Winter 2-28-2013

A Quantitative Analysis of Memory Controller Page Policies

Matthew Blackmore
Portland State University

10.15760/etd.660

A Quantitative Analysis of Memory Controller Page Policies

by

Matthew Blackmore

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Portland State University
2013

Abstract

Two common goals in computing system design are increasing performance and decreasing power consumption. DRAM–based memory subsystems are a major component of both system performance and power consumption.

Memory controllers employ strategies to efficiently schedule DRAM operations to reduce latency and to utilize DRAM low power modes when possible. One of the most important of these is the page policy, which determines when to close pages in DRAM. An effective DRAM memory controller page policy is important to minimizing power consumption and increasing system performance.

This thesis explores the impact memory controller page policy has on performance as measured by the number of page-hits minus page-misses and estimated average memory access latency. I captured real-time DDR3 command and address memory traces for the SPEC CPU2006 benchmarks under three memory controller page policies: closed page, fixed open-page, and Intel's adaptive open-page [1]. Traces were captured using a programmable memory traffic analyzer (PMTA), a device interposed between the DIMM slot and DDR3 DIMM on the motherboard. The memory traces for each benchmark were analyzed to determine the absolute number of page-hits and page-misses that occurred.

In software post-processing I simulated a theoretically perfect "oracle" page policy for each captured trace to compare the efficiency of existing policies. The SPEC CPU 2006 benchmarks under the oracle page policy for each trace exhibited an average increase in the number of page-hits minus page-misses of 280.3% and an average decrease in the average memory latency of 11.1%.

Two new adaptive open-page policies are proposed and simulated using the captured memory traces. These proposed policies result in an average increase of 74.8% and 62.4% in the number of page-hits minus page-misses over Intel's adaptive open-page policy and an average decrease in the average memory latency of 3.8% and 3.4%.

Acknowledgements

I would like to foremost thank Mark Faust from Portland State University. He spent many hours advising and instructing me throughout the duration of my thesis as well as served on my thesis committee. His influence on my research and this thesis was invaluable. I truly appreciate his dedication and patience.

I would like to thank Howard David from Intel. Howard spent quite a bit of his time discussing research methodology and helping analyze data. His input provided direction for this thesis and was always extremely helpful.

I would like to thank Dr. Douglas Hall from Portland State University. Dr. Hall served as the chair for my thesis committee and provided valuable advice during the pursuit of my degree.

I would like to thank Dr. Christof Teuscher from Portland State University. Dr. Teuscher served as one of the members on my thesis committee and provided research and writing advice during the pursuit of my degree.

I would like to thank the original development team of the PMTA, particularly Samuel Burkhart and Rocky Chase. Their hard work laid the foundation for my thesis and made this research possible.

Table of Contents

List of Tables

List of Figures

1 Introduction

Two common goals in computing system design are increasing performance and decreasing power consumption. DRAM–based memory subsystems are a major component of both system performance and power consumption.

Memory controllers employ strategies to efficiently schedule DRAM operations to reduce latency and to utilize DRAM low power modes when possible. One of the most important of these is the page policy, which determines when to close pages in DRAM. An effective DRAM memory controller page policy is important to minimizing power consumption and increasing system performance.

This thesis explores the impact a memory controller page policy has on performance as measured by the number of page-hits minus page-misses[1] and estimated average memory access latency. I captured real-time memory traces for the SPEC CPU2006 benchmarks under three memory controller page policies: closed page, fixed open-page, and Intel's adaptive open-page. The benchmarks were captured on an Intel Greencity server platform using an Intel Xeon X5650 CPU [2] with integrated memory controller.

Real-time memory trace capture was chosen as the method of collecting data due to the completeness of traces that it provides and the ability to capture memory traffic that has already been filtered by CPU caches. The real-time memory traces were captured using a programmable memory traffic analyzer (PMTA) interposed between the DIMM and DIMM slot on the motherboard. The PMTA captures all command and address changes on the memory bus and saves them to a binary trace file over PCI Express on another machine. Prior to the creation of the PMTA, detailed memory traces allowing

[1] Page-empties are ignored for the purposes of this metric because they neither detract from nor add to system performance compared to page-hits and page-misses.

analysis of entire benchmarks were not practical. Without such memory traces, in-depth analysis of page-hits, page-empties, and page-misses were not possible.

Using post-processing software, the efficiency of each benchmark and page policy combination was analyzed by categorizing each read and write command as a page-hit, page-empty, or page-miss based on the state of the bank being accessed. In addition, an “oracle” page policy was simulated for each trace using the trace data. An oracle page policy is a theoretically perfect page policy, as if future memory transactions are known. An oracle page policy would produce a maximum number of page-hits and a minimum number of page-misses. I used the difference between these two numbers as a metric to compare current page policies with an oracle policy to show what potential there is for improvement.

In addition to comparing existing page policies and their theoretical room for improvement, two new policies are proposed and their simulated performance measured on the same benchmarks. We compare estimated average memory access latency under the three existing policies and the proposed policies based on the number of page-hits, page-empties, and page-misses [3] observed in the captured benchmark traces.

1.1 Outline

The remainder of this chapter provides background on the operation of DRAM-based memory systems. It describes the typical configuration of a DRAM-based memory module, the basic mechanism of accessing memory, and the performance categories DRAM memory accesses fall into. Chapter 2 discusses the role of the memory controller and describes current page policies. Chapter 3 examines memory trace capture methods and the advantages of real-time capture versus simulation. Chapter 4 introduces the programmable memory traffic analyzer, what it does, and how it was used to capture memory traces. Chapter 5 describes the methodology employed for this research. This includes the benchmarks being analyzed, the BIOS and memory configuration settings used, and the potential performance improvement observed. Chapter 5 also discusses the post processing techniques used to analyze page polices from captured traces and introduces the two proposed new page policies. Chapter 6 reveals the results of post processing the chosen benchmarks, how they compare, and potential room for improvement. Chapter 7 offers a conclusion of knowledge gained from this research and suggests areas for future work.

1.2 Dynamic Random Access Memory (DRAM)

Modern computers store data using a hierarchy of memory that generally falls into one of three categories: caches, primary memory (typically DRAM), and secondary storage (hard disk) [4]. A cache is small, fast, volatile storage close to the CPU. DRAM is also volatile, but can hold more data and due to the command interface and numerous timing constraints requires a memory controller. The overhead of a memory controller coupled with the added latency of DRAM chips contributes to slower memory access times compared to caches. Hard disks, while even slower, can hold much more data and are non-volatile. Most active process data is kept in caches and DRAM, increasing the need for these storage devices to be as fast as possible.

DRAM stores data bits in cells comprised of a single capacitor and a single transistor. Each memory cell is part of a row (also referred to as a page) and a column, giving it a unique address. A set of rows and columns comprises a bank, with a typical DDR3 DRAM chip having eight banks [5]. Each bank can have one active page at any given time, which allows for reduced memory latency by staggering memory accesses in up to eight pages at a time. If data is interleaved across banks, this can decrease the time it takes to access data in memory.

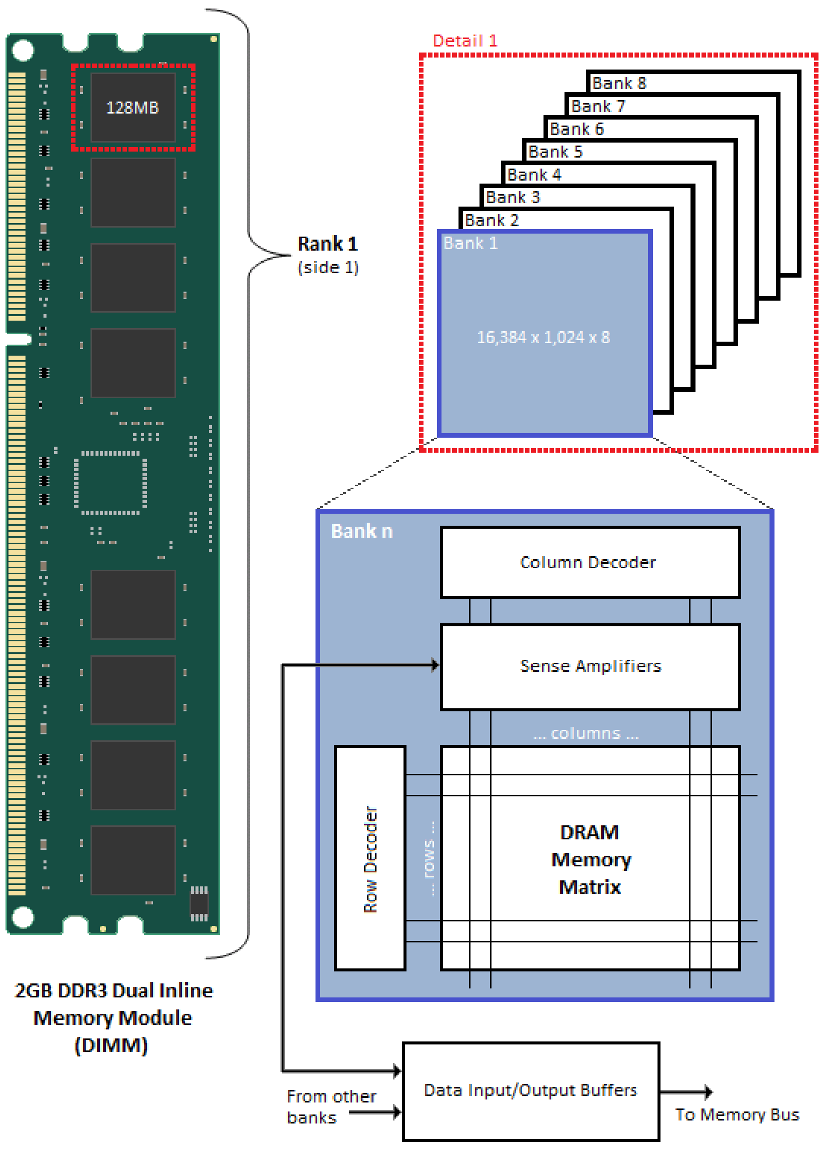


Figure 1: Typical Functional Arrangement of SDRAM Memory Space [1]

Multiple DRAM chips are used on dual in-line memory modules (DIMMs). The DIMM is typically divided into ranks, which form a group of DRAM chips. A single-rank DIMM has one set of DRAM chips connected to the data bus, while a dual-rank or quad-rank DIMM has two or four independent sets of DRAM chips connected to the data bus. Having multiple ranks increases the number of banks that can be accessed in a given amount of time due to interleaving of accesses. Each DIMM will have a unique DIMM slot located on a motherboard and will be part of a memory channel. Any DIMMs that share a channel also share the same memory bus, whereas separate channels have their own memory bus. Increasing the number of DIMM channels and ranks on a DIMM can help increase overall memory bandwidth.

