Near-Memory Caching for Improved Energy Consumption

Nevine AbouGhazaleh, *Student Member*, *IEEE*, Bruce R. Childers, *Member*, *IEEE*, Daniel Mossé, *Member*, *IEEE*, and Rami G. Melhem, *Fellow*, *IEEE*

Abstract—The main memory has become one of the largest contributors to overall energy consumption and offers many opportunities for power/energy reduction. In this paper, we propose a power-aware cached-dynamic-RAM (PA-CDRAM) organization that integrates a moderately sized cache directly into a memory chip. We use this near-memory cache to turn a memory bank off immediately after it is accessed to reduce power consumption. We modify the operation and structure of CDRAM with the goal of reducing energy consumption while retaining the performance advantage for which CDRAM was originally proposed. In this paper, we describe our PA-CDRAM organization and show how to incorporate it into the Rambus memory. We evaluate the approach using a cycle-accurate processor and memory simulator. Our results show that PA-CDRAM achieves up to 84 percent (28 percent on the average) improvement in the energy-delay product and up to 76 percent (19 percent on the average) savings in energy when compared to a time-out power management technique.

Index Terms—Memory design, power management, energy-aware systems, memory power management, cached DRAM.

1 Introduction

Energy consumption is a limiting constraint for both embedded and high-performance systems. In embedded systems, the lifetime of a device is limited by the rate of energy dissipation from its battery. On the other hand, energy consumption in high-performance systems increases thermal dissipation, thus requiring more cooling resources and accordingly increasing the system's maintenance overhead. For the majority of these systems, the memory subsystem consumes a large portion of the overall energy dissipation (for example, memory consumes 41 percent [1] and 23 percent [2] of the total system power in servers and portable devices, respectively), which motivates the need for efficient memory power management.

With the continuing advancement in the design and manufacture of faster and more powerful computing systems, more performance is demanded from the memory system. For example, in current chip multiprocessing (CMP) and simultaneous multithreading (SMT) processors, concurrent applications or threads allow the CPU(s) to issue more load and store requests in each cycle. Even with large caches, this higher demand coupled with a slow memory access time creates a potential performance bottleneck.

Memory has a huge internal bandwidth compared to its external bus bandwidth. The internal memory bandwidth can reach 1.1 Tbytes/s, whereas a fast external memory bus is in the range of 10 Gbytes/s [3]. To exploit the wide internal bus, *cached dynamic RAM* (CDRAM) adds a static RAM (SRAM) cache to the DRAM array on the memory chip [4]. Such a *near-memory cache* acts as an extra memory hierarchy level whose fast latency improves the average memory access time and potentially improves system performance, provided that the near-memory cache is appropriately configured. Moreover, in case of a cache miss, transferring a cache block over an external bus consumes four times more energy than transferring the same data over an internal bus (that is, it does not cross a chip boundary). This reduction is due to the smaller capacitance of internal buses (0.5 pF) compared to external buses (20 pF) [5].

In this paper, we explore the energy saving obtained by placing SRAM caches closer to the memory, rather than closer to the CPU. We integrate a moderately sized cache within the chip boundary of a power-aware multibanked memory. We call this organization *power-aware CDRAM (PA-CDRAM)*. In addition to improving performance, PA-CDRAM significantly reduces energy consumption in caches and in the main memory. Cache energy is reduced because 1) using small caches distributed to the memory chips reduces the cache access energy compared to using a large nondistributed cache and 2) near-memory caches allow the access of relatively large blocks from memory, which is not affordable with *near-processor* caches. Memory energy consumption is reduced by having a longer memory idle period during which DRAM banks can be powered off. PA-CDRAM improves the original CDRAM by tackling the interplay of the cache and memory organizations to optimize the memory's performance and energy consumption.

The innovation of this work is the use of near-memory caches to further reduce the memory's energy consumption beyond the savings achieved from traditional DRAM dynamic power management. CDRAM was proposed to improve performance; however, it may hurt energy consumption (as demonstrated in Section 3). Combining

• *The authors are with the Department of Computer Science, Sennott Square, University of Pittsburgh, Pittsburgh, PA 15260.*
E-mail: {nevine, childers, mosse, melhem}@cs.pitt.edu.

Manuscript received 22 Dec. 2005; revised 7 Oct. 2006; accepted 21 Feb. 2007; published online 6 June 2007.
Recommended for acceptance by M. Dubois.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0459-1205.
Digital Object Identifier no. 10.1109/TC.2007.70740.

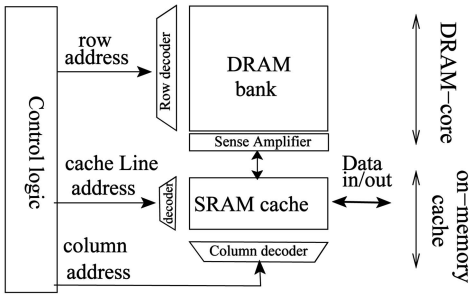


Fig. 1. Functional block diagram of a CDRAM.

CDRAM and DRAM power management would potentially benefit from the performance gains and save energy in memory.

The contribution of this paper is threefold. First, we propose near-memory caches for energy reduction in the memory hierarchy. This reduction comes without affecting the performance gain provided by CDRAM. Second, we describe an implementation of PA-CDRAM that integrates a near-memory cache in a Rambus chip (RDRAM). This description includes the changes made to an RDRAM chip, the near-memory cache controller, and the communication protocol. We also describe how our implementation can maintain backward compatibility with existing Rambus memories. Finally, the paper experimentally evaluates PA-CDRAM and shows that PA-CDRAM is more energy efficient than a traditional Rambus memory hierarchy employing a time-out power management policy.

2 Background

As background for the paper, this section describes the technologies and techniques that serve as the basis for PA-CDRAM.

Embedded DRAM (eDRAM). Integrating DRAM and logic cells on the same chip is an attractive solution to achieving both high performance (from logic cells) and high memory density (from DRAM cells). This integration avoids the high latency of going off chip by doing computation (or even caching) at the memory itself. Currently, manufactured chips with eDRAM and logic are mainly used in applications like computer graphics, networking, and handheld devices [6]. Based on the fabrication technology (either DRAM-based or logic-based), some degradation to the speed (density) of the logic (DRAM) cells may occur. For example, in early DRAM-based chips, logic cells were reportedly slower by 20 to 35 percent [6]. However, emerging fabrication technologies aim to overcome these penalties. For example, NEC's eDRAM chips offer DRAM-like density with SRAM-like performance [7] and IBM's third-generation eDRAM chips support two eDRAM families for high density and high performance to serve both purposes with no degradation [8].

CDRAM. To decrease the average memory access time, Hsu and Smith [4] proposed integrating a small SRAM cache within the memory chip next to the DRAM core, as shown in Fig. 1. Due to high internal bandwidth, large chunks of data can be transferred between the DRAM core and the near-memory cache with low latency. The average memory access time is improved by accessing the data through the fast near-memory cache rather than the slower DRAM. CDRAM chips were first manufactured by Mitsubishi [9] and are typically implemented using synchronous DRAM (*SDRAM*). Each memory bank has its own cache.

Hsu and Smith evaluated the performance of CDRAM in vector supercomputers. They showed an improvement in cycles per instruction (CPI) compared to the traditional memory without this extra cache. To improve the CDRAM performance, Koganti and Kedem [10] proposed a CDRAM with a wide cache line ranging from 4 to 8 Kbytes interleaved across multiple DRAM banks. Although the SRAM cache's miss rate is reduced, the authors did not account for the extra delay and energy spent accessing these large cache block sizes, which may degrade the performance and increase the memory's energy consumption. Hegde et al. [11] proposed using variable-width cache lines that fit the application access pattern to save the energy consumed by unnecessary traffic between the DRAM core and the near-memory cache. Zhang et al. [12] showed that CDRAM is able to improve performance as the instruction-level parallelism (ILP) degree increases. Past work did not explore the energy savings that can be achieved over a currently available alternative, such as power-aware memory, with the same overall system cache and memory capacity.

Rambus technology. Rambus [13] is a family of DRAM architectures that provide a high memory bandwidth. A Rambus chip, *RDRAM*, can be set in one of four power states. The states, in descending order of power consumption, are *active*, *standby*, *nap*, and *powerdown*. RDRAM chips can dynamically transition between power states to reduce energy consumption. Accessing data requires the chip to be in the active state. The lower the power state of a bank, the longer the synchronization delay needed to switch to the active state to service a request. RDRAM is connected using a memory bus, *Rambus Channel* (or, simply, a "channel"). A channel consists of four buses: a row bus, a column bus, a data bus, and a control register bus. Commands from the Rambus memory controller (MC) are sent through the channel as packets to be decoded by the control logic in the RDRAM.

Power-aware memories. Outside the context of CDRAM, Lebeck et al. [14] proposed the use of a power-aware allocation policy where data is allocated sequentially in each bank to increase bank idle periods. An implementation of the memory power manager in the Linux operating system [15] allocates memory pages to banks based on the running applications. Delaluz and Irwin suggested using compiler techniques to group the requested pages in memory based on the application's order of data accesses [16].

3 PA-CDRAM

CDRAM was originally proposed to improve system performance; however, it was not designed as a replacement to power-aware memory. Fig. 2 shows the average performance and energy consumption of CDRAM versus a traditional memory hierarchy for the same near-memory cache configuration used by Hegde et al. [11].[1] Each

1. Data produced when running the SPEC2000 benchmarks on SimpleScalar.

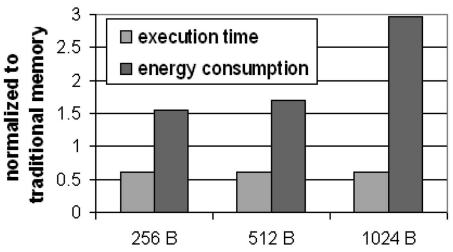


Fig. 2. Average performance and energy consumption for different near-memory cache block sizes.

