MEMORY HIERARCHIES IN INTELLIGENT MEMORIES: ENERGY/PERFORMANCE DESIGN

BY

JOSE RENAU

Ingen., University of Ramon Llull, 1997

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2000

Urbana, Illinois

Abstract

Dramatic increase in the number of transistors that can be integrated on a chip, coupled with advances in Merged Logic DRAM (MLD) technology fuels the interest in Processor In Memory (PIM) architectures. A promising use of these architectures is as the intelligent memory system of a workstation or server. In such a system, each memory chip includes many simple processors, each of which is associated to one or more DRAM banks. Such a design extracts high bandwidth from the DRAM. Recently, advances in MLD technology are allowing the on-chip logic transistors to cycle as fast as in logic-only chips, causing a speed mismatch between the high-speed on-chip processors and the slow DRAM banks. Furthermore, the presence of so many processors on chip, all accessing memory, may create thick spikes of energy consumption.

In this thesis, I address how to design an efficient memory hierarchy inside an intelligent memory chip. This is a multi-dimensional problem that involves optimizing for performance, energy efficiency and, to a lesser extent, area efficiency. This thesis examines and evaluates simple hardware techniques to eliminate excessive power consumption using real-time corrective support. The results indicate that, to minimize the energy-delay product, each DRAM bank should include a sizable cache of about 8 Kbytes, support segmentation and interleaving, and optionally pipelining. Furthermore, a spectrum of real-time corrective schemes to limit power consumption are evaluated. Of these schemes, gating the clock offers the best tradeoff.

To my family and my friends.

Acknowledgments

I really want to thank my advisor Josep Torrellas for letting me work in the FlexRAM group. Special thanks to Michael Huang and Seung-Moon Yoo for giving me all the data that made possible to do this thesis. Without them, It would have been completely impossible to finish my thesis.

I also want to thank all the members in the IACOMA group. Thanks for a nice year to all the members in The People's Office.

More personally, I also want to remember my friends for helping me to "disconnect" from work. Without Sandra, Leonardo, JT, Yusuke and Pedro my life in the University would have been harder. Even more personally I would like to thank my father, my sister and all my other family members.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Continuous advances in VLSI technology are fueling the trend toward processor and memory integration in a single chip. Moore's law predicts sustained dramatic increases in the number of transistors that can be integrated on a chip [37]. Recent advances in Merged Logic DRAM (MLD) technology seem to be able to integrate on-chip logic cycling as fast as in a logic-only chip. The DRAM in this system is only about 10% less dense than in a memory-only chip [18, 22]. As a result, a chip in 0.18 μm may well integrate 64 Mbytes of DRAM and over 50 simple processors cycling at 800MHz.

This integration trend can potentially have an important and lasting impact on computer architecture. By placing the memory so close to the processor the memory latency is reduced dramatically. As a result the processor stall time due to memory accesses decrease. Including several processors on the chip, enables fine-grain parallel execution and speeds up parallelizable benchmarks.

Different architectures based on what has been called Processor In Memory (PIM) architectures, intelligent memory, or Intelligent RAM (IRAM) have been proposed [34]. These architectures can be classified based on the role of the PIM chip: main processor (or processors) in the system, special-purpose processor or co-processor, and intelligent memory system of a workstation or server. The first class of architectures includes Berkeley IRAM [33], Shamrock [27], Raw [41][1], and SmartMems [39] among others. Examples of special-purpose processors include, among others, Imagine [9][1] and engines to run vector applications [25], process data at the disk [32] or control ATM switches [3]. Finally, examples of intelligent memory systems include FlexRAM [24], Active Pages [31], and DIVA [15].

The focus of this thesis is on MLD chips that include many, relatively simple processors, each of which is associated with a memory bank. Such a design, which is often used in the intelligent memory systems class of architectures above, represents an attractive design point. Indeed, including many processors on a PIM chip is a good way to extract high bandwidth from the DRAM. Disappointing performance on some previous PIM architectures are due to the inability to extract high bandwidth from the DRAM [2]. Several large DRAM banks in a chip is a natural organization. For example, to increase the number of high-speed localities that a 1-Gbit chip can maintain, can be organized as 4-32 banks, each with its own row buffers [42]. The proposed design includes a high-frequency, simple processor in each bank.

This chip organization has recently become more interesting thanks to dramatic advances in the MLD logic speed [18, 22]. In each bank, there is a mismatch between the high-speed processor and the slow, high-latency DRAM. For example, the former may be an 800-MHz two-issue processor, while the latter may have a 15 ns access time. Consequently, an efficient on-chip memory hierarchy is needed in every bank. This hierarchy may include a cache, prefetching support, optimized data buffers, row buffers, and a sub-banked DRAM structure. Untuned memory systems may significantly harm the performance of the intelligent memory. Additionally, the on-chip memory hierarchy has to be tuned not only for performance, but for energy-efficiency and, to a lesser extent, for area-efficiency.

The first contribution of this thesis is to examine and evaluate the trade-offs in the design of the on-chip memory hierarchy in a multi-banked intelligent memory chip with many fast, simple processors. All the studies are done examining performance, energy efficiency, and area issues. Special attention is paid to energy consumption, since an intelligent memory chip has special considerations. The presence of so many processors on

[1]These systems currently use SRAM.

chip, all accessing memory, results in thick spikes of energy consumption. For this reason, the second contribution is real-time techniques to limit excessive power consumption.

Overall, the results indicate that, to minimize the energy-delay product [14] in an application, each DRAM bank should include a sizable cache of about 8 Kbytes, support segmentation and interleaving, and optionally pipelining. Furthermore, there are several simple and effective real-time corrective schemes to limit power consumption. The spectrum of schemes offers design points for different emphasis, but on average, gating clock to the whole chip produces the best control of power spikes without unnecessary slowdown.

This thesis is organized as follows: Chapter 2 defines the problem in more detail; Chapter 3 and 4 discuss issues in memory hierarchy design for these systems and techniques to prevent excessive power consumption; Chapter 5 discusses experimental setups for evaluation; and Chapter 6 evaluates the tradeoffs and the proposed designs.

Chapter 2

Problem Addressed and Related Work

To better define the problem two issues are examined: intelligent memory systems and power-saving techniques.

2.1 Intelligent Memory Systems

The focus is on MLD chips which include many, relatively simple processors, each one with a bank of DRAM. This is the typical design proposed by intelligent memory systems. Three such systems are FlexRAM [24], Active Pages [31, 7], and DIVA [15]. Such systems can be represented as in Figure 2.1, where the on-chip processors, memory, and network organization may vary. The network and how the different processors communicate with each other is beyond the scope of this thesis.

To quickly supply data from several localities, the intelligent memory chips are organized in many banks. For example, the 1 Mbyte IBM MLD DRAM macro [17] was designed to be used as a bank. Current advances in MLD technology enable on-chip processors to cycle at frequencies as high as in logic-only chips [18, 22], which is 800-1000 MHz. Fast processors will put pressure on the memory and require a non-trivial memory hierarchy in each bank.

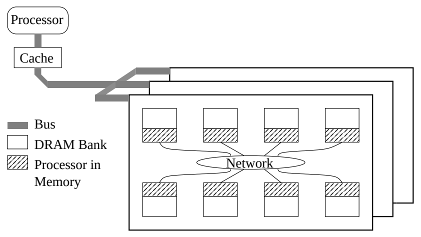


Figure 2.1: Generic organization of an intelligent memory system.

Current designs are more conservative. For example, a FlexRAM chip contains 64 simple processors, each associated with a 1 Mbyte of DRAM. Processors cycle at 400 MHz and each memory bank has 3 2 Kbytes row buffers to provide fast localities. The Active Pages system [31] has no cache between the logic and DRAM, the logic is cycled at 100MHz. A discussion of different processor organizations is presented in [7], where a modest 512-bytes cache is suggested. The DIVA system relies on a 256-bit row buffer to provide fast access.

The communication between processors in memory may be orchestrated from the outside by the host processor as in Active Pages and Diva, or by an on-chip controller as in FlexRAM.

This thesis focus on optimizing the data memory hierarchy of aggressive systems (Chapter 3). The instruction memory hierarchy is less interesting because the codes run by these simple processors have a small footprint. In addition, in some systems like FlexRAM [24], these processors use a simple, highly-encoded ISA that uses little storage. Consequently, the code is likely to reside in fast per-processor instruction memories or caches.

2.2 Power-Saving Techniques

An important aspect of designing the memory hierarchy of an intelligent memory chip involves estimating and limiting excessive energy consumption. Estimating energy consumption in a memory hierarchy is done by counting the number and types of accesses to each level of the hierarchy and then multiplying by the energy of each access [8, 16, 23] (Section 3.3).

To limit excessive power consumption in embedded systems, many techniques have been proposed. For example, some existing techniques target the consumption in the processor, such as gating to control speculative instructions [30], disabling functional units through clock gating [5, 14], or putting the processor in sleeping mode [19, 5]. Other techniques have addressed caches, like instruction cache throttling [5], improving the coding of instructions to reduce instruction cache accesses [14], disabling caches when the processor is stalled [14], cache sub-banking [12], or adding a small filter cache between the processor and the primary cache to intercept accesses with low energy consumption [26]. Finally, there are whole-system techniques [20], like gating the clock of the whole system or putting the whole system to sleep.

However, most of these techniques, designed for processors and caches, may not be the most cost-effective ones in an intelligent memory chip with many processors and large DRAM arrays. In this environment, processors tend to be simple, have smaller caches, and a sizable fraction of the energy consumed may come from data accesses to the DRAM itself. For this reason, processor-oriented techniques like gating speculation have little applicability or limited effectiveness. Furthermore, instructions may be encoded into efficient ISAs [24], which reduces to some extent the need for instruction cache optimizations. Finally, processors have relatively small caches, which may preclude adding filter caches.

