

Energy Conservation in Memory Hierarchies using Power-Aware Cached-DRAM *

Nevine AbouGhazaleh, Bruce Childers, Daniel Mossé, Rami Melhem
(*nevine, childers, mosse, melhem*)@cs.pitt.edu
Department of Computer Science
University of Pittsburgh
Technical Report : TR-05-123

Abstract

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 new memory organization, called Power-Aware Cached-DRAM (PA-CDRAM), that integrates a moderately sized cache directly into a memory module. We use this cache to turn a memory bank off immediately after a memory access to reduce power consumption. While other work has used CDRAM to improve memory performance, we modify CDRAM to reduce energy consumption. In this paper, we describe our memory organization and show how to incorporate it into Rambus memory. We evaluate the approach using a cycle accurate processor and memory simulator. Our results show that PA-CDRAM achieves up to an 84% (28% on average) improvement in the energy-delay product and up to a 76% (19% on average) savings in energy alone when compared to a time-out power management technique.

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, which requires more cooling resources, and has a higher system management overhead. One of the major energy consumers in computing systems is the memory subsystem. With the increasing variety of applications that require high memory capacity, a significant amount of energy is expected to be consumed accessing data, which motivates the need for memory energy management schemes.

The speed gap between the processor and the memory limits the overall system performance [1]. The main contributor to this problem is the large memory access time and the limited transfer bandwidth between the memory and the CPU. Memory architectures aim at decreasing the access time by using faster clocks and increasing the level of concurrency in data accesses. However, the bus transmission speed is becoming a significant bottleneck limiting the memory system performance [2].

Memory has a huge internal bandwidth compared to its external bus bandwidth [3]. To exploit the wide internal bus, *cached DRAM* (CDRAM) adds an SRAM cache to the DRAM array on the memory chip [4]. The on-memory cache acts as an extra memory hierarchy level, whose fast access time

*This work has been supported by the Defense Advanced Research Projects Agency under contract F33615-00C-1736. Bruce Childers was supported in part by the National Science Foundation, Next Generation Software Program, award CNS-0305198

improves the average memory access time and potentially improves system performance, provided that the on-memory cache is appropriately configured.

In this paper, we explore the benefit of having a near-to-memory cache with respect to its energy consumption. We optimize CDRAM by integrating a moderately sized cache within the chip boundary of a power-aware multi-banked memory. We call this organization *power-aware cached DRAM (PA-CDRAM)*. In addition to improving performance, PA-CDRAM can significantly reduce the energy consumption in external memories by powering off the DRAM banks after the data is transferred to the on-memory cache. We also show that PA-CDRAM is more energy efficient than using an extra level of off-chip cache of equal size. The contributions of this paper consists of the following.

1. The integration of a power management policy in CDRAM to reduce energy consumption in the memory system. This reduction in the memory's energy consumption should come without affecting performance gains provided by CDRAM.
2. A design approach to solve two challenges for managing energy consumption in CDRAM: the issue of cache configuration (e.g., deciding the on-memory cache block size and associativity) and the appropriate DRAM power management (e.g., what the timeout values should be).
3. An implementation of PA-CDRAM using current memory technology. We describe how to integrate the on-memory cache in a Rambus chip (to exploit DRAM power management support), and describe its modes of operation (messages, protocols, etc).
4. An evaluation study through simulation that shows the advantages of replacing an off-chip L3 cache with an on-memory PA-CDRAM cache of equal size.

The remainder of this paper is organized as follows. Section 2 gives an overview of the CDRAM architecture and current DRAM memory power management techniques. Section 3 describes the challenges for designing power efficient CDRAM and the design approaches used in PA-CDRAM to overcome them. Section 4 describes an implementation of PA-CDRAM using the Rambus architecture, its modes of operation, and provides an analysis of its energy consumption. An extensive evaluation with respect to delays and energy savings is presented in Section 5. Section 6 reviews closely related work, and Section 7 concludes the paper.

2 Background: Cached DRAM and Memory Power Management

Embedded logic/DRAM Integrating DRAM and logic cells on the same chip has been an attractive solution to achieve both high performance (from logic cells) and high memory density (from DRAM cells) on the same chip. This integration reduces the need to experience high latency from going off-chip. Currently, manufactured chips with embedded DRAM/logic are mainly used in applications like computer graphics, networking, and handheld devices [5]. Based on the fabrication technology (either DRAM-based or logic-based), some degradation to the speed/density of the logic/DRAM cells may occur, respectively. For example, in DRAM-based chips, logic cells can be slower than typical logic cells in the range of 20% to 35% [5]. However, emerging fabrication technologies aim to overcome these penalties. For example, NEC's embedded DRAM chips offer DRAM-like density with SRAM-like performance [6], and IBM's third generation embedded DRAM support two embedded DRAM families: high density and high performance [7].

Cached DRAM To decrease the memory average access time, Hsu et al. proposed to integrate a small SRAM cache within the memory chip next to the DRAM-core, as shown in Figure 1 [4]. This organization is called a CDRAM and serves as the basis for our PA-CDRAM. Due to the large

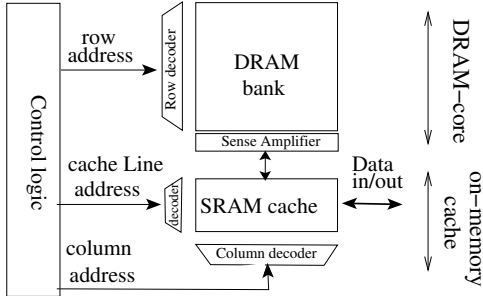


Figure 1: Functional block diagram of a CDRAM chip

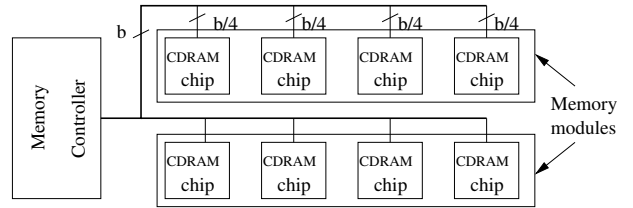


Figure 2: System model for a memory with two modules and eight CDRAM chips (b is the cache block size).

memory internal bandwidth, large chunks of data can be transferred between the DRAM-core and the on-memory cache with low latency. Average memory access time is improved by accessing the data through the fast cache rather than the slower DRAM. The cache controller can be co-located with the memory controller [8] or integrated on-chip with the processor core [9]. For each memory access, a tag comparison takes place in the cache’s tag array. If a CDRAM cache hit occurs, data is directly accessed from the CDRAM cache data array. If a CDRAM cache miss is detected, a cache block is read from the DRAM-core. Data is written to the CDRAM cache and then sent through the data bus. To avoid data inconsistency, data access is always done through the CDRAM cache [1]. CDRAM is typically implemented using Synchronous DRAM, *SDRAM*, chips [1]. Each memory chip contains one or more DRAM banks, and each bank has an associative cache. Memory chips are connected to the memory controller through the memory bus as shown in Figure 2.

Memory Power Management Current memory technologies, such as SDRAM and Rambus, allow for managing energy consumption by setting the DRAM chips to one of two or more power states [10]. A simple form of power management is to alternate between two power states: active and inactive (or sleep). A read/write transaction can only occur while the chip is active; that is, *all* the components (e.g., row decoder, clock, etc) in the memory chip are powered up to service the request. Some of these components can be powered down when the memory is not in use to save energy (sleep state). A chip in the sleep state has to be activated before any read/write operation takes place in the chip. Delay penalties result from transitioning between power states before servicing a memory request. A popular way of scheduling a chip’s power state transitions is through a *timeout* policy. When an inactivity period exceeds a time threshold defined by the timeout policy, the chip is transitioned to a lower power state. Different power saving states can be generated by varying the number of memory components that are kept off.

3 PA-CDRAM

CDRAM was originally proposed to improve system performance; however, it was not designed as a replacement for power-aware memory. Yet, CDRAM can serve as the basis for power-efficient memory by using the CDRAM’s on-chip cache to reduce memory accesses to the DRAM-core; thus, increasing its *idleness*. We use this caching to allow the DRAM-core to be quickly transitioned to a low power state for longer time periods. In using CDRAM as the basis for PA-CDRAM, there are two main challenges that must be addressed: (1) how to configure the DRAM-core’s power management policy, assuming the use of multiple power states; and (2) how to configure the on-memory cache to balance energy and performance. We describe each of these challenges and how we address them below.

3.1 DRAM-core power management

To optimize CDRAM operation for energy savings, we need to minimize the number of active chips at any time as well as the duration of active periods. Bounded by the external memory bandwidth, CDRAM (using SDRAM) interleaves data blocks across multiple chips. At each memory request, n chips within a memory module are activated and each chip provides $1/n$ of the block size requested. This interleaving of data blocks entails overhead in address decoding, and bit-/word- line activation in more than one chip. Thus, a more energy efficient organization would utilize chips that offer (a) an independent access to the DRAM-array and (b) full bandwidth utilization of the system bus to avoid reducing the memory throughput. The energy overhead is reduced by activating a single chip at each access, and performing fewer address decoding operations in the target chip during data retrieval.