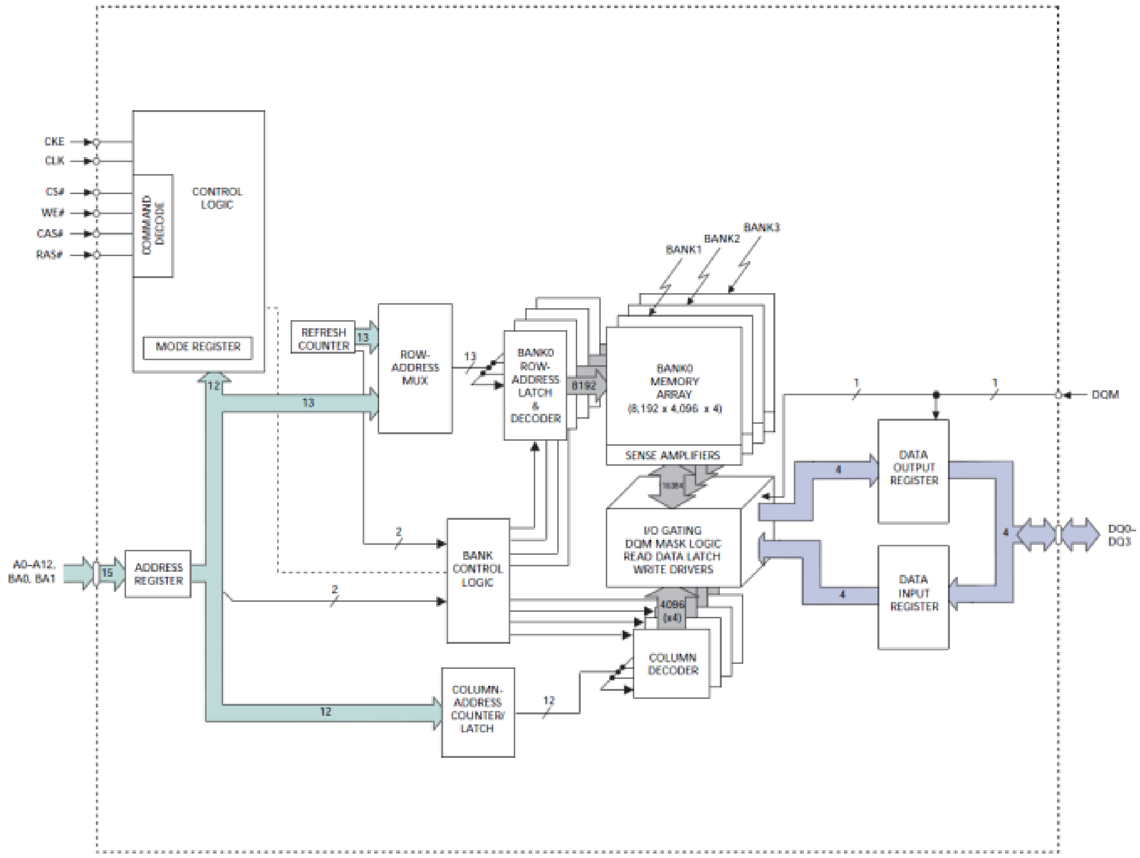
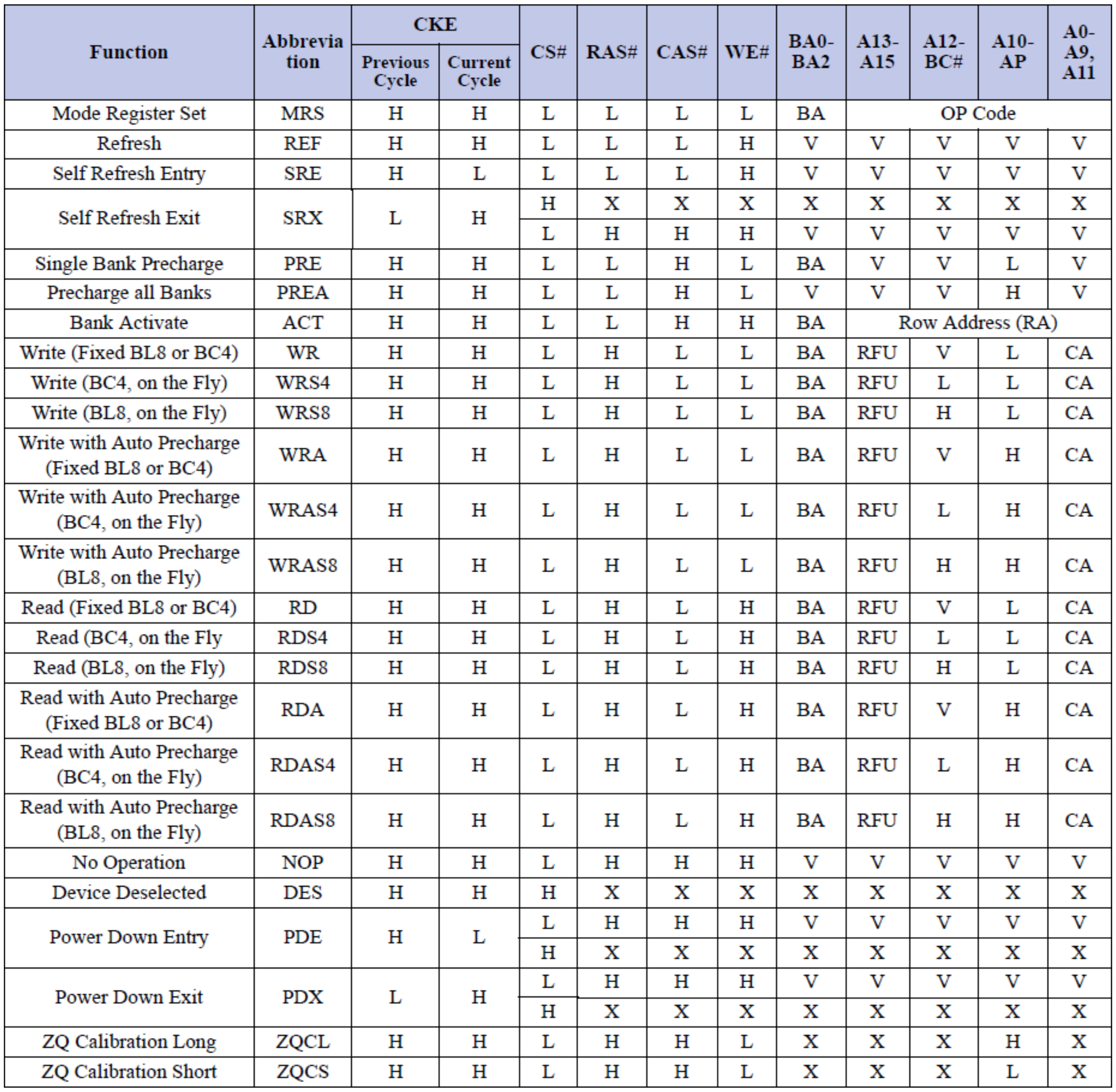


Figure 2: 128 Meg x 4 SDRAM Functional Block [6]

A DRAM command is comprised of a few different bit fields. All commands provide the desired rank to be accessed using the chip select (CS) bits. A command is identified by the combination of row address strobe (RAS), column address strobe (CAS), and write enable (WE) bits. Depending on the command, the bank address (BA) and generic address (A) bits may also be required. The BA bits specify which bank to access, and the A bits specify the row being accessed on an activate (ACT) command or the column being accessed on a read (RD) or write (WR) command.



Function	Abbreviation	CKE		CS#	RAS#	CAS#	WE#	BA0-BA2	A13-A15	A12-BC#	A10-AP	A0-A9, A11
		Previous Cycle	Current Cycle									
Mode Register Set	MRS	H	H	L	L	L	L	BA	OP Code			
Refresh	REF	H	H	L	L	L	H	V	V	V	V	V
Self Refresh Entry	SRE	H	L	L	L	L	H	V	V	V	V	V
Self Refresh Exit	SRX	L	H	H	X	X	X	X	X	X	X	X
				L	H	H	H	V	V	V	V	V
Single Bank Precharge	PRE	H	H	L	L	H	L	BA	V	V	L	V
Precharge all Banks	PREA	H	H	L	L	H	L	V	V	V	H	V
Bank Activate	ACT	H	H	L	L	H	H	BA	Row Address (RA)			
Write (Fixed BL8 or BC4)	WR	H	H	L	H	L	L	BA	RFU	V	L	CA
Write (BC4, on the Fly)	WRS4	H	H	L	H	L	L	BA	RFU	L	L	CA
Write (BL8, on the Fly)	WRS8	H	H	L	H	L	L	BA	RFU	H	L	CA
Write with Auto Precharge (Fixed BL8 or BC4)	WRA	H	H	L	H	L	L	BA	RFU	V	H	CA
Write with Auto Precharge (BC4, on the Fly)	WRAS4	H	H	L	H	L	L	BA	RFU	L	H	CA
Write with Auto Precharge (BL8, on the Fly)	WRAS8	H	H	L	H	L	L	BA	RFU	H	H	CA
Read (Fixed BL8 or BC4)	RD	H	H	L	H	L	H	BA	RFU	V	L	CA
Read (BC4, on the Fly	RDS4	H	H	L	H	L	H	BA	RFU	L	L	CA
Read (BL8, on the Fly)	RDS8	H	H	L	H	L	H	BA	RFU	H	L	CA
Read with Auto Precharge (Fixed BL8 or BC4)	RDA	H	H	L	H	L	H	BA	RFU	V	H	CA
Read with Auto Precharge (BC4, on the Fly)	RDAS4	H	H	L	H	L	H	BA	RFU	L	H	CA
Read with Auto Precharge (BL8, on the Fly)	RDAS8	H	H	L	H	L	H	BA	RFU	H	H	CA
No Operation	NOP	H	H	L	H	H	H	V	V	V	V	V
Device Deselected	DES	H	H	H	X	X	X	X	X	X	X	X
Power Down Entry	PDE	H	L	L	H	H	H	V	V	V	V	V
				H	X	X	X	X	X	X	X	X
Power Down Exit	PDX	L	H	L	H	H	H	V	V	V	V	V
				H	X	X	X	X	X	X	X	X
ZQ Calibration Long	ZQCL	H	H	L	H	H	L	X	X	X	H	X
ZQ Calibration Short	ZQCS	H	H	L	H	H	L	X	X	X	L	X

Figure 3: DDR3 Command Truth Table [5]

When the CPU needs access to data in DRAM it issues a request to the memory controller. The memory controller determines the correct sequence and timing of DRAM commands necessary to fulfill this request. In order for a memory controller to access data in DRAM, it must first ensure that the desired rank has its clock enabled (CKE). Next, it must check the state of the desired bank. If a row is already active, it must compare that row’s address to the desired row’s address. If they match then it can directly issue the RD or WR command. If they do not, that row must first be closed with a precharge (PRE) command. This causes that row’s data to be refreshed, the data bus to be disconnected, and the sense amplifiers to be disabled [4]. If no row was active or one was just closed, the memory controller must send an ACT command. This command contains the address of the row to be accessed and causes that row’s data to be latched into sense amplifiers, thus making it available. Next, the column address is sent in a RD or WR command to determine the starting address at which data will be sent to the data bus or written from the bus. While a row is activated, reads and writes can be issued to it with minimal latency. When access to a row is complete, the memory controller may issue a PRE to it, or it may leave it open for some amount of time. This behavior is determined by the page policy in use. See Figure 4 for a full example of a DDR3 command sequence and timing to access memory.

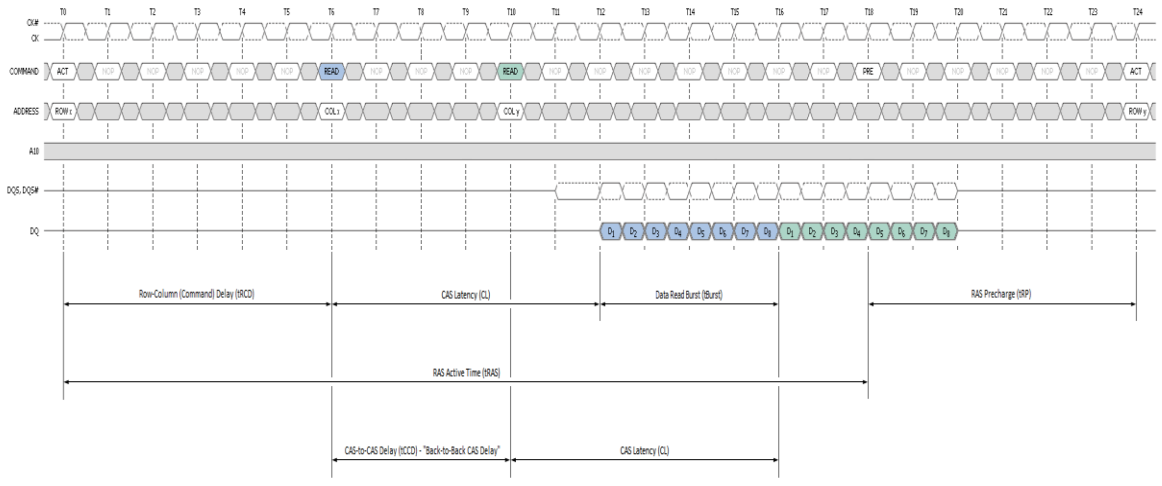


Figure 4: DDR3 Back-to-back Read Timing [1]

1.3 Page-hit vs. Page-empty vs. Page-miss

Each read or write access to DRAM can be separated into one of three categories depending on the current state of the bank being referenced. These categories, from lowest to highest latencies, are page-hit, page-empty, and page-miss.

A page-hit is defined as a read or write to a currently open page. The latency of this operation is the CAS latency of the device, or tCAS. Figure 5 shows the access timing of a page-hit memory reference.

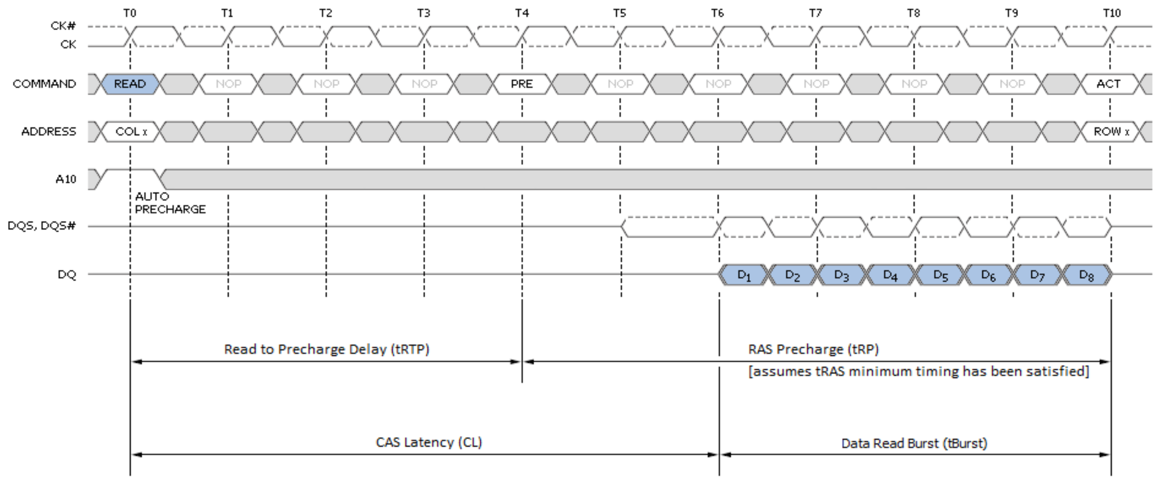


Figure 5: Page-hit Timing [1]

A page-empty occurs when a bank to be accessed has no open page. This requires an ACT command to open the desired page incurring additional latency of tRCD. Figure 6 shows the access timing of a page-empty memory reference.