The optimizations proposed directly reduce the power dissipated in the memory subsystem. The presence of so many processors, all accessing memory, creates thick spikes of energy consumption. Consequently, monitoring the chip conditions and providing a real time corrective action is an interesting possibility to limit power consumption. Other researchers have used sensors that monitor parameters like chip temperature in real time to trigger a processor interrupt when a threshold is reached [5]. The Chapter 4 analyzes interrupt-free low overhead schemes.

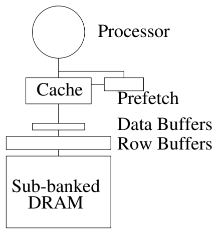
Chapter 3

Memory Hierarchies for Intelligent Memory

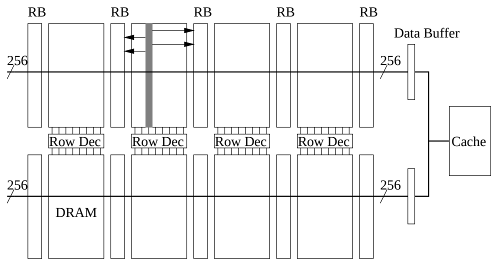
The design of the memory hierarchy for each of the banks in an intelligent memory chip is a multi-dimensional problem that involves performance, energy consumption and, less importantly, area. A good design maximizes performance while limiting the energy dissipated and the area utilized. In the following chapter, these three dimensions are considered.

3.1 Performance Issues

Figure 3.1-(a)considers a very general organization of the memory hierarchy for each of the memory banks on chip. Each bank contains a multi sub-banked DRAM array, row buffers, data buffers, cache and a prefetcher. Many different instantiations are possible. Figure 3.1-(b) shows the DRAM organized into 8 sub-banks with 10 row buffers and 2 data buffers. When a row is read from a sub-bank, half of it goes to the row buffer on the right and half to the one on the left, since one row buffer cell can not fit in a bit-line pair pitch.



(a)



(b)

Figure 3.1: Organization of the memory hierarchy in one bank: generic organization (a) and one possible instance (b).

3.2 Memory Bank Organization

There are different organizations for DRAM array and row buffers, depending on the number of sub-banks and buses, what fraction of the memory bank is activated at a time, and how much access pipelining is supported.

In the *traditional* (Trad) organization, only one access can be processed at a time. When the memory is accessed, every other memory sub-bank is activated. Not all the sub-banks can be activated simultaneously because two consecutive sub-banks share a row buffer. Figure 3.2-(a) shows a 4 sub-bank *traditional* organization.

With *segmentation* (S), only one sub-bank is activated at a time. This requires extra control lines. Figure 3.2-(b) shows a 4 sub-bank segmented organization. A performance advantage of segmentation is that the contents of the row buffers are more decoupled: row buffers 0-1 and 2-3 in the example are loaded at different times. In most benchmarks, increasing the number of localities increases also the row buffer hit rate.

With *interleaving* (I), each sub-bank is sliced and a data bus is assigned to each one of the resulting slices. The area is increased because is necessary to duplicate the column control and row buffers. Figure 3.2-(c) shows a 2-way interleaved organization generated from Figure 3.2-(b). Now, each sub-bank has half the width. This system has higher performance. If requests go to sub-banks connected to different data busses, the data can be extracted in parallel. Assuming a single address bus, the transfer of addresses

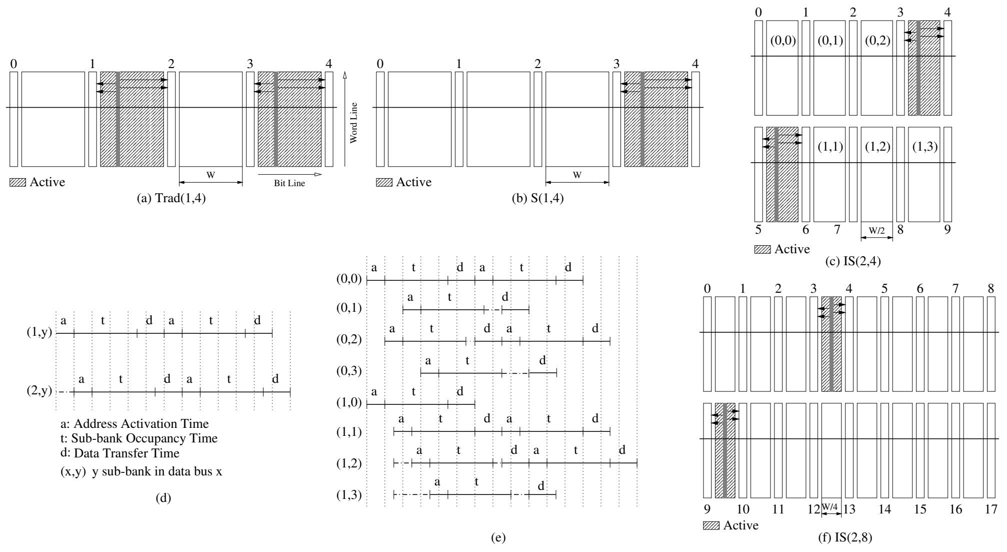


Figure 3.2: Different DRAM bank organizations and timings.

causes some serialization. Each data bus is connected to a data buffer. Figure 3.2-(d) shows a time diagram with the maximum overlap of sub-bank accesses. The duration of the *bank occupancy time* depends on whether the access hits or misses in the row buffer. Getting the data from different sub-banks sharing the same data bus is limited by the time required for a sub-bank to get ready.

With *pipelining* (P), the sub-banks can be started consecutively, overlapping the DRAM accessing. The only serialization happens on the shared address bus and data bus. Figure 3.2-(e) shows a time diagram with the maximum overlap of sub-bank accesses. The result is a faster system.

Finally, the performance could be improved by further slicing each sub-bank horizontally and creating more sub-banks. Figure 3.2-(f) shows a 2-way interleaved system with 8 sub-banks per data bus. The advantage of this system is that, more row buffers provide more localities which can be accessed faster. Moreover, with smaller bit line capacitance, the array can be accessed even faster. An i-way interleaved system with j sub-banks per data bus is represented as (i, j). A compact nomenclature is used in the remaining of the thesis, where Trad(i,j), S(i,j), SP(i,j), IS(i,j), ISP(i,j) corresponds to

Traditional, Segmented, Segmented Pipelined, Interleaved Segmented, and Interleaved Segmented Pipelined respectively.

Other Levels of the Memory Hierarchy

The memory hierarchy of Figure 3.1-(a) also includes a cache and a simple hardware prefetch engine. The line size of the cache is equal to the width of the data buffer. The prefetch engine performs hardware prefetch on first-touch: prefetching the line that follows the one that is accessed for the first time.

Each level of the hierarchy has a different purpose. The cache keeps a few localities very close to the processor. The prefetcher helps to exploit spatial locality beyond the length of the cache lines; it is also quite effective at eliminating compulsory misses. Finally, the row buffers holds additional localities. It is effective in intercepting requests coming from the prefetcher, which could otherwise cause additional memory accesses.

3.3 Energy Consumption Issues

To compute the energy consumed by the memory hierarchy, a model based on the work of other researchers [8, 16, 23] is utilized. First, the number of each access class to the memory hierarchy is counted. Accesses are classified into reads, prefetches, and writes, and on what level of the hierarchy they reach. In addition, there are displacements of dirty lines from the cache, which also consume energy. Then, the average energy consumed is computed by each access class by dividing the access into simpler operations. For example, a prefetch that hits in the row buffer is divided into a cache tag check, a read hit in the row buffer, and a line fill in the cache. The energy consumed for these operations is show in Chapter 5. Finally, the total energy consumed is the addition of the contributions for all kind of accesses:

$$\begin{aligned} Energy &= NumReadsHitCache \cdot EnergyReadHitCache + \\ &\quad NumReadsMissCache\&HitRowBuff \cdot EnergyReadMissCache\&HitRowBuff + \\ &\quad ... + NumDirtyDisplaceCache \cdot EnergyDirtyDisplaceCache \end{aligned}$$

Table 3.1 shows the energy consumed in a single access: a read hit in the cache, a cache read miss that hits in the row buffers, and a cache read miss that misses in the row buffers. The table shows data for the different DRAM organizations. The memory hierarchy includes a 2-way 8 Kbytes cache and a 1 MByte DRAM bank with 4096 columns per sub-bank. The method used to estimate energy consumption of each access is shown in Chapter 6.

Access Type	DRAM Memory Organization						
	Trad(1,4)	S(1,4)	IS(2,4)	IS(2,8)	SP(1,4)	ISP(2,4)	ISP(2,8)
Read Hit in Cache	191	191	191	191	191	191	191
Read Miss in Cache & Hit in Row Buff.	468	468	506	517	537	576	586
Read Miss in Cache& Miss in Row Buff.	6999	3739	2287	1556	3739	2287	1556

Table 3.1: Energy in pJoules consumed in the memory hierarchy by three types of accesses.

The consumption for an access satisfied by the cache is 191 pJ. This is less than the energy for an access satisfied by the row buffer (450-600 pJ), which varies with data line length and data buffer operation. It's also much less than that for one that has to go to the DRAM memory (1,500-7,000 pJ) which strongly depends on the number of cells per bit line and sub-bank organization.

For accesses that reach the DRAM, the energy consumed is roughly proportional to the number of DRAM cells activated in the access. Consequently, segmentation saves energy: Changing from Trad(1,4) to S(1,4), only half of the cells are activated, which saves about 50% of energy. Interleaving saves energy due to smaller capacitance obtained by reducing the number of cells per bit line: Changing from S(1,4) to IS(1,4), bit line capacitance is reduced by half. The savings is about 40% of energy. Furthermore, increasing the number of sub-banks, the same effect occurs: going from IS(2,4) to IS(2,8), also reduces the bit line capacitance by half and save about 1/3 of the energy. Pipelining has little effect in energy consumption.