CDRAM chip has a fully associative 4-Kbyte cache. We show the results for different cache block sizes (256, 512, and 1,024 bytes) for the embedded SRAM cache. Although CDRAM has a good performance improvement over the traditional memory, the memory's total energy suffers dramatically, with an increase of 1.5 to 3 times. This increase is due to the extra energy consumed from accessing the near-memory caches and transferring more data from the DRAM core at large block sizes.

The energy penalty of CDRAM can be overcome by decreasing the miss rate of the near-memory cache and using power management for the DRAM core. As we will show, making the CDRAM power aware not only decreases the energy penalty of CDRAM but also significantly improves *overall* memory energy in comparison to a traditional power-aware memory hierarchy.

Because DRAM power management typically relies on *idleness* to select power states, an improved miss rate in the near-memory cache increases the amount of idleness in the DRAM core, leading to more effective power management. One way to improve the miss rate is to increase the capacity of the near-memory cache. However, the total energy of the whole memory hierarchy is increased due to a greater overall cache capacity. Instead, we propose reallocating the existing cache capacity from the memory hierarchy's lowest cache level to the near-memory cache. For example, it may be possible to allocate the capacity of the L3 cache to CDRAM's near-memory cache. The L3 cache could then be eliminated, possibly without harming application performance. Moving the cache capacity to the near-memory cache has three advantages. First, the near-memory caches are distributed among the memory chips, which leads to lower energy consumption because the individual caches are smaller than one monolithic cache. Second, large data transfers are possible from the DRAM core to the near-memory cache (that is, there is more memory bandwidth, which makes large block sizes feasible). Finally, the near-memory caches can filter and avoid accesses to the DRAM core. Since near-memory caches can achieve lower miss rates than near-processor caches, such filtering increases idleness and lets the DRAM core stay in a low-power state for longer periods of time.

To build a PA-CDRAM, there are two main challenges that must be addressed: 1) how to configure the DRAM core's power management, assuming the use of multiple power states, and 2) what the best configuration for the near-memory cache to balance energy and performance is. We describe each of these challenges and our way of addressing them below.

3.1 DRAM-Core Power Management

With a near-memory cache, we propose applying aggressive power management in the DRAM core. During a chip's idle time, the MC can immediately transition the DRAM core to the idle state after servicing all outstanding requests. This is equivalent to the use of a time-out policy with an idle threshold of 0 sec. Although a zero-threshold policy increases the total inactive time, it can degrade performance and increase the total energy consumption when too many requests are directed to a memory chip. The extra delay and energy overheads are due to the transitional cost between power states.

In PA-CDRAM, we avoid this problem by choosing the near-memory cache configuration in a way that increases the hit rate while reducing the DRAM core's energy consumption.[2] When most data requests are serviced as cache hits in the near-memory cache, the longer interarrival time between requests that reach the DRAM core make it cost-effective to immediately deactivate banks after servicing outstanding requests.

We choose to keep the near-memory cache active all the time to avoid delays that may be caused by on-demand activation of the cache at each request.

3.2 DRAM Core versus Near-Memory Cache Energy Trade-Off

To reduce the memory's energy consumption, we need to consider the effect of the near-memory cache configuration on the energy consumption of both the near-memory cache and the DRAM core. The two factors that affect the cache energy consumption and access latency—for a given cache size and fixed number of cache subbanks—are the *associativity* and the *block size* [17].

The cache associativity directly affects the miss rate. Increasing associativity reduces the cache miss rate and vice versa. One goal of PA-CDRAM is to keep the near-memory cache miss rate as low as possible because it directly influences memory energy consumption in two ways. First, the higher the miss rate, the more activity in the DRAM in terms of transitioning from the idle to the active state, performing address decoding, and transferring data. Second, the lower the miss rate, the longer the DRAM-core idle time. To keep the miss rate at a minimum, we use fully associative caches to eliminate any conflict misses. We argue that, in most cases, performance improvement and energy saving from reducing near-memory miss rates (given the appropriate cache configuration) can outweigh extra delay and energy consumed in accessing higher associative caches [18].

Reducing the miss rate only is not sufficient to reduce the memory's energy consumption and application's overall delay. Thus, in PA-CDRAM, we account for the effect of the cache configuration on the overall memory's energy (in contrast to that of Koganti and Kedem). After selecting the cache associativity, choosing a near-memory cache block size creates a trade-off between the near-memory cache and DRAM-core energy consumption. Small cache blocks have

2. Different configurations for the L3 cache would not have the same effect: Near-memory caches have a much wider bandwidth to memory than L3 caches.

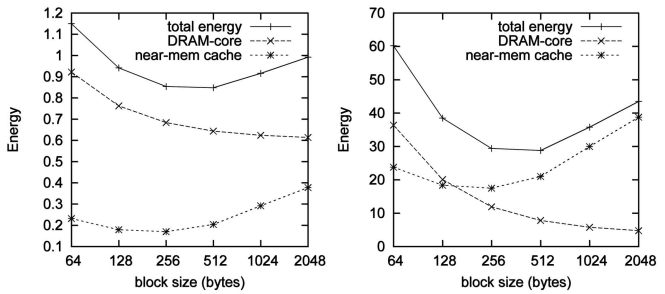


Fig. 3. The effect of different cache block sizes on the memory energy consumption for (left) *bzip* and (right) *mcf*.

the advantage of fast hit time and low energy per access. However, smaller blocks imply frequent accesses, consequently increasing the DRAM-core energy due to the increased activity. The increase in the DRAM-core energy rises as a result of increasing the memory activity (such as power-state transitions, address decoding, and data transfer). Conversely, larger near-memory cache block sizes reduce the DRAM activity but increase the near-memory cache energy consumption and latency due to accessing these large blocks.

This trade-off is illustrated in Fig. 3. The figure shows the energy consumption in the near-memory cache and DRAM core at different cache block sizes. Energy values are obtained using a simplified energy model. The model estimates the near-memory cache energy consumption, E_{cache}, and that of the DRAM core, E_{Dcore}, as a function of the number of near-memory cache and DRAM-core accesses. From this model, given the number of cache accesses and the approximate execution time for an application, we can roughly estimate the memory energy consumption at different block sizes. We use SimpleScalar [19] to estimate the input parameters (the number of cache accesses and execution time).

In Fig. 3, we show estimated energy for two of the SPEC2000 benchmarks, *bzip* and *mcf*, as an example of CPU and memory-intensive applications, respectively. In *bzip*, the DRAM-core idle energy dominates the PA-CDRAM energy, whereas, in *mcf*, the frequent accesses to the near-memory caches makes the cache energy dominate the total energy at large block sizes. From the figure, we see that the trade-off between the near-memory cache and DRAM-core energy consumption creates a sweet spot between block sizes 256 and 512 bytes. Note that finding ideal block size is application dependent. However, from our simulation, in most of the applications, the minimum energy-delay product can be achieved at or within a slight margin of one of these two block sizes [20].

From this section, we conclude that, for the given cache size, the near-memory cache should be fully associative (to reduce the miss rate) and have a block size of either 256 or 512 bytes (to balance the memory's energy and delay). For the DRAM core, setting the chip to the idle state after servicing outstanding requests (that is, $\mathrm{timeout} = 0$) is expected to save the memory energy consumption. The implementation described in the next section uses such a configuration.

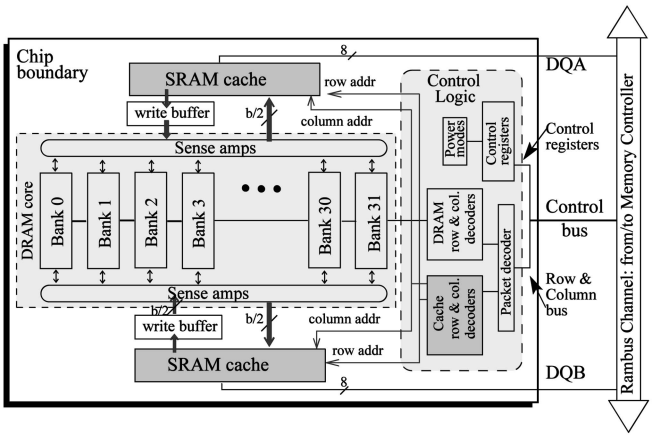


Fig. 4. Functional block diagram of a PA-CDRAM.

4 PA-CDRAM Implementation

In this section, we describe the architectural and operational modifications needed to integrate a near-memory cache in RDRAM. First, we describe the organizational issues of integrating a near-memory cache in the memory chip. Next, we discuss the design of the controller for the near-memory cache because it has energy, performance, and backward-compatibility implications. Finally, we describe the operation of the controller, including some changes needed to the Rambus bus protocol.

4.1 PA-CDRAM Architecture

Our PA-CDRAM design modifies the original RDRAM design for power efficiency. Besides the addition of the near-memory cache, some alterations are needed in the main components of the RDRAM, namely, the DRAM core, the control logic, and the memory and cache controllers (in Section 4.2).

Near-memory cache. We add a fully associative cache (depicted as dark blocks in Fig. 4), with its data array divided into two sections. Since the original RDRAM design has a divided data bus, each section of the cache is connected to one of two internal data buses (DQA and DQB). Each cache section stores half of each cache block. We keep the original RDRAM write buffers before the sense amplifiers. The write buffers are used to store replaced dirty blocks from the near-memory cache to be written to the DRAM core. The power state of this cache is independent of the power state of the other chip components. In the nap state, the RDRAM internal clock is periodically synchronized with the external clock [13]. Thus, the near-memory cache is accessible even when the DRAM is in the nap state.

DRAM core. To accommodate the large transfer sizes between the DRAM core and the near-memory cache, wider internal data buses are required to connect the sense amplifiers with the near-memory cache. The width of each bus connecting the DRAM and near-memory caches is $b/2$ bits, where b is the size of a cache block. The buses DQA and DQB remain at an 8-bit width.

Control logic. Extra cache row and column decoders are added in the chip's control logic for decoding cache addresses. In addition, the existing RDRAM packet decoder in each chip is modified to decode new cache commands.