With an on-memory cache, we propose to apply aggressive power management in the DRAM-core to reduce the duration of active periods. During a chip’s idle time, the memory controller can immediately transition the DRAM-core to the sleep state after servicing all outstanding DRAM-core access requests. This is equivalent to the use of a timeout policy with an idle threshold of zero seconds. 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 our PA-CDRAM design, we avoid this problem by choosing the on-memory cache configuration such that it increases hit rate while reducing the DRAM-core’s energy consumption. When most data requests are serviced as cache hits in the on-memory cache, the inter-arrival time between requests that reach the DRAM-core increases, making it cost effective to immediately deactivate banks after servicing outstanding requests. We choose to keep the on-memory cache in the active state all the time to avoid delays due to on-demand activation of the cache at each request.

3.2 On-memory cache configuration

The on-memory cache miss rate should be kept at a minimum as it directly influences the memory energy consumption (in addition to performance). The higher the miss rate, the more energy is consumed in memory due to more activity taking place in the DRAM-core in terms of transitioning from the sleep to the active state, performing address decoding, and transferring data. For a given cache size and a fixed number of cache subbanks, the two factors affecting the cache energy consumption and access latency are the *associativity* and the *block size* [11]. We used Cacti-3.0 [11] to study a 256KB cache with respect to latency and energy consumption. Figure 3 shows the trend we observed: compared to n -way associative caches, energy and latency cost *per access* in a fully associative cache decreases at large block sizes, in contrast to small block sizes where full associativity is relatively expensive (in latency and energy). Although we only show the per-access metric for a 256 KB cache, other cache sizes with similar $\frac{\text{cachesize}}{\text{blocksize}}$ ratios follow the same pattern.

To further examine this issue, we studied a memory with eight CDRAM chips. Each chip contains a cache of moderate size, 256 KB with 512KB cache blocks. Figures 4 and 5 show the effects of varying the associativity and the block size on the on-memory cache average miss rate of the eight chips. We use these results to motivate the selection of our on-memory cache configuration. The miss rates values are collected from the SimpleScalar architecture simulator [12] and correspond to a set of applications from the SPEC2000 (int and fp) benchmarks.

Figure 4 shows that the higher the associativity, the lower the average miss rate across all caches until it reaches saturation. Note that, in most of the tested applications, the miss rates of the 8-way set associative caches are as low as the fully associative caches. However, from Figure 3, we see that the per-access latency and energy consumption of a cache with large block sizes (512B and larger) are

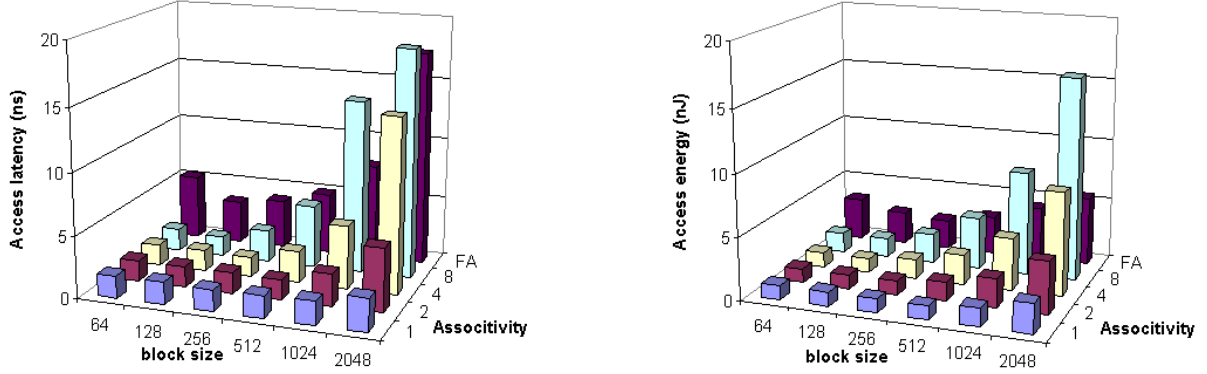


Figure 3: The effect of varying the cache block size and associativity on the cache access latency and energy consumption for a 256KB on-memory cache.

lower in the fully associative cache than the 8-way cache. From these results, we decided to use fully associative caches with relatively large block sizes as our PA-CDRAM cache configuration.

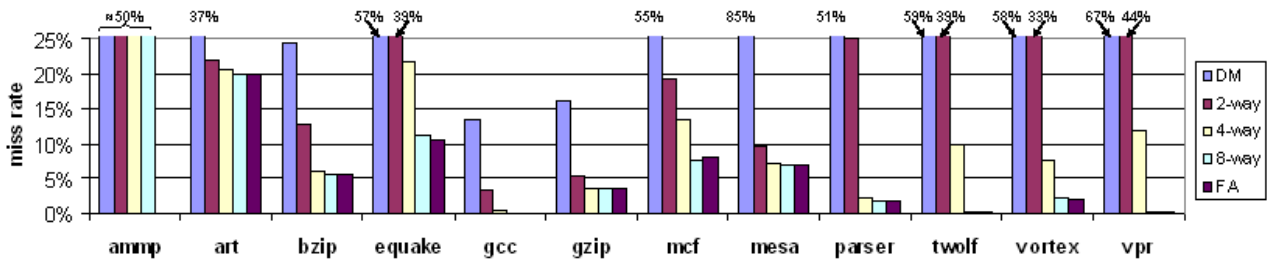


Figure 4: The effect of varying the associativity on the miss rate in caches with 512B blocks.

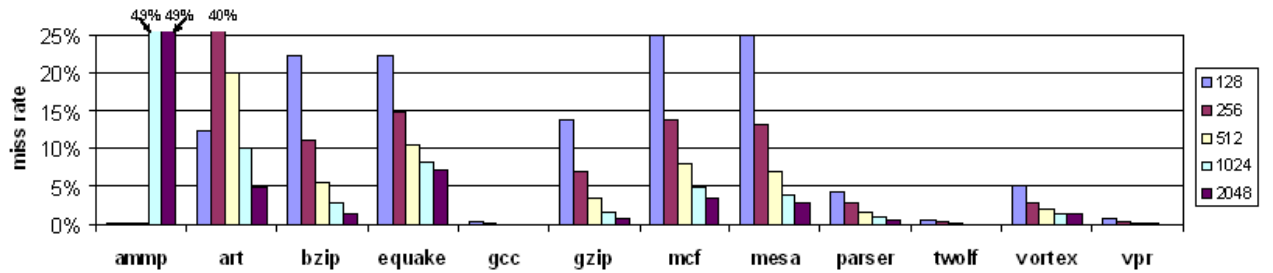


Figure 5: The effect of varying the cache block size on the miss rate in caches with 512B blocks.

The performance of on-chip and external caches are constrained by the width of the system bus. To retrieve data from memory with small latencies, the size of the requested blocks are typically small (64 bytes for L2 and 128 bytes for optional L3 caches in some Pentium4 processors [13]). In contrast, the on-memory caches do not have such a constraint because of the large internal memory bandwidth. Thus, large blocks of data can be transferred to the on-memory caches with low latencies, which favors the use of large block sizes in PA-CDRAM. Figure 5 shows that on-memory caches with larger blocks have lower miss rates than caches with smaller blocks. The larger the unit of transfer is, the fewer accesses to the DRAM-core occur. Hence, a larger block size lowers the average memory access time.

From the energy perspective, Figure 3 shows that for a fixed way-associative cache, accessing large blocks consumes more energy than accessing smaller blocks. However, the energy-per-byte for each access is lower in accessing large blocks. Although the miss rate and the per-byte cache access energy are reduced for large block sizes, there is a potential increase in the memory energy consumption if many unnecessary bytes are transferred between the on-memory cache and the DRAM-core. To achieve a balance between obtaining low miss rates and avoiding excessive memory traffic, we select a relatively large block size around 512B. This is different from the proposal of Koganti et al. [9] for using block sizes that range from 4KB to 8KB. The difference in our evaluation is that we take into account the energy consumption and delay for accessing the caches (L3 or on-memory).

4 PA-CDRAM implementation

We apply the PA-CDRAM approach to Rambus technology to show its potential for reducing energy **and** improving performance. We use Rambus because it supports different power modes and permits controlling memory chips independently. In this section, we describe the needed architectural and operational modifications for integrating the on-memory cache into Rambus chips. We also analyze energy consumption of PA-CDRAM.

4.1 Background: Rambus technology

Rambus [10] is a family of DRAM architectures where the memory bus, the *Rambus Channel*, can operate at high frequencies (800-1600 MHz). A channel consists of four busses: a row bus (3 bits), a column bus (5 bits), a data bus (16 bits) and a control register bus (4 bits). The row, column, and register busses form the *control bus*. Because the control bus has parallel row and column busses, Rambus achieves high bandwidth by allowing the pipelining between row and column commands while maintaining a sustainable data transfer in the data bus. Communication between the memory controller and the memory chips is done by sending command packets through the Rambus channel to be decoded by control logic in the Rambus chips, *RDRAM*. As opposed to SDRAM technologies, a single RDRAM chip can service the entire memory request rather than distributing the request to several SDRAM chips.

An RDRAM chip contains 16 or 32 banks. Each row in a bank is 2KB wide and is connected to two sense amps. The upper half bits of a bank’s row are latched on to one of the bank’s sense amps while the other half is latched on to the second sense amp. The sense amps are divided into two groups, and each group sends data on one of two 1-byte busses. The aggregate of these two internal busses form a data bus of width 16 bits.

RDRAM chips dynamically transition between power states to reduce the chips’ energy consumption. A chip can be set in one of four different power states. The states, in descending order of their power consumption, are: *active*, *standby*, *nap* and *powerdown*. 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.