For accesses that hit in row buffers, the energy depends on the memory organization and data buffer operation. While segmentation has no impact, interleaving increases the energy by 8% because of longer data lines. Pipelining increases the energy by about 15% because, due to the faster cycle time required, the current mode sense amplifiers [36] induce a higher transient power consumption without precharge. Finally, the energy consumed in a row buffer hit may depend on whether or not the row buffers are isolated from memory. If they are, a row buffer write hit is nearly as energy consuming as a miss, because it is necessary to re-activate the word line that was de-activated to isolate the row buffer, and redo cell-sensing operation. However, isolating row buffers allows to precharge bit lines earlier, hiding the RAS precharge time in case of a subsequent row buffer miss. Isolating the row buffers additional performance is obtained.

Although not shown in the table, changes in the cache size induce relatively small changes only since most energy is consumed in sense amplifier to read cell data. For example, a hit in a 256-bytes cache consumes 160 pJ. Finally, the energy consumption of a prefetch that hits in cache is very small, the tag check requires only 8 pJ. Prefetches that miss consume like a regular read miss.

3.4 Area Issues

The different levels of the memory hierarchy have different area requirements. Using a 0.18 μm technology, the DRAM cell area, the cache, the row buffer, and the data buffer is approximately 0.3 μm^2, 7 μm^2, 16 μm^2, and 320 μm^2 respectively. The row buffer cells and, especially, the data buffer cells are large because the former needs to sense small destructive signal difference from a cell and the latter uses current mode sense amplifier and drives a long wire. To get an idea of the overall area required, Table 3.2 compares some caches to different DRAM organizations.

Config	DRAM(1 MByte) + Row Buffer + Data Buffer + Control			Cache			
				4 Way		2 Way	
	(1,4)	(2,4)	(2,8)	256 Bytes	1 Kbyte	8 Kbytes	16 Kbytes
Area (mm^2)	4.25	4.83	5.23	0.07	0.16	0.60	1.15

Table 3.2: Comparing the approximate area required by different organizations.

The cache area is much smaller than the memory: even an 8 Kbytes cache takes less than 15% of the memory area. For the memory area, neither segmentation nor pipelining has much significance. What increases the area is the higher number of row buffers needed for increasing the number of sub-banks. Going from (1,4) to (2,4) and (2,8) in the table, the number of row buffers changes from 5 to 10 and 18, increasing the are by 14% and 23% respectively. Part of the increase is due to extra data buffers introduced for *Interleaving*, going from (1,4) to (2,4). Compared against the DRAM, the prefetcher takes a negligible area.

3.5 Design Points

The analysis focus on three design points for the memory hierarchy. The first one is the design that *maximizes performance*. Since embedded systems are often limited by the energy that they can consume, a second design point is the *energy-delay product* minimization. Delay is the inverse of performance and, therefore, this second design balances performance and energy consumption [14]. Finally, the third design *minimizes area-delay product*. This design represents the one that maximizes transistor utilization.

Chapter 4

Limiting Power Consumption Dynamically

Even the best memory hierarchy designs from the previous section are unlikely to be good enough for intelligent memory chips. Many processors can access memory simultaneously, creating thick energy consumption spikes, and potentially increasing chip temperature dangerously. This can be seen from Figure 6.9-(a), which shows how power consumption varies across time on a 64-processor intelligent-memory chip. The memory hierarchy used and the application run will be described later.

To address this problem, it may not be optimal to control the power dissipated in the chip with static power-limitation techniques. The result may be a design for the worst case scenario that degrades performance in common cases unnecessarily. This chapter examines dynamic or real-time power-limitation techniques. The goal is to minimize the spikes while not affecting the performance much for the rest of the time.

As indicated in Chapter 2.2, the thesis focus on power-limiting techniques that target mainly the memory hierarchy. Indirectly, of course, these techniques will also reduce the power consumed in the rest of the system.

4.1 Techniques Used

Ideally, the energy consumption should be reduced through the elimination of wasteful operations. Doing so would reduce the energy-delay product for the application run. Possible wasteful operations are useless prefetches and clock transitions in idle memory banks. However, none of these two cases wastes much energy: the prefetching scheme supported is very conservative and wastes few prefetches, while clock-induced signal toggling in idle memory banks consumes little energy compared to memory operation. Consequently, techniques that may increase the energy-delay product should be analyzed.

The techniques assume that the chip includes a sensor like the one in the PowerPC [5]. The sensor measures the total energy consumed in the chip and that it can be sampled at around 1 μs intervals. At the end of every time interval T, the sensor is sampled, computing the average power consumed ($P_{measure}$) and the $PowerRatio$ in the interval. The $PowerRatio$ is $P_{limit}/P_{measure}$, where P_{limit} is the maximum sustained power that the chip is allowed to dissipate. Only very fine spikes are allowed over P_{limit}.

In most of the following techniques, the $PowerRatio$ is used to start different corrective actions. Although the $PowerRatio$ includes the contribution of the whole chip, it is fine to use it in memory-oriented feedback control because the goal is to control total power consumption of the chip. In addition, total power dissipation will be easier to obtain in a real chip implementation.

4.2 Slowing Down Memory Through Clock Gating

In this technique, the memory is slowed down during power consumption spikes by gating the clock in the DRAM banks for a certain fraction of each time interval T. The number of cycles in the interval that the clock is not gated is called the memory available cycles ($AvailCycles$). At the time when the interval finishes, the $AvailCycles$ for the next interval is computed by multiplying the current values of $PowerRatio$ and $AvailCycles$. Consequently, when it is over the allowed power consumption, the number of available cycles in the next interval decreases, which is likely to decrease the power consumption. The opposite occurs if the $PowerRatio$ is more than 1. Therefore, it will tend toward a P_{limit} consumption.

In the naive implementation of this technique evaluated, the clock is gated in all the memory banks for a period of $T - AvailCycles$ cycles at the beginning of each interval T. More advanced schemes can choose to break down the gated cycles into several short-duration periods inside the T interval. Additionally, instead of applying clock gating to all the memories at once, gating can be applied in different regions of the chip at different times within T in a round-robin manner. These techniques curve temperature increases better.

This scheme is also extended to gate the clock in the processors and cache as well.

4.3 Slowing Down Cache Accesses

This approach is appropriate when most of the memory accesses hit in the caches and, therefore, the cache accounts for most of the energy dissipated. The techniques proposed are to increase the latency of a cache hit and to reduce the number of outstanding processor-initiated requests that a cache can support.

The algorithm proposed is as follows. For every time interval T that finishes and finds that $P_{measure}$ is greater than P_{limit}, all the caches are slowed a bit. Specifically, the first few times, the number of outstanding loads and stores is reduced by half. Once one outstanding load and one store is reached, the cache hit latency is progressively increased by 1 cycle until the power is set to the limit. The reason behind reducing outstanding accesses before increasing the cache hit time is that, otherwise, the pipelining of many outstanding accesses hides any changes in cache hit latency.

These changes are in the reverse order for every time interval T that finishes and finds that $P_{measure}$ is less than 90% of P_{limit}. Note that a 10% hysteresis is allowed as a corrective mechanism to prevent oscillating corrections. Very small hysteresis thresholds tend to increase oscillation, while very large ones discourage recovery, therefore adversely affecting performance.

The hardware required for these changes is simple. To reduce the number of outstanding requests, some of the pending request buffers are marked as unavailable. To increase the latency of a hit, delay stages are inserted to the data return path. This delay logic should be inserted such that it does not affect the cycle time of the cache or the processor.

4.4 Slowing Down Memory Through Limiting Concurrent Busy Banks

In this technique, the memory is slowed down during power consumption spikes by temporarily limiting the number of memory banks that can be busy at the same time. To support this technique, *memorybusy* and *memoryrequest* signals from all the memory banks are routed to a central module that controls the degree of concurrency. If number of busy banks reaches a certain *AllowedBusy* threshold set for the time interval T, all the banks are disabled from servicing new requests. This state continues until some banks finish their transactions and the number of busy banks falls below *AllowedBusy*. Then, the memory banks can be enabled again. However, enabling all the banks, an avalanche of request could be produced. Consequently, knowing what banks have incoming requests, the controller enables only a few banks, ensuring that no more than *AllowedBusy* banks are active at the same time.

The value of *AllowedBusy* for an interval T is set as follows. In each interval, the maximum number of banks that are busy concurrently is recorded in register $MaxBusy$. This value, multiplied by *PowerRatio* becomes the *AllowedBusy* for the next time interval T.

Note that, in a real implementation, the central controller described will be quite hardware intensive. It will be challenging to make it function at the high frequencies assumed for the logic. In this thesis, an ideal implementation is assumed to see the potential of this power-limiting technique.

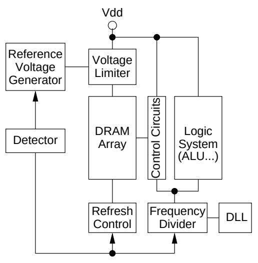


Figure 4.1: Simple circuit to control the DRAM array voltage and the chip frequency.

4.5 Reducing the Voltage

Another approach to reduce power consumption is to temporarily lower the operating voltage of the DRAM array when a spike is detected. Changes to the voltage have to be managed carefully because they can cause unexpected malfunctioning in the memory control circuits and the logic systems in the chip. The reason is the non-linear variation of transistor characteristics with voltage changes. This technique assumes to lower the voltage only in the DRAM array. When $P_{measure}$ in a time interval T is larger than P_{limit}, the voltage is changed in the next interval from the nominal value to the reduced value. Conversely, when $P_{measure}$ is less than 90% of P_{limit}, the voltage is restored for the next interval. As usual, some hysteresis is allowed to prevent oscillations.

The voltage in the DRAM array can be controlled by changing the reference voltage used in an on-chip voltage converter [21] according to the outputs of the detector as shown Figure 4.1. Detector outputs also need to change DRAM refresh intervals. Overall, this technique minimizes the increase in the energy-delay product for the application.

4.6 Reducing the Frequency

Finally, a coarse-grained and simple technique to reduce spikes of energy is to temporarily reduce the frequency of the chip when the spike is detected. As usual, when $P_{measure}$ in a time interval T is larger than P_{limit}, the frequency is changed in the next interval from the nominal value to the reduced one. Conversely, the frequency is restored when $P_{measure}$ is less than 50% of P_{limit}. Such a large hysteresis is used because the frequency is divided and multiplied by two, therefore dividing and multiplying the power consumption by two too.