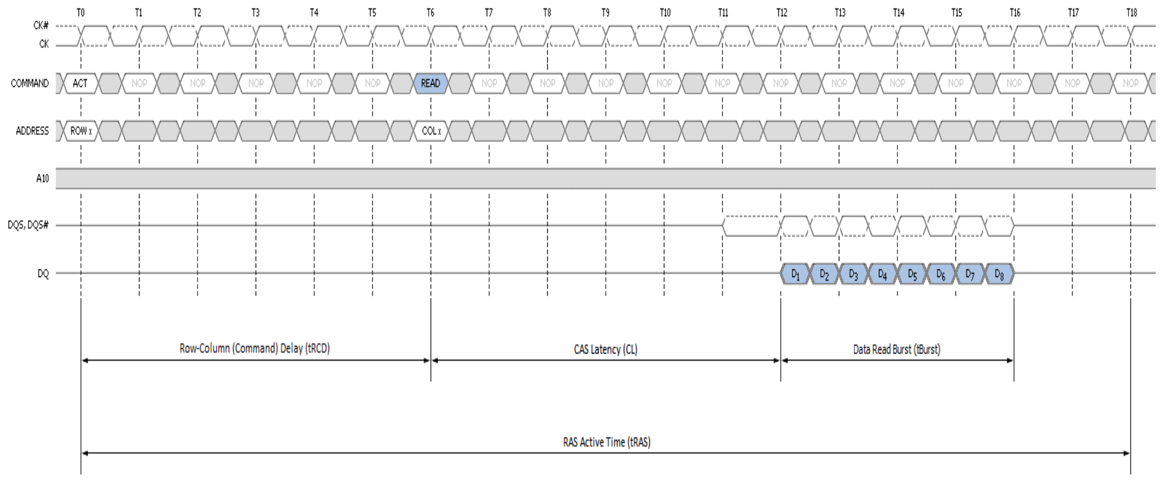


Figure 6: Page-empty Timing [1]

If a page is currently open and an access to another page is needed this is a page-miss. Before the desired page can be opened with an ACT command it must first be closed with a PRE command, incurring a penalty of tRP. Figure 7 shows the access timing for a page-miss memory reference.

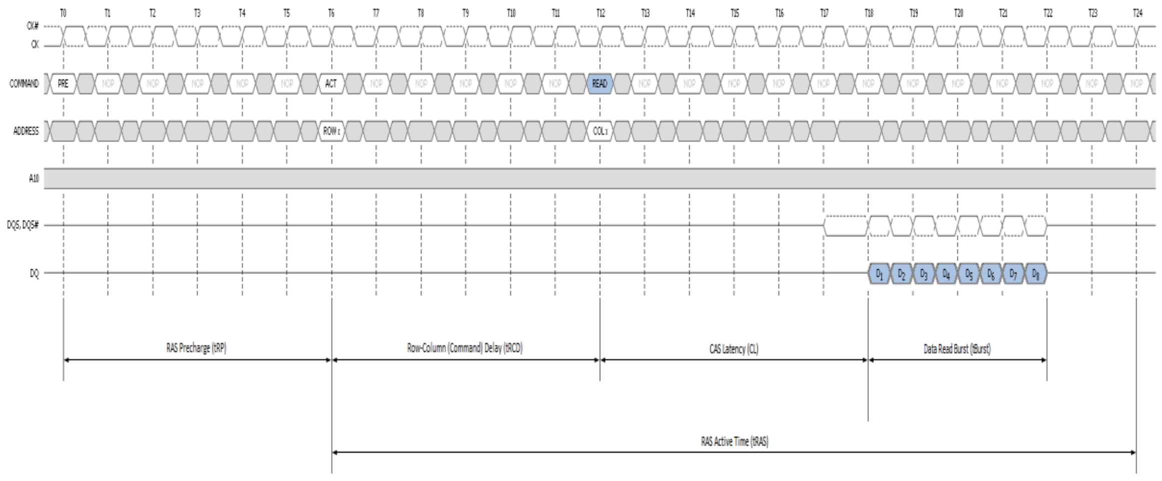


Figure 7: Page-miss Timing [1]

In summary, a page-hit occurs when the desired row in the desired bank is already active. This incurs a latency of only tCAS. A page-empty occurs when no row is active in the desired bank, and has a latency of tCAS + tRCD. A page-miss occurs when the wrong row is active in the desired bank and has a latency of tCAS + tRCD + tRP. The penalties for page-empties, and more importantly page-misses, increase both latency and power consumption, which decrease system performance.

2 Page Policies

A memory controller's page policy determines whether (and how long) to leave a DRAM page open in the expectation that the next reference to a bank will be to the same page, resulting in lower latency. The tradeoff a page policy must make is between latency and power consumption. Leaving a page open requires power and makes page-hits a possibility. However open-page policies have the potential for page-misses which can be costly in terms of access latency. Memory controllers typically employ one of three page policies:

- closed-page
- fixed open-page
- adaptive open-page

2.1 Closed-page Policy

A closed-page policy ensures that a DRAM page is closed immediately after every read or write. This effectively eliminates both page-misses and page-hits, making every access a page-empty. This may not allow for maximum memory bandwidth, but makes the access latency predictable. The closed-page policy can be effective in situations where many different DRAM pages are accessed frequently. Data can be interleaved in pages across banks, allowing multiple banks to be accessed almost simultaneously.

2.2 Fixed Open-page Policy

A fixed open-page policy allows the memory controller to leave a page open for a fixed amount of time after a read or write. Counters are used to track the clock cycles that have elapsed since the last read or write to each bank. If no reads or writes are issued to a bank before that bank's cycle counter reaches a specified threshold that page is said to timeout and is closed with a PRE command. The next access to that bank will be a page-empty. Allowing pages to be left open throughout the timeout interval permits the possibility of page-hits due to the spatial locality of data in memory. However, a DRAM bank becomes susceptible to page-misses if a page is left open longer than necessary.

2.3 Adaptive Open-page Policy

In an attempt to reduce average memory latency, the adaptive open-page policy was introduced [1]. This policy is similar to the fixed open-page policy but it dynamically adjusts the page timeout interval. In this thesis I analyze the performance of the adaptive open-page policy employed by Intel's Xeon X5650 CPU.

2.3.1 Intel Adaptive Open-page Policy Implementation

The memory controller used in the Intel Xeon X5650 CPU can be configured to employ a closed-page, fixed open-page, or adaptive open-page policy, typically selected by the BIOS at boot time. The Intel adaptive open-page policy uses hardware counters to track memory access activity across pages in DRAM. Some of these counters are checked periodically and compared to thresholds to determine whether a more aggressive or less aggressive page closing policy is required. If too many page-empties (that could have been page-hits) are occurring, then the memory controller will increase the time that pages are left open in the hope of converting subsequent page-empties into page-hits. On the other hand, if too many page-misses are occurring, the memory controller will decrease the page timeout value in the hopes of converting subsequent page-misses to page-empties.

This is implemented by using a page timeout counter (similar to fixed open-page policy), a mistake counter, and last row accessed registers [1]. The address of the last row accessed in each bank is saved in the last row accessed registers. If a page-empty could have been a page-hit, because the row being accessed is the same as the last row accessed, then the mistake counter is incremented. If a page-miss could have been a page-empty, because it occurred greater than tRP from the last PRE to that bank, then the mistake counter is decremented. At certain intervals, the mistake counter is compared against the high and low thresholds. If it is above the high threshold, then the page timeout value is increased so that pages will stay open longer. Conversely, if the counter is below the low threshold, then the page timeout value is decreased so that pages will close sooner.

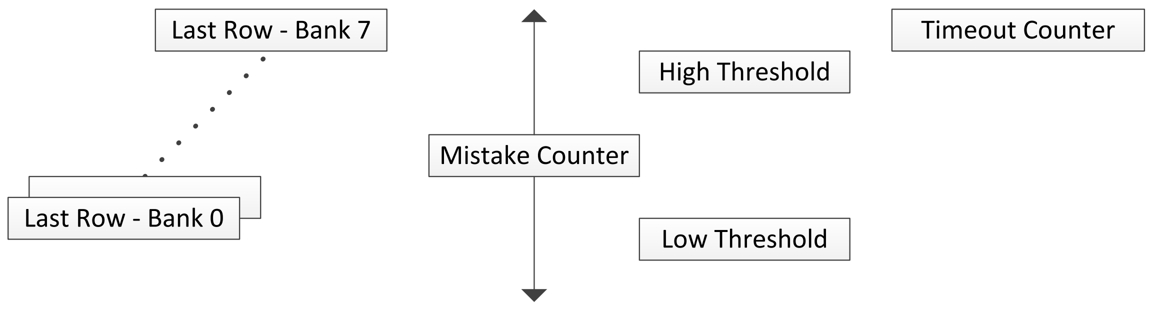


Figure 8: Intel Adaptive Open-page Registers

3 Memory Tracing

To study the behavior of a computer system, particular standardized programs or “benchmarks” can be executed on the system under test. These benchmarks have known CPU instruction behavior and observed memory access patterns. A researcher has a few options for using benchmarks to simulate the performance of potential architectural changes.

Execution driven simulation attempts to simulate the behavior of components in a system. This can include the CPU, caches, the memory controller, and more [7]. In order to do this, a benchmark is run in a simulated environment in software. This allows the simulation and validation of components that might not even exist yet. However, exact functionality of some components may only be known by their manufacturer, which can make this method impractical or can introduce potentially inaccurate models. Additionally, execution driven simulation can take long amounts of time to simulate complex components.

3.1 Trace Collection and Simulation

Trace driven simulation eliminates the need to simulate an entire system by capturing complete memory traces. This is usually done by either utilizing hardware counters, or capturing real-time data from the memory bus [8]. Hardware counters allow the tracking of CPU instructions or memory accesses, but can be restricted in the amount of data they can capture. Due to limited die space, there may only be room to retain a few hardware counters.

Real-time data capture provides the most complete traces [9] at the cost of requiring external hardware, post-processing software, and potentially creating huge data sets. This type of memory tracing is accomplished by attaching probes to the memory bus and logging all command and address changes externally. This method of tracing has no visibility of memory requests that are caught by CPU caches, and only shows transactions which go to DRAM. However, by only having visibility at the DRAM-level, it can be difficult to distinguish between memory accesses that are due to the benchmark in question or the operating system kernel, for example. This can also be an advantage because it shows the interaction of all processes and their impact on memory performance. External probe trace collection has the benefit of creating complete traces while incurring no time dilation or memory dilation, but runs the risk of introducing discontinuities [9].

3.1.1 Trace Distortion

Time dilation and memory dilation are forms of trace distortion that occur when a trace collection method causes a monitored benchmark to run slower or consume more memory than it normally would. This is usually not an issue with external probe-based data collection methods because they monitor the memory bus without interfering with the host.

On the other hand, discontinuities are possible with external probe-based data collection and are directly related to the method of trace storage. Discontinuities occur when there is a disparity in the speed of the trace storage mechanism and the rate at which data is being produced from the data probe. Trace-driven collection methods usually employ one of two methods to deal with discontinuities: buffering, or halting the target [9]. One way to handle disparities between the trace acquisition rate and trace storage rate is to buffer data before writing it to disk. If the buffers are in danger of being overrun, halting the target processor can reduce the possibility of losing trace data. However, this can cause time dilation because the benchmark will take longer to complete. In some cases halting the target system is not possible or practical and the data probe has to rely on buffering alone to prevent discontinuities. The size of the buffer in use is directly related to the amount of discontinuities that can be introduced into a trace. Smaller buffers are more likely to lead to trace data loss. The trace acquisition rate for a DDR3 DRAM command and address (CA) bus could be between 400MHz and 1066MHz [5]. A PCI Express 2.0 x4 link can transfer data at up to 2GB/s unidirectionally [10]; however a 7200rpm hard disk drive (HDD) can store data at around 145MB/s [11]. Depending on the trace buffer size, transferring data using PCI Express coupled with storing data to a mechanical hard disk could be slower than the acquisition rate at times, adding backpressure to the acquisition buffer, and potentially creating an overflow leading to trace data loss.

4 Programmable Memory Traffic Analyzer

A team of electrical and computer engineering students at Portland State University collaborated with Intel Corporation to create a programmable memory traffic analyzer (PMTA), an FPGA-based real-time memory capture device which sits in a DIMM slot interposed between the motherboard and the DIMM. This device snoops all command and address changes on the DDR3 memory bus to the DIMM attached to it. The PC in which this device is installed is denoted the target machine. The PMTA sends captured memory traces via PCI Express (PCIe) to another PC, called the capture machine, where software captures and saves these traces to binary disk files.

