FINDING ENERGY SUSTAINABLE TECHNIQUES FOR COMPUTER MEMORY

Thesis submitted in fulfillment of the requirements for the Degree of

DOCTOR OF PHILOSOPHY

By

AASTHA MODGIL

Under the supervision of

DR. VIVEK KUMAR SEHGAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,
WAKNAGHAT, H.P.

CANDIDATE'S DECLARATION

I hereby declare that the work reported in the Ph.D. thesis entitled **"Finding Energy Sustainable Techniques for Computer Memory"** submitted at **Jaypee University of Information Technology, Waknaghat, Solan (HP), India** is an authentic record of my work carried out under the supervision of **Dr. Vivek Kumar Sehgal**. I have not submitted this work elsewhere for any other degree or diploma. I am fully responsible for the contents of my Ph.D. thesis.

Aastha Modgil
(Enrollment No.: 136216)
Department of Computer Science & Engineering
Jaypee University of Information Technology, Waknaghat, Solan (HP), India
Date:

SUPERVISOR'S CERTIFICATE

This is to certify that the work reported in the Ph.D. thesis entitled "**FINDING ENERGY SUSTAINABLE TECHNIQUES FOR COMPUTER MEMORY**", submitted by **Aastha Modgil** at **Jaypee University of Information Technology, Waknaghat, Solan (HP), India** is a bonafide record of his original work carried out under my supervision. This work has not been submitted elsewhere for any other degree or diploma.

Dr. Vivek Kumar Sehgal
Associate Professor
JUIT, Solan
Date:

TABLE OF CONTENTS

ACKNOWLEDGEMENT

I express my warm gratitude and profound respect to my supervisor **Dr. Vivek Kumar Sehgal, Associate Professor and Ph.D. coordinator**, Department of Computer Science & Engineering and IT, Jaypee University of Information Technology, Waknaghat, Solan, Himachal Pradesh, India, under whose valuable guidance the work has been completed successfully. One can learn a lot from him and I found his guidance as a golden opportunity to me. The author would like to express special thanks to **Dr. S.P. Ghrera, Professor and Head**, Department of Computer Science & Engineering and IT, Jaypee University of Information Technology, Waknaghat, Solan, Himachal Pradesh, India, for motivating and encouraging me in every crucial moment.

My committee members have also had a big impact on the quality and scope of my work. I am thankful for the valuable suggestions I received from my DPMC members, **Prof. (Dr.) Sunil Kha**, Department of Physics and Material Science, **Dr. Shailendra Shukla,** Assistant Professor (Senior Grade), Department of Computer Science & Engineering and IT, and **Dr. Geetanjali**, Assistant Professor (Senior Grade), Department of Computer Science & Engineering and IT.

I express my gratitude to **Professor (Dr.) Vinod Kumar**, Vice Chancellor of Jaypee University of Information Technology and **Professor (Dr.) Samir Dev Gupta**, Director and academic head for providing an inert and conducive research environment.

I am thankful to lab staff for providing good experimental environment and equipment which helped me to accomplish my simulation work.

I am also grateful to my parents, **Mr. Arvind Modgil** and **Mrs. Kusum Modgil** for their love, care and moral support. I would like to thank my younger brother, **Mr. Aviral Modgil** for his valuable inputs and support. Last but not the least, I thank my husband **Maj. Ashish Thakur** for being pillar of strength throughout this research period.

Finally, I thank to eminent reviewers of my research work and every person from whom affection, help, guidance, support, and inspiration I received during my research work.

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The increasing growth of internet based information infrastructure consists of powerful computers and data centers has led to the development of personal/mobile computing devices. These datacenters are required to perform heavy data processing while ensuring quality of service. These highly powerful performance oriented computing platforms are used to model and analyze various complicated scientific problems and natural phenomenon. These computing devices are capable of providing high performance requirements of various complex applications only if they are facilitated with efficient memory system. The memory system stores data during and after execution and provides data to processing cores necessary to complete execution of applications. It is main memory system that stores data structure necessary for completion of a program. Main memory constitutes a major part in overall system's power consumption. Researchers have reported that in mid-range IBM eServer machine, main memory contributed 40% of the total system's power consumption. Many researchers have worked to make DRAM based main memory system efficient in terms of performance, energy and access locality.

The prime factor that affects the efficiency of main memory system is its memory controller. Memory accesses constitute an important part of total applications energy consumption. It is memory controller of main memory system that is responsible for its efficient working. The decision regarding what command to issue and when, is dependent on main memory controller. Main memory controller makes this decision on the basis of memory access scheduler used. Based on utilized memory scheduling policy DRAM requests are serviced, the sequence of servicing requests targeted to Dynamic-RAM largely affects the performance as well as energy consumed by main memory sub-system.

This thesis investigates the impact of memory scheduling policy on energy consumption and performance of main memory system in several situations. We propose four scheduling policies that try to service main memory requests in more energy efficient manner while maintaining the performance of main memory system. We have tried to rationalize main memory's energy consumption by reducing its active power consumption and access latency. First, we have worked on, i) reducing DRAM's active power consumption by optimizing row buffer hits and ensuring fairness among simultaneously running threads; ii) by optimizing read-write switching; iii) by reducing DRAM's refresh induced energy and performance overheads.

Second, we have worked on reducing DRAM's access latency by prefetching memory requests and issuing precharge and activate commands accordingly.

CHAPTER 1

INTRODUCTION

Advancement in the performance of current computer systems is due to improvements in silicon process technology. As per Moore's law due to improvement in silicon process technology, the count of transistors on a single chip can double in every two years. As a corollary to Moore's law the increase in the performance of processor has also doubled in approximately every two years (*i.e.*, during same time period) due to increased switching speed of transistors. This increase in switching speed is observed because of increase in transistor count. However, solely improving the performance and energy consumption of processor does not always lead to reflect same performance and energy consumption gain while considering whole computer system. The reason for such disparity in performance and energy consumption advancement at processor level and whole computer system level is that performance of computer system and energy consumed by computer system is dependent on performance and energy consumption of processor, computer memory and interaction between processor and computer memory as well. Moreover, in passing years, in contrast to the rapid advancement for improving performance of processor and energy consumed by processor, computer memory has shown modest improvement. Imbalanced performance scaling and energy consumption trends shown by processor and memory results in restricted advancement of whole computer system. In order to gain benefits at whole computer system level, detailed prominence should be made on efficiently managing the performance as well as energy consumption of computer memory. As per [1], main memory contributes towards 40% of the total system's power consumption. However, JEDEC [2] and different DRAM vendors are continuously working on DRAM memory sub-system [3-8], to make DRAM based main memory subsystem efficient in terms of bandwidth and energy. Due to increased requirement for more memory capacity and improved performance, larger and faster main memory system is incorporated with every new release, which further results in increased power consumption of main memory system [9], hence main memory sub-system is required to be improved day by day.

With increase in operating frequency of processor, power dissipation is also increased. This "heat wall" is the main reason for shifting towards multicore processors, *i.e.*, increasing the number of cores than to increase frequency in order to manage processor's energy

consumption. On multicore processors multiple threads run simultaneously for fast execution. These concurrently executing threads share main memory resource for storing intermediate results or to retrieve data required to complete execution. So, main memory is the major resource that is being shared by all the running programs. Hence, main memory is the key contributor towards overall system's performance and energy consumption [10]. DRAM being a crucial component for energy optimization in industries [10-14] as well as in academia [15-27], many researchers are working on finding the solutions at system level down to circuit-level for energy optimization. Researchers are trying to optimize active as well as idle power consumption of DRAM. So, in current scenario while designing a computer system two major goals are required to be met, i) To improve the performance of the main memory sub-system at same level of energy consumption, or ii) To decrease energy consumption of the main memory sub-system at same level of performance.

1.1 Future's Performance Requirement

Demand for improved performance of the main memory is on rise. Various commercial applications like applications from the field of "Big Data" make extensive use of main memory resource. Applications related to Big data intend to find out useful and meaningful information from many thousands of petabytes of available data. In order to extract this information multiple threads run concurrently, thanks to multicore processors. This high degree of parallel execution of data results in extensive pressure on memory in terms of capacity (bandwidth) to fulfil processor's demands. This aroused demand is accomplished by DRAM vendor by increasing frequency of DRAM pin. However, total available DRAM bandwidth is constrained by the number of pins available on socket. As per ITRS road map [28], over a period of 8 years, pin counts are estimated to increase by only 1.47 times unlike 16 times growth of transistor count of processor. With different growth trends observed for processor and memory, it is very important that queuing delay observed by requests generated by multiple core contending for limited memory pins is required to be reduced. The efficient utilization of main memory bandwidth is dependent on optimal use of DRAM banks, which can be achieved by efficient scheduling of memory requests, hence motivates to design efficient memory controller.

1.2 DRAM Latency

Over a period of time, in different DRAM generations a significant increment in DRAM pin count is observed, whereas, very less decrement in DRAM core latency is achieved. tRCD and tRC are timing parameters that decides DRAM latency. On one hand, where the transmission

latency to transmit data to processor over DRAM interface has reduced but DRAM core latency is not scaled. This leads to a scenario where concurrently running threads contend with each other for accessing DRAM banks resulting increased overall latency. Increased overall latency not only increases the overall execution time of programs but also energy consumed by the program for completion. This issue further motivates to design efficient memory controller.

1.3 Thesis Overview

From the above discussion, it is very much clear that the memory latency wall and energy consumption is a major issue that requires to be addressed. In this thesis work, we look for the optimization of memory system that not only satisfy performance requirements of workloads but also provide increased performance level at less or same energy consumption, or reduced energy consumption constraints for same performance gain. In order to gain such benefits we chose memory controller which is most important part of main memory system and where changes can be made at minimum cost impact. Memory controller is responsible for issuing memory commands in every DRAM clock cycle and memory controller issues DRAM commands based on underlying memory access scheduler. Many studies [29-38] have been conducted to understand the influence of memory controller's scheduling algorithms on performance and energy consumption of main memory system. These studies report that memory scheduling policies affect main memory subsystem's performance and the amount of energy consumed in significant way. In this thesis, we try to find answers to the questions like, How to prioritize threads? What are those memory requests which are to be served first for improved performance and energy consumption? How to improve fairness among threads? Answers to these questions are provided through different memory scheduling policies. In this dissertation, we propose four memory scheduling policies that focus on improving performance, fairness among threads and energy consumption of main memory system.

1.4 Problem Statement

Main memory's performance and energy consumption affects computer system's performance and energy consumption in a significant way. Memory controller of the main memory system is a key component that have a major impact on main memory system's performance and energy consumption. The key to design performance neutral, energy efficient main memory system lies in the formulation of intelligent as well as efficient memory controller policies that are aware of memory access scenarios and timing constraints and issues memory requests accordingly to maintain the balance between energy consumption and performance.

1.5 Proposed Solutions

To address the issue of rationalizing energy consumption and performance gain of main memory sub-system, we propose following performance-neutral DRAM energy optimization strategies.

1 To reduce DRAM's power consumption while maintaining the performance level.
2 To reduce DRAM access latency

DRAM based main memory system's power consumption can be managed by either reducing its active power consumption or by minimizing DRAM's standby power consumption. DRAM's active power consumption can be curtailed by, i) maximizing row-buffer hits, ii) reducing read-write switches, iii) managing DRAM refreshes while servicing memory requests. The downside of active power consumption optimizing strategy is that it reduces DRAM access time which leads to increased DRAM idleness. Also, these idle DRAM periods should be optimized in order to reduce overall DRAM energy consumption. Power consumed by DRAM system while in standby (idle) mode can be curtailed by, i) exploiting power down modes, ii) curtailing power consumed during refresh operations when memory is idle, iii) frequency scaling to reduce memory's idleness. The disadvantage of exploiting power down mode and frequency scaling is that they may result in increased latency and energy consumption to serve memory accesses caused because of additional power-up latency.

1.6 Thesis Layout

The thesis has seven chapters, out of which CHAPTER 1 presents Introduction. CHAPTER 2 enlightens about fundamental building blocks of main memory system (DRAM based), basic terminologies and existing memory scheduling policies is provided. CHAPTER 3 presents the proposed energy efficient and fair memory scheduling algorithm to optimize row hits while maintaining fairness among threads during execution. CHAPTER 4 presents memory access scheduling policy proposed to reduce read write switches in order to minimize bus turn around delay and power consumed to switch the bus, hence serves to reduce thread's execution time and energy consumption. CHAPTER 5 presents proposed memory access scheduling algorithm to minimize DRAM access latency. CHAPTER 6 includes scheduling approach proposed to manage DRAM refresh induced performance and energy overheads. Finally, CHAPTER 7 presents the conclusion of the dissertation supported with the result of experiments and simulations. Conclusion is further followed by the future scope of the research work commenced in the thesis.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

2.1 Basic DRAM Device Architecture and Circuits

In order to enlighten the researchers about DRAM based main memory system, this chapter provides an insight into basic architecture and circuits of DRAM devices. Though it is not possible to complete all aspects of DRAM based circuits and architectures in depth in one chapter, the fundamental aim of this chapter is to facilitate sufficient details about DRAM devices and their architecture and circuits to ensure basic understanding about DRAM devices. Only if basic understanding about fundamentals of DRAM devices is available to the researchers then only more advanced discussions about its architecture and circuits would be feasible. Current chapter begins with the description of basic DRAM device, *i.e.*, Fast page mode (FPM) DRAM device. Then varied components like DRAM cell, sense amplifier are discussed separately.

2.1.1 *DRAM Device Organisation*

Figure 2.1, demonstrates the fundamental structure and basic details of DRAM device.

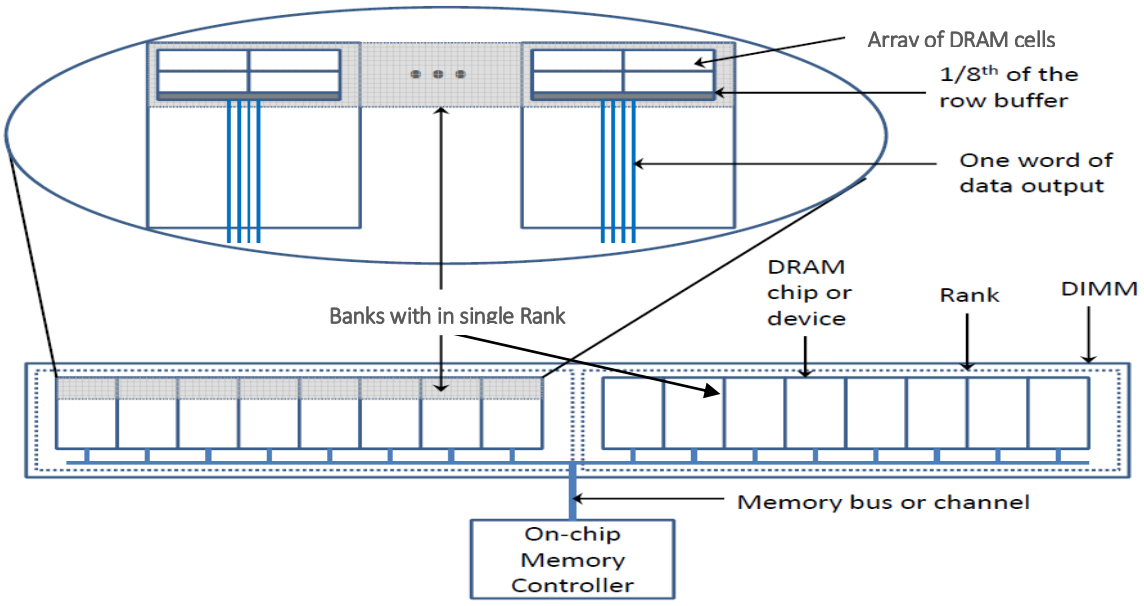

Figure 2.1 Basic DRAM Device based Memory Organisation

Figure 2.2 [39], illustrates that DRAM storage cells are organized as an array, consisting of 4096 rows and 1024 columns in each row. In each column 16 bits of data is facilitated. In order

to access data stored in DRAM array, Row Access Strobe (RAS) occurs, in which 12-bit address is placed on address bus. Internally in DRAM device, the address inserted by external memory controller is stored in row address buffer which is further sent to row address decoder for decoding the address. The row decoder then decodes the address and selects the corresponding row out of 4096 rows available. The data stored in chosen row is then sensed and stored in sense amplifier. Each row contains 1024 columns, where each column is 16-bit wide, *i.e.*, fundamental addressable unit of this memory device is 16-bit, and during column access strobe (CAS) 16-bits of data is either read or write from selected row of DRAM device. CAS can further be issued as column access strobe low (CASL) to access lower 8-bits and column access strobe high (CASH) to access higher 8-bits independently. External memory controller places 10-bit address to access specified column from chosen row on address bus, then asserts appropriate CASH or CASL signals. Then the data of the selected column is placed onto the bus.

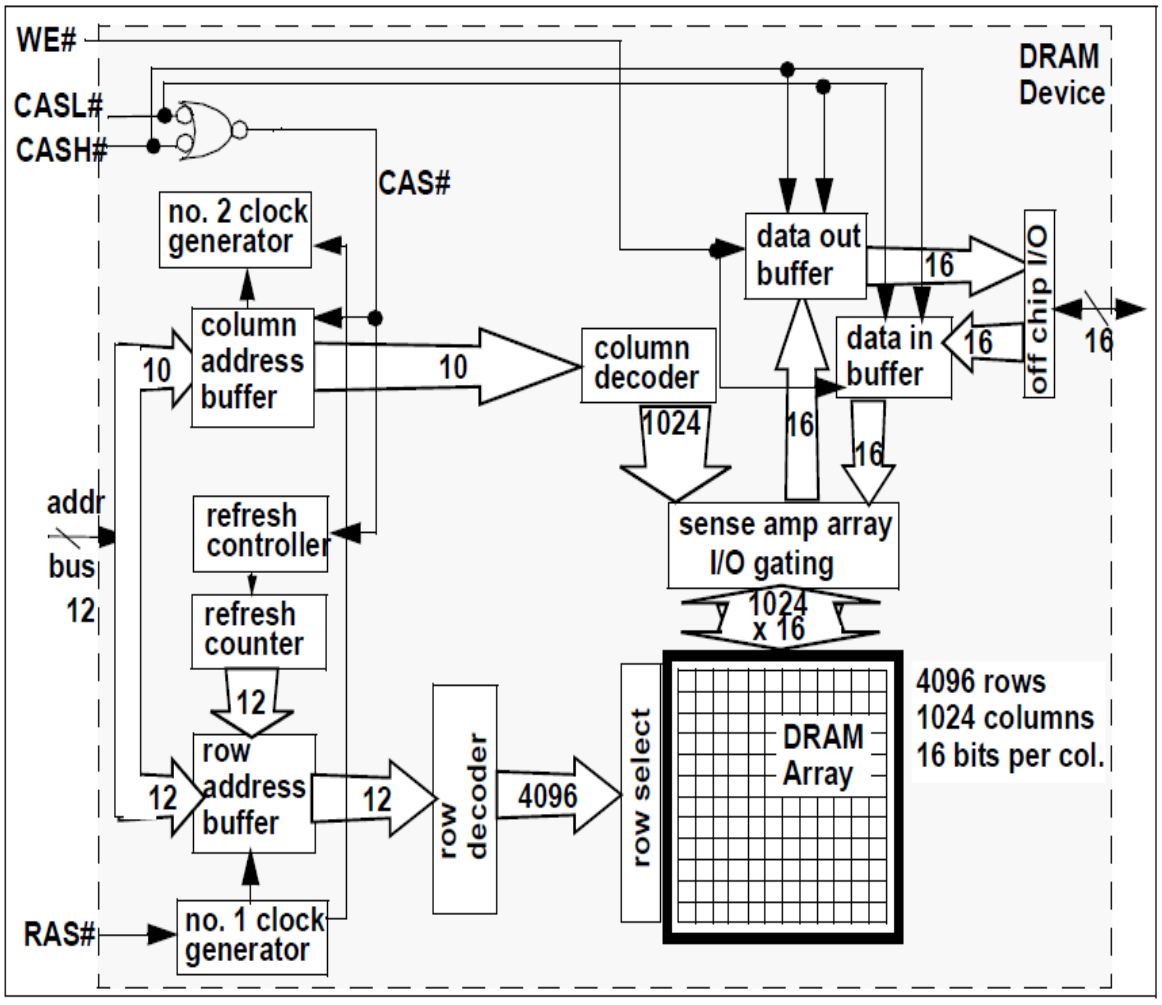

Figure 2.2 DRAM Cell Organisation

This basic organization is similar for all DRAM devices ranging from basic FPM DRAM to DDRx (Dual Data Rate) SDRAM devices. In all DRAM devices DRAM cells are organized as an array, *i.e.*, as rows and columns. All DRAM devices contains one or more such DRAM cell arrays, where column is the basic addressable memory unit. All DRAM devices are enabled with some logic circuits to control timings and sequence of operation of device. The FPM DRAM device keeps track of the address of next row that is to be refreshed. Due to restricted pin usage on DRAM devices, dual-sided input-output pins are attached to the system for moving data into and out of the device. More innovative DRAM devices like ESDRAM, Direct RDRAM and RLDRAM contains more functionalities and logic circuitry but burden the die cost of Dynamic-RAM device. So, these advanced devices are not included as standard DRAM devices.

2.1.2 *DRAM Cell*

Figure 2.3, illustrates the circuit diagram of basic DRAM cell, containing one transistor and one capacitor, which is the fundamental storage unit of DRAM device. Access transistor is activated by supplying voltage on the wordline gate of transistor. The data to be stored may be placed onto the bitline by applying voltage representing data value which is used to charge the storage capacitor. The capacitor then holds the charge for restricted duration after the removal of voltage from wordline. The transistor turns off when voltage supply from the wordline is removed. The charge stored on capacitor tends to leak over time, so, it becomes necessary to recharge (refresh) the cell after a regular intervals to retain the stored charge, or else the deposited charge will leak away and the value maintained on the device will vanish.

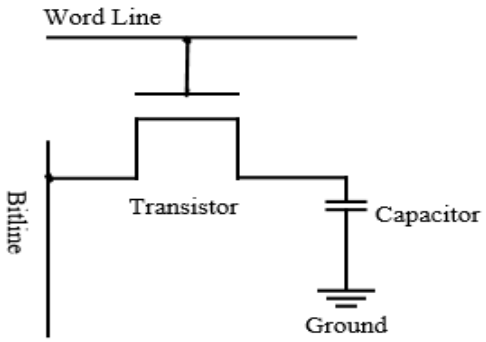

Figure 2.3 DRAM cell

2.1.3 *Sense Amplifier*

In DRAM devices, sense amplifier perform three different functions. Out of these functions, primary function is to sense the minor variation in voltage that may occur when capacitor places

its charge on the bitline. The sense amplifier makes a comparison between voltage on bitline and a reference voltage. Sense amplifier then amplifies the voltage so that stored value can be read as zero or one. Another function of sense amplifier is to restore cell value after the value present on bitline is sensed and amplified. When capacitor places its charge on bitline, it gets discharged. As a result, its charge needs to be restored. The third function of sense amplifier is that sense amplifier also act as temporary storage in the DRAM array. After seeing and amplifying the data contained in storage cells, the sense amplifier continue to retain the data values until the precharge operation is not executed on DRAM array. This way, data in the row present in sense amplifier can be accessed without issuing RAS. For this array of sense amplifiers collectively act as a row buffer and hence sense amplifier is also addressed as row buffer. Researchers have worked on managing the state of sense amplifier, *i.e.*, for how long row buffer would continue to retain data. These policies are referred to as row buffer management policies. Effective management of state of sense amplifier is very important for obtaining optimal balance between performance and energy consumption. Basic sense amplifier circuit diagram is shown in Figure 2.4[39].

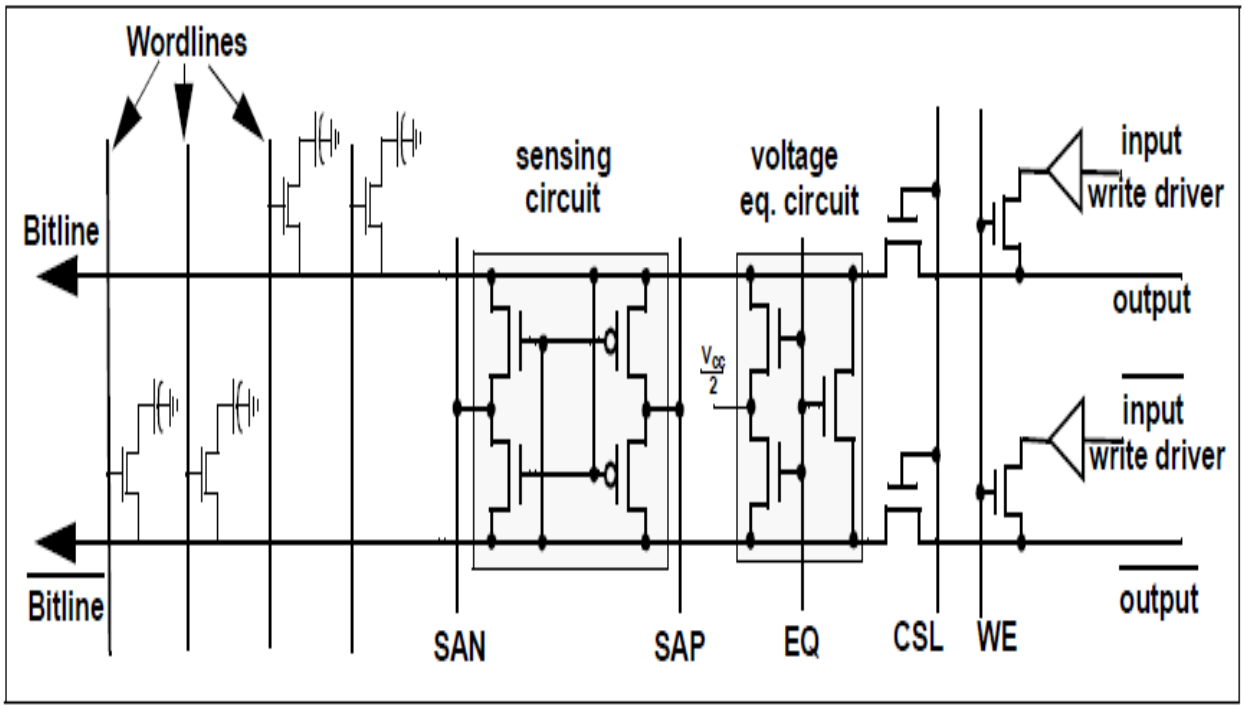

Figure 2.4 Circuit diagram of Sense Amplifier

2.1.4 *DRAM Device Control Logic*

Movement of data onto, within and out of the DRAM device is controlled by basic logic circuit present in DRAM devices. Logic circuitry accepts signals and control that are fed externally

and then controls the data movement by properly sequencing the commands as per timing constraints. The issuing of control signal at proper time is controlled by logic circuitry. Figure 2.5 [39], represents the logic circuitry of FPM DRAM device. External interface of DRAM device's control logic consists of three signals: row access strobe (RAS), write enable (WE), and column access strobe (CAS). The device interface is asynchronous and memory controller controls the issue of command pertaining to timing constraints for regulating data movement into and out of DRAM device. Asynchronous nature of interface reveals that different memory controllers working at different frequencies can be implemented but the controller should be able to control different DRAM devices.

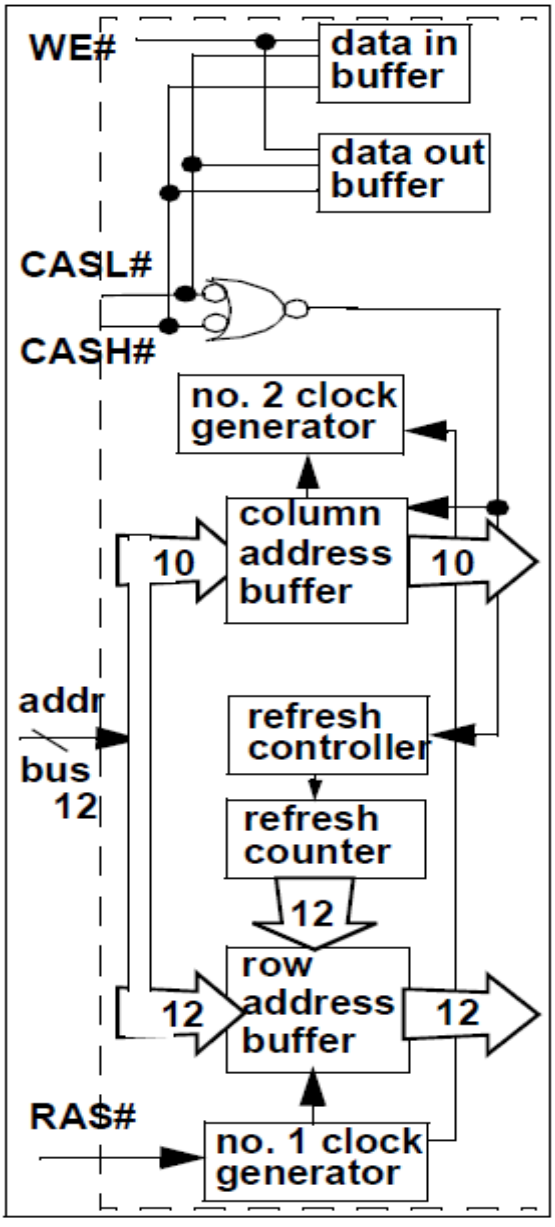

Figure 2.5 Control Logic of DRAM Device

2.1.5 *Data Input/Output*

In DDRx SDRAM and SDRAM devices, variable number of columns are moved on issuing column read command. According to programing done on an SDRAM device, it can return a single burst of 1, 2, 4 or 8 columns of data in 1, 2, 4 or 8 cycles. In 2, 4 or 8 column burst of Synchronous-DRAM based device, each column is individually addressable, *i.e.*, even if a column is provided in between burst, the Synchronous-DRAM device reorders the burst in order to at first facilitate data of requested address. This characteristic of device is called as critical-word forwarding. Each column in the burst is moved separately from the sense amplifier to external data bus. The individual control over each column and the operational data rate of DRAM device is constrained. To overcome this issue large number of bits are moved in parallel from DRAM and are then pipelined to external data bus through multiplexer.

2.2 DRAM based Main Memory Organization

In this section an insight about fundamental building blocks of main memory system (DRAM based), basic terminologies is provided. In previous section main focus was on single DRAM device, whereas, in this section details about whole main memory system is provided. In context with whole main memory system information about organisation, construction and working of multiple DRAM devices together is provided. The aim to include detailed information about DRAM based main memory system organisation and nomenclature in this chapter is to provide proper understanding about the building blocks and operations of memory system, so that the nomenclature used and discussions made in subsequent chapters can be better understood.

2.2.1 *Main Memory Subsystem*

In a computer system, main memory subsystem acts as an intermediate storage between caches and secondary memory. During processing, it is used for storing intermediate data that can be used by the processor in near future, or to store data expelled by the cache memory. In order to behave like an intermediate memory, it has to be faster than hard drives to accomplish better performance. For faster access to the data, memory should be randomly accessible, hence main memory is random access memory. Constant charge leaking behaviour of DRAM makes main memory volatile in nature, hence it requires periodic refreshes. The main memory subsystem is accessed through a hardware device called memory controller which is placed on motherboard, in recent computer system it is integrated on CPU die. Memory controller controls the access made to main memory through system buses, data buses and address buses.

Modern main memory system is made up of JEDEC style dual data rate (DDR) synchronous DRAM (SDRAM) [4, 5]. Dual data rate part of main memory reveals that the data bus operates two times faster than the address bus and command bus. Synchronous DRAM means on issue of RAS and CAS the DRAM device acts at falling edge of clock signal, not immediately upon access strobe signal change. Dynamic part of DRAM depicts its volatile nature. Constant charge leaking behaviour of DRAM makes main memory volatile, hence it requires periodic refreshes to retain the stored data. Main memory subsystem is organized hierarchically as channels, ranks and banks. Within a bank, DRAM devices are organized as an array of rows and columns. Memory controller is designated to manage one or more main memory channels. Each channel is provided with a data bus, an address bus and a command bus. Multiple Dual Inline Memory Module (DIMM) is provided at each channel. DIMM is a collection of multiple ranks. Ranks on same channel share common data bus, command bus and address bus. Within a memory module multiple ranks can work together in parallel to service different memory accesses. Figure 2.6 [39] shows the hierarchical organisation of main memory system. Rank is a collection of multiple DRAM chips. A particular rank is chosen by issuing chip-select signals. On selecting a rank all the DRAM chips that are part of rank receive signals (address signals and command signals) issued by memory controller on command system buses. A rank is further partitioned into multiple banks, (typically ranging from four to sixteen). Each bank can concurrently service different memory accesses but the data being transferred into or out of the bank through common data system has to be serialized. Each bank is logically thought to be arranged as a two-dimensional array of DRAM cells. Physically, each two-dimensional array is further divided into sub-arrays [39, 40] in order to manage factors like current draw and latency.

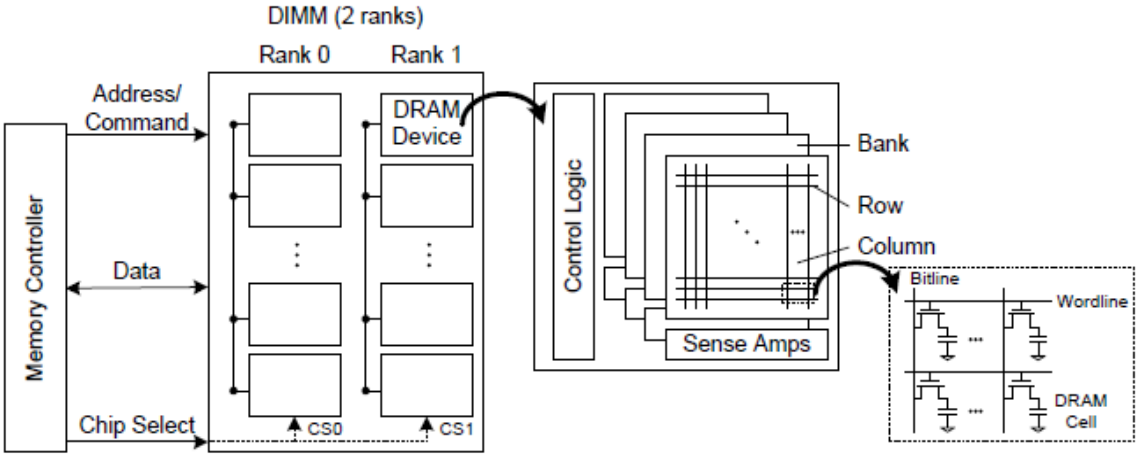

Figure 2.6 Main Memory Hierarchy

2.2.2 *DRAM Commands*

In main memory system DRAM cells are organized as rows and columns. This 2-D array arrangement of DRAM cells facilitates to address particular bit of information through addressing specific row and column. A row of sense amplifiers, one sense amplifier per column is also present in each array which act as a buffer for data management during cell refreshes. First of all PRECHARGE command is issued to access the data stored at row X and column Y. PRECHARGE command prepares the sense amplifier to receive data. PRECHARGE command is followed by activate command, which reads out data from DRAM cells to row buffer (sense amplifiers). Once sense amplifiers contain data, the corresponding row (row X) is said to be open. To access a particular column (column Y) from the row, column access strobe corresponding to read/write column Y is issued. On completing a particular memory request after ACTIVATE command next PRECHARGE command prepares row buffer for storing next row being addressed. ACTIVATE and PRECHARGE operations on a row refreshes the DRAM cells. In addition to these, REFRESH command is issued on periodic intervals to refresh (recharge) the DRAM cells.