4.2 PA-CDRAM architecture

Our PA-CDRAM design modifies the original RDRAM design for power efficiency. Beside the addition of the on-memory cache, some alterations are needed in the main components of the RDRAM, namely: the DRAM-core, the control logic, and the memory controller.

On-memory cache: We add a fully associative cache (depicted as dark blocks in Figure 6) 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 busses (DQA and DQB). Each cache section stores half of each cache block. We keep the write buffers in the original RDRAM before the sense amplifiers. The write buffers are used to store a replaced dirty block from the on-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. That is, the on-memory cache is accessible even when the DRAM is in the nap state.

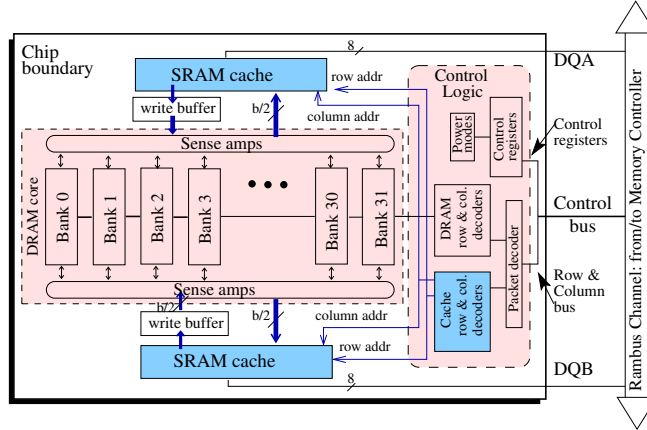


Figure 6: Functional block diagram of a PA-CDRAM.

DRAM-core: To accommodate the large transfer sizes between the DRAM-core and the on-memory cache, wider internal data busses are required to connect the sense amplifiers with the on-memory cache. The width of each bus connecting the DRAM and on-memory caches is $b/2$ bits, where b is the size of a cache block. The busses 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. The new cache commands indicate whether the access is a hit/miss in the on-memory cache, and whether replacement is needed. The new commands are defined as follows:

- `cache_read_addr`, *CRA*, a 24 bit packet that uses the row bus: `opcodeL2e(4)`, `chip_id(4)`, `block_addr(15)`.
- `cache_write_addr`, *CWA*, a 24 bit packet that uses the row bus: `opcode(4)`, `chip_id(4)`, `block_addr(15)`.
- `cache_tag_test`, *CTT*, a 40 bit packet that uses the column bus : `opcode(4)`, `chip_id(4)`, `block_addr(15)`, `tag_flag(1)`, `dirty_flag(1)`, `replaced_addr(15)`

The semantics of each command is given in Table 1. The cache commands use the row and column control busses to increase memory pipelining. More details are described in the next section.

Memory and cache controllers: The on-memory cache controller can be integrated with either the PA-CDRAM chip or the memory controller chip. Integrating the cache controller with PA-CDRAM has the advantage of being compatible with the existing Rambus memory controller. However, if the chip area is a concern, then having a single cache controller for all PA-CDRAM caches can be more area efficient than duplicating it into each PA-CDRAM chip. In this work, we extend the memory

Table 1: PA-CDRAM cache commands send across the control bus

command	description (see Section 4.3)
cache_read_addr	Issues a read request for <code>block_addr</code> ; data is sent on the data bus only if the chip receives the <i>hit</i> as result of the tag comparison
cache_write_addr	Issues a write request for <code>block_addr</code> (data is sent later on the data bus).
cache_tag_test	Sends the tag comparison result (<code>tag_flag</code> can be <i>hit</i> or <i>miss</i>). If <i>hit</i> , the following fields are ignored. If <i>miss</i> , it indicates the address of block to be replaced and whether it is dirty or not.

controller to include a cache controller. The cache controller has a tag array for each chip connected to the Rambus channel. Each tag array is identified by the chip ID hosting the corresponding cache data array. The cache controller also keeps track of the control data for each block such as a valid bit, dirty bit and LRU counters. This way, the cache controller is able to locally decide on replacement policies and locally update the tag array. The cache controller communicates with the cache data arrays by sending cache commands through the channel.

4.3 PA-CDRAM operation

In our PA-CDRAM design, we extend the Rambus communication protocol [10] between the memory/cache controllers and the memory chips. To accomplish a data `read/write` request, a sequence of commands is issued along the Rambus channel. Figure 8 illustrates the sequence of commands and data on the row, column and data busses (timing diagrams) in the channel for a `read miss` (it also implicitly shows the `read hit` command sequence). The communication protocol for command packets for a `write hit/miss` is similar to a `read hit/miss` with the substitution of a `CRA` by a `CWA` command.

While Figure 8 shows the Rambus channel activity, Figure 7 details the activity in both the on-memory cache and the DRAM-core in our PA-CDRAM for a `read` request. For each request, the memory controller identifies which chip the requested data resides in, then lets the cache controller search the tag array corresponding to this target chip¹. During the tag comparison, the cache controller sends the requested block address through the channel using the `CRA` command. After the tag comparison, the cache controller sends the comparison result using the `CTT` packet. In the case of an on-memory cache hit (`tag_flag` is set), data is read directly from the on-memory cache (see the *cache read hit* section in Figure 8). Other fields in `CTT` are ignored.

In case of an on-memory cache miss, the cache controller uses the `CTT` packet to send the address of the block to be replaced and whether it is dirty or not. If the block is dirty (i.e., `dirty_flag` is set), then it is written to the write buffer. Meanwhile, the memory controller activates the target chip and sends an `ACT` packet to activate a row in a bank, followed by a read command, `RD` from a memory address. Then, the memory controller precharges the accessed row using a `PRER` command. After data is read from the memory address, it is copied to the on-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 memory controller issues a command to send the chip to nap.

The Rambus protocol imposes some constraints on the earliest time to send subsequent packets to the same chip. In Figure 8, we show a few time constraints: `RAS` delay (time between activating a row and a precharge in the same bank), `RAS` to `CAS` delay (earliest time to issue a column access strobe

¹The access latency and energy in Figure 3 include the cost of both tag comparison and data array access.

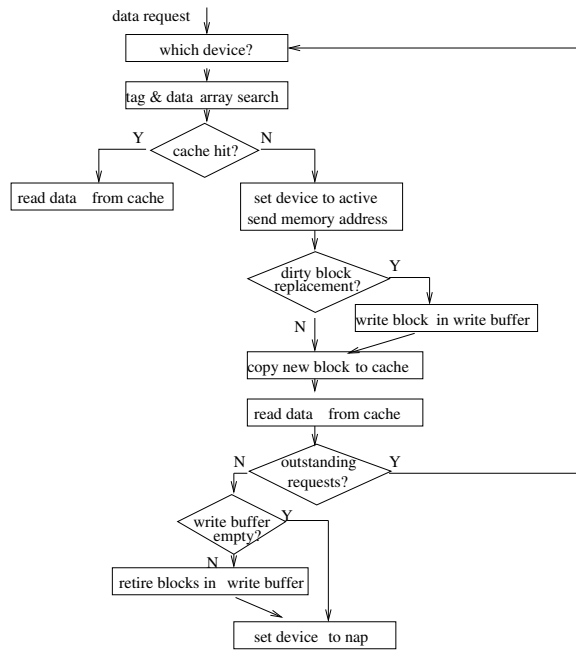


Figure 7: Flow diagram of a data access.

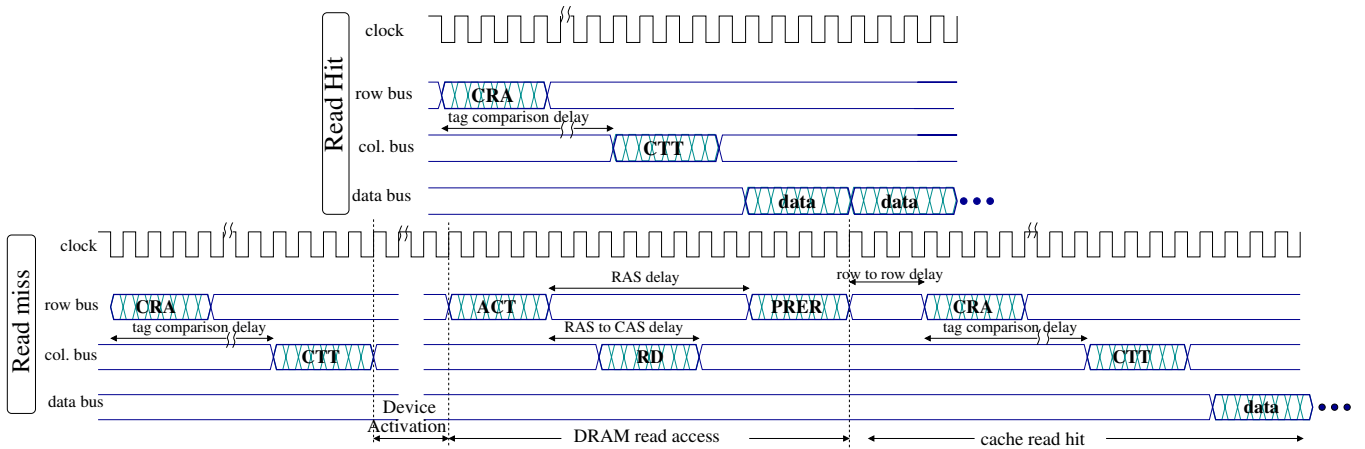


Figure 8: The timing diagram in the Rambus channel for an on-memory cache read hit and read miss transactions.

after a row access strobe is issued), and row to row delay (the minimum time to issue consecutive row commands to the same chip). For the cache access commands, an extra restriction is needed for the tag comparison delay. The tag comparison delay is the earliest time between issuing a CTT command after CRA or CWA commands for the same chip, which should be larger than or equal the tag comparison time in the tag array of the target on-memory cache. The memory controller sends chip activation commands through the control register bus (not shown in figure). This activation dictates a restricted period where no commands can be sent to the target chip.