TABLE 1
PA-CDRAM Cache Commands Sent across the Control Bus

command	description
cache_read_addr *(CRA)*	Issues a read request for block_addr; data is sent on the data bus only if there is a *hit* in the tag comparison. Command packet is sent through the row bus. Format: opcode(4b), chip_id(4b), block_addr(15b).
cache_write_addr *(CWA)*	Issues a write request for block_addr (data is sent later on the data bus). Command packet is sent through the row bus. Format: opcode(4b), chip_id(4b), block_addr(15b).
cache_tag_test *(CTT)*	Sends the tag comparison result (tag_flag can be *hit* or *miss*). If *hit*, the following fields are ignored. If *miss*, it indicates the address of block to be replaced and whether it is dirty or not. Command packet is sent through the column bus. Format: opcode(4b), chip_id(4b), block_addr(15b), tag_flag(1b), dirty_flag(1b), replaced_addr(15b)

The new cache commands indicate whether the access is a hit/miss in the cache and whether replacement is needed. The new commands and their fields are defined in Section 4.3.

4.2 Near-Memory Cache Controller

In the design of PA-CDRAM, a controller is needed to handle hits and misses in the near-memory cache. An important design question is where this controller should be placed. One alternative is to have a single near-memory cache controller (called a *centralized controller*) that is integrated into the main Rambus MC. Another alternative is to have a near-memory cache controller *per* Rambus chip (called a *distributed controller*), with that controller integrated into the Rambus chip itself.

In the centralized-controller case, there is a tag array for each memory chip. The centralized controller keeps track of the control data for each block, such as a valid bit, a dirty bit, and LRU counters. This way, the cache controller is able to locally decide on replacement policies and update the tag arrays. The centralized controller communicates with cache data arrays by sending cache commands through the channel (as described in Section 4.3). In the distributed controller case, the cache controller is integrated more tightly with the near-memory cache (they are on the same chip). This design provides backward compatibility with current Rambus MC chips, as well as flexibility of connecting PA-CDRAM and RDRAM chips in the same channel. In the distributed design, each controller has its own tag array and the same functionality as the centralized controller. The distributed design has to intercept the control packets received from the MC through the channel and issue consequent command sequences to either the near-memory cache or the DRAM core.

The centralized controller has the advantage of performing a relatively faster tag comparison because it is implemented in the same technology as the MC. Thus, the centralized controller has a faster average memory access time compared to the distributed design, where the controller runs at a slower speed (due to the mixed use of logic and DRAM). For example, a delay penalty of 25 percent for logic cells with the distributed design has a 25 percent slower cache hit time versus a roughly 16 percent penalty with a centralized controller design.[3] On the other hand, the bus activity (and, hence, the energy) in the Rambus channel is reduced by using a distributed design since only memory control packets are sent across the Rambus channel, whereas, in the centralized design, cache and memory control packets have to be sent across the channel. Hence, the distributed design has less bus energy but suffers a performance penalty relative to the centralized design. In our evaluation (in Section 6), we examine the trade-off between these two alternatives.

Next, we describe PA-CDRAM's operation. The difference between the centralized and distributed controllers is that, in the former, we need extensions to the Rambus protocol. The protocol extension is required because the cache controller resides in the MC and therefore needs to command the near-memory cache from afar, necessitating a new protocol.

4.3 PA-CDRAM Operation

In the PA-CDRAM distributed-cache-controller design, after receiving the data request on the Rambus channel, the cache controller performs the tag comparison and reads data from the near-memory cache if the tags match. Otherwise, the cache controller internally sends a sequence of control signals to activate a DRAM bank, read data, and precharge the data line according to the DRAM-core timing constraints.

For PA-CDRAM with a centralized cache controller, we need to extend the Rambus communication protocol between the memory/cache controllers and the memory chips. This extension is needed because the cache controller resides within the MC chip; thus, it needs to drive the caches (invalidate lines, evict lines, and so forth) in the PA-CDRAM chips through the Rambus channel. For that, we define three new commands to communicate with the near-memory caches, as shown in Table 1.

To perform a data read/write request from the memory, a sequence of commands is issued along the Rambus channel. A read (write) can result in either a near-memory cache hit or a miss. Fig. 5 illustrates the sequence of commands and data on the row, column, and data buses (timing diagrams) in the channel for a read hit and read miss. The communication protocol for command packets for a write hit/miss is similar to the read hit/miss, respectively, with the substitution of CRA by CWA command and the direction of data. The figure also shows the timing

3. According to Cacti, tag comparison takes around 35 percent of the total cache hit time for the selected configuration. That is, $x + (1 - .35)x \cdot .25 = 1.16x$.

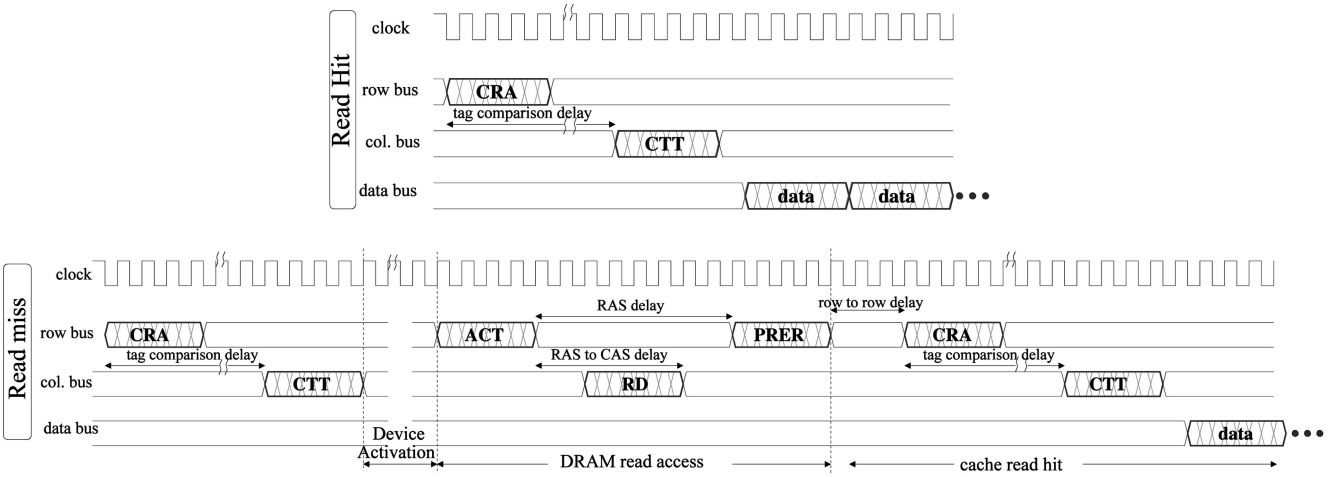


Fig. 5. Timing diagram of the Rambus channel for near-memory cache read hit (upper) and read miss (lower) transactions.

constraints imposed by the Rambus protocol on the earliest time to send subsequent packets to the same chip.

For each request, the MC identifies which chip the requested data resides in and then lets the cache controller search the tag array corresponding to this target chip. During the tag comparison, the cache controller sends the requested block address through the channel using the CRA command. After the comparison, the cache controller sends the comparison result using the CTT packet. In the case of a near-memory cache hit (tag_flag is set), data is read directly from the near-memory cache. In case of a near-memory cache miss, the cache controller uses the CTT packet to send the address of the block to be replaced and whether or not it is dirty. If the block is dirty (that is, dirty_flag is set), then it is written to the write buffer. Meanwhile, the MC sends a packet to activate the chip (ACT), followed by a read command for a memory address (RD), and then an optional command to precharge the data line (PRER). The memory command sequence RD-ACT-PRER is part of the original Rambus protocol. After the data is read from the memory address, it is copied to the near-memory cache through the sense amplifiers. The cache controller then sends another command sequence requesting the new block, which is treated as a cache hit.

If there are no outstanding requests to a chip, then the RDRAM chip does an automatic copy of the write buffers to the correct banks during the chip idle periods. After writing the contents of the write buffers, the MC issues a command to send the chip to nap. Note that the cache commands use both the row and column control buses. This is to increase the memory pipelining and decrease the delays for memory access (even if it is near-memory cache accesses).

5 Energy and Delay Modeling of PA-CDRAM

In this section, we develop a simple analytical energy model to highlight the main factors that affect PA-CDRAM energy savings compared to the traditional memory. The memory hierarchy consists of L1, L2, L3, or near-memory caches, and the DRAM core. PA-CDRAM uses the same cache capacity as in the traditional memory; however, the capacity of the L3 cache is distributed on near-memory caches. Our model is based on system parameters (such as caches' access energy and latency, as well as the time and power consumed by DRAM in each power state) and application parameters (such as the number of L2, L3, and near-memory cache misses).

The model accounts for energy consumed in the caches (E_{cache}), the DRAM-core (E_{Dcore}), and system buses (E_{bus}). We do not account for the cache leakage energy (proportional to the size of the SRAM) as we compare our results against a base case with equal cache capacity and we assume it to be negligible:

$$E_{cache} = E_{c\_accs} \cdot \#c\_accs,$$
$$E_{Dcore} = E_{d\_accs} \cdot \#d\_accs + E_{trans} \cdot \#trans + P_{idle} \cdot T_{idle},$$
$$E_{bus} = E_{b\_accs} \cdot \#b\_accs,$$

where $\#c\_accs$, $\#d\_accs$, and $\#b\_accs$ are the number of L3 (or near-memory) cache accesses, DRAM-core accesses, and bus transactions, respectively. The cache energy per access, E_{c_accs}, is obtained from the Cacti tool [17] for both the L3 and near-memory caches, whereas the DRAM core's energy per access (E_{d_accs}), the power state transition energy (E_{trans}), and the idle power (P_{idle}) are specifications of the particular RDRAM memory chip used [13]. T_{idle} is the time spent in the DRAM idle state. To simplify the memory's energy estimation, we assume an application with a limited number of write-backs and capacity misses in the L3 or near-memory caches. We also assume a memory power management technique that immediately deactivates the DRAM-core after each access. In other words, the number of memory power state transitions is double the number of DRAM accesses: $\#trans = 2 \cdot \#d\_accs$. We derive the number of cache and DRAM-core accesses as follows:

$$\#c\_accs^B = \#c\_accs^{PA} = \#L2\_misses,$$
$$\#d\_accs^B = \#L3\_misses = \#c\_accs^B \cdot L3\_miss\_rate,$$
$$\#d\_accs^{PA} = \eta \cdot \#c\_accs^{PA} \cdot L3\_miss\_rate,$$

where the superscript indicates whether the metric represents PA-CDRAM (PA) or the traditional memory (called B, the base case). $\eta = \frac{\#d\_accs^{PA}}{\#d\_accs^B} = \frac{near-memory\ miss\ rate}{L3\ miss\ rate}$ represents the improvement in the near-memory average miss