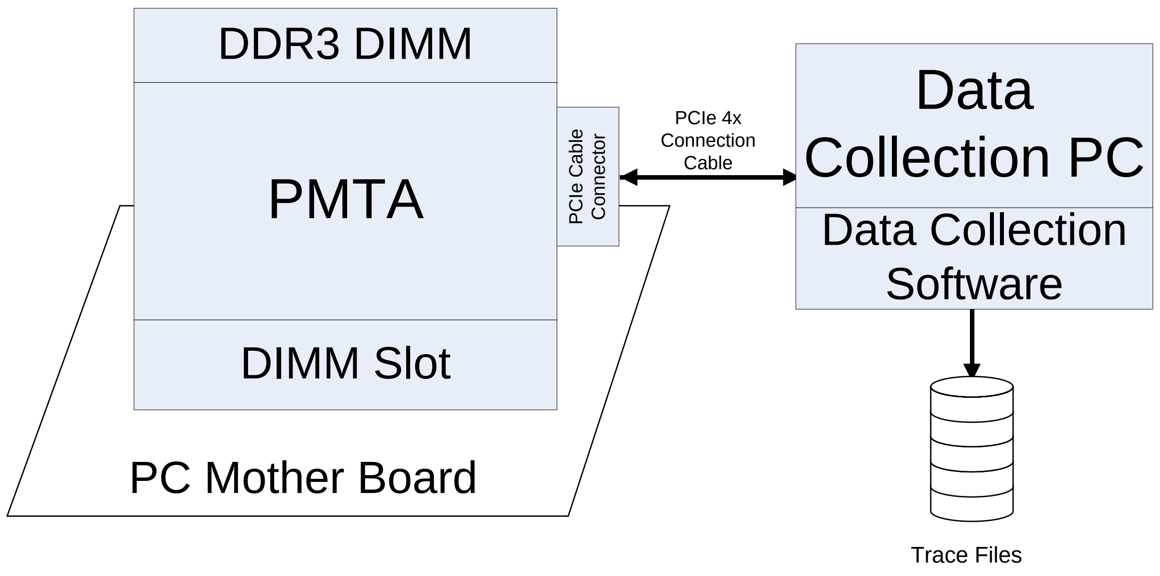
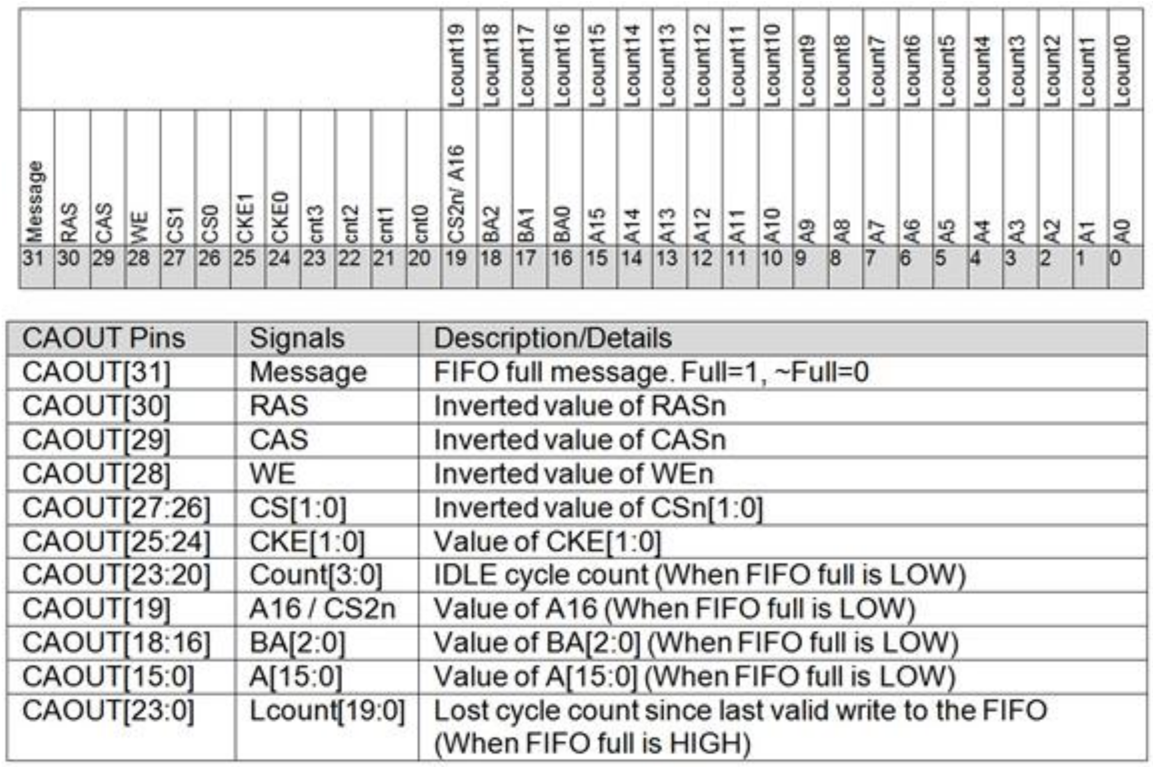


Figure 9: PMTA Connection Diagram [12]

A trace is comprised of packets where each packet contains all of the pertinent information which accompanied the DRAM command that the PMTA snooped. This includes the clock enable (CKE), chip-select (CS), row address strobe (RAS), column address strobe (CAS), write enable (WE), bank address (BA), and address (A) bits. Also included in a PMTA memory trace command packet is the number of clock cycles between the current command and the previous command. Using the PMTA, one can capture all memory transactions issued by the memory controller to DRAM.



												Lcount19	Lcount18	Lcount17	Lcount16	Lcount15	Lcount14	Lcount13	Lcount12	Lcount11	Lcount10	Lcount9	Lcount8	Lcount7	Lcount6	Lcount5	Lcount4	Lcount3	Lcount2	Lcount1	Lcount0
Message	RAS	CAS	WE	CS1	CS0	CKE1	CKE0	cnt3	cnt2	cnt1	cnt0	CS2n/ A16	BA2	BA1	BA0	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

CAOUT Pins	Signals	Description/Details
CAOUT[31]	Message	FIFO full message. Full=1, ~Full=0
CAOUT[30]	RAS	Inverted value of RASn
CAOUT[29]	CAS	Inverted value of CASn
CAOUT[28]	WE	Inverted value of WEn
CAOUT[27:26]	CS[1:0]	Inverted value of CSn[1:0]
CAOUT[25:24]	CKE[1:0]	Value of CKE[1:0]
CAOUT[23:20]	Count[3:0]	IDLE cycle count (When FIFO full is LOW)
CAOUT[19]	A16 / CS2n	Value of A16 (When FIFO full is LOW)
CAOUT[18:16]	BA[2:0]	Value of BA[2:0] (When FIFO full is LOW)
CAOUT[15:0]	A[15:0]	Value of A[15:0] (When FIFO full is LOW)
CAOUT[23:0]	Lcount[19:0]	Lost cycle count since last valid write to the FIFO (When FIFO full is HIGH)

Figure 10: PMTA C/A Packet Format [12]

The current revision of the PMTA has one significant limitation. It attempts to prevent trace discontinuities by using a FIFO to buffer command and address changes before sending them over PCIe to the capture machine. See Figure 11 for a block diagram of the PMTA. If the C/A capture block is capturing commands faster than the PCIe manager can transfer data or the capture machine can write to disk then the FIFO buffer will overflow. This causes a loss of command and address data. There is a mechanism in the FPGA to detect this situation and count the number of DDR3 cycles during which commands are lost. Once the PMTA recovers from an overflow by beginning to empty the FIFO, a special packet is sent to the capture machine to indicate that an overflow occurred and the number of cycles of trace data that was lost.

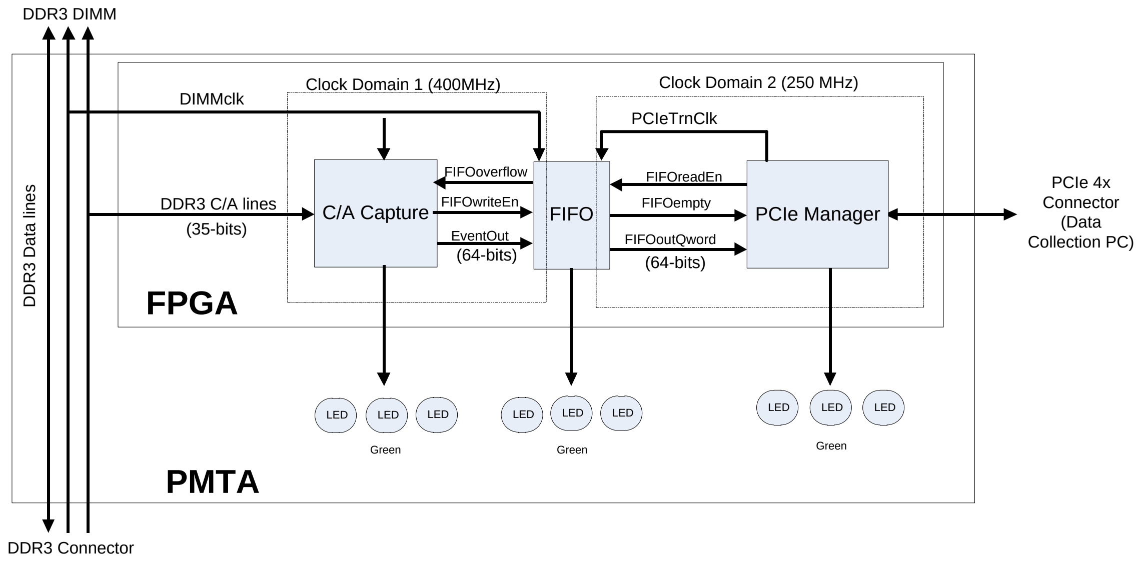


Figure 11: PMTA Block Diagram [12]

If a benchmark requires high memory bandwidth it is more likely to cause FIFO overflows. If an overflow occurs during a memory trace capture it can corrupt the capture. The number and size of overflows can be determined using post-processing software and it is up to the user to determine if a trace is fully corrupt or if the amount of data lost is within acceptable bounds.

5 Methodology

This research explores current memory controller page policy algorithms and their effectiveness for a range of benchmarks. It is up to the memory controller to efficiently schedule DRAM accesses to minimize the number of page-misses and maximize the number of page-hits. An effective page policy is the key to achieving these goals. If the page policy doesn't fit well with the behavior of a program's memory accesses, page-empties and page-misses can add additional latency and increase power consumption.

With the ability to capture and analyze all memory transactions, one can quantitatively describe how effective each page policy is in reducing latency and more importantly what improvements can be made. The closed-page and fixed open-page policies do not have many degrees of freedom in their algorithms. However, the adaptive open-page mode could possibly be tuned to better fit a range of benchmarks. This policy will be analyzed for the SPEC CPU2006 suite of benchmarks to see what, if any, improvements could be made in its algorithm. To compare performance the difference between page-hits and page-misses and the average memory access latency will be used as metrics. The benchmarks which demonstrated greater than 2GB/s in total memory bandwidth were chosen for in-depth post-processing analysis.

The main steps involved in the analysis of each benchmark include:

1. Capture memory trace for a benchmark using the PMTA.
2. Post-process trace to determine:
    a. Runtime
    b. Memory bandwidth
    c. Number of page-hits, page-empties, page-misses

   d. Number of oracle page-hits, oracle page-empties, oracle page-misses
   e. Estimated average memory access latency
3. Post-process trace to estimate the number of page-hits, page-empties, page-misses, and estimated average memory access latency for the two proposed page policies.
4. Compare the real performance metrics from the captured trace with the results of an oracle page policy to determine how much improvement can be made.
5. Compare the real performance metrics from the captured trace with the metrics of the new simulated page policies to see if any improvement was made.
6. Repeat steps 1 thru 5 for each existing page policy.

This process was repeated for each benchmark to analyze a range of program types and to determine if a new page policy could be used to improve performance across multiple workloads.

5.1 Target System

The target system is the Intel Greencity server platform. It supports up to two sockets, three DIMM channels per socket, and two DIMM slots per channel of DDR3 memory. Due to the method of trace capture, only one Samsung 8GB dual-rank DDR3 DIMM was populated in the system. This allowed for the capturing of all memory traffic. The socket corresponding to that DIMM was populated with an Intel Xeon X5650 CPU. This processor operates at 2.66GHz, has 6 cores, with 12 total threads, and an Intel integrated memory controller. The operating system was SUSE Linux Enterprise Server 11. BIOS settings were left to their defaults, except that the memory frequency was forced to DDR3 800MHz, and the memory controller page policy was selected for each capture.

5.2 Benchmarks

The SPEC CPU2006 suite of benchmarks were captured individually and used for analysis. The CPU2006 suite is designed to benchmark a system's processor, memory subsystem and compiler [13]. Some of the benchmarks are memory-intensive, while others are almost exclusively CPU-intensive. Some benchmarks are single-threaded, while others are multi-threaded. The memory-intensive benchmarks will be determined based on the memory bandwidth over the lifetime of the program.

5.3 Post Processing

Once a trace has been captured it can be processed for various types of memory analysis. The analysis explored here is the behavior of the memory controller in regards to closing pages. A C++ program was written to decode and process PMTA memory traces. This program tracks the state of each bank in a DIMM throughout a trace. When an ACT command is detected, the state of that bank is set to active. When a RD or WR command is detected, the corresponding bank’s status indicates this. When a PRE command or precharge-all (PREA) command is seen, the appropriate bank or banks are marked as closed.

The program calculates the absolute timing between ACTs, RDs, WRs, and PREs/PREAs to each bank based on the known state of the bank. In doing so, it can determine the number of page-hits, page empties, and page-misses that occurred. A page-hit is detected when a RD or WR to a bank is followed by another RD or WR to the same bank. A page-miss is detected when an ACT follows a PRE to the bank by tRP cycles later. This is the minimum possible time that an activate could occur after a precharge and indicates that another row needed to the accessed. A special case may also occur where a page in a bank is precharged due to a miss but the corresponding ACT to that bank is delayed longer than tRP after the PRE because of an access to another bank. This case is considered in the software. A page-empty is detected when an ACT is seen to a bank where the previous command to that bank was a REF, PREA, or PRE that occurred more than tRP before the ACT (unless the ACT was delayed because of an access to another bank). This means the bank was idle prior to the ACT.

