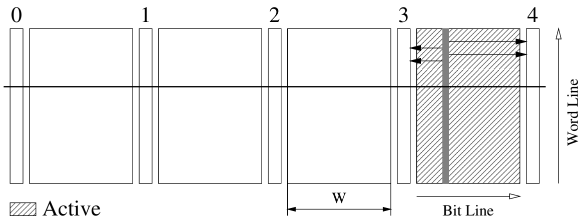

Figure 9: Segmented sub-banked DRAM bank (S).

8.4.2 Interleaving

Interleaving (I) slices vertically each sub-bank and assigns a data bus to each of the resulting slices. The area is increased because it is necessary to duplicate the column control and row buffers. Figure 10 shows a 2-way interleaved organization. Now, each new sub-bank has half the width. The reduced bit line length and capacitance results in faster access and less energy for each access. The quantitative results of the improvement depend on the actual dimension of the sub-banks. With this organization, the data can be extracted in parallel when requests go to different sub-banks connected to different data busses. The adoption of a single address bus (not shown in the figure) causes some serialization in the address transfer. The detailed timing is shown in Section 8.5.

8.4.3 Pipelining

Since bit-line sensing is the dominant factor in DRAM access latency, overlapping multiple bit-line sensing in different sub-banks lessens contention. This is called pipelining. Note that sub-banks connected to different data busses are intrinsically independent. Therefore, pipelining does not apply to them. It applies to sub-banks that share the same data bus. While sense amplifying one sub-bank, another independent access

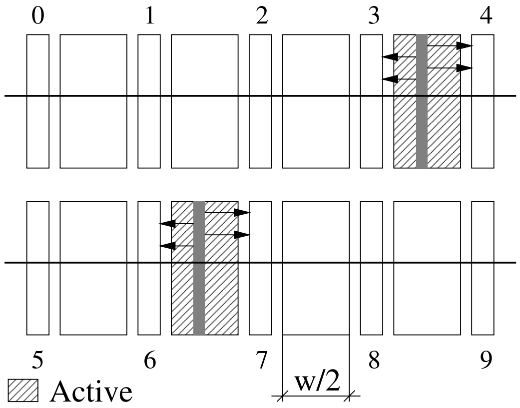

Figure 10: Interleaved and segmented sub-banked DRAM bank (IS).

can be sent to a different sub-bank in a streamlined fashion, without waiting for the first sub-bank to finish.

Finally, we can improve the performance by further slicing each sub-bank vertically and creating more sub-banks. Figure 11 shows a 2-way interleaved system with 8 sub-banks per data bus. More row buffers offer more localities, decreasing the contention. Moreover, with smaller bit line capacitance, the array can be accessed faster and with less energy. The cost of this is larger area. We represent an i-way interleaved system with j sub-banks per data bus as (i, j). A compact nomenclature is used in the remainder of the paper, where Trad(i,j), S(i,j), SP(i,j), IS(i,j), ISP(i,j) corresponds to Traditional, Segmented, Segmented Pipelined, Interleaved Segmented, and Interleaved Segmented Pipelined, respectively.

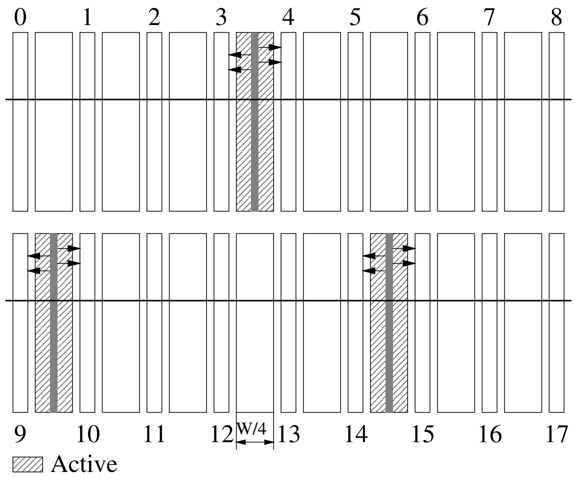

Figure 11: Interleaved, segmented, and pipelined sub-banked DRAM bank (ISP).