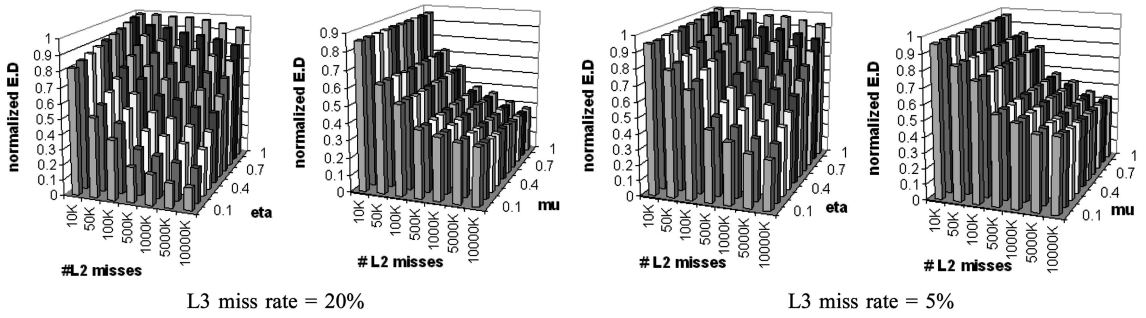


Fig. 6. The effects of varying η and μ on the energy-delay product.

rate with respect to the traditional L3. η is affected by the spatial and temporal locality of the application's data and can be estimated using cache models that predict cache miss rates [21].

To compute the amount of energy spent in the DRAM core, we estimate the time spent at each of the DRAM active and idle power states. We compute T_{idle} in terms of the execution time of an application (T_{exec}) and the time spent in the DRAM active state (T_{active}), as shown in the equations below. T_{d_accs} is the average DRAM access time. T_{exec} is derived from the execution time of an application with an access latency equal to zero for data accesses beyond L2 ($T_{perfect}$)[4] and the average access time of the combined DRAM and L3 (or near-memory) cache (T_{mem_accs}):

$$\begin{aligned} T_{idle} &= T_{exec} - T_{active}, \\ T_{active} &= T_{d\_accs} \cdot \#d\_accs, \\ T_{exec} &= T_{perfect} + \mu\,(T_{mem\_accs} \cdot \#d\_access). \end{aligned}$$

μ is the fraction of memory access time that does not overlap with the processor's computation (resulting in CPU stalls). μ is affected by the processor's design (for example, issue width, depth of Load/Store queue) and its effect on hiding memory latency. Both η and μ are application dependent and their values range from 0 to 1 under the above assumptions. η can exceed one in case of high-capacity misses and large write-back traffic.

Fig. 6 shows the effect of varying the number of L2 misses, η and μ on the energy-delay product. PA-CDRAM memory consists of a 256 Kbyte near-memory cache with 512 byte blocks versus a 2 Mbyte L3 cache with 128 byte blocks in the base case.[5] The results shown are for a duration of $T_{perfect}$ = 250 ms and fixed L3 miss rates: 20 percent and 5 percent as examples for moderately high and moderately low miss rates. The graphs show that varying η highly affects the savings in the energy-delay product in PA-CDRAM compared to the traditional memory. This is due to the fewer accesses to the DRAM core and consequent idle time in memory. On the other hand, μ has less visible impact on the energy-delay product primarily because the majority of the L2 misses can be serviced by either the L3 or the near-memory caches, which have similar cache access latency. For smaller L3 miss rates, the PA-CDRAM energy-delay saving is smaller due to the lower memory traffic.

Thus, given the application's L2 misses, cache and memory power, and delay characteristics, and estimation of the L3 and the near-memory miss rates, we can estimate whether using PA-CDRAM is more efficient than using the traditional memory hierarchy under the above assumptions.

The intention of the above model is to highlight the runtime factors affecting PA-CDRAM energy and delay with respect to traditional memories. Other static factors, such as per-access cache and DRAM energy and latency (considered as fixed parameters in the above model), play a role in evaluating the effectiveness of PA-CDRAM. Next, we present a detailed analysis of the two memory models using simulations and discuss the effect of the different system parameters on the energy and delay of PA-CDRAM.

6 Evaluation of PA-CDRAM

In this section, we describe several experiments that explore the energy and performance of PA-CDRAM. We evaluate the different PA-CDRAM configurations and their impact on energy and performance. Then, we analyze the performance and energy benefits of PA-CDRAM compared to the conventional power management employed by Rambus. We also evaluate the energy and performance implications of the centralized and distributed near-memory cache controller designs and how well PA-CDRAM works in a multitasking environment, where Rambus memories are most likely to be used. Finally, we present a sensitivity analysis of PA-CDRAM with respect to varying the different system parameters. We begin with a discussion of our experimental methodology.

6.1 Methodology

To evaluate PA-CDRAM, experiments are performed using the SimpleScalar architecture simulator [19] combined with a memory module [22] that models a set of RDRAM chips connected to a single Rambus channel. The memory model also simulates the MC and its request scheduling. We extended the memory model by implementing the CDRAM memory structure and RDRAM's power management scheme with multiple power levels and a time-out policy for power mode transitions.[6] Simulations are performed using a set of applications from the SPEC2000 benchmark

4. $T_{perfect}$ is the execution time when using perfect memory and L3 (or near-memory) caches.

5. The cache hierarchies used are the same as in Section 6.

6. We limit our approach to using the *nap* state as the only low-power state because the results in [14] and [15] showed that the nap state is energy efficient and has relatively low transition delay.

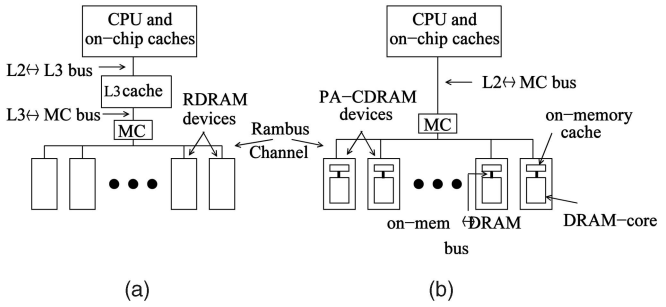


Fig. 7. Memory organizations of (a) the base case and (b) the proposed PA-CDRAM.

TABLE 2
System Configuration

Processor: 4-issue out-of-order, 2GHz
Cache hierarchy: L1: on-chip 32KB iL1 & dL1 caches, DM, 32B, 2 cyc. lat. L2: on-chip 256KB, 8-way, 64B, 5 cyc. lat. L3: off-chip 2MB, 8-way, 128B, 15 cyc. lat near-memory: 8x256KB, FA, 512B, 19 cyc.(14+5 slowdown)
DRAM-core: 8 x 32MB RDRAM , 85-120 cyc. lat
Buses: address width = 32b, C_{in} = 0.5 pF, C_{ex}= 10pF

suite. To avoid the cold-start effect, we fast-forward the simulation two billion instructions and simulate the following 200 million instructions as in [23].

Our study evaluates the energy consumption and delay of PA-CDRAM against a base case that employs traditional power-saving policies implemented by Rambus. Since we are concerned with the memory subsystem, we measure the energy-delay product considering memory energy and overall delay. This metric would improve if we account for the CPU energy. Since we do not apply any processor power management and PA-CDRAM reduces the execution time (as will be shown in Section 6.3), PA-CDRAM reduces the processor static energy by reducing the application's runtime. However, we only include memory energy in our results to be conservative.

Fig. 7 illustrates the memory models evaluated in our experiments. Table 2 summarizes the system configurations used in the two memory models. The base case is illustrated using a specific cache hierarchy configuration similar to the Pentium 4 Extreme Edition (EE) processor [24]. Latency values are given in terms of CPU cycles. Note that this is a more modern cache hierarchy setting that is different than the one used in Fig. 2. In the base case, data allocation is done linearly to keep the least number of chips in the active state [14], [15]. In PA-CDRAM, we use interleaved memory mapping to make use of all near-memory caches and use the same L1 and L2 configurations as the base case. DRAM power management in the base case uses a time-out policy for the deactivation of the RDRAM chips to the nap state.

We compute the energy consumption in the DRAM core, caches, and buses. During DRAM-core accesses, dynamic energy is consumed which is proportional to the number of DRAM accesses. Otherwise, the DRAM core consumes static energy that is proportional to the time spent in each power state. The timing and power characteristics of the simulated RDRAM chip are for a typical RDRAM chip, namely, the 256 Mbits/1,066 MHz/32 split bank architecture [13].

Access energy and latency for each cache configuration is obtained using Cacti 3.0 for 130 nm (the same manufacturing technology as the Pentium EE). In Cacti, the per-access energy and latency is divided into portions consumed in tag array and data array (including sense amplifiers and output latches). In n-way set associative caches, tag and data arrays are accessed concurrently to reduce the total access time. In fully associative caches, a tag array is replaced by a fully associative decoder, after which the data array is accessed. Tag comparison takes place in the decoder. Then, the decoder drives the wordline associated with the cache entry. This optimization reduces cache per-access energy; however, it increases per-access latency. We obtain access latencies and energy when operating at cache voltage $V_{dd} = 1.3$ V. As discussed in Section 2, we add a conservative delay penalty of 35 percent for accessing logic cells in the memory chip [6]. We also add four cycles of delay penalty for accessing off-chip caches.