5.4 Oracle Page Policy

The trace post-processing software not only counts the total number of page-hits, page-empties, and page-misses, it also estimates what those numbers could be under an "oracle" page policy. An oracle policy is a perfect page policy, as if the next memory reference address were known in advance. For example, if a page-empty could have been a page-hit (because it was a RD or WR to the same page as the last RD or WR, but that page was closed prematurely), then that access would be counted as an oracle page-hit. See Figure 12 for an example of page-empty memory access that could become an oracle page-hit. While not achievable in practice, this analysis is useful for determining the theoretical optimal page policy.

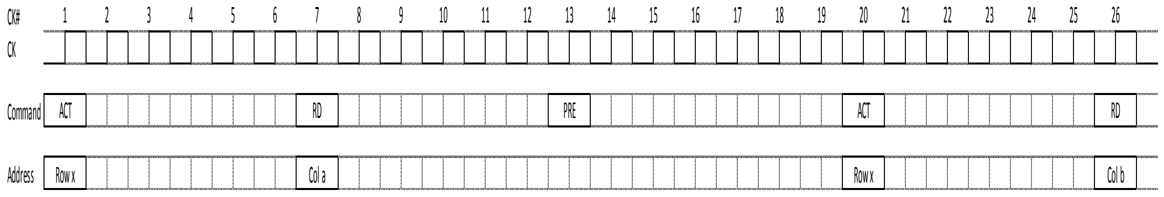


Figure 12: Actual Page-empty, Oracle Page-hit Example

Figure 12 illustrates an instance where a page-empty could have been a page-hit under an oracle page policy. The RD to column b was a page-empty because it immediately followed an ACT command that came greater than tRP cycles after the previous PRE command. It could have been an oracle page-hit if row x had been left open instead of being closed with a PRE command at cycle 13. See Figure 13 for an example of a page-miss memory access that could become an oracle page-empty. The RD to column b was a page-miss because it immediately followed an ACT command that came exactly tRP cycles after the PRE command. Because the PRE command at cycle 13 occurred greater than tCAS from the last RD, row x could have been closed earlier, causing the RD to column b to be an oracle page-empty.

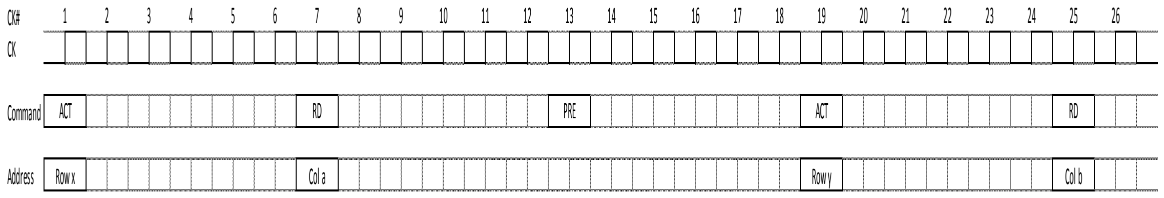


Figure 13: Actual Page-miss, Oracle Page-empty Example

By comparing oracle page-hits, page-empties, and page-misses to actual page-hits, page-empties, and page-misses, the post-processing tool can estimate the theoretical performance improvement that a perfect page policy could provide. This gives us an upper-bound on the performance improvement that a perfect page policy could provide.

5.5 Inter-arrival Times

The time between consecutive RDs/WRs to a bank can be called their inter-arrival time. This time is calculated and analyzed on a per-interval basis and can help determine what would have been the best time to close pages in that interval. The purpose of this is to compare the inter-arrival distribution of page-hits vs. the inter-arrival distribution of page-empties + page-misses (a.k.a. non-page-hits). I track inter-arrival distributions for traces in post-processing software by keeping counters for each bin of a histogram. Every RD/WR command increments a counter in one of the bins in the hit or non-hit distributions.

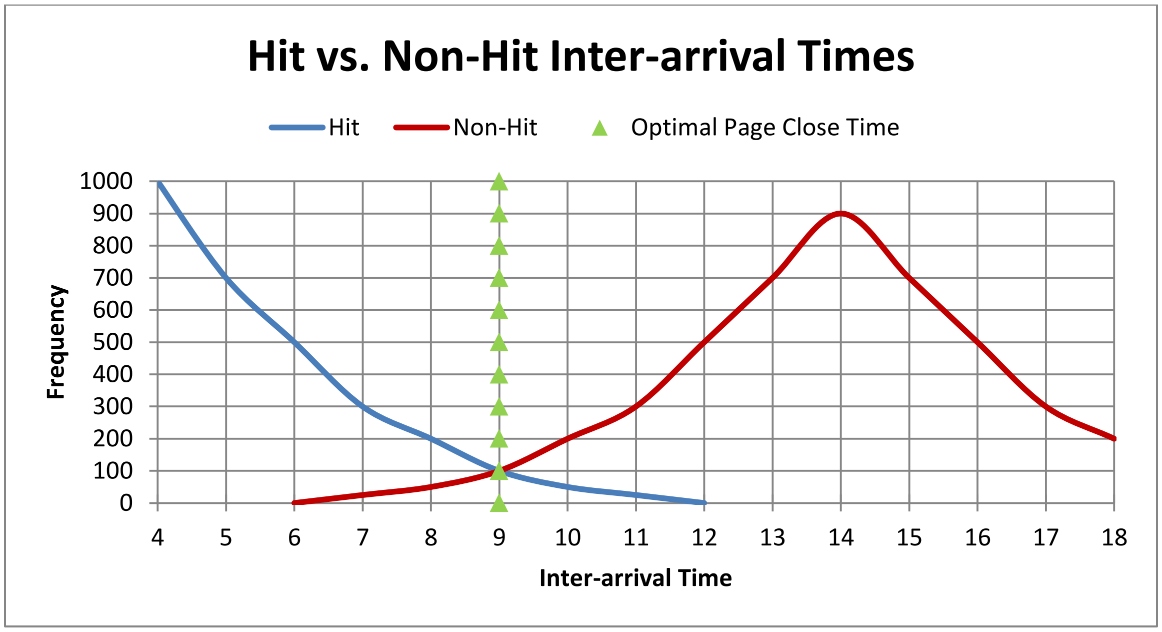


Figure 14: Inter-arrival Example

Figure 14 shows example distributions of page-hit inter-arrival times versus not page-hit inter-arrival times. Figure 14 does not represent data from a real benchmark; it was generated based on observations from real data to illustrate inter-arrival times. In this example, both series are roughly standard distributions above four clocks (consecutive RDs and WRs cannot be less than four clocks apart). The optimal time to close pages here would be after nine clock cycles with no RD/WR being issued. This allows enough time to capture all of the RDs/WRs issued less than nine clocks apart as page-hits, and would convert most of the RDs/WRs issued greater than nine clocks from the last RD/WR from page-misses into page-empties. We still incur penalties for the page-misses issued less than nine clocks from the last RD/WR and page-hits issued greater than nine clocks from the last RD/WR, but the number of page-hits minus page-misses would increase.

The inter-arrival distributions created and used for this thesis are oracle distributions. This represents optimal memory access timing and is most useful in determining the best time to close a page in memory. For a page-hit, the time of the last RD or WR to the current bank is subtracted from the current RD or WR and is used as the inter-arrival time. For a page-empty, the time from the last RD or WR to the current bank is subtracted from the most recent ACT and is used as the inter-arrival time. For a page-miss, the time from the last RD or WR to the current bank is subtracted from the most recent PRE and is used as the inter-arrival time. These are the times that the memory controller would have ideally issued the current memory access if the necessary bank and row were active.

5.6 Inter-arrival Open-page Policy

In an attempt to improve upon the existing Intel adaptive open-page policy, I decided to simulate a policy which tracks inter-arrival distributions over a given interval and uses that knowledge to set the page close timeout value for the next interval. The new policy accumulates the oracle page-hit and non-page-hit (page-empty and page-miss) inter-arrival distributions for 1000 reads and writes, determines the new page timeout value from the distributions, and then clears the counters for all bins in both distributions. One thousand reads and writes were used as the threshold because this represented enough transactions to show access patterns, but was quick enough to be able to adapt to a changing benchmark.

Two different algorithms were tested to determine the optimal page timeout value from the inter-arrival distributions. The first algorithm choses a new page timeout value from the statistical mean of the two inter-arrival distribution peaks. The most frequently occurring page-hit inter-arrival time is averaged with the most frequently occurring non-page-hit inter-arrival time to determine the new timeout value. This method was chosen after I viewed inter-arrival distributions for many of the SPEC CPU2006 workloads and saw that they followed the general pattern of single-peak distributions.

The second algorithm choses the new timeout value by finding the first intersection of the inter-arrival distributions. This is the inter-arrival time where the number of page-hits first becomes less than the number of non-page-hits. Both algorithms aim to find the page close time which will maximize the number of page-hits and minimize the number of page-misses. In the example in Figure 14 the new timeout value would be chosen as nine clock cycles for both methods.

5.6.1 Hardware Requirements

The hardware requirements of implementing a new page policy can be a major factor in whether or not it is a viable option. Because integrated memory controllers share precious die area with the CPU the less logic that is required to manage the page policy the better. There would be some hardware overhead associated with both versions of the inter-arrival open-page policy. Each requires two histograms with up to 200 bins to track inter-arrival rates for page-hits and non-page-hits. Each bin represents one DDR3 clock. If the threshold for updating the page timeout value is 1000 reads and writes, then the counters for each bin would not exceed 10 bits to cover the unusual case where all accesses fall into one bin. Thus 4000 bits for storage of the inter-arrival distributions suffices.

The two policies differ in how they determine the optimal page timeout value. The simpler of the two requires finding the intersection of inter-arrival distributions. A single comparator can be used to iterate through the bins sequentially and compare the values in a single page-hit histogram bin with the matching non-page-hit histogram bin. When it finds the bin where the page-hit value is less than the non-page-hit value then it is done.

The second policy requires finding the mean of inter-arrival distribution peaks. This policy would also require a comparator to iterate through both histograms and find the bin with the highest count (the distribution peak). It additionally requires logic to perform addition and division on the peaks of the distributions to determine the mean. Division is not a trivial task in hardware, but this page policy requires only integer division. After the mean is calculated then the page policy logic has completed its task of determining the new page timeout value. Both proposed page policies would be comparable with existing page policies in terms of die area and complexity required.

5.7 Performance Metrics

There are two metrics which I chose to compare performance between existing and simulated page policies. The first is page-hits minus page-misses. Page-hits provide an equivalent amount of benefit to performance as page-misses do to hinder it. In this comparison, page-empties have no net effect on performance so they are ignored. An efficient page policy aims to maximize page-hits and minimize page-misses. A performance increase would yield a positive delta in this metric.

The second metric is the average estimated memory latency of each benchmark. Using a carefully constructed benchmark, David et al. [3] found average idle memory latencies under a closed-page policy at 800MHz to be 60ns for a page-hit, 75ns for a page-empty, and 90ns for a page-miss. Using these latencies as references, I can find the estimated average memory access latency for a DIMM trace by multiplying the absolute number of page-hits, page-empties, and page-misses found by their respective latencies. This metric does not show the exact average memory latency, but is useful in comparing the performance of page policies because it takes all three types of page accesses into account.

6 Results

The first order analysis of the SPEC CPU2006 benchmarks consists of comparing program runtimes and memory bandwidth when varying the page policy. Runtimes can show a high-level comparison of a page policy's effect on performance while bandwidth can show which of the benchmarks are memory-intensive and deserve closer analysis.

6.1 Runtime and Bandwidth

Figure 15 shows the runtime ratios of SPEC CPU2006 benchmarks captured on the target machine using each existing page policy. The runtime ratios are shown with the fixed open-page policy as the reference.