4.4 Design Considerations

Several parameters and trade-offs can affect the design process of PA-CDRAM. Possible issues to consider are the design compatibility and the side effect of integrating logic and memory cells on the same chip. Here we present an alternative design that can maintain compatibility with existing Rambus based motherboards and discuss the potential trade-offs of using this design compared to the one presented in Section 4.2.

We presented the modifications needed for both the memory controller and the RDRAM chips. However, limiting the modification to only the memory chip would guarantee its compatibility with the standard Rambus memory controller and provide the flexibility of combining traditional RDRAM chips with PA-CDRAM chip on the same channel.

To achieve this compatibility we can build the cache controller on the memory chip rather than being co-located with the memory controller. We call this an *on-memory cache controller* design as opposed to *off-chip cache controller* design. In each PA-CDRAM chip, the on-memory cache controller only maintains the local on-memory cache. For each memory request, an ACT packet and RD or WR commands are sent to the target memory chip. The on-memory cache controller checks its local tag array then directs the request to either the request to either the cache array or the DRAM logic control.

This design would also affect the Rambus channel activity and accordingly the total bus energy. With an on-memory cache controller, for each L2 miss, the memory controller sends an ACT (24bit), a RD/WR (24 bits) commands and a possible PRER (24 bits) packet. On the other hand, with a separate cache controller, each on-memory cache hit results in CR and CTT packets (no need for a PRER command). On the downside, for an on-memory cache miss, two CR and two CTT packets are sent in addition to the ACT, RD and a possible PRER command. Thus, finding which design consumes higher energy depends on the number of on-memory misses versus the number of precharge packets.

Another effect associated with the cache controller location is the area consideration. Having a local cache controller on each PA-CDRAM chip would potentially increase the area of the memory chips. However, the design maintains the same pin count of standard RDRAM chip. In Section 5, we quantitatively evaluates these trade-offs, and show the energy and delay in each design.

4.5 Memory Address Mapping

Memory address mapping determines the location (chip, bank, and row within a bank) of each data address in the memory. The two most popular address mapping schemes are *linear* and *interleaved* mapping. Let there be n chips, m banks in each chip, and k rows in each bank. Further, let each memory address be of the form $\langle chip, bank, row \rangle$. In linear mapping, the memory controller maps data in a chip-wise fashion, that is, varying first *row*, then *bank*, then *chip*, until reaching k, m, n , respectively. The linear mapping policy has the benefit of energy savings when combined with a timeout policy for transitioning between the DRAM power states [14]. This is because linear mapping succeeds in grouping memory accesses into a small set of chips that stay busy servicing requests, while the remaining chips are less active and sleep for longer periods.

On the other hand, interleaved address mapping maps data varying different indices at a time. We consider here only one interleaved scheme for illustration purposes: memory is filled in a row-wise fashion, that is, by varying first *bank*, then *chip*, then *row*, until reaching m, n, k , respectively. In our memory organization, we use interleaved mapping that stores subsequent rows in subsequent banks to distribute the on-memory cache accesses to all chips. This mapping has two benefits. First, it fully exploits the on-memory cache resources. As a linear mapping implies heavy usage of just a subset of the available chips, an interleaved mapping takes full advantage of the total available on-memory cache resources. An interleaved mapping distributes the data accesses among all the chips, and increases the data lifetime in the on-memory caches before being replaced when compared to the linear mapping. Second, since memory accesses to different non-adjacent Rambus banks are faster than accessing the same bank [15], we use an interleaved mapping with a bank interleaving factor of 2 (i.e., using $bank = 0, 2, \dots, m - 2, 1, 3, \dots, m - 1$). Thus, an interleaved mapping in PA-CDRAM improves performance and energy consumption as a result of preserving the contents of the row buffers for longer periods and increasing the number of concurrent accesses across all the chips by distributing the load.

4.6 PA-CDRAM energy model

To measure energy consumed by PA-CDRAM, we need to take into account the busses, the DRAM-core and the on-memory cache, and their associated control logic (see below). Other smaller components like the write buffers consume negligible energy compared to the DRAM-core. Also, since there is no significant modification proposed in the memory and the cache controllers, we do not include their energy consumption in our model.

DRAM-core Energy in the DRAM-core is consumed during read/write activity (dynamic energy) and during idle periods (static energy). Dynamic energy is consumed in address decoding, E_{addr_decode} , and data transfer to/from DRAM-core. At each on-memory cache miss, the memory controller decodes the row and column addresses then precharges a row in the desired bank containing the requested data. Data access energy is proportional to the number of bytes transferred during read or write transactions. During inactivity, the chip’s static energy is a function of the time spent at each power state. A transition energy, E_{trans} , is consumed when a chip transitions to/from the active state. We limit our approach to use the *nap* state as the only low power state, because the results in [14] and [16] showed that the *nap* state is energy efficient and has a relatively low transition delay. Thus, energy consumed in the DRAM-core, E_{DRAM} can be expressed as follows:

$$E_{DRAM} = d E_{addr_decode} + c P_{access} t_{access} + \sum_s P_s t_s + r E_{trans} \quad (1)$$

where d is the number of data requests; c is the number of transferred bytes; P_{access} , and P_s are the power dissipated during read/write while in the s^{th} state (standby and *nap*), respectively; t_{access} and t_s are the time for transferring a single byte and the time spent in the s^{th} state, respectively; and r is the number of transitions to and from the active state. E_{addr_decode} , P_{access} , t_{access} , P_s , and E_{trans} are memory specific parameters while d , c , t_s , and r are factors of both an application’s memory access pattern and the memory hierarchy configuration.

Caches The energy consumed in the on-memory cache has two parts: dynamic and static energy. Dynamic energy is related to the switching activity in the cache cells, and is consumed during a cache read/write of data. Dynamic energy varies based on the on-memory cache organization; that is, it depends on the length of the word and bit lines. The static energy is leakage energy [17] and is a

function of the number of SRAM cells and the manufacturing technology. The total on-memory cache energy E_{cache} , is:

$$E_{cache} = f (E_{tag} + E_{data}) + P_{leak} t_{tot} \quad (2)$$

where f is the number of data transfers to/from the on-memory cache; and E_{tag} and E_{data} are the energy consumed per access during the tag comparison in the cache controller and the data access in the chip, respectively. P_{leak} is the leakage power and t_{tot} is the total execution time.

Busses Energy consumed in the busses is proportional to the total bus capacitance and the activity that takes place on each bus. Bus capacitance differs based on whether it is an external or internal bus. I/O pads are attached to external busses as a chip interface. We add the I/O pads capacitance to the total external bus capacitance. Switching activity in both busses and pads are a function of the number of misses and writebacks of the higher level cache attached to the bus (e.g., if there is a L2 miss, we count the activity on the bus that connects L2 to L3 or to the memory). We use the model in [18] to estimate the energy consumed in busses and pads.

5 Evaluation

In this section, we evaluate through extensive simulations, the energy efficiency of PA-CDRAM and show its effect on performance. We vary the main factors affecting both the energy and performance of the memory such as the on-memory cache organization, as well as the CPU-memory speed ratio and the execution environment (preemptive or non-preemptive scheduling).

5.1 Experimental Setup

To evaluate PA-CDRAM, we perform experiments using the SimpleScalar architecture simulator [12]. We integrated a memory module [19] that models a set of RDRAM chips connected to a single Rambus channel. The memory module also simulates the memory controller and its request scheduling. We extended the memory model by implementing the RDRAM power management scheme: multiple power levels² and a timeout policy for power mode transitions. We also extended the Rambus memory model by integrating a small cache attached to each chip and accounting for the high transfer bandwidth between each chip and its associated cache. The simulated memory consists of eight chips, each of size 32 MB for a total 256 MB memory.

We evaluate the PA-CDRAM power management organization using three metrics, namely: execution time of an application, energy consumption in the external memory hierarchy, and the energy-delay product, $E \cdot D$. Experiments are performed using a set of C applications from the SPEC2000 benchmark suite. To avoid the cold start effect of an application, we fast forwarded the simulation 2 billion instructions and simulated the following 200 million instructions, as in [20] and [21].

Our study evaluates the energy consumption of PA-CDRAM against a base case that employs traditional power saving policies provided by the Rambus architecture. Figure 9 illustrates the memory models evaluated in our experiments. The base case model uses a cache hierarchy configuration similar to the one used in Pentium4 EE processor running at 2GHz [13] as shown in Table 2. The size of the L3 cache in the base case is equal to the total size of all eight on-memory caches in PA-CDRAM. We use a linear memory address mapping in the base case to keep the least number of chips in the active state [14, 16]. DRAM power management in the base case uses a timeout policy for the deactivation of the RDRAM chips. When the chip is not reading or writing data, it is transitioned to the standby

²We limit our approach to use the *nap* state as the only low power state, because the results in [14] and [16] have shown that the *nap* state is energy efficient and has relatively low transition delay.

state. If no data requests arrive to a chip within c cycles, the memory controller transitions this chip to the nap state. From our experiments, with thresholds $c = 100, 500, 1000$ and 5000 cycles, $c = 1000$ is the best fixed threshold (i.e., it exhibits the least energy-delay product results in most of the tested application for the base case configuration).