8.5 DRAM Timing

In an embedded environment, we can exploit the fact that both row address and column address can be sent very fast without the need to drive long lines. In our system architecture, we have a hierarchy of processors, each sitting at a different distance to the DRAM cell array. This affects the timing of the access to the DRAM. As an example, in Figure 12, to buffer an address issued by a P.Array to its DRAM bank, one cycle is enough because of the proximity of processor and memory. However, when the P.Mem accesses the DRAM, the address may be driven across the chip to a module sitting on the edge. To ensure correctness, we have to wait for the address lines to become stable before starting the decode. As a result, one extra cycle is required.

8.5.1 Access Timing Breakdown

Figure 12 shows the breakdown of the timing for accessing different levels of the DRAM hierarchy as shown in Figure 7. It shows the cases of a miss in the row buffer, a hit in the row buffer, and a hit in the data buffer.

In case of a row buffer miss, the data is transferred from the DRAM sub-bank to the row buffer, before being transmitted to the data buffer. In this process, row addresses are decoded and a new word line is activated. Then, the small charge in the memory cell is shared with the bit line. The effect on the bit lines is amplified by the sense amplifier to the full level. In case of an access from the P.Mem, we add one more cycle to transfer the address. An additional cycle is also added for data transferring from the DRAM to P.Mem due to their long lines.

In case of a row buffer hit, data can be fetched by changing column addresses. Most of the time is consumed by data transferring from the row buffer to data buffer. For the P.Mem, we also add one more cycle to transfer the address and to transfer the data. In case of data buffer hit, we only need to transfer the data from the data buffer to the cache or the ALU. There is one more cycle for address comparison to communicate with the P.Mem.

Although the DRAM hierarchy looks like a traditional multi-level cache hierarchy, it is different. To avoid excessive circuit overhead, control logic is minimized. For example, although a multi-port DRAM sub-bank is possible, the extra area and complexity is difficult to justify. From our simulation results, a cache filters most of the requests and, as a result, the DRAM hierarchy does not need to be very sophisticated.

8.5.2 Precharge

The sense amplifiers need the bit lines to be precharged to $V_{dd}/2$ before enabling the word lines. This precharge adds up to the cell array access latency. To hide this latency, we start precharging the bit lines immediately after a memory access. To enable early precharge we apply *Pulsed RAS*. To avoid disturbing the sense amplifier, we use a pass gate to separate the bit line and the differential amplifier. Then, the data latched can be used to service a read to the same row. By adopting early precharge, we hide the precharge time well. On average, less than 1% of cell accesses are delayed by precharging.

To stabilize the signal for further amplification, we assume that precharging the bit lines takes about 10 ns, which translates into 8 cycles under a 800 MHz clock.

8.6 Energy

This section provides the DRAM energy values used in the FlexRAM architecture. The different energies are calculated based on the DRAM cell size unit line capacitance. The DRAM block size is estimated by multiplying the number of cells in the block and the unit cell size. The length of the control lines is estimated by considering DRAM block sizes. Bit line capacitance, which takes a major portion of the power consumption in the row buffer miss, is estimated by multiplying the number of cells per bit line and the unit bit line capacitance.

Table 11 shows the energy consumption for different DRAM organizations. For each configuration, an access consumes different energy depending of the hierarchy level accessed (row buffer or DRAM sub-bank) and the kind of access (read or write). The row buffer write has two configurations, depending on whether it needs to be precharged or it is still active.

A row buffer hit includes the energy consumption related to column addressing, pre-decoding, decoding, word line (main and local) enabling, and data buffer. In case of a row buffer miss, the DRAM core control signals like the bit line isolation, equalization, and bit line sensing are also considered. Compared with the row buffer hit, energy consumption is much bigger.