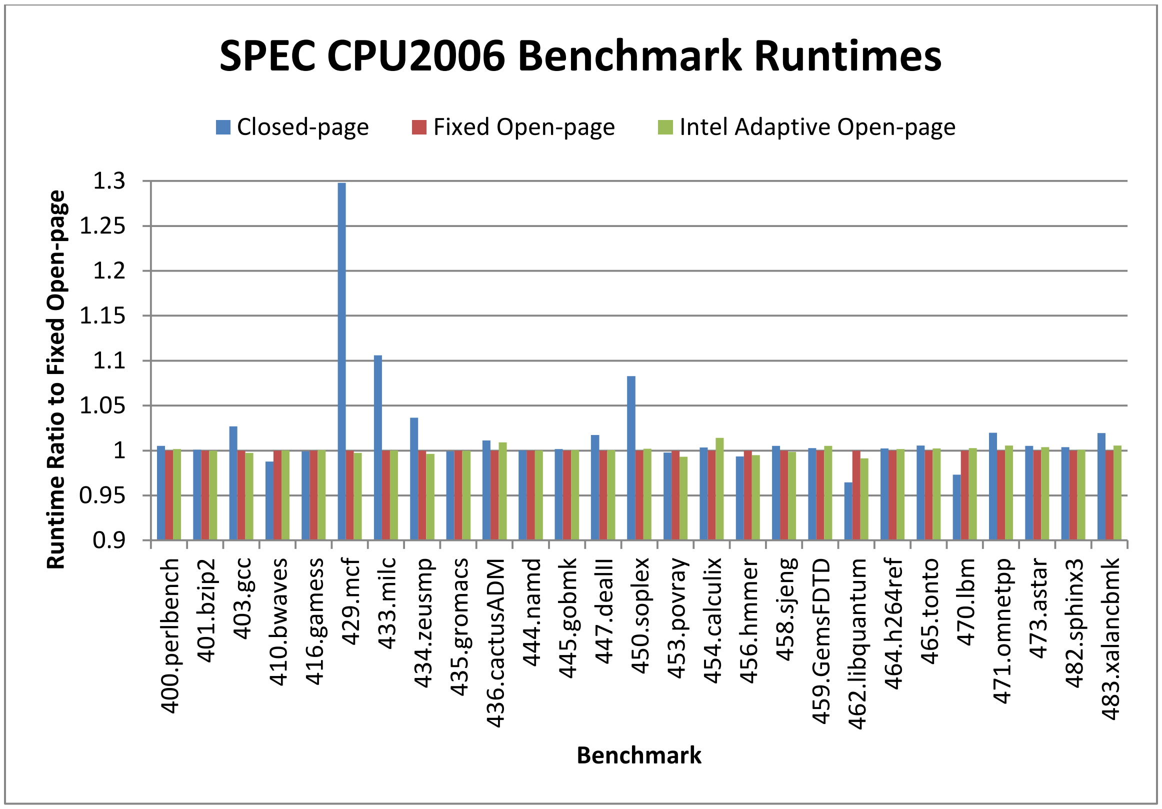


Figure 15: Benchmark Runtime Ratios

For many benchmarks, the closed-page policy resulted in significantly increased runtime. However in a few cases, this policy reduced the runtime. A benchmark which benefits from using a closed-page policy would be one that accesses many unique pages in memory with very little sequential accesses to a single page.

In almost all cases, there was little difference between the fixed open-page policy and adaptive open-page policy runtime. The variations in runtimes that did occur between the two open-page policies would statistically be considered noise because the benchmark could easily vary by +/- 1% runtime between runs. This is due to minor interference from the operating system or background processes on the CPU and illustrates why it is important to analyze the absolute number of page-hits, page-empties, and page-misses.

Memory bandwidth can show which benchmarks are memory-intensive, and therefore would have the most potential to be affected by a change in page policy. Below are the total memory bandwidth numbers for each benchmark. The bandwidths were calculated by multiplying the total number of reads and writes for the duration of a benchmark by 64 bytes, then dividing by the runtime of the benchmark in seconds. This provides the bytes per second memory bus bandwidth for a benchmark.

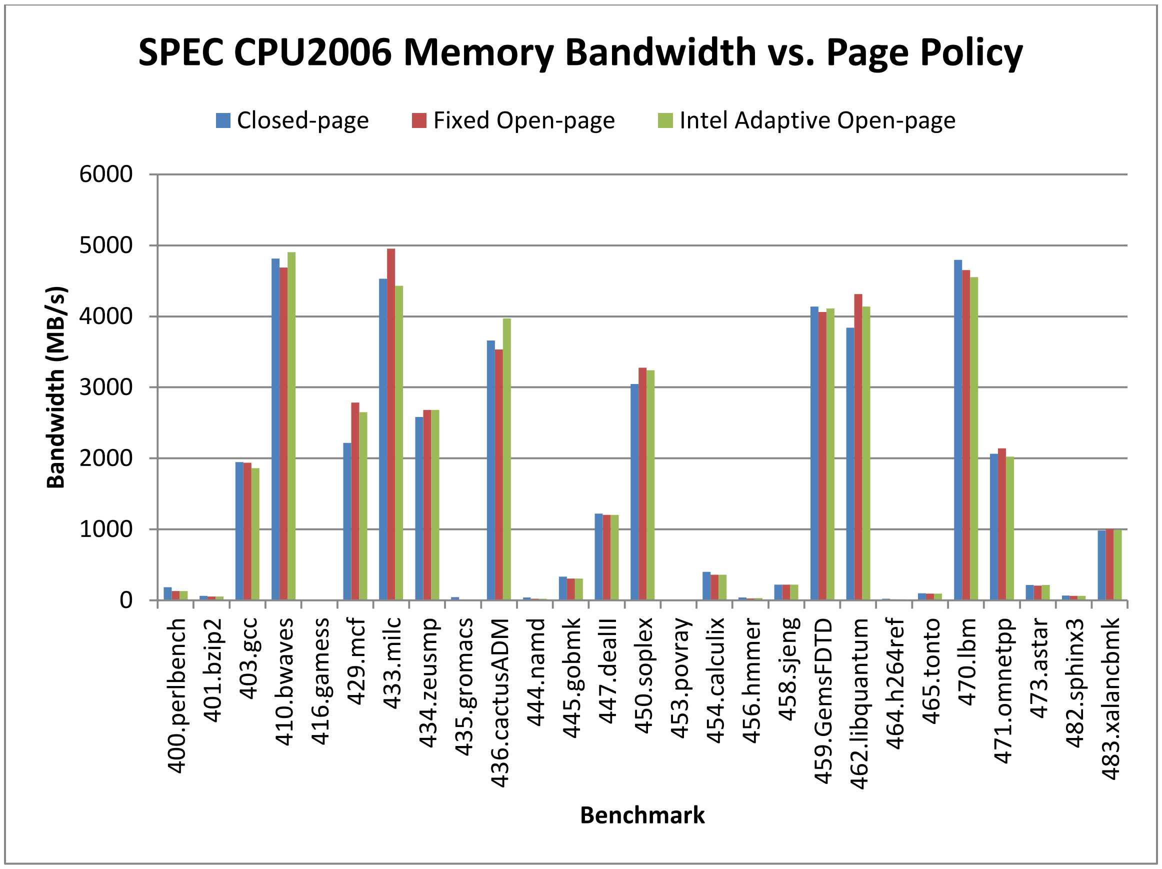


Figure 16: Benchmark Memory Bandwidths

There were 10 benchmarks which exceeded 2GB/s total memory bandwidth, which is approximately 1/3 of the maximum DDR3 channel bandwidth at 800MHz. There are bandwidth variations for the same benchmarks run under a different page policy because caching can affect the number of transactions that go to memory, there can be runtime variations caused by the operating system, and some benchmarks perform more efficiently with certain page policies. The 10 benchmarks that showed greater than 2GB/s total memory bandwidth are used throughout rest of this paper for in-depth analysis because they are memory intensive enough that a change in page policy could affect their performance.

6.2 Page-hit Minus Page-miss

Figure 17 shows the page-hit minus page-miss results for the 10 SPEC CPU2006 benchmarks which had greater than 2GB/s memory bandwidth. This figure compares existing page policies with their simulated oracle policies.

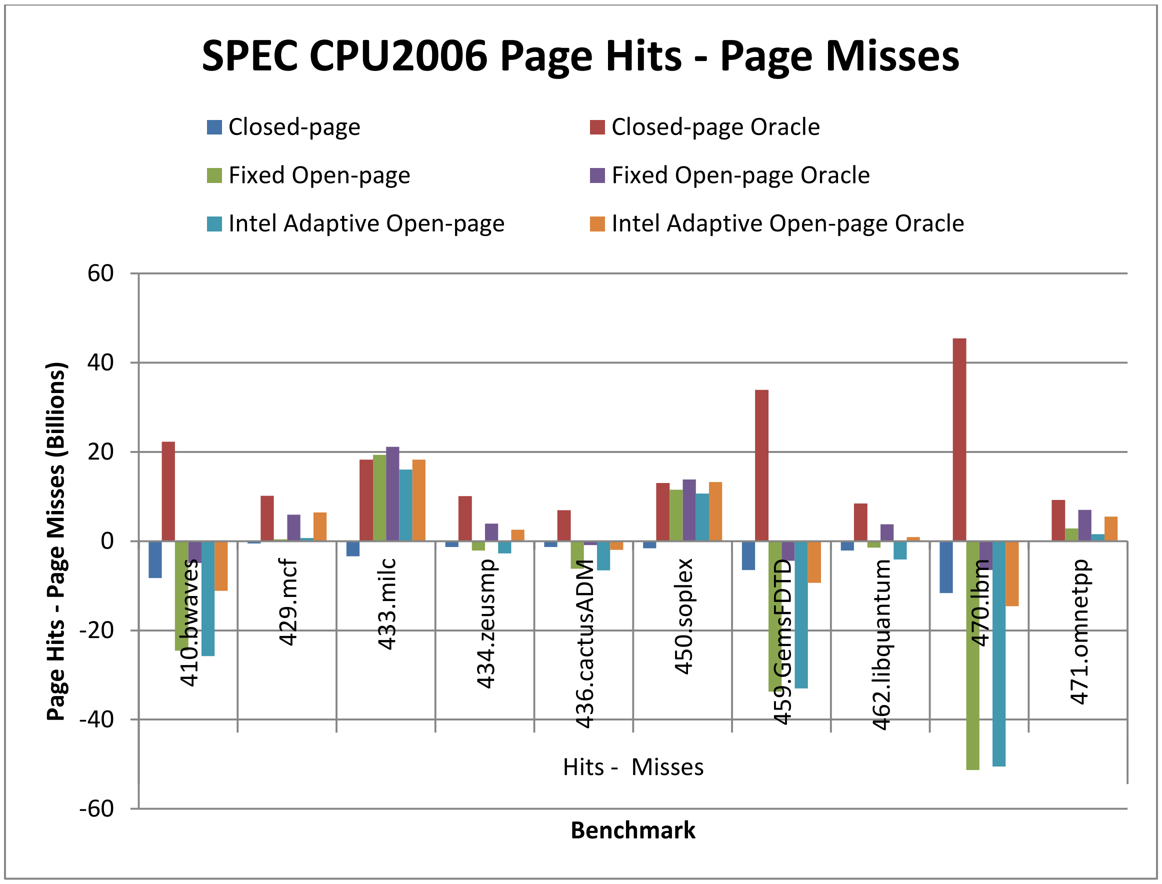


Figure 17: Page-hit Minus Page-miss – Existing Policies Compared to Their Oracle

Each of the existing policies was analyzed by capturing memory traces using the PMTA. The Intel memory controller in the machine studied here alters its address mapping to DDR3 memory based upon the page policy selected at boot time. This is why there are different oracle results for each benchmark; each oracle estimates an ideal page policy based upon the captured stream of DDR3 commands and the timing restrictions of DDR3. For some benchmarks, the closed-page policy has a lot of potential for improvement. Table 1 compares page-hit minus page-miss results for existing page policies and their oracle estimates. Larger positive numbers indicate better performance.

	Closed-page	**Fixed Open-page**	**Intel Adaptive Open-page**	**Closed-page Oracle**	**Fixed Open-page Oracle**	**Intel Adaptive Open-page Oracle**
Average Page-hit - Page-miss	-3.67	-8.52	-9.36	17.79	3.92	1.00
% Improvement over Closed-page	0.0%	-132.0%	-154.9%	584.3%	206.7%	127.2%
% Improvement over Fixed Open-page	56.9%	0.0%	-9.9%	308.8%	146.0%	111.7%
% Improvement over Intel Adaptive Open-page	60.8%	9.0%	0.0%	290.0%	141.8%	110.7%

Table 1: Page-hit Minus Page-miss % Improvement – Existing Policies Compared to their Oracle

Of the existing page policies, the closed-page policy showed the best average performance in terms of page-hits minus page-misses. It also showed the most room for improvement with its oracle policy providing a 584.3% improvement. The Intel adaptive open-page policy showed the worst performance for this metric.