Other than basic DRAM commands (PRECHARGE, ACTIVATE, COL-READ, COL-WRITE, and REFRESH), DDR SDRAM supports many other commands to manage DRAM's power state. PWR-DWN-FAST (Power Down Fast) can put rank in either activate power down mode or fast-power-down mode (precharge-power-down). While issuing power down if all the banks are in precharge state then chips enter in precharge-power-down mode, whereas, if even a single bank is active then chip enters in active-power-down mode. In both states on chip DLL (Delay Lached Loop) remains active. The power consumed in active power down mode is more when compared to power consumed in precharge power down mode. In both power down states, the on-chip DLL is ON that enables the chip to power-up with least latency. To ensure transition into the lower power state, it may be necessary to first precharge all banks in the rank. Another command that can be issued to manage power state of DRAM is PWR-DWN-SLOW (Power Down Slow), which transitions a rank into power down slow mode. It can be applied only if all banks are in precharge state. In power down-slow-mode on chip DLL is inactive. A rank can be woken up from low-power mode using power-up command, refresh command, precharge command, or precharge-all-banks command. Time taken by power-up command to transition ranks into active state depends upon whether the DRAM banks are in power-down-slow or power-down-fast mode. If the chip is in active-power-down state then

power up command retains the data in row buffer when the chip is powered up. Precharge-all-banks forcibly precharges all the banks in a rank.

2.2.3 *Timing Constraints*

Physical implementation of DRAM devices imposes certain timing limitations. These limitations arise due to signalling constraints, power profiles, and wiring limitations. Due to timing constraints only few commands among all DRAM commands can be issued during a clock cycle. Which command is to be issued depends on current DRAM state. Table 2.1, provides a list of timing parameters for a typical Micron DDR3 chip. The value of tRFC parameter is dependent on chip capacity and varies in accordance with capacity.

Table 2.1 Timing Parameters

Timing Parameter	Default Value (cycles at 800 Mz)	Description
tRCD	11	Delay between Row and Column Command. It represents time duration between data open at sense amplifiers and accessed from row buffer.
tRP	11	It represents Row Precharge, *i.e.*, time duration needed to precharge DRAM array for another row access.
tCAS	11	It constitutes the latency encountered in Column Access Strobe. The period of time between column access command and the beginning return of data by the main memory device. It is also known as tCL.
tRC	39	Row Cycle. Time required to access multiple rows within a bank. tRC=tRAS+tRP
tRAS	5	It represents the minimum delay between one Row activation to another Row activation command. It restricts the maximal current profile.
tFAW	32	Four (row) bank Activation Window. Utmost time period for engaging maximal four activated banks.
tWR	12	Write Recovery time. It represents the minimal time duration betwixt the initiation of a precharge command and the termination of a write data burst.

tWTR	6	Write to Read delay time. The minimal interim time betwixt the beginning of a column read command and the culmination of a write data burst.
tRTP	6	Read to Precharge.The interlude duration between a read and a precharge command.
tCCD	4	Column-to-Column Delay. The minimal column command timing, governed by internal burst (prefetch) length.
tRFC	128	Refresh Cycle Time. The interlude between Refresh and Activation commands.
tREFI	6240	Refresh interval period.
tCWD	5	Column Write Delay. The interim time between the deployment of data on the data bus by the DRAM controller and issuance of the column-write command.
tRTRS	2	Rank-to-rank switching time. Utilized in DDR and DDR2 SDRAM memory systems; not used in SDRAM or Direct RDRAM memory systems. One full cycle in DDR SDRAM.
tPDMIN	4	Minimal power down duration.
tXP	5	Time to depart fast power down.
tXPDLL	20	Time to depart slow power down.
tDATATRANS	4	It represents Data transfer time from CPU to memory or conversely.

At a temperature of 85 degree Celsius DRAM rows need to be refreshed with in a time interval of 64ms. The refresh operation takes place in following steps. First, memory controller issues REFRESH command in every 7.8µs (refresh interval). This command triggers refresh to multiple rows in all the banks in the channel. For tRFC interval after issuing REFRESH command DRAM chips are not available to service any other command. In accordance to JEDEC standard REFRESH command can be delayed up to 8 times tREFI, if average rate of refresh command is one per tREFI [41].

2.2.4 *DRAM Access Latency*

The sense amplifiers are used to sense each bank of memory. Within a DRAM chip the range of row size is 1-2 KB [42]. First, Activation command is allotted in order to fetch a row from memory array to row buffer. Data present in row buffer can be read out by issuing column read command, whereas data can be written into address in row buffer by column write command. To read data from the memory addresses or to write data into a particular memory addresses present in row buffer, column address command is only required only [43]. The energy consumed for serving memory request and stall time experienced throughout the execution of a memory request is henceforth dependent on the status of address, *i.e.*, requested address is already fetched in sense amplifier or not. Requests generated for main memory fall into following categories:

1) Row Hit: Row hit is a scenario in which requested memory address is present in row buffer. This condition is called open row buffer. In this case for serving requested memory request only CAS memory command is required to be issued and it request minimum number of operations to serve so.

2) Row Closed: In this case sense amplifier is closed, i.e., no address is present in sense amplifier. So, first requested address is fetched to sense amplifier by issuing activate command then CAS is issued.

3) Row Conflict: In this case requested address is not present in sense amplifier. Some other address is already present in row buffer.

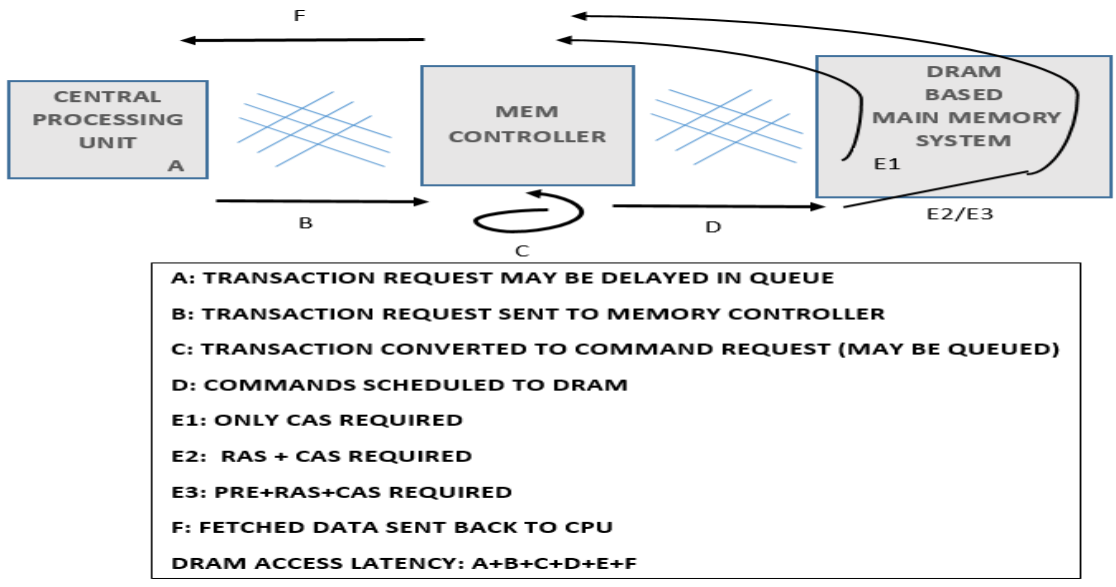

Figure 2.7 DRAM Latency from Processor's Perspective

Figure 2.7, represents comprehensive DRAM latency from processor's outlook. Along with afore mentioned latencies additional recess induces relaying of whole cache line from the memory bank, else to the memory bank of the main memory data bus.

2.2.5 *Row Buffer Management*

Each bank has a row of sense amplifiers called row buffer to store the row being addressed. If new memory request addresses a row present in row buffer, it results in row buffer hit. Row buffer hit consumes least amount of time and energy to serve a memory request. Row buffer miss a condition in which memory request addresses a separate row other than present in row buffer. If row present in row buffer is not further requested by memory requests in near future then it is better to close row buffer. Closing row buffer facilitates more efficient service of future memory requests. Many row buffer management policies [44-48] are investigated that are used in memory controllers. An efficient row buffer management affects the performance and energy consumption of main memory sub-system in a significant way. Hence, it becomes necessary to effectively handle row buffer.

2.2.6 *Address Mapping*

Intelligent address mapping process is responsible for affecting the performance and energy consumption of memory system. Address mapping process affects the level of parallelism and row hits achievable in a memory system. Several papers [41, 47, 49], have investigated the impact of address mapping schemes on performance of memory system. Address mapping translates an address obtained from system's memory address space onto logical DRAM organisation, *i.e.*, address mapping process maps the request address into a set of <channel (L), rank (K), bank (B), row (R), column (C)> which specifies the location of data being addressed. Address mapping process determines which part of address bits are used to address which logical component of DRAM organisation and this choice greatly affects the performance as well as energy consumed by memory sub-system. For example, two requests trying to access different rows of the same bank results in increased request serving latency as we need to first service the first request then this row is required to be closed in order to service second request. As per the values provided in Table 2.1, it would consume 38 DRAM cycles to serve both memory requests, provided the bank was in precharge state initially. 38 DRAM cycles consumed in following manner. First request would consume $t_{REQ1} = t_{RCD} + t_{CAS}$ cycles for execution. It is followed by PRECHARGE command to close the row, PRECHARGE command can be issued t_{RAS} cycles after ACTIVATE command. t_{RAS} value is 5, which is

greater than t_{REQ1}. Thus, it would be t_{RAS} + t_{RP} cycles before issuing next ACTIVATE command. So, in total t_{RAS} + t_{RP} + t_{RCD} + t_{CAS} = 38 cycles are required for execution of both memory requests. Whereas, if both requests were addressing to the same row but different columns of memory address then only 33 cycles are required to service both. They can be serviced one after another serially, so require t_{RCD} + 2*t_{CAS} = 33 cycles. In this case energy consumption as well as latency, both are decreased.

Another option for memory access pattern would be to request different banks. In this case both requests can be serviced in parallel hence require t_{RCD} + t_{CAS} =22 cycles for issuing ACTIVATE commands, although in order to satisfy t_{RCD} timing parameter, second ACTIVATE is required to be delayed by five cycles. Thus, total execution time to execute both memory requests would be 27 cycles. This mapping policy would consume more energy as both banks are active, *i.e.*, two row buffers are activated rather than single to service memory requests. In order to achieve more parallelism, accessing two different channels would be even more beneficial but it will consume more energy. All the above discussed scenarios impact the performance but only in accordance with the stream of addresses. In general, applications leverage spatial locality while issuing memory requests, *i.e.*, memory addresses with difference in lower order bits of the addresses are expected to lie in shorter time frame. This makes us to use lower order bits to address column values to address channel, rank or bank values. If lower order bits address column values then it would lead to more row buffer hits and if used to address channels, ranks and banks then it provides benefits of increased parallelism. Increased parallelism using lower order bits result in performance degradation in long term because of increased need of opening and closing the rows and further frequent change in rank and channels increase energy consumption.

2.2.7 *DRAM Refresh Management*

DRAM cell stores data in the form of charge on capacitor. The capacitor tends to loose charge hence needs to be refreshed periodically in order to retain stored data. These periodic recharge (refresh) operations imposes penalty in terms of increased power consumption and decreased performance. When a memory bank undergoes refresh operation, it stalls servicing read and write requests intended to same bank undergoing refresh operation henceforth resulting in increased memory request service latency. In addition refresh operations consume energy in terms of reading data and restoring it while performing refreshes. As per study conducted in [50], in 32Gb DRAM device 20% of the DRAM's energy is consumed in terms of refresh energy and refreshes degrades system's performance by more than 30%. In SDRAM two

refresh modes, auto-refresh and self-refresh are used to reform refresh operation. The refresh operation should be performed within DRAM cell's retention time. Retention time is the time interval for which DRAM cell can retain its stored data without being again recharged. Researches have been conducted to improve DRAM's retention time and retention failures [51-54].

2.2.8 *Basics on DRAM Current Parameters*

In this section, a brief description about different DRAM currents and insight about their measurement settings is provided. Detailed insight about DRAM current measures is provided in [55].

1. I_{DD0}: I_{DD0} represents One Bank Active-Precharge Current. It is evaluated across activate and precharge commands in respect of one bank while other banks are maintained in precharged state.
2. I_{DD1}: I_{DD1} stands for One Bank Active-Read-Precharge Current and is calculated over the activate, column-read and precharge commands with respect to one bank. Rest banks are in closed state.
3. I_{DD2N}: I_{DD2N} depicts Precharge Standby Current. It is calculated when complete banks are in the precharged state.
4. I_{DD2P0}: I_{DD2P0} renders to Precharge Power-Down Current-Slow-Exit. It is calculated in power-down mode at the time when Clock Enable is at Low state and the DLL is off, in this phase external clock is kept On and complete banks are in precharged state.
5. I_{DD2P1}: It represents Precharge Power-Down Current-Fast-Exit which is measured during power-down mode when Clock Enable is at Low state, DLL is on and external clock is On while complete banks are kept closed.
6. I_{DD3N}: I_{DD3N} corresponds to Active Standby Current that is calculated when at least one bank is in active state.
7. I_{DD3P}: Active Power-Down Current is evaluated for the power-down mode with Clock Enable Low and the DLL locked, when the external clock is active and minimal one bank is in active state.
8. I_{DD4R}: I_{DD4R} represents Burst Read Current and is evaluated while performing Read operation, during its measurement all banks are active performing seamless read data burst along with all data bits switching between the bursts and column read commands are considered to be driving across all the banks.

9. I_{DD4W}: Burst Read Current is evaluated while performing write operation, while performing seamless write data burst with all data bits toggling between bursts. All banks are in active state, with the column write commands driving across complete banks and the ODT (On Die Termination) steady at HIGH.
10. I_{DD5}: I_{DD5} represents Refresh Current that is evaluated while performing refresh process. Commands to schedule Refresh are imposed in each nRFC cycles.
11. I_{DD6}: Self Refresh Current is evaluated during self-refresh mode when the clock enable at Low state and DLL Off and reset. In addition external clock is kept Off and all banks are in precharged state.
12. I_{DD1W}: One Bank Active-Write-Precharge Current is not a JEDEC benchmark. But, its reference measures may be calculated by replacing I_{DD4W} instead of I_{DD4R} in I_{DD1} current and represents activation-write-precharge current.

For a MICRON 512MB Dual Data Rate (DDR3-800) Dual In Memory Module current measurements are shown in Table 2.2.

Table 2.2 Current Measures for DDR3

Current	Measure (mA)
I_{DDO}	360
I_{DD1R}	440
I_{DD1W}	410
I_{DD2N}	180
I_{DD2P0}	40
I_{DD2P1}	100
I_{DD3N}	200
I_{DD3P}	100
I_{DD4R}	840
I_{DD4W}	840
I_{DD5}	800
I_{DD6}	24

2.2.9 *DRAM Power Model*

In this section, we describe the power model used to calculate memory system power consumption. DRAM power consumption can be factored into two components consumed by

memory elements (core power consumption) [56], and power consumed while driving data into or out of the data bus (I/O power consumption). Power consumed by memory elements, *i.e.*, core power consumption comprised of three main elements, i) Average power consumption when memory is in idle state (base power consumption), is the summation of power consumed in standby mode and during refresh operation ii) Power consumption when DRAM is active (active power consumption) and Power consumption while servicing read/write requests. The equations used for power modeling are based on Micron Memory System Power Technical Note [57] and Micron power calculator [58]. For better assimilation, $P_{(XX)}$ is used to denote power consumed by XX sub-component. Total power consumed by DRAM chip is calculated as

$$P_{chip\_power} = P_{(read)} + P_{(write)} + P_{(referesh)} + P_{(activate)} + P_{(background)} + P_{(terminate)} \quad (2.1)$$

$$P_{total\_dram\_power} = P_{chip\_power} * N_{DRAM\_chips} \quad (2.2)$$

Where, N_{DRAM_chips}, represents total number of DRAM chips available in memory system and $P_{(read)}$, power consumed in read operation, $P_{(write)}$, represents power consumed in write operation, $P_{(referesh)}$, depicts power consumed during refresh, $P_{(background)}$, represents power consumed in background processes and $P_{(terminate)}$, represents termination power.

$$P_{(read)} = (I_{DD4R} - I_{DD3N}) * V_{DD} * N_{readcycles} \quad (2.3)$$

$$P_{(write)} = (I_{DD4W} - I_{DD3N}) * V_{DD} * N_{writecycles} \quad (2.4)$$

$$P_{(referesh)} = (I_{DD5} - I_{DD3N}) * V_{DD} * {}^{T_{RFC}}/_{T_{RFEI}} \quad (2.5)$$

$$P_{(activate)} = P_{(maxactivate)} * {}^{T_{RC}}/_{(avg.\,gap\ between\ activates)} \quad (2.6)$$

$$P_{(maxactivate)} = (I_{DD0} - {}^{(I_{DD3N} * T_{RAS} + I_{DD2N} * (T_{RC} - T_{RAS}))}/_{T_{RC}}) * V_{DD} \quad (2.7)$$

$P_{background}$, represents background power dissipation and is combination of following components

$$P_{act\_pdn} = I_{DD3P} * V_{DD} * T_{act\_pdn} \quad (2.8)$$

$$P_{act\_stdby} = I_{DD3N} * V_{DD} * T_{act\_stdby} \quad (2.9)$$

$$P_{pre\_dwn\_slow} = I_{DD2P0} * V_{DD} * T_{pre\_dwn\_slow} \quad (2.10)$$

$$P_{pre\_dwn\_fast} = I_{DD2P1} * V_{DD} * T_{pre\_dwn\_fast} \quad (2.11)$$

$$P_{pre\_stby} = I_{DD2N} * V_{DD} * T_{pre\_stby} \quad (2.12)$$

$$P_{background} = P_{act\_pdn} + P_{act\_stdby} + P_{pre\_dwn\_slow} + P_{pre\_dwn\_fast} + P_{pre\_stby} \quad (2.13)$$

Power dissipated in ODT resistors constitutes termination power. For termination power consumption not only active rank is responsible but also other ranks in same channel participate in same.

$$P_{read\_terminate} = pds_{rd} * N_{data\_read} \quad (2.14)$$

$$P_{write\_terminate} = pds_{wr} * N_{data\_write} \quad (2.15)$$

$$P_{read\_terminate\_other} = pds_{termRoth} * N_{data\_read\_oth} \quad (2.16)$$

$$P_{write\_terminate\_other} = pds_{termWoth} * N_{data\_write\_oth} \quad (2.17)$$

The values for pds_{rd}, pds_{wr}, $pds_{termRoth}$, $pds_{termWoth}$ are taken as per micron technical note[57].

$$P_{total\_chip\_power} = P_{(read)} + P_{(write)} + P_{(referesh)} + P_{(activate)} + P_{(background)} + P_{(terminate)} \quad (2.18)$$

2.2.10 *Memory Controller*

As mentioned earlier, the main memory system is comprised of storage devices and a memory controller. In order to hide hardware details, memory controller plays a role of translator and control wrapper for storage devices. Memory controller receives memory requests and as per the received request, issues commands to DRAM devices. Memory controller can be categorized into two layers: the outer layer and inner layer. Outer layer communicates with other computer components and inner layer controls the DRAM devices. Memory scheduler is that part of memory controller which issues DRAM commands as per the selected memory request. The outer layer accepts memory requests and enqueue them in single transaction queue (TQ). Often write requests and their data are kept in separate write queue (WQ). Requests placed in transaction queue and write queue are mapped into a series of DRAM commands and then these are handled to inner layer. Simple way to translate incoming request is to map them into a PRECHARGE command, ACTIVATE command and a WRITE or READ command. After translation of incoming request, these commands are placed in inner layer's command

queue (CQ). The memory commands translated are issued as per the timing constraints. REFRESH commands are also issued at periodic intervals to retain the data stored.

2.3 Memory Access Scheduling

Memory scheduling policies were developed to enhance the performance and decrease energy consumption of superscalar, multicore and multithreaded processors. Transition from unicore to multicore processor resulted in change in behaviour of memory scheduling policies also. From the role of reordering and scheduling requests issued from same thread, its behaviour changed to schedule requests from different threads for better and efficient resource utilization. In the following sections we will discuss different memory access scheduling policies.

2.3.1 *Memory Scheduling Policies for Single-Threaded Single Core Processor*

Memory access scheduling policies in single threaded single-core processors mainly focus on re-ordering memory access requests in order to reduce gap between processor and memory latency. Researchers in [31], have proposed a memory access scheduling policy in order to re-order DRAM commands like bank PRECHARGE, ACTIVATE and Column Access Strobe.

In [59], authors have focused on designing parallelized memory controller by introducing memory access scheduler which is responsible for issuing read requests, write requests, ACTIVATE and PRECHARGE commands. The Ph.D. thesis [60], have proposed a compiler based technology called access ordering for solving memory bandwidth issue in scalar processor by reordering memory requests. Authors in [61], investigated memory access ordering to find the limitations for performance enhancement. In [62], researchers introduced a memory scheduling unit to prefetch memory read requests, buffer memory write accesses, and dynamically reordering memory accesses to maximize efficient memory bandwidth utilization. The key limitation of before discussed algorithms is that they are beneficial for single-threaded processors only. They cannot handle requests from multiple threads. Our main focus in this thesis is on memory access schedulers for multi-threaded and multicore processors. As per [63], there are some scheduling policies that were developed for single threaded processors but can be used in SMT processors. Memory access scheduling policies under this category are FCFS (First Come First Serve), age-based policy, hit-first scheduling policy and read-first scheduling policy.

FCFS scheduling policy serves requests in accordance with their arrival time. Request that arrives first is served first, irrespective of all other factors. It is very simple hence requires

very less hardware for implementation. Its limitation also lies in being very simple. It does not consider the criticality of other resources and requests. Hit-first scheduling policy prioritizes row-buffer hits over row-buffer miss requests. So, it prioritizes memory requests that take less time to complete. Read-first scheduling policy gives more priority to memory reads over writes because memory reads are more critical for system's performance than memory writes. Both hit-first and read-first scheduling policies can be used along with other scheduling algorithms.

Next section contains memory access scheduling policies that are developed for multi-threaded and multi-core processor.

2.3.2 *Scheduling Policies for Multi-threaded and Multi-core processors:*

The idea of main memory scheduling policy for multi-threaded processors is discussed in [63]. In multithreaded and multi-core processors multiple threads run simultaneously and hence contention to access system resources (memory resource) among threads also increases. In [63], researchers presented three thread-aware scheduling policies. Introduced scheduling policies are request-based, reorder buffer-based and IQ-based (Issue Queue-based) scheduling policies. ROB-based and IQ-based scheduling policies are resource based scheduling policies and request-based scheduling algorithm is request based scheduling policy. They compared the performance of these scheduling strategies with algorithms like hit-first, read-first and age-first.

Researchers categorized memory access scheduling algorithms in two categories, resource-based algorithms and request-based algorithms. Two more categories are added for classification of memory scheduling policies i.e., fairness-based algorithms and parallelism-based algorithm.

Resource-Based Algorithms: The key concept behind these scheduling policies is that they try to decrease conflict among threads for accessing main memory. The contention among threads serves as a bottleneck in system's performance. Three resource based scheduling policies are discussed here. i.e., ROB-based policy, IQ-based policy and scheduling policy using RIR (Read-to In-flight Ratio) metric. ROB-based memory access scheduling policy prioritizes thread having maximum entries in reorder buffer. The key idea behind doing so is that serving request from threads having more re-order buffer entry releases more waiting instructions than serving requests from other threads. This scheduling policy decreases contention on reorder buffer, thus improves throughput of the system.

IQ-based scheduling policy prioritizes requests from the thread having highest amount of issue queue entries. The idea behind prioritizing requests from threads having highest number of issue queue entry is that it is more advantageous in terms of performance to serve these requests than to serve requests from other threads. It helps to improve throughput and decrease contention among threads on issue queue.

In [64], researchers have presented a new performance metric called RIR. RIR is the fraction among the amount of ready instructions in the issue queues and the amount of in-flight instructions from issue to write back stages. High amount of RIR means the thread has made good progress during execution with whatever resources were available with it. Researchers utilized this metric in phase co-scheduling for a dual-core chip multiprocessor of dual-threaded SMT processors. Using this metric gives good results in memory access scheduling.

Request-based Scheduling Policy: Request-based scheduling strategies prioritize requests from thread having minimum amount of pending requests that is why sometimes it is called as LREQ (Least REQuest). This algorithm improves throughput of the memory system but does not affect fairness. As the number of threads during execution increases, performance of request-based scheduling policy decreases. Figure 2.7, reveals the simulation trends obtained for FCFS, hit-first, age-based, ROB-based, IQ-based and LREQ based scheduling policies as per assumptions and simulation setup considered in [63].

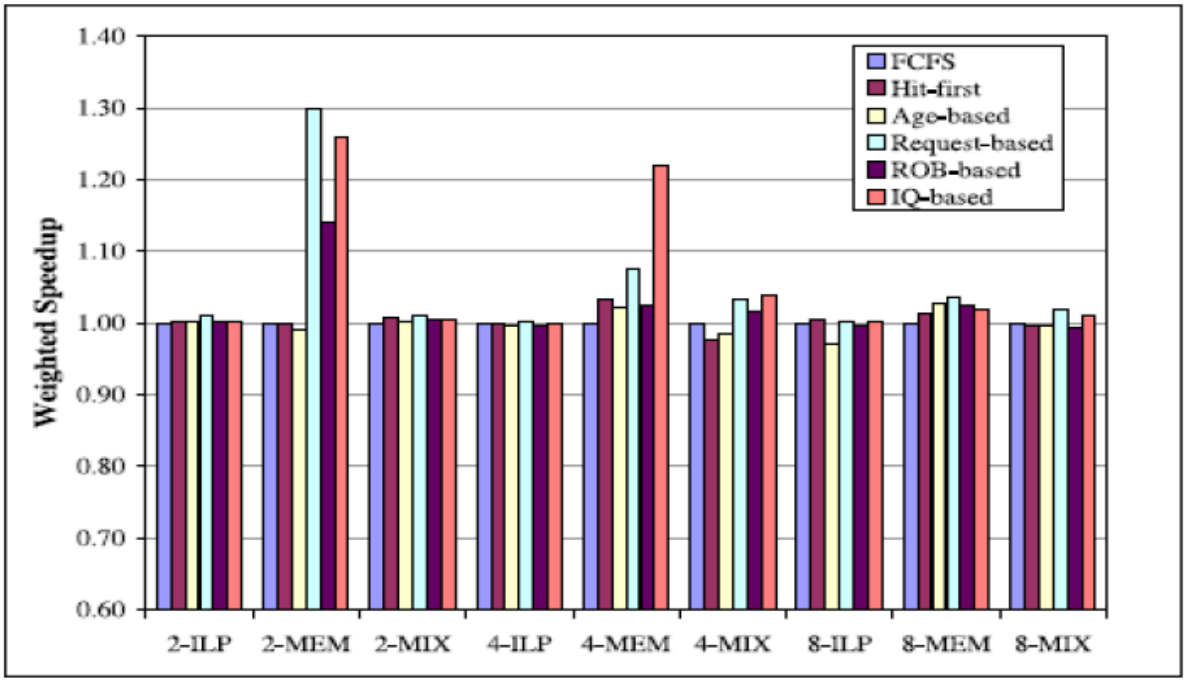

Figure 2.8 Comparison between Scheduling Policies[63]

In [65], authors proposed a new scheduling strategy named ME-LREQ (Memory Efficiency with Least REQuest). It is request based scheduler but researchers further attached a new parameter named memory efficiency to it. Memory efficiency of an application is narrated as Instructions per clock (IPC) of this application divided by memory bandwidth usage under single-core environment, given by equation 2.19.

$$ME[i] = \frac{IPC_{single}[i]}{BW_{single}[i]} \tag{2.19}$$

In [65], as per results obtained, proposed scheduling policy improved performance by 6.4% on average and upto 9.2% as compared to original request-based scheduling policy. The limitation of this scheduling policy is that it does not support both online and offline profiling.

Fairness-Based Algorithms: Fairness is an important aspect in system's performance and energy consumption that is being addressed by only few algorithms.

For single threaded processors age-based algorithms were proposed. Same is still applicable for multi-threaded and multicore processors. Other than age-based scheduling policy there are other fairness-based scheduling algorithms proposed for multi-threaded multicore processor environment. One among such scheduling policies is Round Robin (RR) scheduling policy. Round Robin scheduling policies iteratively issues pending memory requests from threads, one at a time from each thread in one iteration. Round Robin scheduling policy can be presented by following equation:

$$ThreadID_{next\_request} = \frac{ThreadID_{last\_served\_request}+1}{Total\ no.of\ threads\ having\ pending\ requests} \tag{2.20}$$

Round Robin scheduling strategy aims to improve fairness among threads in slightly different manner than age-based scheduling policy. Age-based scheduling strategy aims to achieve fairness among requests irrespective of threads that issued them. That is, if at a time I, thread X issued three memory requests and at time i+10, thread Y issued one memory request, the requests generated by thread X will be served first being older than requests generated by thread Y. Whereas, RR scheduling policy does not consider how old an request is. For above mentioned scenario, in case of RR scheduling policy, first a request from thread X is served than one from thread Y, iteratively. In other words, it can be said that age based memory scheduling policy tries to achieve fairness at memory request level, whereas RR scheduling policy tries to achieve fairness at thread level.

In [42], authors proposed another fairness-based memory scheduling policy named STFM (Stall Time Fair Memory scheduler). This scheduling policy primarily focuses on improving fairness. The scheduling policy estimates stall-time T_{shared} and T_{alone}. T_{shared} represents memory waiting time observed by the thread when executing along with multiple threads in the memory system. T_{alone} is the wait time experience by the thread if it had running alone in the memory system. Using the values of T_{shared} and T_{alone}, the researchers in [42] calculated slowdown for each thread. Then the maximum slowdown and minimum slowdown corresponding to each pending request is calculated. The requests having ratio between maximum and minimum slowdown and above certain threshold are scheduled first to decrease stall time experiences by the threads. If this ratio is below threshold then simple FR-FCFS scheduling policy is used to schedule memory requests.

Parallelism-based Scheduling Algorithm: Schedulers under this category focus on enhancing parallelism among main memory sub-system components or among applications themselves. Authors in [66], proposed a memory access scheduling policy named PAR-BS (parallelism-aware Batch Scheduling). PAR-BS is primarily based on two key ideas, one is batch scheduling, *i.e.*, to schedule memory requests in batches in accordance to their arrival time and corresponding requests belonging to oldest batch have highest priority in order to avoid starvation. Another scheduling idea is parallelism aware scheduling, *i.e.*, to facilitate bank level parallelism with a batch.

A scheduler named ATLAS proposed in [67], prioritized threads that are least served in past by memory controllers. The key limitation of ATLAS is that it provides good results in terms of performance and fairness if memory system have multiple memory controllers. TCM scheduler proposed in [68], first categorised threads in two groups, i.e., memory-intensive threads and memory non-intensive (compute intensive) threads. TCM prioritize threads belonging to compute intensive category over memory intensive ones because they are light and easy to service, henceforth resulting in improved performance. In order to exploit fairness among threads TCM shuffles priority of threads under memory-intensive category.

Refresh based Scheduling Policies: DRAM based memory system must incorporate efficient and intelligent refresh mechanism that should be able to decide: 1) timing regarding scheduling of refresh operations while meeting timing constraints of additional memory commands, and 2) which DRAM rows are to be refreshed. In this section further we briefly discuss approaches

for deciding when to issue refresh commands while meeting timing parameters of other memory commands.

Decision Regarding When to issue Refresh Command

Classification of Refresh scheduler is based on how they issue refresh operations [69] and deciding whether they issue regular accesses around [70] or within timing constraints [35]. In [69] Stuecheli et al. propose refresh mechanism named Elastic Refresh [69] that dynamically fits refresh period according to currently executing workload. In [70], authors propose Dynamic Command Expansion (DCE) and Pre-emptive Command Drain (PCD) that prevent the memory controller queue from halting useful memory accesses if the memory controller queue is filled with memory commands addressed to the bank to be refreshed. At first, DCE delays commands to the banks that are to be banks refreshed, and then proactively issues commands to the banks under refresh operation. In [35], Nair et al. proposed refresh pause operation allowing regular accesses to operate with less delay.

Decision Regarding What Not to Refresh

Another basis for categorization of refresh schemes is on the basis of refresh data on which refresh operation is to be performed. It is based on cell retention time, error tolerance of the data, access recency, and validity of row, *i.e.*, validity of row includes decision regarding whether valid row is under refresh operation. A row is said to be valid if operating system has allocated the physical pages containing those rows.

Cell Retention Time: Retention time refers to the duration for which DRAM cell maintains data integrity, *i.e.*, retaining data without changing or disrupting the stored value on it as capacitor tends to leak stored data gradually with time. The retention time of cells varies from cell to cell across the chip due to process variation [71, 72]. The scheduling approaches that make use of retention time information do not consider system workloads and global memory usage. On hardware level, retention aware approach may refresh cells with longer retention time less frequently and cells with less retention time more frequently, as used in the Variable Refresh Architecture (VRA) proposed by Ohsawa et al. in [73] and in Retention-Aware Intelligent DRAM Refresh (RAIDR) approach proposed by Liu et al. in [74]. In VRA approach, each row's expected refresh period is maintained in registers inside DRAM based memory system. Retention Aware Approach makes use of fact that only a few rows require very frequent refreshes and such rows are tracked inside memory controller. At software level refresh period of cells can be improved by allocating addresses that refers to cells with sufficient retention time. In [75], authors proposed the Retention-Aware Placement approach in DRAM (RAPID) and Refresh Incessantly but Occasionally (RIO) policy is proposed by

Baek et al. [76]. These two before mentioned approaches are two such Retention based solutions that work on minimizing device's refresh rate by isolating pages requiring frequent refreshes.

Error Tolerance: Some applications like machine learning, media processing and unstructured information analysis can tolerate shortcomings in some of their data and still they produce acceptable outcomes. Approximate computation [77, 78] makes use of such approximate data to find out the tradeoffs among performance, energy, and accuracy. So, DRAM cells storing such error-tolerant data are not required to be refreshed as frequently as cells storing critical data. The number of cells falling into fault-tolerant category depends on application's characteristics. Liu et al. [79] proposed approach that partitions DRAM banks into two regions, *i.e.*, critical and noncritical regions. The proposed approach, Flikker, extends the self-refresh time in order to refresh non-critical regions less frequently. Proposed solution focuses on smartphones that keeps DRAM in self-refresh mode when in idle state, similar technique can be applied to auto refresh mode in operating mode. DRAM must be repartitioned if workload characteristics change in such a way that more data becomes critical. Such partitioning is coarse grained and it simplifies hardware and reduces area overhead.