Energy consumption in the bit line sensing operation is calculated assuming that the bit line swing is half of the DRAM array voltage. To calculate the energy consumption for column activation, we study the column addressing, decoding, column selection and data buffer enabling operations. Energy consumption due to the data buffer is estimated based on a latch-type fast current mode sense amplifier. The consumption is smaller in the presence of a precharge pulse. The number of data lines is assumed to be 256 to fill the cache

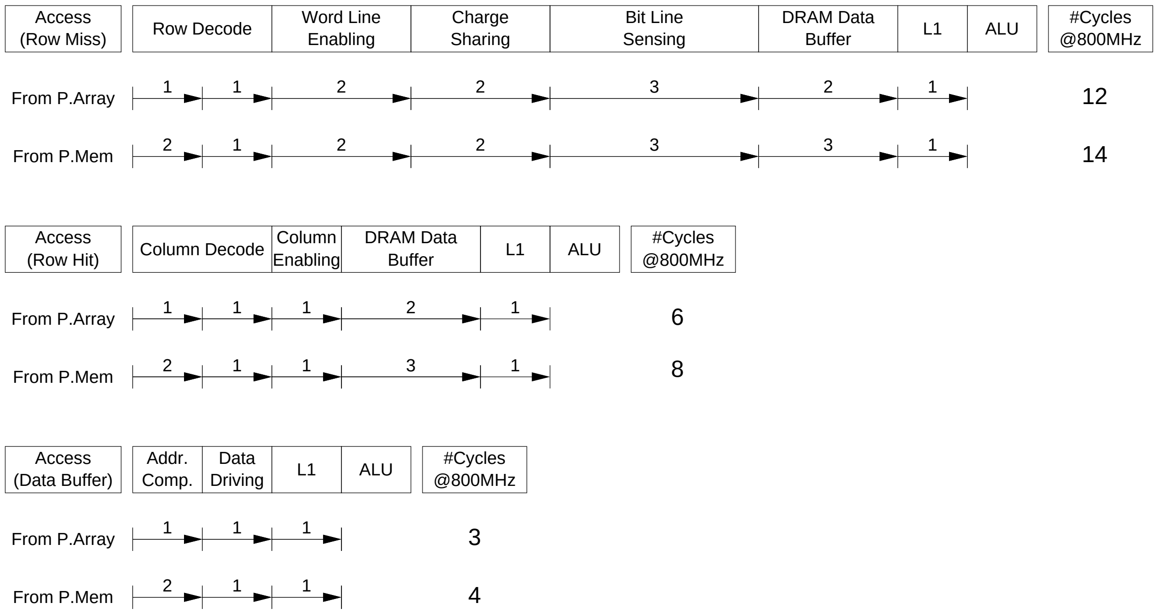

Figure 12: DRAM access timing breakdown.

block in one cycle. We assume a hierarchical data line structure where data lines are running in parallel with the bit lines. The data lines are located at the end in the DRAM block, close to the cache and ALU.

8.7 Miscellaneous

An important issue is the cell refresh interval. With so many processors in the chip, the chip temperature may be high. Since DRAM cells are so vulnerable to heat due to increased leakage current, we need to design the chip carefully. We assume that future cell technology will have more capacitance due to advanced dielectric material and cell structures. In that case, we can manage the refresh interval effectively with the help of a cooling system. The expected performance degradation due to the refresh cycle can be mitigated with a more intelligent DRAM refresh, like a hidden refresh in a conventional DRAM.

Power noise can be another issue. We assume separate power pins (actually power balls, since we are using BGA) for the DRAM and for the processor core. We also assume a meshed power structure to minimize power noise.

To increase yield, redundant cells are used in DRAM and cache. A full scan-based Built-In-Self-Test (BIST) with Linear Feedback Shift Register (LFSR) can be used to improve testability while minimizing hardware overhead.

9 Energy-Management Techniques

Finally, we discuss several dynamic energy-management techniques that we use in FlexRAM [6].