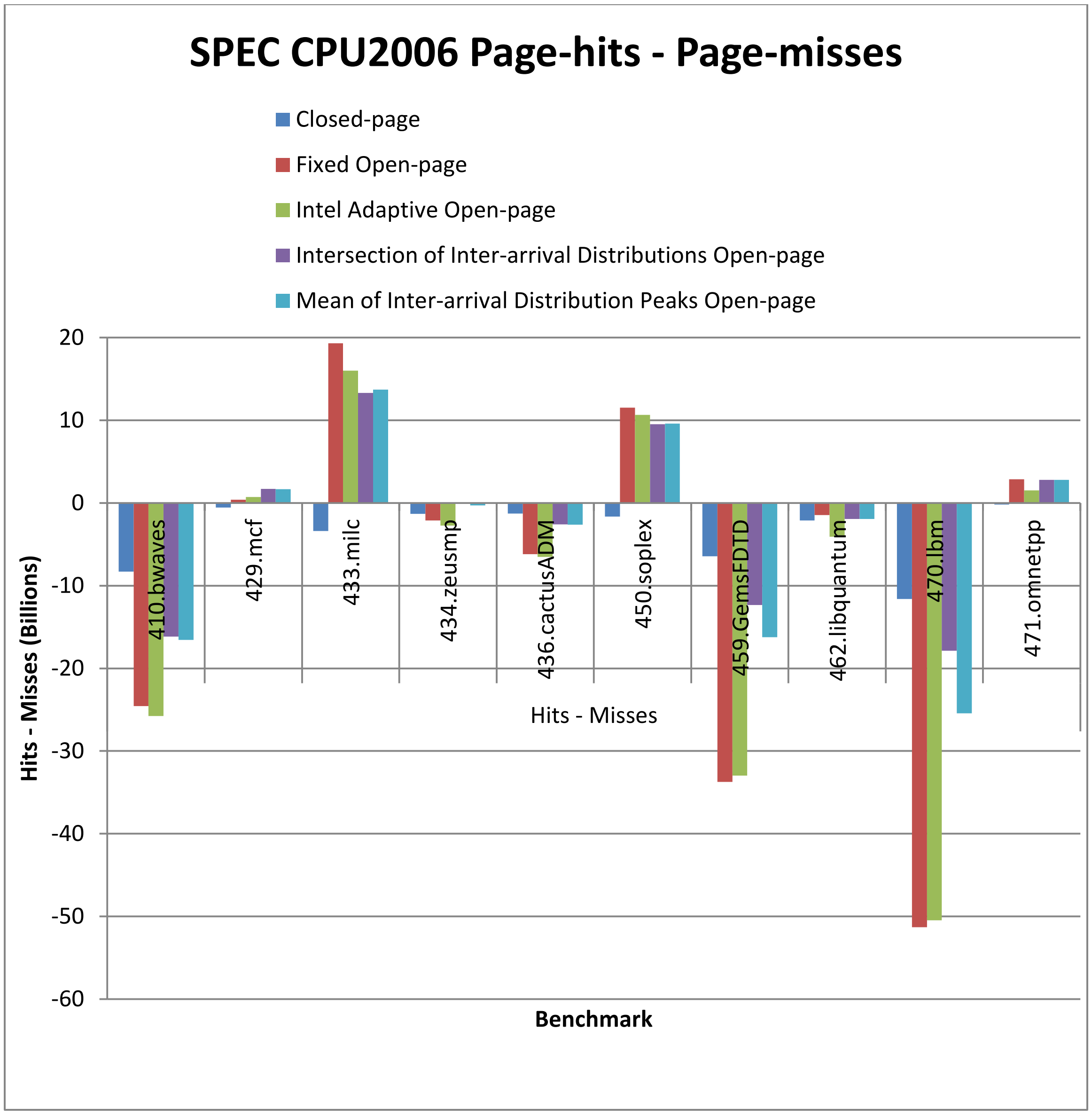


Figure 18: Page-hit Minus Page-miss – New Policy Compared to Existing

	Closed-page	Fixed Open-page	Intel Adaptive Open-page	Intersection of Inter-arrival Distributions Open-page	Mean of Inter-arrival Distribution Peaks Open-page
Average Page-hit - Page-miss	-3.67	-8.52	-9.36	-2.36	-3.52
% Improvement over Closed-page	0.0%	-132.0%	-154.9%	35.8%	4.2%
% Improvement over Fixed Open-page	56.9%	0.0%	-9.9%	72.3%	58.7%
% Improvement over Intel Adaptive Open-page	60.8%	9.0%	0.0%	74.8%	62.4%

Table 2: Page-hit Minus Page-miss % Improvement – New Policy Compared to Existing

Both inter-arrival open-page policies performed better than all existing page policies in terms of page-hits minus page-misses. The intersection of inter-arrival distributions policy showed a marked advantage over the mean of inter-arrival distribution peaks policy. The two benchmarks on which the new page policies performed worse than the existing open-page policies were 433.milc and 450.soplex. Both of these benchmarks exhibit inter-arrival distributions with a large discrepancy between the number of page-hits and not page-hits.

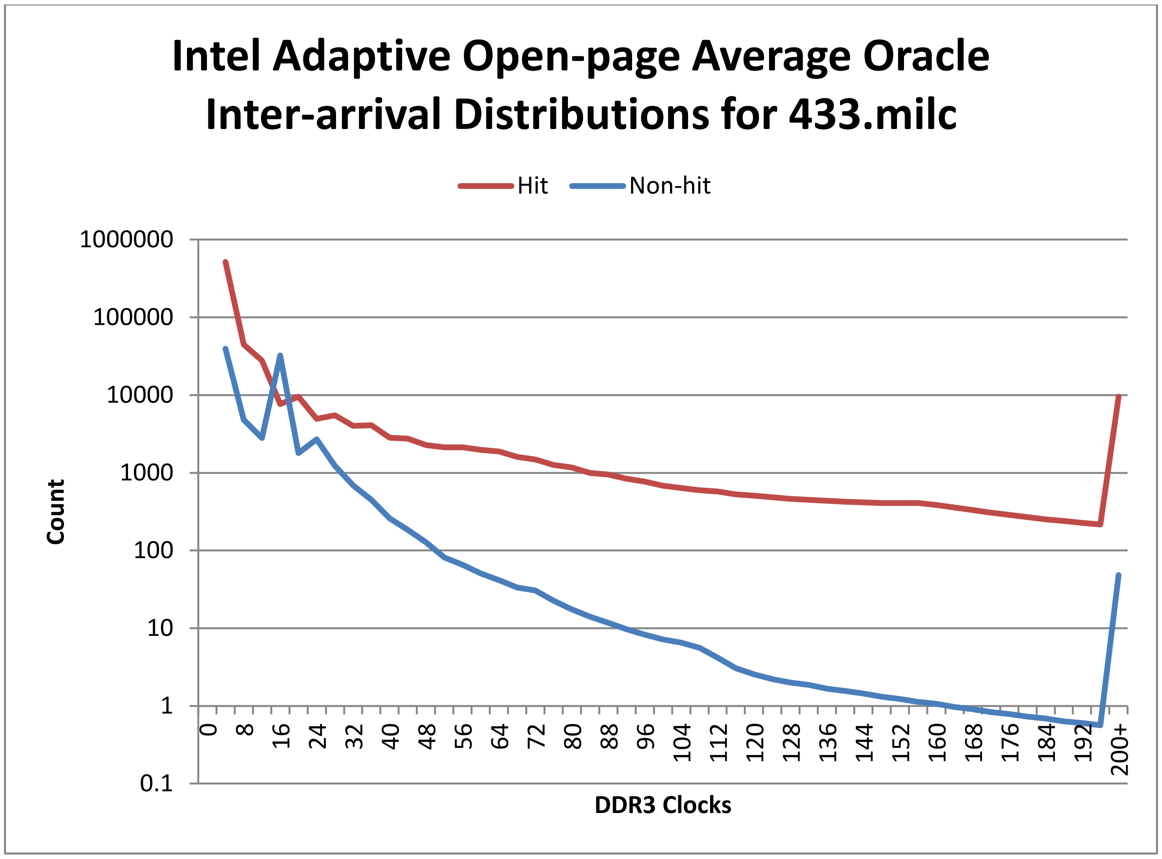


Figure 19: Intel Adaptive Open-page Average Oracle Inter-arrival Distributions for 433.milc

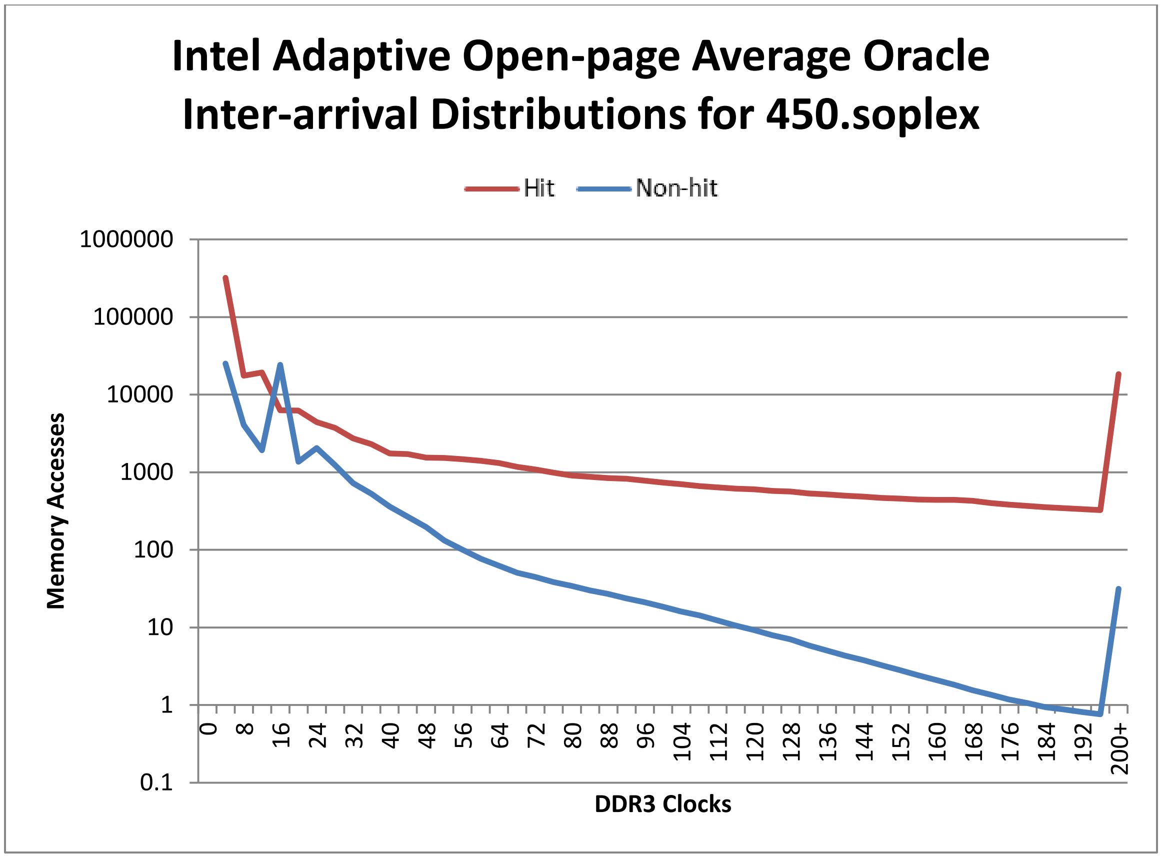


Figure 20: Intel Adaptive Open-page Average Oracle Inter-arrival Distributions for 450.soplex

Both new page policies aim to close pages in memory to minimize the number of page-misses while maximizing page-hits. In the two cases of 433.milc and 450.soplex, there are so many potential page-hits that closing an open page in memory at all can be detrimental to performance. The non-hit distribution spikes at 16 cycles correspond to most of the page-misses for these benchmarks. The previous graphs show that these benchmarks could possibly benefit from a pure open-page policy. A policy that leaves pages open indefinitely has the appeal of capturing the maximum number of page-hits for these types of benchmarks, but leaving pages open can be expensive in terms of power usage.

6.3 Average Memory Access Latency

Figure 21 shows the average estimated memory access latency results for the 10 SPEC CPU2006 benchmarks which had greater than 2GB/s memory bandwidth. This figure compares existing page policies with their simulated oracle policies. Lower latency indicates better performance.

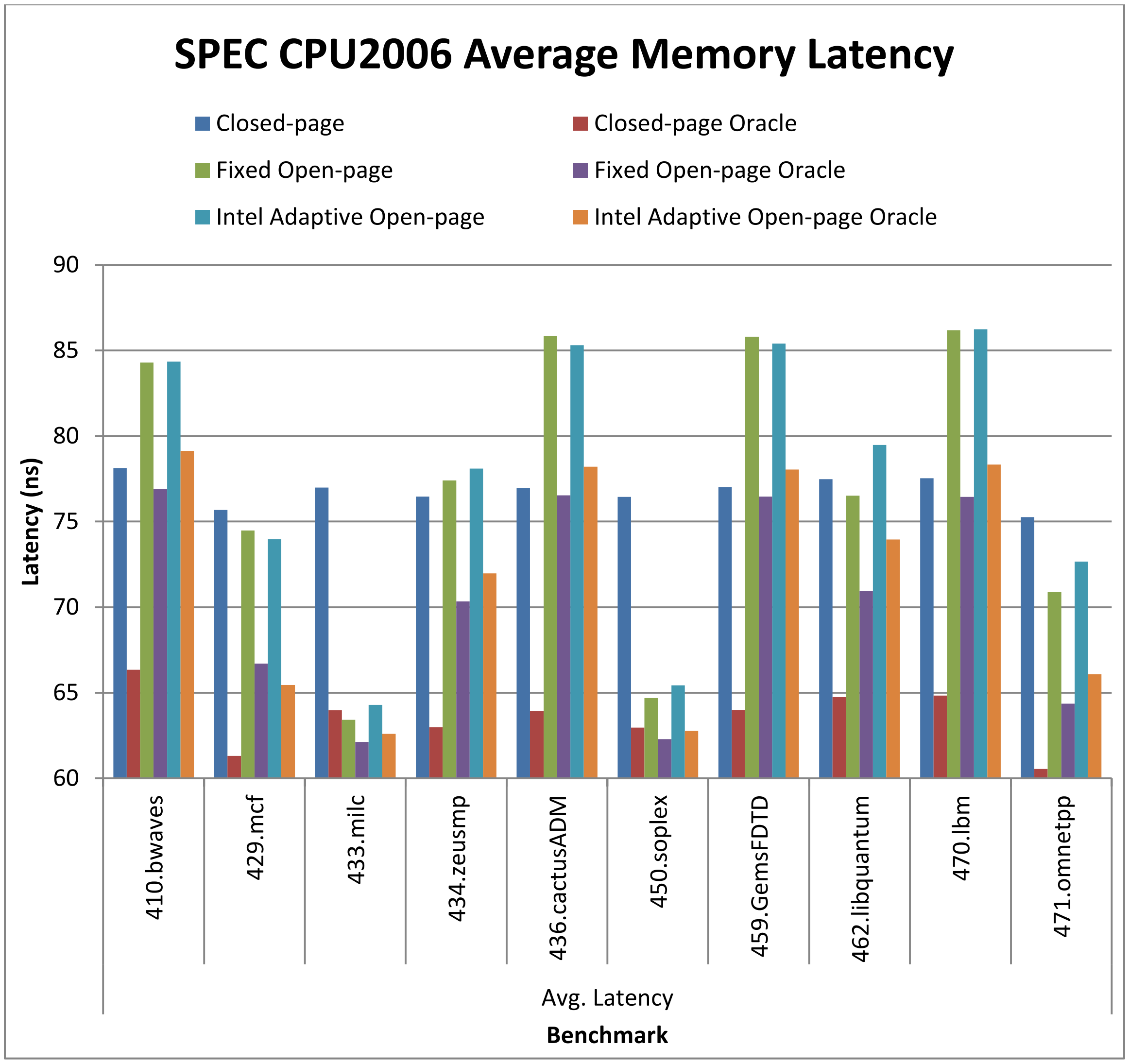


Figure 21: Average Memory Latency – Existing Policies Compared to their Oracle

	Closed-page	**Fixed Open-page**	**Intel Adaptive Open-page**	**Closed-page Oracle**	**Fixed Open-page Oracle**	**Intel Adaptive Open-page Oracle**
Average Memory Latency	76.8	76.9	77.5	63.6	70.3	71.7
% Improvement over Closed-page	0.0%	-0.2%	-0.9%	17.2%	8.5%	6.7%
% Improvement over Fixed Open-page	0.2%	0.0%	-0.7%	17.4%	8.6%	6.9%
% Improvement over Intel Adaptive Open-page	0.9%	0.7%	0.0%	18.0%	9.3%	7.6%