Chip frequency can be divided by an integer according to the outputs of the detector shown Figure 4.1-(b). The DLL core is operating at the same frequency as the main PLL, while the output signals are sampled and used at divided frequencies when desired.

Chapter 5

Evaluation Environment

The memory hierarchy organizations of Chapters 3 and 4 are evaluated with detailed software simulations at the architectural level. The simulations are performed using a MINT-based [40] execution-driven simulation system that models superscalar processors and detailed memory systems [28]. With this simulator, a variety of intelligent memory chip architectures are evaluated running a set of memory-intensive applications.

5.1 Architecture Modeled

A single intelligent-memory chip is modeled with 64 800-MHz simple on-chip processors and 64 Mbytes of DRAM. Each on-chip processor is associated with a 1 Mbyte DRAM bank. Table 5.1 shows some architectural parameters for a single memory bank and processor pair. This configuration is referred to as *baseline*. It corresponds to the *traditional* memory organization (Chapter 3.1).

The memory system simulator includes detailed description of buses, caches, row buffers and memory banks. The memory bus utilized by P.Host is a fast version of RAMBUS. A main processor L2 miss issues a request to the memory trough an 800MHz a dedicated command channel. The reply is performed through an 800MHz 16 bits data channel. The data bus provides 1.6GBytes/s, and allows pipelining of multiple requests. The minimum data request is 16 bytes.

The processors inside the memory chip have four possible L1 cache configurations. A small 256Bytes 4 way associative, a medium size 1K 4 way, a 8K 2 way associative, and a 16K 2 way associative. All the processors have 1 cycle cache access time. The cache has a 256bit wide data bus from the Data Buffer.

Processor		Cache	Data Buffer	Row Buffer	DRAM Sub-Bank
Freq: 800/400 MHz (Nominal/Reduced)		Size: 8 KB	Number: 1	Number: 5	Number: 4
Issue Width: 2	BR Penalty: 2 cyc	Assoc: 2	Size: 256 b	Tot sz: 4 Kbytes	Num Cols: 4096
Int Units: 2	FP Units: 0	RTrip: 1.25 ns	Data Bus: 256 b	RTrip: 7.5 ns	RTrip: 15 ns
Ld/St Units: 1	Prefetching: No	Line: 32 B			Num Rows: 512
Pending Ld/St: 1/1	Static Issue: Yes				

Table 5.1: Parameters of the baseline architecture simulated for a single memory bank and processor pair. In the table, *BR* stands for branch and *RTrip* for contention-free round-trip latency from the processor.

Table 5.3 shows the parameters evaluated in this thesis. In the table, *Trad*, *S*, *SP*, *IS*, and *ISP* stand for the traditional, plain segmented, segmented pipelined, interleaved segmented and interleaved segmented pipelined memory configurations, respectively. As indicated in Chapter 3.1, the two numbers in parenthesis indicate the degree of interleaving and the number of sub-banks per data bus respectively. For the power reduction techniques, slowing down memory through clock gating (*CkGate*), extending the gating to processors and caches (*CkGate+*), slowing down cache hits (*SloHit*), slowing down memory through limiting the number of concurrent busy banks (*BusyBk*), reducing the DRAM supply voltage (*RedVol*), and reducing the frequency (*RedFreq*) are the techniques evaluated. The *RedVol* technique reduces the voltage from the nominal value (1.8V) to the reduced value (1.2 V). The memory access timings for different voltages are shown in Table 5.2. In the same way the *RedRreq* scheme changes the frequency from the nominal frequency (800MHz) to the reduced frequency (400MHz).

As described in Chapter 3.3, the energy consumption is computed by multiplying the number of instructions executed of each type for the average energy consumed for each instruction. The average energy is calculated by applying scaling-down theory to existing devices [10, 4]. The estimation includes the energy required to fetch the instruction (amortized over several instructions). For multiply and divide instructions,

Operation	1,8v	1.2v
X-address buffer	1	1
X-address decoder	1	1
Wordline enabling	2	2
Charge sharing	2	4
Bit line sensing	2	4
DRAM Data buffer	2	2
L1 cache	1	1
Total	11	15

Table 5.2: Memory Access Timings with two possible memory bank voltages. All the timings are in cycles at 800Mhz.

Processor Issue Width	1, 2
Hardware Prefetching	Yes, No
Cache Assoc & Size	4-Way 256 bytes, 4-Way 1 Kbytes, 2-Way 8 Kbytes, 2-Way 16 Kbytes
Memory Configuration	Trad(1,4), S(1,4), SP(1,4), IS(2,4), ISP(2,4), IS(2,8), ISP(2,8)
Power Reduction Technique	CkGate, CkGate+, SloHit, BusyBk, RedVol, RedFreq

Table 5.3: Parameters varied in the architecture.

this value is 210 pJ, while for the other, simpler instructions, the value is 81 pJ. Recall that the processor is simple and that its 28 16-bit instructions are optimized for intelligent memory operation [24]. The energy consumed by the clock in the whole chip is estimated to be 907 pJ per cycle assuming 1 main PLL [6] and 16 distributed local DLLs [38] with meshed clock signal routing [24]. The calculation is performed for 800 MHz and 1.8 V and extended for different voltage and frequency as required by schemes described in Chapter 4.

Operation performed	256B	1K	8K	16K
Read hit	158	163	191	236
Read miss	8	8	4	4
Write hit	70	86	212	376
Write miss	8	8	4	4
Line fill	31	31	31	31
Prefetch tag check	8	8	4	4

Table 5.4: Cache Energy consumption.

The energy required for different cache operations is shown in Table 5.4.

Table 5.5 is the complete spreadsheet with the used values. For the experiments of Chapter 6.2, the sustained maximum power consumption in the chip is limited to 12 W (P_{limit}), and the power consumption sensor is sampled every 1.25 μs. P_{limit} may be a reasonable power budget for a memory chip with simple heat dissipation system.

Finally, the complete chip is modeled after a *FlexRAM* chip [24]. The on-chip processors are interconnected to each other in a ring. Each processor can see the DRAM in its left and right neighbors, and processors communicate through memory. As in *FlexRAM*, the chip contains an on-chip controller (*P.Mem*) that executes the limited serial sections in the applications, like initialization, broadcasts, and reduction operations [24]. P.Mem is an 800 MHz, 2-issue in-order processor with a 2-way set-associative 8 Kbytes cache. Its round-trip latency to the cache and to a row-buffer in memory is 1.25 ns and 8.5 ns respectively. Its involvement in the applications is modest: on average, it executes for about 8% of the application time, and is usually running at isolated periods of time.

Although its real impact is small, its contribution is present in the execution time and energy dissipation numbers presented.

Operation	Trad(1,4)		S(1,4)		SP(1,4)		IS(2,4)		ISP(2,4)		IS(2,8)		ISP(2,8)	
performed	1.8v	1.2v	1.8v	1.2v	1.8v	1.2v	1.8v	1.2v	1.8v	1.2v	1.8v	1.2v	1.8v	1.2v
Read Hit	431	431	431	431	500	500	469	469	538	538	480	480	549	549
Read Miss	6962	4825	3702	2633	3702	2633	2250	1667	2250	1667	1519	1194	1519	1194
Write Hit CA	541	541	541	541	541	541	308	308	308	308	249	249	249	249
Write Hit CNA	5253	2513	2739	1369	2739	1369	1516	782	1516	782	965	548	965	548
Write Miss	6245	3007	3285	1616	3285	1616	1813	917	1813	917	1136	628	1136	628
Send data	265	265	265	265	265	265	344	344	344	344	344	344	344	344

Table 5.5: Row Buffer energy consumption values for P.Arrays. CA stands for Columns Active, CNA stands for Columns No Active. All the units are in pico Joules (pJ)

5.2 Applications Evaluated

The different techniques are evaluated with 6 memory-intensive, highly-parallel applications that are suitable for intelligent memory. They have been parallelized in threads by hand. All the benchmarks generate several millions of data references. Table 5.6 shows some of their characteristics.

GTree is a data mining application that generates a decision tree given a collection of records to classify [35]. The records are distributed across the processors. Periodically, the P.Mem decides what attributes should split the tree and tells the processors what branch they should examine next. The processors then process their records.

DTree uses the tree generated in *GTree* to classify a database of records. Each processor has a copy of the decision tree and a portion of the database. The P.Array processes sequentially its local records. At the end, the results are accumulated by the P.Mem.

BSOM is a neural network that classifies data [29]. Each processor processes a portion of the input, updates the local weight and synchronizes with the P.Mem. The P.Mem combines the partial results and sends them again to the processors. The original algorithm uses floating point but, since processors do not support it, the algorithm has been converted to use fixed point.

BLAST is a protein matching algorithm [1]. The goal is to match an amino acid sequence sample against a large database of proteins. Each processor keeps a portion of the database and tries to match the sample against it. The P.Mem gathers result in the end.

Mpeg performs MPEG-2 motion estimation. The reference image and the working image are distributed across the processors. Each 8x8 block in the working image is compared against the reference image.

FIC is a fractal image compression algorithm [11]. The algorithm developed by Yuval Fisher encodes a image using a fractal scheme with a quadtree partition. Each processor has a portion of the image and some calculated characteristics. Processors perform a local transformation to their portion of the picture. This operation can cause significant load imbalance.

Applic.	What It Does	Problem Size	Cache Hit (%)
GTree	Tree Generation	5 MB database, 77.9 K records, 29 attributes/record.	50.7
DTree	Tree Deployment	1.5 MB database, 17.4 K records, 29 attributes/record.	98.6
BSOM	BSOM Neural Network	2 K inputs, 104 dimensions, 2 iter, 16-node network, 832 KB network.	94.7
BLAST	BLAST Protein Matching	12.3 K sequences, 4.1 MB total, 1 query of 317 bytes.	96.9
Mpeg	MPEG-2 Motion Estimation	1 1024x256-pixel frame plus a reference frame. Total 512 KB.	99.9
FIC	Fractal Image Compressor	1 512x512-pixel, plus a 4 512x512 internal data	97.8

Table 5.6: Characteristics of the applications in the *baseline* defined in Table 5.1.