We refer to a bus (address and data) by the two memory elements that the bus connects; for example, $\mathrm{L2} \leftrightarrow \mathrm{L3}$ is the bus connecting L2 and L3 caches. In our evaluation, we account for $\mathrm{L2} \leftrightarrow \mathrm{L3}$, $\mathrm{L3} \leftrightarrow \mathrm{MC}$, and the Rambus channel in the base case and account for $\mathrm{L2} \leftrightarrow \mathrm{MC}$, the Rambus channel, and $\text{near-memory} \leftrightarrow \mathrm{DRAM}$ in PA-CDRAM, as shown in Fig. 7. The energy of external and internal buses is computed using the model presented in [5] and [25]. Unless stated otherwise, we evaluate a centralized-cache-controller design. The PA-CDRAM and base-case energy models are detailed in [26].

We set our experiments using memory/cache configurations that exhibited the best energy-delay product results across all applications. From experiments with different time-out values (0, 100, 500, 1,000, 5,000, and 10,000 cycles), we found that a time-out of 1,000 cycles is the best fixed threshold for the base case, whereas a zero time-out threshold achieved the lowest energy-delay product on the average for the PA-CDRAM. Experimenting with the cache block sizes (128, 256, 512, and 1,024 bytes) and associativity (eight-way set and fully associative), the least energy-delay product for the L3 cache in the base case was realizable using a cache with an eight-way set with 128-byte blocks. The best configuration for the PA-CDRAM was fully associative with 512-byte blocks. Further details can be found in [20]. We omit these results due to space limitations.

6.2 Energy and Delay

Across all SPEC2000 benchmarks, the average savings in energy-delay product is 28 percent and up to 84 percent, as shown in the last column in Fig. 8. This is an aggregate behavior; to analyze the benefits of PA-CDRAM on energy and performance independently, we decompose the energy-delay product into execution time and energy consumption. Fig. 8 shows these metrics for each application normalized to the base case.

6.2.1 Effect on Delay

For most applications, there is no significant improvement in the delay over the base case for two reasons. First, in most of the applications, L2 can service a large number of

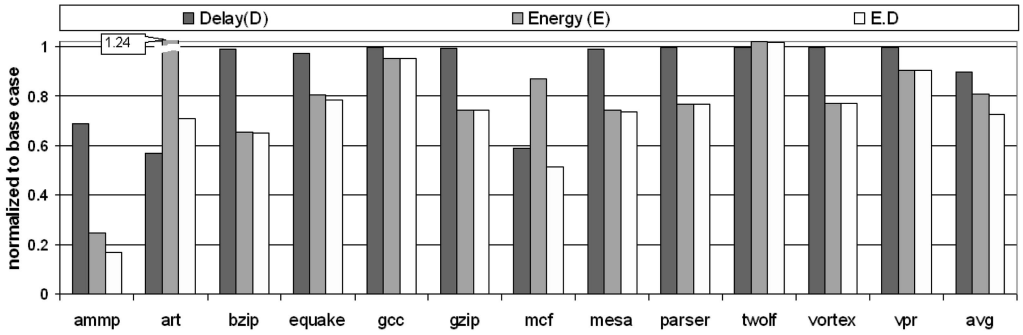
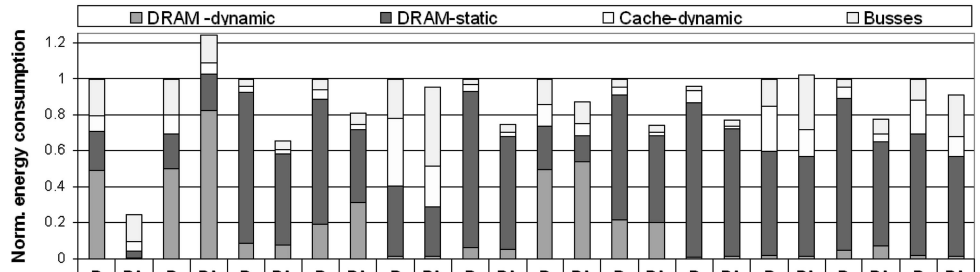


Fig. 8. PA-CDRAM energy-delay breakdown.



	ammp	art	bzip	equake	gcc	gzip	mcf	mesa	parser	twolf	vortex	vpr
B	49.96	40.10	22.19	22.73	0.30	13.97	23.90	25.66	4.30	0.59	5.28	0.80
PA	0.09	20.00	5.59	10.57	0.08	3.51	8.00	6.98	1.82	0.15	2.05	0.21

Fig. 9. PA-CDRAM and base-case energy breakdown and cache miss rates.

the memory requests. Second, most of the CPU stalls resulting from L3 misses can be masked by the execution of other independent instructions in the pipeline, that is, μ is very small. Three exceptions in Fig. 8 are *ammp*, *art*, and *mcf*. For these benchmarks, the total number of DRAM-core accesses is two orders of magnitude larger than the other applications. With so many memory accesses, a reduction in the average memory access time significantly improved performance. This shows that near-memory caching is well suited for memory intensive applications.

Compared to the motivational example presented in Section 3, there is a difference in performance improvement. This difference is due to the use of a different cache hierarchy in each experiment. In Fig. 2, we use the same cache hierarchy used in [11], which proposes a different cache configuration. Additionally, the results in Fig. 2 use an extra cache level in memory chips for CDRAM (resulting in a larger overall cache capacity than the base case). The extra cache capacity improves execution times. However, for a more fair comparison, we use the same overall capacity in both memory models and a more modern cache hierarchy that includes the L2 cache. Hence, the overall execution time becomes less sensitive to the memory latency, which is a desirable effect in system design.

6.2.2 Effect on Energy Consumption

In Fig. 8, we also see that savings in energy consumption alone reached up to 76 percent (for *ammp*). All applications exhibit energy savings except for *art* and *twolf*. To analyze these savings (and higher consumption in *art* and *twolf*), we further decompose the energy consumption to show where the energy is spent in each of the memory's individual components. Fig. 9 shows the relative consumption of the DRAM-core dynamic energy (DRAM-dynamic), DRAM-core static energy (DRAM-static), cache access energy (cache-dynamic), and bus energy. The bottom of Fig. 9 shows the cache miss rates for L3 in the base case and the average miss rate of all near-memory caches in PA-CDRAM. Note that the miss rates in PA-CDRAM are lower than those in the base case in all applications. Thus, there exist fewer accesses to the DRAM core in the case of PA-CDRAM. In the figure, η ranges from 0.25 to 0.5 (except in $ammp = 0.002$), which indicates a potential for energy saving as discussed earlier.

DRAM-core energy. In all applications, the DRAM-static energy was reduced due to the increase in the duration of DRAM idle periods versus active periods in the base case. With respect to the DRAM-dynamic energy, some applications, namely, *ammp*, *bzip*, *gcc*, *gzip*, *mesa*, *twolf*, and *vpr*, have lower energy due to lower miss rates in the near-memory cache that filter some of the accesses to the DRAM core. All of the above applications have relatively small η (around 0.26). Note the effect of the extremely low value of η on reducing the energy-delay product of *ammp*. However, applications like *art*, *equake*, *mcf*, *parser*, and *vortex* suffer an increase in DRAM dynamic energy, even though near-memory cache miss rates were reduced (η values around 0.42). This increase is due to the relatively large near-memory cache block size that caused these applications to access unnecessary data from the DRAM core.[7] During these excessively large transfers, the DRAM consumes extra energy by reading/writing data that is never used by the application. We deduce that increasing the spatial locality

7. This factor was not considered in the model in Section 5 for simplicity.

TABLE 3
Per-Access Latency and Energy Breakdown of L3 and Near-Memory Caches

cache	tag-array	data-array	total
L3 latency (ns)	3.69	5.41	5.41
L3 energy (nJ)	0.388	4.614	5.00
PA latency (ns)	-	4.94	4.94
PA energy (nJ)	-	3.084	3.44

for an application saves further energy in PA-CDRAM dynamic energy compared to the base case and vice versa.

Cache energy. As cache access energy is proportional to the length of activated bitlines and wordlines at each access, dividing the cache into smaller near-memory caches reduces energy. Table 3 lists the breakdown of per-access energy and delay for both L3 and near-memory caches as obtained from Cacti. For near-memory caches, the tag decoder latency and energy are included in the data-array side. Near-memory access latency in the table excludes the 35 percent slowdown penalty. The figure shows that, when using PA-CDRAM, there is a decrease in cache energy across all applications, even with small reduction in miss rates.

Bus energy. The total energy of a bus depends on its capacitance and activity. PA-CDRAM can reduce bus energy consumption (as in *ammp*, *art*, *equake*, *mcf*, and *mesa*) by reducing the total bus capacitances compared to the base case (three external buses versus two external and one internal bus for PA-CDRAM). However, in the other applications, bus energy increases due to the increased Rambus channel activity. Thus, applications with high L2 misses and low L3 misses consume more bus energy in PA-CDRAM. On the other hand, for applications with relatively high L3 misses, frequent activity occurs in $\mathrm{L3} \leftrightarrow \mathrm{MC}$ and the channel, resulting in higher bus energy in the base case.

6.3 Near-Memory versus Near-Processor Caches

In this study, we evaluate the potential energy and performance benefit of allocating the capacity of an L3 cache closer to the processor versus closer to the memory. In our experiments, we compared against systems where the majority of the cache capacity is allocated as 1) a large on-chip L2 cache and no L3 cache (not shown here), 2) a large on-chip L3 cache, or 3) a large off-chip cache. The capacity of the large cache in each of these systems is equal to 2 Mbytes. In the first case, we observed that the energy-delay product is much higher than having a large L3 cache (our base case described earlier). The increase is, on average, 2.97 times and up to 5.34 times the base case. A larger L2 size causes longer access latency and energy per access than a smaller L2. Since L2 is very frequently accessed, increasing its access latency and energy results in a significant degradation in performance and increase in the total energy consumption. Thus, we show the more fair comparison against a system with a large L3 cache.