***Access Recency*:** DRAM accesses recharge data stored in cell so subsequent refresh operation to same row is not required and may be postponed. The amount of rows influenced depends on how many various rows are retrieved within the maximum refresh period, which may be few for many workloads. Ghosh et al. [80] presented Smart Refresh approach that maintains per-row timeout counter in memory controller and divides the refresh period into phases. During each phase the memory controller decrements the counter and when counter reaches zero value it issues RAS-only refresh. In [81], Emma et al. proposed cleverer refresh policies for embedded DRAM caches. ECC ensures error tolerance and timestamp guide scheduling selective refreshes. In [82] Agrawal et al. similarly work on eDRAM caches with Refrint. Authors incorporated eager writeback policy for rarely used lines additionally with maintaining access recency. Limited rows are there in eDRAM so overhead for tracking an information in eDRAM is much more bearable than information tracking overhead for DRAM based main memory.

***Validity*:** If operating system does not allocate memory addresses then refreshes made to corresponding addresses are wasteful. Many researchers work on software approaches that attempt to initiate refreshes only for memory addresses with valid data. The policies based on validity of data is sensitive to the total memory usage of the computing system. In addition to variable retention aware policy, Ohsawa et al. in [73] presented a Selective Refresh

Architecture (SRA) approach that uses an A bit per row and accordingly decide whether to refresh it or not. In this scheme modification to ISA is made so that the compiler, operating system, or memory controller can prevent refreshes to invalid data. Isen and John [83], proposed a combination of hardware/software approach ESKIMO that make use of SRA for tracking data significance. For an instance, in newly allocated memory addresses the values of uninitialized data are insignificant. The information regarding allocation and de-allocation of virtual addresses is maintained with operating system. Baek et al. [76] propose Placement-Aware Refresh In situ (PARIS) that uses physical memory usage information maintained at operating system instead of virtual addresses. PARIS maintains RD bits in the memory controller. The storage overheads is reduced by tracking valid bits for coarse row granularities. However, using coarse row granularity for maintaining valid bits increases unnecessary refreshes.

CHAPTER 3

DRAM SCHEDULER OPTIMIZED FOR ROW BUFFER HITS AND FAIRNESS AMONG THREADS

3.1 Impact of Row Buffer Hits on DRAM Performance

Row buffer hit is condition in which the address being requested by memory access is already present in the sense amplifier. In this case minimum number of operations are required to be performed to service a memory request. How prioritizing row buffer hits affects the behaviour of DRAM in terms of energy consumption and performance is evaluated first to identify its role in efficient memory scheduling. The dynamic energy consumed to perform column read command (to read data from cell) on DRAM memory cell is given by

$$E_{RD} = (I_{DD4R} - I_{DD3N}) * V * T_{data} \tag{3.1}$$

where: I_{DD4R} denotes current withdrawn to perform column read and I_{DD3N} corresponds to current withdrawn in active standby mode. Time taken to transfer data in M column accesses is represented by T_{data} and is given by (3.2).

$$T_{data} = M * T_{burst} \tag{3.2}$$

T_{burst} represents data transfer latency and is given by (3.3)

$$T_{burst} = BL * {t_{clk}}/{2} \tag{3.3}$$

Along with E_{RD} additional dynamic energy (E_{DQ}) is also expended to read data out from DRAM cell, given by (3.4).

$$E_{DQ} = P_{DQ(R)} * (N_{DQ} + N_{DQS}) * T_{data} \tag{3.4}$$

where, $P_{DQ(R)}$ represents power consumed per pin while extracting output [84]. $N_{DQ(R)}$ denotes number of data pins and N_{DQS} corresponds to number of strobe pins.

When writing data into DRAM cell, dynamic energy E_{WR} is expanded and is given by (85).

$$E_{WR} = (I_{DD4W} - I_{DD3N}) * V * T_{data} \tag{3.5}$$

where, I_{DD4W} and I_{DD3N} represents write current drawn and stand by current drawn during T_{data}.

While writing data, write termination energy is also spent to writes, (3.6).

$$E_{term} = P_{DQ(W)}(N_{DQ} + N_{DQS} + N_{DM}) * T_{data} \quad (3.6)$$

In equation (6), N_{DM} represents number of data mask pins and $N_{DQ(M)}$ denotes power per pin during write termination.

Equation (3.7) and equation (3.8), gives dynamic energy consumption during read miss and write miss.

$$E_{DRAM(readmiss)} = E_{DD0} + E_{RD} + E_{DQ} \quad (3.7)$$

$$E_{DRAM(writemiss)} = E_{DD0} + E_{WR} + E_{term} \quad (3.8)$$

where,

$$E_{DD0} = (I_{DD0} - {}^{1}\!/_{t_{rc}} (I_{DD3N} + t_{RAS} + I_{DD2N} * (t_{rc} - t_{RAS}) * V * t_{rc})) \quad (3.9)$$

where, I_{DD0} is average current drawn during issuing activate command. t_{rc} is delay between two activate command. After delay of t_{ras} activate command is preceded by precharge command.

Dynamic energy spent in row hit situation, *i.e.*, read hit and write hit, is in the form of dynamic energy consumed to perform read column access and write column access, respectively.

Dynamic energy consumed in read hit access and write hit access is given by (3.10) and (3.11)

$$E_{DRAM(readhit)} = E_{RD} + E_{DQ} \quad (3.10)$$

$$E_{DRAM(writehit)} = E_{WR} + E_{term} \quad (3.11)$$

By analysing equation (3.7), (3.8) and (3.10), (3.11), it is clearly revealed that row buffer hits require lesser number of operations to access the desired page.

3.1.1 *Motivational Results*

We start by analysing the impact of row buffer hits on DRAM performance. For conducting such analysis we simulated and compared existing memory scheduling policies that are using this feature with ones that do not take advantage of row buffer hits. We simulated chosen scheduling policies on cycle accurate DRAM's main memory system simulator, USIMM [86]. In USIMM, memory controller issues device level memory commands and this decision is dependent on present status of channel(s), rank(s) and bank(s) of main memory system. In order

to conduct comparative evaluation we simulated workloads using proposed scheduling approach and same workloads using existing scheduling approach under two different memory configuration, *i.e.*, single-channel memory configuration and four-channel memory configuration. In single-channel memory configuration, memory system is formulated with one channel and that channel two ranks are present and in each rank four banks are present. In four-channel memory configuration, memory system is composed of four channels and in each channel two ranks are there and in each rank four banks are present. In simulator power related calculations are performed on the basis of Micron's power calculation methodology. The detailed information regarding power simulation are included in [57].

Table 3.1 Workload Description

Trace File(s)	Workloads with Single-Channel m/m Configuration	Workloads with four-Channel m/m Configuration
comm2	1C_1Chn_1	1C_4Chn_1
comm1 comm1	2C_1Chn_1	2C_4Chn_1
comm1 comm1 comm2 comm2	4C_1Chn_1	4C_4Chn_1
fluid swapt comm2 comm2	4C_1Chn_2	4C_4Chn_2
face face ferret ferret	4C_1Chn_3	4C_4Chn_3
black black freq freq	4C_1Chn_4	4C_4Chn_4
stream stream stream stream	4C_1Chn_5	4C_4Chn_5
fluid fluid swapt swapt comm2 comm2 ferret ferret	-	8C_4Chn_1
fluid fluid swapt swapt comm2 comm2 ferret ferret black black freq freq comm1 comm1 stream stream	-	16C_4Chn_9

Result Analysis

Table 3.2, brings out the outcomes for simulated scheduling policies in respect of row buffer hit rate. The outcomes in Table 3.2, highlights that RLDP scheduling policy stems to highest page hit rate. Subsequently PRWL scheduling algorithm conducted better in terms of overall page hit rate. Execution of close page policy is least amongst all simulated scheduling policies. Amid all simulated policies RLDP and PRWL prefers row buffer hits over other memory read

and write commands, however, FCFS and close page policy does not utilize such feature, therefore the results received for row buffer hits reveals the similar pattern.

Table 3.2 Comparative statement of simulated scheduling policies on the basis of row buffer hit

Workload	Read Hit Rate				Write Hit Rate			
	FCFS	**Close**	**RLDP**	**PRWL**	**FCFS**	**Close**	**RLDP**	**PRWL**
MT-c1	0.0033	-0.0290	0.0144	-0.0356	-0.2097	-0.2099	-0.0345	-0.4018
4C_1Chn_4	0.6291	0.5178	0.5709	0.527	0.1603	0.1508	0.3823	0.1356
2C_1Chn_1	0.5996	0.4846	0.5053	0.4746	-0.1653	-0.2475	0.1673	0.1274
4C_1Chn_1	0.5294	0.4167	0.4728	0.4272	-0.1619	-0.2084	0.1164	-0.2847
1C_1Chn_1	0.5749	0.4605	0.4743	0.4498	-0.2850	-0.2890	0.0854	-0.0189
4C_1Chn_3	0.6996	0.5952	0.6543	0.6011	0.3990	0.3860	0.5761	0.3563
4C_1Chn_2	0.5545	0.4430	0.4982	0.4528	-0.0861	-0.1163	0.1861	-0.1125
4C_1Chn_5	0.6461	0.5340	0.5930	0.5444	0.1837	0.1662	0.3985	0.1452
MTc-4	0.0185	0.0073	0.0065	0.0002	-0.6412	-0.7449	0.0091	-0.0476
4C_4Chn_4	0.0479	0.0057	0.0096	0.0026	-0.4024	-0.4406	0.0129	-0.0300
2C_4Chn_1	0.0595	0.0074	0.0063	0.0038	-0.0707	-0.1321	0.0968	0.0075
4C_4Chn_1	0.0141	0.0041	0.0041	-0.0001	-0.4801	-0.5352	0.0242	-0.0254
1C_4Chn_1	0.0160	0.0026	0.0026	0.0016	-0.0699	-0.0853	0.0040	-0.0013
4C_4Chn_3	0.0638	-0.0038	0.0130	-0.0013	-0.3775	-0.4282	0.0534	-0.0613
4C_4Chn_2	0.0197	0.0025	0.0038	-0.0002	-0.3988	-0.4230	0.0034	-0.0315
4C_4Chn_5	0.0466	0.0043	0.0085	0.0002	-0.3885	-0.4572	0.0097	-0.0595
8C_4Chn_1	0.0074	-0.0058	0.0025	-0.0091	-0.4052	-0.4768	-0.0037	-0.1556
16C_4Chn_9	-0.0140	-0.0222	-0.0025	-0.0270	-0.2988	-0.3438	-0.0169	-0.3256
1-Channel **4-Channel**	0.5296 0.0280	0.4279 0.0002	0.4729 0.0054	0.4301 -0.0029	-0.0206 -0.3533	-0.0460 -0.4067	0.2347 0.0193	-0.0067 -0.0730
Average	0.2509	0.2132	0.2132	0.1896	-0.2054	-0.2464	0.1150	-0.0435

Influence of row buffer hits on Dynamic RAM based main memory system's energy utilization and performance is analysed by evaluating performance metrics like, energy-delay product, total execution time, and maximum slowdown time.

Energy-Delay Product

The results obtained are depicted in Figure 3.1 for EDP reveals that RLDP and PRWL consumed less energy than FCFS and close page policy while maintaining the performance level for all simulated scenarios. In Figure 3, results convey that RLDP rationalized EDP by 17.33% and 12.95% in one-channel memory configuration when compared with FCFS and close page policy, respectively. 14.62% and 10.09% decrease in EDP is achieved when PRWL scheduling policy is compared to FCFS and close policy in single- as well as four- channel memory configurations, respectively.

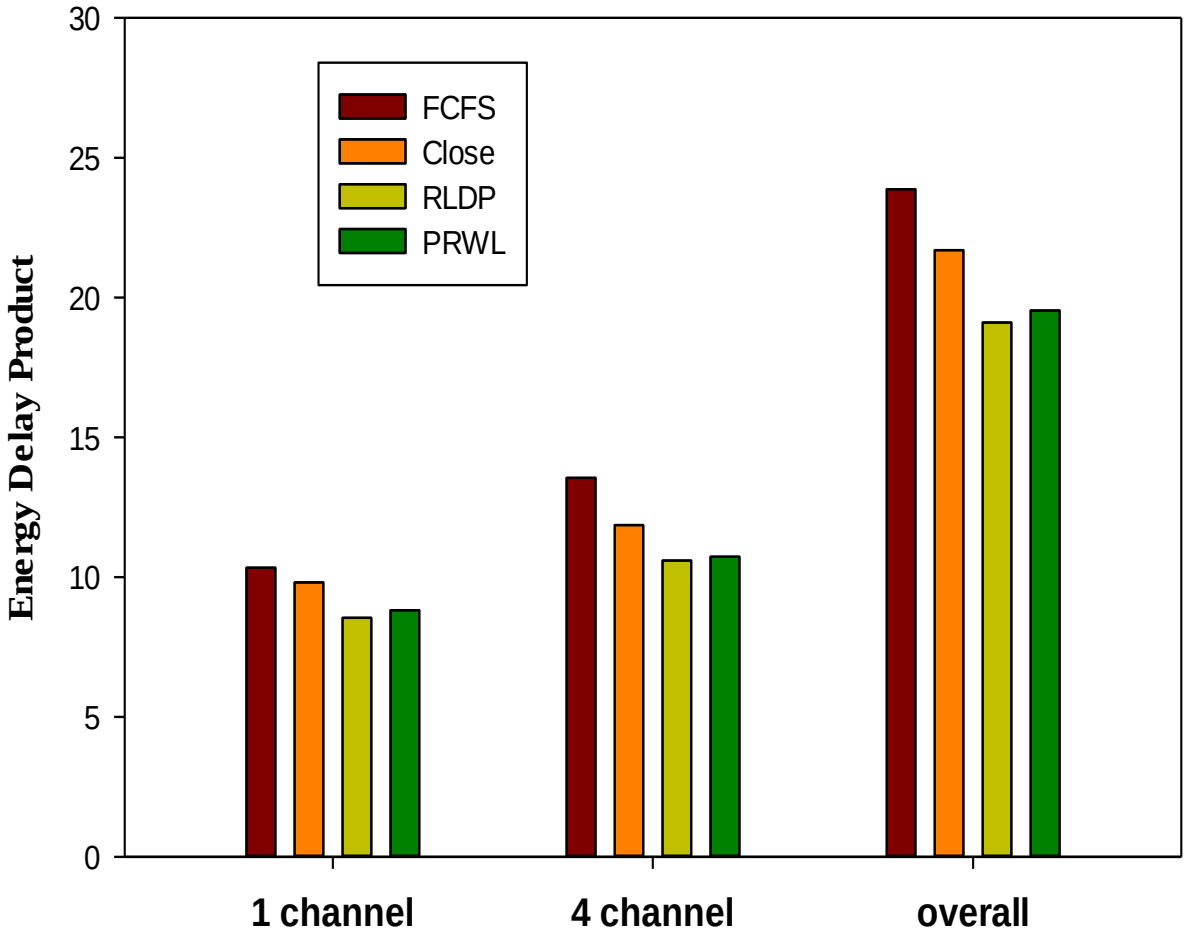

Figure 3.1 EDP (Js) Comparison

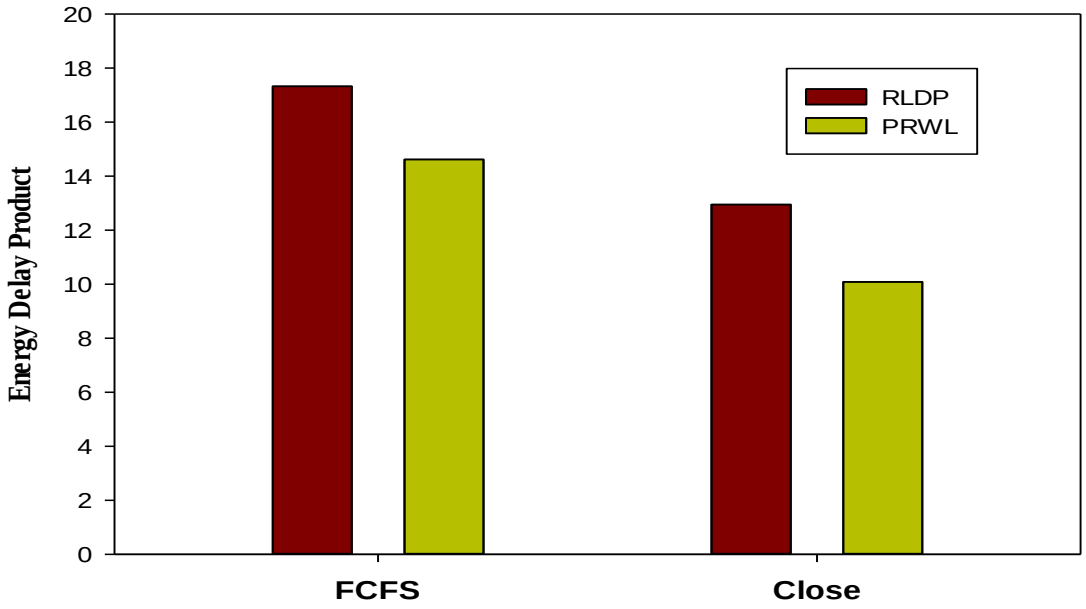

Figure 3.2 % decrease in EDP for 1-channel configuration.

Results obtained after simulation in 4-channel configuration, Figure 3.2 depicts that 21.84% and 10.98% reduction in EDP is observed when RLDP is compared to FCFS and close policy, respectively, in 4-channel configuration. Whereas, PRWL reduced EDP by 20.81% and 9.53%, when compared to FCFS scheduling policy and close page policy, respectively in 4-channel configuration.

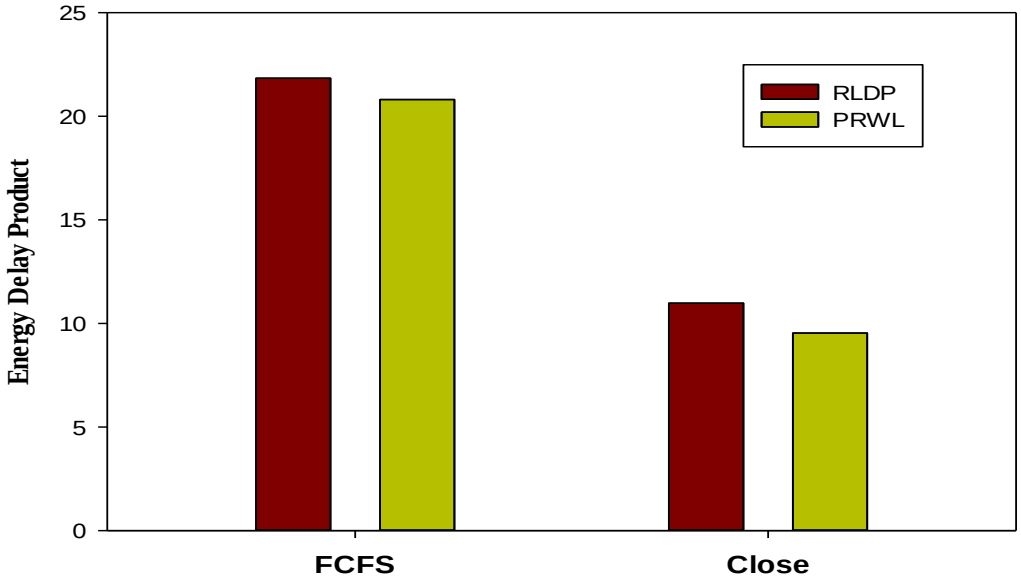

Figure 3.3 % decrease in EDP for 4-channel configuration

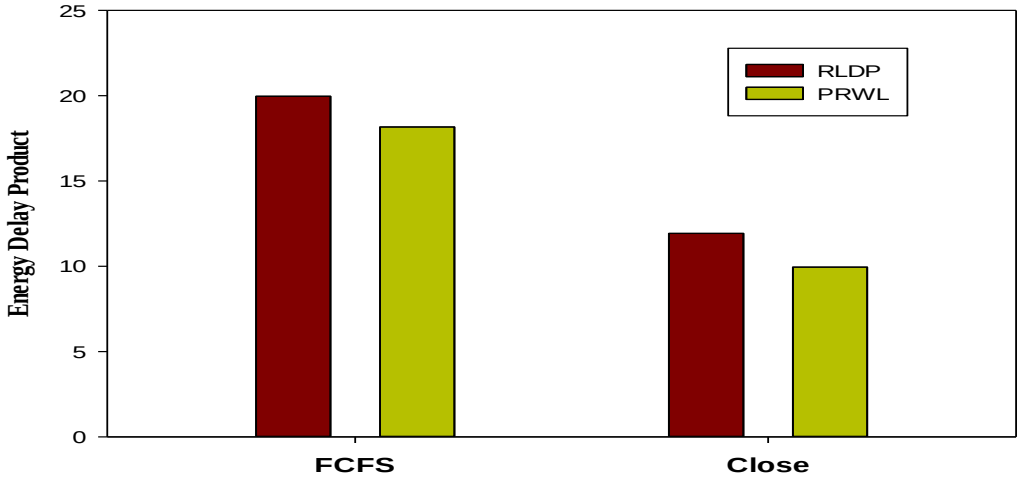

Figure 3.4 % of overall decrease in EDP

In total RLDP has reduced EDP by 19.97% and PRWL has reduced EDP by 18.17% in comparison with FCFS scheduling policy. Similarly, 11.93% and 9.95% total reduction in EDP has been observed when RLDP and PRWL is compared to close page scheduling policy, Figure 3.4.

Total Execution Time

The results shown in Figure 3.5 reveal that the performance of RLDP is best among all simulated scheduling policies for both memory configurations. After RLDP, PRWL performed better than other simulated policies. RLDP took 8.94% and 6.75% less time to complete their execution in comparison to FCFS and close scheduling policy, respectively, in single-channel memory configuration. Whereas, 7.86% and 5.64% reduction in execution time is observed when PRWL is compared to FCFS and close page policy, respectively, Figure 3.6.

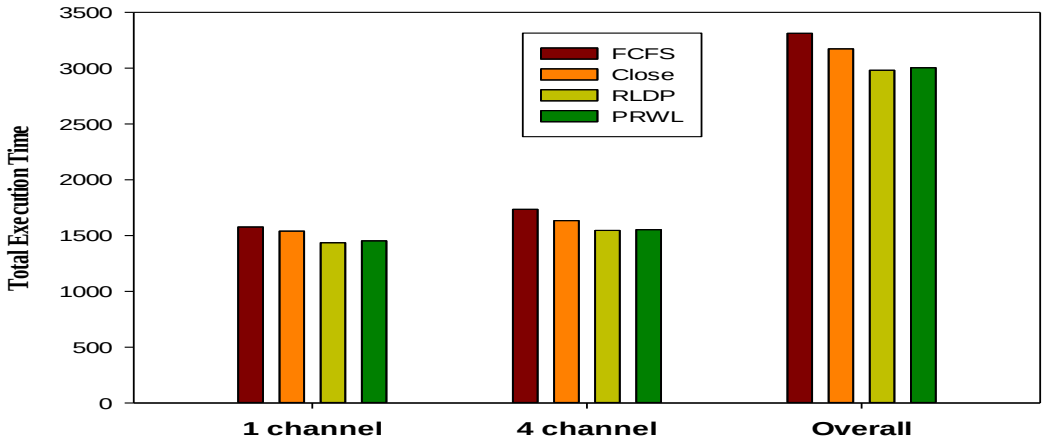

Figure 3.5 Total Execution Time (mCyc) Comparison

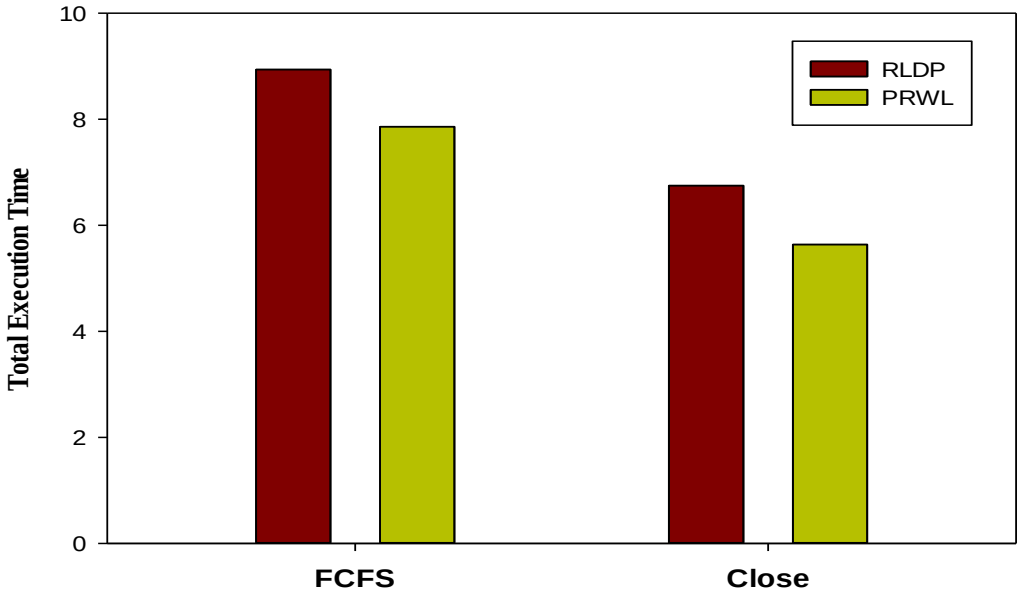

Figure 3.6 % decrease in Total Execution Time for 1-channel configuration

In 4-channel memory configuration 10.89% and 10.49% reduction in execution time is observed when RLDP and PRWL is compared to FCFS scheduling policy, respectively. When RLDP is compared to close, 5.38%, and when PRWL is compared to close policy 4.96% reduction in execution time is observed, Figure 3.7.

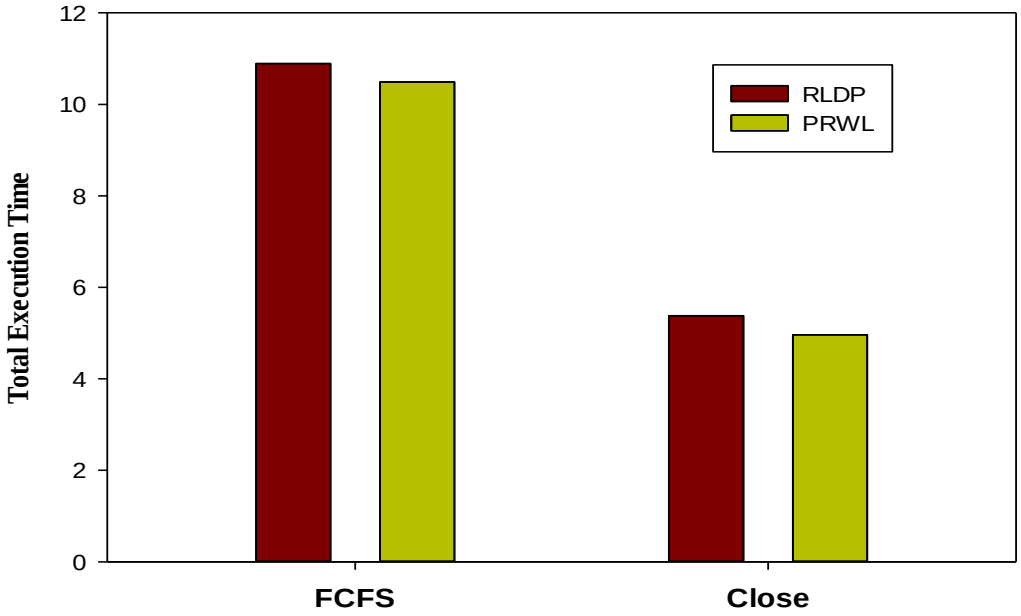

Figure 3.7 % decrease in Total Execution Time for 4-channel configuration

RLDP reduced execution time of workloads by 9.99% and 6.05%, in total when compared to FCFS scheduling policy and close page policy, respectively. When PRWL is compared to FCFS scheduling policy 9.30% reduction in execution time is observed, whereas, with respect to close page policy 5.33% reduction in execution time can be seen, Figure 3.8. Both RLDP and PRWL prioritize row buffer hits which further lead to reduced execution time because of reduced service time required to complete requests made for main memory. Row buffer hits require least number of operations for servicing a memory request.

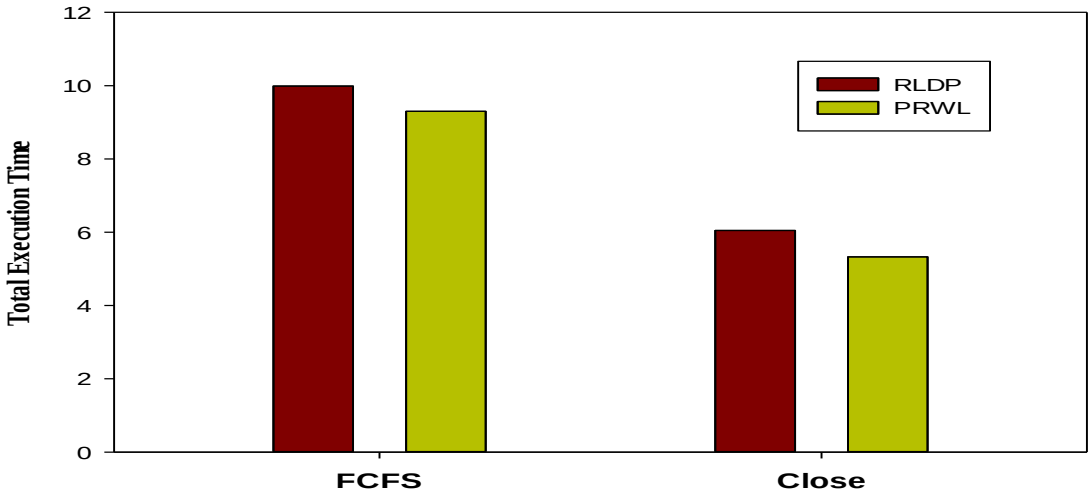

Figure 3.8 % of Overall decrease in Total Execution Time

Maximum Slowdown Time

The results obtained for maximum slowdown time reveals that the performance of RLDP is best among all scheduling policies simulated for evaluation. After RLDP, PRWL scheduling policy is fair while issuing memory requests intended for memory, Figure 3.9.

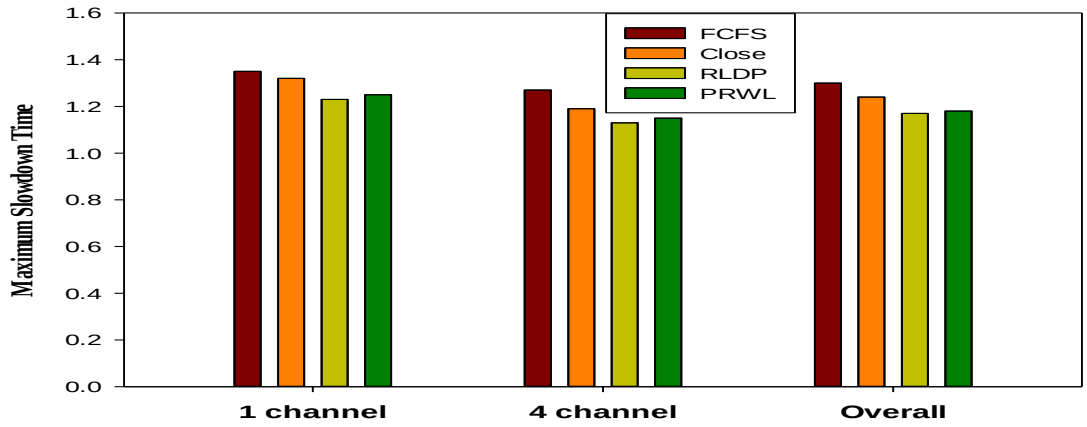

Figure 3.9 Maximum Slowdown Time Comparison

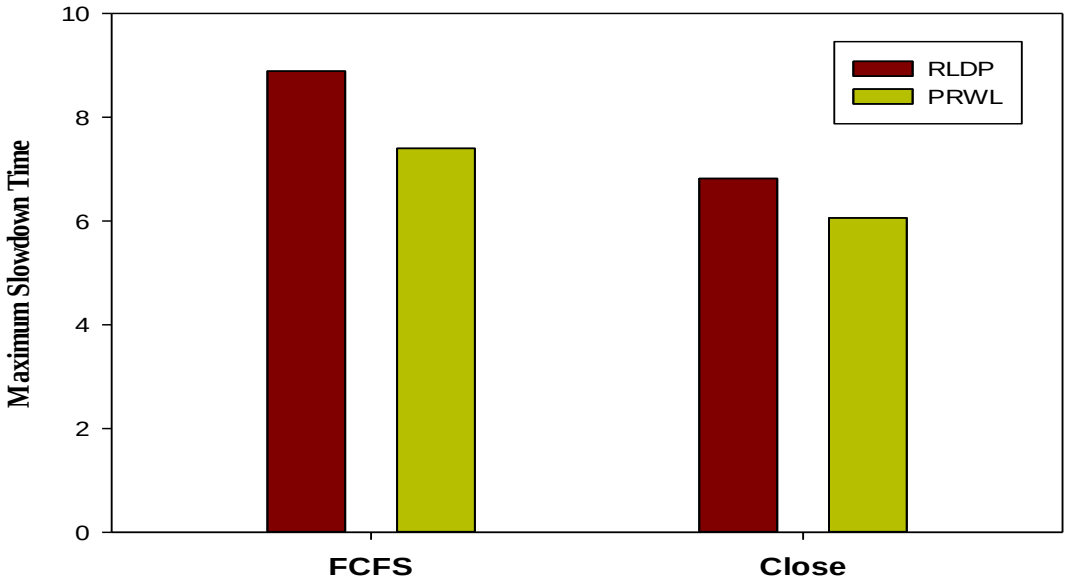

Figure 3.10 % decrease in Maximum Slowdown Time for 1-channel configuration.

Figure 3.10, shows that 8.89% and 7.4% decrement in maximum slowdown time is obtained for single-channel memory configuration when RLDP is compared to FCFS and PRWL is compared to FCFS, respectively. In comparison to close page policy 6.82% and 6.06% reduction is found when RLDP and PRWL is compared to close page policy.

