The SCALE DRAM Subsystem

by

Brian S. Pharris

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2004

Author .....
Department of Electrical Engineering and Computer Science
May 20, 2004

Certified by .....
Krste Asanović
Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

The SCALE DRAM Subsystem

by

Brian S. Pharris

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Dynamic Random Access Memory (DRAM) is consuming an ever-increasing portion of a system's energy budget as advances are made in low-power processors. In order to reduce these energy costs, modern DRAM chips implement low-power operating modes that significantly reduce energy consumption but introduce a performance penalty. This thesis discusses the design and evaluation of an energy-aware DRAM subsystem which leverages the power-saving features of modern DRAM chips while maintaining acceptable system performance. As this subsystem may employ a number of different system policies, the effect of each of these policies on system energy and performance is evaluated. The optimal overall policy configurations in terms of energy, delay, and energy-delay product are presented and evaluated. The configuration which minimizes the energy-delay product demonstrates average energy savings of 41.8% as compared to the high-performance configuration, while only introducing an 8.8% performance degradation.

Thesis Supervisor: Krste Asanović
Title: Associate Professor

Acknowledgments

I'd like to thank my advisor, Krste Asanović, for all his support and patience as I worked on this thesis. Whenever I was stuck, he always seemed to have an immediate and effective solution. As things became hectic towards the thesis deadline, he was extremely understanding and supportive.

I'd also like to thank Chris Batten for his invaluable assistance in testing and improving the SCALE DRAM simulator. His ample talent for nit-picking and exhaustive testing led to a far more robust and useful simulator than I would have produced otherwise.

Finally, I'd like to thank my beautiful fiancée Sarah for her incredible support and understanding throughout the process. I'm sure she found herself to be a thesis widow more often than she would have liked, but she never failed to encourage and support me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As advances are made in low-power processors, the portion of a system's energy budget due to the memory system becomes significant. It is therefore desirable to reduce memory energy consumption while maintaining acceptable performance.

To address this need for low-power memory systems, modern SDRAM chips may be powered down when not in use, dramatically reducing their power consumption. Reactivating a chip that has powered down, however, can introduce a significant performance penalty. DRAM systems must therefore effectively manage this energy-performance tradeoff.

Effective performance and energy management of a modern DRAM system is quite complex. As the system generally consists of a number of independent memory channels and SDRAM memory chips, the system controller must effectively map memory accesses into the appropriate hardware. These mapping policies can have a dramatic effect in the performance and energy consumption of the system.

In addition, modern DDR-II SDRAM chips implement a number of different power-down states, which offer varying degrees of energy savings at the cost of performance. An energy-aware memory system must therefore effectively manage the transitions of all chips between these different states. As different states are preferable in different circumstances, this can be a complex task.

This thesis addresses these issues in memory system design by demonstrating how a real-world DRAM subsystem may be designed to maximize performance while

minimizing the energy consumed. The thesis discusses the design of the energy-aware SCALE DRAM subsystem as well as the design and evaluation of various system policies.

SCALE (Software-Controlled Architectures for Low Energy) is an all-purpose programmable computing architecture. The SCALE DRAM subsystem is a hierarchically-structured, fully-pipelined DRAM system designed for use with the SCALE-0 processor. The SCALE-0 processor consists of a MIPS control processor and a 4-lane vector-thread unit[6]. The processor includes a unified 32-kB cache. All accesses to the SCALE DRAM subsystem are refills from and writebacks to this cache.

The SCALE DRAM subsystem is modularly designed such that system policies may be easily interchanged, allowing evaluation of many policies and implementation of the best policy for a given application. Although a number of different system policies can greatly influence the performance and energy characteristics of the system, this thesis focuses primarily on three policies: the address mapping policy, the access scheduling policy, and the powerdown scheduling policy. The address mapping policy determines how memory accesses are mapped into the system's hardware. The access scheduling policy determines the order in which outstanding requests are issued to a given SDRAM chip. Finally, the powerdown scheduling policy determines when a chip is to be powered down, and which of the various powerdown states it should enter.

The SCALE DRAM subsystem has been implemented both in hardware, on a programmable logic device connected to DDR SDRAM chips, and as a cycle-accurate software model. The software model allows rapid policy development and evaluation while the hardware implementation will provide real-world power measurements. The primary contribution of this thesis is the use of this simulator to develop and evaluate the system policies.

1.1 Related Work

Previous work in energy-aware memory controllers has focused primarily on power-down scheduling. Delaluz et al.[3] have demonstrated that in cache-less systems, a significant reduction in energy consumed can be achieved while maintaining good performance by dynamically scheduling chip powerdown. The controller may allow the chip to remain idle for a certain number of cycles, therefore, before powering down. This takes advantage of spatial locality in memory references: if a memory request accesses a certain chip, it is likely that a subsequent request will map to the same chip. The controller should therefore wait to power down the chip to avoid unnecessarily paying the performance penalty of reactivating the chip for a subsequent access.

For a system with a two-level cache, however, the cache filters out locality from external memory accesses. Fan et al. [5] have demonstrated that in such a system running standard performance benchmarks, the best energy-delay product is attained by simply powering down the chips as soon as possible.

However, for successive interleaved cache line accesses and for applications that do not cache well but may still exhibit chip locality (vector scatter-gather operations, for example), powerdown scheduling techniques are still valuable, as demonstrated in this document.

Delaluz et al.[3] augment the hardware power-down scheduling strategy with a scheduler in the operating system, which powers down and activates memory banks upon context switches. Every chip that will likely not be accessed by the current process is powered down. This approach has the advantage of reducing the hardware requirements, but the intervals between powerdown and activation are much coarser.

Lebeck et al.[7] also propose software augmentation of hardware power management techniques, focusing primarily on energy-aware page placement techniques. These techniques assume that pages exist on single chips, and so do not apply when accesses are interleaved across chips.

This earlier work, however, focuses on a small subset of computing tasks for mobile computing devices. Techniques employed by high-performance memory systems,

such as interleaving cache lines across multiple memory chips, are not explored. Additionally, high-performance systems including a vector unit will generate significantly different memory access patterns than the scalar processors used in these earlier studies. The aforementioned policies may therefore lead to very different results in such a system. In addition, DDR-II SDRAM chips can behave quite differently, both in terms of performance and in the relative energy-performance tradeoffs of powerdown, than the RDRAM chips used in these studies. This thesis therefore revisits some of these policies within the framework of the SCALE SDRAM system. As the software approaches discussed above rely on complementary hardware, this thesis focuses solely on hardware power-management strategies, with the assumption that software could enhance their effectiveness.

In addition to the aforementioned studies on powerdown scheduling, Rixner et al.[8] propose rescheduling memory accesses to improve DRAM bandwidth. They demonstrate that for certain streaming applications, aggressively rescheduling memory accesses can dramatically improve the DRAM bandwidth. Although this work only studied the performance implications of such rescheduling, it can also reduce energy consumption; as less time will be spent precharging and activating rows, the chip will be active for less time. Chips may therefore power down more quickly and conserve power. This thesis therefore revisits memory scheduling policies within the framework of an energy-aware DRAM subsystem.

This thesis builds upon this previous work by developing a real, high-performance DRAM subsystem which implements these policies. In addition, the thesis evaluates these policies within the framework of this design. Finally, the thesis discusses additional methods for improving the energy-efficiency of the DRAM subsystem.

Chapter 2

Memory System Overview

The high-performance, fully pipelined SCALE DRAM subsystem system implements a number of performance-enhancement and power reduction techniques. These techniques are implemented by a series of modules which determine how memory accesses are mapped to hardware, in what order they are issued, and when chips power down.

This chapter presents an overview of the SCALE DRAM system. First, it presents an overview of how DDR-II SDRAM chips operate. It then discusses the architecture of the SCALE DRAM subsystem. Finally, it discusses the policies that may be implemented by the system.

2.1 DDR-II SDRAM Overview

The SCALE DRAM subsystem is designed to interface with DDR-II (Dual Data Rate, second generation) SDRAM (Synchronous Dynamic RAM) memory chips. DRAM (Dynamic Random Access Memory) memories store data by storing a charge on internal capacitors. Synchronous DRAM (SDRAM) chips are simply clocked DRAM chips. This allows the chips to be pipelined, greatly improving chip bandwidth.

DDR-II SDRAM chips are second-generation dual-data-rate SDRAM chips. Dual-data-rate indicates that the chip's data bus operates at twice the frequency of the command bus. Data is therefore available on both the rising and falling edge of the clock. This doubling of the data frequency effectively doubles the width of the data

bus; an 8-bit DDR chip therefore has the same bandwidth as a 16-bit single-data-rate SDRAM chip. DDR-II chips differ from DDR chips primarily in terms of power consumption. DDR-II chips operate at a lower supply voltage in addition to providing a number of low-power operating modes.

2.1.1 SDRAM Structure

The memory locations of an SDRAM chip are not simply arranged as a linear array, but are arranged in a hierarchical structure. Each SDRAM chip is organized into banks, rows, and columns, as illustrated in Figure 2-1. At the highest level, each chip contains a number of independent banks, each of which contains an independent DRAM memory array. These arrays are divided into rows and columns. Each row contains some number of columns which contain bits of storage.

Associated with each of these arrays is an independent set of sense amplifiers, equal in size to the total number of memory locations in a single row of the array. These sense amplifiers detect the small changes in voltage on the array's bitlines and generate a strong logical output for each. As this detection can be time-consuming, the sense amplifiers also serve as a row cache. The row need only to be read once; subsequent accesses to the same row simply require a multiplexor which selects the appropriate column to be written to or read from the data pins. Accessing different rows, however, requires precharging the bitlines (Precharge) followed by loading the new row onto the sense amplifiers (Activation). Each bank may have one active row.

When an SDRAM chip is written to, the data is driven into the appropriate cells. This requires an amount of time known as the write-recovery time. The chip must therefore wait for this write-recovery time to pass after data has been written before the bank can be precharged or activated.

Due to this complex structure of SDRAM memories, varying memory access patterns of the same length may require dramatically different lengths of time to perform. Access patterns which exhibit a high degree of spatial locality will be much faster as row misses, which require precharge and row activation, will be uncommon.

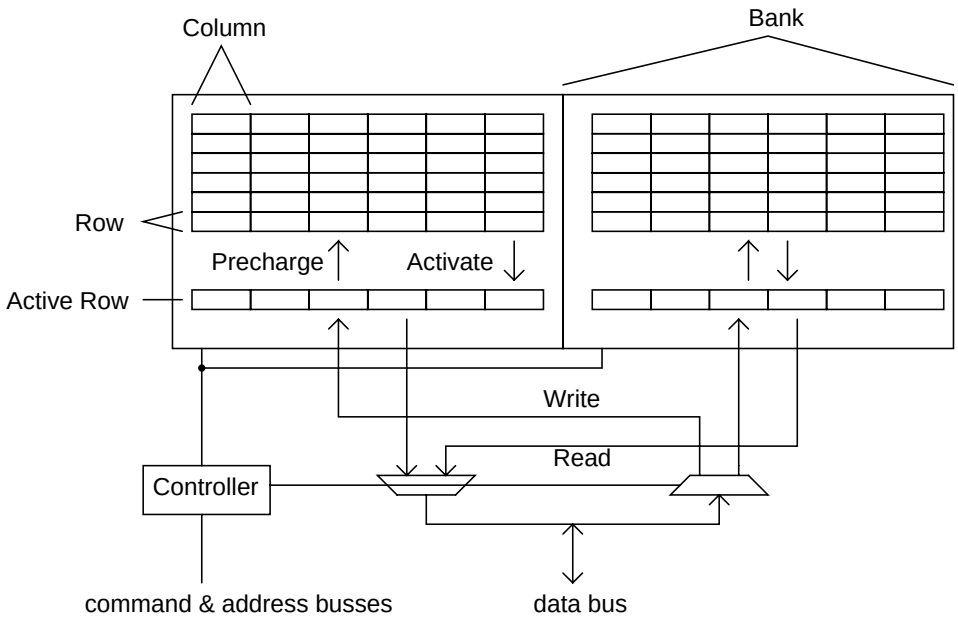

Figure 2-1: SDRAM Chip Architecture

2.1.2 DDR-II SDRAM Memory Accesses

As SDRAM memories are not simply linear arrays, memory accesses must occur in several stages. First, the row to be accessed must be activated. This reads the contents of the row and caches them on the bank's sense amplifiers. Once the appropriate number of cycles has passed, the contents of the active row can be accessed. If a read command is issued, the command propagates through the pipeline and the data is available on the data bus some number of cycles later. The duration in cycles between the issue of the read command and the data's availability on the bus is known as CAS (Column Access) latency. SDRAM chips may also operate in burst mode, in which a single read or write command accesses a number of sequential columns. If a chip is configured for a burst length of 4, for example, a read from row 10, column 4 would return the contents of columns 4 through 7 in succession. DDR, or Dual-Data-Rate SDRAM is designed to take advantage of this burst operation for greater bandwidth. A DDR data bus changes on both the rising and falling edge of the clock. A burst of 4, therefore, can be completed in just two cycles (offset by the CAS latency). A single activation-read sequence, with burst length 4, is illustrated in figure 2-2.

Successive reads or writes to the active rows can be issued as quickly as the data

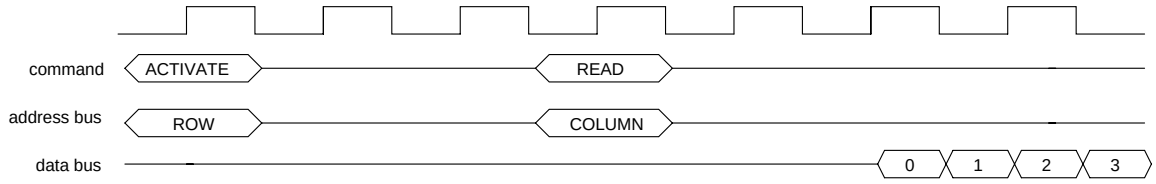

Figure 2-2: DDR SDRAM Read Operation

bus can support them. In the case of DDR SDRAM configured with a burst length of 4, for example, reads or writes to the active rows may be issued every other cycle as the data from each read requires 2 cycles on the data bus. The results of these pipelined reads will appear on the databus in a contiguous stream, offset from the commands by the CAS latency, as illustrated in Figure 2-3. The pipeline of SDRAM chips is such that reads and writes can not be issued in this contiguous fashion. Performing a read after a write, or vice versa, requires a certain number of idle cycles between the operations as illustrated in Figure 2-4. Interleaving read and write operations therefore can introduce significant overhead.

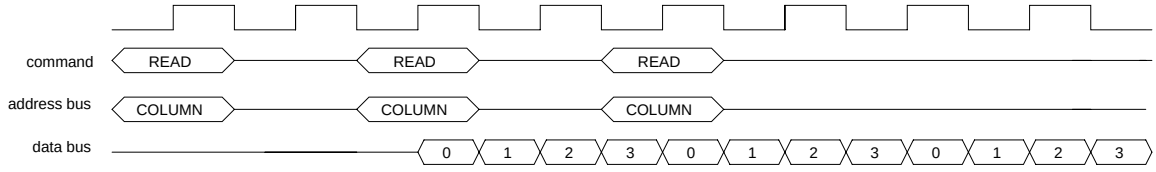

Figure 2-3: Pipelined DDR SDRAM Reads to Active Rows

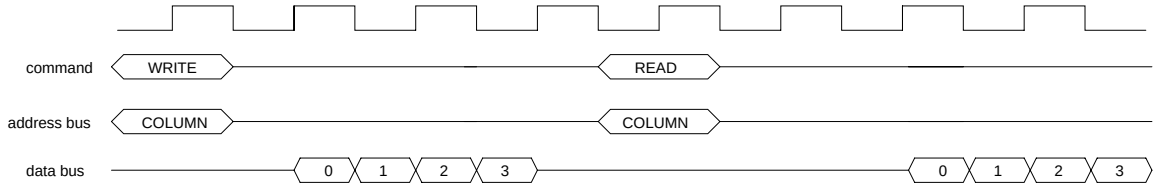

Figure 2-4: Write Followed by Read to Active Row

Accessing different rows in the SDRAM bank can also lead to significant overhead. As demonstrated in Figure 2-5, a read to a closed row requires precharge of the bank and activation of the new row before the read can be issued. The precharge and activation overhead in this case is more than 75% of the chip's bandwidth. This overhead can be reduced by issuing commands to different banks during the dead

cycles between the commands issued in 2-5.

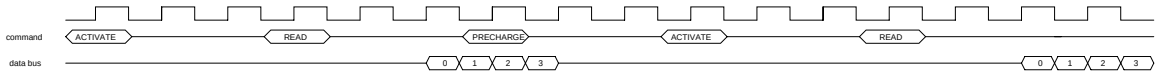

Figure 2-5: DDR SDRAM Reads to Different Rows in Bank

As the banks operate independently, a command can be sent to a particular bank completely independently of what another bank is doing, so long as the command will not lead to data bus contention between the banks. The dead cycles between activation and read of one bank, for example, could be used to activate a different bank. Although this does allow some improvement in bandwidth, the large latency penalty incurred when a request accesses a different row than the last request to that bank is unavoidable.