9.1 Cache Sub-Banking

Power consumption in a cache is a strong function of the number of activated cache lines, memory cells in the line, and bit-line sense amplifiers. Cache sub-banking is the activation of only the fraction of the line selected instead of the whole cache line [4, 23].

Cache sub-banking is a useful way to reduce power consumption since the number of active sense amplifiers can be reduced (usually to one-eighth).

To implement sub-banking, it is necessary to add several transmission gates in the word lines. Since we do not want to increase the area much, we are forced to use small transmission gates. This extra logic has a negligible affect on the access time when all the transmission gates are enabled. This is no longer true when we only activate a subsection of the word line.

As the cell size and the minimum feature size are getting smaller, the delay mentioned increases due to the reduced cell pitch and the higher word line resistance (usually polypide or silicide poly gate). Therefore, the access time through a hierarchical word line structure can be longer than through a conventional one with metal strapped word line structure. It depends strongly on memory architecture and technology

Operation	Trad(1,4) (pJ)	S(1,4) (pJ)	IS(2,4) (pJ)	IS(2,8) (pJ)	SP(1,4) (pJ)	ISP(2,4) (pJ)	ISP(2,8) (pJ)
Row Buffer Read	431	431	469	480	500	538	549
Row Buffer Write (Active)	541	541	308	249	541	308	249
Row Buffer Write (Precharge)	5253	2739	1516	965	2739	1516	965
Sub-Bank Read	6962	3702	2250	1519	3702	2250	1519
Sub-Bank Write	6345	3285	1813	1136	3285	1813	1136

Table 11: DRAM bank energy consumption.

used. We may need to assign two cycles instead of one cycle since its timing margin may be a bit tight to meet the given operating frequency specification.

Cache Activity	Charge (pC)	Energy (pJ)
Word Line Activation	1.45	2.61
Bit Line Activation	5.10	9.20
Sense Amplifier	9.90	17.80
Tag Comparison	2.12	3.81
Peripheral Circuits	19.80	35.64
Total Cache Hit	**38.37**	**69.06**
Total Cache Miss	**23.37**	**42.06**

Table 12: 8-Kbyte sub-banked cache charge and energy.

Applying cache sub-banking dynamically is done as follows. When the operating frequency is high and only performance is emphasized, the whole cache block is activated. However, when we want to save energy, at the cost of some slowdown, sub-banking is used.

Table 12 shows the energy for accessing the cache when sub-banking is activated. All the numbers remain similar to the default cache operation (Table 5). The only differences are the bit line activation energy and the sense amplifier energy. The new energies are one eight of the original ones plus some extra energy for the additional logic. To compute the energy for a cache miss, we add the energy for tag comparison and peripheral circuits (as before) plus the energy for word line activation.

9.2 Dynamic Filter Cache

Adding a small cache between the L1 cache and the processor to reduce energy consumption was proposed in [12]. Though much more energy-efficient to access, these filter caches often have a much lower hit rate than typical L1 caches. Therefore, they cause performance losses. Here, we propose a variation that minimizes performance losses. We add a small filter cache (I-cache in Figure 13) between the processor and the instruction memory (I-memory), but we propose two working modes. In the normal mode, the processor only accesses I-memory. Implemented in large-transistor SRAM and big enough to store all the program code, I-memory has an access time of one cycle and consumes a lot of energy for each access. In the low-power mode, I-cache is activated, and the I-memory is accessed only when there is a miss in I-cache. Being much smaller in size, I-cache consumes much less energy for each access but only captures a portion of the code and, therefore, suffers more misses.

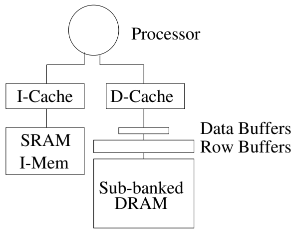

Figure 13: FlexRAM memory hierarchy.

Parameter	Value
Associativity	1
Cache line size (bits)	64

Table 13: I-cache parameters.