	Configuration
L1 size	8KB, on-chip, Direct Mapped
L1 hit lat.	2 cycles
L2 size	256KB, 64B block size, 8way
L2 hit lat.	10 cycles, on-chip
L3 size	2MB: 128B block size, 8way
L3 hit lat.	15 cycles, off-chip

Table 2: Base case cache configuration.

We compute energy consumption in the DRAM-core, caches and busses following the model presented in Section 4.6. The timing and power characteristics of the simulated RDRAM chip are for the RDRAM 256Mb/1066MHz/32 split bank architecture [10]. These characteristics are summarized in Tables 3 and 4. Access latency and energy consumption for each cache configuration is obtained using Cacti 3.0 for the 130 nm technology (as in the Pentium EE [13]). We impose a slow-down of 35% on the logic cells in the memory. We do not account for the cache leakage energy (proportional to the number of SRAM cells) as we compare our results against a base case with equal L3 cache size. We model the address and data busses connecting the L2, L3, memory controller (MC), on-memory caches and the DRAM core. Each of these busses is categorized as either an external or an internal bus with a capacitance of 10pF [22] or 0.5pF [23], respectively. We refer to a bus (address and data) by the two memory elements that the bus connects; for example L2↔L3 is the bus connecting L2 and L3 caches. We account for L2↔L3, L3↔MC, and the Rambus channel in the base case, while accounting for L2↔MC, the Rambus channel, on-memory↔DRAM in PA-CDRAM as shown in Figure 9.

Table 3: RDRAM Timing (in memory cycles)

RAS to CAS delay	7
CAS access delay	8
CAS write delay	6
row precharge time	8
write buffer retire delay	8
read to precharge delay	4
precharge to precharge delay	8
time to transmit a data packet	4

Table 4: DRAM-core power profile

Nap power state	10.5 mW
Standby power state	225 mW
Active - reading	1112 mW
Active - writing	1187 mW
Active-Nap transition	17 nJ
address decoding energy	17 nJ

5.2 Effect of varying the cache organization

We study the effect of the cache organization on energy and performance of PA-CDRAM with respect to the base case by varying the configuration of the on-memory and L3 caches. A cache organization is represented with the tuple $\langle \text{block size: associativity} \rangle$. We use $B\langle \text{cache-organization} \rangle$ to refer to the base case with a L3 cache, and $PA\langle \text{cache-organization} \rangle$ to refer to PA-CDRAM with its on-memory cache organization. We experimented with many combinations of block sizes and associativities for the PA-CDRAM caches. Here, we show the results for (a) the base case: $B\langle 128:8 \rangle$, (b) on-memory cache with large blocks: $PA\langle 512:FA \rangle$ (from Section 3.2), (c) the base case with an L3 cache configured

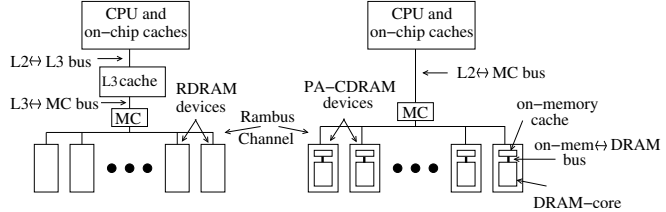


Figure 9: Memory organizations of the base case (left) and the proposed PA-CDRAM (right).

with large block size and high associativity: B<512:FA>, and (d) on-memory cache with base case L3 configuration: PA<128:8>. Figure 10 shows the energy-delay product of these different configurations. Other on-memory cache configurations (not shown for space limitation) perform worse in terms of energy-delay product than PA<512:FA> in most applications.

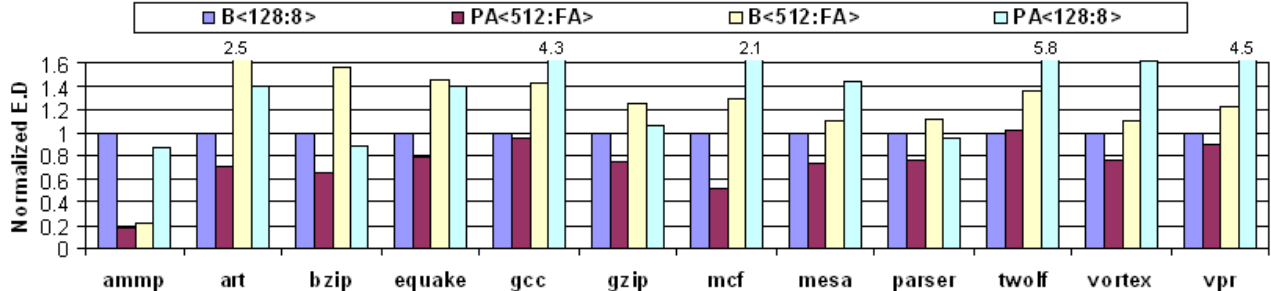


Figure 10: The effect of varying the cache organizations on the energy-delay product, normalized to B<128:8>.

The figure shows that the energy-delay product in B<512:FA> is always worse than PA<512:FA>. There are two main reasons contributing to this. First, the transfer bandwidth of the system bus limits the transfer size, causing a larger block to take longer to be transferred to the external cache in the base case compared to transferring it to the on-memory cache through the internal bus in PA-CDRAM. This improves the performance when using PA<512:FA>. Second, since accessing a small cache is faster and consumes less energy than accessing a larger cache with an equal block size and associativity (longer word/bit -lines), the access energy and access latency of the B<512:FA> cache is larger than PA<512:FA>. With respect to PA<128:8>, although the cache access energy is lower than PA<512:FA> (smaller blocks with lower associativity), the miss rate is higher. This results in an increase in the DRAM dynamic energy for PA<128:8>, because more accesses to the DRAM-core are requested, each involves an activation for the chip, address decoding in addition to the data transfer.

From our experiments with other block sizes (from 128B to 2KB) for the on-memory cache, we found that PA<512:FA> achieves the best energy-delay product among other fixed cache configurations across all applications.

In summary, we found that our PA-CDRAM configuration is best suited for on-memory caches and would not be as energy and delay efficient if located off-memory (large blocks cause extra delays). On the other hand, PA-CDRAM with an external cache configuration (as in the base case cache configuration) consumes more energy than the proposed on-memory configuration. Unless stated otherwise, we use PA<512:FA> and B<128:8> as our default settings.

5.3 PA-CDRAM energy and delay

In this study, we evaluate the potential energy and performance benefit of PA-CDRAM compared to the base case. When comparing the energy-delay product in each case as shown in Figure 11, there is up to an 84% (average 28%) savings over the base case. The figure also shows the individual savings in both energy and delay.

5.3.1 Effect on delay

For most of the applications, there is no significant improvement in the delay over the base case. This is because most of the CPU stalls resulting from L3 misses can be masked by the execution of

other independent instructions in the pipeline. Figure 11 shows results for a 4-wide processor, but our experiments with processors with smaller issue widths (not shown here for lack of space) show that there is a potential for a larger performance improvement over the base case (smaller width implies more stalls).

Three noteworthy exceptions in Figure 11 are *ammp*, *art* and *mcf*. For these three benchmarks, the total number of DRAM-core accesses is two orders of magnitude larger than the other applications. With so many memory accesses, it is hard for the processor to mask all memory stalls. PA-CDRAM improves delay over the base case because it reduces the total L2 miss penalty over all references (by reducing the on-memory miss rates compared with L3 miss rates). Thus, a memory intensive application’s performance can greatly improve when using PA-CDRAM.

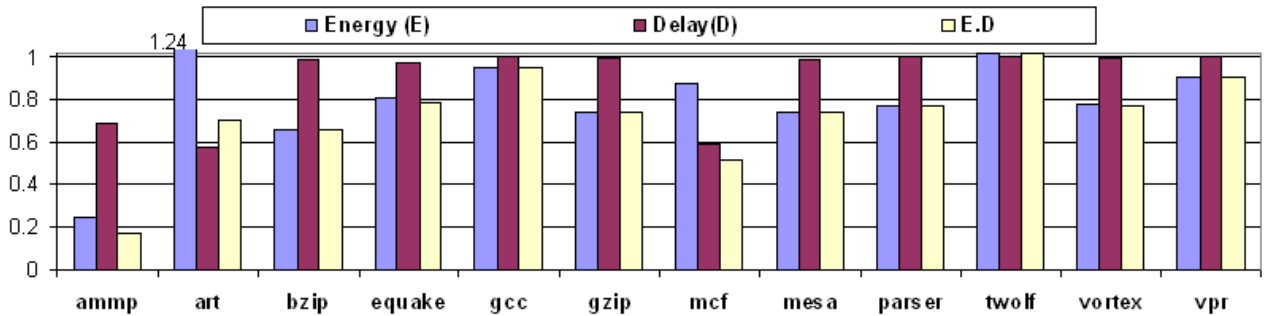


Figure 11: The breakdown of the energy-delay savings in PA-CDRAM (512:FA) normalized to B (128:8).