In order to free the control bus from unnecessary commands, read and write operations may also include an auto-precharge flag. When this flag is set, the bank will automatically precharge the bank after the read or write is performed. This is desirable when it is assumed that the next access to a bank will be to a different row than the current access.

As the SDRAM memory is dynamic, charge leakage may render the contents of the memory invalid. In order to prevent this invalidation of data, the SDRAM contents must be periodically refreshed. Each refresh command issued to an SDRAM chip refreshes a single row in the memory array. An internal counter determines which row will be refreshed upon each refresh command. Refresh commands must be issued in such a way that the average interval between refresh commands is (1 / number of rows) times the maximum interval over which data is guaranteed to be held valid. For this refresh to occur, the chip can have no active rows. All banks must therefore be precharged before the refresh command can be issued. This can lead to significant overhead when a refresh interrupts a stream of memory accesses.

As activation of a row will refresh the contents of a row, issuing refresh commands is unnecessary if all rows currently storing data are activated within the refresh interval. If the memory controller can guarantee that the rows will be accessed with appropriate frequency, the refresh performance penalty discussed above can be

avoided.

2.1.3 DDR-II Power Modes

In order to conserve power when idle, DDR-II chips offer a number of power modes. Each of these modes offers significant savings in power while degrading performance. Active powerdown mode, in which rows may remain activated, offers significant power savings with minimal performance degradation. When the powerdown mode is exited, the command bus must remain idle for a small number of cycles for resynchronization. Active powerdown mode is further divided into Fast-Exit and Slow-Exit mode. Exiting Slow-Exit Active Powerdown requires a greater resynchronization time, but the power consumption while in the Power-down mode is greatly reduced.

Precharge Powerdown is similar to Active Powerdown, but requires that all banks be precharged before entering the mode. This mode consumes less power than either Active Powerdown mode, and requires a similar length of time as Slow-Exit Active Powerdown to return to activity. The possible transitions between all DDR-II power modes, with associated resynchronization times, are illustrated in Figure 2-6.

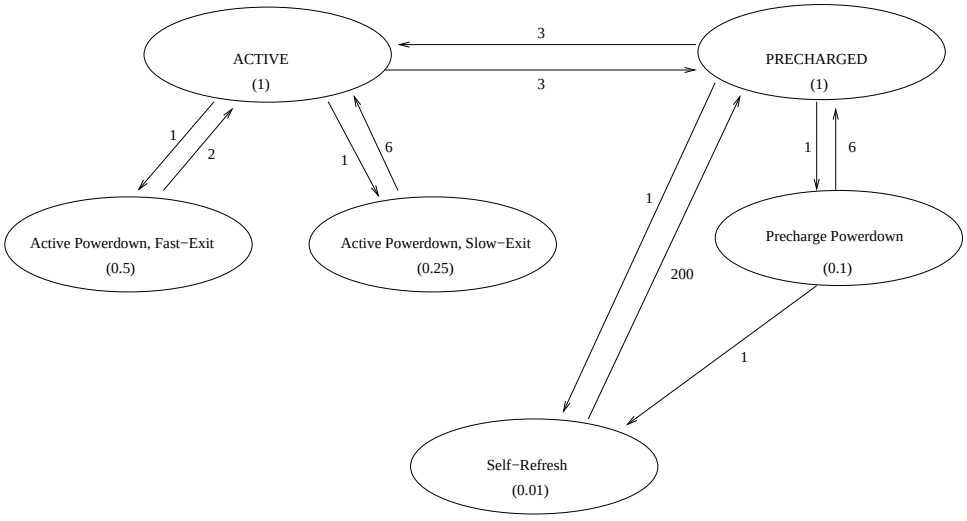

Figure 2-6: DDR-II SDRAM Power State Transitions (Relative Energy in Parentheses, Arcs labelled with transition time in cycles)

Both Active Powerdown and Precharge Powerdown require that the chip be periodically activated, refreshed, and then powered back down. Since all banks must be

precharged in order for the chip to be refreshed, this will cause a transition to the Precharge Powerdown state even if the chip was initially in the Active Powerdown State. Awakening the chip, refreshing, and then powering back down can consume a considerable amount of power. DDR-II chips can enter the Self-Refresh mode in which the chip refreshes itself rather than being instructed to do so by the controller. The chip consumes less power in Self-Refresh mode than in any other mode. Activating the chip, however, introduces a tremendous performance penalty, as activating a chip that has been in Self-Refresh mode requires several hundred cycles.

2.2 Memory System Design

The SCALE DRAM subsystem, illustrated in Figure 2-7, is a high-performance, energy-aware DRAM subsystem designed to allow rapid implementation of a number of system policies. The system consists of a SIP (System Interface Port) Interface, a Request Dispatcher, a Master Request Buffer, a Completed Request Manager, and a number of memory channels. Each of these memory channels interfaces with one or more SDRAM chips. These channels operate independently, although a single request may map to multiple channels. This provides benefits to both performance, as requests can be serviced in parallel, and energy, as idle channels may power down their SDRAM chips.

2.2.1 SIP Interface

The SCALE memory system interfaces with the SCALE chip via a dedicated SIP (Serial Interface Protocol) connection, running at 200 MHz. Each SIP channel consists of a tag, an operation to perform, and a multiplexed 32-bit address/data bus. The SIP protocol provides for load and store requests for bytes, half words, 4-byte words, and 8-word cache lines. All load requests require a single cycle to be transmitted over the SIP channel. Byte, Half-Word, and Word stores all require 2 cycles (1 cycle for the address, 1 for data), while Line stores require 9 cycles.

The SIP data is queued by an asynchronous FIFO that allows the DRAM sub-

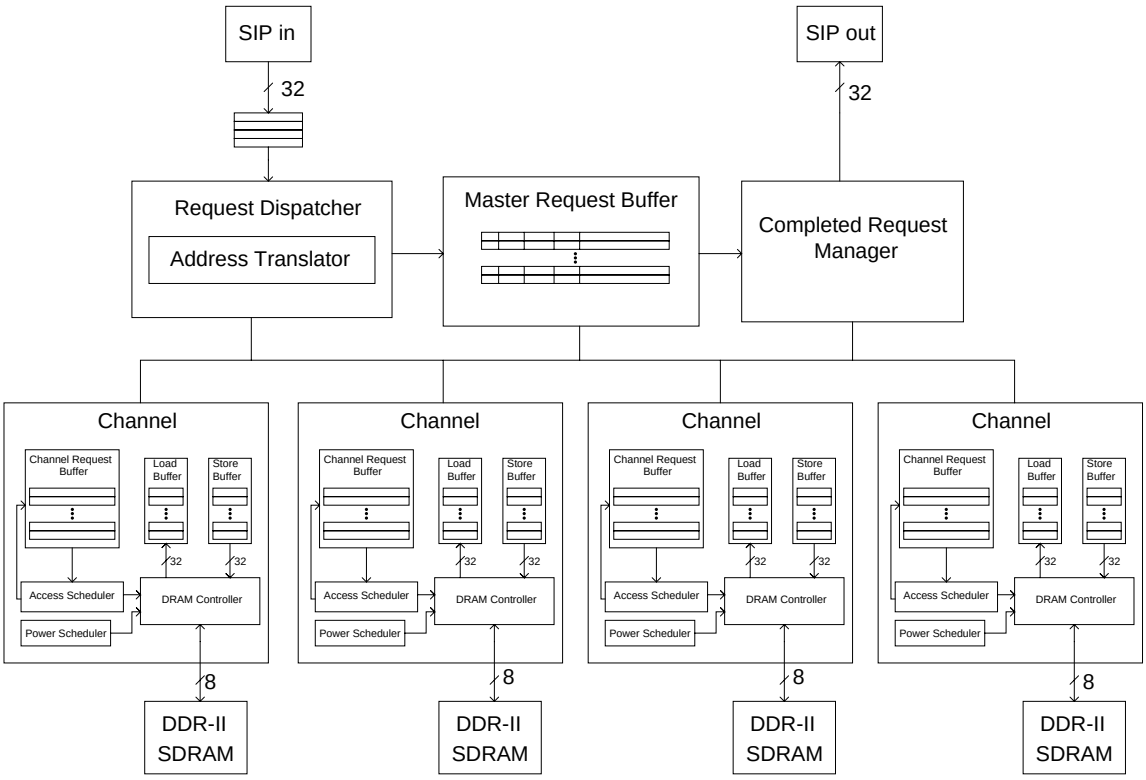

Figure 2-7: SCALE Memory System

system to operate at a different clock frequency than the clock provided by the SIP channel. If the DRAM subsystem is clocked by the SIP clock, this queue is unnecessary.

2.2.2 Request Dispatcher

The Request Dispatcher is responsible for determining which hardware resources will be used to service an incoming request, and for dispatching these requests to the appropriate modules.

When the Request Dispatcher receives a request from the SIP interface, it passes the request's address through the Address Translator. The Address Translator indicates which channel or channels should service the request. The Address Translator also indicates how the request should map into the SDRAM chips of the channel(s) by indicating to which SDRAM chip, bank, row, and column the request should map.

The Request Dispatcher must dispatch the request to the appropriate channels,

as indicated by the Address Translator, as well as the Master Request Buffer (MRB). In the case of a load request, this dispatch only requires one cycle: the hardware mapping information, as determined by the Address Translator, is broadcast across the channel request bus and the Request Dispatcher asserts the channel buffer write enable signals such that the appropriate channels receive the request. The Request Dispatcher simultaneously writes a new entry to the MRB, containing information about the request as well as pointers to the entries written in the Channel Request Buffers of the affected channels.

In the case of a store, the Request Dispatcher must write the incoming store data to the appropriate Channel Store Buffers (CSBs). Only when a channel has received all the data for which it is responsible may the request be written to the Channel Request Buffer (CRB), as by writing to the CRB the Request Dispatcher is guaranteeing that the data is ready to be written to DRAM. This requirement is due to flow-control constraints in the Request Dispatcher. The Request Dispatcher may be interrupted by flow-control signals as it is reading store data from the SIP FIFO. If the Request Dispatcher does not wait until it has read the last word from the FIFO that is required by a certain channel before indicating that the channel may service the request, the channel may require this data before it becomes available. Due to this constraint, it is preferable to send adjacent words to the same channel, rather than interleaving the words across channels. This allows the earlier channels to begin to service the request even if there is an interruption in the SIP data stream. In the case where a store is interleaved across multiple channels, the channels receive the request in a staggered fashion, as illustrated in Figure 2-8.

The Request Dispatcher must write the request to the MRB on the cycle that it begins dispatching a request, ensuring that the appropriate MRB entries have been created before any of the channels service the request.

2.2.3 Memory Channel

The memory channels, illustrated in figure 2-9, are responsible for issuing requests to the DRAM controller and returning the results to the MRB and Completed Request

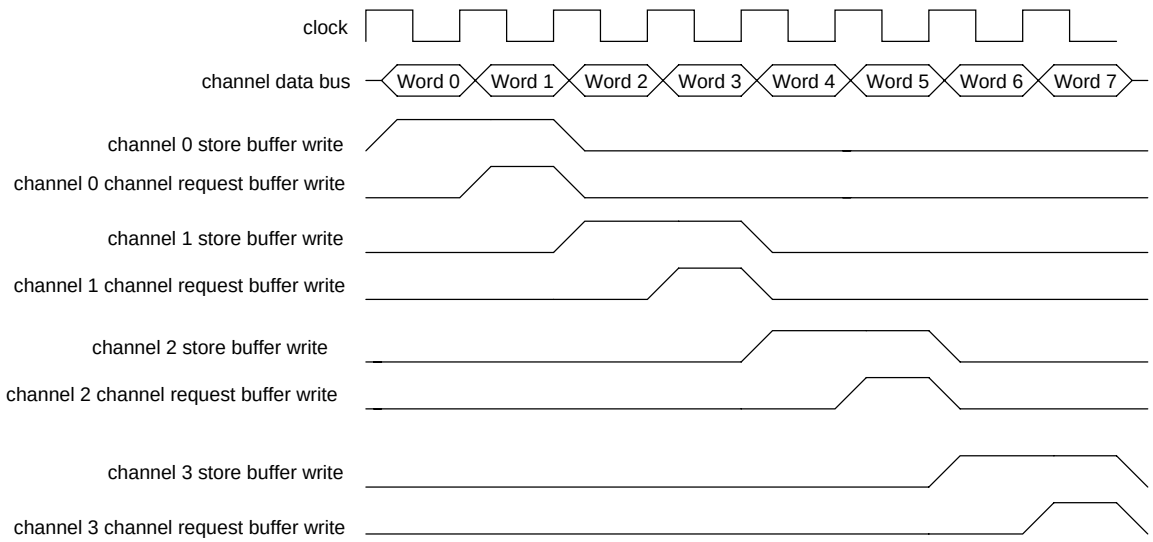

Figure 2-8: Timing: 8-word Store Interleaved Across 4 Channels

Manager. When the request dispatcher issues a request to a channel, the request is first written to the Channel Request Buffer (CRB). The CRB stores all information needed by the DRAM controller as well as a pointer to the MRB entry to which this request refers. In the case of a store, the data to be written to memory is also written to the Channel Store Buffer (CSB).

The channel issues requests to the DRAM controller as they are produced by the CRB. The Access Scheduler controls which request is issued by generating the appropriate indices to the CRB. As the Access Scheduler may indicate that requests should be issued out-of-order due to certain characteristics of the request, the Access Scheduler must monitor the data written to the CRB so that it is aware of which request should be selected next. The Access Scheduler is also responsible for indicating to the Channel Controller whether the current output of the CRB is a valid, unserviced request.

The Channel Controller serves as an interface between the CRB and the DRAM controller. When the Access Scheduler informs the Channel Controller that the current request as output by the CRB is valid, the Channel Controller issues this request to the DRAM controller and informs the Access Scheduler that the request has been serviced. As the DRAM controller only accepts 32-bit word requests, the Channel

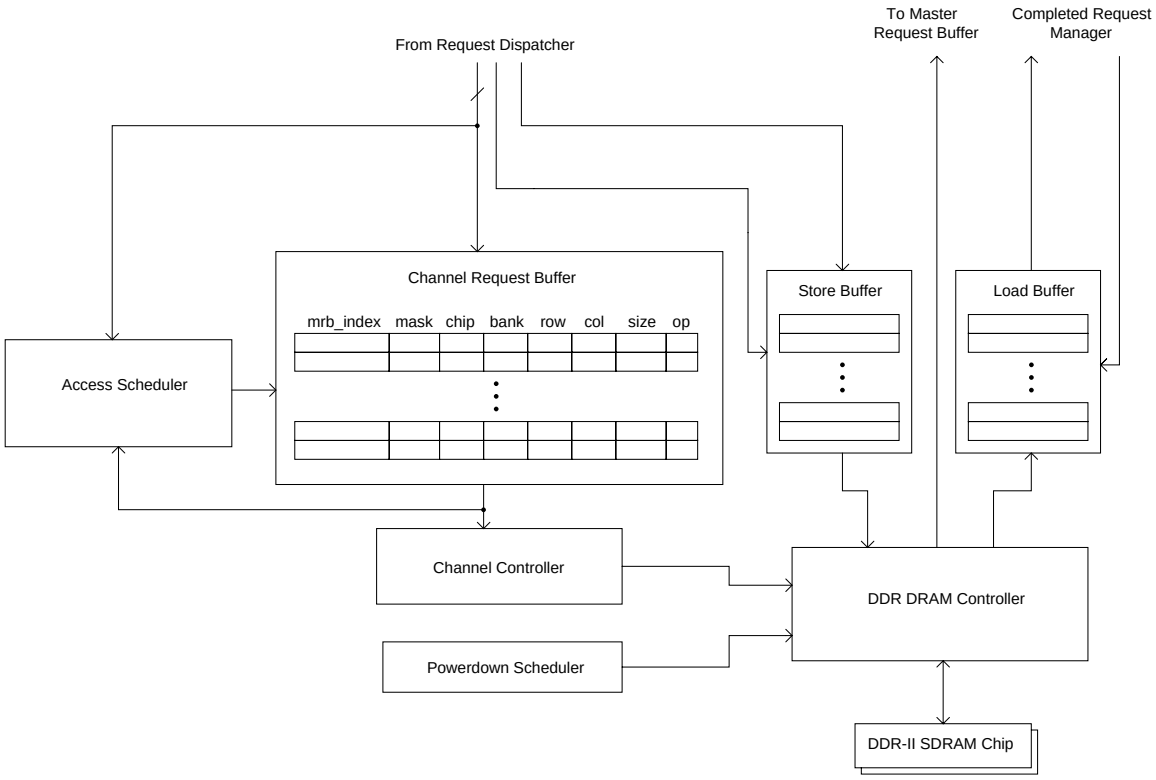

Figure 2-9: Memory Channel

Controller must break larger requests into 32-bit sub-requests for issue to the DRAM Controller. In the case of a store request, the Channel Controller must also generate the appropriate indices to the CSB so that the DRAM controller writes the appropriate word. Finally, the Channel Controller must construct a tag which is associated with the request. This tag includes a pointer to the request in the CRB so that it may be properly invalidated by the Completed Request Manager, a pointer to the request's entry in the Master Request Buffer (MRB) so that the DRAM controller may inform the MRB that the request has completed, and in the case of a Load request, a pointer to the target location in the Channel Load Buffer (CLB). Additionally, the tag contains a flag to indicate whether the request is the final word in a request that is larger than one word.