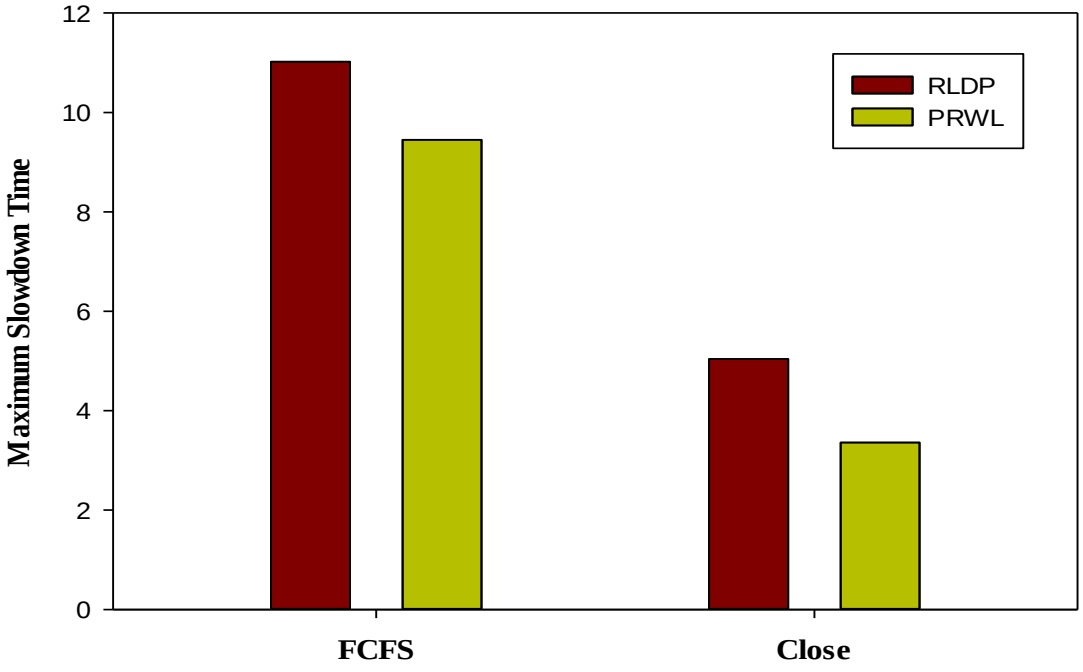

Figure 3.11 % decrease in Maximum Slowdown Time for 4-channel configuration

In 4-channel memory configuration, 11.02% decreased maximum slowdown time is obtained when RLDP is compared to FCFS scheduling policy and 5.04% reduced maximum slowdown time is obtained with respect to close page policy. When PRWL is compared to FCFS scheduling policy 9.4% reduction in maximum slowdown time is obtained. In comparison to close page policy, 3.36% reduced maximum slowdown time is obtained, Figure 3.11. Overall, 9.95% reduction in maximum slowdown time is obtained when RLDP is compared to FCFS scheduling policy and 5.98% reduced maximum slowdown time is obtained when RLDP compared to close page policy. When PRWL is compared with FCFS and close scheduling policy 8.4% and 4.71% reduction in maximum slowdown time is observed, Figure 3.12.

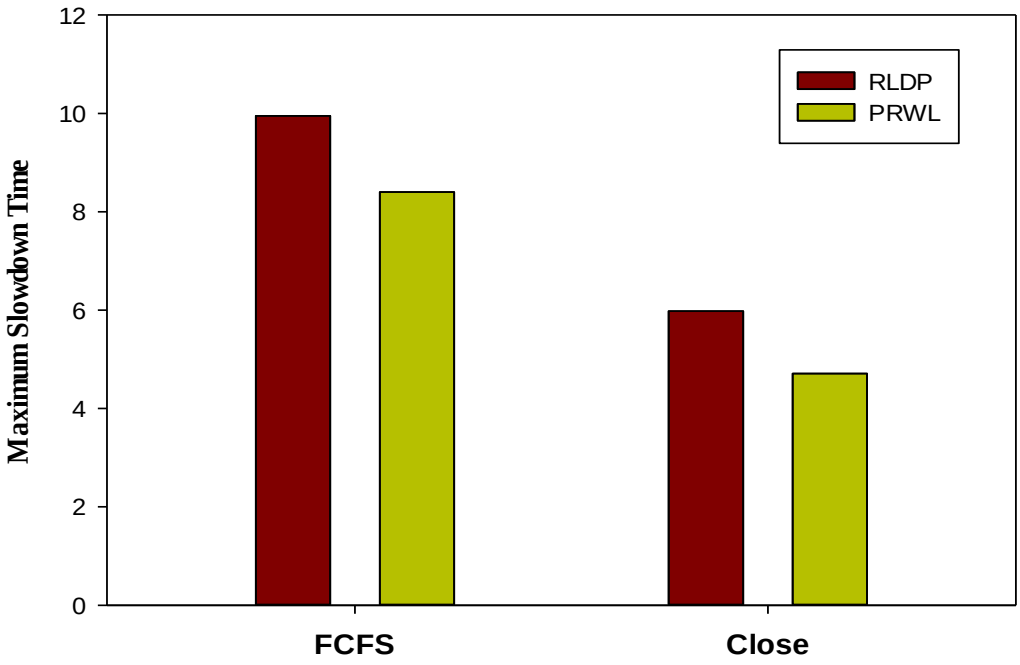

Figure 3.12 % decrease in overall Maximum Slowdown Time

Conclusion

The performance metrics chosen for evaluating the performance are not fully independent. These metrics affect the performance of each other. The scheduler exploiting maximum row buffer hits requires least number of operations to service a memory request which leads to reduced execution time required to complete operation of requests. Less number of operations required for service of a memory request may lead to reduced energy consumption also. Among all simulated policies, the performance of RLDP is best. This is because RLDP prioritizes row buffer hits, prioritizes read requests over writes and delayed close page policy. Similarly, PRWL also works on improving row hits and improves bank level parallelism. So, we

summarize that both memory scheduling policies, *i.e.*, RLDP as well as PRWL focuses on reducing the number of operations required to service a memory request because of which both RLDP scheduler and PRWL scheduler performed better than FCFS scheduling policy and close page scheduling policy. FCFS and close scheduling policies treat each memory request equally. They do not prioritize row hit and read requests over other memory requests. Hence, for an efficient scheduler, in terms of energy consumption as well as performance, row buffer hits should be prioritized over other memory accesses.

3.2 Impact of Inter-thread Fairness on DRAM Performance

The DRAM based main memory is a system's resource that is being shared by all concurrently executing threads in chip multiprocessor system. Main memory resource is shared by all concurrently running threads to fetch the required data and to store the intermediate results produced during the execution the threads. In chip multiprocessor system multiple cores are integrated onto single chip. These multiple chips facilitates multiple threads to run simultaneously for faster and more energy efficient execution [42]. In single core system only single thread run at a time so there exists no contention for accessing the main memory resource. So, with increase in number of cores the necessity of an efficient as well as intelligent scheduler is also increasing. If memory scheduler does not schedules memory requests intelligently then for some threads starvation condition may arise while others running parallel may get unfair priority. So, the starved threads derives to increased maximal slowdown time which additionally leads to increased execution time and energy consumption. So, a scheduler is said to be efficient in terms of energy and performance, if in addition to prioritizing row hits, it also considers fairness among threads.

Concurrently executing applications on multiple cores contend with each other for main memory resource causing inter-thread interference. Interference among threads results in increased wait time for some threads. The increased stall time of a thread is because of two factors, *i.e.,* when other thread's requests are prioritized $T_{interf(others)}$ and stall time due to conflicts generated from same thread $T_{interf(own)}$.

$$T_{\text{int}\,erf} = T_{\text{int}\,erf\,(others)} + T_{\text{int}\,erf\,(own)} \tag{3.13}$$

$T_{interf(others)}$ is further due to two factors, *i.e.*, $T_{interf(bus)}$, interference due to wait time in bus and halt time if interference occurs in DRAM bank, $T_{interf(bank)}$.

$$T_{\text{int } erf}(others) = T_{\text{int } erf(bank)} + T_{\text{int } erf(bus)} \quad (3.14)$$

Every read request or write request is issued to DRAM bank through DRAM bus. The DRAM bus remains unavailable for other requests during this transfer period (T_{bus} cycles). The value T_{bus} depends on type of DRAM used in memory subsystem. T_{bus} value for DDR2 SDRAM is given by

$$T_{bus} = BL/2 \quad (3.15)$$

3.2.1 *Motivational Results*

To analyse the impact of inter-thread fairness, we conducted a comparative analysis among scheduling policies exploiting thread fairness with scheduling policies in which inter-thread fairness is ignored. Table 3.3, presents the simulation environment and simulated policies used for evaluation. Among simulated scheduling policies FCFS and close page policies do not ensure fairness among threads, whereas, FR-FCFS and PBFS scheduling policy ensures fairness among threads.

Table 3.4, describes the workloads simulated on USIMM simulator under two memory configurations, *i.e.*, one having one channel in memory system and other having four channels in memory system to evaluate the impact of inter thread fairness on DRAM's performance. Dual Data Rate-3 DRAM based main memory system is simulated for evaluation. Workloads are formed as the combination of traces, Table 3.4. Traces are extracted from PARSEC [87] and commercial transaction processing workload benchmarks. These workloads are executed using selected scheduling policies and then their performance is evaluated in terms of selected performance metrics.

Table 3.3 Simulation Parameters

Parameter	Description
Examined Schedulers	FCFS, Close, FR-FCFS, PBFS
Simulator	USIMM
Processor Clock Speed	3.2GHz
Memory Bus Speed	800 MHz (plus DDR3)
Cache lines per row	128

Memory Configuration	1-channel configuration, 4-channel configuration
Write Queue Capacity	64
Number of Ranks per channel	2
Number of Banks per channel	8

Table 3.4 Workload Description

Selected Trace (s)	Workloads with Single-Channel m/m Configuration	Workloads with four-Channel m/m Configuration
comm2	1Core_1Chn_1	1Core_4Chn_1
comm1 comm1	2Core_1Chn_1	2Core_4Chn_1
comm1 comm1 comm2 comm2	4Core_1Chn_1	4Core_4Chn_1
fluid swapt comm2 comm2	4Core_1Chn_2	4Core_4Chn_2
face face ferret ferret	4Core_1Chn_3	4Core_4Chn_3
black black freq freq	4Core_1Chn_4	4Core_4Chn_4
stream stream stream stream	4Core_1Chn_5	4Core_4Chn_5
fluid fluid swapt swapt comm2 comm2 ferret ferret	-	8Core_4Chn_1
fluid fluid swapt swapt comm2 comm2 ferret ferret black black freq freq comm1 comm1 stream stream	-	16Core_4Chn_9

The evaluation is conducted on the basis of behaviour metrics like total execution time that depicts total time taken by simulated threads for completing their execution, maximum slowdown time, maximum stall time experienced by simultaneously executing threads, energy-delay product, revealing energy consumed during execution and delay in executing last thread. These performance metrics are inter-dependent on each other. If the threads are interfering with each other to access main memory scheduler then due to contention their maximum slowdown time would be more which further leads to increased execution time and power consumption. It may further impact energy consumed by scheduling policy and performance of schedulers. So, for efficient servicing of memory accesses, memory scheduling policies should be fair while scheduling memory requests generated to access main memory resource.

Total Execution Time

Evaluation made in terms of total execution time depicts that PBFS performed best among all simulated policies. In comparison to FCFS scheduling policy 5.96%, 8.81% reduction in total execution time is obtained in 1-channel, 4-channel memory configuration. In total, 7.46% less time is consumed by PBFS scheduling policy when compared to FCFS scheduling policy. When PBFS is compared to close page policy 3.70%, 3.28%, reduced execution time is obtained, in 1-channel, 4-channel memory configuration. Overall, 3.40% reduction in execution time obtained when PBFS is compared to close page policy. This behaviour of PBFS is because it tries to ensure fairness among threads on the basis of priority among threads. Memory intensive threads are given higher priority than compute intensive threads. PBFS tries to ensure fairness among threads to avoid starvation experienced by threads executing simultaneous with each other. Which results in decreased execution time of threads.

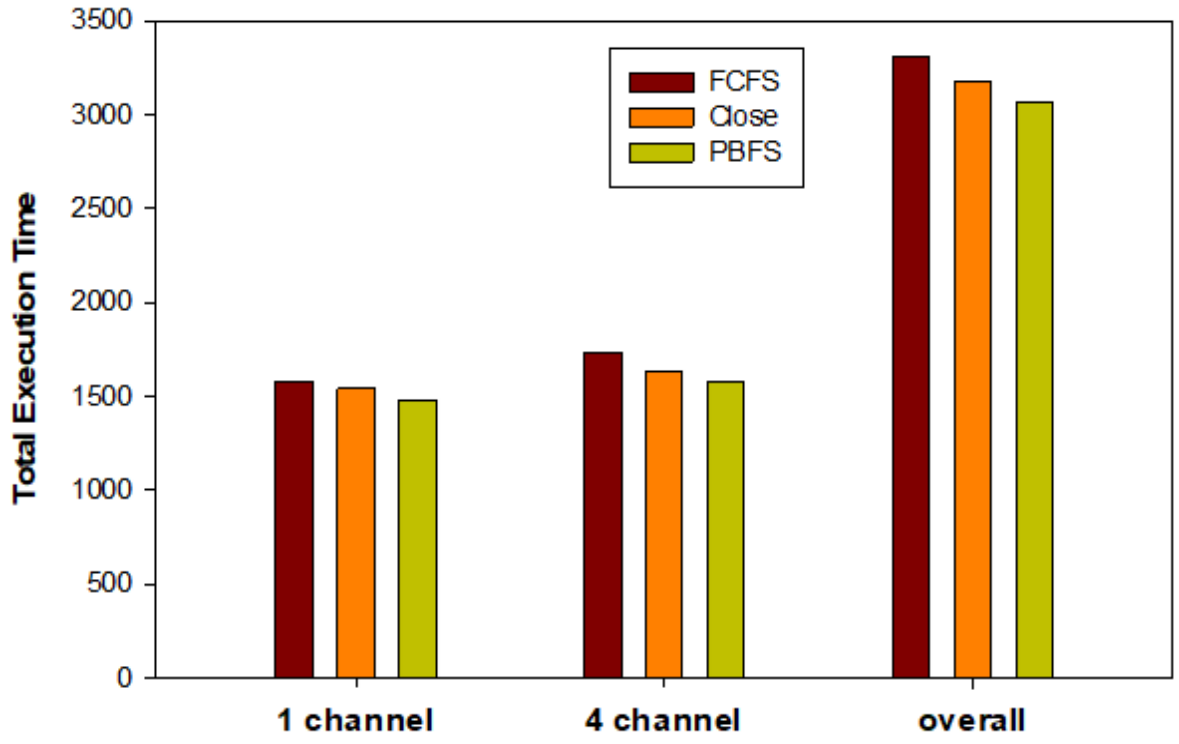

Figure 3.13 Total Execution Time (mCyc) Comparison

Energy-Delay Product

As shown in Figure 3.14, PBFS has consumed least amount of energy while maintaining the performance during execution. This behaviour of PBFS is shown because it tries to maintain fair environment among threads running along each other which leads to reduced congestion

observed by threads hence reduced energy-delay product. In 1-channel memory configuration and 4-channel memory configuration 9.68% and 16.97% reduction is observed when PBFS is compared to FCFS scheduling policy, whereas 4.89%, 5.14% is obtained when compared to close page policy. In terms of energy-delay product, overall 13.82% and 5.16% reduction is obtained when PBFS is compared to FCFS scheduling policy and close page policy, respectively.

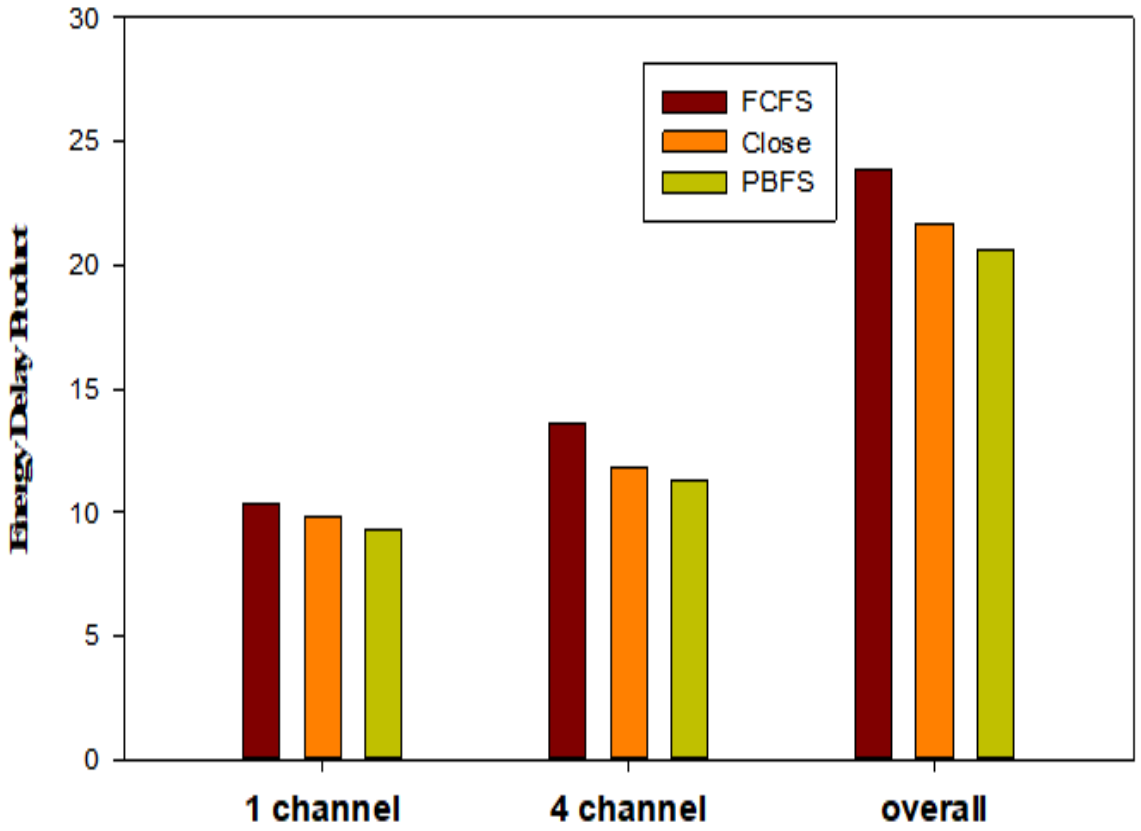

Figure 3.14 EDP (Js) Comparison

Maximum Slowdown Time

As shown in Figure 3.15, PBFS has observed least slowdown. A scheduling policy is said to be fairer if it shows least maximum slowdown. Scheduler being fairer among all simulated policies observes less stall time as compared to other scheduling policies not ensuring fairness among threads. In terms of maximum slowdown time, total, 6.92% reduction is observed when compared to FCFS scheduling policy and 2.42% decreased maximum slowdown time is observed when PBFS is compared to close page policy.

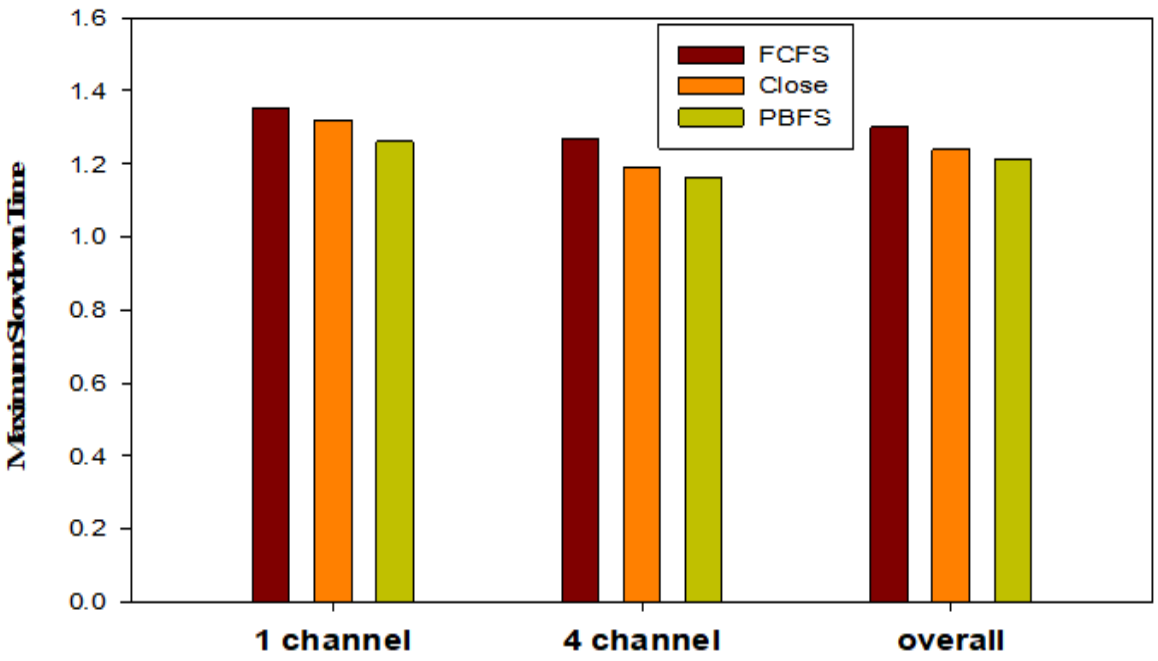

Figure 3.15 Maximum Slowdown Time Comparison

Conclusion

In CMP the threads processing on several cores professes for the common system resources and may intervene with one another for acquiring them. In absence of a genuine memory scheduling policy it is possible that some threads are unjustly computed, leading to a prolonged stall time for other threads. Extended pause time to access memory, results in enhanced maximum slowdown time which further marshals increment in execution time and therefore results in escalated energy consumption. Hence, a noble memory scheduler is required to be legitimate in terms of scheduling threads.

3.3 Proposed Memory Access Scheduling Algorithm

In order to achieve inter-thread fairness and increased performance or decreased energy consumption we propose a scheduling policy named Energy-Efficient Fairness-Aware Memory Access Scheduling (EEFA). The flow chart of proposed scheduler is presented in Figure 3.16.

Proposed scheduling policy tries to ensure that each thread gets equal chance to access shared main memory sub-system by giving more priority to requests generated from threads blocking reorder buffer head. In proposed scheduling policy row buffer hit requests are prioritized over other memory accesses henceforth results in reduced average access latency of memory request

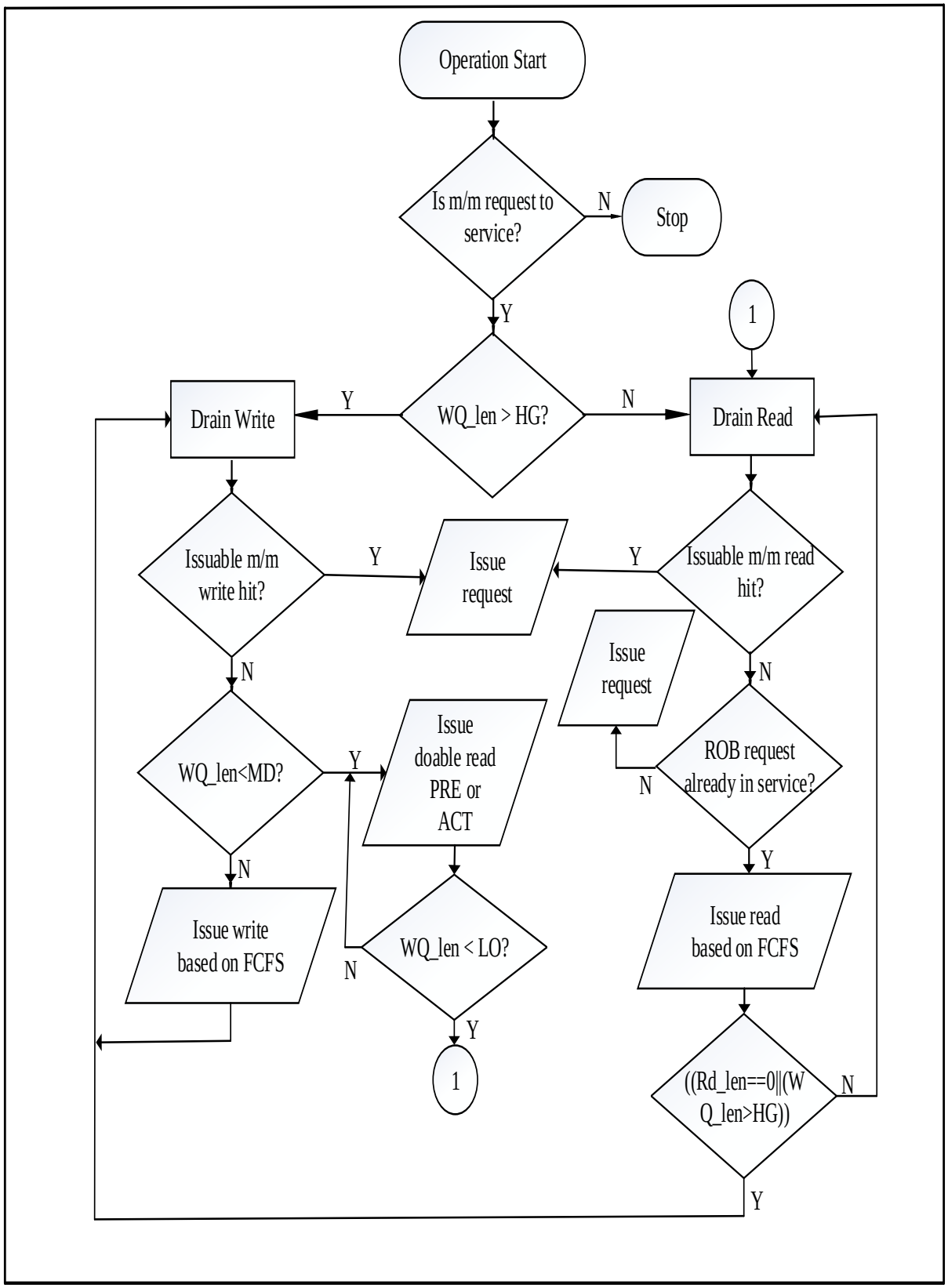

Figure 3.16 Flow Chart of Energy-Efficient Fairness-Aware Memory Access Scheduling

and power consumption during servicing the requests. Another key aspect covered by proposed scheduler is to prioritize memory reads over writes during memory request scheduling. Memory read requests largely impacts the system's performance by halting the processor during servicing the read requests, whereas memory writes requests do not stops the processing of processor. As memory writes do not halt the processor during execution so they are not on the critical for system's performance. Because of this memory read requests should be given high priority than memory write requests. But scheduling of memory write requests to store the required data is also important, as if write queue gats completely filled then it may stop the processing of processor. Hence memory writes can't be ignored completely. This scenario results in a situation where it becomes difficult to decide that when we should schedule memory write requests. In conventional memory access schedulers [29, 30, 42, 67, 88, 89, 90] memory write requests and memory read requests are served in batches. In drain-write mode memory writes are served and in read mode memory reads are served and these modes are strictly separated from one another. Here exists the opportunity to enable parallelism at bank level. Bank level parallelism is enabled by pre-issuing issuable read commands while servicing the write requests on finding command bus is idle. Along with this inter-thread fairness and row buffer hits are also facilitated in the proposed scheduler. Bank level parallelism is also included in proposed scheduler.

The proposed memory scheduling policy enhances system fairness, energy consumption and decreases execution time for chosen workloads and multi-core environment for varied memory configurations.

3.3.1 *Prioritizing Read Requests over Writes*

Memory read requests are more important for the performance of computing system than memory write operations. In proposed scheduling approach memory reads are given high priority over write requests. In proposed scheduling algorithm memory read requests are scheduled first unless memory write requests reaches the high watermark in write queue. When the write queue touches high watermark, *i.e.*, it is about to be full, memory scheduler enters in drain-write mode and starts servicing memory write requests.

3.3.2 *Row Buffer Hit*

In proposed scheduler row buffer hits are increased to ensure that minimum number of operations are required to service memory requests. To enhance row hits, memory hits, *i.e.*, read hits and write hits are prioritized than other memory read/write accesses requested to main

memory for service. Row buffer hits requires minimum amount of time and takes least amount of energy to service a memory request as only column access read strobe operation is required to be issued in order to fetch the data [91] and column access write strobe required to issued, in order to store data. Proposed scheduling tries to maximize row buffer hits.

3.3.3 *Fairness among Threads*

In proposed scheduling policy inter-thread fairness is provided by facilitating equal opportunity to each thread running simultaneously with each other and waiting for being serviced by memory system. Inter-thread fairness is accomplished by giving high priority to requests blocking the reorder buffer head. So, the requests queued in the middle or at the end of reorder buffer also get chance to access the shared main memory resource. Row buffer is blocked by memory intensive threads during execution as they tend to produce more memory requests than compute intensive threads. Therefore, by servicing requests generated by reorder buffer head compute intensive threads observe less stall time to service main memory resource.

3.3.4 *Bank Level Parallelism*

In addition to above mentioned features proposed scheduling algorithm also exploits bank level parallelism.

In conventional scheduling approach, in drain-write mode only memory writes are serviced, memory read requests are not serviced during this mode. As memory reads and writes are serviced in bursts. In proposed memory access scheduling approach, scheduler issues timing constraint satisfying issuable read commands on finding idle memory cycles and when write requests reach low threshold value, *i.e.*, memory writes are nearly drained. Memory controller opens the sense simplifier for forthcoming memory read requests by pre-issuing issuable read commands during drain-write mode. Pre-scheduling of schedulable memory read commands may result in reduced read request service latency and partial enhancement in read-write parallelism.

This read-write interleaving may cause increased service time overhead in terms of additional bus turnaround delay. This effect is rationalized by ensuring that only those memory read commands issued during write mode that do not cause data bus switch its direction, *i.e.*, only PRE and ACT commands are issued during this period. PRE and ACT commands do not cause data bus to change its direction, hence do not result in turnaround delay.

3.4 Experimental Evaluation Methodology

Proposed scheduling policy is built on a device level memory command simulator named USIMM. In USIMM, memory controller schedules device level memory operations based on current status of main memory system, *i.e.*, on the basis of nature of channel(s), rank(s) and bank(s). In order to conduct comparative evaluation we simulated workloads using proposed scheduling approach and same workloads using existing scheduling approach under two different memory configuration, *i.e.*, single-channel memory configuration and four-channel memory configuration. In single-channel memory configuration, memory system is formulated with one channel and in that channel two ranks are present and in each rank four banks are present. In four-channel memory configuration, memory system is composed of four channels and in each channel two ranks are there and in each rank four banks are present. In simulator power related calculations are performed on the basis of Micron's power calculation methodology. The detailed information regarding power simulation are included in [57].

3.4.1 *Benchmarks: Characteristics and Classification*

A set of benchmarks derived from PARSEC benchmark suite were used to constitute workloads. The workloads were run to conduct comparative evaluation. Description of benchmarks is provided in Table 3.5. Experiments were carried out in varied core environment, *i.e.*, ranging from one, two, four, eight and sixteen cores in order to simulate multi-threaded environment. Workload details are provided in Table 3.6. For better understanding, nC_mChan_i, convention is used to represent n-core, m-channel simulation running workload i. For calculating maximum slowdown time MT-Canneal workload is not used because only multithreaded workloads are taken to calculate maximum slowdown time.

Table 3.5 Benchmark Description

Benchmark	Trace Driven	Application Domain
Blackscholes	Black	Financial Analysis
bodytrack	Body	Computer Vision
Facesim	Face	Animation Physics
Ferret	Ferret	Similarity Search
Fluidaminate	Fluid	Animation Physics

Streamcluster	Stream	Data Mining
Swaption	Swapt	Financial Analytics
Canneal	Canneal	Engineering

Table 3.6 Simulated Workloads Description

Traces	Workloads with 1-Channel configuration	Workloads with 4-Channel Configuration
comm2	1Core_1Chan_1	1Core_4Chan_1
comm1 comm1	2Core_1Chan_1	2Core_4Chan_1
comm1 comm1 comm2 comm2	4Core_1Chan_1	4Core_4Chan_1
fluid swapt comm2 comm2	4Core_1Chan_2	4Core_4Chan_2
face face ferret ferret	4Core_1Chan_3	4Core_4Chan_3
black black freq freq	4Core_1Chan_4	4Core_4Chan_4
stream stream stream stream	4Core_1Chan_5	4Core_4Chan_5
fluid fluid swapt swapt comm2 comm2 ferret ferret	-	8Core_4Chan_1
fluid fluid swapt swapt comm2 comm2 ferret ferret black black	-	16Core_4Chan_9

3.4.2 *Performance Analysis Metrices*

We comparatively analysed the performance of proposed scheduling approach with chosen existing scheduling policies, FCFS, Close and PBFS by simulating workloads described in Table 3.2. The comparative performance is evaluated based on selected performance parameters like total execution time, energy delay product, maximum slowdown time, page hit rate and total memory system power consumption.

Total Execution Time: Total execution time includes time consumed by all threads running simultaneously to complete their execution.

Energy-Delay Product: EDP for a simulation is calculated by multiplying the energy consumed for that simulation and the delay to finish the last program in that workload. The

objective of achieving reduced energy consumption while maintaining performance is captured by this performance metric [92]. A scheduling policy is more efficient in terms of energy and performance if its energy delay product is less.

Maximum Slowdown Time: This performance metric is used to measure the fairness among threads [93]. A simulation environment is fairer if maximum slowdown time is less. To achieve fairness among threads by reducing interference among threads stall time experienced by each thread running simultaneously should be curtailed rather considering stall time in total. Maximum slowdown time represents maximum of slowdown time experienced by each thread running simultaneously. Slowdown time is measured by dividing stall time experienced by thread when running simultaneously with other threads (T_{shared}) by stall time experienced by thread when running alone (T_{alone}). Slowdown time for a thread can be calculated by equation 3.16.

$$SlowdownTime = \frac{T_{shared}}{T_{alone}} \tag{3.16}$$

Row hit rate: Row hit is a scenario in which the address being requested by a memory request is already present in the row buffer. In this case least number of steps are required to be performed for servicing a memory request. Row hit rate is a total of read hit rate and write hit rate. Read hit represents page hits obtained while servicing read requests and write hit represents page hits obtained while servicing write requests. Formula for computing read hit rate and write hit rate is given by equation 3.17 and equation 3.18.

$$Read\ Page\ Hit\ Rate = \frac{No_{read_{comnds}} - read_{act} - speculative_{act}}{No\_read_{comnds}} \tag{3.17}$$

$$Write\ Page\ Hit\ Rate = \frac{No_{write_{comnds}} - write_{act}}{No_{write_{comnds}}} \tag{3.18}$$

where, $read_{act}$, represents number of activate commands issued to bring required row in sense amplifier for servicing read requests and $speculative_{act}$, represents speculative activates issued by memory controller.

Total memory System Power Consumption: Total memory system power consumption represents total power consumed by main memory system during execution. Memory system power consumption is a combination of power consumed by different components in terms of read power, write power, refresh power, activate power, background power and terminate power. Total memory system power consumption is power consumed by all memory chips,

i.e., power consumed by one memory chip multiplied by total number of memory chips present in memory system.