The I-cache is very similar to the caches described in Section 6. The main difference is the size: the I-cache is much smaller. Table 13 shows some I-cache parameters.

As previously explained, we assume a better kind of sense amplifier than the one provided by CACTI. After CACTI has been modified to support the more advanced sense amplifiers, the total energy consumption for one access to the I-cache is 56.2 pJ. Each access to the I-cache provides one line (four instructions). The line is stored in a 1-entry P.Array internal buffer so that the instructions can also be used in the following cycles. Since some of the instructions in the line may not be used at all, the energy consumed in I-cache fetch per single instruction tends to be higher than simply one quarter of 56.2 pJ. Based on the line usage observed in our applications, the average energy consumed in I-cache fetch per single instruction is 15.4 pJ.

9.3 Voltage Scaling

Voltage scaling is one of the best known methods for reducing power consumption. Power is proportional to the square of the operating voltage. The usual way to do that in the chip is to use a voltage limiter, called an IVC (internal voltage converter), to reduce the operating voltage. In this scheme, the supply voltage is determined and controlled by an internal reference voltage. It takes about 10 to 20 μs to change the voltage.

Although OS-based voltage control is realistic, we assume a zero time overhead for changing the voltage and frequency. This is clearly impossible to implement, but our voltage scaling technique is an ideal technique that we use for comparison with other techniques.

We assume that FlexRAM has only one reduced voltage mode. When voltage scaling is activated, instead of operating at 800 MHz and 1.8 V, FlexRAM operates at 640 MHz and 1.44 V.

10 Simulation Environment

In previous chapters, we have described the FlexRAM architecture. The analysis included structure, energy, and latency numbers. To calculate the performance and the total energy consumption, we have built an execution-driven simulator for the FlexRAM architecture.

The simulation environment consists in three main blocks, *mint*, *smt*, and *flex*. *mint* is a execution-driven simulator that models a MIPS processor. To model different kinds of processors simultaneously, the I-ACOMA group developed *smt*. *smt* allows us to simulate different kinds of processors (issue width, in- and out-of-order, functional units, frequency...). This module also calculates energy for the processor core. It charges different energy depending on the instruction to execute.

The *flex* module simulates all the memory backend. This includes the I-memory, I-cache, D-cache, row buffers, buses, and memory banks. Each time that there is a read or a write, the energy consumption is also computed.

The resulting environment models with great detail the whole FlexRAM architecture, providing detailed information about energy and performance.

11 Summary

In this report, we reviewed the FlexRAM architecture [10] and its recent extensions. We showed the structure of building blocks, the chip, and the system. We also showed the breakdown of the estimated energy consumption of the chip and validated our model against existing studies of various types. Finally, we went over some dynamic techinques for energy managment and the simulation environment of the system.

The architecture will be constantly extended and updated on our website at http://iacoma.cs.uiuc.edu/flexram.

References

[1] N. Bowman, N. Cardwell, C. Kozyrakis, C. Romer, and H. Wang. Evaluation of Existing Architectures in IRAM Systems. In *First Workshop on Mixing Logic and DRAM: Chips that Compute and Remember*, June 1997.

[2] A. Brown, D. Chian, N. Mehta, Y. Papaefstathiou, J. Simer, T. Blackwell, M. Smith, and W. Yang. Using MML to Simulate Dual-Ported SRAMs: Parallel Routing Lookups in an ATM Switch Controller. In *First Workshop on Mixing Logic and DRAM: Chips that Compute and Remember*, June 1997.

[3] G. Gerosa, S. Gary, C. Dietz, et al. A 2.2 W, 80 MHz Superscalar RISC Microprocessor. In *IEEE Journal on Solid-State Circuits*, December 1994.

[4] K. Ghose and M. Kamble. Reducing Power in Superscalar Processor Caches Using Subbanking, Multiple Line Buffers and Bit-Line Segmentation. In *International Symposium on Low Power Electronics and Design*, pages 70–75, August 1999.