The Channel Load Buffer and Channel Store Buffer are 32-bit register files, with a number of entries equal to the number of entries in the CRB multiplied by the maximum request size, in words. The high-order bits of a word's index match the

corresponding request's index in the CRB, while the low-order bits contain the word's offset in the request. Having separate buffers for Loads and Stores is redundant, as no request will need space in both the CLB and the CSB. If a register file with 2 write ports and 2 read ports is available, these buffers can be combined into a single Channel Data Buffer. The buffers are separate in this design so that standard 2-port register files may be used.

2.2.4 DDR-II SDRAM Controller

The DDR-II SDRAM Controller, illustrated in Figure 2-10, consists of a number of bank controllers (one for each internal DRAM bank) and a chip controller. When a request is written to the DDR Controller, it is written to an asynchronous FIFO in the appropriate bank controller. This asynchronous FIFO allows the overall system to operate at a different clock frequency than the DDR Controller itself, which must operate at the frequency of the SDRAM chips. The chip controller is responsible for chip initialization, refresh, and power mode transitions. As all chips in a channel share control and data busses, the controller can be used for multiple chips by treating the chips as a single chip with N*M banks, where N is the number of banks per chip, and M is the number of chips.

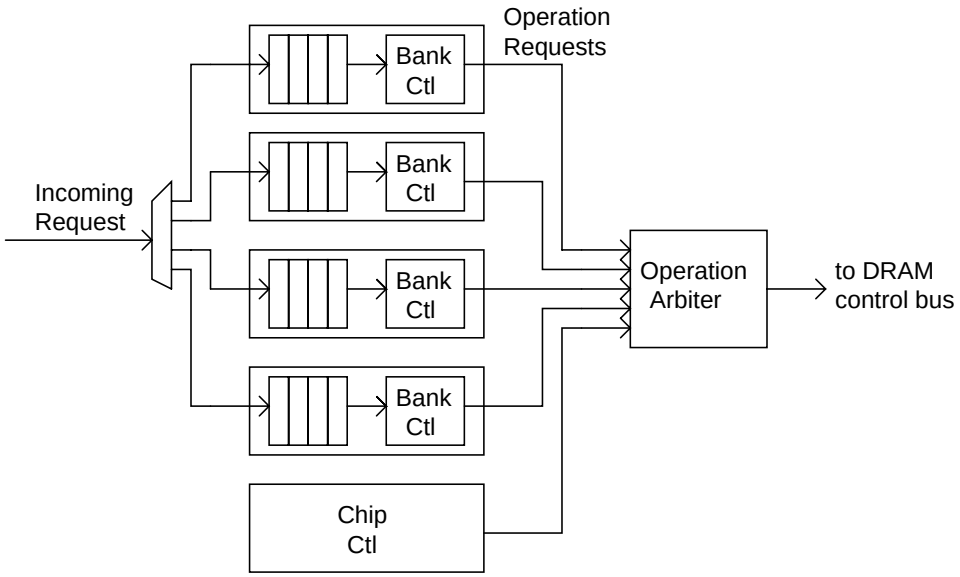

Figure 2-10: DDR Controller Dataflow

Each bank controller tracks the current state of the bank for which it is responsible. It first reads a request from the FIFO and determines which operations to perform. If the request is to the active row, the bank controller may simply issue a READ or WRITE operation. Otherwise, if another row is active, it must issue a PRECHARGE command. If the bank has been precharged, the bank controller may issue an ACTIVATE command.

As each bank may have a different outstanding request, an Operation Arbiter must select which of the requested operations to issue over the SDRAM chip's control bus. This arbiter issues bank requests in a round-robin fashion, and issues refresh and power mode transition operations as they are requested by the Chip Controller.

Hot-Row Prediction

Bank READ and WRITE requests may also request that the bank automatically precharge the active row once the operation has completed. This has the advantage of freeing the control bus for the cycle that a PRECHARGE command would be issued if the following request was to a different row. This is a disadvantage, however, in the case of subsequent requests to the same row; the bank is unneccessarily precharged and the row reactivated for each request, introducing significant delay and energy overhead. In order to profit from Auto-Precharge when appropriate, but to avoid the overhead when requests access the same row, the bank controller includes a Hot-Row Predictor. This predictor determines whether to issue an Auto-Precharge request based on past request history.

The Hot-Row Predictor in implementation is similar to a simple microprocessor branch predictor. Every time the SDRAM chip is accessed, the hot-row predictor determines whether the access is a hit, an access to the active row, or a miss, an access to a different row. The predictor uses this information to determine whether to issue an auto-precharge command with the access, thus speeding up a following request to a different row. The state diagram of Figure 2-11 illustrates the 2-bit prediction scheme used by the hot-row predictor.

The SCALE DRAM subsystem uses a single 2-bit predictor for each DRAM bank.

This predictor effectively takes advantage of the row locality in the current access stream while introducing minimal complexity to the design. More advanced designs, in which more predictors indexed by the request address are used, are beyond the scope of this document.

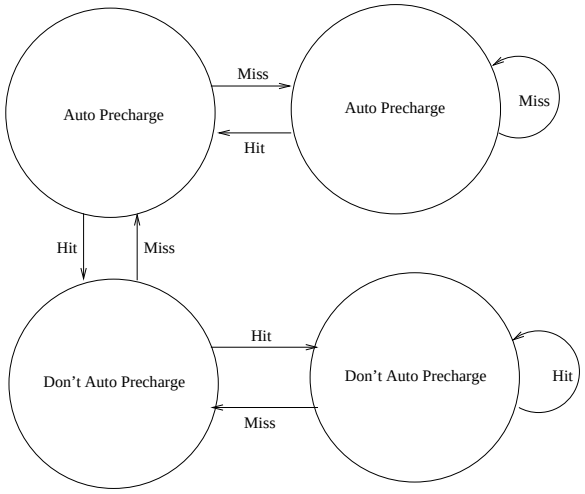

Figure 2-11: Hot-Row Predictor Policy

Cut-Through

As illustrated in the pipeline diagram of Figure 2-12, it takes a number of cycles for a request acknowledgement to propagate through the return pipeline. The DRAM subsystem may therefore overlap this acknowledgement latency with the SDRAM access. The controller need simply guarantee that the request will have completed before the Copmleted Request Manager will need to read the results from the Channel Load Buffer.

The controller does not know when a given request will complete until the bank arbiter issues the read or write command. At this point, the controller can determine how early it can signal that the request has been completed.

If the number of bytes that the channel can read or write per cycle (data bus width * 2, for DDR) is greater than the width of the SIP interface, the controller can indicate that the request has been completed after a number of cycles equal to

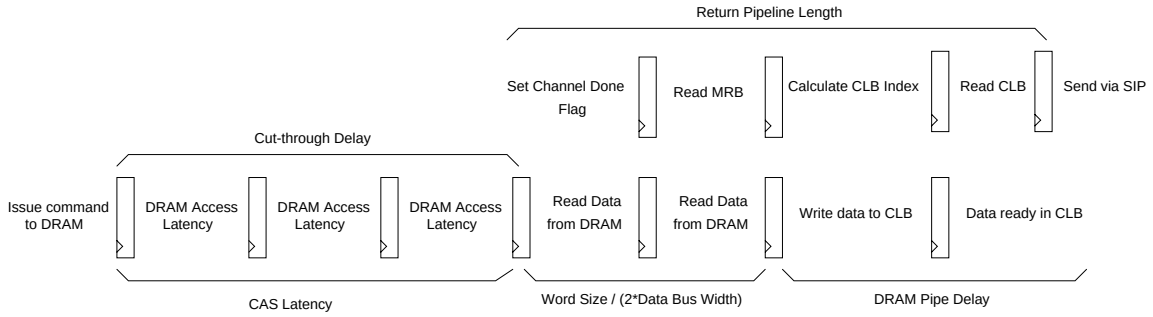

Figure 2-12: Cut-Through Pipeline Illustration

$$(CAS\ Latency) + \frac{2 * (Data\ Bus\ Width)}{SIP\ Width}$$

$$+(DRAM\ Pipe\ Delay) - (Return\ Pipeline\ Length)$$

The DRAM Pipe Delay represents the number of cycles required for the data from the SDRAM to become available at the output of the Channel Load Buffer The Return Pipeline Length is the number of pipeline stages between the SDRAM controller and the Completed Request Manager. If the calculated cut-through delay is less than 0, the controller can issue an acknowledgement immediately.

If the number of bytes that the channel can read or write per cycle is less than the width of the SIP interface, the controller must wait a number of cycles equal to

$$(CAS\ Latency) + \frac{2 * (Data\ Bus\ Width)}{SIP\ Width} - \frac{Request\ Size}{SIP\ Width}$$

$$+(DRAM\ Pipe\ Delay) + 1 - (Return\ Pipeline\ Length)$$

Again, if this value is less than 0, the controller can issue an acknowledgement immediately.

2.2.5 Master Request Buffer

The Master Request Buffer (MRB) is responsible for determining when all channels involved in a particular request have completed their portion of the request. In implementation, the MRB is similar to a superscalar microprocessor's reorder buffer.

When the Request Dispatcher issues a request, it writes a new entry in the MRB. For each channel which is involved, the Request Dispatcher also writes a pointer to the appropriate Channel Request Buffer (CRB) to the MRB.

In addition to the buffer containing this request information, the MRB module contains a set of channel done flags for each channel, equal in number to the number of entries in the MRB. These flags track whether the channel has completed the entry referred to by the corresponding entry in the MRB. When the Request Dispatcher writes a new request to the MRB, it also sets the appropriate flags for each channel. If a channel is not involved in a given request, that channel's flag corresponding to the request's location is set to 1. If the channel is involved, the appropriate flag is set to 0.

As channels complete their portions of a request, they indicate to the MRB that they have done so. Included with this completion signal is a pointer to the request's entry in the MRB. The MRB uses this information to set the appropriate channel done flag to 1. Once all channel done flags for the oldest pending request have been set to 1, the MRB issues the request to the Completed Request Manager and moves on to the next queued request.

2.2.6 Completed Request Manager

The Completed Request Manager (CRM) is responsible for gathering the appropriate data to return over the SIP channel once it receives a completed request from the Master Request Buffer (MRB). In the case of a store request, the Completed Request Manager simply sends a store acknowledgement with the appropriate tag over the SIP channel. In the case of a load request, it must first generate the appropriate indices to the Channel Load Buffers so that the appropriate data will be sent over the SIP channel in the right sequence. As this data arrives from the Channel Load Buffers, the CRM must sent the data over the SIP channel.

As the CRM sends a request over the SIP channel, it invalidates that request's entries in the MRB and in each of the Channel Request Buffer (CRB). This invalidation is pipelined such that it may occur concurrently with the CRB's servicing of

the next completed request.

2.3 Policy Modules

The memory system's power management and performance enhancement policies are implemented by three policy modules: the Address Translator module, the Access Scheduler module, and the Power Scheduler module. In this thesis, hot-row prediction policies are considered a fixed component of the DDR controller, and are therefore not treated as a separate policy module. Each of these three modules implements a fixed interface, and can therefore be treated by the rest of the memory system as a black box. Changing a controller policy simply requires swapping in the appropriate policy module; the rest of the design remains unchanged. These policies are evaluated in Chapter 4.

2.3.1 Address Translator

The Address Translator module determines how a physical address maps into hardware. Given a physical address, the Address Translator determines what channel(s) should handle the request, how many words should be sent to each of these channels, and which DRAM chip, bank, row, and column the request should map to. For example, if full cache line interleaving is implemented, the Address Translator indicates that the request should be sent to all channels, and each of the 4 channels should handle 2 words of the 8-word request. Alternatively, cache lines could map to only 2 channels. This would lead to reduced performance, but potentially less energy, as the unused channels could be powered down. The address translator module would simply need to be modified to indicate that only 2 channels should be used, and 4 words should be handled by each of them.

2.3.2 Access Scheduler

The Access Scheduler module determines which of a channel's queued requests should be issued to the DRAM controller. The module determines which request is issued by monitoring the Channel Request Buffer's control signals and data busses and then generating appropriate read and write indices for the Channel Request Buffer. For example, an open row policy[8] may be implemented in which all queued accesses to an SDRAM row are to be issued before any other access. The Access Scheduler module would track which row is accessed by each request, and if another request accesses the same row, the Access Scheduler will provide the index for that request's buffer entry to the channel. This may lead to redundancy, as the row information for each request would be stored both by the Channel Request Buffer and the Access Scheduler, but it would retain the modularity of the design. The Access Scheduler is also responsible for tracking whether a given Channel Request Buffer entry is valid as well as whether it has been dispatched to the SDRAM controller.

2.3.3 Power Scheduler

The Power Scheduler module determines when an SDRAM chip should be powered down as well as the low-power state into which it should transition. The module listens to the SDRAM controller's status signals to determine how long the chip has been idle and signals to the controller when and to which mode it should power the chip(s) down. For example, if a channel has been idle for 10 cycles, the Power Scheduler module could tell the SDRAM controller to power down all the channel's SDRAM chips.

Chapter 3

Methodology

The goal of the SCALE memory system is to perform well in typical applications of the SCALE processor while using as little energy as possible. Analyzing the performance and energy characteristics of the system is quite complex due to the interplay of system policies and the non-linear nature of the DRAM chips themselves.

This chapter presents the methodology used in this thesis to accurately analyze the performance and energy consumption of the SCALE DRAM subsystem. Section 3.1 discusses the SCALE DRAM simulator, a cycle-accurate model of the SCALE DRAM subsystem. The section also discusses the energy model used in this simulator. Section 3.2 briefly discusses the physical implementation of the SCALE DRAM subsystem which will provide for more accurate power measurements upon its completion. Finally, Section 3.3 discusses the characterization and selection of the benchmarks to be used for the policy characterization of Chapter 4.

3.1 The SCALE DRAM Simulator

In order to characterize the system and its various policies, the system policies must be rapidly prototyped and evaluated for a number of benchmarks. This prototyping and evaluation is made possible by the SCALE DRAM simulator, which models the SCALE DRAM subsystem discussed in Chapter 2. This cycle-accurate, execution-driven simulator interfaces with the SCALE microarchitectural simulator as illus-

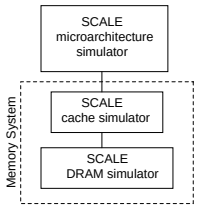

Figure 3-1: The SCALE Simulator

trated in Figure 3-1. The simulator allows precise characterization of the system and rapid evaluation of various system policies.

The SCALE DRAM Simulator is a C++ model of the entire SCALE DRAM system, from the processor interface to the SDRAM chips. It implements the SplitPhaseMemIF memory interface, thus allowing it to be integrated with the SCALE Simulator or any other simulator using the same interface. The SplitPhaseMemIF is a split-phase memory interface consisting of request and response methods. When modelling a synchronous system, these request and response calls are called once per cycle; the request method introduces a new memory request to the system, and the response method returns any completed memory requests.

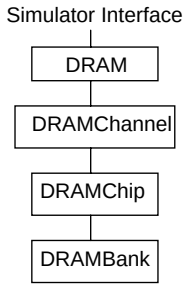

Figure 3-2: SCALE DRAM Simulator Object Hierarchy

The SCALE DRAM Simulator consists of a number of C++ objects arranged in a hierarchy which mirrors that of the physical system. The hierarchy is illustrated in Figure 3-2. The DRAM object, which subclasses SplitPhaseMemIF, is responsible for the tasks of the SIP interface, Request Dispatcher, Master Request Buffer, and Completed Request Manager. The DRAM object contains references to an array of DramChannel objects, which model the independent DRAM channels, with their respective channel request buffers. The DRAM controller and SDRAM chips are

modeled by the DramChip and DramBank objects. The timing of the SDRAM model is cycle-accurate, including effects due to row activation and precharge, refresh, power mode switching, and bus contention.

Each object in the simulator hierarchy contains a Cycle method. Clocking of the simulator is performed by calling the DRAM object's Cycle method. All objects cycle the objects below them in the hierarchy. This occurs before the object does any processing, thus ensuring that the entire system has cycled before the state is modified.