Table 3: Average Memory Latency % Improvement – Existing Policies Compared to their Oracle

Similar to the page-hit minus page-miss metric, the closed-page policy provides the best performance in terms of the average memory latency and has the most potential for improvement.

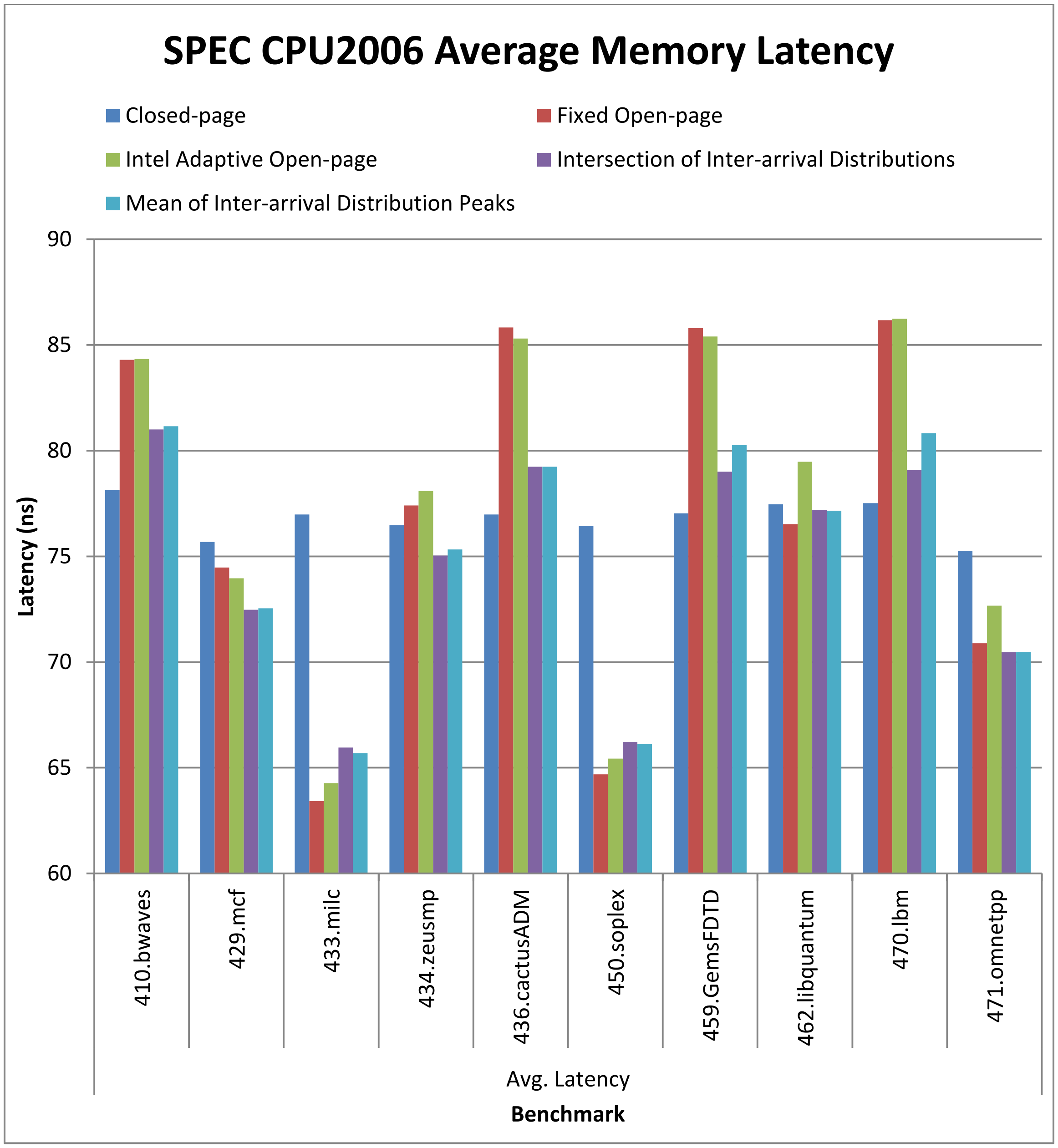


Figure 22: Average Memory Latency – New Policy Compared to Existing

	Closed-page	**Fixed Open-page**	**Intel Adaptive Open-page**	**Intersection of Inter-arrival Distributions Open-page**	**Mean of Inter-arrival Distribution Peaks Open-page**
Average Memory Latency	76.8	76.9	77.5	74.6	74.9
% Improvement over Closed-page	0.0%	-0.2%	-0.9%	2.9%	2.5%
% Improvement over Fixed Open-page	0.2%	0.0%	-0.7%	3.1%	2.7%
% Improvement over Intel Adaptive Open-page	0.9%	0.7%	0.0%	3.8%	3.4%

Table 4: Average Memory Latency % Improvement – New Policy Compared to Existing

Both versions of the inter-arrival open-page policy show improvement over all existing page policies in terms of the average memory latency. The intersection of inter-arrival distributions showed a slight advantage over the mean of inter-arrival distribution peaks.

7 Conclusion

This thesis examined the performance of three existing page policies on an Intel CPU with an integrated memory controller using benchmark runtime, the difference between page-hits and page-misses, and estimated average memory latency as metrics. I introduced the concept of an oracle page policy to establish theoretically optimal performance bounds. I proposed two new page policies based on inter-arrival distributions for the SPEC CPU2006 suite of benchmarks. I then simulated the performance for the SPEC CPU2006 benchmarks under the new page policies using the same metrics used to compare existing policies.

An average decrease in the number of page-hits minus page-misses of 9.9% was shown in the Intel adaptive open-page policy compared to the fixed open-page policy and a decrease of 154.9% compared to the closed page policy. There was an average increase of 110.7% in the page-hits minus page-misses between the Intel adaptive open-page oracle policy and the Intel adaptive open-page policy.

An average increase in the average memory access latency of 0.7% was shown in the Intel adaptive open-page policy compared to the fixed open-page policy and an increase of 0.9% compared to the closed page policy. There was an average increase of 7.6% in the average memory access latency between the Intel adaptive open-page oracle policy and the Intel adaptive open-page policy.

The highest performing of the proposed new inter-arrival-based page policies showed an increase of 35.8% in the number of page-hits minus page-misses over the closed-page policy, 72.3% over the fixed open-page policy, and 74.8% over the Intel adaptive open-page policy. It also displayed a decrease of 2.9% in the average memory access latency over the closed-page policy, 3.1% over the fixed open-page policy, and 3.8% over the Intel adaptive open-page policy.

Visibility into the behavior of memory controllers under different page policies is a big step in validating and tuning the interaction between a CPU and DRAM. Current page policies showed a large variation in effectiveness across the SPEC CPU2006 benchmarks. On average, the existing closed-page policy showed the best performance. However, this policy may exhibit higher power consumption due to the increased number of ACTs that are required. This is why it is important to have an efficient open-page policy. The inter-arrival open-page policy is one option that can serve to keep power consumption low and impose minimal hardware requirements all while improving upon the performance of existing open-page policies.

7.1 Future work

There are a several potential areas where future research is warranted. For simplicity and completeness, only a single DIMM and single processor were used in the target system for memory trace captures. With multi-core computing becoming common-place, the impact of having multiple DIMMs and multiple processors installed in the target should be studied further.

An effective page policy which decreases page-misses and increases page-hits not only affects system performance, but also power consumption. This paper touches on the fact that page policies can alter power consumption of a target machine while running benchmarks, but no quantitative analysis was performed. Further research could include measuring power consumption of a system while it runs benchmarks under varying page policies, including potential new page policies.

The system management bus, or SMBus, is a two-wire bus introduced by Intel in 1995 to provide lightweight system management communication on a motherboard [14]. It can allow peripheral devices to exchange basic information such as device ID, power states, and error information. It is possible that this bus could be utilized to provide real-time feedback to the target system for updating adaptive open-page register values. This could allow closed-loop control of page policy thresholds by linking the PMTA with the target's memory controller. Future research could investigate if this is a viable way to test page policy algorithm improvements.

The existing closed-page policy showed a performance advantage over the existing open-page policies for some of the SPEC CPU2006 benchmarks. This was an unexpected result and was not investigated in this thesis due to time constraints. It may be possible to create a hybrid page policy that behaves like a closed-page policy sometimes and like an open-page policy during other times. Such a policy could detect memory access patterns that are non-sequential and utilize the DDR3 RDA (read with auto-precharge) and WRA (write with auto-precharge) commands to close pages quickly, thus reducing page-misses. The same policy could also detect sequential memory access patterns and begin leaving pages open for a fixed or adaptive amount of time, thus increasing the number of page-hits.

The new inter-arrival open-page policies discussed in this research used 1000 reads and writes as a threshold to re-calculate a new page timeout value. This number was chosen because it represented enough memory accesses to show access patterns, while still being responsive enough to keep up with quickly changing workloads. Future research could investigate altering this threshold number of reads and writes to attempt to gain further performance improvement.

References

[1] K. Boughton and R. Gill, "Everything you always wanted to know about sdram memory but were afraid to ask," August 2010. [Online]. Available: http://www.anandtech.com/show/3851/everything-you-always-wanted-to-know-about-sdram-memory-but-were-afraid-to-ask.

[2] Intel, "Intel® Xeon® Processor X5650," [Online]. Available: http://ark.intel.com/products/47922/Intel-Xeon-Processor-X5650-12M-Cache-2_66-GHz-6_40-GTs-Intel-QPI. [Accessed 25 October 2012].

[3] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte and O. Mutlu, "Memory Power Management via Dynamic Voltage/Frequency Scaling," in *ICAC*, Karlsruhe, Germany, 2011, pp. 5-7.

[4] S. Jacob, S. Ng and D. Wang, Memory Systems: Cache, DRAM, Disk, Morgan Kaufmann Publishers, 2007.

[5] JEDEC, "DDR3 SDRAM Specification," 2010. [Online]. Available: http://www.jedec.org/sites/default/files/docs/JESD79-3E.pdf.

[6] Micron Technology, "Synchronous DRAM Datasheet," [Online]. Available: http://download.micron.com/pdf/datasheets/dram/sdram/512MbSDRAM.pdf.

[7] S. Albert, "Development of a fast DRAM Analyzer and Measurement of Typical and Critical Memory Access Sequences in Applications," *PhD Dissertation, Institute for Integrated Systems, Technical University of Munich*, 2008.

[8] B. Flanagan, B. Nelson, J. Archibald and K. Grimsrud, "BACH: BYU Address Collection Hardware, The Collection of Complete Traces," in *International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*, 1992.

[9] R. Uhlig and T. Mudge, "Trace-driven Memory Simulation: A Survey," *ACM Computing Surveys*, vol. 29, no. 2, 1997.

[10] PCI-SIG, "PCIe Base 2.1 Specification," [Online]. Available: http://www.pcisig.com/specifications/pciexpress/base2/. [Accessed 17 June 2012].

[11] W. Digital, "WD Caviar Black Desktop Hard Drives," 2008. [Online]. Available: static.highspeedbackbone.net/pdf/WD-Caviar-Black-Datasheet.pdf.

[12] S. Burkhart, R. Chase, J. Arada and K. Morris, "PMTA Specification," 2010.

[13] Standard Performance Evaluation Corporation, "SPEC CPU2006," [Online]. Available: http://www.spec.org/cpu2006/. [Accessed 7 May 2012].

[14] "SMBus," [Online]. Available: http://smbus.org/. [Accessed 7 May 2012].