5.3.2 Effect on energy consumption

From Figure 11, we also see that savings in energy consumption alone reached up to 76%. All applications exhibit energy savings except for *art* and *twolf*. To analyze these savings (and higher consumption in *art* and *twolf*) we further break down energy consumption to show the relative energy consumption of each memory component normalized to the base case. Figure 12 shows the relative consumption due to the DRAM-core dynamic energy (DRAM-dynamic), DRAM-core static energy (DRAM-static), cache access energy (cache-dynamic), and bus energy.

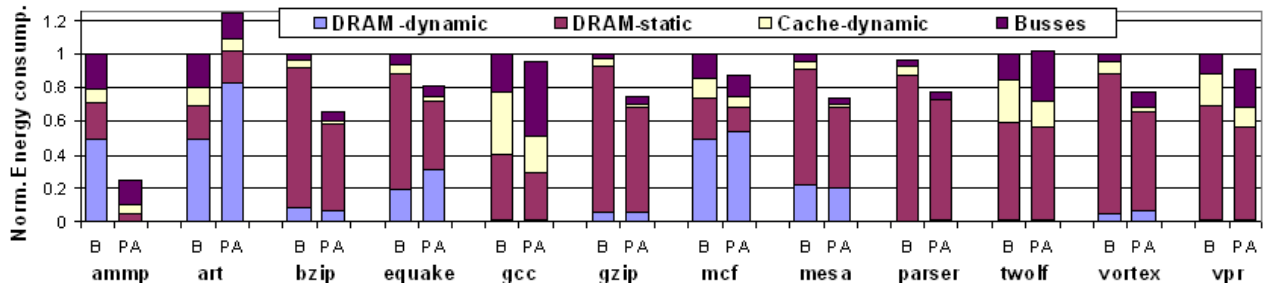


Figure 12: The breakdown of energy consumption in PA-CDRAM normalized to the base case.

DRAM-core energy: In all applications, the DRAM-static energy was reduced due to the increase in the duration of DRAM sleep 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 on-memory cache that filter some of the accesses to the

DRAM-core. However, applications like *art*, *equake*, *mcf*, *parser*, and *vortex*, suffer an increase in DRAM dynamic energy, even though the on-memory cache miss rates were reduced. This increase is due to the relatively large on-memory cache block size that caused these applications to access unnecessary data from the DRAM-core. 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 for an application saves further energy in PA-CDRAM dynamic energy compared to the base case.

Cache energy: As cache access energy is proportional to the length of activated bitline and wordline at each access, dividing the cache into small separate on-memory caches reduces the cache energy. The figure shows a decrease in cache dynamic energy across all application, even with the least reduction in miss rates (as in *vortex* from 5% in base to 2% in PA-CDRAM).

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 busses 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, for applications with high L2 misses and low L3 misses consume more bus energy in PA-CDRAM. On the other hand, applications with relatively high L3 misses, frequent bus activity occurs in L3 \leftrightarrow MC and the Rambus channel, resulting in higher bus energy in the base case. To quantify this trade-off, PA-CDRAM bus energy consumption is lower in applications if: $\#L3_misses > 0.3 \#L2_misses + 0.18 \#onmem_misses$ is satisfied. We derive this condition from the model in [23]. The derivation of this condition is detailed in the Appendix.

In the following sections we show the effect of varying several parameters in the memory hierarchy on energy and performance of the memory system.

5.4 Effect of the cache controller location

We quantify the trade-offs of choosing the location of the cache controller with respect to the overall performance and the total bus energy. We simulate on-memory and off-chip cache controllers with their associated cache access penalties. Figure 13 show the execution time of the benchmarks in both cache controller designs. We vary the performance penalty of the embedded logic in the PA-CDRAM from 10% to 50%. The execution time is normalized to the case where there is no performance penalty for the logic cells. This variation in penalty is used to show the effect of the manufacturing process on the overall performance of PA-CDRAM. From Figure 13, we notice that the performance degradation in the on-memory cache controller design is relatively insignificant compared to using an off-chip cache controller (on average 0.35% and up to 1.5%).

To test the effect of changing the cache controller location on the bus energy, Figure 14 shows the total bus energy of the on-memory cache controller design normalized to the case with an off-chip cache controller in the range from 0% to 5%. However, since the bus energy corresponds to less than 20% (on average) of the overall PA-CDRAM energy, the savings in bus energy is overshadowed by energy consumed in other memory components. Furthermore, the performance degradation due to the slower on-memory controller forces PA-CDRAM to consume more static energy than the off-chip controller design. As a result, the total energy of the memory (not shown) is slightly higher than using off-chip cache controller.

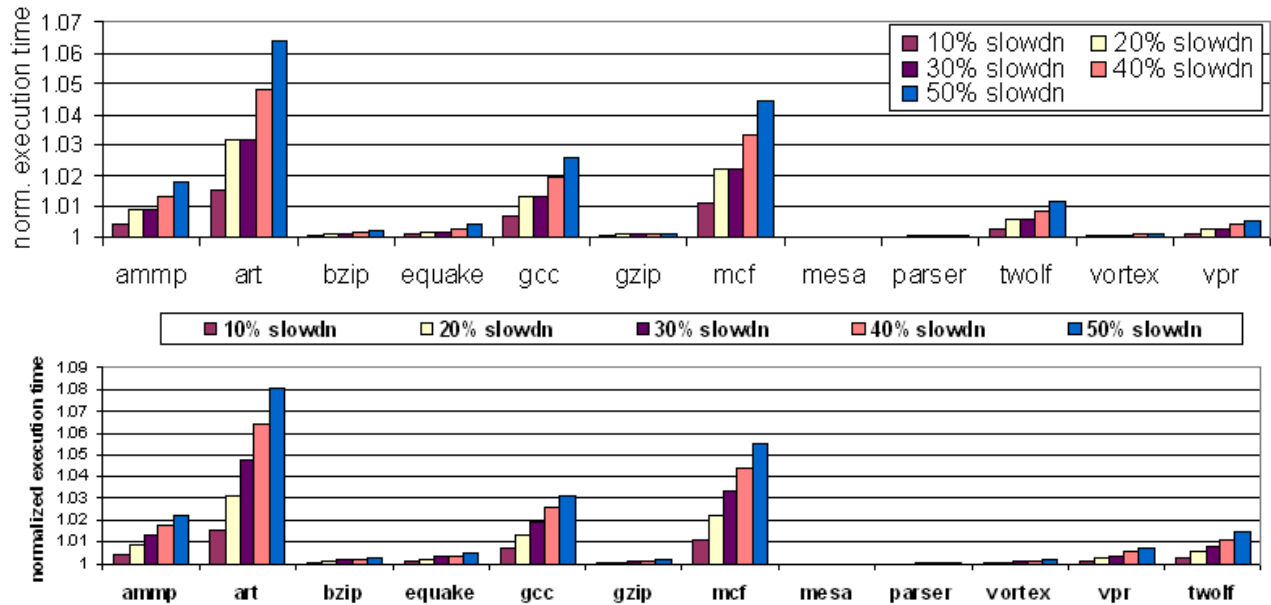


Figure 13: The effect of using an off-chip cache controller (above) versus on-memory cache controller (below) on performance.

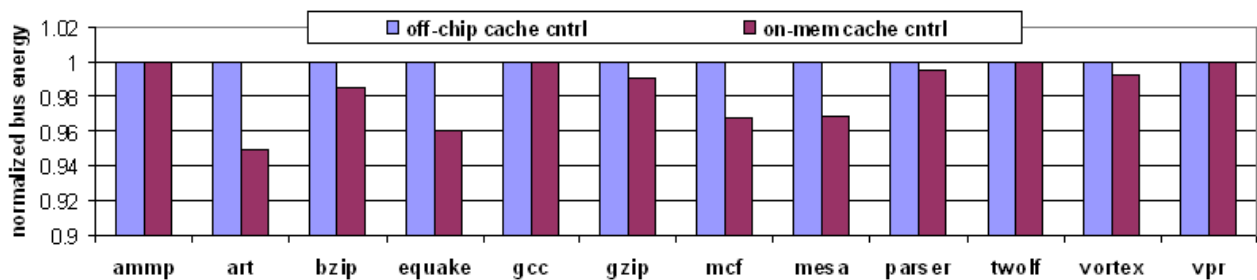


Figure 14: Total bus energy when using off-chip and on-memory cache controllers normalized to the bus energy consumed in case of off-chip controllers.

5.5 Effect of varying the cache size

Varying the cache size affects both the miss rate and the cache access costs (latency and energy). Increasing the cache size reduces the L3/on-memory miss rates, thus better performance is achieved in both cases. However, with respect to energy consumption, the effect of varying the cache capacity is not as trivial. Figure 15 shows energy consumption of the base case and PA-CDRAM while varying the total cache size (excluding L1 and L2) from 512KB to 4MB. Note that while the small cache sizes tested (512KB and 1MB) are too small for an L3 cache, we test at those sizes to demonstrate the trend. The results are normalized to B(128:8) with 2MB L3 cache. 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. This effect shows in *ammp*, *bzip*, *gcc*, *gzip*, *parser*, *twolf*, *vortex* and *vpr* where the total energy in the base case and PA-CDRAM increases again at large cache sizes. However, the energy increase is smaller in PA-CDRAM than the base case, which is due to accessing smaller individual on-memory caches.