Clocking of the system is implicit in calling the response method. As this method should only be called once per cycle, the DRAM object clocks itself each time response is called. The simulator interface should therefore never directly call the Cycle method.

All timing and energy calculations refer to a cycle counter internal to the DRAM object. It is therefore essential that the DRAM system be clocked even when there are no outstanding requests. The SCALE microarchitectural simulator command-line parameter -msys-dram-always-cycle ensures that this is the case.

3.1.1 Policy Modules

In order to support rapid development and evaluation of various system policies, the system policies are implemented by a series of classes. With the exception of address translation, implementing a new policy simply requires subclassing the appropriate virtual class and implementing this class' virtual functions.

Class AddressTranslator

The translation of a physical address to hardware, including cache line interleaving and channel assignments, is performed by the AddressTranslator class. The address translation policy is determined by setting the simulator's granularity and ibank_mapping parameters. Address mapping is performed by class methods which, given an address, determine the number of channels that a request maps to, the num-

ber of bytes sent to each channel, the first channel to which the request maps, and the chip, bank, row and column to which it maps within each channel.

Class DramScheduler

The DramScheduler virtual class controls the order in which requests are issued from the channel request queue to the appropriate chips. A scheduler policy module must sub-class DramScheduler and implement the nextRequest virtual method, which is responsible for removing the appropriate request from the channel request queue and returning it to the calling function.

Class DramPowerScheduler

The DramPowerScheduler virtual class controls when the power mode of a DRAM chip is to be changed. A power policy is implemented by subclassing the DramPowerScheduler class and implementing a number of virtual functions. The nextMode function determines which mode is next in the policy's power state sequence. The Cycle function is called by the channel on every cycle, and is responsible for calling the channel's Powerdown method when appropriate. Although a chip will automatically wake up when it receives a request, the power scheduler may indicate that the chip should wake in anticipation of a future request by returning true from its requestWake function. Finally, the notify function is called by the chip whenever it changes states. The power scheduler updates its internal state within this function.

3.1.2 System Parameters

The SCALE DRAM Simulator is highly parametrizable. The most important parameters used for the studies in this document are summarized in Table 3.1. These parameters may written in an ASCII data file, the name of which must be passed to the simulator with the -msys-dram-params argument.

Parameter	Description	Acceptable Values
return_ordering	system acknowledges requests in-order	0 or 1 (bool)
clock_divider	system clock = ddr clock / clock_divider	integer
dram_stats	instructs simulator to track statistics	0 or 1 (bool)
scheduler	scheduling algorithm	FIFO, OPEN_ROW, ALT_BANK, RD_BF_WR
scheduler_throttle	scheduler throttle, in cycles	integer
hot_row_policy	hot row prediction policy	CLOSE, OPEN, PREDICTOR
hot_row_predictor_bits	hot row predictor size	integer
powerdown_policy	system powerdown policy	ALWAYS_AWAKE, CTP, ATP
powerdown_wait	idle cycles before entering shallow powerdown state	integer
deep_powerdown_wait	idle cycles before entering deep powerdown state	integer
power_sequence	powerdown sequence	AAPDF, AAPDS, AAPDFSR AAPDSSR, APPD, APPDSR
granularity	address mapping granularity	power of 2, greater than max request size divided by number of channels
ibank_mapping	address translator bank mapping	0 or 1
max_requests	most concurrent requests system can handle	integer
reqqueue_size	Channel Request Queue size	integer
bankqueue_size	size of asynchronous FIFOs in DDR Bank controllers	integer
num_channels	Number of memory channels	power of 2
chips_per_channel	SDRAM chips per channel	integer
interface_width	SIP interface width (bytes)	power of 2
banks_per_chip	Number of banks in SDRAM chips	power of 2
burst_length	SDRAM burst length	power of 2
dbus_width	SDRAM data bus width	power of 2
data_rate	SDRAM Data Rate	2 for DDR
num_rows	SDRAM Rows per Bank	power of 2
row_size	SDRAM Columns per Row	power of 2

Table 3.1: SCALE DRAM Simulator Parameters

Operation	Energy Cost (nJ)
Idle Cycle	0.288
Active Powerdown Cycle - Fast Exit	0.135
Active Powerdown Cycle - Slow Exit	0.063
Precharge Powerdown Cycle	0.032
Self Refresh Cycle	0.027
Byte Written	0.585
Byte Read	0.495
Activation	2.052
Precharge	2.052
Refresh	3.762

Table 3.2: Energy Requirements for DDR-II SDRAM Operations

3.1.3 Statistics Gathering and Energy Calculation

If the dram-stats parameter has been set, the simulator will track a number of statistics as it operates. For each DramChip object, the simulator tracks the number of cycles spent in each power state, the number of writes as well as the number of bytes written, the number of reads and bytes read, and the number of precharges, activations and refreshes performed. Each of these operations has an associated energy cost that can be set as one of the simulation parameters. Once the simulation is completed, the simulator calculates the total energy consumed during the simulation and reports it along with these statistics.

Ultimately, the energy costs of various operations will be measured as discussed in Section 3.2. At the time of writing of this thesis, these measurements are unavailable. The results shown therefore use the energy costs listed in Table 3.2. These costs are derived from the data sheet for Micron MT47H32M8 DDR-II SDRAM chips [1].

3.2 Physical Implementation

The physical implementation of the SCALE memory system, illustrated in Figure 3-3 and photographed in Figure 3-4, consists of an FPGA and a number of DDR-II SDRAM chips. The Verilog model of the memory system is synthesized and programmed onto the FPGA. The FPGA and SDRAM chips have separate power sup-

plies, so the power consumed by the DRAM chips may be measured separately. When this board becomes available, the power drawn by the SDRAM chips will be measured and used to generate more accurate energy measurements for use in the simulator.

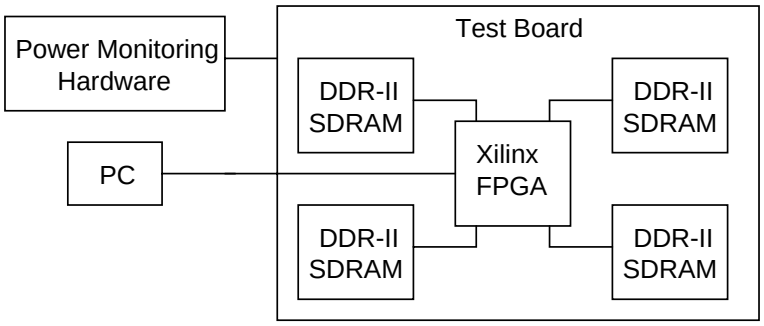

Figure 3-3: DRAM Energy Measurement Board

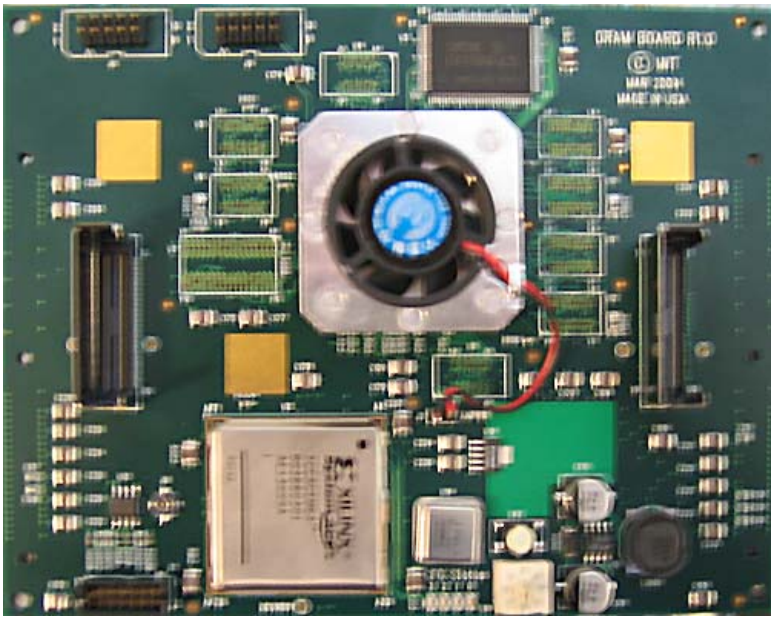

Figure 3-4: DRAM Energy Measurement Board (Photograph)

3.3 Benchmark Selection

In order to evaluate the performance of the SCALE DRAM system, it is necessary to select benchmarks that represent a wide range of typical applications. The benchmarks used in this thesis are selected from the EEMBC (Embedded Microproces-

sor Benchmark Consortium) benchmark suite. The EEMBC benchmark suite is an industry-standard means of evaluating embedded processors[4]. The assembly code for the benchmarks used in this thesis has been hand-optimized for optimal performance on the SCALE processor.

The DRAM system's performance is affected by a host of factors, but primary among them are the frequency with which the system is issued memory requests, the size of these requests, and the spatial locality of the requests. In order to aid in the selection of benchmarks for use in policy evaluation, the simulator calculates the relative memory request frequency, the average request size, and the average distance between successive requests.

The frequency with which the memory system receives requests when running a certain benchmark is calculated as

$$Request\ Frequency = \frac{Number\ of\ Requests}{Simulation\ Length\ (Cycles)}$$

As the current SCALE system only generates cache-line requests, all requests are of size 32 (the size of a SCALE cache line).

The memory locality is calculated as

$$Request\ Locality = Log2(\frac{2^{64}}{Average\ number\ of\ bytes\ between\ request\ addresses})$$

Each time the system receives a request, the address of the last request is subtracted from the address of the current request. The absolute value of this difference is averaged over the course of the simulation and inverted. As this value is inverted, the log of the value is taken to produce a linear Request Frequency metric.

The memory request frequency and locality for a number of benchmarks are illustrated in Figures 3-5 and 3-6, respectively.

In order to evaluate various system policies for the SCALE DRAM subsystem, it is important to use benchmarks which vary significantly in these characteristics, thus illustrating the effect of the policies for a wide range of applications. In addition, selection of benchmarks that vary significantly in one metric while remaining similar

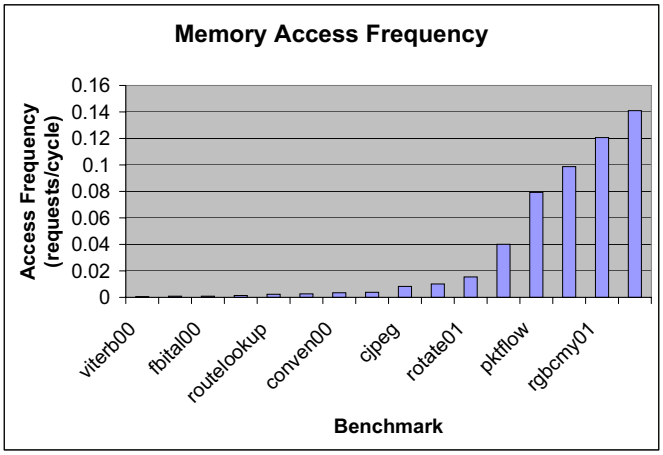

Figure 3-5: Memory Access Frequency for EEMBC Benchmarks

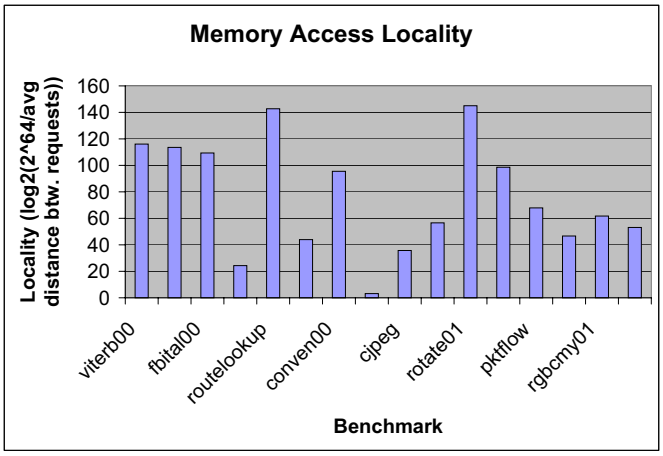

Figure 3-6: Memory Access Locality for EEMBC Benchmarks

in the others allows the effect of each metric to be isolated. In consideration of these factors, the benchmarks rgbyiq01, pktflow, dither01, rotate01, and routelookup are used to characterize policy performance in Chapter 4. rgbyiq01 and dither01 exhibit similar locality while differing significantly in access frequency. dither01 and rotate01 exhibit similar access frequency while varying greatly in access locality. rotate01 and routelookup exhibit similar locality while exhibiting different access frequency. As these benchmarks were selected because of their extreme locality or frequency, pktflow is also used as it exhbits medium behavior for both frequency and locality. These five benchmarks present a wide spread in these characteristics while allowing the contributions of individual characteristics to be analyzed.

The rgbyiq01 benchmark involves the conversion implemented in an NTSC video encoder wherein RGB inputs are converted to luminance and chrominance information. This operation involves a matrix multiply calculation per pixel. As the images involved are quite large, the operations do not cache well and the memory system is therefore hit quite hard.

The dither01 benchmark models printing applications in which a grayscale image is dithered according to the Floyd-Steinberg Error Diffusion dithering algorithm. The benchmark converts a 64K grayscale 8bpp image to an 8K binary image and maintains two error arrays. The benchmark was designed to test the system's ability to manage large data sets, but the benchmark's low access frequency as illustrated in Figure 3-5 suggests that the benchmark caches reasonably well.

The rotate01 benchmark involves the 90-degree rotation of an image. The benchmark was designed to test bit manipulation capability of the processor rather than to stress the memory system, but as the image file does not cache well, the DRAM system is still taxed. As the DRAM traffic consists primarily of cache overflows of relatively localized memory addresses (pixels in the image), the benchmark exhibits high locality as illustrated in Figure 3-6.

The routelookup benchmark models the receiving and forwarding of IP datagrams in a router. The benchmark implements IP lookup based on a Patricia Tree data structure. The benchmark repeatedly looks through this data structure over the

course of execution. As this structure caches well, the DRAM traffic is low, but cache misses are quite localized.

Finally, the pktflow benchmark performs IP network layer forwarding functionality. The becnhmark simulates a router with four network interfaces, and works with a large (512KB-2MB) datagram buffer stored in memory. The use of this large buffer leads to frequent cache misses and limited access locality.

All memory requests seen by the SCALE DRAM system as these benchmarks are executed are filtered by the SCALE cache. The SCALE cache is a unified 32-way set-associated cache, divided into four banks. Further discussion of the SCALE prototype processor, the SCALE cache configuration, and the operation of these benchmarks in this system can be found in Krashinsky[6].

Chapter 4

System Policy Evaluation

The performance and energy consumption of the SCALE DRAM system are greatly impacted by the policies implemented by the system. This chapter explores three different types of policy: address mapping policies, scheduling policies, and powerdown policies. Although the hot-row predictor policy also influences system performance, only the simple hot-row policy discussed in section 2.2.4 is used in this thesis. Each of these policies is implemented by a separate module in the SCALE DRAM system, allowing them to be easily interchanged. These policies, though implemented independently, can interact dynamically and influence the system performance. Therefore, although many of these policies have been previously explored, interaction between them can lead to quite different results.

In order to isolate and analyze the effects of a given policy, the system policies are fixed as described in section 4.1. The policy in question is then implemented in place of the default value while the rest of the system policies remain fixed. Sections 4.2, 4.3, and 4.4 evaluate various address mapping, scheduling, and powerdown policies respectively. Each section first describes various policies that could be implemented and the policies' predicted effect on system performance and energy consumption. The results section then presents and analyzes the performance, energy, and energy-delay product of running the benchmarks as discussed in section 4.1, with the appropriate alternative policies in place. At the conclusion of each of these sections, the optimal policy, which minimizes energy-delay product, is chosen. Any time the term "average

performance" or "average energy" are used in the analysis, this refers to the average of the normalized values across all five benchmarks. Therefore, if two benchmarks offer a speedup of 5% and 10% as compared to their base configuration, the average speedup is 7.5%. The dotted line on energy vs. performance graphs represents a constant energy*delay equal to that of the base configuration. Finally, section 4.5 summarizes the findings of the chapter and presents three policy configurations: a high-performance system, a low-energy system, and a system which minimizes the energy-delay product.

4.1 System Configuration

All results in this chapter result from running SCALE-optimized versions of the EEMBC benchmarks discussed in Section 3.3. The benchmarks were run on the SCALE microarchitectural simulator, using the SCALE DRAM model discussed in section 3.1 as the memory backend. The simulator configuration, illustrated in figure 4-1 involves the SCALE microarchitectural simulator, the SCALE cache simulator, and the SCALE DRAM simulator.