Chapter 6

Evaluation

This chapter evaluates the different memory hierarchies for intelligent memories proposed in Chapter 3 and 4. First, an evaluation of which organizations are best under different metrics. Then, an evaluation of the proposed schemes to limit the power consumption.

6.1 Optimization Analysis

The best memory hierarchy organization depends on the metric to optimize. This chapter considers several metrics, namely maximizing performance, minimizing energy-delay product, minimizing area-delay product, and absolute power consumption.

Since several parameters are studied, The analysis is systematically organized assuming the *baseline* architecture defined in table 5.1. The effect of the memory bank organization, the cache size, the data prefetching activation, and the issue width are examined.

6.1.1 Maximizing Performance

Some designs try to deliver the maximum possible performance without considering power or area consumption. Of course both parameters should have feasible values. The average

IPC delivered by P.Array processors for the duration of their execution is measured to identify the highest-performing design.

First, the memory bank organization effect is analyzed. Figure 6.1 shows the IPC of the applications running on the baseline architecture for different memory bank organizations. An *Average* line has been added to simplify the analysis. Charts (a) and (b) correspond to systems with 1 and 8 Kbytes caches respectively. The memory organizations are ordered from the simpler ones on the left side to the more sophisticated ones on the right side. The rightest point corresponds to a perfect memory system, where all memory subsystem accesses are satisfied in 1 cycle.

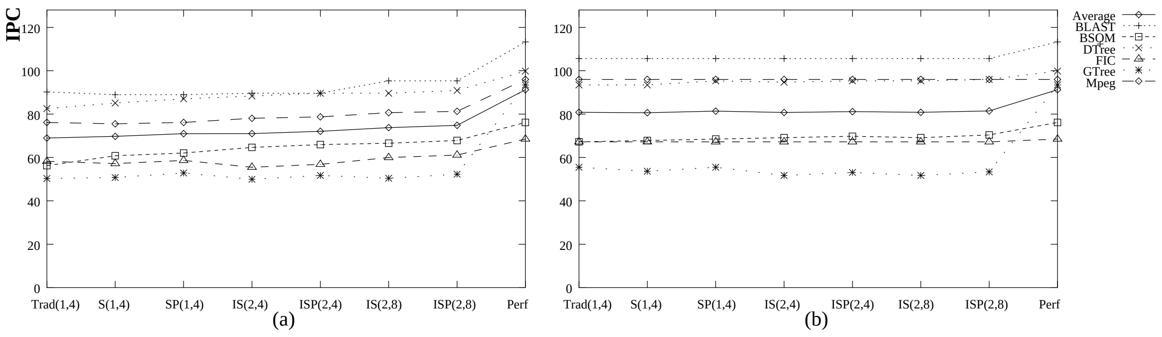


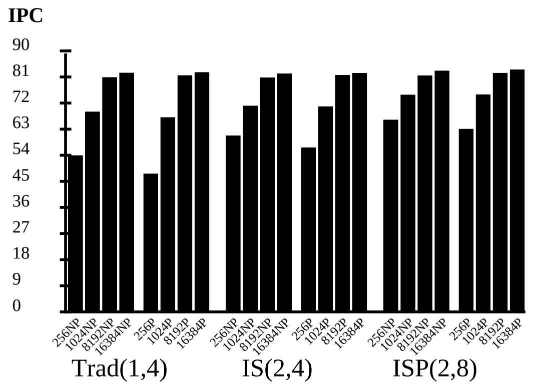
Figure 6.1: Effect of the memory bank organization on the IPC.

On average, in figures 6.1-(a) and (b), the performance keeps increasing when moving from the left to the right, from simple to more sophisticated configurations. *GTree* is special, it wants large prefetch (*Trad(1,4)*,*S(1,4)*,*SP(1,4)*), mainly for its stride characteristic. The largest boosts in IPC occurs when more sub-banks are added to the memory, going from (2,4) to (2,8), and adding pipelining going from *S(1,4)* to *SP(1,4)*.

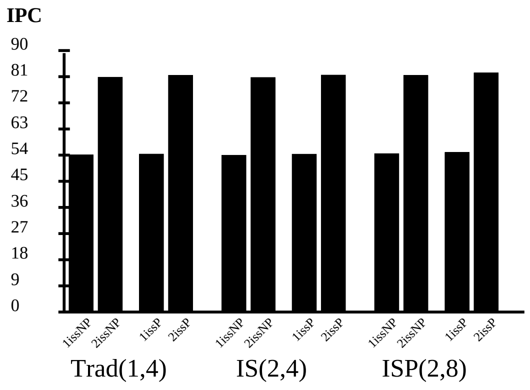
In Figure 6.1-(a), all the applications but *BSOM* have less than 12% improvement between *Trad(1,4)* and *ISP(2,8)*. The curves in Figure 6.1-(b) change little with different memory bank organizations. This is because, for 8 Kbytes caches, there are relatively few cache misses and, as a result, the performance is less sensitive to the type of memory organization. The ratio between *Perf* and *ISP(2,8)* shows the percentage of memory time. All the applications but *GTree* have less than 10% memory time. Only in *GTree* the memory time represents more than 40% of all the execution time.

Three memory organizations are studied to see the cache size and prefetching effect. A conservative one (*Trad(1,4)*), a medium class (*IS(2,4)*), and an aggressive one (*ISP(2,8)*). All three configurations have a two-issue processor, only varies the cache size and prefetching support.

The resulting IPCs for the average of all programs is shown in Figure 6.2-(a). As indicated in Chapter 5, caches of size 256 bytes, 1 Kbyte, 8 Kbytes, and 16 Kbytes are analyzed. All the caches are analyzed with and without prefetching support, P and NP respectively .



(a)



(b)

Figure 6.2: Effect of the cache size (a), prefetching (a and b), and processor issue width (b) on the IPC.

The data shows that, as expected, the best performance is achieved with the largest cache. Recall that, for these small cache sizes, there is no difference in cache speed. Prefetching, on the other hand, has barely any impact on performance. Part of the problem is that the memory latencies that prefetching is supposed to hide are small. If anything, prefetching degrades IPC for small caches, especially in the conservative memory organization (*Tra(1,4)*). The reason is pollution of the row buffers and cache. In only one application, *BSOM*, prefetching has a very positive effect, where performance increases by 8% across a wide range of configurations. Therefore, prefetching could be enabled on an application basis. Finally, in other, pointer-chasing applications, a more aggressive form of prefetching may be beneficial. The congestion introduced by aggressive prefetch is more tolerated with advanced memory systems like *ISP(2,8)*.

Figure 6.2-(b) shows the average IPC for all the programs. The bars use three memory bank organizations with 8 Kbytes caches. The parameters studied are the issue width (*1ISS* and *2ISS*), and the prefetching activation (*NP*) and (*P*). The chart shows that these codes benefit from increasing the issue width from one to two. The resulting IPC goes up 50%.

6.1.2 Power Consumption

Power consumption is a very important metric, not to minimize the value but to fulfill design constraints. Some battery supplied systems have strong constraints. While battery capacity is increased 5% yearly [19], the processor power consumption increases at higher rate. Also a high power consumption could burn the chip.

The contribution in energy consumption includes the processor and the memory system. Figures 6.3-(a) and 6.3-(b) use the same configurations than Figures 6.1-(a) and 6.1-(b). The memory organization has a strong effect on the power dissipated in the application. The average power tends to decrease as the memory system becomes more sophisticated. The reductions are large, in the order of several watts, especially in *BSOM* and *DTree*. *FIC* has a lower consumption because the P.Arrays finish the job gradually. To decrease the power consumed, the best strategies include segmentation (from *Trad(1,4)* to *S(1,4)*), interleaving (from *SP(1,4)* to *IS(2,4)*), and adding extra sub-banks (from *(2,4)* to *(2,8)*). This is consistent with what was discussed in Chapter 3.3.

Figures 6.4-(a) and 6.4-(b) correspond to the same environments as Figures 6.2-(a) and 6.2-(b). Since accessing the memory in *Trad(1,4)* is quite expensive, increasing the cache size, reduces the average power consumption is reduced. Something different happens with *IS(2,4)* and *ISP(2,8)*. On average require more power when the increase in the cache power consumption does not compensate the increase in the hit rate. *FIC*, *DTree*, *GTree* have a sweet point with 8192 caches, *BLAST* and *BSOM* consume less

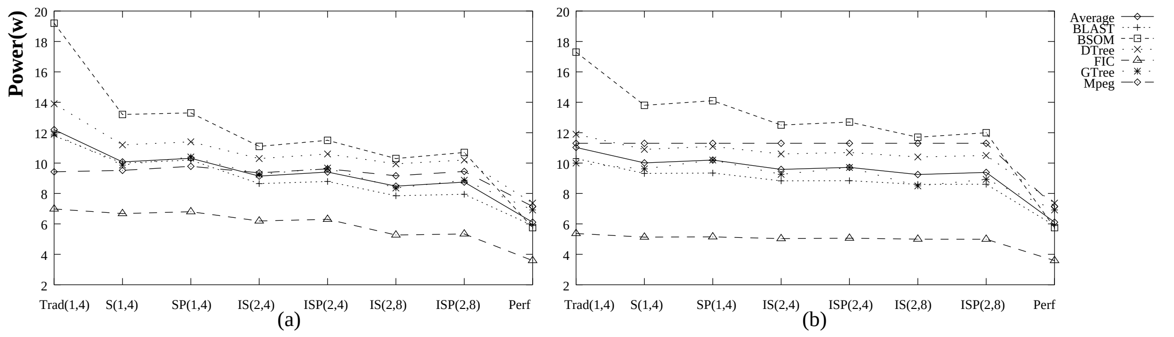


Figure 6.3: Effect of the memory bank organization on the power consumed.

energy with 1024 caches, the most spectacular case is *Mpeg* requires less power with 256 caches.