Comparing the energy consumption of 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*, or (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, too many on-memory cache replacements takes place due to the limited number of blocks ($\frac{512KB}{8*512B} = 128$ blocks) per individual on-memory cache.

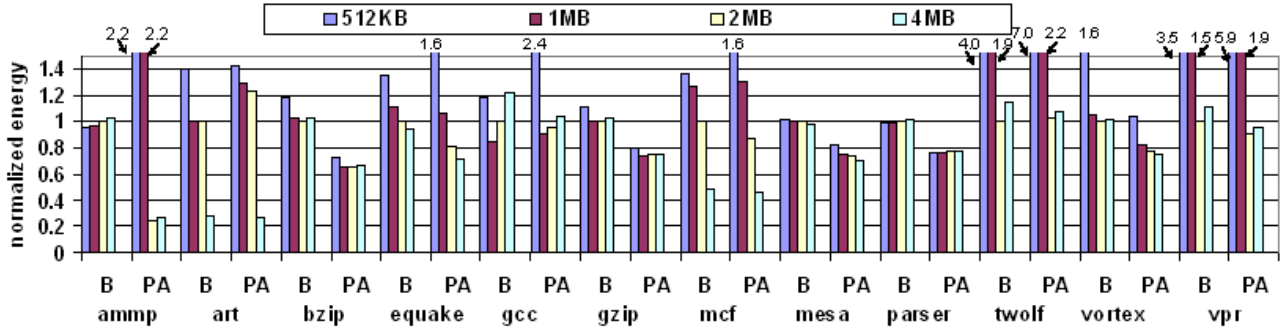


Figure 15: The effect of varying the cache size on the memory energy consumption normalized to B(128:8) with 2 MB L3 cache.

5.6 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. Figure 16 shows the execution times and energy consumption for PA(512:FA) at different clock rates normalized to B(128:8) at 2 GHz. Figure 16(a) shows that 9 applications exhibit minimal variation in execution times. 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 base case (as described in Section 5.3). 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.

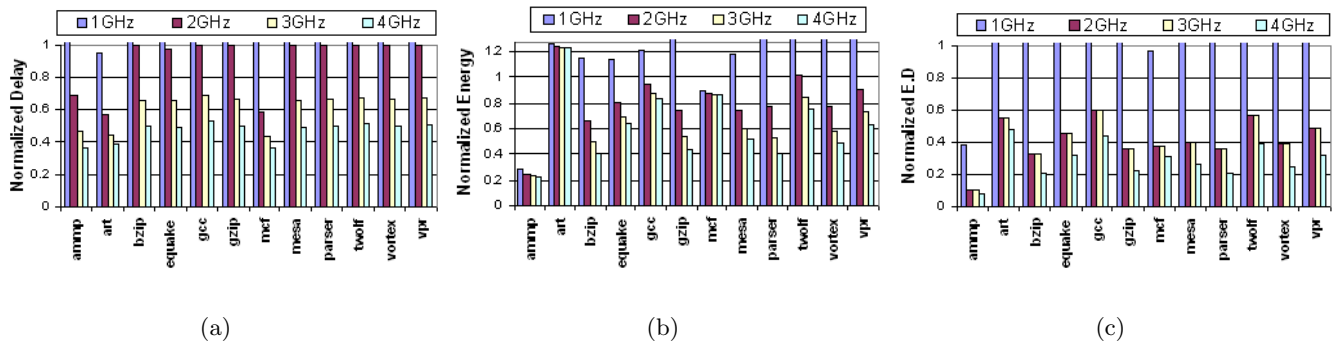


Figure 16: The effect of varying the CPU frequency for the PA-CDRAM, normalized to B(128:8) on the (a) delay, (b) energy and (c) energy-delay product.

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. Figure 16(c) shows the effect on the energy-delay product.

5.7 Effect of multitasking environment

The energy profile of an application running in a single task environment may differ from running the same application in a multi-tasking system with preemption. This difference rises from the fact that an application’s locality of reference is disturbed by the execution other applications, which will populate the cache with their own data. The result is higher miss rates experienced by each application. Unfortunately, our simulation tool does not directly support a concurrent multi-task environment. However, in order to approximate the effect of context switching on the memory system, we consider a scheduler that invalidates all the cached data³ for the expiring application (or task) before resuming execution of the ready task. The invalidation process involves writing back all the dirty cache blocks to memory. Our emulation of the context switch effect initiates an interrupt service routine every 10ms (similar to the time slice in Linux operating systems). The interrupt drains the processor pipeline then flushes the data in the caches. Upon the interrupt termination, the application’s execution is resumed.

Figure 17 illustrates the overhead of context switching compared to single task execution. All data is normalized to the base case when executing without preemptions. In all 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 on-memory caches as flushing all the small on-memory caches can be done simultaneously, while flushing the L3 cache serializes the write backs to the memory chips. Second, since the number of blocks in the L3 cache (2MB/128B) is larger than the on-memory caches (2MB/512B), the L3 cache has more address decoding for a chip 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. The increase in the DRAM energy is due to the relatively large number of on-memory cache writebacks factored by the large block size (512B vs. 128B) to be written to the DRAM-array.

³This is a worse-case behavior since usually the cache will not be completely flushed.

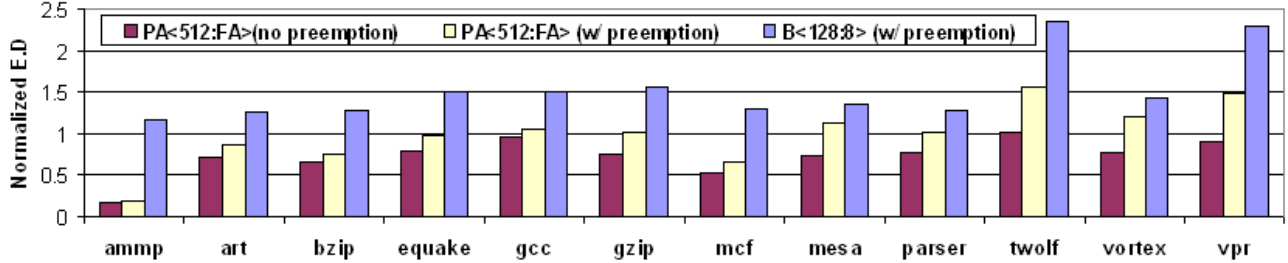


Figure 17: The effect of context switching every 10ms on the energy-delay product, normalized to the B<128:8> case without preemptions.

5.8 Summary of the evaluation study

For all applications, PA-CDRAM improves the energy-delay product over the base case. PA-CDRAM improves performance due to accessing the fast on-memory cache rather than the slower DRAM-core. This improvement is prominent in memory intensive applications. PA-CDRAM also reduces energy consumption in applications with spatial locality. The energy saving is the result of using (1) small distributed on-memory caches and (2) large on-memory blocks. The large blocks reduce the average miss rate, but it can increase the energy consumption in applications with poor spatial locality. This increase is due to the transfer of unnecessary data between the DRAM-core and the on-memory caches.

The proposed on-memory cache configuration is not suited for external caches and vice versa. Exploiting the fact that the memory chip has large internal bandwidth, reading data from the DRAM-core to on-memory caches does not hurt the performance; whereas, transferring data outside the memory chip induces more delay due to the constrained external memory bus.

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. Larger caches imply lower miss rates, but also higher access cost (delay and energy). Since doubling the L3 size corresponds to increasing each on-memory by only 25%, increasing the L3 size, at a point, can increase the total memory energy consumption. However, this increase is less prominent in PA-CDRAM due to the smaller increase in access energy and delay in each individual on-memory cache. As CPU frequency increases, longer CPU stalls occur due to memory accesses, thus the base case consumes more energy and delay. Nevertheless, PA-CDRAM improves both energy consumption and performance by filtering more memory accesses.

A multi-tasking environment, PA-CDRAM can lower the context switch overhead of cache invalidation through simultaneous flushing of dirty blocks in all on-memory caches. This process takes less time than serially flushing a single large L3 cache.

Due to the increase in the number of transferred bytes to/from the DRAM-core, PA-CDRAM can increase the total memory energy in case of applications with poor spatial locality (as in *art*). Also, PA-CDRAM does not perform as well in case of applications with relatively low L3 miss rates compared to the L2 miss rate (as in *twolf*), due to the extra energy consumed by the Rambus channel in PA-CDRAM.

6 More on Related work

To exploit the large internal bandwidth of memory chips, an extra on-chip cache was introduced by Mitsubishi [1]. The on-chip cache has faster access time thus reducing the average memory access time. Authors in [4] evaluated the performance of CDRAM in vector supercomputers. To improve the CDRAM performance, Kedem et al [9] proposed a cached-DRAM with wide cache line ranging from 4KB to 8KB interleaved across multiple DRAM banks. Whereas Hegde et al [24] proposed using a variable

width cache lines that fit the application access patterns to save energy consumed in unnecessary traffic between the DRAM-core and the on-memory cache.

Lebeck et al [14] proposed the use of power aware allocation policy where data is allocated sequentially in each bank to increase a bank idle periods. An implementation of the memory power manager in an Linux operating system [16] allocates memory pages to banks based on the allocating process. Other researchers suggested using some compiler techniques to group the requested pages in memory based on the running application’s order of data accesses [25] [26].