[5] M. Hall et al. Mapping Irregular Aplications to DIVA, a PIM-based Data-Intensive Architecture. In *Supercomputing*, November 1999.

[6] M. Huang, J. Renau, S.-M. Yoo, and J. Torrellas. A Framework for Dynamic Energy Efficiency and Temperature Management. In *Proceedings of the 33th Annual International Symposium on Microarchitecture*, December 1999.

[7] IBM Microelectronics. Blue Logic SA-27E ASIC. http://www.chips.ibm.com/news/1999/sa27e, February 1999.

[8] S. Iyer and H. Kalter. Embedded DRAM Technology: Opportunities and Challenges. *IEEE Spectrum*, April 1999.

[9] M. Kamble and K. Ghose. Analytical Energy Dissipation Models for Low Power Caches. In *International Symposium on Low Power Electronics and Design*, pages 143–148, August 1997.

[10] Y. Kang, W. Huang, S. Yoo, D. Keen, Z. Ge, V. Lam, P. Pattnaik, and J. Torrellas. FlexRAM: Toward an Advanced Intelligent Memory System. In *International Conference on Computer Design*, pages 192–201, October 1999.

[11] S. Kaxiras, R. Sugumar, and J. Schwarzmeier. Distributed Vector Architecture: Beyond a Single Vector-IRAM. In *First Workshop on Mixing Logic and DRAM: Chips that Compute and Remember*, June 1997.

[12] J. Kin, M. Gupta, and W. Mangione-Smith. The Filter Cache: An Energy Efficient Memory Structure. *International Symposium on Microarchitecture*, pages 184–193, December 1997.

[13] P. Kogge, S. Bass, J. Brockman, D. Chen, and E. Sha. Pursuing a Petaflop: Point Designs for 100 TF Computers Using PIM Technologies. In *Frontiers of Massively Parallel Computation Symposium*, 1996.

[14] K. Mai et al. Smart Memories: A Modular Reconfigurable Architecture. In *International Symposium on Computer Architecture*, 2000.

[15] J. Montanaro et al. A 160-MHz, 32-b, 0.5-W CMOS RISC Microprocessor. *IEEE Journal Solid State Circuits*, 31(11):1703–1714, November 1996.

[16] M. Oskin, F. Chong, and T. Sherwood. Active Pages: A Computation Model for Intelligent Memory. In *International Symposium on Computer Architecture*, pages 192–203, June 1998.

[17] D. Patterson et al. A Case for Intelligent DRAM. *IEEE Micro*, pages 33–44, March/April 1997.

[18] D. Patterson et al. ISTORE: Intelligent Store. http://iram.cs.berkeley.edu/ istore/index.html, 1998.

[19] D. Patterson and M. Smith. Workshop on Mixing Logic and DRAM: Chips that Compute and Remember. 1997.

[20] J. Renau. Memory Hierarchies in Intelligent Memories: Energy/Performance Design. Master Thesis, Institution, January 2000.

[21] S. Rixner et al. A Bandwidth-Efficient Architecture for Media Processing. In *International Symposium on Microarchitecture*, November 1998.

[22] Semiconductor Industry Association. *The National Technology Roadmap for Semiconductors*. SIA, 1998. http://notes.sematech.org/ntrs/PublNTRS.nsf.

[23] C-L. Su and A. Despain. Cache Design Trade-offs for Power and Performance Optimization: A Case Study. In *International Symposium on Low Power Electronics and Design*, pages 63–68, April 1995.

[24] E. Waingold et al. Baring It All to Software: Raw Machines. *IEEE Computer*, pages 86–93, September 1997.

[25] S. Wilton and N. Jouppi. CACTI: An Enhanced Cache Access and Cycle Time Model. *IEEE Journal on Solid-State Circuits*, 31(5):677–688, May 1996.

[26] N. Yeung. et al. The Design of a 55SPECint92 RISC Processor under 2W. *ISSCC Digest of Technical Papers*, pages 206–207, February 1994.