3.5 Result Evaluation

Simulated scheduling policy's results are evaluated by comparing it with three already implemented memory access scheduling policies, FCFS, Close-page policy and PBFS scheduler.

3.5.1 *Total Execution Time*

The simulation scenario revealed in Figure 3.17, in terms of execution time shows that proposed scheduling approach showed best performance amongst all chosen scheduling policies.

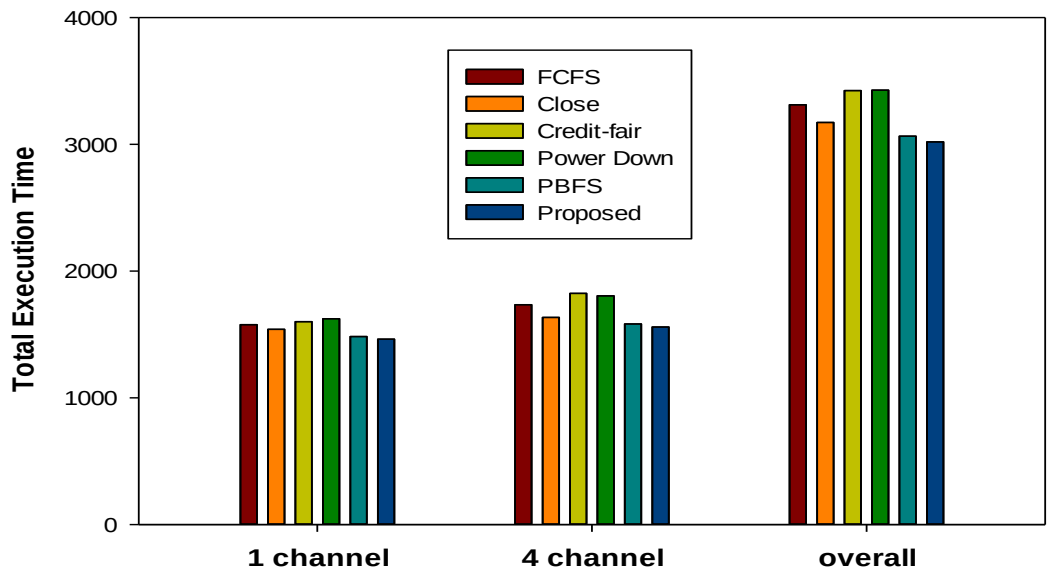

Figure 3.17 Total Execution Time (mCyc) comparison

Proposed scheduler performed best among all simulated schedulers for both memory system configurations. In comparison to FCFS memory scheduling approach, proposed scheduling policy decreases the total execution time by 7.2% for single-channel memory configuration and 10.2% for 4-channel memory configuration. In total, proposed scheduler has shown 8.8% enhancement in terms of total execution time when compared to FCFS approach. In comparison to close page policy proposed memory access scheduling approach has curtailed the total execution time by 4.82%. Similar simulation scenario is observed when comparison is made with respect to PBFS memory access scheduler. 1.42% and 1.52% deduction in total

execution time is observed with respect to PBFS scheduling approach for single-channel memory configuration and 4-channel memory configuration. In total 1.47% reduction is observed in comparison to PBFS scheduling policy.

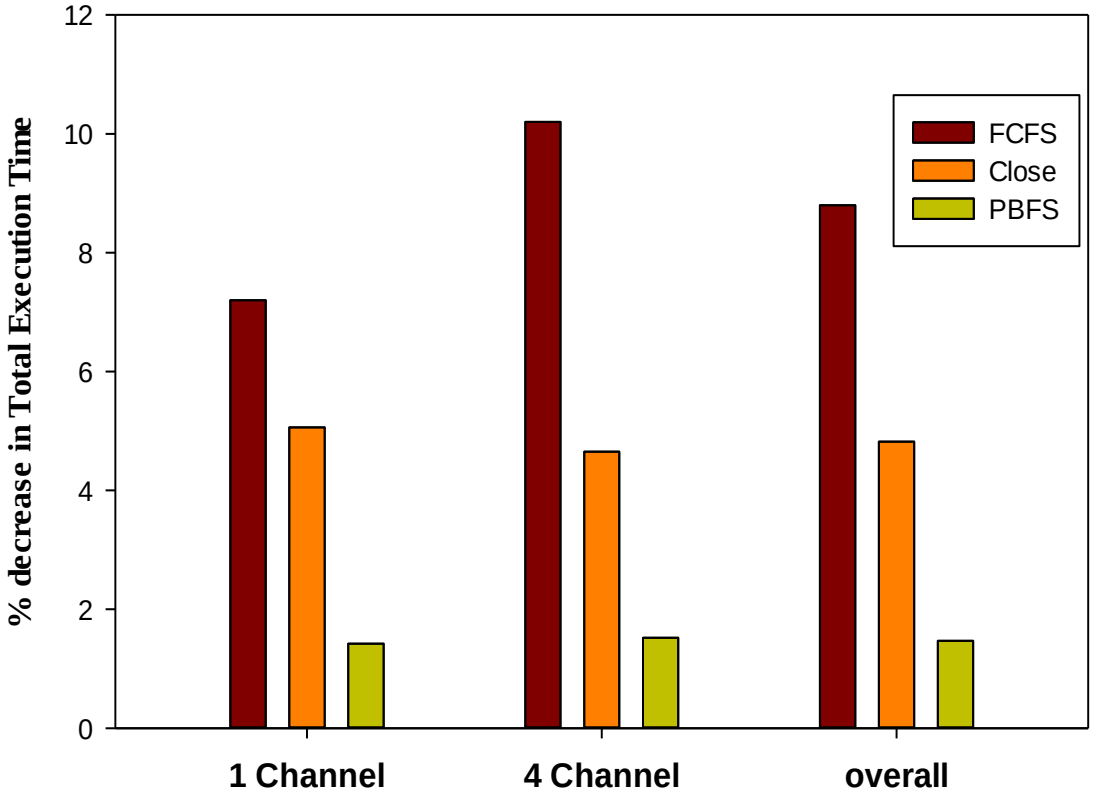

Figure 3.18 % decrease in Total Execution Time

This behaviour is shown by proposed algorithm is because proposed scheduler tries to utilize maximum row buffer hits. Along with this proposed scheduler itemizes read requests over write which reduces processor's halt time and therefore final execution time is also reduced.

3.5.2 *Maximum Slowdown Time*

Figure 3.19, reveals the results obtained in terms of maximal slowdown time. This simulation results obtained depicts that for maximal slowdown time proposed scheduler is better than FCFS, Close and PBFS scheduling approach. This simulation trend shows that proposed scheduling policy increases the fairness and improves the scheduling environment for all simulated threads. On average proposed scheduler performs 0.79% and 1.72% better than PBFS for single channel memory configuration and four channel memory configurations, respectively. Proposed scheduler has reduced overall maximum slowdown time by 1.65%

when compared to PBFS. Figure 3.20, presents the same simulation scenario in terms of maximal slowdown time in relation to close page scheduling approach and FCFS scheduling policy.

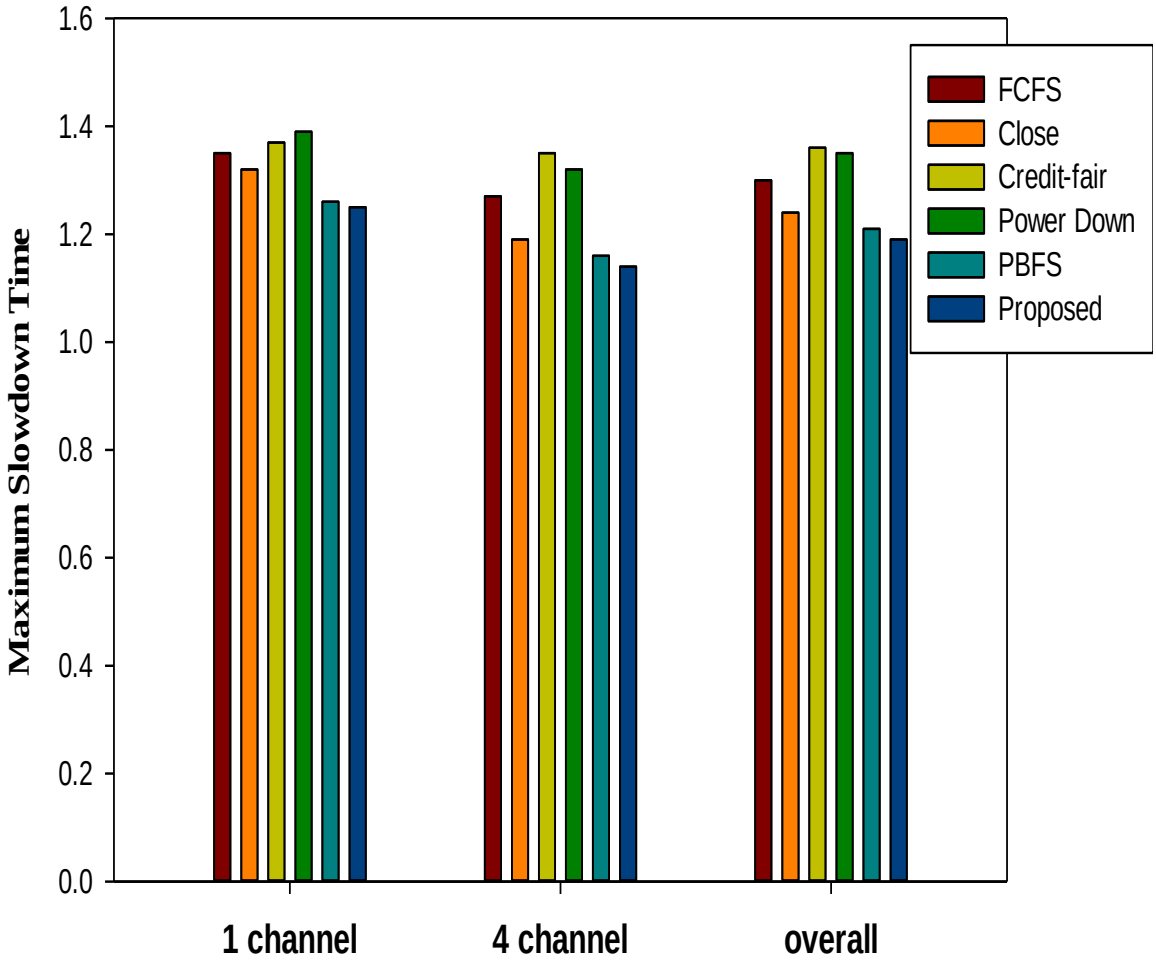

Figure 3.19 Maximum Slowdown Time comparison

In comparison with close page policy 5.3% reduction in maximum slowdown time is seen in single-channel memory configuration, whereas, for 4-channel memory configuration 4.2% deduction is observed. In total 4.03% deduction is observed. Similarly, with respect to FCFS scheduling policy 7.4% reduced maximum slowdown time is found in 1-channel configuration. Proposed scheduler reduced maximum slowdown time by 10.24% in comparison to FCFS scheduling policy for 4-channel memory configuration. Overall, 8.48% reduction is achieved when compared to FCFS scheduler.

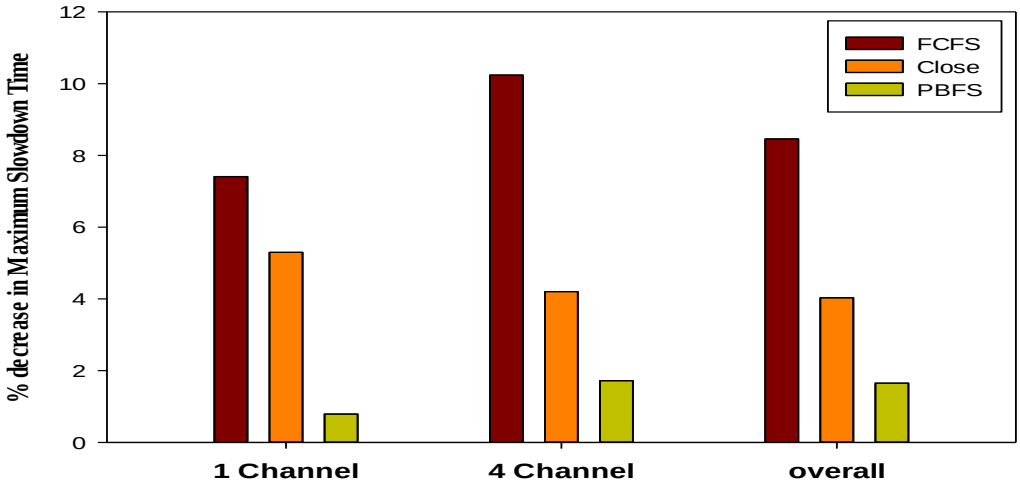

Figure 3.20 % decrease in Maximum Slowdown Time

The observed simulation trend is achieved because proposed scheduling approach tries to attain maximum inter-thread fairness by prioritizing reorder buffer head requests. Due to increased level of fairness, stall time experienced by threads waiting for accessing main memory resource in reorder buffer is now reduced, which further leads to reduced slowdown time for threads and hence minimized maximum slowdown time.

3.5.3 *Energy-Delay Product*

In terms of EDP performance metric again proposed scheduling approach proved to be best among all simulated scheduling policies, Figure 3.21, reveals the results and presents that proposed scheduler has decreased EDP by 17.92% in comparison to FCFS scheduling approach. And with respect to PBFS scheduler also proposed scheduling policy has rationalized EDP by 5.14%, for 1-channel configuration. When comparison is made with respect to PBFS scheduling policy 4.44% deduction is achieved in 4-channel memory confirmation and in total proposed scheduler deducted energy-delay product by 4.76% in comparison to PBFS scheduling policy. Overall, proposed approach has reduced energy-delay product by 9.68% in comparison to close page approach and 4.76% when compared to PBFS scheduling algorithm. This improvement in terms of energy-delay product stems due to considerable reduction in number of operations achieved by proposed scheduling approach. Proposed scheduling policy has reduced the number of operations required to service a memory request by exploiting maximum row buffer hits. Reductions in number of operations further results in rationalized energy consumption.

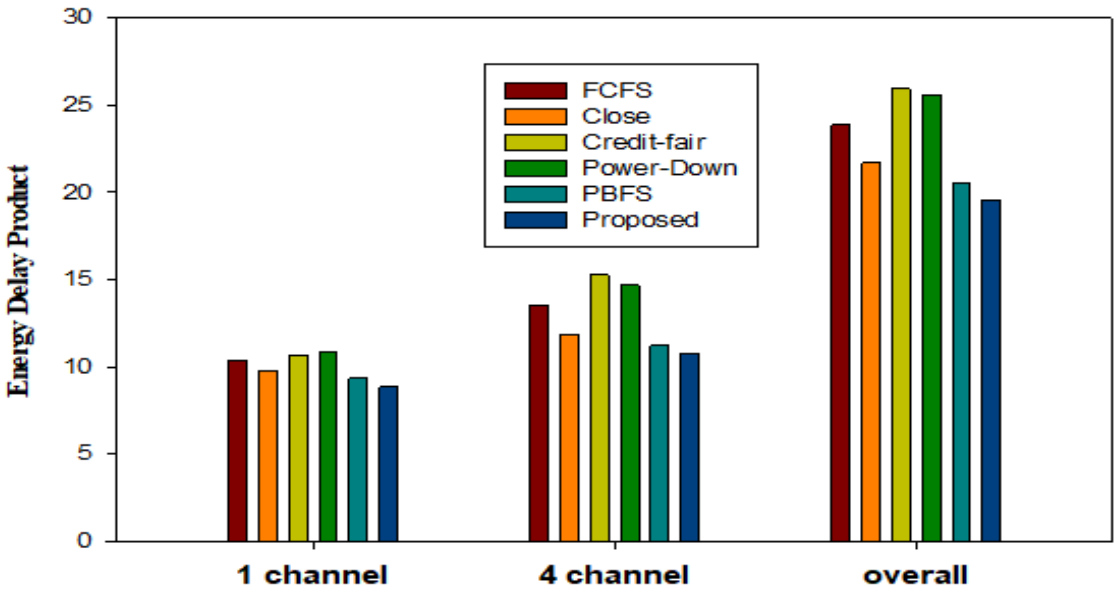

Figure 3.21 EDP (Js) comparison

In addition, proposed scheduler is able to achieve fairness among threads by reducing thread's waiting time in reorder buffer. Reduced stall time of threads leads to reduced slowdown time observed by threads which further results in curtailed delay in complete execution of a thread.

3.5.4 *Total Memory System Power Consumption*

As depicted earlier, proposed memory scheduling approach improves system's performance for all workloads under both memory configurations in comparison to FCFS scheduler, Close page policy and PBFS scheduling approach. As depicted in Figure 3.22, total memory system power consumption is marginally increased with respect to FCFS policy but this increment is not consistent. Whereas, in overall scenario when compared to close page scheduling approach, proposed policy has reduced memory system power consumption by 1.74%. Whereas, when comparison is made with respect to PBFS scheduling policy proposed scheduling reduced power consumed by memory system. With respect to PBFS scheduling approach 7.012% decrement in total memory system power consumption is observed in 1-channel memory configuration. For 4-channel memory configuration 4.83% reduced memory system power consumption is observed. In total, proposed scheduling policy is able to reduce total memory system power consumption by 5.58% with respect to PBFS scheduling approach.

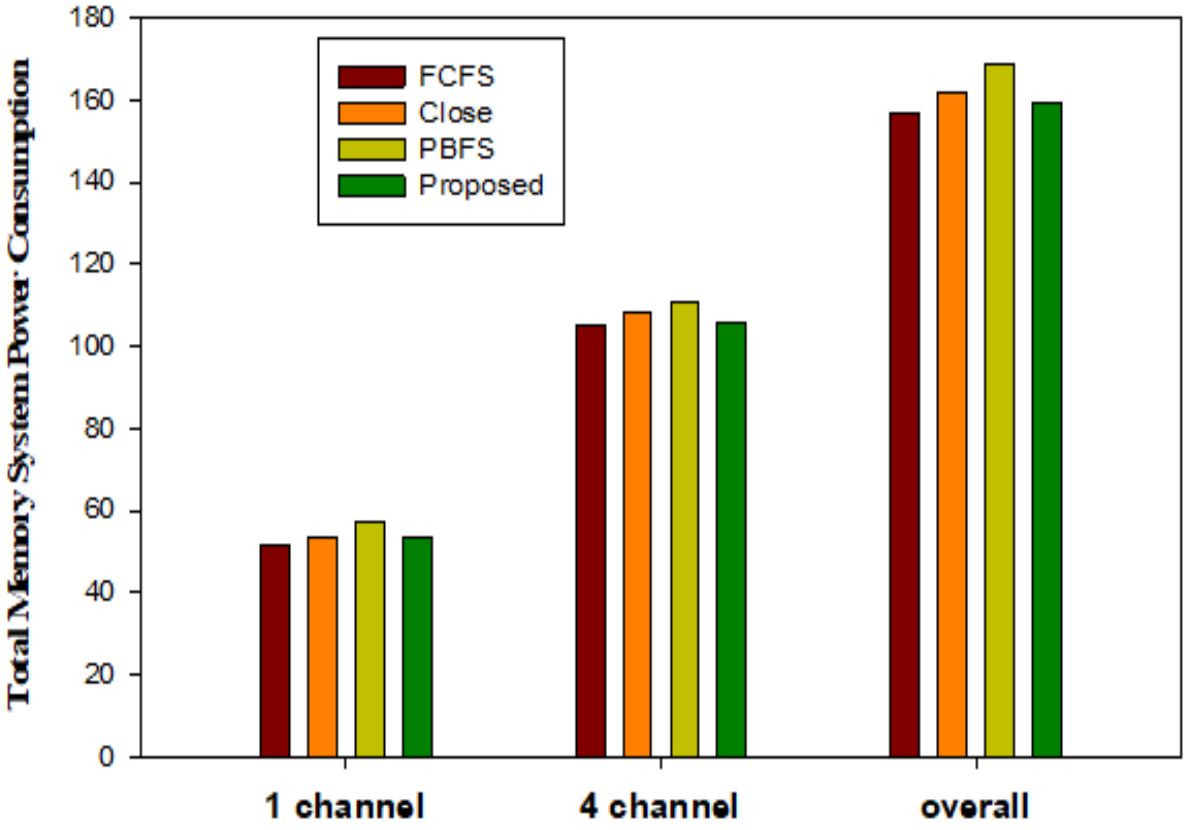

Figure 3.22 Total Memory system Power (W) comparison

Reduction in memory system power consumption is achieved because proposed scheduler improves row buffer hits that further leads to reduced number of operations required to service memory requests. Reduced number of operation leads to reduced power consumption required to service memory requests.

3.5.5 *Row Hit Rate*

The results revealed in Table 3.7, shows that proposed scheduling approach performed best amongst all simulated scheduling approaches when evaluated in terms of row hits, *i.e.*, summation of both read hit rate as well as write hit rate. Proposed scheduler has achieved highest row hit rate. As shown in Table 3.7, taking read hit rate in consideration proposed policy has performed best and same simulation scenario can be observed for write hit rate also, because in proposed scheduler row hit requests are prioritized over other memory commands. Proposed scheduler pre-issues issuable read commands while serving writes in write-drain mode and writes are almost drained. By pre-issuing issuable read commands, the row buffer is kept open for forthcoming memory read requests.

Table 3.7 Row Hit Rate

Workload	Read Page Hit Rate				Write Page Hit Rate			
	FCFS	**Close**	**PBFS**	**Proposed**	**FCFS**	**Close**	**PBFS**	**Proposed**
MT-c1	0.0033	-0.0290	-0.045	-0.0255	-0.2097	-0.2099	-0.934	-0.0859
4C_1Chn_4	0.6291	0.5178	0.503	0.5388	0.1603	0.1508	-0.032	0.3198
2C_1Chn_1	0.5996	0.4846	0.480	0.4788	-0.1653	-0.2475	-0.358	0.1457
4C_1Chn_1	0.5294	0.4167	0.389	0.4413	-0.1619	-0.2084	-0.834	0.0307
1C_1Chn_1	0.5749	0.4605	0.496	0.4518	-0.2850	-0.2890	-0.107	0.0691
4C_1Chn_3	0.6996	0.5952	0.552	0.6187	0.3990	0.3860	0.238	0.4918
4C_1Chn_2	0.5545	0.4430	0.419	0.4645	-0.0861	-0.1163	-0.432	0.1045
4C_1Chn_5	0.6461	0.5340	0.510	0.5548	0.1837	0.1662	-0.03	0.3223
MTc-4	0.0185	0.0073	-0.0095	0.0038	-0.6412	-0.7449	-0.089	-0.0003
4C_4Chn_4	0.0479	0.0057	-0.0093	0.0066	-0.4024	-0.4406	-0.089	-0.0034
2C_4Chn_1	0.0595	0.0074	0.0066	0.0052	-0.0707	-0.1321	0.092	0.1193
4C_4Chn_1	0.0141	0.0041	-0.0063	0.0032	-0.4801	-0.5352	-0.036	0.0270
1C_4Chn_1	0.0160	0.0026	0.00004	0.0019	-0.0699	-0.0853	-0.001	0.0041
4C_4Chn_3	0.0638	-0.0038	-0.0301	0.0062	-0.3775	-0.4282	-0.196	0.0118
4C_4Chn_2	0.0197	0.0025	-0.1003	0.0028	-0.3988	-0.4230	-0.354	-0.0210
4C_4Chn_5	0.0466	0.0043	-0.0435	0.0051	-0.3885	-0.4572	-2.467	-0.0115
8C_4Chn_1	0.0074	-0.0058	-0.0123	-0.0038	-0.4052	-0.4768	-0.065	-0.0427
16C_4Chn_9	-0.0140	-0.0222	-0.0151	-0.0194	-0.2988	-0.3438	-0.128	-0.0684
1-Channel	0.5296	0.4279	0.1543	0.4404	-0.0206	-0.0460	-0.281	0.1748
4-Channel	0.0280	0.0002	0.1850	0.0012	-0.3533	-0.4067	-0.582	0.0015
Average	0.2509	0.1903	0.1713	0.1964	-0.2054	-0.2464	-0.3234	0.0785

3.6 Conclusion

Memory access latency, energy consumption while servicing memory requests and memory capacity to support multi-threaded environment are major memory design concerns these days. Amongst these prime factors, memory latency and energy consumption can be optimized by efficiently and intelligently re-scheduling the memory access requests. We propose a memory access scheduling approach that significantly reduces the memory access latency and energy consumption of DRAM main memory while servicing memory requests while creating a fair environment for each thread. In proposed scheduling approach prioritize row hits are prioritized over other memory requests to obtain reduced total execution time and energy delay product.

Along with aforementioned goals, bank-level parallelism is also exploited to make optimal use of available system's resources. Among a varied variety of chosen workloads using both memory system configurations for 1-, 2-, 4-, 8- and 16-core environment, we showed that proposed scheduling approach is consistently able to facilitate

- high level of inter-thread fairness, 1.63%, 8.46%, 4.03% of improvement in terms of fairness is observed in comparison to PBFS, close and FCFS scheduling policy.
- improved energy-delay product, 4.76% reduction in energy delay product is observed when compared to PBFS.
- rationalized total execution time, 1.47%, 8.8%, 4.82% of improvement in terms of total execution time is observed in comparison to PBFS, FCFS and close scheduling policy
- improved row buffer hits, 12.78% improvement in read page hit rate is observed compared to PBFS.
- reduced total memory system power consumption, 4.83% reduction in total memory system power consumption is observed when compared to PBFS.

CHAPTER 4

DRAM SCHEDULER OPTIMIZED FOR READ-WRITE SWITCHES

In current scenario, energy consumption, performance and capacity of main memory system are key aspects that affect computing system design. These days, computing systems are facilitated with multiple cores. Multicore system enables simultaneous execution of multiple applications. These concurrently running applications interfere at main memory. Main memory is a major resource demanded by running threads because it stores data structures that are required for execution of an application. Main memory sub-system's performance and energy consumption can be improved by rationalizing the number of operations required to access its memory contents and by limiting the delay to service the memory access. It can be achieved by intelligently scheduling the memory requests and it is underlying memory access scheduler that decides the scheduling of memory accesses. We proposed a memory access scheduling scheme, EEPAF, for limiting the energy consumption and refining the performance of main memory. EEPAF, prioritizes reads over writes, implements delayed write drain policy, exploits row buffer hits, increases bank level parallelism and ensures fairness among threads. The results quantify the main memory energy consumption for different workloads under varied core environment and demonstrate significant reduction in power consumption, energy-delay product, and execution time, while improving performance.

4.1 Proposed Memory Access Scheduler

4.1.1 *Baseline Scheduler*

We consider a baseline scheduler proposed earlier in previous chapter, which minimally switches between read and write drain mode by first servicing read and write requests in bursts and second by pre issuing issuable read commands during write drain mode. In previously proposed scheduler memory reads are given more preference than memory writes, because memory reads are more significant for improved system's performance. When processor issues memory read request, it stops its execution waiting for results fetched from main memory in response to issued memory read instruction. To prioritize memory read accesses the scheduler always enter in read drain mode (in which only read instructions are serviced) unless write queue is about to full. If write queue is about to be full, scheduler enters in drain write mode. While servicing memory requests row buffer hits are prioritized over other memory read/write requests as row hits requires less number of steps to perform hence reduced service time as

well as reduced related power consumption. These two factors collectively results in reduced energy consumption of the memory system. After prioritizing row buffer hits in order to ensure thread fairness memory accesses issued from threads blocking the reorder buffer are serviced. This is so because memory intensive threads tends to block reorder buffer and compute intensive threads sometimes enter into starvation state. To ensure that such starvation condition do not arise, memory requests blocking the reorder buffer are serviced first than other memory requests. After servicing the requests generated from reorder buffer other memory read/write requests are serviced. If no row hit request is there or request generated from reorder buffer is already in service the scheduler issues requests to service in FCFS manner. The requests that arrives first are serviced first. In proposed scheduler bank level parallelism is achieved by interleaving doable reads and writes. In write drain mode on finding idle cycle EEPAF issues non-conflicting read commands opening the sense amplifier for upcoming read requests. This write-read interleaving helps to exploit bank level parallelism and also increases read hits for upcoming read requests.

4.1.2 *Reduced Read-Write Switching*

In proposed memory access scheduler, EEPAF, delayed write drain policy is employed on top of baseline scheduler. Delayed write drain policy further prioritizes read requests and exploits row buffer hit. In addition, it reduces the frequency of entering in write drain mode which leads to reduced read-write switching. In conventional scheduling policies once all read requests are served, *i.e.*, read queue gets empty, the scheduler enters into write drain mode. In proposed scheduler instead of immediately entering into write mode, scheduler delays entering in write drain mode and waits for incoming read requests. Delayed write drain is applied only when memory traffic is not heavy otherwise conventional drain policy is employed. Whether the traffic is heavy or not, depends on historic memory request frequency. For a certain amount of time memory requests issued for a particular channel are observed, if memory requests exceeds a certain threshold value memory traffic is considered heavy otherwise not. So, by extending read drain mode proposed scheduler prioritizes read requests, more read hits are achievable now and considerably reduces latency caused due to bus turnaround time. Reduced turnaround time is achieved by reducing the frequency of switching between servicing memory reads and writes. Delayed write drain policy enhanced the scheduler's ability to reduce energy consumption and performance. Flow chart of our implemented scheduling policy is given in Figure 4.1.

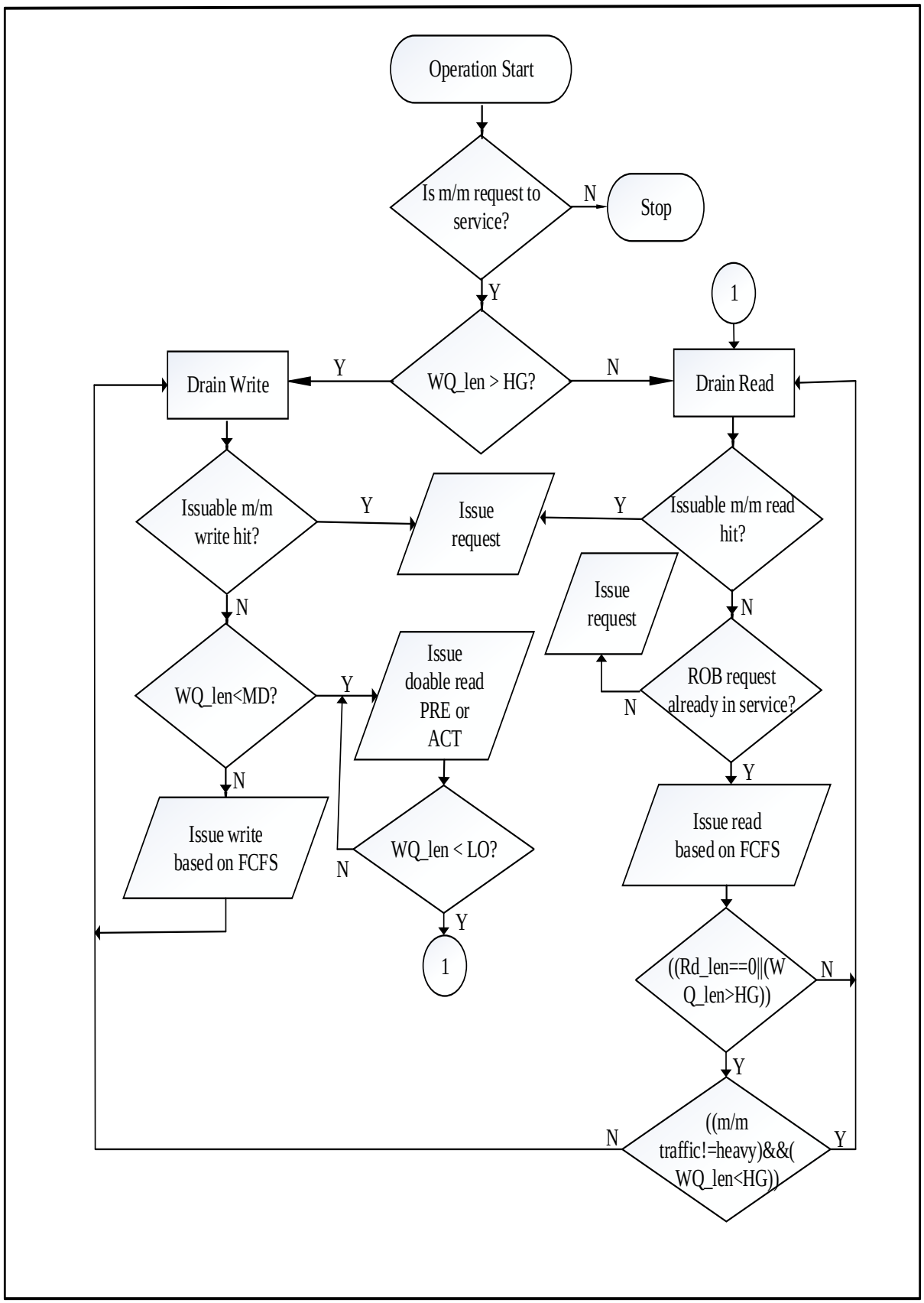

Figure 4.1 Flow Chart of EEPAF

4.2 Methodology

In this section, we at first briefly describe the system configuration and workloads used for evaluating the performance of proposed scheduler. Then we present the performance metrics used for conducting quantitative analysis of EEPAF.

4.2.1 *System Configuration and Workloads*

We build our proposed scheduler on simulator named USIMM. USIMM directs device level memory commands based on current memory status. To evaluate proposed scheduler experiments are run using two memory configurations, Table 4.1 provides details of both memory configurations. In simulator power related calculations are made on the bases of micron's power calculation methodology.

We evaluate the performance of EEPAF in multicore environment varying from 1, 2, 4, 8 and 16 cores for varied variety of workloads. Multithreaded workloads from commercial transaction processing (*e.g.*, comm1 and comm2) and PARSEC (e.g., black, face, ferret, fluid, freq, stream, swapt, MT*-canneal) are used for simulation. Using before mentioned trace files ten different workload combinations are made and simulated for both memory combinations in varied core environment.

4.2.2 *Metrics*

We quantitatively compare EEPAF with four previously proposed memory access schedulers, *i.e.*, FCFS, close, RLDP and PBFS. The comparison is conducted in terms of power consumption, fairness and performance. We use energy-delay product to capture the goal of improved performance at reduced energy consumption or same energy consumption [92]. To measure unfairness among threads, maximum slowdown time performance metric is used [93]. Total memory system power consumption is used to calculate power consumed in memory system. In addition to before mentioned metrics, total execution time performance metric is used to capture the thread's execution time.

4.3 Evaluation

We evaluated sensitivity of proposed scheduler EEPAF, to varying core count and memory configuration. For analyzing the impact of memory configuration, we run experiments using both two memory configurations, *i.e.*, configuration-1 and configuration-2 (details in Table 4.1). Sensitivity to core count is evaluated by varying number of cores using simulation. For quantitative analysis, we evaluated EEPAF in comparison to four previously proposed

schedulers (FCFS, Close, RLDP and PBFS) in terms of Memory system power consumption, Energy Delay Product, Total Execution Time and Maximum Slowdown Time.