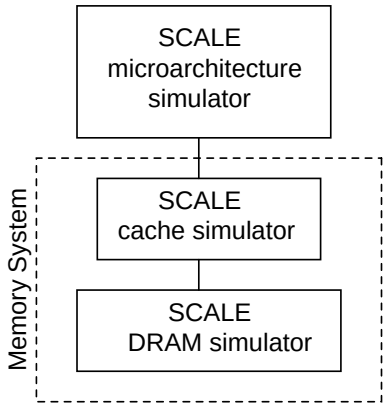

Figure 4-1: SCALE Simulator Configuration

The cache simulator sits between the processor and the DRAM simulator; all memory requests handled by DRAM are cache refills and write-backs. The cache is a 4-bank, non-blocking, 32-way set-associative cache. Each of the 4 banks is 8kB, leading to a cache size of 32kB. As the cache is non-blocking, each bank can handle

up to 32 cache misses, and therefore multiple outstanding requests to the DRAM subsystem, at any time. Cache lines are 32 bytes in size.

The DRAM subsystem and SIP channels operate at 200 MHz, while the SCALE processor operates at 400 MHz. The term cycle may therefore refer to different periods of time depending on context. When discussing DRAM subsystem policies, such as the number of cycles to wait before powerdown, these cycles are the 200 MHz DRAM cycles. Benchmark performance graphs refer to the normalized benchmark execution time as measured in 400 MHz SCALE cycles.

Unless otherwise indicated, the DRAM subsystem is configured as a 4-channel system with one SDRAM chip per channel. Each channel has a data bus width of 8 bits, leading to an effective bandwidth of 16 bits/cycle as the chips are DDR. Each SDRAM chip is 256 Mb, leading to a total system capacity of 128MB. Each 32-byte cache line is interleaved across two channels, with adjacent line addresses mapping to different channels. The Hot-Row Predictor is enabled and operates as discussed in Section 2.2.4. The SDRAM chips power down after a single idle cycle to the Active Powerdown - Fast Exit mode if the chip has one or more open rows, or to the Precharge Powerdown mode if the chip has no open rows. After 50 additional idle cycles in which the chip is in the Active Powerdown mode, the chip will be activated, precharged, and powered down to the Precharge Powerdown mode. Each channel will issue its queued requests in FIFO order. When this configuration is changed to study various policies, the change is noted in that policy's section.

4.2 Address Mapping Policies

The address mapping policy determines how an address is mapped into hardware. This can have a dramatic impact on the performance and energy requirements of the DRAM system. The system policy must determine which channels to map a request into and how to map the request into the banks, rows, and columns of the SDRAM chips.

The SCALE Memory system address mapping policies, illustrated in Figure 4-

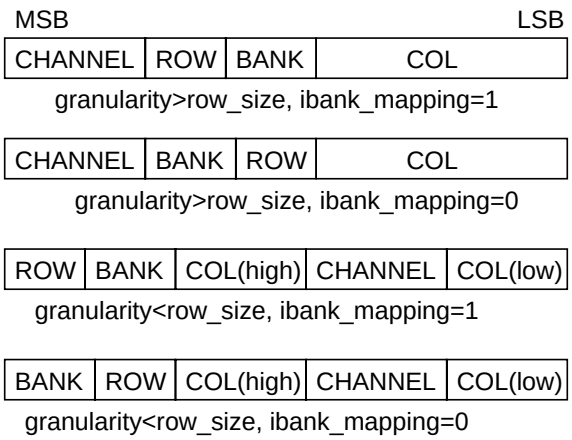

Figure 4-2: Physical Address Translation

2, rely on two parameters: granularity and bank mapping. Granularity determines how many contiguous bytes map to a channel. Bank mapping determines whether contiguous rows in the address space map to the same DRAM bank, or are interleaved across DRAM banks.

4.2.1 Granularity

The address mapping granularity determines how many contiguous bytes are mapped to a channel. If the granularity is less than the size of a memory access request, the request is split and sent to multiple channels. If, for example, the system receives a cache line load of 32 bytes and is configured with a granularity of 8, the request will be spread across 4 channels. If the granularity is 16, the same request will be spread across only 2 channels. The effect of granularity on how a 32-byte cache line is mapped to memory is illustrated in Figure 4-3.

In the case that the granularity is greater than or equal to the request size, but is less than the DRAM row size, the request will map to a single channel but requests that are adjacent in the memory space may map to a different channel. A granularity of 32, for example, would map a cache line to a single channel, but adjacent lines would map to different channels.

In the case where the granularity is greater than or equal to the number of bytes in a row, the granularity is ignored. The address space is linearly divided by the

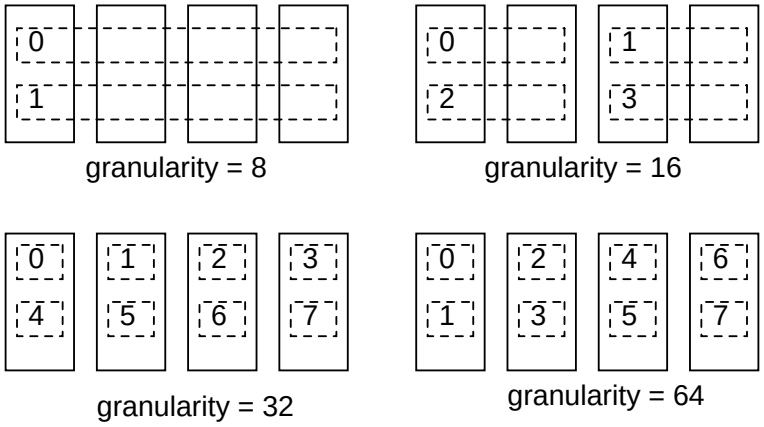

Figure 4-3: Effect of Granularity on Mapping Successive 32-byte Cache Lines

number of channels, and all addresses within one of these divisions map to a single channel.

Granularity can have a dramatic effect on the performance and energy requirements of the memory system. A low granularity in which single accesses are spread across multiple channels will generally result in the best performance, as pieces of a request are handled in parallel. This can have severe energy implications, however, as all channels involved must remain active.

On the other hand, a high granularity will conserve energy while reducing performance. In applications that exhibit spatial locality in memory references, most consecutive requests will only access a single channel. This will cause a performance bottleneck, as one channel must service all the requests. The other channels may be powered down, however, consuming considerably less energy. As granularity is increased, memory accesses may be spaced further apart but still demonstrate this effect.

Results

Figure 4-4 illustrates the effect of granularity on benchmark performance. As can be seen in this figure, performance is inversely proportional to the granularity. Spatial locality in memory references does play a role in this effect, as more localized requests build up in the channel queues, degrading performance but allowing the un-

used channels to power down. However, the performance impact is dominated by the memory access frequency. The rgbyiq benchmark, with an extremely high access frequency, demonstrates a performance hit up to 81% as granularity increases, while the low-frequency routelookup benchmark only demonstrates a 1.7% hit, despite its greater locality. This is due to the fact that at low access frequencies, requests do not arrive quickly enough to fill up the channel queues; there is therefore no bottleneck at the channels. For high-traffic applications, however, the high granularity means successive accesses will back up in the channel queues, leading to the performance degradation.

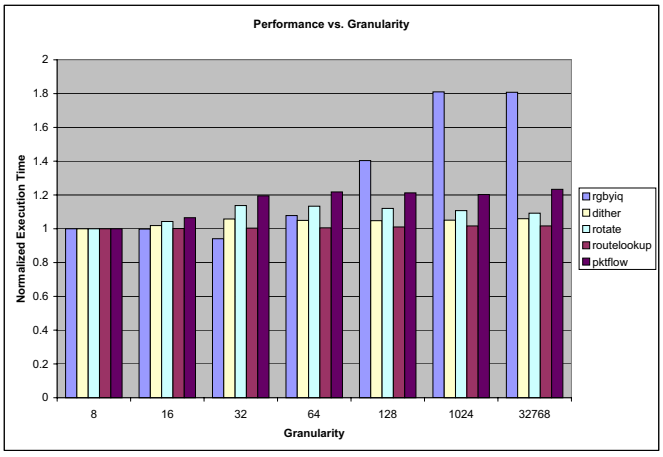

Figure 4-4: Granularity vs. Energy

The performance cost of granularity 16 vs. 8 should be minimal, as the SIP channel only operates at twice the bandwidth of each channel. Each channel is 8 bits wide, but as it is DDR, has an effective bandwidth of 400 MB/sec. The SIP channel, operating at 200 MHz with a 32-bit data width, has a bandwidth of 800 MB/sec. Therefore, as a granularity of 16 will use two channels for a given access, each channel will contribute 400 MB/sec for a total of 800 MB/sec, thus saturating the SIP channel. However, as can be observed in Figure 4-4, a granularity of 16 introduces a performance hit of up to 6.5% over a granularity of 8. This performance

hit is due to the fact that the data bus bandwidth of the involved channels is not saturated; row misses and bus turnaround between reads and writes lead to dead cycles on the data bus and thus reduce effective channel bandwidth. A granularity of 8 compensates for these dead cycles, maximizing performance but at a considerable energy cost due to the activation of all chips.

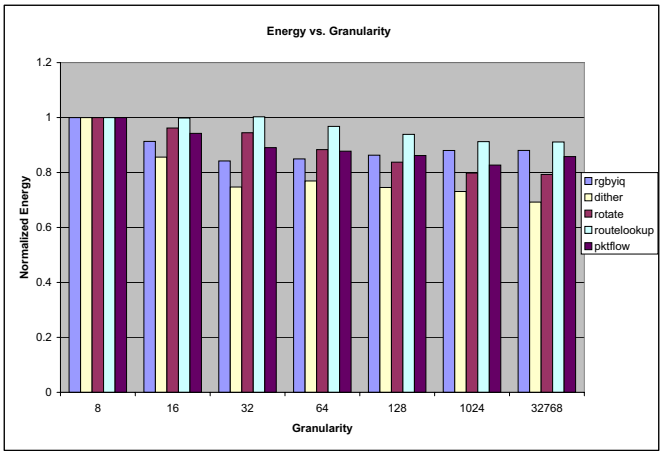

Figure 4-5: Granularity vs. Energy

The energy characteristics, as seen in Figure 4-5 behave as expected. As the granularity increases, fewer channels are in operation at any time. These unused channels may be powered down, conserving energy. This effect is limited, however, as the active channels must remain active longer. The energy cost continues to drop as granularity is increased.

As can be seen in the energy*delay graph of Figure 4-6, these energy savings outweigh the performance penalties of granularity for the low-frequency benchmarks. For high-frequency benchmarks, however, the massive performance penalty introduced by large granularity values invalidates the energy savings.

The performance penalty of higher granularity outweighs the energy savings for high-frequency benchmarks, leading to an energy-delay penalty of up to 60% versus the 8-byte case. As illustrated in Figure 4-7, a granularity of 16 is optimal. The aver-

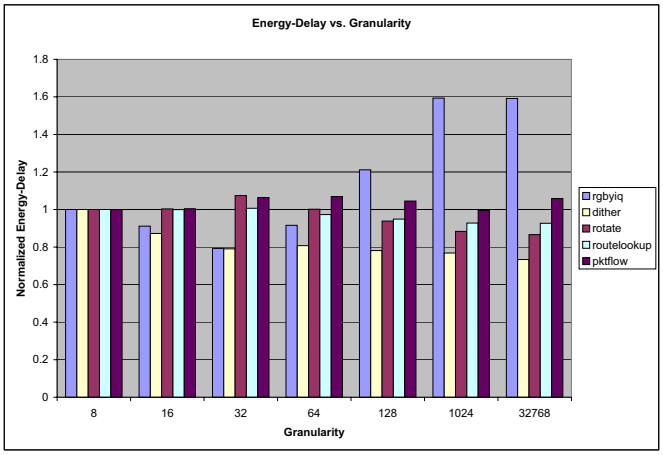

Figure 4-6: Granularity vs. Energy*Delay

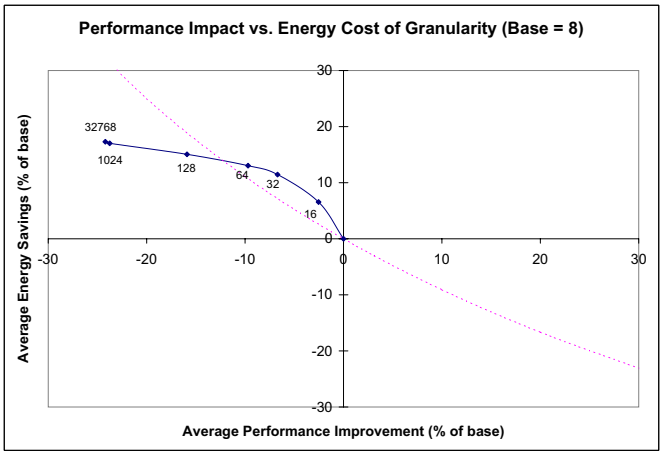

Figure 4-7: Performance Improvement vs. Energy Savings for Granularity

age performance degradation with granularity 16 is only 2.5%, while average energy savings are 6.5%. Although more energy may be saved by increasing granularity, Figure 4-7 illustrates that these energy savings as compared to performance degradation are greatly diminished as granularity is increased above 32.

4.2.2 DRAM Bank Mapping

Bank mapping also effects performance and energy requirements of the system. If bank mapping is set to 0, the DRAM chip's address space is divided by the number of DRAM banks, and contiguous rows are assigned to the same bank. If bank mapping is set to 1, adjacent rows in the address space are spread across the DRAM banks. Finer-grained bank interleaving is undesirable, as all accesses referencing a single row may be cleanly pipelined, requiring no row precharges or activations. Interleaving accesses that would fit within a single row across banks therefore provides no performance advantage, but may require more energy as multiple banks are in operation.

A bank mapping of 1 should exhibit greater performance than a bank mapping of 0 in cases where the memory references exhibit spatial locality. As each bank may have an active row, four adjacent rows may be kept active for the localized accesses if bank mapping is 0. If bank mapping is 1, these accesses will only map to a single bank; as a bank may only have one row open at any time, this may require considerably more precharges and activations. However, fewer banks will be in operation, potentially reducing energy consumption.

Results

Figures 4-8 and 4-9 illustrate the effect of DRAM bank mapping, with the same configuration as the simulations for the granularity tests of section 4.2.1 and a granularity of 16. Although for certain memory access patterns, a bank mapping of 0 is preferable, the high-frequency benchmarks exhibit a significant degradation in performance; as this requires the chip to remain active for a longer period of time and requires more row precharges and activations, this also leads to the consumption of more energy. As

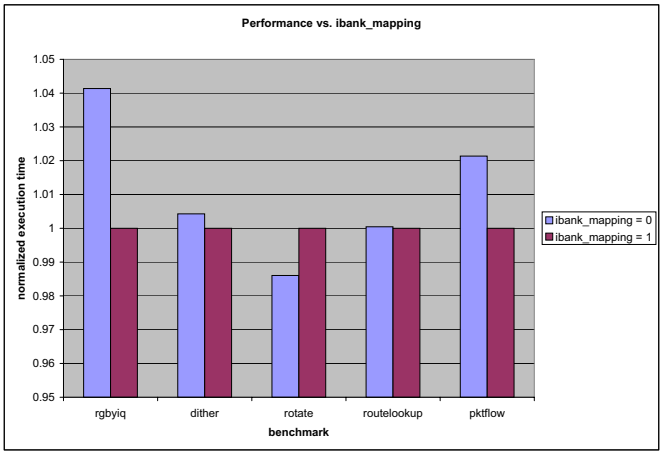

Figure 4-8: Ibank Mapping vs. Performance

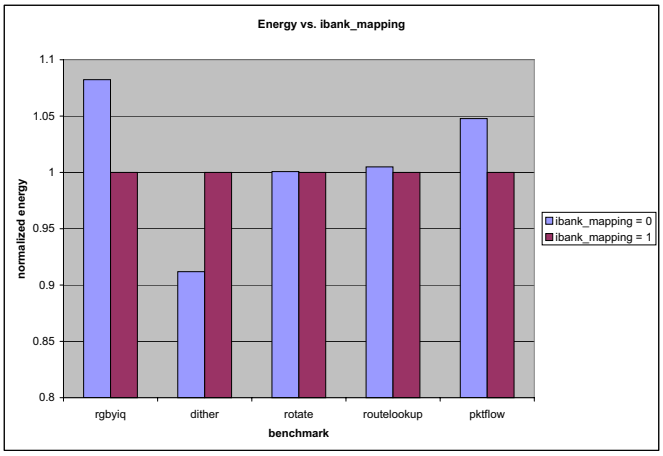

Figure 4-9: Bank Mapping vs. Energy

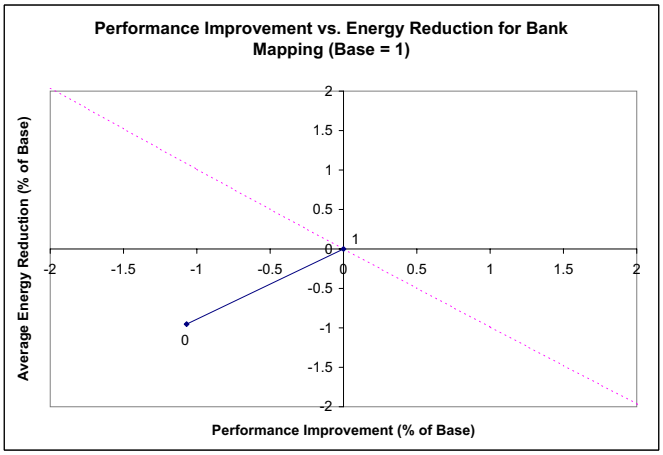

Figure 4-10: Performance Improvement vs. Energy Savings for Bank Mapping

illustrated in Figure 4-10, a bank mapping of 0 introduces on average a 1% penalty to both energy and performance. A bank mapping of 1 is therefore preferable for both high-performance and low-energy applications.