7 Conclusions

In this paper, we proposed a near-to-memory cache organization for reducing both energy consumption in memory and the delay seen by an application. While cached DRAM has previously been proposed to improve memory access time, in this paper we explore the energy efficiency of power-aware cached DRAM as an alternative to a traditional power-aware memory. For this, we address the challenges and the trade-offs between maximizing the performance and minimizing energy consumption, and we balance those trade-offs from the DRAM-core and the on-memory caches perspective. To realize these ideas in a realistic setting, we describe a design of PA-CDRAM using Rambus. We show the architectural changes for integrating caches in the Rambus chips and describe the needed changes in the operation and the modifications to the Rambus bus protocol.

There are many factors affecting energy consumption and performance of a PA-CDRAM memory system. Our evaluation shows that, when compared to traditional memory using time-out power management, PA-CDRAM saves up to 76% energy (19% on average). Moreover, PA-CDRAM reduces the energy-delay product by up to 84% (28% on average). In general, PA-CDRAM is best suited for a class of applications that exhibit a large number of DRAM-core accesses and reasonable data spatial locality. The energy benefits of PA-CDRAM becomes even better with the increase in CPU frequency and total on-memory cache size.

is greater with the increase in the CPU frequency and the increase in the total on-memory cache capacity.

Extension to this work will adapt the volume of the traffic between the on-memory cache and the DRAM-core (through varying the block size and the prefetching mechanism) as well as adaptively adjusting the aggressiveness of the timeout policy used for the DRAM power management. By adjusting these two factors according to the application’s memory access pattern, further energy savings can be gained from using PA-CDRAM.

Appendix

Here, we derive the condition where PA-CDRAM bus energy is lower than the base case’s bus energy. We use the bus model presented in [18].

Base case bus energy: L2 requests go through the address and data bus of L2↔L3, and an L3 miss goes through the address and data busses of both the L3↔MC and the Rambus channel. From the bus model, we get the bus energy, B_{bus} :

$$B_{bus} = C_{ex} \alpha V^2(L2_miss W_{addr}) + C_{ex} \alpha V^2(L2_miss + L2_wb) L2_bsize + 2 [C_{ex} \alpha V^2(L3_miss W_{addr}) + C_{ex} \alpha V^2(L3_miss + L3_wb) L3_bsize]$$

where C_{ex} and C_{in} are the capacitance of the external and internal bus, respectively, α is the bus transition factor (value between 0 and 1), V is the supply voltage, and W_{addr} is the width of the

address bus. $L2(3)_miss$ and $L2(3)_wb$ are the number of L2(3) misses and write backs, respectively. $L2(3)_bsize$ is the L2(3) block size.

PA-CDRAM bus energy: L2 requests go through the address and data bus of L2↔MC and the Rambus channel. On-memory cache miss goes through the address bus of the Rambus channel while the data go through the on-mem↔DRAM bus. From the bus model we get the bus energy, PA_{bus} , equals:

$$PA_{bus} = 2 [C_{ex} \alpha V^2(L2_miss W_{addr}) + C_{ex} \alpha V^2(L2_miss + L2_wb) L2_bsize] + C_{ex} \alpha V^2(M_miss W_{addr}) + C_{in} \alpha V^2(M_miss + M_wb) M_bsize$$

where M_miss and M_wb are the number of on-memory misses and write backs, respectively. M_bsize is the on-memory block size.

Base case versus PA-CDRAM bus energy The bus energy in the base case is higher than PA-CDRAM when $B_{bus} > PA_{bus}$. Divide both sides by $C_{ex} \alpha V^2$, and assume that no writebacks take place, ($L2_wb = L3_wb = M_wb = 0$), then:

$$L2_miss (W_{addr} + L2_bsize) + 2 [L3_miss (W_{addr} + L3_bsize)] > 2 [L2_miss (W_{addr} + L2_bsize)] + M_miss (W_{addr} + \frac{C_{in}}{C_{ex}} M_bsize)$$

By substituting with the system configuration for both the base case and PA-CDRAM ($W_{addr}=32$, $L2_bsize=64$, $L3_bsize=128$, $M_bsize=512$, $C_{in}=0.05$, and $C_{ex}=10$), we get:

$$\begin{aligned} L3_miss (64 + 256) &> (32 + 64) L2_miss + (32 + 0.05 * 512) M_miss \\ L3_miss &> 0.3 L2_miss + 0.18 M_miss \end{aligned}$$

References

- [1] B. Davis, *Moderan DRAM Architectures*, PhD thesis, University of Michigan, Ann Arbor, 2000.
- [2] V. Cuppu, B. Jacob, B. Davis and T. Mudge, “High-Performance DRAMs in Workstation Environments”, *IEEE Trans. Comput.*, vol. 50, n. 11, pp. 1133–1153, 2001.
- [3] D. Elliott, W. Snelgrove and M. Stumm, “Computational RAM: A Memory-SIMD Hybrid and its Application to DSP”, in *Custom Integrated Circuits Conference*, pp. 30.6.1–30.6.4, 1992.
- [4] W. Hsu and J. Smith, “Performance of cached DRAM organizations in vector supercomputers”, in *Proc. Intl. Symp. on Computer Architecture*, pp. 327–336. ACM Press, 1993.
- [5] Doris Keitel-Schulz and Norbert Wehn, “Embedded DRAM development: Technology, physical design, and application issues”, *IEEE Trans. on Design & Test of Computers*, vol. 18, n. 3, pp. 7–15, 2001.
- [6] NEC embedded DRAM, <http://www.necelam.com/edram90/>.
- [7] Steve Tomashot, “IBM embedded DRAM approach”, http://www-306.ibm.com/chips/techlib/techlib.nsf/products/Embedded_DRAM.
- [8] *Mitsubishi chips*, <http://www.mitsubishichips.com/>.
- [9] R. Koganti and G. Kedem, “WCDRAM: A Fully Associative Integrated Cached-DRAM with Wide Cache Lines”, Technical report, Duke University, CS dept., 1997.
- [10] “Rambus products”, <http://www.rambus.com/products>.
- [11] P. Shivakumar and N. Jouppi, “CACTI 3.0: An Integrated Cache Timing, Power, and Area Model”, Technical Report 2001.2, Compaq research labs, 2001.

- [12] “SimpleScalar simulator”, <http://www.simplescalar.com>.
- [13] “Intel Pentium 4 EE processor”, <http://www.intel.com>.
- [14] A. Lebeck, X. Fan, H. Zeng and C. Ellis, “Power aware page allocation”, in *Proceedings of the Intl. conference on Architectural support for programming languages and operating systems*, pp. 105–116. ACM Press, 2000.
- [15] A. Romer M. Gries, “Performance Evaluation of Recent DRAM Architectures for Embedded Systems”, Technical Report 82, Swiss Fedral Institute of Technology, 1999.
- [16] H. Huang, P. Pillai and K. Shin, “Design and Implementation of Power-Aware Virtual Memory”, in *USENIX Annual Technical Conf.*, pp. 57–70, 2003.
- [17] N. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. Hu, M. Irwin, M. Kandemir and V. Narayanan, “Leakage Current: Moore’s Law Meets Static Power”, *IEEE Computer*, vol. 36, n. 12, pp. 68–72, 2003.
- [18] H. S. Kim, V. Narayanan, M. Kandemir and M. J. Irwin, “Multiple Access Caches: Energy Implications”, in *WVLSI ’00: Proceedings of the IEEE Computer Society Annual Workshop on VLSI (WVLSI’00)*, page 53. IEEE Computer Society, 2000.
- [19] “SDRAM and RDRAM modeling for SimpleScalar simulator”, http://www.tik.ee.ethz.ch/~ip3/software/simplescalar_mem_model.html.
- [20] W. Lin, S. Reinhardt and D. Burger, “Reducing DRAM Latencies with an Integrated Memory Hierarchy Design”, in *Proc. of the Intl. Symp. on High-Performance Computer Architecture*, page 301. IEEE Computer Society, 2001.
- [21] Z. Zhu, Z. Zhang and X. Zhang, “Fine-grain Priority Scheduling on Multi-channel Memory Systems”, in *Proc. Intl. Symp. on High-Performance Computer Architecture*, page 107. IEEE Computer Society, 2002.
- [22] Y. Aghaghiri, F. Fallah, and M. Pedram, “Transition reduction in memory buses using sector-based encoding techniques”, *IEEE Trans. on Computer Aided Design*, vol. 23, n. 8, pp. 1164–1174, 2004.
- [23] I. Kadayif, T. Chinoda, M. Kandemir, N. Vijaykirsnan, M. J. Irwin and A. Sivasubramaniam, “vEC: virtual energy counters”, in *Workshop on Program analysis for software tools and engineering*, pp. 28–31. ACM Press, 2001.
- [24] A. Hegde, N. Vijaykrishnan, M. Kandemir and M.J. Irwin, “VL-CDRAM: variable line sized cached DRAMs”, in *Proc. of the Intl. Symp. on Hardware/software codesign & system synthesis*, pp. 132–137. ACM Press, 2003.
- [25] V. De La Luz and M. J. Irwin, “DRAM Energy Management Using Software and Hardware Directed Power Mode Control”, in *Proc. Intl. Symp. on High-Performance Computer Architecture*, page 159. IEEE Computer Society, 2001.
- [26] V. De la Luz, M. Kandemir and I. Kolcu, “Automatic data migration for reducing energy consumption in multi-bank memory systems”, in *Proceedings of the 39th conference on Design automation*, pp. 213–218. ACM Press, 2002.