Table 4.1 Memory Configurations

Parameters	Configuartion-1	Configuration-2
Processor clock speed	3.2GHz	3.2GHz
Processor ROB size	128	160
Memory bus speed	800 MHz (plus DDR)	800 MHz (plus DDR)
Memory channels	1	4
Ranks per channel	2	2
Banks per Rank	8	8
Cache lines per row	128	128

4.3.1 *Memory System Power Consumption*

For both memory configurations, proposed scheduler outperforms all simulated memory access policies under multi-core environment in terms of memory system power consumption. Figure 4.2, depicts the performance of EEPAF in comparison to simulated schedulers for i-channel memory configuration and varied core count.

Here exception is FCFS scheduling policy. The performance of FCFS scheduling policy is better than EEPAF in terms of memory system power consumption because FCFS employs simple mechanism and does not exhaust power to limit other factors. On analyzing the simulation trend we find that there is increase in FCFS power consumption as core count increases. For 4-core environment FCFS power consumption is greater than EEPAF. This because of the fact that FCFS does not work well in multicore environment due to thread's interference. With increase in number of cores, simultaneously running threads also increases. These parallel executing threads contend with each other for accessing main memory resource. If memory scheduling policy is not fair while scheduling memory accesses then some threads may feel starvation condition, which further leads to increased power consumption. FCFS is simplest scheduling policy. It does not ensures fairness among threads while scheduling requests generated by these memory requests hence consumes more power with increased core count.

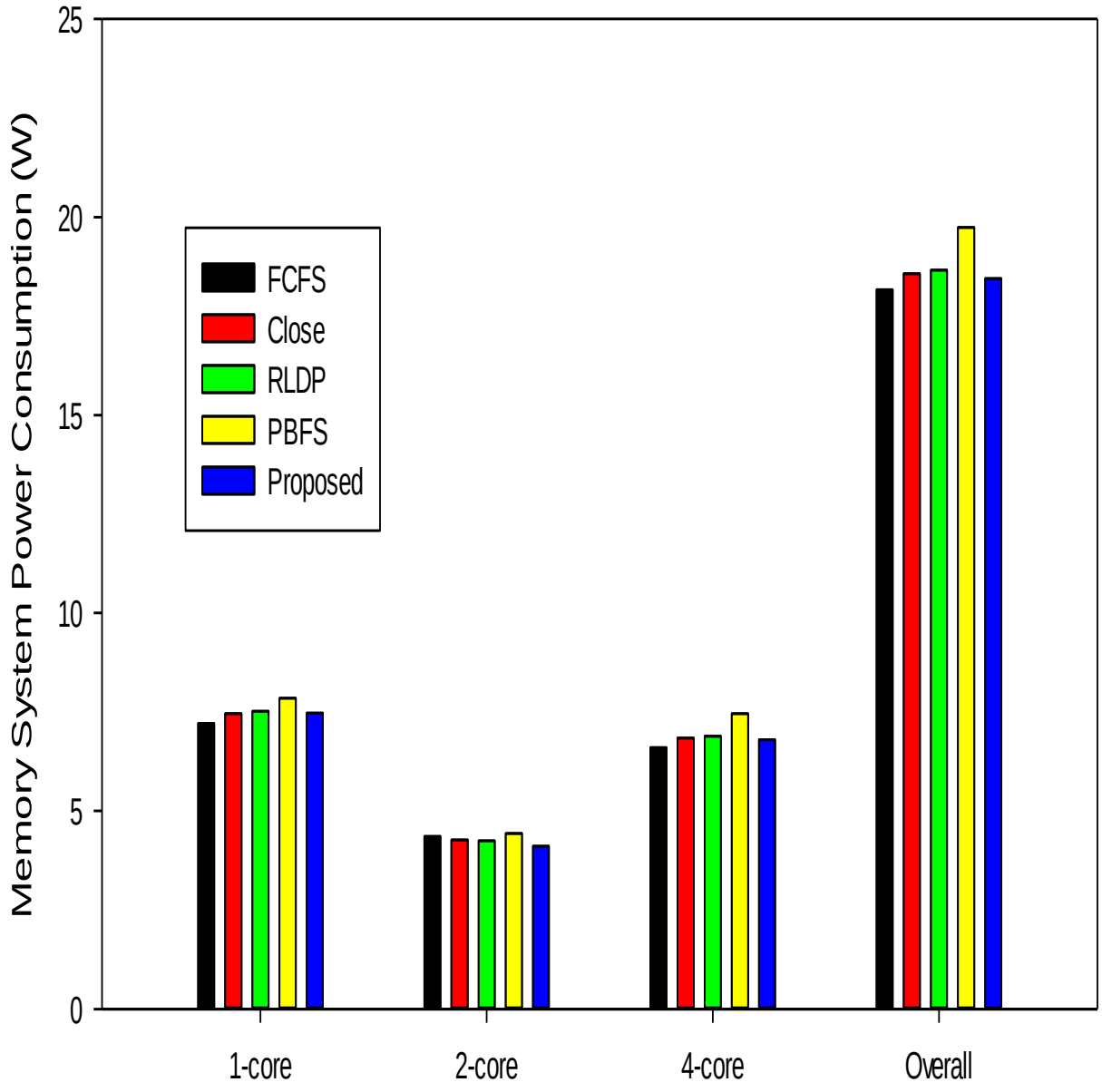

Figure 4.2 Comparison based on Memory System Power Consumption using memory configuration-1

Figure 4.3, depicts the simulation trend obtained in terms of memory system power consumption for simulated scheduling policies in 4-channel memory configuration. For 4-channel memory configuration proposed scheduler consumed less power when compared to close page policy, RLDP scheduling policy and PBFS scheduler. But in comparison to FCFS scheduling approach marginal increase in power consumption is observed.

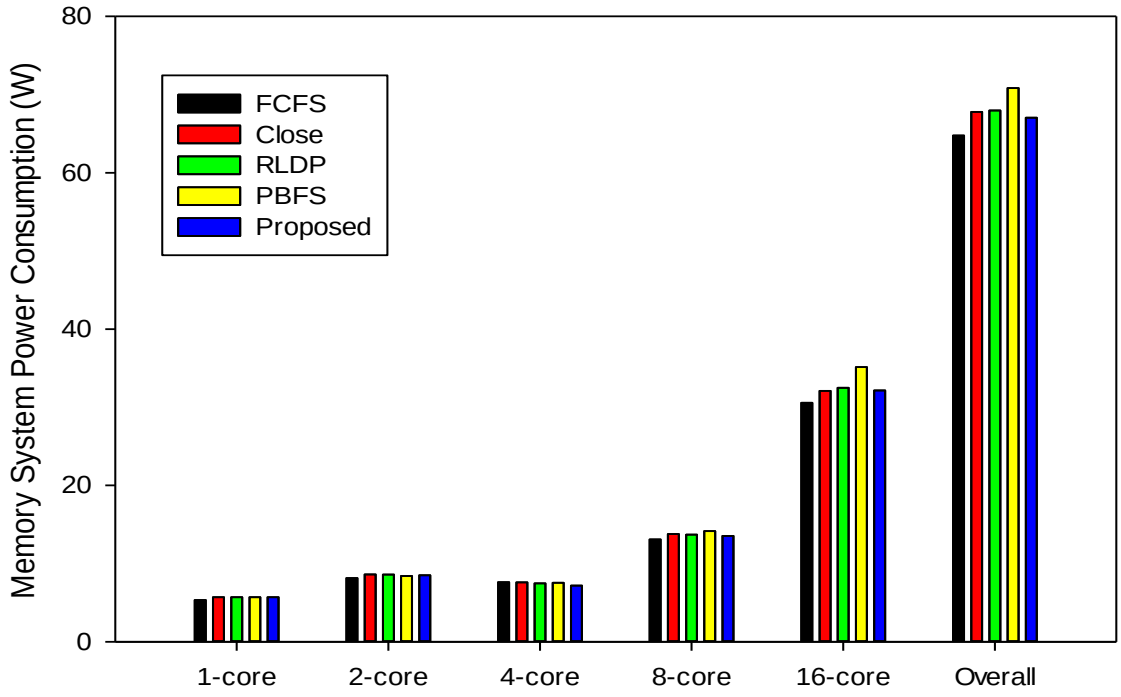

Figure 4.3 Comparison based on Memory System Power Consumption using memory configuration-2

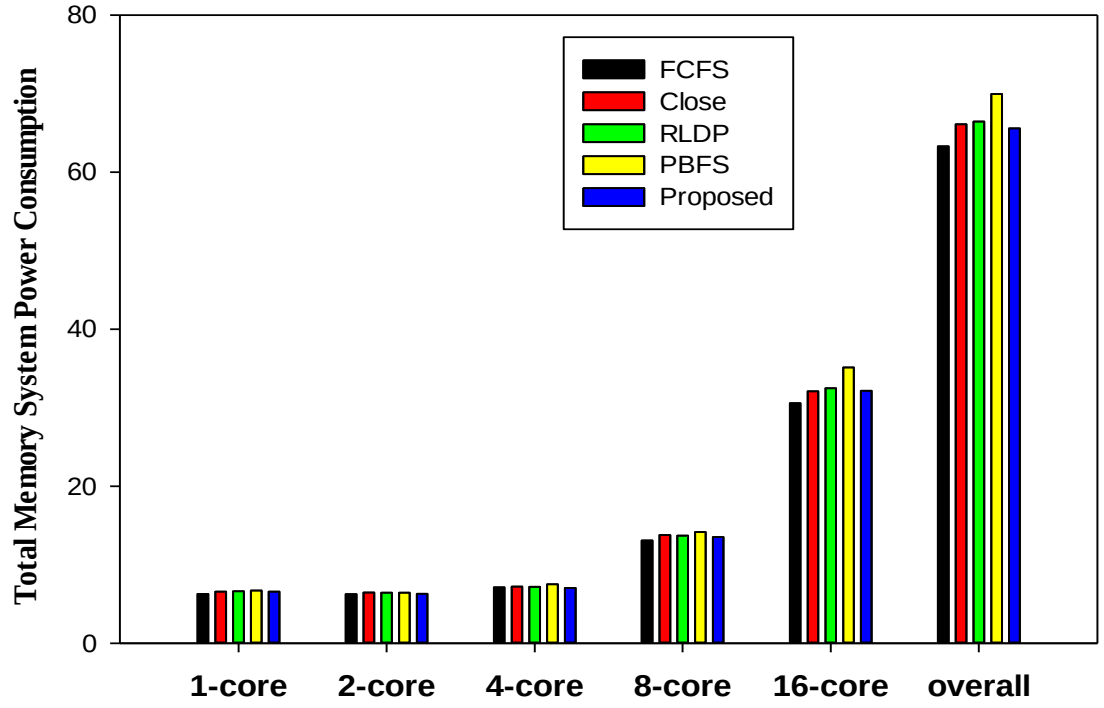

Figure 4.4 Overall Memory System Power Consumption

Simulation trend obtained for all simulated scheduling policy as whole, depicted by Figure 4.4 conveys that in total proposed scheduler consumed less power than close page policy, RLDP and PBFS scheduling policy. However, marginal increase in power consumption is observed when compared to FCFS scheduling policy.

4.3.2 *Energy Delay Product*

The results shown in Figure 4.5, for Energy Delay Product reveals that proposed scheduler's performance is best among all memory access scheduling policies (i.e., FCFS, Close, RLDP and PBFS) for configuration-1 and configuration-2, memory configurations.

Figure 4.5, depicts the simulation trend obtained in terms of energy delay product in memory configuration-1, facilitating one channel in memory system. As per obtained results, proposed scheduler performed best among all simulated policies. In memory configuration-1, proposed scheduler reduced energy consumption while maintaining performance of the system more as compared to all simulated scheduling policies. Despite of increased power consumption in comparison to FCFS, proposed scheduling policy is able to reduce energy delay product because proposed scheduler has managed to reduce total execution time required to complete execution of simulated workloads.

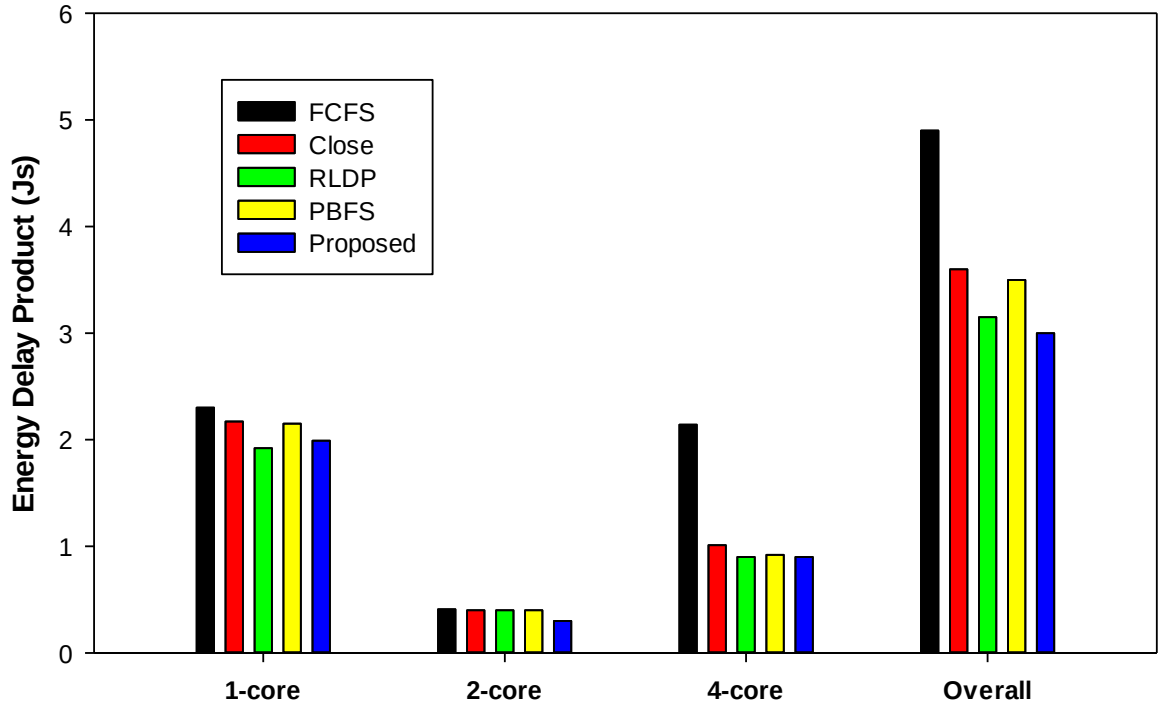

Figure 4.5 Comparison based on Energy Delay Product using memory configuration-1

Figure 4.6, depicts the simulation trend obtained in terms of energy delay product in memory configuration-2. The simulation trend shown in Figure 4.7, reveals that in multi-channel environment proposed scheduling policy outperforms all simulated schedulers. Using configuration-2, in comparison to PBFS, RLDP, Close and FCFS, EEPAF reduced energy delay product by 3.5%, 0.41%, 10.16% and 21.05% respectively.

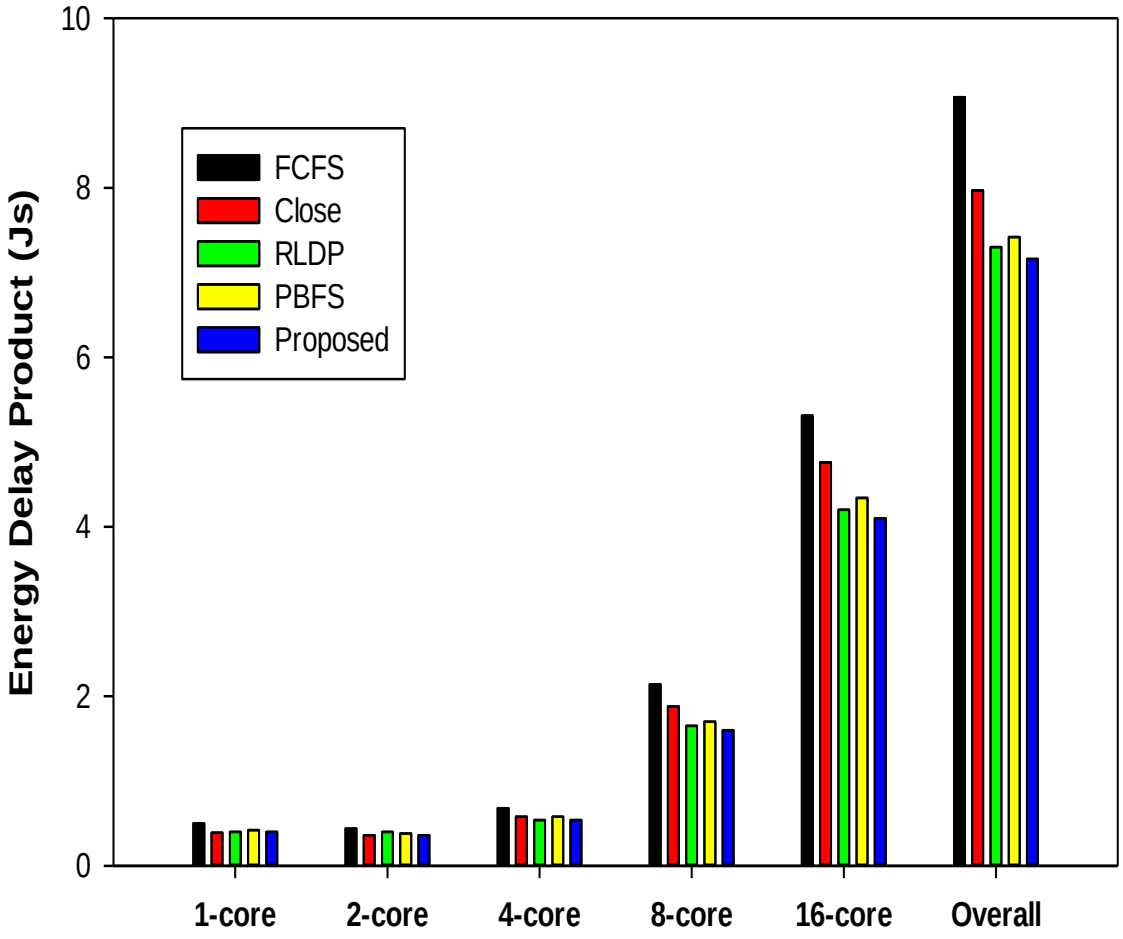

Figure 4.6 Comparison based on Energy Delay Product using memory configuration-2

In total, proposed scheduler has shown most reduction in energy delay product as compared to all simulated policies, Figure 4.7. For uni-core environment 8.5% and 2.5% reduction in energy delay product is observed when compared to PBFS and RLDP, respectively. With respect to PBFS 5.1%, 6.08%, 5.88%, 1.84% reduction in energy-delay product is observed in 2-, 4-, 8-, 16- core environment. In total 4.2% and 0.12% reduction in energy delay product is observed with respect to PBFS and RLDP, respectively.

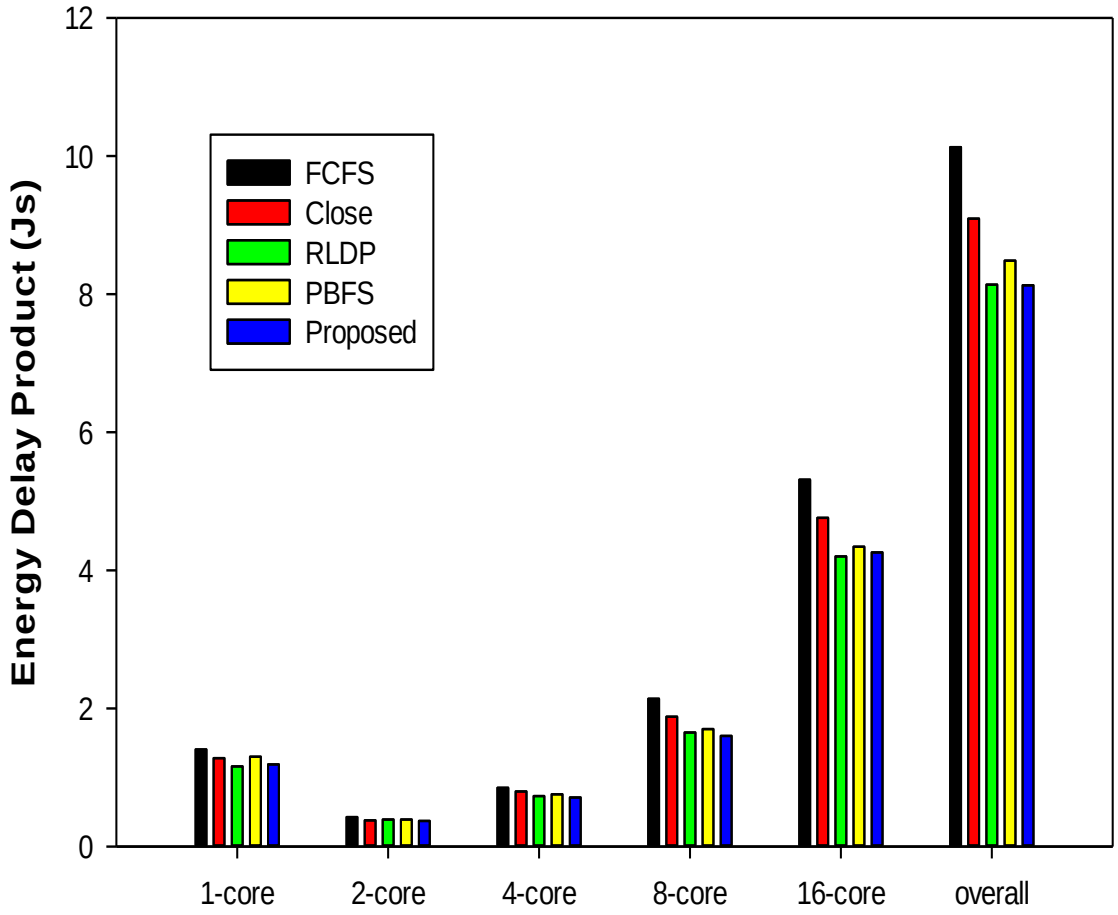

Figure 4.7 Overall Energy Delay Product

4.3.3 *Total Execution Time*

The simulation trend seen in Figure 4.10, reveals that overall performance of proposed scheduler is better than PBFS, FCFS and close page policy. Proposed scheduling policy shows significant reduction in execution time required by workloads to complete their execution. In 1-channel environment proposed scheduler reduced total execution time when compared to FCFS, PBFS and close page policy, Figure 4.8. However, when compared to RLDP scheduling policy proposed scheduler has shown increase in total execution time. For memory configutaion-2, also same simulation trend is seen. In 4-channel memory configuration, proposed approach performed best among all simulated policies in terms of total execution time. Proposed scheduler took less time as compared to close, RLDP and PBFS scheduling policy but marginal increase in execution time is observed when compared to RLDP scheduling policy to complete execution of simulated workloads.

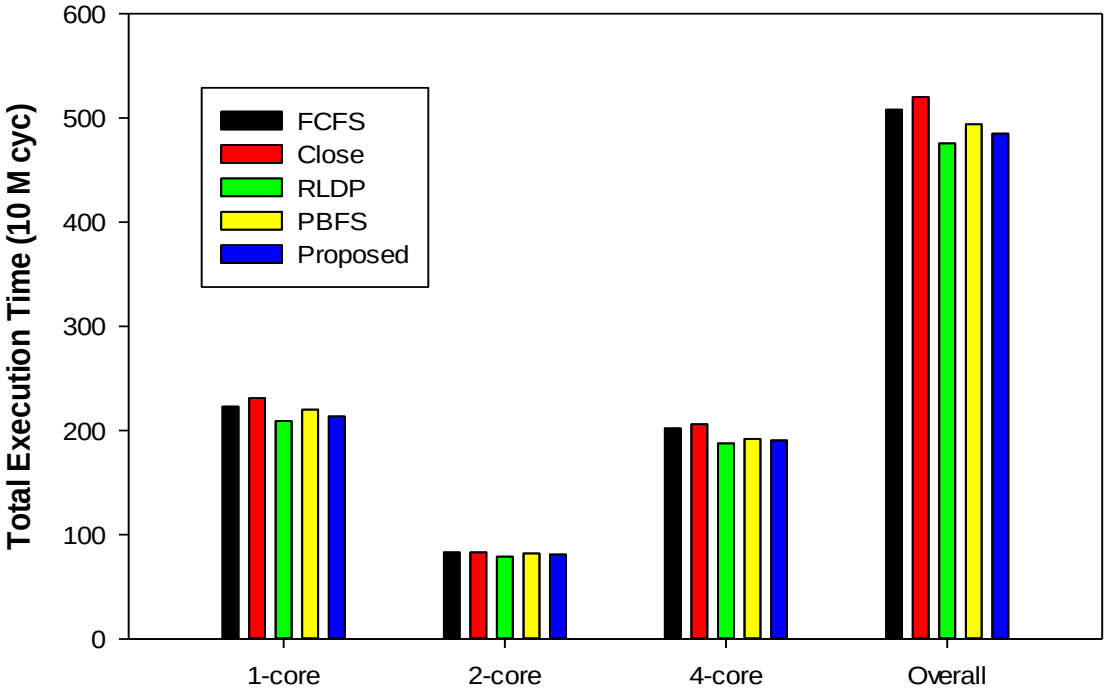

Figure 4.8 Comparison based on Total Execution Time using memory configuration-1

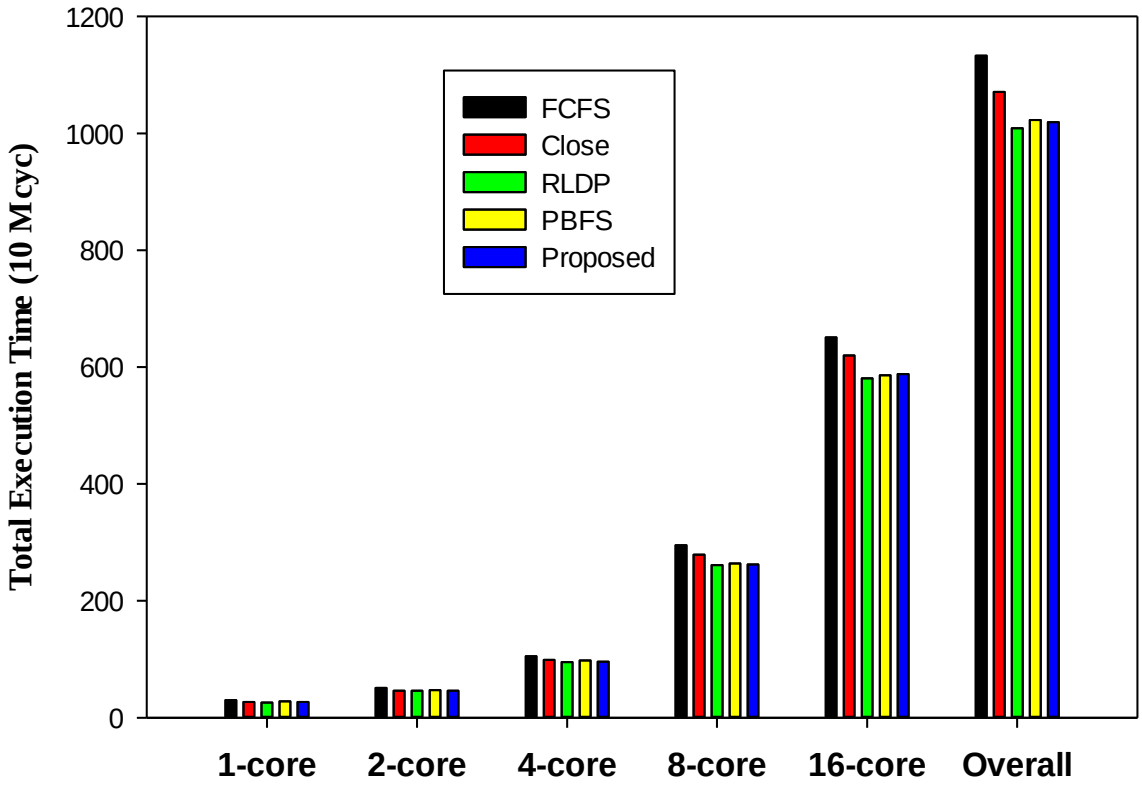

Figure 4.9 Comparison based on Total Execution Time using memory configuration-2

As a whole, Figure 4.10, proposed scheduling policy shows significant reduction in execution time, i.e., 10.06%, 4.85% and 0.43% when compared to FCFS, close and PBFS scheduling policies respectively. In comparison to RLDP scheduling policy, proposed approach shows 1.10% increase in execution time. But, if we consider energy consumption then proposed scheduler reduced energy consumption in comparison to RLDP while maintaining performance of the memory system depicted by energy-delay product performance metric. Increased complexity of proposed scheduling policy resulted in increased total memory system power consumption.

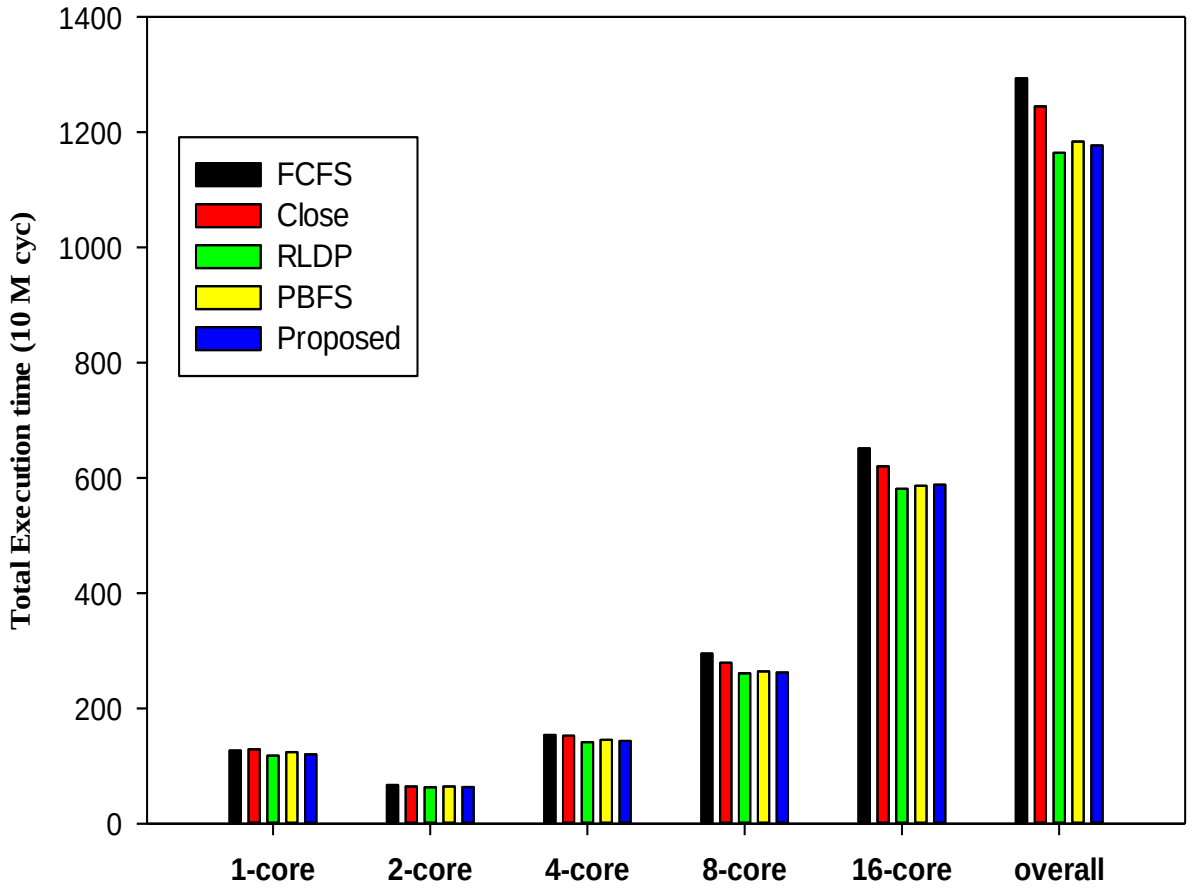

Figure 4.10 Overall Total Execution Time

4.3.4 *Maximum Slowdown Time*

For maximum slowdown time performance metric, EEPAF showed best performance in comparison to other simulated memory access schedulers for memory configuration-1. Proposed scheduler reduced stall time observed by threads running simultaneously significantly in varied core environment. In memory system using 1-channel proposed

scheduler reduced thread's waiting time, *i.e.*, proposed scheduler is able to ensure fair environment for scheduling of requests generated for main memory system. The results obtained for simulated policies in terms of maximum slowdown time is revealed by Figure 4.11. In memory configuration-2 under varied core environment, proposed scheduling policy outperforms all simulated scheduling approaches, *i.e.*, in 4-channel memory configuration also proposed scheduler performed best among simulated approaches, shown in Figure 4.12.

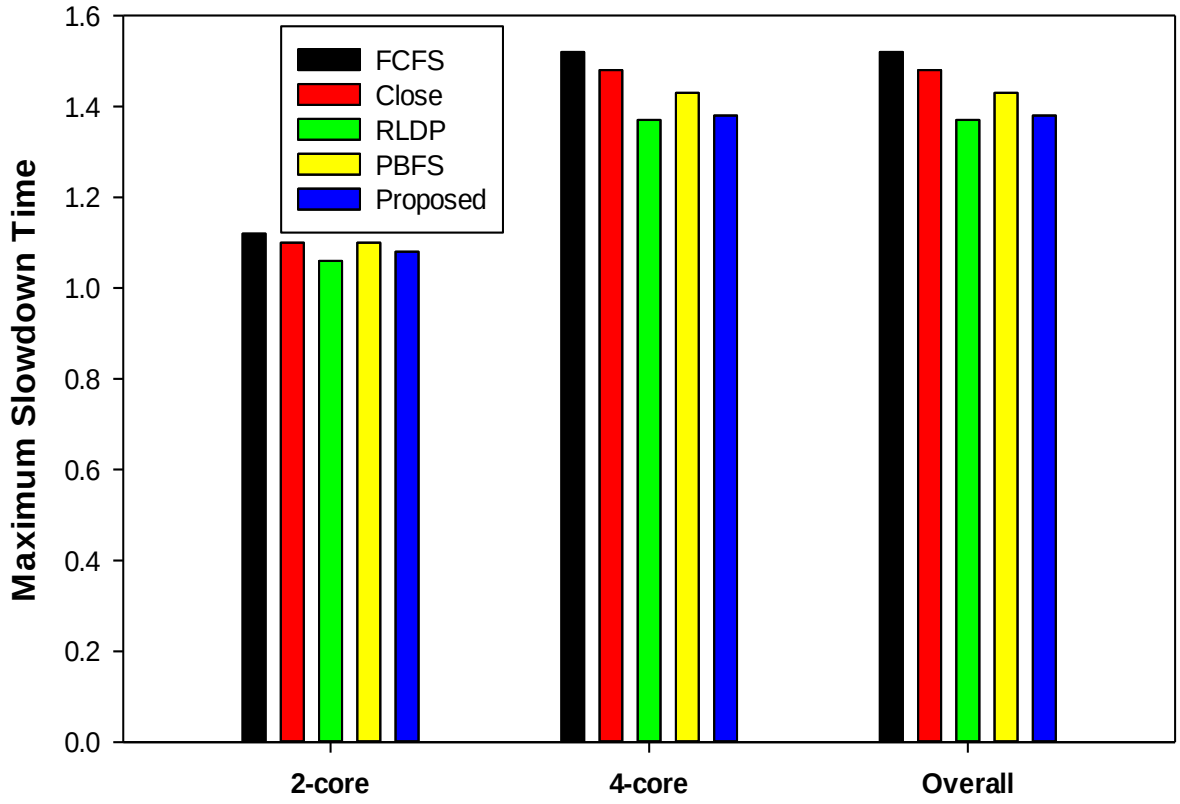

Figure 4.11 Comparison based on Maximum Slowdown Time using memory configuration-1

By limiting maximum slowdown time, EEPAF managed to reduce un-fairness among simultaneously running threads in multicore platform.