4.3 Access Scheduling Policies

The access scheduling policy determines which request should be selected from the queued requests for issue to the DRAM controller. For cases in which the system receives requests at a rate greater than it can service them, requests will build up in the channel request queues. These requests may be reordered to increase efficiency, thus improving performace and reducing energy requirements. This section discusses reordering policies based on the row, bank, or operation of the queued requests.

4.3.1 Row-Based Scheduling Policies

Access Scheduling policies in which the accesses are issued according to the row they reference are explored extensively by Rixner et. al[8]. The scheduler must track which rows are open and issue requests accordingly. In one such policy, the Open

Row policy, all requests to a single row are issued before any other requests are issued, ensuring that there are no unneccessary precharges and activations. In the Closed Row policy, a row is precharged as soon as there are no more outstanding requests to that row. The row-based scheduling policies discussed in this document differ from those discussed by Rixner et. al[8] in that precharge is not controlled by the scheduler, but by the hot-row predictor. For accurate modelling of the Open Row policy, the hot-row predictor must be disabled, only precharging when an incoming request is to a different row. Implementation of the Closed Row policy would require that the access scheduler override the hot-row predictor, thus violating the modularity of the design. Only the Open Row policy is therefore evaluated here.

Adaptive Row-Based Policies

In order to gain the advantages of both the Open and Closed policies as discussed in Rixner[8], the hot-row predictor can be used to create an adaptive row-based scheduling policy. In cases where most accesses are to the same row, the hot-row predictor will act essentially as an Open Row policy, thus providing the benefits of the Open policy. In the case where accesses are spread across rows, the hot-row predictor will auto-precharge after most accesses. The system will act as though it is implementing the Closed policy. The scheduler therefore need not concern itself with row precharges; the system will adapt to the optimal policy.

4.3.2 Bank Interleaved Scheduling

An alternative approach to access scheduling, which may be integrated with the other policies, is bank-interleaved scheduling. A bank-interleaved scheduler will attempt to issue requests to different banks upon each issue. This will load a request into the bank controller of as many banks as possible, leading to increased efficiency of the DRAM controller.

4.3.3 Read-Before-Write Scheduling

A final scheduling approach, adapted from the Rixner *load-over-store*[8] policy, involves reordering accesses according to the operation they are to perform. As switching between reads and writes introduces significant overhead, reordering accesses such that all reads are issued together before writes are issued can enhance efficiency of the DRAM controller. It is important, however, that before a read is issued, the queue is checked for prior writes to the same address. Otherwise, memory access ordering is not maintained. Additionally, the policy periodically flushes old writes out of the queue, preventing write request starvation.

4.3.4 Results

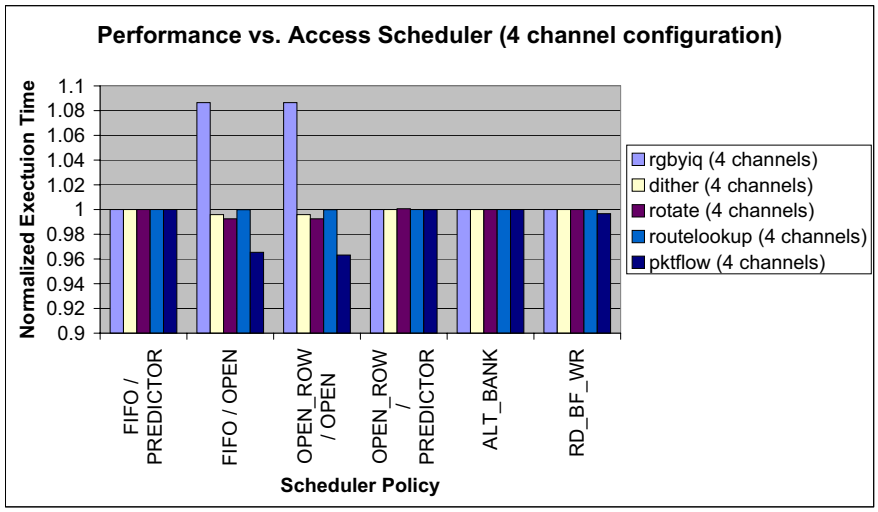

Figure 4-11: Access Scheduler Performance Impact

As illustrated in Figure 4-11, access scheduling is ineffective for the standard 4-channel configuration of the SCALE DRAM subsystem for the selected benchmarks with the exception of a slight advantage of the Read-Before-Write policy for the pktflow benchmark. Any other differences are due simply to the effects of the hot-row predictor. This ineffectiveness of the schedulers is due to two factors: the system

services requests quickly enough that not enough requests queue up in the Channel Request Buffers for the scheduling algorithms to be effective, and the SCALE processor cache filters out most locality in memory accesses; scheduling algorithms that take advantage of spatial locality such as the Open-Row policy therefore are limited in effectiveness.

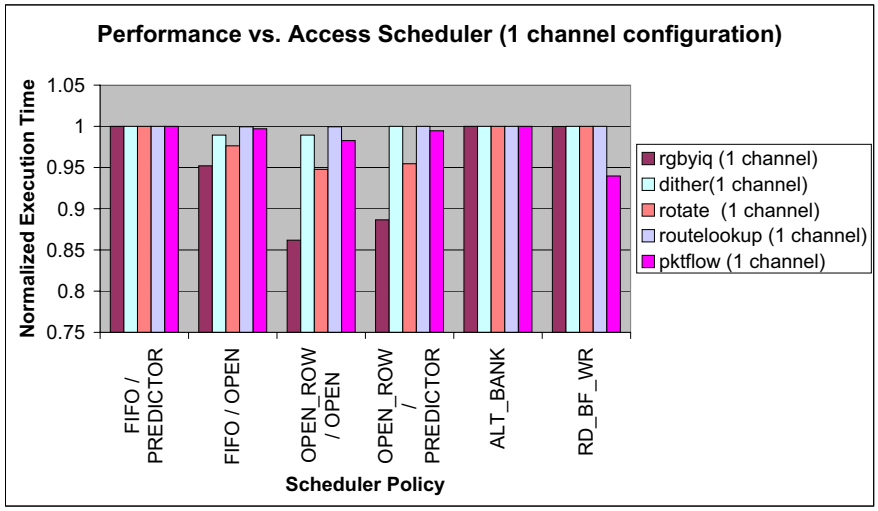

Figure 4-12: Access Scheduler Performance Impact (1-channel configuration)

These policies may still prove useful, however, in cases where a large number of memory accesses build up in the queues. As the total size of the open SDRAM rows is significantly larger than the cache, the policies may still take advantage of large-scale spatial locality. This effect is demonstrated in Figures 4-12 and 4-13. These charts demonstrate the results of running the benchmarks in which the memory system contains only a single channel. This single-channel configuration, in which the channel bandwidth is considerably less than the SIP bandwidth, allows a large number of requests to build up in the queues. The performance effects are modest, but as effective scheduling increases performance as well as reducing energy, the Energy*Delay impact can be substantial for certain benchmarks.

Although the Open Row scheduler with the Hot-Row Predictor enabled is less effective than the Open-Closed hybrid with the Hot-Row Predictor disabled for the

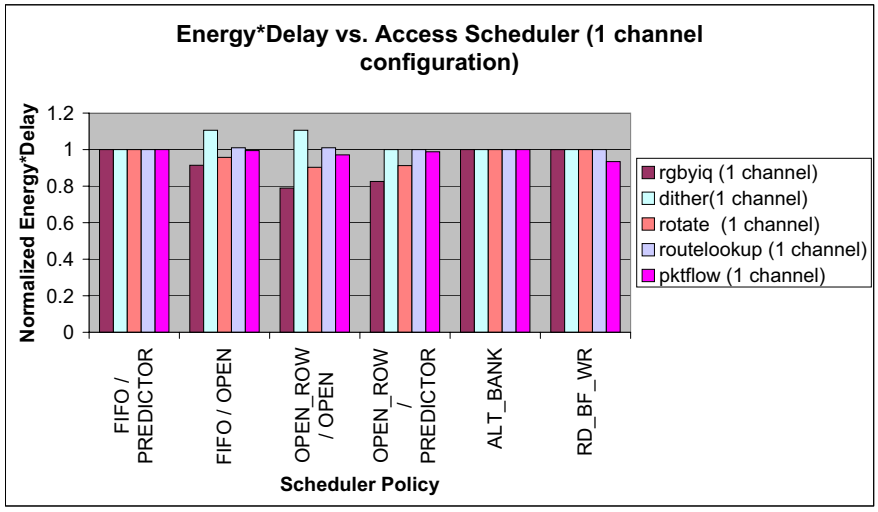

Figure 4-13: Access Scheduler Energy*Delay

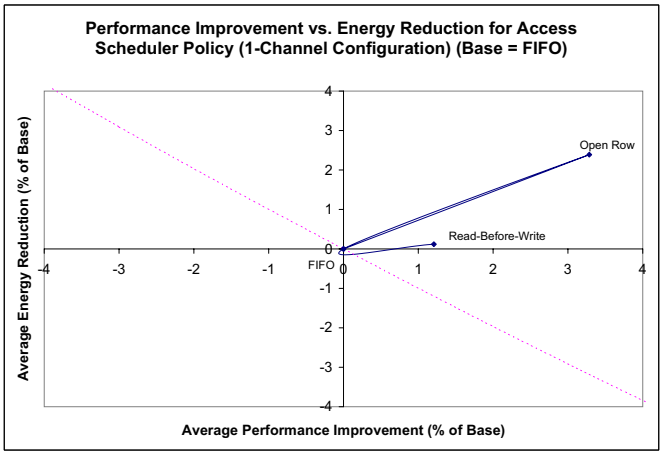

Figure 4-14: Performance Improvement vs. Energy Savings for Access Scheduler

single-channel configuration, it incurs no performance penalty for the 4-channel configuration. It is therefore the optimal policy, as it incurs no penalty for typical applications but may lead to improvements in cases where the memory request frequency is extremely high. As illustrated in Figure 4-14, for the 1-channel configuration this scheduler introduces an average performance improvement of 3.2% and an average energy savings of 2.3%.

4.4 Powerdown Policies

Perhaps the most important task that an energy-aware DRAM controller must perform is the intelligent management of DRAM power states. As power state selection involves a tradeoff between power consumption and resynchronization time, the controller's powerdown policy can greatly impact both performance and energy consumption of the system. Powering down too rapidly or into too deep a power state will adversely impact performance, as the resynchronization cost must be paid each time the chip is to be woken. If the chip powers down too late, energy will be wasted as the chip sits idle in a higher power state.

Delaluz et. al [2] have extensively explored various hardware and software-directed power state transition policies. However, the power states modeled in this work are only loosely representative of the power states of DDR-II SDRAM. In addition, the memory access patterns generated by the SCALE processor are likely to be considerably different from those used in this work. Because of these differences, it is valuable to re-explore these policies for the SCALE DRAM system.

In the SCALE DRAM system, chip powerdown is controlled by a powerdown scheduler module. The module monitors DRAM chip activity, and signals the DDR controller when the chip should change power states. The chip is automatically woken up as soon as it receives a request from the access scheduler.

4.4.1 Powerdown Sequences

As illustrated in Figure 2-6, the DDR-II power states present a complex network of possible state transitions. Effective exploration of all possible state transition sequences is impractical. The policies here explored therefore employ a simplified set of state transitions as illustrated in Figure 4-15. If a chip is idle, it will eventually power down to a "shallow" powerdown state. As different power states are available depending on whether a chip has an open row or has been precharged, there is an open-row shallow powerdown state and a precharged shallow powerdown state. As the deeper DDR-II powerstates require that the chip be precharged, there is only a single deep power state. Transition from an open-row state to a precharged state includes an implicit wakening of the chip, precharge, and then transition to the target state. A single DDR-II power state may be mapped to each of these simplified states, or this state may be dynamically assigned by the powerdown policy.

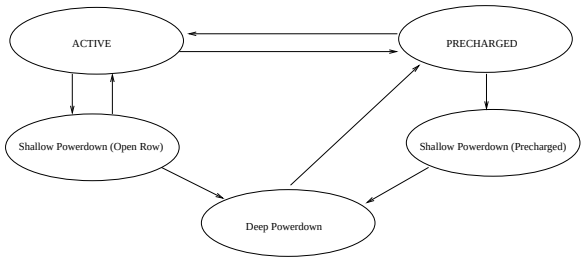

Figure 4-15: Simplified SDRAM Power State Transitions

There are a number of possible mappings from the DDR-II power states to the simplified states of Figure 4-15. Table 4.1 illustrates the mappings implemented in the SCALE DRAM simulator. The shallow powerdown for the case when the bank is precharged can be Precharge Powerdown or Self Refresh. As Self-Refresh has a huge resynchronization cost, it is only considered as a deep powerdown state. The shallow, precharged powerdown state is therefore always precharge powerdown. The shallow state when the bank has an open row, however, can be either Active Powerdown, Fast-Exit or Active Powerdown, Slow-Exit. The chip could also precharge all banks, and then power down to Precharge Powerdown. The deep powerdown state can be either Precharge powerdown or Self Refresh. This section discusses the assignment of

Powerdown Sequence	Shallow Powerdown State (Open Row)	Shallow Powerdown State (Precharged)	Deep Powerdown State
AAPDF	Active Powerdown, Fast-Exit	Precharge Powerdown	Precharge Powerdown
AAPDS	Active Powerdown, Slow-Exit	Precharge Powerdown	Precharge Powerdown
AAPDFSR	Active Powerdown, Fast-Exit	Precharge Powerdown	Self-Refresh
AAPDSSR	Active Powerdown, Slow-Exit	Precharge Powerdown	Self-Refresh
APPD	Precharge Powerdown	Precharge Powerdown	Precharge Powerdown
APPDSR	Precharge Powerdown	Precharge Powerdown	Self-Refresh

Table 4.1: Powerdown Sequences

Power State	Energy/Cycle (nJ)	Resynchronization Time Cycles
Active Powerdown, Fast Exit	0.135	2
Active Powerdown, Slow Exit	0.063	6
Precharge Powerdown	0.032	6
Self-Refresh	0.027	200

Table 4.2: DDR-II Powerstate Statistics

one of these DDR-II power state to each of these simplified states.

Shallow Powerdown State Selection

As the shallow powerdown state for the case when a row is open can be one of several states, selection of the appropriate state requires an understanding of the energy and performance impacts of each selection. As shown in Table 4.2, each successive state consumes less energy but requires a greater resynchronization time.

As shown in Figure 4-16, the greater resynchronization costs of the lower-energy states are directly reflected in system performance. Although Active Powerdown, Slow-Exit and Precharge Powerdown mode have the same resynchronization cost, the Precharge-Powerdown mode also reflects the cost of precharging all banks before powering down. The performance impact is directly correlated with the memory access frequency of the benchmark; higher-traffic applications see a greater slow-down

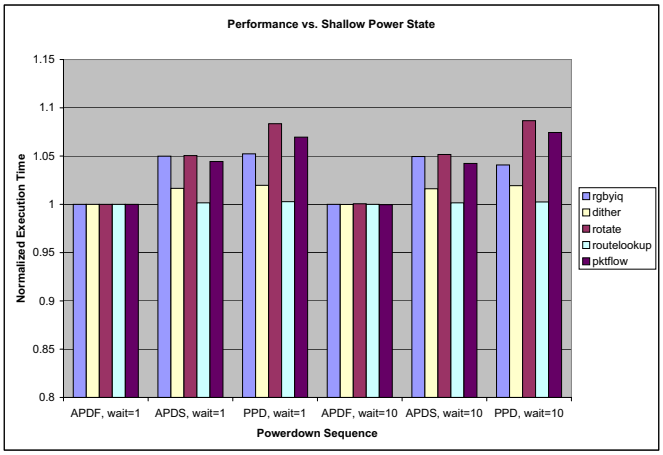

Figure 4-16: Shallow Power State vs. Performance

when using the lower-energy DDR-II states as the open-row shallow powerdown state.