From Figure 6.4-(a) shows that prefetching uniformly decreases the power consumed across all configurations. This counter intuitive result is produced by BSOM which improves with prefetch.

The reason is that although applications consume more Energy, the IPC is slightly degraded. The ratio between energy and execution time is decreased with prefetch. Then although the power consumption is reduced the activation of prefetch is not a good idea.

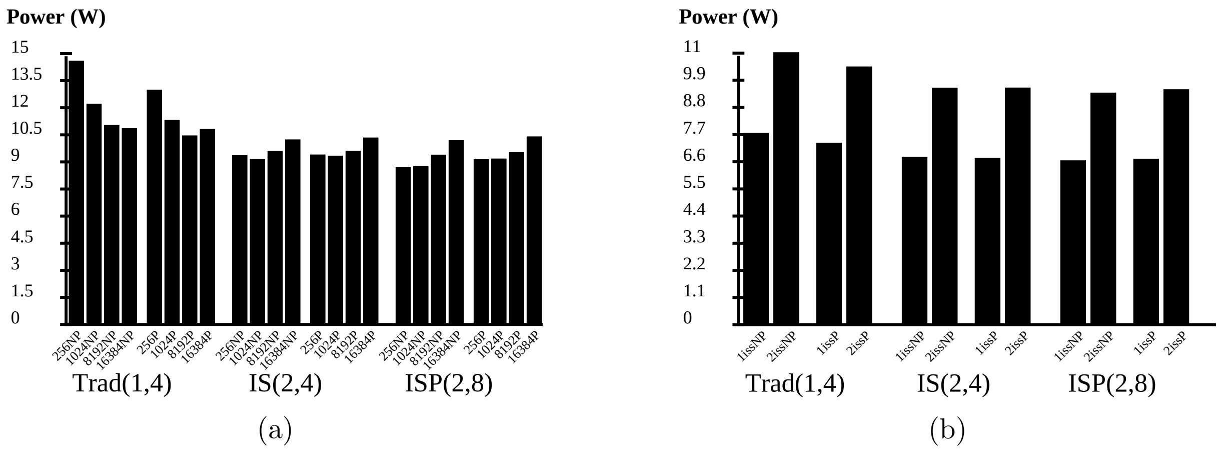


Figure 6.4: Effect of the cache size (a), prefetching (a and b), and processor issue width (b) on the power consumption.

Finally, Figure 6.4-(b) shows that the power consumption of the 2-issue system is higher than the 1-issue system for all memory organizations. However, the relative increase is smaller than the relative increase in IPC (Figure 6.2-(b)). The reason is that the 2-issue consumes less overall energy to finish the program than the 1-issue system. As indicated by Horowitz et al [13], parallelism is a good way of saving energy. Overall, the 2-issue systems consume 40% less energy than the 1-issue systems.

6.1.3 Minimizing Energy-Delay Product

The power consumption by itself is a bad metric for choosing between systems. A processor that does nothing have a zero power consumption. In many cases, the best figure of merit for embedded systems is the energy-delay product [14]. A low product implies that the system is both fast and energy-efficient. This section measures the energy-delay product for the same memory hierarchy configurations of the previous two sections. We add the contribution of both processor and memory system.

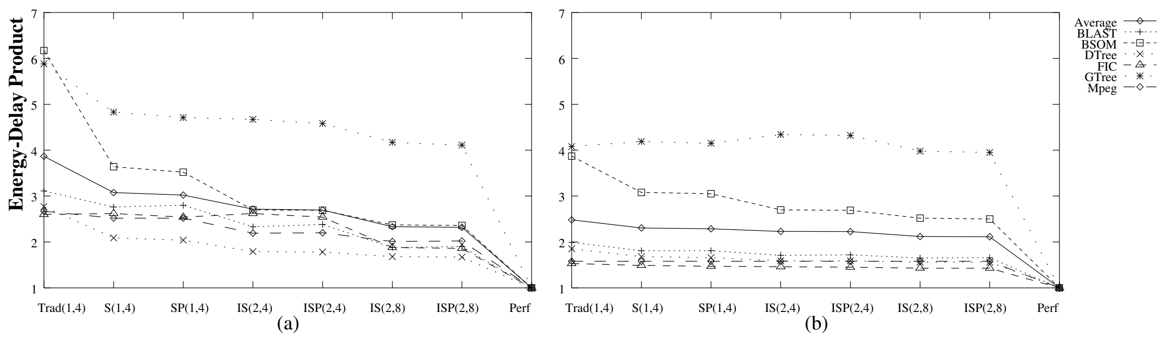
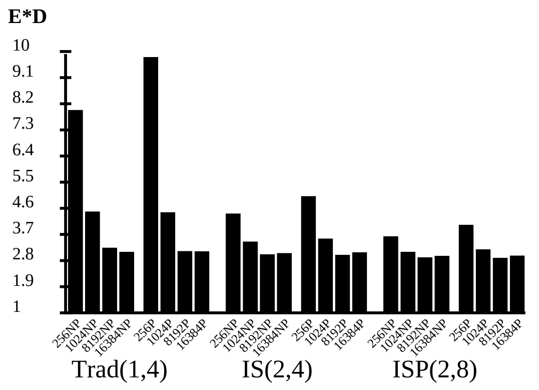


Figure 6.5: Effect of the memory bank organization on the energy-delay product.

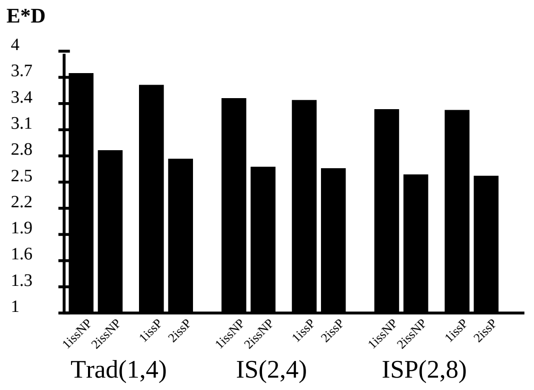
Figures 6.5-(a) and 6.5-(b) correspond to the same environments as Figures 6.1-(a) and 6.1-(b). The charts are normalized to *Perf* for all applications. The *Perf* configuration always has a cache hit, and zero energy is required for a Data Cache hit. This implies that all the memory operations require only one cycle, and the energy is spent only in the instruction cache, the processor and the clock.

The new charts are quite different than the IPC charts. While the IPC varies little across memory organizations, the average energy-delay product changes by over 40% across memory organizations with in 6.5-(a). Simpler memory systems have larger energy-delay product than advanced configurations. The reason is that, traditional systems slightly degrade IPC, but they consume more energy in the process. Consequently, it is desirable to support segmentation and interleaving. Although less, increasing the number of sub-banks from (2,4) to (2,8) reduces the energy-delay product.

Figures 6.6-(a) and 6.6-(b) correspond to the same environments as Figures 6.2-(a) and 6.2-(b). The bars in each chart are normalized to the *Perf* case. Looking at the effect of cache sizes, the configuration with 8 Kbytes caches has the best energy-delay product. Caches have a two-barreled effect: they speed up the program and, in addition, reduce the energy consumption because they eliminate energy-consuming memory accesses. From the figure, *Trad(1,4)* reduces the energy-delay product in more than 75% when the cache increases from 256 to 8 Kbytes. If it is affordable the approximately 10% increase in total memory bank area required, going from 1 to 8 Kbytes of cache is a good tradeoff.



(a)



(b)

Figure 6.6: Effect of the cache size (a), prefetching (a and b), and processor issue width (b) on the product energy-delay product of architectures.

Figure 6.6-(a) proves that hardware prefetching, for small latencies is not a desirable technique. Finally, Figure 6.6-(b) shows that, across all memory configurations, a 2-issue processor system improves the energy-delay product. A 2-issue processor reduces the clock energy and the keeps the memory energy. Since the performance is also improved the energy-delay product is reduced by a 25% on average.

6.1.4 Minimizing Area-Delay Product

A final possible metric of good design is the transistor utilization, which can be defined as inversely proportional to the area-delay product. The processor and memory system is considered to calculate the area.

Figures 6.7-(a) and 6.7-(b) correspond to the same environments as Figures 6.5-(a) and 6.5-(b). The *Perf* configuration has zero memory area, only P.Array area. To understand the charts, recall that the greatest area increase comes from adding extra row buffers for interleave memory (*SP(1,4)* to *IS(2,4)*), or add more sub-banks (*IS(2,4)* to *IS(2,8)*).

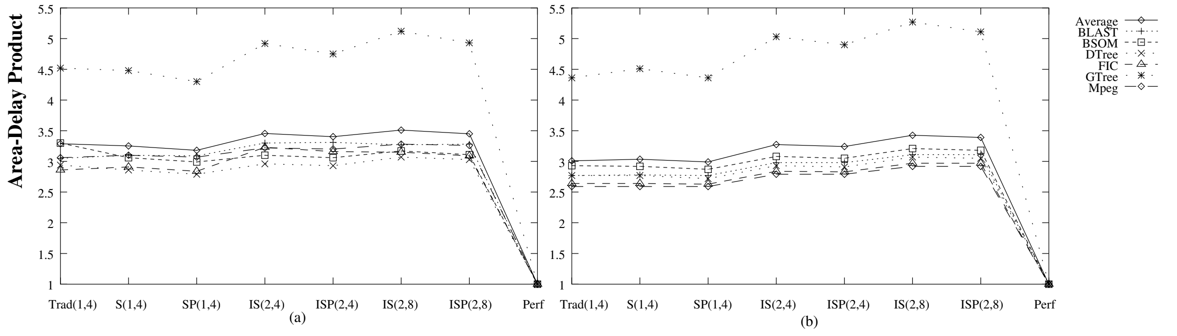
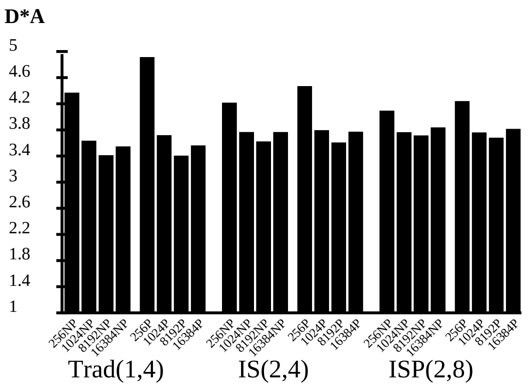
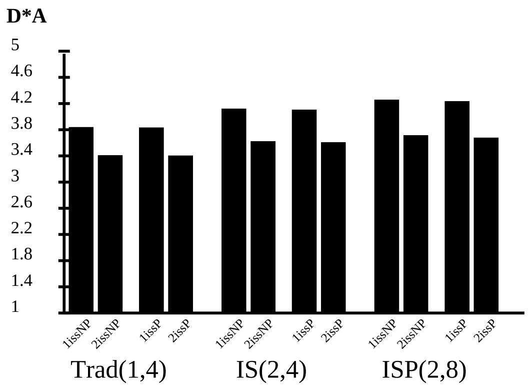


Figure 6.7: Effect of the memory bank organization on the area-delay product.