Fig. 10 compares the energy-delay product of using the traditional memory with off-chip L3 or on-chip L3 caches versus PA-CDRAM with near-memory caches. On-chip L3 improves over off-chip L3 as it saves the energy consumption and the extra delay spent in accessing the $\mathrm{L3} \leftrightarrow \mathrm{L3}$ external bus. Comparing PA-CDRAM against a memory hierarchy with a large on-chip L3 cache shows that most of the applications achieve a lower energy-delay product. This is due to the use of smaller near-memory caches and the lower miss rates in PA-CDRAM. On the other hand, *gcc*, *twolf*, and *vpr* experience high $\mathrm{L2} \leftrightarrow \mathrm{MC}$ and $\mathrm{MC} \leftrightarrow \mathrm{near\text{-}memory}$ bus traffic, which overshadow savings in the other memory components. However, across all applications, PA-CDRAM improves the average energy-delay product over using a large on-chip L3 cache by 17 percent.

Moreover, we compare the benefit of using an L3 cache (on-chip and off-chip) that is divided into eight banks. We find that, even when compared with an eight-bank on-chip L3 cache (configuration with the lowest energy consumption in the base case), PA-CDRAM achieves a 15 percent lower energy-delay product.

6.4 Cache-Controller Location

We quantify the trade-offs of choosing the location of the near-memory cache controller with respect to the overall performance and energy consumption. We simulate the centralized and distributed cache controllers and penalize the near-memory cache access latency accordingly. For the centralized controller, we use Cacti's latency computation to slow down the access latency, except for the tag comparison (done outside the DRAM chip). For the distributed design, we apply the penalty (slowdown) to the entire near-memory access latency (including tag comparison, which takes place inside the DRAM chip). As expected, the total execution time of an application is slightly faster (by up to 1.5 percent) when using a centralized controller. On the other hand, the energy consumed in the memory buses is up to 5 percent less for the distributed controller design. Fig. 11 shows the effect of

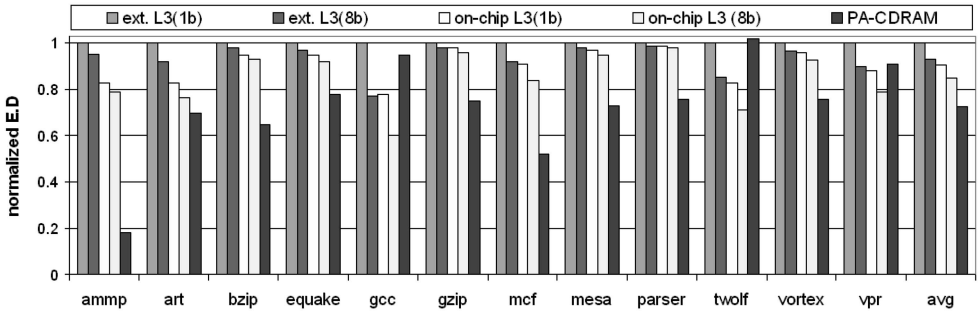


Fig. 10. Energy-delay product of near-memory versus near-processor cache organizations.

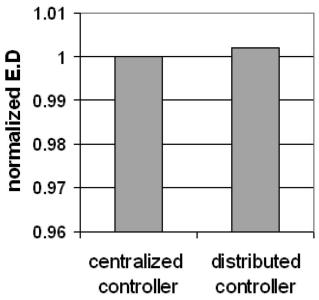


Fig. 11. Energy-delay product of the distributed controller normalized to the centralized-cache-controller design.

the cache-controller location on the bus energy, the application's execution time, and the overall energy-delay product. Although the distributed design has less bus energy, the overall energy was actually increased. This is due to the longer execution times that increase the DRAM-static energy, which overshadows the bus energy savings. Although the centralized design has a 0.4 percent average improvement in energy-delay, the distributed design is likely to be preferable because it is backward compatible. Hence, we stress an important feature of the distributed controller: It enables the use of traditional Rambus and PA-CDRAM chips interchangeably while connected to the same MC. In that sense, it is possible to do incremental/partial deployment of the PA-CDRAM chips if it is not possible to do it homogeneously. We leave further study of this issue for future work.

6.5 Effect of Multiprocess and Multithreaded Environments

The energy profile of an application running in a single task environment may differ from running the same application in a multiprocess system with preemption or in a multithreaded environment with multiple processors. This difference arises from the fact that an application's locality of reference is disturbed by the execution of other applications which will populate the cache with their own data. The result is the higher miss rates experienced by each application. It is important to evaluate how these higher miss rates affect the energy-delay of PA-CDRAM since it is intended for such environments.

We first start with single processors running multiple applications. Although our simulation infrastructure does not directly support a multitasking workload, we can approximate the effect of context switches. Our approximation uses a task scheduler that invalidates all of the cached data for the expiring application (task) before resuming execution of the ready task.[8] The invalidation writes back all of the dirty cache blocks to memory. We trigger an interrupt service routine every 10 ms (similar to the time slice in Linux). The interrupt drains the processor pipeline and flushes the data in the caches. Upon the interrupt termination, the application's execution is resumed.

Fig. 12 illustrates the overhead of context switching compared to a single task execution. In all of the applications except *mesa* and *vortex*, the overhead of context switching is lower in PA-CDRAM than in the base case for two reasons. First, the time overhead of flushing the L3 cache is larger than the near-memory caches as flushing all of the small near-memory caches can be done simultaneously, whereas flushing the L3 cache serializes the write-backs to the memory chips. Second, since the number of blocks in the L3 cache (2 Mbytes/128 bytes) is larger than the near-memory caches (2 Mbytes/512 bytes), the L3 cache has more address decoding and, thus, more energy is consumed. In *mesa* and *vortex*, the energy-delay product of the context switching overhead is higher in PA-CDRAM due to an increase in DRAM-dynamic energy that exceeds the time savings from the cache invalidation, as mentioned above. This energy increase is due to the relatively large number of near-memory cache write-backs factored by the large block size to be written to the DRAM array.

Furthermore, we evaluate PA-CDRAM energy consumption when used in a multiprocessor environment. To emulate the multiprocessor effect, we generate traces of L3 accesses for all applications. For each application pair, we merge their traces based on the timestamp of each cache access. We divide the memory address space such that each application has access to half of the total memory size. To have all data reside in the memory, we increase the memory size to 512 Mbytes divided among eight chips. We maintain the L3 (in the base case) and near-memory (in PA-CDRAM) caches at the same capacity.

Table 4 shows the results of running pairs of application traces on our simulator. We report energy values normalized to the base case. Delay results are irrelevant since we are running timestamped traces rather than the actual execution of applications. As expected, running multiple applications simultaneously increases the number of cache misses compared to the sum of cache misses when running each application individually. This observation is true for both PA-CDRAM and the base case. Although the number of misses in near-memory caches is higher than that in the single-processor case, the total number of near-memory cache misses is still lower than L3 misses. The lower misses in PA-CDRAM is because PA-CDRAM originally achieves much lower miss rates than the base case when running a single application, as shown in Fig. 9. In Table 4, most application pairs experience lower energy consumption by using the PA-CDRAM memory than when using the base case. In the base case, access to multiple DRAM chips (from the two applications) increases the number of active chips and, hence, increases the DRAM-core static energy. However, less energy impact is noticed in PA-CDRAM due to the interleaved data allocation and the immediate deactivation of the DRAM core. The average energy savings across all application pairs is 20 percent.

6.6 Sensitivity to Design Parameters

We investigate the effect of varying the different system parameters on the energy and performance of PA-CDRAM with respect to the traditional memory. We vary the following parameters: 1) the cache capacity for both the L3 and the near-memory caches to study the effect of the capacity misses on the system, 2) the CPU frequency to study the effect of improving the average memory access

8. This is a worse case behavior since usually the cache will not be completely flushed.

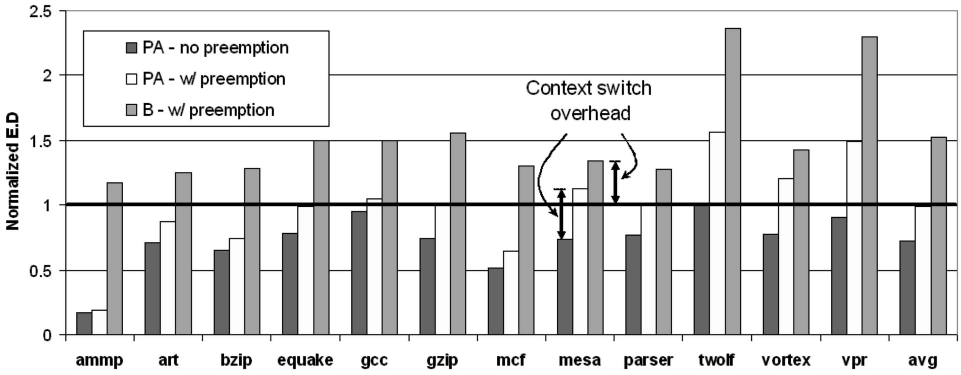


Fig. 12. Energy-delay product with and without preemption, normalized to the base case without preemption.

TABLE 4
PA-CDRAM Energy Consumption Normalized to the Base Case

	ammp	art	bzip	equake	gcc	gzip	mcf	mesa	parser	twolf	vortex	vpr
ammp	1.82	0.94	0.14	0.16	0.66	0.14	1.10	0.14	0.13	0.73	0.17	0.80
art	0.94	1.48	1.31	1.30	1.24	1.23	1.36	1.18	1.26	1.35	1.19	1.32
bzip	0.14	1.31	0.63	0.80	0.65	0.66	1.01	0.72	0.69	0.75	0.71	0.70
equake	0.16	1.30	0.80	1.00	0.69	0.74	1.06	0.73	0.75	0.82	0.74	0.81
gcc	0.66	1.24	0.65	0.69	0.63	0.66	1.01	0.61	0.66	0.64	0.66	0.70
gzip	0.14	1.23	0.66	0.74	0.66	0.72	1.02	0.67	0.72	0.71	0.70	0.70
mcf	1.10	1.36	1.01	1.06	1.01	1.02	1.06	0.79	0.81	0.92	0.90	0.95
mesa	0.14	1.18	0.72	0.73	0.61	0.67	0.79	0.73	0.70	0.60	0.73	0.63
parser	0.13	1.26	0.69	0.75	0.66	0.72	0.81	0.70	0.75	0.72	0.75	0.74
twolf	0.73	1.35	0.75	0.82	0.64	0.71	0.92	0.60	0.72	0.77	0.66	0.89
vortex	0.17	1.19	0.71	0.74	0.66	0.70	0.90	0.73	0.75	0.66	0.71	0.70
vpr	0.80	1.32	0.70	0.81	0.70	0.70	0.95	0.63	0.74	0.89	0.70	0.97

time on the total performance, and 3) the slowdown experienced by the logic embedded in the DRAM chip.