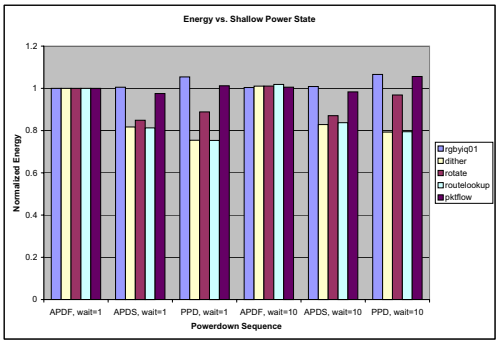

Figure 4-17: Shallow Power State vs. Energy

Although the lower-energy states incur a performance penalty, they lead to a dramatic reduction in energy consumed for low-traffic benchmarks, as illustrated in Figure 4-17. On average, as illustrated in Figure 4-19, the Active Powerdown - Slow Exit state provides significant energy savings over Active Powerdown - Fast Exit (10.8%) while only impacting performance by 3.25%. As shown in the Energy*Delay plot 4-18, this effect is much greater than the performance degradation for low-traffic applications. However, for high-traffic applications, the substantial performance degra-

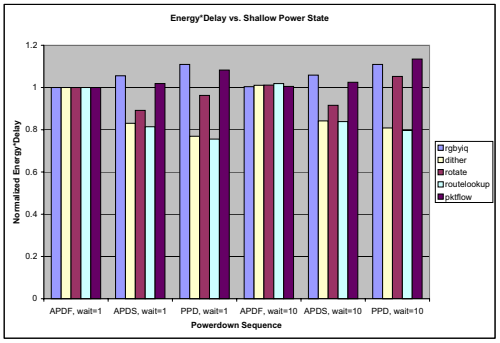

Figure 4-18: Shallow Power State vs. Energy*Delay

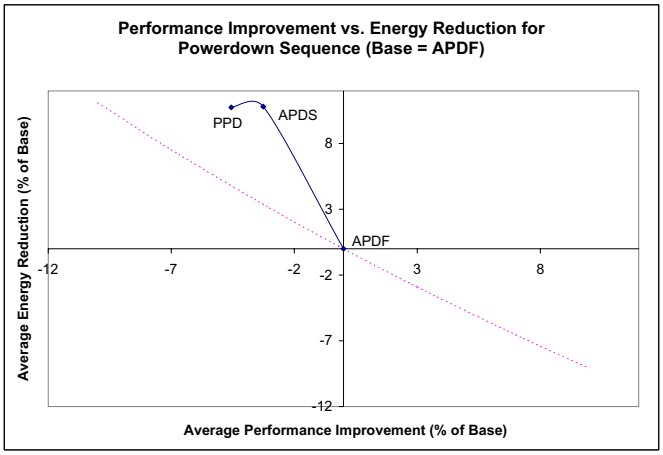

Figure 4-19: Performance Improvement vs. Energy Savings for Shallow Powerdown State

dation leads to undesirable Energy*Delay characteristics in the lower-energy states. This suggests that a policy that can alter the shallow powerdown state as a function of the memory access frequency could greatly improve the general Energy*Delay performance of the system. If a single state must be selected for the shallow powerdown state, however, Active Powerdown - Slow Exit is the optimal decision as it exhibits the best energy-delay product.

Deep Powerdown State Selection

As the synchronization time of the Self-Refresh mode is dramatically larger than that of the other modes, but the energy consumption is only marginally better than precharge powerdown, the chip must be idle for a considerable length of time before entering Self Refresh mode is advisable. The benchmarks used in this study access the memory system too frequently for the chips to ever be idle for this long. The deep powerdown state is therefore always Precharge Powerdown, under the assumption that Self Refresh will ultimately be entered after a very large number of idle cycles.

4.4.2 Static Powerdown Policies

The simplest power management policy is dubbed by Delaluz et. al[2] as the Constant Threshold Predictor, or CTP. The CTP waits for the chip to be idle for a certain number of cycles, then transitions the chip to the shallow powerdown state. If the chip remains idle past a second threshold value, the chip is transitioned into the deep powerdown state. Selection of the shallow powerdown state, as discussed above, as well as modification of these threshold values can significantly impact the performance and energy characteristics of the system.

Shallow Powerdown Wait Evaluation

The number of idle cycles the controller waits before powering down should have an inverse energy-delay relationship; powering a chip down too early can adversely impact performance due to the resynchronization cost, and powering a chip down too

late can lead to wasted energy in a high-energy state. Fan et. al [5] have demonstrated this relationship, but have shown that for a system with a two-level cache, the energy-delay product is best when the chip powers down immediately.

As the SCALE DRAM system is fully pipelined, the powerdown scheduler may wake a chip as soon as a memory request enters the DDR controller pipeline. The time it takes for the memory request to propagate through the controller pipeline may therefore overlap with the time spent waking the chip, thus reducing the performance impact of resynchronization.

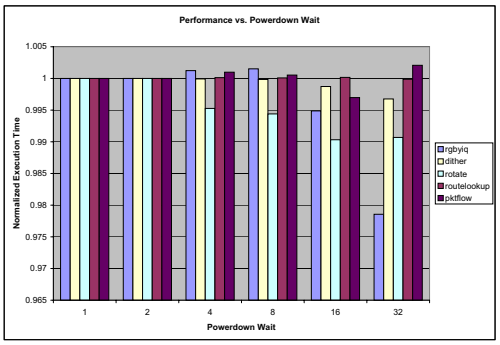

Figure 4-20: CTP Powerdown Wait vs. Performance

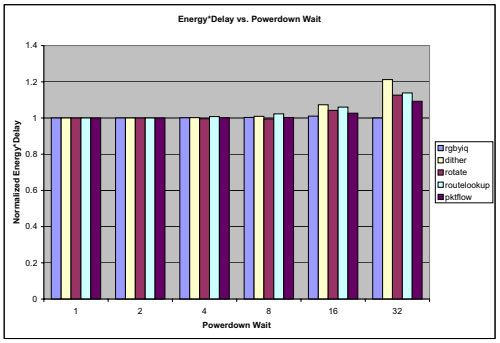

Figure 4-21: CTP Powerdown Wait vs. Energy*Delay

As illustrated in Figure 4-22, the performance benefits of waiting to power down the chip are greatly outweighed by the energy savings of powering down immedi-

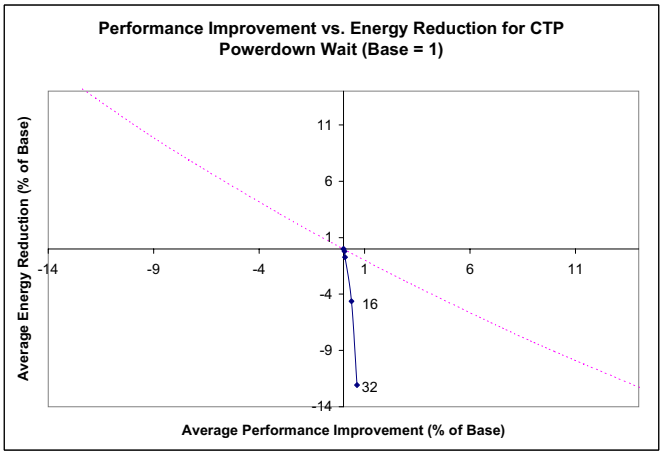

Figure 4-22: Performance Improvement vs. Energy Savings for CTP Shallow Powerdown Wait

ately. Waiting 16 cycles to power down only introduces an average performance improvement of 0.38%, while increasing energy consumption by 4.6%. The chip should therefore be powered down as soon as it becomes idle.

CKE Penalty Effects

For the SCALE DRAM system, the energy-delay product is best when the chip powers down immediately, as shown in Figure 4-21. However, the performance impact of the powerdown wait is surprising. As illustrated in Figure 4-20, performance may actually degrade with certain, higher powerdown wait values.

This effect is due to the fact that when a chip is powered down, it must remain powered down for a minimum number of cycles (tCKE). If a chip is awakened soon after it is powered down, it must wait for the chip to have been idle for this number of cycles before it can be activated. The total number of cycles spent waiting for this condition to be met correlates strongly with the number of cycles required to execute the benchmark, as illustrated in Figure 4-23. The performance hit is so strong when powerdown_wait is 32 because that happens to be close to the rate at which the system is issuing requests; the chip keeps powering down immediately before receiving a new

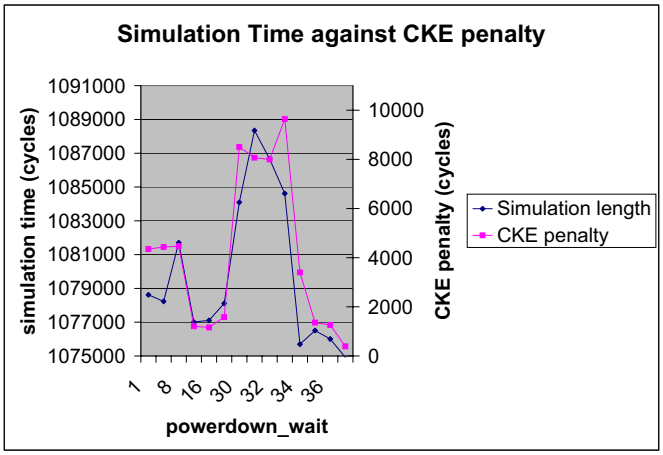

Figure 4-23: Performance against CKE penalty

request, thus requiring that it wait for the CKE constraints to be satisfied in addition to paying the resynchronization costs.

In order to avoid this effect, and thus maximize performance at minimal energy cost, the chip should therefore be powered down as soon as possible.

Deep Powerdown Wait Evaluation

Figures 4-24 - 4-27 illustrate the effects of deep powerdown wait, or the number of idle cycles before the chip transitions from the shallow powerdown mode to its deep powerdown mode. These results are generated by a system using the AAPDF powerdown sequence, with a shallow powerdown wait of one cycle. As the resynchronization cost of the precharge powerdown state is greater than that of active powerdown, entering the deep powerdown state after fewer cycles does adversely impact performance. However, the substantial energy savings introduced by entering this state earlier produce desirable Energy*Delay characteristics for the lower wait values. As with the shallow powerdown wait, the low resynchronization costs coupled with high energy savings in the lower states suggest that the chip should transition from the shallow powerdown state to the deep powerdown state after 10 idle cycles.

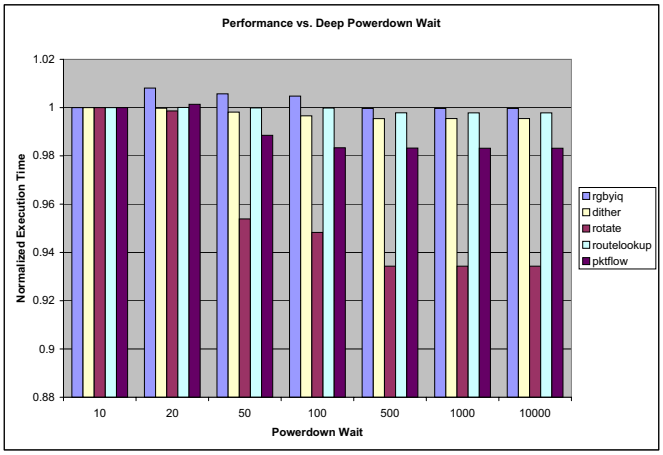

Figure 4-24: CTP Deep Powerdown Wait vs. Performance

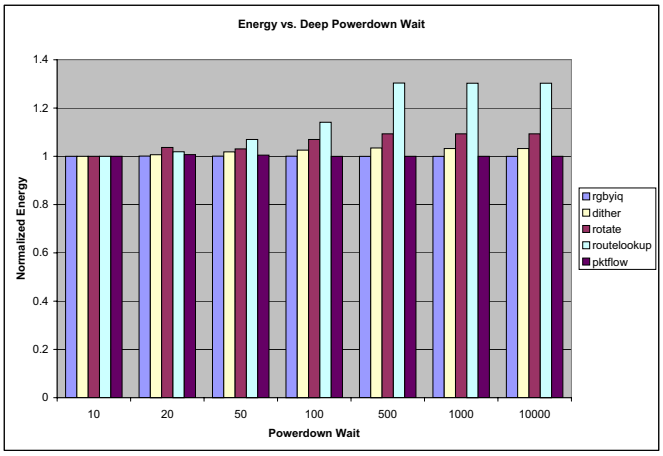

Figure 4-25: CTP Deep Powerdown Wait vs. Energy

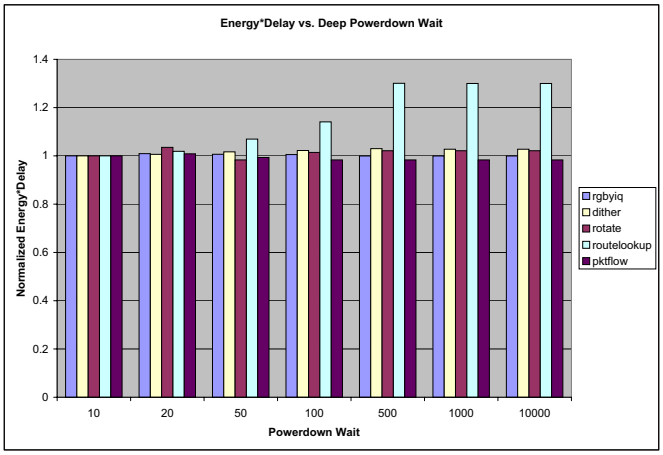

Figure 4-26: CTP Deep Powerdown Wait vs. Energy*Delay

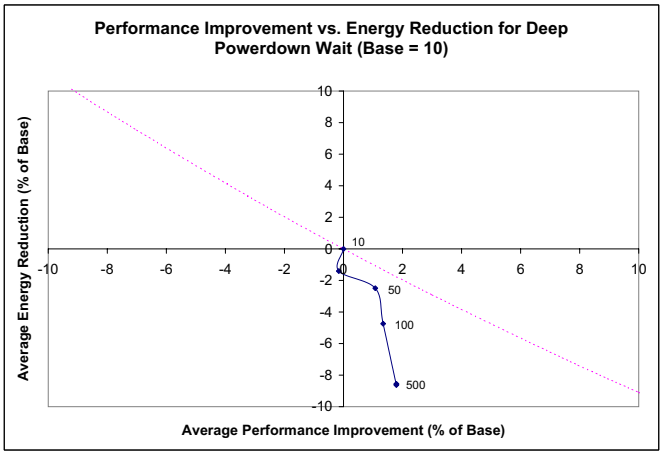

Figure 4-27: Performance Improvement vs. Energy Savings for CTP Deep Powerdown Wait

4.4.3 Dynamic Powerdown Policies

As discussed in sections 4.4.1 and 4.4.2, the performance and energy impact of different DDR-II to shallow power state mappings and powerdown waits for Constant-Threshold Predictor policies vary considerably for different benchmarks, with different shallow powerdown states proving preferable for different benchmarks. This suggests that if the Powerdown Scheduler can adapt to the current access patterns, it may select the optimal mapping for the current application.

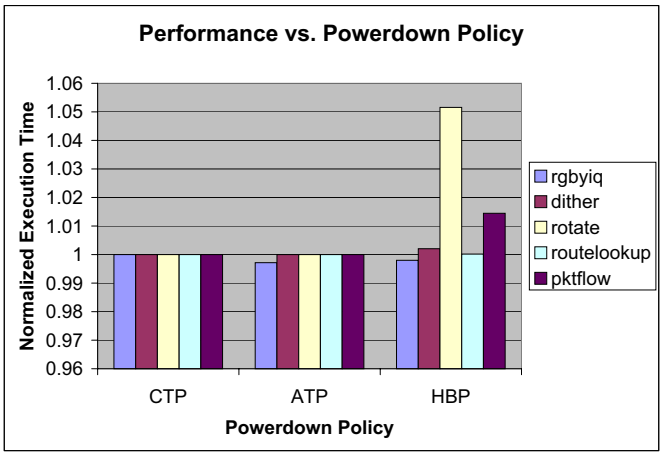

Figure 4-28: Powerdown Policy vs. Performance

Adaptive Threshold Predictor, or ATP[2] policies, alter the powerdown threshold as a function of the current access patterns. Each time a chip is awakened, the number of cycles that it has been idle is evaluated. If the chip was idle for a large number of cycles, it should have powered down earlier; the threshold is therefore reduced. Conversely, if the chip was only powered down for a few cycles, the chip should have stayed active to handle the next request; the threshold is therefore lengthened. However, since powering down immediately in the case of the CTP scheduler proved optimal, the ATP policy should provide minimal, if any improvement. As illustrated in Figures 4-28 - 4-30, this holds true; ATP only provides a minimal benefit over the CTP policy.