As depicted by Figure 4.13. In comparison to PBFS proposed memory scheduling policy reduced unfairness among threads by 1.82%, 3.5%, 0.81%, 0.59% 1-, 2-, 4-, 8- and16-, core environment. In total, 0.6% reduction in maximum slowdown time is obtained when compared to PBFS scheduling policy.

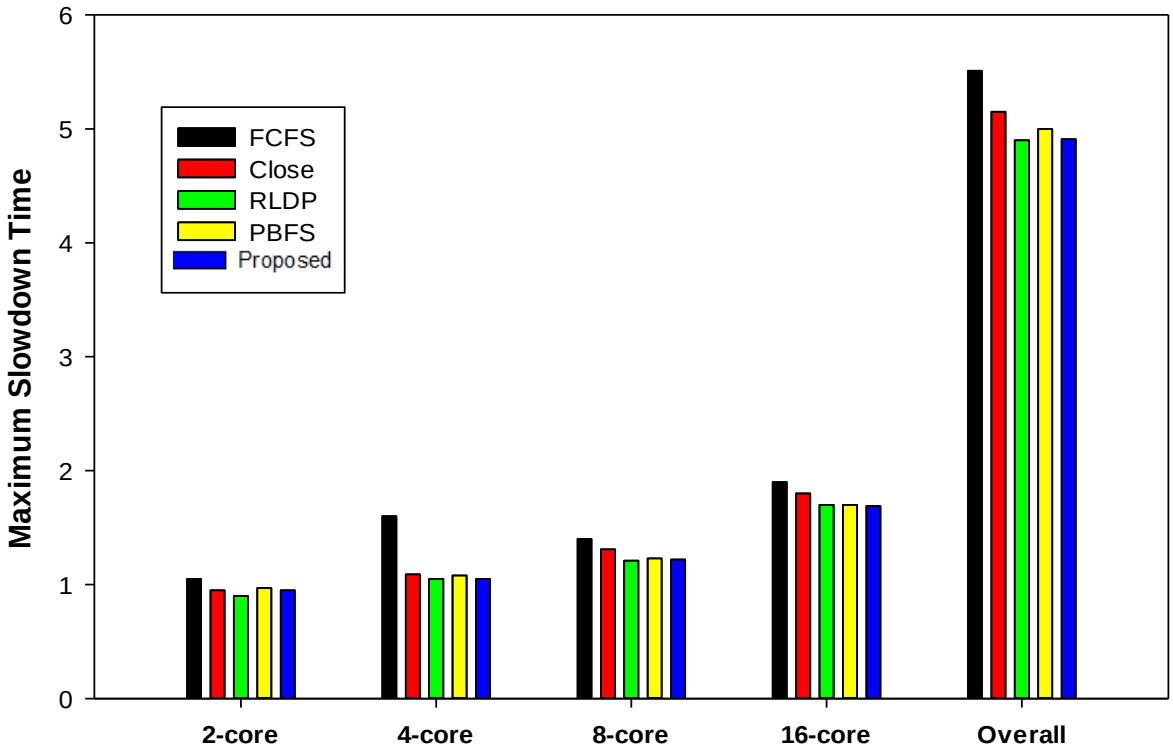

Figure 4.12 Comparison based on Maximum Slowdown Time using memory configuration-2

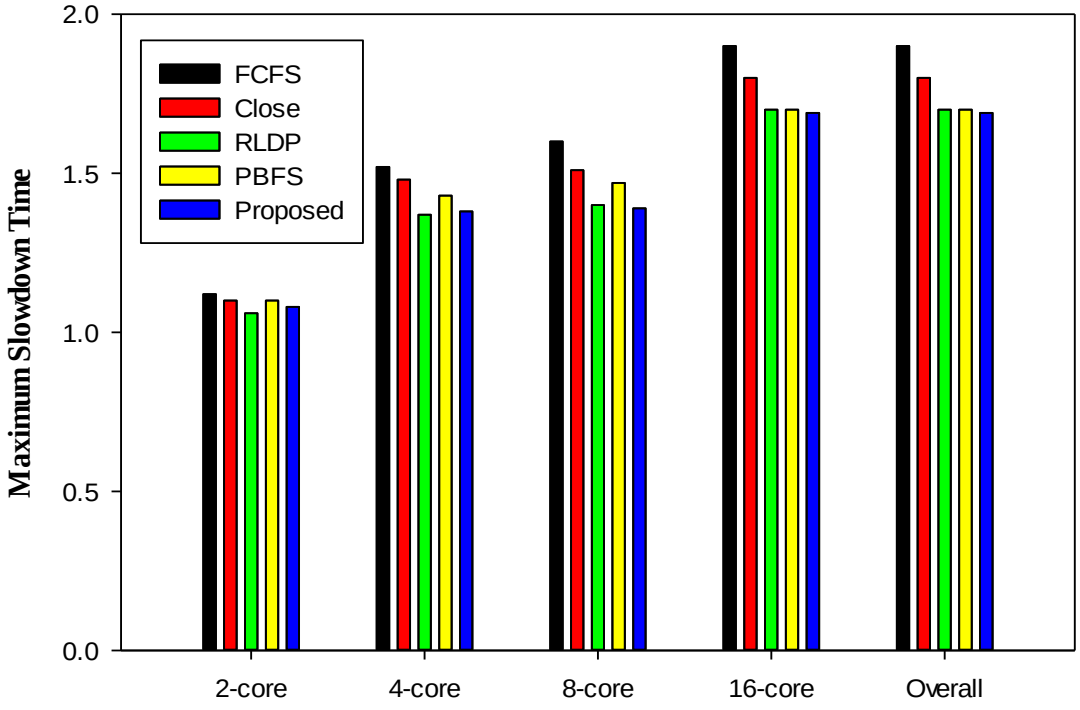

Figure 4.13 Overall Maximum Slowdown Time.

4.4 Conclusion

We introduce energy-efficient performance aware fair memory scheduler, EEPAF. The detailed analysis conducted across a wide variety of workload in varied core environment reveals that EEPAF significantly reduces energy consumption and improves performance of memory system while maintaining fairness among threads. EEPAF reduces the issue of energy consumption by rationalizing power consumption and execution time of a thread. Reduction in power consumption is achieved by limiting the operations required to service the issued memory request. Reduction in number of operations is achieved by maximizing row hits. Whereas, thread's execution time is reduced by i) reducing processor's stall time (by prioritizing reads over writes) ii) minimizing the slowdown time of a thread (reducing unfairness) iii) enhancing bank level parallelism (write-read interleaving) and iv) reducing requests service time (exploiting row hits). EEPAF does not adversely affect the performance of system while reducing energy consumption because it considers both quantities, *i.e*., power and execution time while scheduling commands. Conclusion can be drawn from performed analysis that for multicore environment EEPAF can be an efficient and efficacious memory access scheduling strategy. In future, further more efficient memory schedulers can be explored. Also interaction of EEPAF with other scheduling policies can be an interesting area to work on.

CHAPTER 5

A DRAM SCHEDULER OPTIMIZED FOR DRAM ACCESS LATENCY

To improve the performance and energy consumption of chip multiprocessor (CMP) system, memory request serving latencies should be minimized. These latencies can be minimized by scheduling appropriate memory command at appropriate time. We proposed a scheduler that reduces latency related to serving memory read requests by delaying switching into write drain mode when memory traffic is not heavy and write queue is not full. Memory reads are more important to handle than memory writes for system's performance. Further, precharge and activate operations are performed using constant stride prefetcher. In idle memory cycles the scheduler issues row precharge commands using cache prefetching technique based on Global History Buffer. Authors in [94] have used stride detector and Global History Buffer based speculative precharges and activates, but they treat memory reads and memory writes equally. Whereas, proposed scheduler in this paper prioritizes reads over writes for better system performance. Our evaluations show that proposed scheduling policy significantly outperforms previous schedulers [94, 95] in varied multicore environments in terms of performance as well as energy consumption.

5.1 Proposed Scheduling Policy

Proposed scheduling policy is a combination of delayed write drain policy [96], constant stride prefetcher [97] and global history buffer based prefetching technique [98]. So, the proposed scheduler is divided into three entities and their issuing power is as per the listing below:

- Delayed write drain and FR-FCFS based scheduler.
- Constant stride prefetcher based predictor.
- Close page policy based on global history buffer based prefetcher.

5.1.1 *Delayed Write Drain and FR-FCFS based Base Scheduler*

Delayed write drain policy is based on the assumption that while serving read requests in read mode, read queue gets empty, it is more beneficial to wait for upcoming requests than immediately entering in write-drain mode. It is so, because reads are more critical for system's performance than writes.

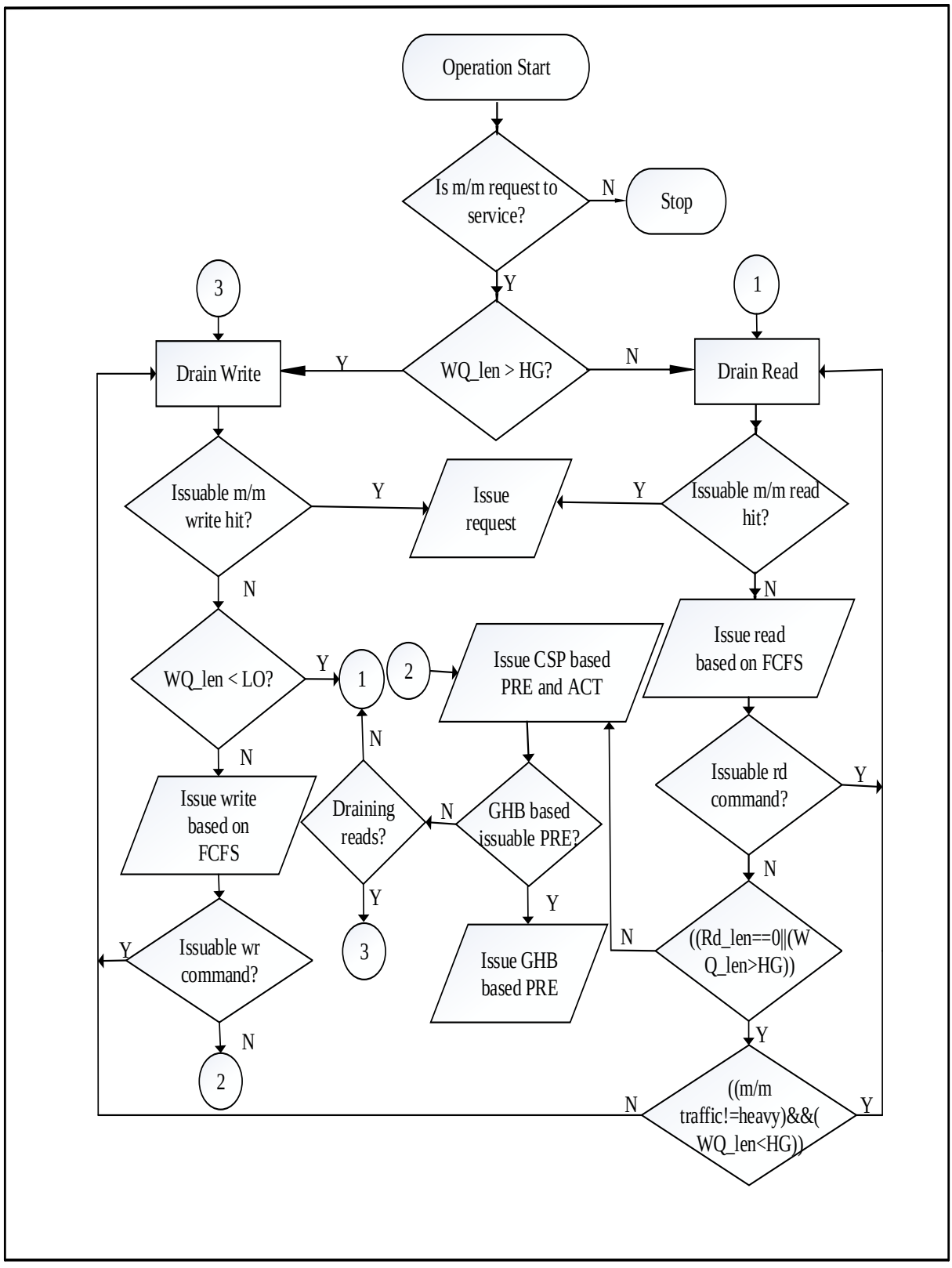

Figure 5.1 Flow chart of proposed scheduling approach

When no read requests arrive in certain amount of time or when write queue gets full up to high watermark, write drain mode is activated. Delayed write drain mode is applied adaptively

according to memory request traffic. If memory request traffic is heavy, then, in this scenario it is better to service write requests immediately after read mode rather than waiting for more read request's arrival. Whereas, when there is no massive read/write traffic then delayed write drain is activated. Whether the memory request traffic is high or low, it is dependent on historic request frequency. Requests are observed for some CPU cycles (in this paper 10k CPU cycles), if number of request for a particular channel exceeds a threshold limit (200 in our approach), delayed write drain is activated. In proposed scheduler read requests and write requests are served in bursts to avoid delay encountered due to switching of bus direction. Scheduler enters in write drain mode when write queue is about to be full, i.e., reaches high watermark. In write drain mode writes are serviced in FR-FCFS manner until lower threshold limit (low watermark) is met. When in write drain mode no write request can be issued in any memory cycle then in that idle cycle issuable (precharge or activate) commands are issued in accordance with pending read requests. In case of read mode, the scheduler issues read hit requests first then older read requests are served. If no pending read request can be issued in any memory cycle, then, in that idle memory cycle non-conflicting issuable write commands (precharge command) are issued. If there is no pending read requests in read queue and if delayed write drain can be activated, scheduler waits for upcoming read requests, otherwise enters drain write mode. The other entities of scheduler issues commands only when there are no command issuable by base scheduler. The flow chart of proposed scheduler is depicted in Figure 5.1.

5.1.2 *Stride Prefetcher based Precharge/Activate Command Predictor*

Stride prefetcher is implemented by maintaining a constant stride prefetch (CSP) table for holding stride related history. CSP table contains previous stride value, last address accessed and a found bit, Figure 5.2. Existing constant stride between memory access requests having same value of instruction program counter are stored in it. When the scheduler goes through the read queue during read mode and write queue during write drain mode, on encountering a memory request having same stride value as previous stride value in CSP table, the found bit is set.

Previous Stride Value	*Last Address Accessed*	*Found Bit*

Figure 5.2 Constant Stride Table

Initially, all constant stride table entries are set to zero when the scheduler goes through read/write queue for detecting issuable row hits. After issuing issuable row hits when the scheduler looks through read/write queue for checking any other command issuable during that phase constant strides are detected in current read queue or write queue. If no issuable command is found, the scheduler then goes through CSP table entries. With respect to every entry for which found bit is set, scheduler issues precharge or activate commands to the rows of banks at x+y, x+2y, …., x+dy physical addresses, where, x is last address accessed value in CSP table, y is previous stride value in CSP table and d is degree of prediction.

If a memory access prediction corresponds to different channel, then it is stored in separate array and can be read by scheduler if there is no issuable command while issuing commands in that channel.

5.1.3 *Prefetcher based Close Page Policy*

In this paper, prefetcher used global history buffer [89], which is earlier used for data cache prefetching. GHB stores prefetch history. GHB prefetcher is implemented in two levels, Figure 5.3.

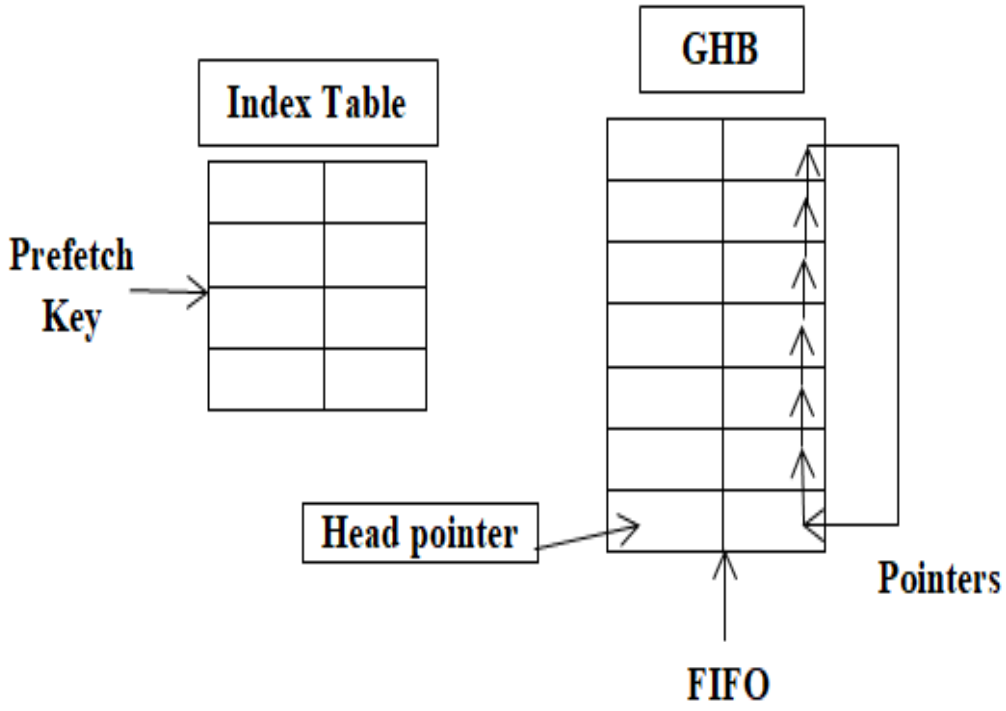

Figure 5.3 Global History based Close Page Predictor

- Index Table: Index table has capacity of 1024 entries and references to GHB. Index table can be accessed with a key, eight least significant bits of the key are xor of instruction program counter and memory address of memory request, top two most significant bits corresponds to thread identifier of memory request.
- Global History Buffer: GHB is a FIFO table having 512 entries and is maintained as circular buffer. It holds 512 most recent memory requests. Each entry in GHB table is linked to previous entry having same index table key using a pointer.

GHB is maintained for read requests. At onset of every memory cycle, prefetcher adds new request in GHB table. When there is no command that can be issued, GHB prefetcher tries to issue precharges to close the pages. Scheduler first accesses the head of the GHB table, i.e., the most recent request and then follows the link downwards to find older memory requests, if any. Scheduler closes the pages of older memory requests which do not conflict with pending read/write command.

5.2 Methodology

In this section we provide the pertinent details about simulation environment, simulated workloads and evaluation metrics used during quantitative analysis of proposed scheduling policy.

The proposed scheduling policy is built on device level main memory commands simulator named USIMM. Proposed scheduling policy is simulated under two memory system configurations, one with single channel containing eight banks per rank and four ranks per channel, mem-config-1. In this memory configuration address mapping is enabled in order to maximize row hits. Second memory system setup simulated for evaluation supports four channels and each channel is further configured same as first configuration, mem-config-4. In second memory configuration address mapping is set to 0 so as to increase memory access parallelism.

The proposed scheduling policy is evaluated in terms of performance and energy consumption for chip multiprocessor systems using workloads consisting of multithreaded programs. The workloads are constituted with trace files of PARSEC [87] benchmark suite representing applications from diverse domains such as image processing, animation physics, financial analysis etc. Trace files are obtained using Simics [99] and are described in Table 5.1.

The performance and energy consumption of proposed scheduling policy is analysed in terms of total execution time, row buffer hits, energy-delay product and total memory system power consumption. The evaluation is made under various multicore environments, i.e., 2-core, 4-core and 8-core.

Table 5.1 Benchmark Description

Benchmark	***Application Domain***
Blackscholes	Financial Analysis
bodytrack	Computer Vision
Facesim	Animation Physics
Ferret	Similarity Search
Fluidaminate	Animation Physics
Streamcluster	Data Mining
Swaption	Financial Analytics
Canneal	Engineering
transaction processing workload	Server

5.3 Result Analysis

Quantitative analysis of proposed scheduling approach is made to check the impact of varying core count and memory configuration on system's performance and energy consumption. For comparative analysis, we compare proposed approach with two existing schedulers, scheduler-1: FR-FCFS [30] and scheduler-2: scheduling approach proposed in [95]. Comparative analysis is made in terms of execution time, row hits, power consumption and energy-delay product.

5.3.1 *Total Execution Time*

The simulation trend seen in Figure 5.6, reveals that overall performance of proposed scheduler is better than scheduler-1 and scheduler-2. Proposed scheduling policy shows significant reduction in execution time required by workloads to complete their execution. In 1-channel environment proposed scheduler reduced total execution time when compared scheduler-1, Figure 5.4. For memory configutaion-1, 0.037% increase in total execution time is observed when compared to scheduler-2, whereas, 45.63% decrease is found when compared to scheduler-1.

In 4-channel memory configuration, proposed approach performed best among all simulated policies in terms of total execution time, Figure 5.5. Proposed scheduling approach took least time to complete execution of workloads in memory system comprised of 4 channels. Proposed scheduler took 0.34% less time as compared to scheduler-2, in 4-channel memory configuration. However, when comparison is made with respect to scheduler-1, 0.71% decrement in execution time is observed to complete execution of simulated workloads.

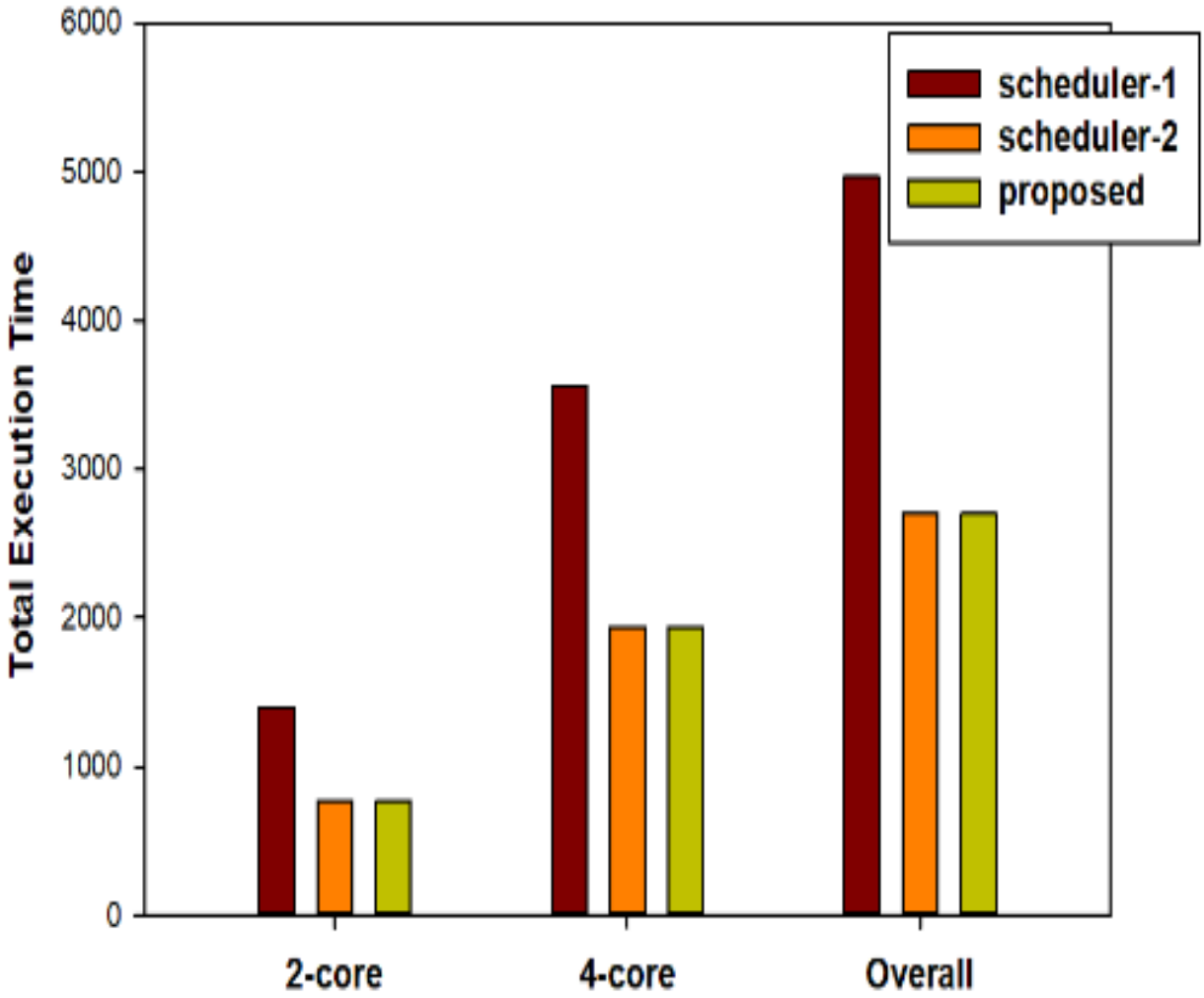

Figure 5.4 Total Execution time based comparison for mem-config-1

As a whole, Figure 5.6, proposed scheduling policy shows significant reduction in total execution time. When compared to scheduler-1, 26.88% reduction in total execution time is observed. Whereas 0.16% less execution time, when compared to scheduler-2 scheduling policy.

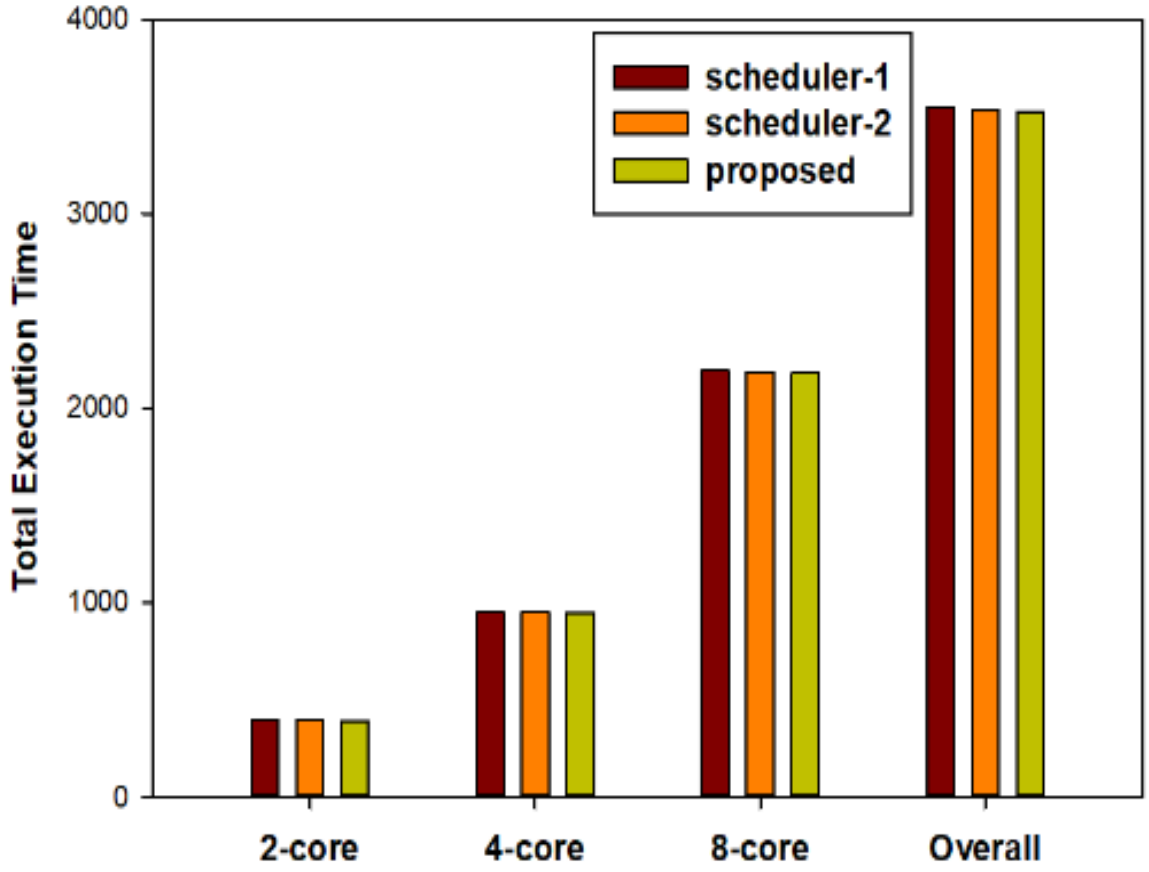

Figure 5.5 Total Execution time based comparison for mem-config-4

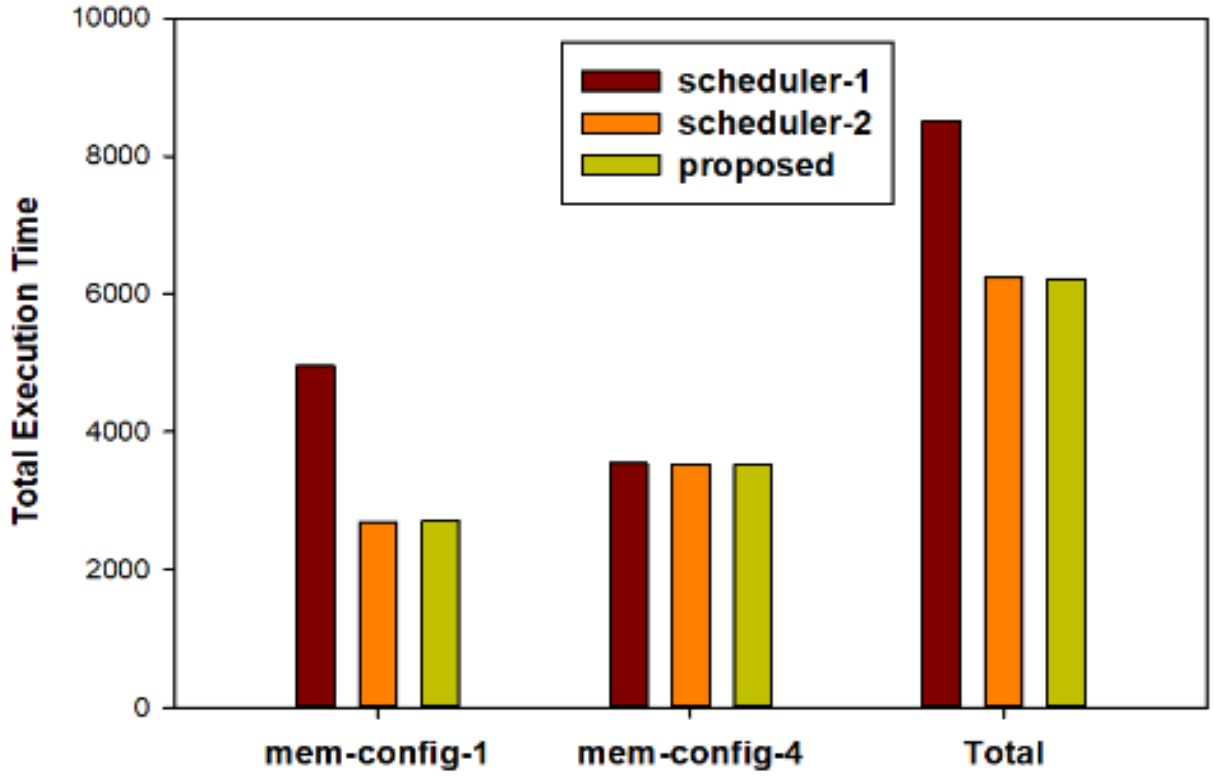

Figure 5.6 Total time consumed during execution

5.3.2 *Row Hits*

For both simulated memory configurations, scheduling policy proposed in this paper performed better than all simulated schedulers under varied core environment in terms of row hits, Figure 5.7. In terms of read row hits, proposed scheduler performed better than scheduler-1 and its performance is comparable to scheduler 2. Proposed scheduler shows, 1.01%, overall increase in row hits when compared to scheduler-1 and in mem-config-1.