6.6.1 Effect of Varying the Cache Size

Varying the cache size affects both the miss rate and the cache access costs (latency and energy). Fig. 13 shows the average delay, energy, and energy-delay product of PA-CDRAM while varying the total cache size from 512 Kbytes to 4 Mbytes (excluding L1 and L2). The results are normalized to the base case with an L3 cache of corresponding size. Note that, although the small cache sizes tested (512 Kbytes and 1 Mbyte) are too small for a typical L3 cache, we test at those sizes to demonstrate the trend. Increasing the cache size reduces the L3 (or near-memory) miss rates; thus, better performance is achieved in both cases. When the miss rate is reduced, lower accesses to the DRAM core occur and, accordingly, the relative savings in delay of PA-CDRAM to the base case is lower when the cache size increases.

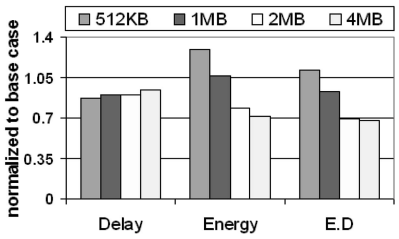


Fig. 13. The effect of varying the cache size on execution time, energy, and energy-delay product normalized to the base case.

With respect to energy consumption, the effect of varying the cache capacity is not as trivial. The general trend is that increasing the cache size reduces the capacity misses; thus, less energy is consumed due to fewer cache replacements and DRAM accesses. However, when increasing the cache size, the cache energy per hit access increases, causing an increase in the total memory energy consumption. Comparing the energy consumption of the base case versus PA-CDRAM, we find that applications are divided into two groups: 1) where PA-CDRAM consumes less energy at all cache sizes, as in *bzip*, *gzip*, *mesa*, *parser*, and *vortex*, and 2) where PA-CDRAM consumes less energy only at large cache sizes. For the first group, PA-CDRAM performs better due to accessing smaller individual caches (with lower per-access energy). For the second group, at the small cache sizes tested (512 Kbytes and 1 Mbyte), too many near-memory cache replacements take place due to the limited number of blocks ($\frac{512\ Kbytes}{8 \cdot 512\ bytes} = 128$ blocks) per individual near-memory cache. Results for individual applications can be found in [26].

6.6.2 Effect of Varying the CPU Frequency

Increasing the CPU frequency increases the speed gap between the memory and the CPU, thus increasing the total execution cycles of an application. Fig. 14 shows the energy-delay product for PA-CDRAM at different clock rates normalized to the base case at 2 GHz. We choose to fix the frequency at the base case to demonstrate the effect on the energy consumption. In our experiments, most of the applications exhibit minimal variation in execution times.

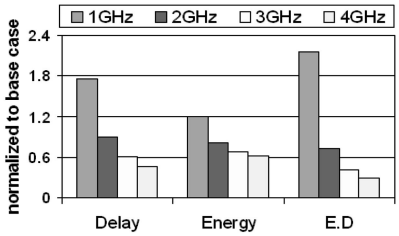


Fig. 14. The effect of varying CPU frequency for the PA-CDRAM normalized to the base case on execution time, energy, and energy-delay product.

These applications are mainly CPU bound, where the processor is able to mask most of the L2 miss stalls. Thus, the relative benefit of PA-CDRAM on delay is negligible compared to the base case (as described in Section 6.3). In memory-bound applications (*ammp*, *art*, and *mcf*), increasing the CPU frequency compounded with the larger average memory access times in the base case results in the base case suffering from significantly larger delays than PA-CDRAM. That is, at higher CPU frequency, the value of μ increases significantly in these memory-intensive applications, making PA-CDRAM a more efficient solution.

From the energy perspective, increasing the CPU frequency reduces the application's execution time and, thus, the duration of the idle periods in the DRAM. This reduces the consumption of the static energy in the DRAM core. In contrast, the dynamic energy in the DRAM and caches is not affected by the processor frequency as they are mainly dependent on the size of the transfer rather than the frequency. In modern processors, although higher processor frequencies—with all other factors unchanged—reduce the memory's total energy, the memory's energy consumption can increase due to other factors, like larger memory capacities and higher memory traffic in modern processors. Fig. 14 shows the effect on the energy-delay product.

6.6.3 Effect of Logic Slowdown

To show the effect of the embedded logic manufacturing technology on the overall performance of the system using PA-CDRAM, we vary the logic slowdown factor of the near-memory data array (assuming a slower distributed cache controller). We add a delay penalty to the overall cache access delay obtained from Cacti. This penalty ranges from 0 percent (fast SRAM) to 50 percent. The execution time is normalized to the PA-CDRAM case, where there is no performance penalty for the logic cells. In Fig. 15, we notice

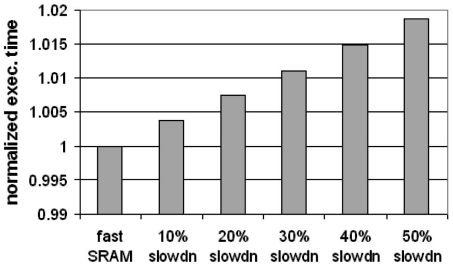


Fig. 15. The effect of the logic slowdown in the near-memory cache on the total execution time.

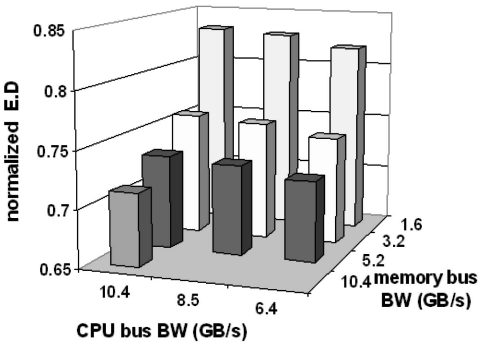


Fig. 16. The effect of different CPU and memory bus bandwidths.

that the performance degradation in the near-memory-cache-controller design is relatively insignificant. Even when compared to a centralized-cache-controller design (not shown here), the overall slowdown in performance over fast SRAM is 1.5 percent on the average (up to 6.3 percent) with a 50 percent slowdown penalty. This result shows that delays resulting from manufacturing embedded logic in DRAM chips do not represent any critical delays on the overall system when using the PA-CDRAM memory.

6.6.4 Effect of CPU and Memory Bus Bandwidths

External buses can pose a performance bottleneck for accessing external caches and memory. Our earlier results assume ideal external buses with infinite bandwidth to avoid the impact of such a bottleneck. To study the impact of bus latencies on the overall performance, we limit the bandwidths of the CPU and memory buses in the base case and PA-CDRAM. The CPU bus bandwidth is limited by the bus speed, bus width, and data rate (single, dual, or quad). We assume a 64 bits wide CPU bus. Intel uses quad-rate (sends 4 bits/cycle) buses that operate at 200, 266, or 333 MHz [27]. Hence, a CPU bus bandwidth can range from 6.25 to 10.4 Gbytes/s ($\frac{64}{8} * bus\_freq * 4$). The limited bandwidth increases the latency of filling an L2 block between 13 and 21 cycles in our setting. The RDRAM memory bus provides a data bandwidth between 1.6 and 12.4 Gbytes/s [28]. This range of bus bandwidth increases the latency of reading an L2 block from the memory by 11 to 80 cycles. It is typical for the CPU bus to be faster than the memory bus.

Fig. 16 shows the normalized energy-delay product using a mixture of CPU and memory bus bandwidths. The figure shows that energy-delay saving is more sensitive to memory bus latency since near-memory caches use this bus to fill the L2 cache. Although it is faster to transfer an L2 block from an L3 cache than from a near-memory cache (faster CPU buses), an L3 miss has a much higher latency than a near-memory cache miss. Hence, PA-CDRAM can improve performance even with a limited bus bandwidth. In general, the larger the bandwidth of both buses, the higher the savings PA-CDRAM can achieve.

7 Discussion

The manufacture of PA-CDRAM can benefit from new advances in eDRAM technology. New fabrication technologies are successful in manufacturing eDRAM cells with a

significantly smaller size than traditional SRAM cells. Each eDRAM cell consists of a single transistor and a single capacitor (1T1C cell) as opposed to six transistors in the case of an SRAM cell. The eDRAM from NEC and 1T-SRAM from Mosys are examples of technologies that apply the 1T1C concept. Due to the smaller cell sizes, higher densities can be achieved. Hence, a larger capacity in a smaller chip area is realizable. On the other hand, the access latency of eDRAM is slightly higher, as mentioned in Section 2.

The smaller form factor of 1T1C reduces both active and leakage energy. The relatively large SRAM cell size contains longer metal lines, which create higher capacitance than the ones found in a 1T1C cell. Lower cell capacitance translates to lower current draw and power consumption, hence reducing the cell's active power. In addition, a 1T1C cell contains one leakage path, as opposed to four leakage paths in an SRAM cell. Hence, lower leakage current is drawn in an eDRAM cell. Furthermore, the eDRAM-optimized refresh current is significantly less than the leakage current of an equivalent 6T memory array [29].

8 Conclusion

In this paper, we explore the energy efficiency of *near-memory caches* rather than conventional cache hierarchies, where most of the cache capacity is allotted "closer" to the CPU. PA-CDRAM can be used as an alternative to traditional power-aware memories to conserve energy and improve performance. PA-CDRAM reduces the memory's energy consumption by 1) bringing the cache closer to the memory to exploit the high memory bandwidth, 2) distributing the external cache into smaller caches that have a low access energy and latency, and 3) increasing the DRAM-core idle periods due to the low miss rates of near-memory caches. Three main parameters affect the degree of benefit of PA-CDRAM over the traditional memory: higher L2 misses, lower η (miss rate ratio), and higher μ (memory stalls) all lead to higher energy and delay benefits obtained from PA-CDRAM. Compared to the traditional memory using a time-out power management, PA-CDRAM saves up to 76 percent energy consumption (19 percent on the average). Moreover, PA-CDRAM reduces the energy-delay product by up to 84 percent (28 percent on the average), where the highest gains are for memory-intensive applications and for applications with relatively high spatial locality.