When the cache is large enough it masks most of the performance impact of changing the memory organization (Figure 6.7-(b)). The chart clearly shows two steps. Interleaving increases the product by 8%, while going all the way to *IS(2,8)* results in an overall 14% higher product. If, instead, the cache is small (Figure 6.7-(a)), the chart is more confused, but the product usually increases with fancier memories. Since pipelining improves the performance without increasing area, it is always better to have pipelining. Overall, if only modest values of area-delay product can be tolerated, *SP(1,4)* and *ISP(2,4)* organizations have the best tradeoffs.



(a)



(b)

Figure 6.8: Effect of the cache size (a), prefetching (a and b), and processor issue width (b) on the area-delay product of architectures.

Figures 6.8-(a) and 6.8-(b) correspond to the same environments as Figures 6.6-(a) and 6.6-(b). Once again, the organizations with 8 Kbytes caches are the best, although the organizations with 1 Kbytes caches are nearly as good. The 16 Kbytes cache increases the area-delay product in all the configurations. The 2-issue processor organizations use the transistors about 10% better than the 1-issue processor organizations. Prefetching makes little difference.

6.1.5 Overall Summary

Instead of just examining the IPC, less traditional energy-delay and area-delay products have been considered. The resulting pictures indicates that sizable caches (up to 8 Kbytes) are desirable: they speed up the application, are energy-efficient, and consume modest area. For the memory organization, interleaved segmented is highly attractive, a more advance solution would also include pipelining. Increasing the number of sub-banks from (2,4) to (2,8) is less attractive because it increases the area by 8% for modest gains. The pipelining improvement is more significative with no-interleaved configurations. Prefetching is unnecessary, while 2-issue support is good for both performance and energy efficiency.

6.2 Limiting Power Consumption

In this chapter, the schemes for limiting the maximum power dissipation proposed in Chapter 4 are evaluated. As indicated in Chapter 5, the thesis assumes an arbitrarily set the maximum sustained power dissipation threshold for the chip (P_{limit}) to $12W$. While different thresholds may change the conclusions quantitatively, they are unlikely to change them much qualitatively. In the experiments, the power consumption sensor is sampled every 1,000 cycles (1.25μs at the 800 MHz operation assumed).

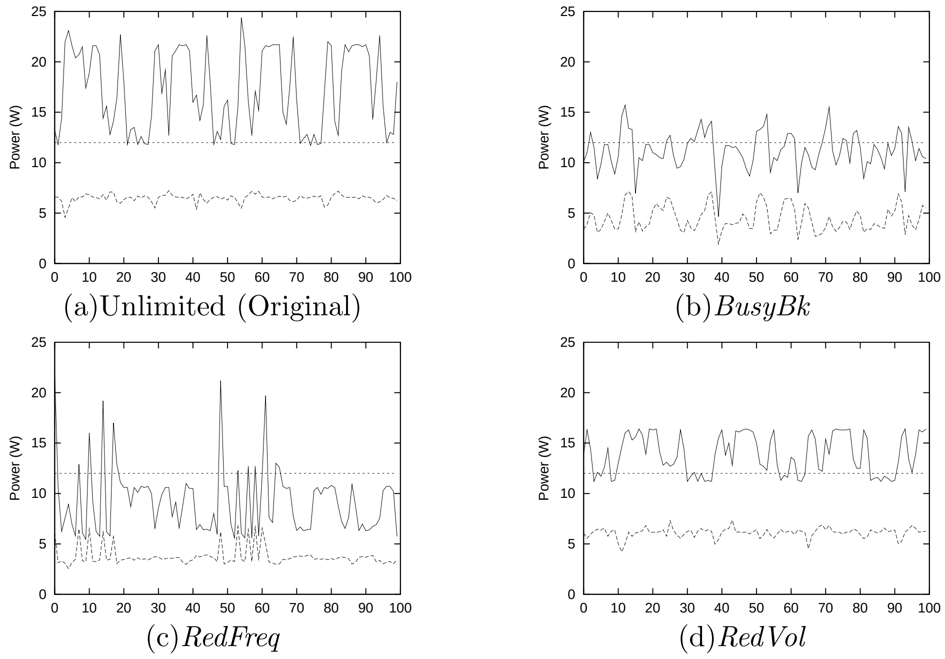


Figure 6.9: Variation of the power dissipated in an intelligent memory chip as time advances for different real-time power-limiting techniques: no power limitation (a), limiting the number of concurrent busy banks (b), reducing the chip frequency (c), and reducing the DRAM array voltage (d).

Two experiments are performed. First, a graphical examination if the techniques proposed can minimize the thick spikes of power consumption observed in the intelligent memory chip. Second, an evaluation of the different schemes under several metrics.

To understand if the techniques proposed can be effective in limiting power consumption, the variation of the power consumption is sampled as one application runs on the intelligent memory. The application chosen is *BSOM*. It is simulated running on the baseline architecture with the *Trad(1,4)* memory system. Figure 6.9 shows the results. The figure has four charts, which correspond to a system with no power limitation, and the same system with three power-limiting techniques: limiting the number of concurrent busy banks (*BusyBk*), reducing the frequency (*RedFreq*), and reducing the voltage (*RedVol*). In each chart for the Figure 6.9, the upper line indicates the total power consumption in the chip, the middle line is set to P_{limit} (12 W), and the lower one is the power consumption in the processors only.

An examination of the Figure 6.9-(a) shows that without any power-limiting technique, the intelligent memory architecture analyzed suffers thick, frequent spikes of power consumption. Furthermore, a large fraction of this power consumption comes from the memory subsystem, as shown by the difference between the upper and lower curves.

Looking at the other charts, all the techniques can significantly curve the power consumption. However, they have different effectiveness. *RedFreq* and *RedVol* are the two extreme schemes. *RedFreq* eliminates most of the over-the-limit power consumption and allows only very fine spikes. However, it forces the chip to work in very low-performance mode. In this case, it increases the execution time of the application by about 80%, compared to about 20% in other schemes.

RedVol is attractive because it reduces the energy-delay product by about 2%. However, it leaves frequent, thick spikes over P_{limit}. Most of the time the power is still over limit. The other scheme, *BusyBk*, shows a good behavior. It eliminates most of the spikes and leaves the power dissipation very close to P_{limit} throughout the whole execution. The other schemes proposed (*SloHit*, *CkGate+*, and *CkGate*), although not shown in the figure, have a behavior similar to *BusyBk*. The conclusion is that the proposed schemes work reasonably well.

To gain a better insight of how all these techniques compare to each other, a second experiment is performed. The baseline architecture of Chapter 5 is used with the memory organization that gives the best energy-delay product (*ISP(2,8)*) and run all the applications for each power-limiting technique. For each run, five parameters are measured: execution time of the application (*Delay*), energy consumed, energy-delay product (*E*D*), and two metrics of how higher is the resulting power consumption over the allowed sustained maximum P_{limit}. These two metrics are the two moments M_1 and M_2 of excessive power dissipation. Calling P_i the power consumed in a sample and n the number of samples, the definition of the moments is:

$$M_1 = \frac{1}{n}\sum_{i=1}^{n} \frac{P_i - P_{limit}}{P_{limit}}\ \bar{\delta}(i)$$

$$M_2 = \frac{1}{n}\sum_{i=1}^{n} (\frac{P_i - P_{limit}}{P_{limit}})^2\ \bar{\delta}(i)$$

$$\bar{\delta}(i) = \begin{cases} 1 & \text{when } P_i > P_{limit} \\ 0 & \text{otherwise .} \end{cases}$$

The first moment M_1 shows the average magnitude of the power spikes. Since high spikes are particularly bad, the second moment M_2 is very useful too. It is interesting to minimize M_1 and, especially, M_2 without penalizing *E*D* much. The measured parameters for the 6 applications are shown in Table 6.1. The values in the table are normalized to the system with no power limitation (*Unlimit*).

Two schemes that are different from the others. *RedVol* slows down the system very little, and in fact decreases the energy-delay product. However, it is not able to decrease the spikes, as shown by the high M_1 and M_2 values. *RedFreq* is the opposite case: it reduces the M_1 and M_2 values greatly at the expense of slowing down the system significantly.

Comparing *BusyBk* to *CkGate*, *CkGate* controls the power spike problem better than *BusyBk*, but slows down the execution more. Since these techniques more or less keep the energy consumption the same, the energy-delay product is increased proportionally. Extending the gating of the clock to the processor (*CkGate+*) only increases program execution time marginally over *CkGate*, but manages to contain the power spikes much better than *BusyBk* and *CkGate*. Consequently, *CkGate+* is a very attractive scheme.