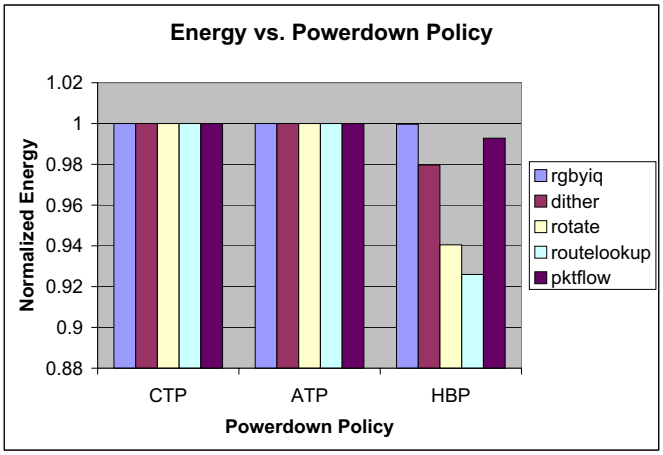

Figure 4-29: Powerdown Policy vs. Energy

Alternatively, the Powerdown Scheduler may alter the shallow powerdown state to match that which is optimal for the current benchmark. As the selection of the shallow powerdown state introduces large variations in both the energy and performance of the benchmarks, an effective policy will modify the shallow powerdown state to whatever state is optimal for the current access pattern. The HBP policy, a modified version of the Delaluz policy of the same name[2], does just this. The policy adjusts the shallow powerdown state as a function of how long the chip has been powered down upon being awakened. If it has only been powered for a short time, the shallow powerdown state should be Active Powerdown, Fast-Exit, as the resynchronization costs are lower. However, if the chip has been powered down for some time, the resynchronization costs of the deeper power states will be of minimal impact while allowing significant power savings.

For low-frequency applications, the HBP policy does present improvements up to 10% over the CTP policy with Active Powerdown, Fast-Exit as the shallow power-state, which is the optimal shallow power state for the high-frequency benchmarks. HBP only introduces a minimal penalty for the high-frequency benchmarks when compared with their optimal policy, but introduces significant improvements for bench-

marks for which the AAPDF policy is not optimal. As illustrated in Figure 4-31, HBP introduces average energy savings of 3.22% when compared to the CTP policy, while only degrading the average performance by 1.3%.

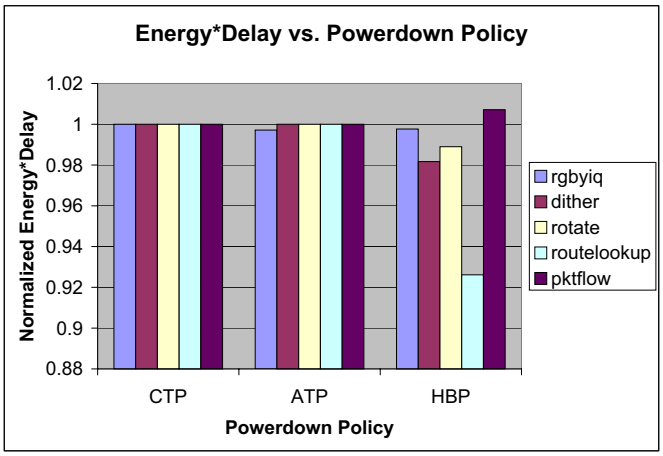

Figure 4-30: Powerdown Policy vs. Energy*Delay

4.4.4 Impact of Hot-Row Predictor

The hot-row predictor can greatly impact the power scheduler performance, as the shallow powerdown state depends on whether the chip has an open row or not. If the hot-row predictor always precharges rows, for example, the chip will never enter the open-row shallow powerdown state. The AAPDS will behave like the poorly-performing APPD policy, since the chip will never have an open row and will therefore always enter the Precharge Powerdown mode. This impact is illustrated in Figures 4-32 and 4-33. Alternatively, if the hot-row predictor never precharges rows until forced to do so by a request to a different row, the chip will waste energy in a higher-energy state. In order to take advantage of the shallow modes with lower resynchronization times while not paying the energy cost of using these open-row states for cases in which the row should be closed, the Hot-Row Predictor is essential.

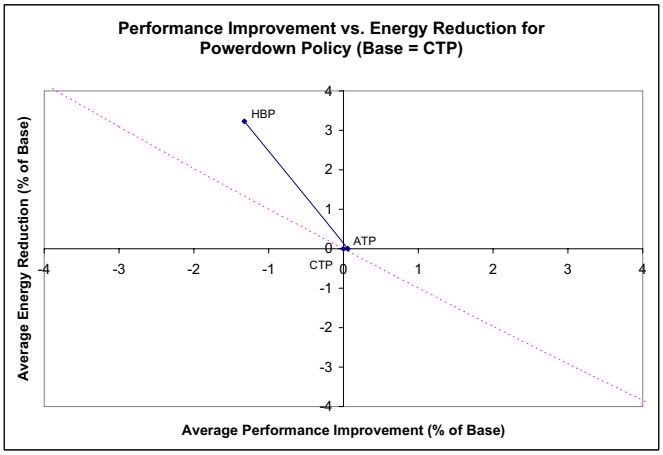

Figure 4-31: Performance Improvement vs. Energy Savings for Powerdown Policy

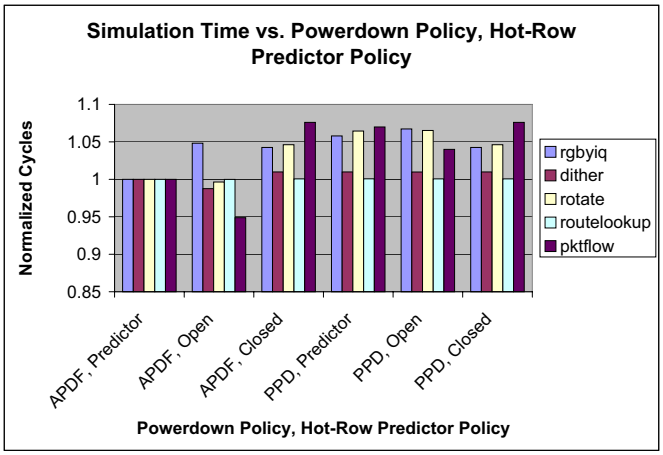

Figure 4-32: Performance Impact of Hot-Row Predictor on Powerdown Policies

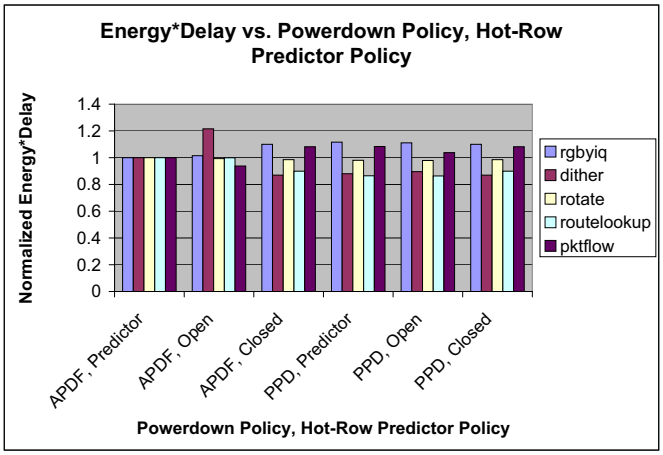

Figure 4-33: Energy*Delay Impact of Hot-Row Predictor on Powerdown Policies

4.5 Summary

Figure 4-34 illustrates the relative impact of the policies discussed in this section, ignoring the second-order effects due to interaction between these policies. Granularity has the greatest overall effect on system performance and energy consumption. The powerdown sequence can also have up to a 12% impact on energy consumption, with a smaller impact on performance. As the memory system is not taxed enough by these benchmarks for a significant number of concurrent requests to queue, the impact of the access scheduler is minimal.

4.5.1 Second-Order Effects Due to Policy Interaction

Although the policies operate independently, their effects are not independent. The granularity of 16 leads to only half the channels being used for a given operation; the other chips may power down as indicated by the powerdown policy. A different granularity would lead to different idle intervals between operations, thus impacting which powerdown policy would be preferable. The granularity also determines how many requests back up in a channel queue, thus determining whether the access scheduling policy will be of use. The hot-row predictor influences which powerdown

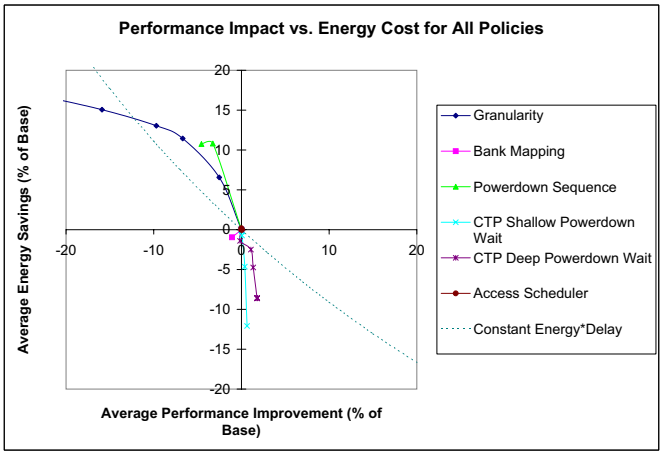

Figure 4-34: Performance Improvement vs. Energy Savings for All Policies

states are used, which in turn influences how quickly a channel can service its queued requests. These complex interactions between policies enhance the Energy*Delay performance benefits that the policies introduce; the whole is indeed greater than the sum of its parts.

4.5.2 Policy Selection

Selection of the appropriate policies for use in the SCALE DRAM Subsystem requires careful weighing of the importance of performance versus energy savings, as there is a marked trade-off between these two metrics as illustrated in Figure 4-34. If performance is the only metric of interest, chips should never power down, and granularity should be as small as possible (8 bytes). If, on the other hand, the system is to consume as little energy as possible, the granularity should be very large, and the powerdown schedulers should use a CTP scheduling policy, powering down to the Active Powerdown - Slow Exit state immediately. If the energy*delay product is to be minimized, granularity should be mid-range (16 or 32), and an HBP powerdown scheduling policy should be used. Although the access scheduler introduces a minimal impact, since it improves both energy and performance the Open Row scheduling

policy should be used in all three cases.

Figures 4-35 - 4-37 illustrate the performance, energy consumption, and energy*delay of three systems for the five benchmarks used in this thesis. System 1 is the high-performance, high-energy system described above. System 2 is the low-energy, low-performance system described above. System 3 is the system which minimizes the energy*delay product, with granularity 16. All three of these configurations are valuable in different situations, but for a general-purpose energy-aware memory system, System 3 is the optimal configuration as it minimizes the energy*delay product, consuming an average of 41.8% less energy than System 1, while only degrading average performance by 8.8%.

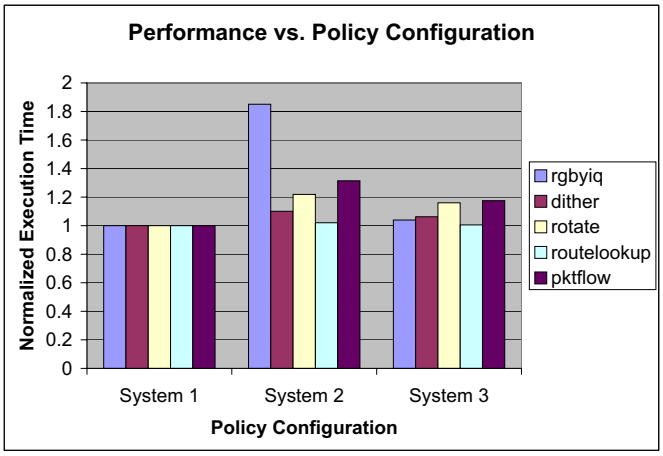

Figure 4-35: Performance vs. Policy Configuration

It is important to note that these energy savings are only calculated during the execution of the benchmarks. In practice the processor will often be idle, thus leading Systems 2 and 3 to consume considerably less energy than System 1.

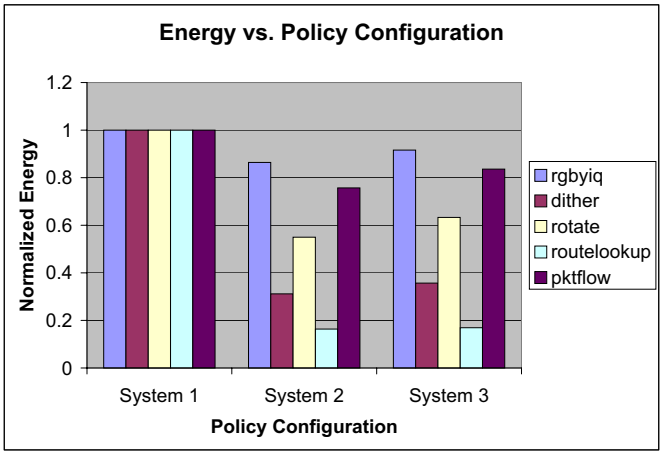

Figure 4-36: Energy Consumption vs. Policy Configuration

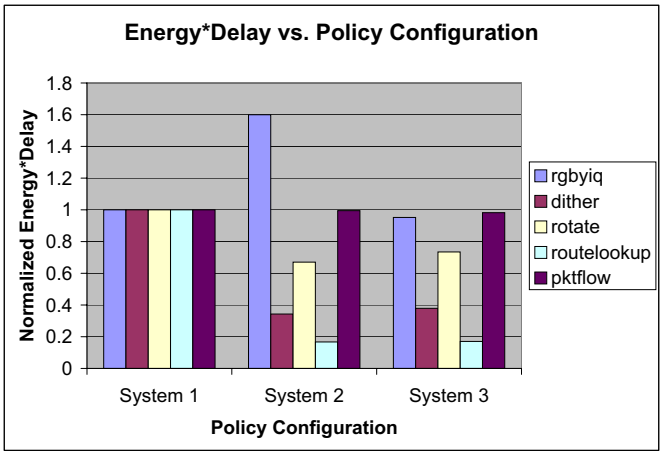

Figure 4-37: Energy*Delay vs. Policy Configuration

Chapter 5

Conclusion

This thesis demonstrates that a properly-designed DRAM subsystem using DDR-II SDRAM chips can significantly reduce DRAM energy consumption while only minimally degrading performance. This requires appropriate implementation and selection of various system policies, including the address-to-hardware mapping policy, the DRAM chip powerdown policy, and the memory access scheduling policy. Different policies introduce significant variation in the system's energy consumption and performance. The thesis therefore proposes the optimal combination of system policies to minimize the energy*delay product, leading to average energy savings of 41.8% with an 8.8% performance degradation.

5.1 Future Work

Much work remains to be done in energy-aware DRAM systems. By no means does this thesis include an exhaustive study of all possible system policies; further policy developments may lead to increased energy and performance gains. Additionally, the thesis only discusses hardware policies. Energy-aware software policies can complement the hardware policies discussed in this thesis.

An energy-aware virtual memory system could greatly improve performance of this system. As address mapping policies must trade energy for performance, as demonstrated in this thesis, it would be valuable for the address mapping policy to

adapt to the incoming access pattern: if the requests are of low frequency, the address mapping policies that lead to energy savings at the cost of performance could be used; however, for high-frequency accesses, the performance cost of these low-energy policies would be unacceptable. However, the address mapping policy may not change dynamically without recopying every used DRAM location to match the new scheme. A solution would be to linearly partition the address space, implementing different policies for different segments of the address space. The energy-aware virtual memory system could determine the energy and performance requirements of a certain virtual address page and map it into a physical DRAM page which uses the best address mapping policy.

Finally, the physical implementation of the SCALE DRAM subsystem could not be completed at the time of writing of this thesis. Once the printed circuit board has been completed, it can be used to gather more accurate energy measurements for the memory system.

Bibliography

[1] Datasheet, Micron MT47H32M8BP-5E http://download.micron.com/pdf/datasheets/dram/ddr2/256Mb_x4x8x16_DDR2_B.pdf

[2] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam and M. J. Irwin DRAM Energy Management Using Software and Hardware Directed Power Mode Control In *HPCA 2001*, January 20-24, 2001, Nuevo Leone, Mexico.

[3] V. Delaluz, A. Sivasubramaniam, M. Kandemir, N. Vijaykrishnan and M. H. Irwin Scheduler-Based DRAM Energy Management In *DAC 2002*, June 10-14, 2002, New Orleans, Louisiana, USA.

[4] The Embedded Microprocessor Benchmark Consortium http://www.eembc.hotdesk.com

[5] X. Fan, C. Ellis and A. Lebeck Memory Controller Policies for DRAM Power Management In *ISLPED'01*, August 6-7 2001.

[6] R. Krashinsky, C. Batten, M. Hampton, S. Gerding, B. Pharris, J. Casper and K. Asanovi'c The Vector-Thread Architecture In *ISCA 2004*

[7] A. Lebeck et al. Power Aware Page Allocation In *ASPLOS-IX*, November 2000.

[8] S. Rixner, W. Dally, U. Kapasi, P. Mattson and J. Owens Memory Access Scheduling In *ISCA-27* 2000.