Our evaluation shows that PA-CDRAM is more energy-delay efficient than allocating a large fraction of the total cache capacity to an on-chip L2, on-chip L3, or off-chip L3. In a multitasking environment, PA-CDRAM can lower the context switch overhead of cache invalidation through simultaneous flushing of dirty blocks in all on-memory caches. With the increase in cache capacity and CPU frequency in current and future processors, the energy savings in PA-CDRAM is expected to grow higher.

References

[1] V. Freeh, D. Lowenthal, F. Pan, N. Kappiah, and R. Springer, "Exploring the Energy-Time Tradeoff in MPI Programs on a Power-Scalable Cluster," *Proc. 19th Int'l Parallel and Distributed Processing Symp. (IPDPS '05)*, 2005.

[2] O. Celebican, T. Simunic, and V. Mooney, "Energy Estimation of Peripheral Devices in Embedded Systems," *Proc. 14th ACM Great Lakes Symp. VLSI (GLSVLSI '04)*, pp. 430-435, 2004.

[3] D. Elliott, W. Snelgrove, and M. Stumm, "Computational RAM: A Memory-SIMD Hybrid and Its Application to DSP," *Proc. IEEE Custom Integrated Circuits Conf. (CICC '92)*, pp. 30.6.1-30.6.4, 1992.

[4] W. Hsu and J. Smith, "Performance of Cached DRAM Organizations in Vector Supercomputers," *Proc. 20th Ann. Int'l Symp. Computer Architecture (ISCA '93)*, pp. 327-336, 1993.

[5] I. Kadayif, T. Chinoda, M. Kandemir, N. Vijaykirsnan, M.J. Irwin, and A. Sivasubramaniam, "vEC: Virtual Energy Counters," *Proc. ACM SIGPLAN-SIGSOFT Workshop Program Analysis for Software Tools and Eng. (PASTE '01)*, pp. 28-31, 2001.

[6] D. Keitel-Schulz and N. Wehn, "Embedded DRAM Development: Technology, Physical Design, and Application Issues," *IEEE Design and Test of Computers*, vol. 18, no. 3, pp. 7-15, July-Sept. 2001.

[7] "NEC Embedded DRAM," http://www.necelam.com/edram90/, 2005.

[8] S. Tomashot, "IBM Embedded DRAM Approach," http://www-306.ibm.com/chips/techlib/techlib.nsf/products/Embedded_DRAM, 2003.

[9] B. Davis, "Modern Dram Architectures," PhD dissertation, Univ. of Michigan, Ann Arbor, 2000.

[10] R. Koganti and G. Kedem, "WCDRAM: A Fully Associative Integrated Cached-DRAM with Wide Cache Lines," technical report, Dept. of Computer Science, Duke Univ., 1997.

[11] A. Hegde, N. Vijaykrishnan, M. Kandemir, and M. Irwin, "VL-CDRAM: Variable Line Sized Cached DRAMs," *Proc. First IEEE/ACM/IFIP Int'l Symp. Hardware/Software Codesign and System Synthesis (CODES+ISSS '03)*, pp. 132-137, 2003.

[12] Z. Zhang, Z. Zhu, and X. Zhang, "Cached DRAM for ILP Processor Memory Access Latency Reduction," *IEEE Micro*, vol. 21, no. 4, pp. 22-32, July/Aug. 2001.

[13] Rambus, "Products Data Sheets," http://www.rambus.com/products/rdram/documentation, 2005.

[14] A. Lebeck, X. Fan, H. Zeng, and C. Ellis, "Power Aware Page Allocation," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, pp. 105-116, 2000.

[15] H. Huang, P. Pillai, and K. Shin, "Design and Implementation of Power-Aware Virtual Memory," *Proc. Usenix Ann. Technical Conf.*, pp. 57-70, citeseer.ist.psu.edu/582414.html, 2003.

[16] V. Delaluz and M.J. Irwin, "DRAM Energy Management Using Software and Hardware Directed Power Mode Control," *Proc. Seventh Int'l Symp. High-Performance Computer Architecture (HPCA '01)*, pp. 159-169, 2001.

[17] P. Shivakumar and N. Jouppi, "Cacti 3.0: An Integrated Cache Timing, Power, and Area Model," Technical Report 2001.2, Compaq Research Laboratories, 2001.

[18] N. AbouGhazaleh, B. Childers, D. Mossé, and R. Melhem, "Near-Memory Caching for Improved Energy Consumption," *Proc. 23rd Int'l Conf. Computer Design (ICCD '05)*, pp. 105-110, 2005.

[19] SimpleScalar, "Architecture Simulator," http://www.simplescalar.com, 2004.

[20] N. AbouGhazaleh, B. Childers, D. Mossé, and R. Melhem, "Energy Conservation in Memory Hierarchies Using Power-Aware Cached-DRAM," *Proc. Dagstuhl Seminar Power-Aware Computing Systems*, Apr. 2005.

[21] Y. Zhong, S. Dropsho, and C. Ding, "Miss Rate Prediction across All Program Inputs," *Proc. 12th Int'l Conf. Parallel Architectures and Compilation Techniques (PACT '03)*, pp. 79-90, 2003.

[22] M. Gries and A. Romer, "SDRAM and RDRAM Modeling for Simplescalar Simulator," http://www.tik.ee.ethz.ch/ip3/software/simplescalar_mem_model.html, 2004.

[23] W. Lin, S. Reinhardt, and D. Burger, "Reducing DRAM Latencies with an Integrated Memory Hierarchy Design," *Proc. Seventh Int'l Symp. High-Performance Computer Architecture (HPCA '01)*, pp. 301-312, 2001.

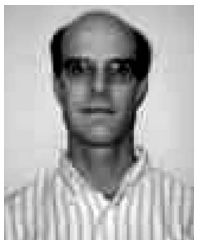
[24] Pentium, "Intel Pentium 4 EE Processor," http://www.intel.com, 2003.

[25] Y. Aghaghiri, F. Fallah, and M. Pedram, "Transition Reduction in Memory Buses Using Sector-Based Encoding Techniques," *IEEE Trans. Computer-Aided Design*, vol. 23, no. 8, pp. 1164-1174, 2004.

[26] N. AbouGhazaleh, B. Childers, D. Mossé, and R. Melhem, "Energy Conservation in Memory Hierarchies Using Power-Aware Cached-DRAM," Technical Report TR-05-123, Dept. of Computer Science, Univ. of Pittsburgh, 2005.

[27] Intel, "Desktop Chipset Datasheets," http://www.intel.com/products/desktop/chipsets/index.htm, 2006.
[28] RDRAM, "RDRAM Technology Summary," http://www.rambus.com/assets/documents/products/RDRAMTechnology/Summary_091905.pdf, 2005.
[29] J. Bond, "Memory Amnesia Could Hurt Low-Power Design," http://www.commsdesign.com/design_corner/OEG20030730S0018, 2003.

Nevine AbouGhazaleh received the BE and MS degrees in computer engineering from the Arab Academy for Science and Technology in 1996 and 1999, respectively, and the MS degree in computer science from the University of Pittsburgh in 2003. She is a PhD student at the University of Pittsburgh. Her research interests include computer architecture, operating systems, and embedded systems. Currently, she is researching power management in real-time systems. She is a student member of the IEEE. She is a recipient of Google's Anita Borg Fellowship and a Josephine DeKarman Fellowship.



Bruce R. Childers received the BS degree in computer science from the College of William and Mary in 1991 and the PhD degree in computer science from the University of Virginia in 2000. He is an assistant professor in the Department of Computer Science at the University of Pittsburgh. His research interests include computer architecture, compilers and software development tools, and embedded systems. Currently, he is researching continuous compilation, power-aware computer architecture for small and portable systems, and compiler optimization for embedded systems. He is a member of the IEEE and the IEEE Computer Society.

Daniel Mossé received the BS degree in mathematics from the University of Brasilia in 1986 and the MS and PhD degrees in computer science from the University of Maryland in 1990 and 1993, respectively. He has been a professor at the University of Pittsburgh since 1992. His research interests include fault-tolerant and real-time systems, as well as networking. The current major thrust of his research is real-time systems, power management issues, and networks (wireless and security). Typically funded by the US National Science Foundation and the US Defense Advanced Research Projects Agency, his projects combine theoretical results and actual implementations. He was an associate editor of the *IEEE Transactions on Computers* and is currently on the editorial board of the Kluwer *Journal of Real-Time Systems*. He has served on program committees (PCs) and as a PC chair for most major IEEE and ACM-sponsored real-time conferences. He is a member of the IEEE and the IEEE Computer Society.

Rami G. Melhem received the BE degree in electrical engineering from Cairo University in 1976, the MA degree in mathematics and the MS degree in computer science from the University of Pittsburgh in 1981, and the PhD degree in computer science from the University of Pittsburgh in 1983. He was an assistant professor at Purdue University prior to joining the faculty of the University of Pittsburgh in 1986, where he is currently a professor of computer science and electrical engineering and the chair of the Computer Science Department. His research interests include real-time and fault-tolerant systems, optical interconnection networks, high-performance computing, and parallel computer architectures. He was on the editorial board of the *IEEE Transactions on Computers* and the *IEEE Transactions on Parallel and Distributed Systems*. He is the editor for the Kluwer/Plenum book series on computer science and is on the editorial board of the *IEEE Computer Architecture Letters* and the *Journal of Parallel and Distributed Computing*. He has served on the program committees of numerous conferences and workshops and was the general chair of the Third International Conference on Massively Parallel Processing Using Optical Interconnections (MPPOI '96). He serves on the advisory boards of the IEEE Technical Committees on Parallel Processing and on Computer Architecture. He is a fellow of the IEEE and a member of the IEEE Computer Society and the ACM.