Scheme	Benchmark	Delay	Energy	E*D	M_1	M_2
Unlimit		1.000	1.000	1.000	1.000	1.000
CkGate	*BLAST*	1.003	1.000	1.003	0.615	0.602
	BSOM	1.140	1.000	1.140	0.206	0.147
	DTtree	1.017	1.001	1.018	0.041	0.027
	GTtree	1.000	1.000	1.000	0.240	0.122
	FIC	1.178	1.011	1.191	0.127	0.058
	Mpeg	1.026	1.003	1.029	0.910	0.888
	Average	**1.061**	**1.003**	**1.064**	**0.357**	**0.308**
CkGate+	*BLAST*	1.005	1.000	1.005	0.020	0.009
	BSOM	1.143	1.008	1.152	0.194	0.140
	DTtree	1.025	1.002	1.027	0.010	0.010
	GTtree	1.000	1.000	1.000	0.162	0.087
	FIC	1.194	1.016	1.213	0.028	0.009
	Mpeg	1.234	1.021	1.259	0.003	0.000
	Average	**1.100**	**1.008**	**1.109**	**0.070**	**0.043**
SloHit	*BLAST*	1.007	1.000	1.007	0.035	0.011
	BSOM	1.195	1.008	1.205	0.161	0.116
	DTtree	1.026	1.001	1.027	0.164	0.067
	GTtree	1.000	1.000	1.000	0.116	0.086
	FIC	1.287	1.029	1.324	0.002	0.007
	Mpeg	1.272	1.024	1.302	0.010	0.001
	Average	**1.131**	**1.010**	**1.144**	**0.082**	**0.048**
BusyBk	*BLAST*	1.010	1.000	1.010	0.443	0.400
	BSOM	1.177	1.008	1.187	0.196	0.121
	DTtree	1.034	1.002	1.036	0.116	0.088
	GTtree	1.000	1.000	1.000	0.723	0.602
	FIC	1.022	1.000	1.022	0.903	0.794
	Mpeg	1.000	1.000	1.000	0.997	0.995
	Average	**1.041**	**1.002**	**1.043**	**0.564**	**0.500**
RedVol	*BLAST*	1.002	0.997	1.000	0.846	0.748
	BSOM	1.016	0.966	0.982	0.720	0.551
	DTtree	1.002	0.997	1.000	0.432	0.213
	GTtree	1.005	1.000	1.005	0.550	0.535
	FIC	1.015	0.992	1.007	0.890	0.789
	Mpeg	1.000	1.000	1.000	0.999	0.998
	Average	**1.007**	**0.993**	**0.999**	**0.740**	**0.639**
RedFreq	*BLAST*	1.706	1.052	1.794	0.005	0.004
	BSOM	1.792	1.041	1.865	0.050	0.048
	DTtree	1.140	1.012	1.155	0.007	0.008
	GTtree	1.009	1.001	1.010	0.373	0.190
	FIC	1.000	1.000	1.000	0.002	0.003
	Mpeg	1.892	1.074	2.032	0.000	0.000
	Average	**1.423**	**1.030**	**1.476**	**0.073**	**0.042**

Table 6.1: Evaluation of the different power-limiting schemes for all applications. *Unlimit* corresponds to the system without power limitation.

Finally, *SloHit* is less efficient than *CkGate*: it results in slightly higher energy-delay product and slightly less power control. However, it is very simple. Consequently, it may also be a technique of choice. Overall, *CkGate+* and *SloHit* seem to be the most attractive ones. However, in a real chip implementation, the designer can select one or a combination of the schemes according to the desired emphasis.

Chapter 7

Conclusions

Two issues motivated the interest in the memory hierarchy of a multi-banked intelligent memory chip. Firstly, advances in MLD technology exacerbate the mismatch between high-speed on-chip processors and slow, high-latency DRAM. Secondly, the presence of so many on-chip processors, all possibly accessing memory, creates thick spikes of energy consumption in these chips.

The results of the analysis indicate that, to minimize energy-delay product, each bank should include a sizable cache of about 8 Kbytes, and support segmentation and interleaving, and optionally pipelining. Given the memory latencies involved, hardware prefetching is unnecessary, while 2-issue support is good for both performance and energy efficiency. Furthermore, these chips consume a sizable fraction of the energy in the memory, and that this consumption is often spiky. This thesis analyzes simple techniques that effectively limit excessive power consumption using real-time corrective support. On average, gating the clock to both the memory system and the processor gives very good power consumption limit while keeping performance degradation at a minimum. This technique also delivers stable control across different applications.

Bibliography

[1] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic Local Alignment Search Tool. In *Journal of Molecular Biology*, pages 403–410, 1990.

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

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

[4] A. Inoue et al. A 4.1ns compact 54x54b multiplier utilizing sign select booth encoders. In *ISSCC Digest of Technical Papers*, pages 416–417, February 1997.

[5] H. Sanchez et. al. Thermal management system for high performance powerpc microprocessor. Technical report, Motorola Inc. and Apple Computer Corporation, 1998.

[6] J. Alvarez et al. A wide-bandwidth low-voltage pll for powerpc microprocessors. *IEEE J. of Solid-state Circuits*, 30(4):383–391, April 1995.

[7] M. Oskin et. al. Exploiting ilp in page-based intelligent memory. In *Proceedings of the International Symposium on Microarchitecture*, 1999.

[8] R. Fromm et. al. The Energy Efficiency of IRAM Architectures. In *Proceedings of the 24th Annual International Symposium on Computer Architecture (ISCA)*, pages 327–337, June 1997.

[9] S. Rixner et. al. A Bandwidth-Efficient Architecture for Media Processing. In *31st International Symposium on Microarchitecture*, November 1998.

[10] Yeung. N.K et al. The design of a 55specint92 risc processor under 2w. *ISSCC Digest of Technical Papers*, pages 206–207, February 1994.

[11] Yuval Fisher. *Fractal Image Compression: Theory and Application.* Springer Verlag, 1995.

[12] K. Ghose and M. Kamble. Reducing Power in Superscalar Processor Caches Using Subbanking, Multiple Line Buffers and Bit-Line Segmentation. In *Proceedings of the International Symposium on Low-Power Electronics and Design*, 1999.

[13] Ricardo Gonzales and Mark Horowitz. Energy dissipation in general purpose processors. In *IEEE International Symposium on Low Power Electronics*, pages 12–13, 1995.

[14] Ricardo Gonzales and Mark Horowitz. Energy dissipation in general purpose microprocessors. In *EEE Journal of Solid-State Circuits*, pages 277–1284, 1996.

[15] Mary Hall, Peter Kogge, Jeff Koller, Pedro Diniz, Jacqueline Chame, Jeff Draper, Jeff LaCoss, John Granacki, Jay Brockman, Apoorv Srivastava, William Athas, Vicent Freeh, Jaewook Shin, and Joonseok Park. Mapping irregular aplications to diva, a pim-based data-intensive architecture. In *Proceedings of Supercomputing 1999*, November 1999.

[16] P. Hicks, M. Walnock, and R. Owens. Analysis of Power Consumption in Memory Hierarchies. In *Proceedings of the 1997 International Symposium on Low-Power Electronics and Design*, page 239, 1997.

[17] IBM Corporation. http://www.chips.ibm.com/services/foundry/offerings/cmos/7ld. 1998.

[18] IBM Microelectronics. Blue Logic SA-27E ASIC. In News and Ideas of IBM Microelectronics, http://www.chips.ibm.com/ news/1999/sa27e/, February 1999.

[19] Intel Corporation. *Mobile Power Guidelines 2000, Rev 1.0*, 1998.

[20] Intel, Microsoft and Toshiba. *Advanced Configuration and Power Interface Specification*, 1999.

[21] K. Itoh et al. An Experimental 1Mb DRAM with On-Chip Voltage Limiter. In *ISSCC Digest of Technical Papers*, pages 84–85, Feb 1981.

[22] Subramanian S. Iyer and Howard L. Kalter. Embedded dram technology: opportunities and challenges. *IEEE Spectrum*, April 1999.

[23] M. Kamble and K. Ghose. Analytical Energy Dissipation Models for Low Power Caches. In *Proceedings of the International Symposium on Low-Power Electronics and Design*, 1997.

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

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

[26] J. Kin. The filter cache: An energy efficient memory structure. *IEEE International Symposium on Micro Architecture 30*, 1997.

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

[28] V. Krishnan et al. An Execution-Driven Framework for Fast and Accurate Simulation of Superscalar Processors. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, October 1998.

[29] R. Lawrence, G. Almasi, and H. Rushmeier. A Scalable Parallel Algorithm for Self-Organizing Maps with Applications to Sparse Data Mining Problems. Technical report, IBM, January 1998.

[30] Srilatha Manne, Artur Klauser, and Dirk Grunwald. Pipeline gating: Speculation control for energy reduction. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, 1998.

[31] M. Oskin, F. Chong, and T. Sherwood. Active Pages: A Computation Model for Intelligent Memory. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 192–203, June 1998.

[32] D. Patterson. ISTORE: Intelligent Store. Personal Communication, 1998.

[33] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Tomas, and K. Yelick. A Case for Intelligent DRAM. In *IEEE Micro*, pages 33–44, March/April 1997.

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

[35] J. R. Quinlan. *C4.5 - Programs for Machine Learning*. Morgan Kaufmann Inc., San Francisco, CA, 1993.

[36] E. Seevinck, P.Beers, and H.Ontrop. Current-mode techniques for high-speed vlsi circuits with application to current sense amplifier for cmos sram's. *IEEE Journal Solid State Circuits*, 26(4):525–536, April 1991.

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

[38] S. Sidiropoulos and M. Horowitz. A semidigital dual delay-locked loop. *IEEE Journal of Solid-state Circuits*, 32(11):1683–1692, November 1997.

[39] http://velox.stanford.edu/smart_memories. 1999.

[40] J. Veenstra and R. Fowler. MINT: A Front End for Efficient Simulation of Shared-Memory Multiprocessors. In *Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'94)*, pages 201–207, January 1994.

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

[42] J. H. Yoo et al. A 32-Bank 1Gb DRAM with 1GB/s Bandwidth. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 378–379, February 1996.