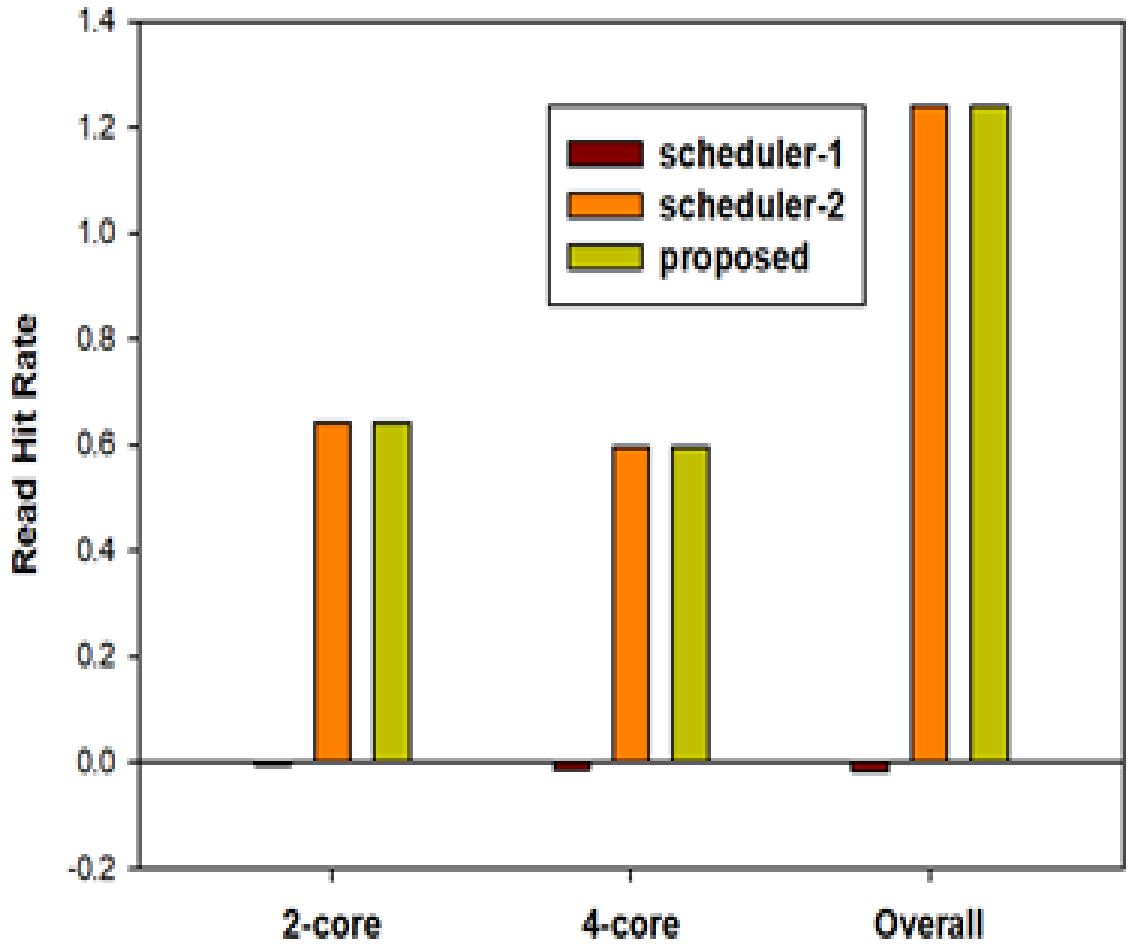

Figure 5.7 Read Page Hit Rate for mem-config-1

5.3.3 *Energy-Delay Product*

The simulation trends, Figure 5.10, obtained for energy-delay product can be summarized as, proposed scheduler reduces energy consumption in both memory configurations for all simulated core environments. Overall, 62%, 56.4%, 0.93%, decrease in energy consumption is observed in 2-core, 4-core and 8-core environment in comparison to scheduler-1. From the results obtained it is observed that proposed scheduler out-performs all simulated schedulers in terms of energy-delay product.

The results have shown in Figure for Energy Delay Product reveals that proposed scheduler's performance is best among all memory access scheduling policies (i.e., scheduler-1 and scheduler-2) for configuration-1 and configuration-2, memory configurations.

Figure 5.8, depicts the simulation trend obtained in terms of energy delay product in memory configuration-1, facilitating 1 channel in memory system. As per obtained results, proposed scheduler performed best among all simulated policies. In memory configuration-1, proposed scheduler reduced energy consumption by 0.065% and 69.3% when compared to scheduler-2 and scheduler-1, respectively, while maintaining performance of the system more as compared to all simulated scheduling policies.

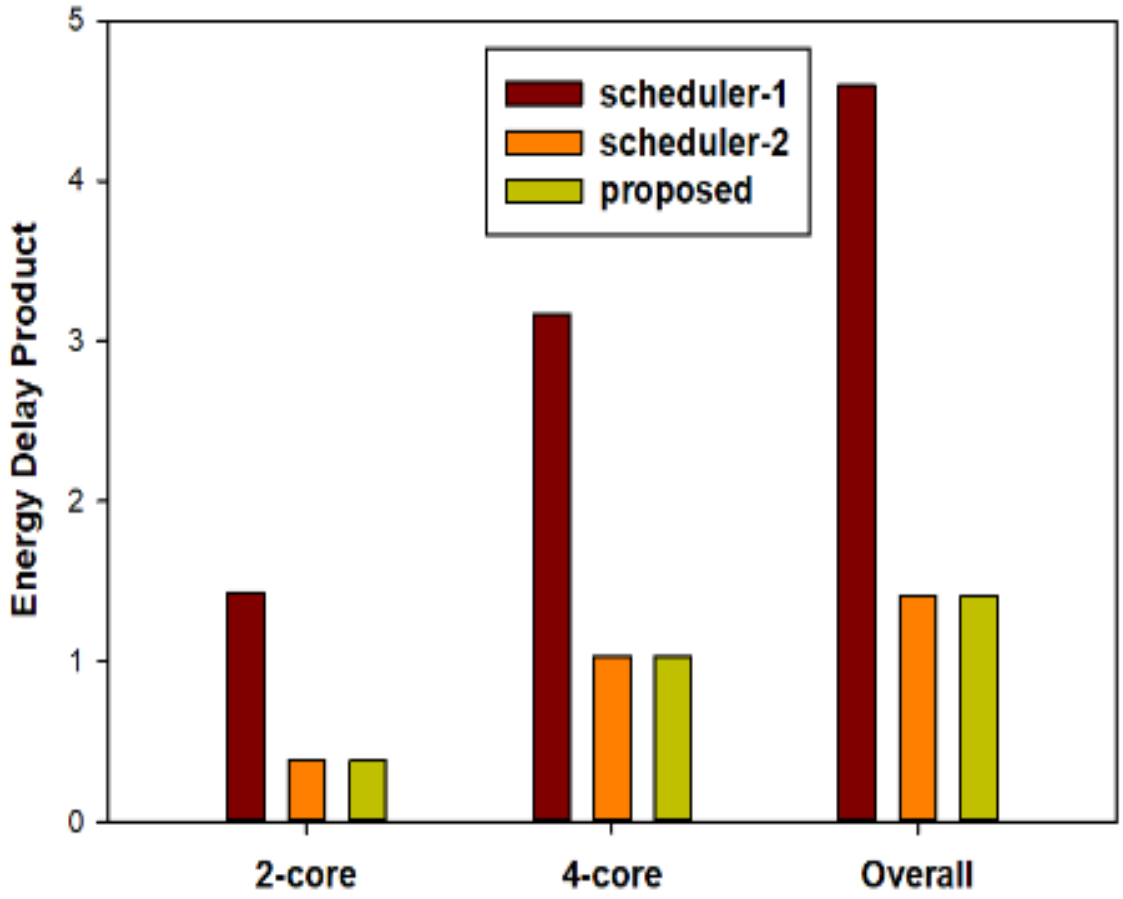

Figure 5.8 Energy Delay Product based comparison for mem-config-1

Figure 5.9, depicts the simulation trend obtained in terms of energy delay product in memory configuration-2. The simulation trend shown in Figure 5.9, reveals that in multi-channel environment proposed scheduling policy outperforms all simulated schedulers. Using configuration-2, in comparison to scheduler-1 and scheduler-2, proposed scheduler reduced energy delay product by 3.5% and 0.41%, respectively.

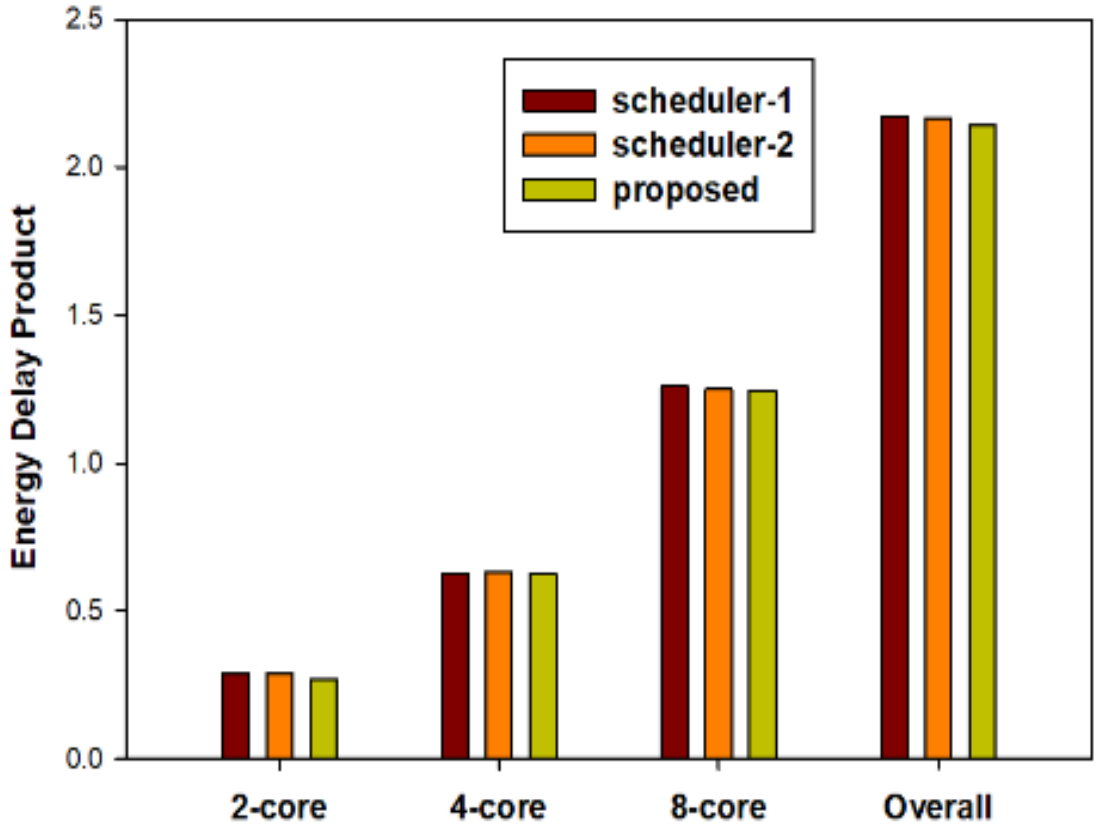

Figure 5.9 Energy Delay Product based comparison for mem-config-4

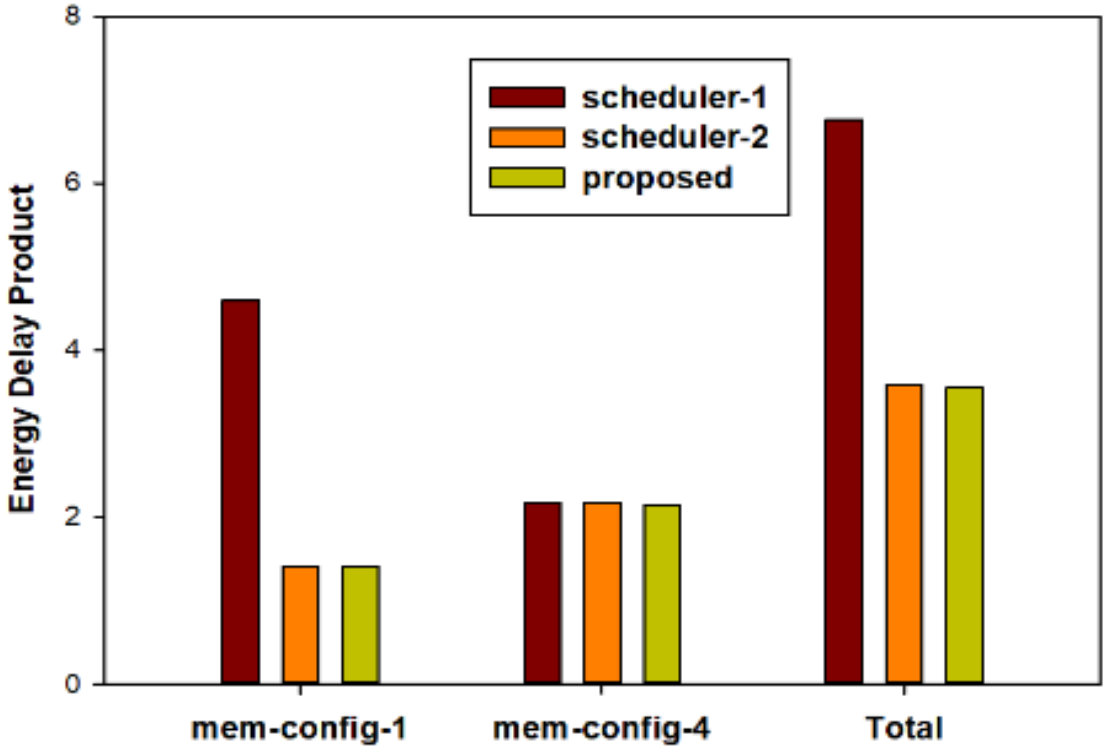

Figure 5.10 Total Energy Delay product

In total, proposed scheduler has shown most reduction in energy delay product as compared to all simulated policies. 47.5% and 0.84% reduction in energy delay product is observed with respect to scheduler-1 and scheduler-2, respectively.

5.3.4 *Total Memory System Power Consumption*

Fig. 5.13, depicts the performance of proposed scheduling approach in terms of memory power consumption in comparison to selected schedulers chosen for comparative study under multi-core environment and both memory sub-system setups. By result analysis it is depicted that proposed scheduler consumed more power than all simulated scheduling approaches. This is because of the increase caused in hardware of the system to implement prefetchers. In 1-channel memory configuration, Figure 5.11, 25.78% increase in memory power consumption is observed in comparison to scheduler-1 and 0.052% increment is observed when compared to scheduler-2. In 4-channel memory configuration, Figure 5.12, 0.42% increase in power consumption is observed with respect to scheduler-2. Total 8.6% increment in power consumption is found when compared to scheduler-1 and 0.2% increment is observed when compared to scheduler-2.

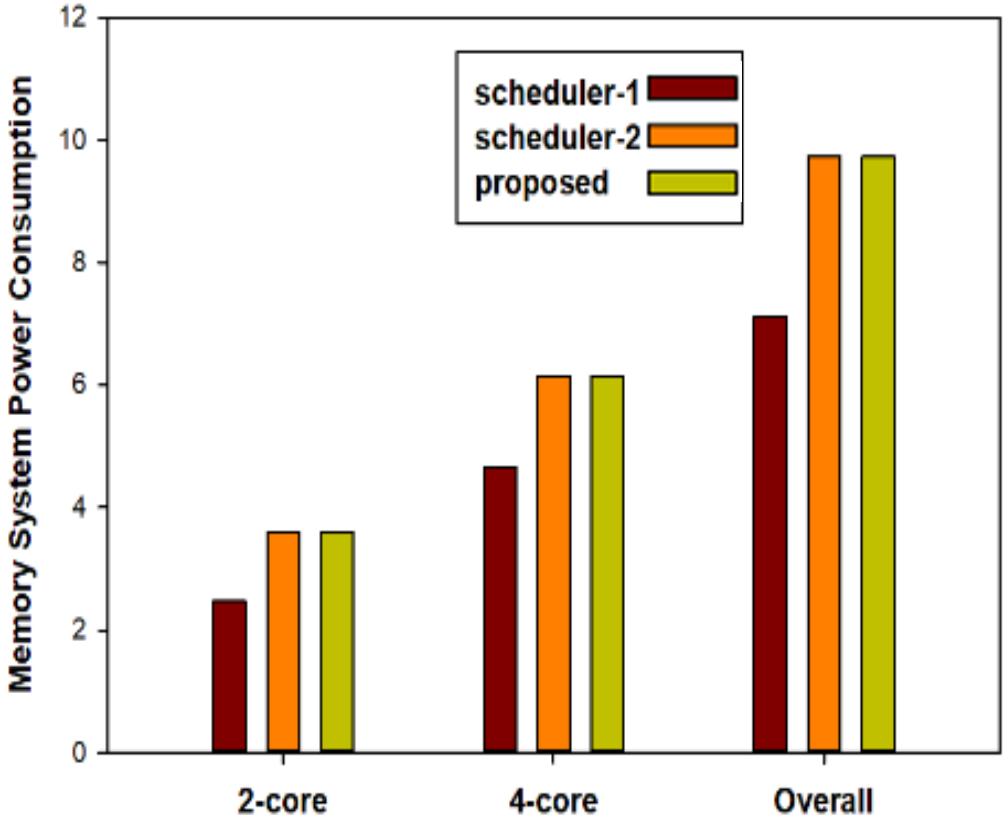

Figure 5.11 Memory System Power Consumption based comparison for mem-config-1

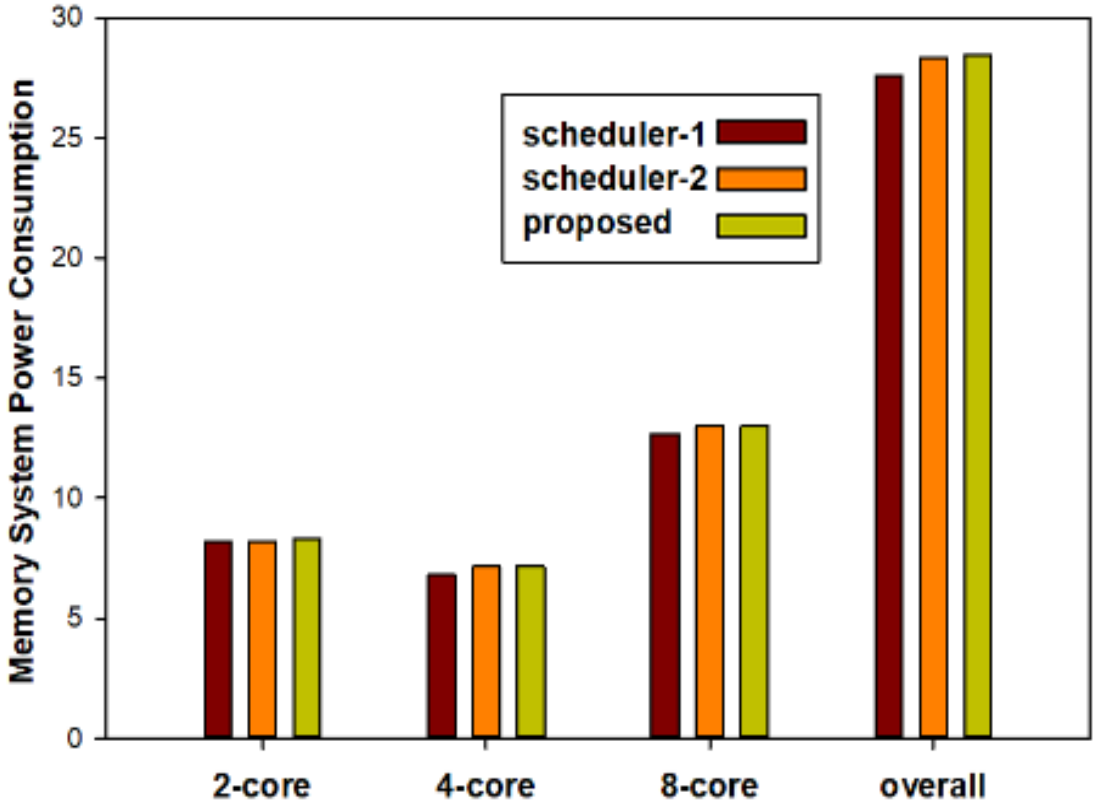

Figure 5.12 Memory System Power Consumption based comparison for mem-config-4

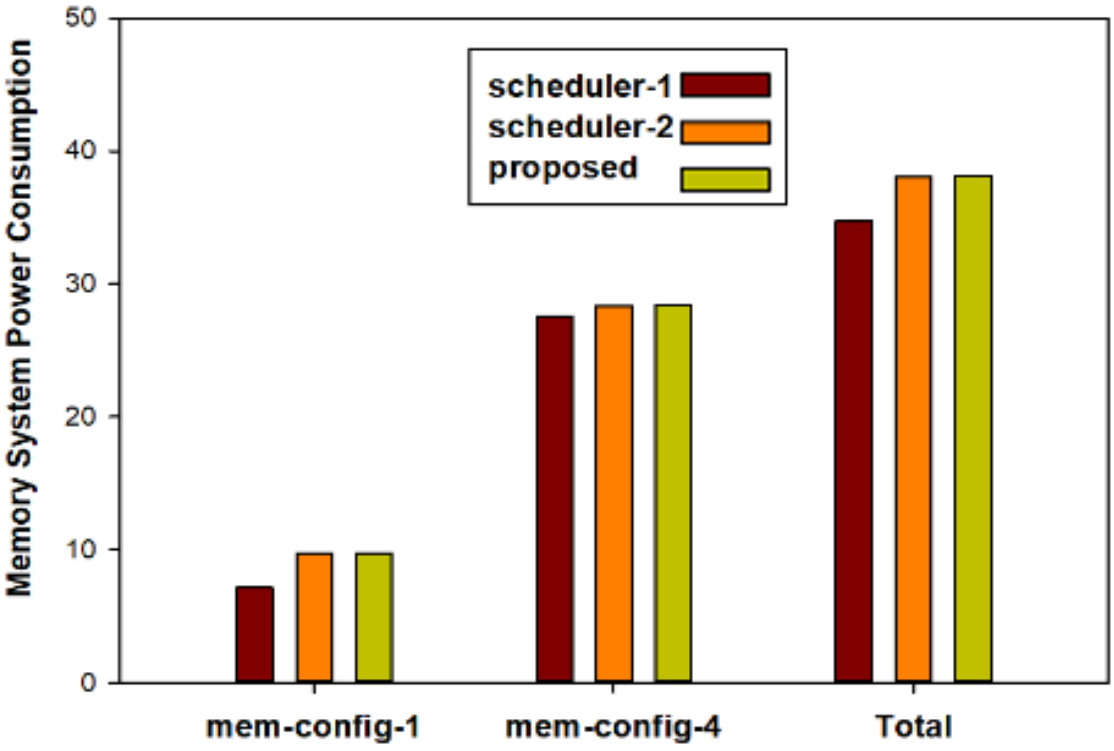

Figure 5.13 Total power consumed by Memory System

5.4 Conclusion

In this section, we propose a memory scheduler that issues commands on the basis of delayed write drain and FR-FCFS policy to the memory. Scheduler uses prefetching techniques that are earlier used for data cache prefetching. Prefetching techniques are used for predictive row precharge/activate operations. From implementation aspect, the scheduler is easy to implement, and its memory requirement is also very less. In comparison with scheduler-1 memory scheduling approach, overall, proposed scheduling approach rationalizes total execution time by 26.8%, 0.161% in mem-config-1 and mem-config-4, respectively, whereas, in comparison to scheduler-2 our scheduling policy decreases total execution time by 0.34% in mem-config-4. We can see that proposed scheduler performed better in terms of execution time in multi-channel configuration than single channel memory system. In terms of power consumption performance of proposed scheduler is better in comparison to other simulated policies. Proposed scheduler consumed less amount of power considering the increase in hardware. Also, the proposed scheduler successfully achieved increased row hits than scheduler-1 and scheduler-2. This increase in performance is because proposed scheduler issues speculative precharge and activate commands. Moreover, row hit read/write commands are prioritized over memory requests. Proposed approach consumed least amount of energy among all simulated policies, i.e., overall, 47.54%, 0.827% less in comparison to scheduler-1 and scheduler-2, respectively. Delay in entering in write drain mode also helps to achieve increased performance in terms of energy consumption, row hits and execution time. In future, proposed approach can be attached with other scheduling policies to make it more efficient. Further, appropriate scheduling mechanisms can also be incorporated to improve performance degradation in terms of energy as well power consumption due to refresh operations.

CHAPTER 6

MANAGING REFRESH INDUCED PENALITIES IN DRAM BASED MAIN MEMORY SYSTEM

Increased use of big data applications results in increasing demand for larger and faster main memory system. DRAM based main memory system is chosen to satisfy these growing needs because of its low service latency as well as high density. But DRAM cell being volatile in nature requires periodic refreshes to retain its data. The periodic refresh operations negatively affect the performance and power consumption of the system. Earlier, refresh caused power and performance overheads were not paid much attention but with increased capacity and speed of DRAM based main memory system, these overheads have also increased significantly. Increased usage of memory intensive applications, increase in number of cores for faster and efficient processing and growth in I/O speed capabilities have resulted in significant development in main memory capacity and bandwidth availability. Computing systems used these days use DRAM based main memory system. DRAM is preferred over SRAM (Static Random Access Memory) because of its comparatively higher density and preferred over non-volatile memory technologies like phase change memory, MRAM and flash memory etc. because it is having lower latency, higher tolerance and bandwidth. Advancement in DRAM technology for increased capacity speed resulted in inclined power and performance overhead.

6.1 Proposed Scheduling Policy

In proposed scheduling policy refreshes are managed to reduce refresh caused energy and performance overhead of the DRAM based main memory system. Refreshes are managed by issuing write commands to banks that are undergoing refresh operation. By parallelizing write commands with refresh operation. By parallelizing write commands with refresh operation memory cycle that was earlier wasted in conventional scheduling policy for refreshes is now utilized to service memory write requests. By parallelizing memory writes with refresh operations performance overhead is reduced by reducing memory write service latency. In proposed scheduling policy first read requests are prioritized over write requests by always entering in drain read mode unless write queue is about to be full. If write queue is about to be full, *i.e.*, write requests are above high watermark, scheduler starts issuing write requests to be serviced until write requests get less than low watermark. When in write drain mode, row hit

write requests are serviced first then other write requests. If there is no other row hit write request available then writes are issued in FCFS manner. If scheduler is in drain write mode and there is no issuable write command then precharge and activate commands based on constant stride prefetching are issued to open and close rows. If constant stride prefetcher based precharge and activate commands are not issuable then global history buffer based precharge command is issued to close the rows. When in read mode, first read hit requests are served then if no read hit request is left, reads are served in first come first serve manner. If read queue is about to be empty and write queue is not yet full then in this case if memory traffic is also not heavy then instead of entering in drain write mode immediately, memory controller stays in read mode and waits for upcoming read requests. Here, again read requests are prioritized over write requests because reads are more critical for system's performance. If write queue is full or memory traffic is not heavy then scheduler simply enters in drain write mode and starts servicing write requests. If read queue is not empty and there is no issuable read requests then constant stride prefetcher based precharge and activate commands are issued. If constant stride prefetcher based commands are also not issuable then global history buffer based precharge commands are issued speculatively.

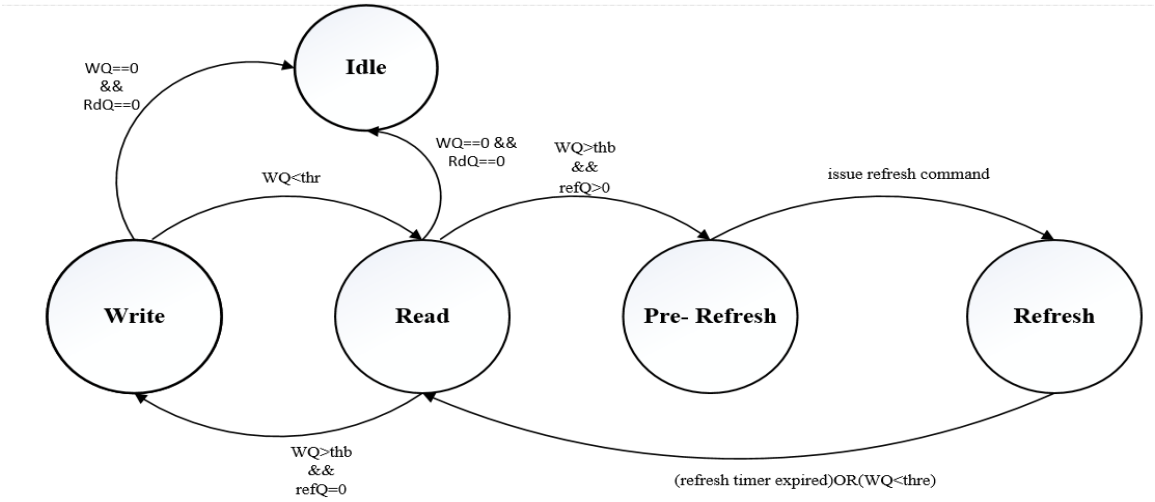

Figure 6.1 Memory Controller Transition States

In proposed scheduler along with read drain mode and write drain mode two more controller states are added, *i.e.*, pre-refresh and refresh states. In pre-refresh state memory controller stops issuing write command to the banks that are not refresh target in upcoming refresh cycle. In refresh state the banks that are not refreshed are capable servicing issued write requests while banks undergoing refresh operation do not service any read/write command. Figure 6.1,

describes the transition criteria and transition between memory controller states. The flow chart for proposed scheduling policy is detailed in Figure 6.2.

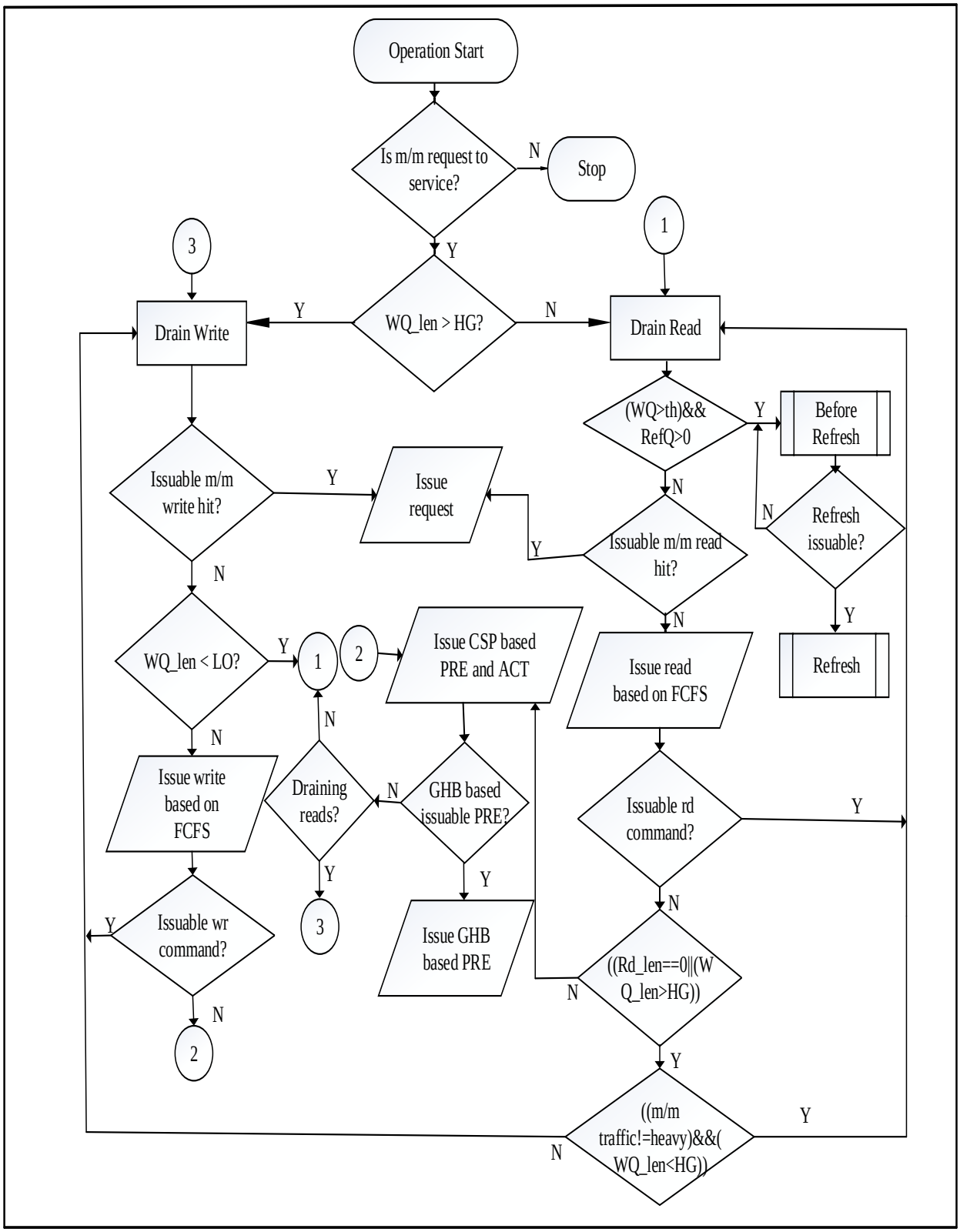

Figure 6.2 Proposed Scheduling Approach

6.2 Evaluation Methodology

The proposed scheduling policy is simulated on DRAM based main memory scheduler named USIMM. Simulation environment considered for comparative evaluation of proposed evaluation is presented in Table 6.1. The proposed scheduler is comparatively analyzed in comparison to existing scheduling policies, scheduler-1[100], scheduler-2[101] and scheduler-3[102] in terms of energy delay product, total execution time, maximum slowdown time and total memory system power consumption.

Table 6.1 Memory Configuration

Parameters	Configuration
Processor clock speed	3.2GHz
Processor ROB size	160
Memory bus speed	800 MHz (plus DDR)
Memory channels	4
Ranks per channel	2
Banks per Rank	8
Cache lines per row	128

6.2.1 *Energy-Delay product*

In terms of EDP performance metric, proposed scheduling approach proved to be best among all simulated scheduling policies, Figure 6.3, reveals the results and presents that proposed scheduler has rationalized EDP by 3.3% with respect to scheduler-1. With respect to scheduler-2 and scheduler-3, proposed scheduling policy has decremented EDP by 1.33%.

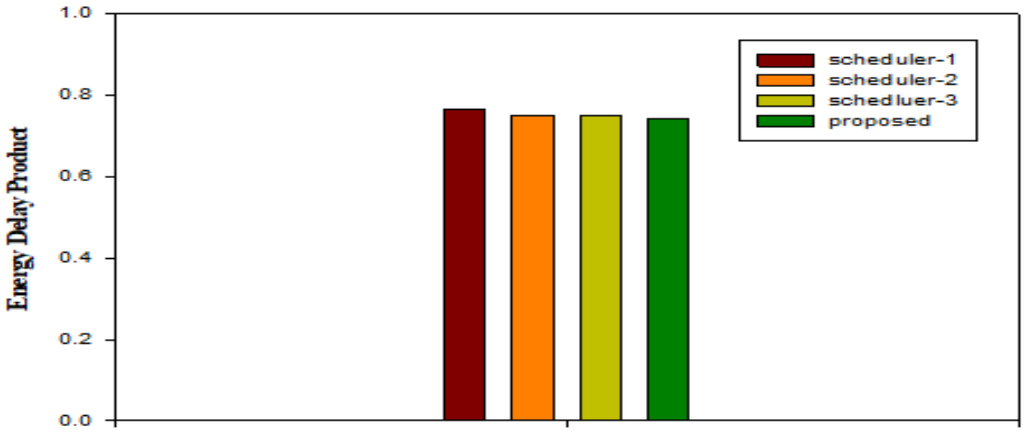

Figure 6.3 Comparison based on Energy-Delay Product

This improvement in terms of energy-delay product stems due to considerable reduction in number of operations achieved due to prefetching based pre-charge and activate command scheduling. Reductions in number of operations further results in reduced execution time which may further lead to curtailed energy consumption.

6.2.2 *Total execution time*

The simulation scenario presented in Figure 6.4, depicts that proposed scheduling approach has shown best performance among all simulated scheduling approaches in terms of total execution time. In comparison to scheduler-1, scheduler-2, scheduler-3, proposed scheduling policy reduces the total execution time by 1.22%, 0.95% and 0.6% respectively. Parallelizing writes and pre-issuing constant stride prefetch based precharge and activate commands helps to reduce memory cycles required to complete execution of simulated workload. Servicing write requests along with bank refreshes utilizes idle memory cycle time and benefits in achieving reduced execution time of programs.

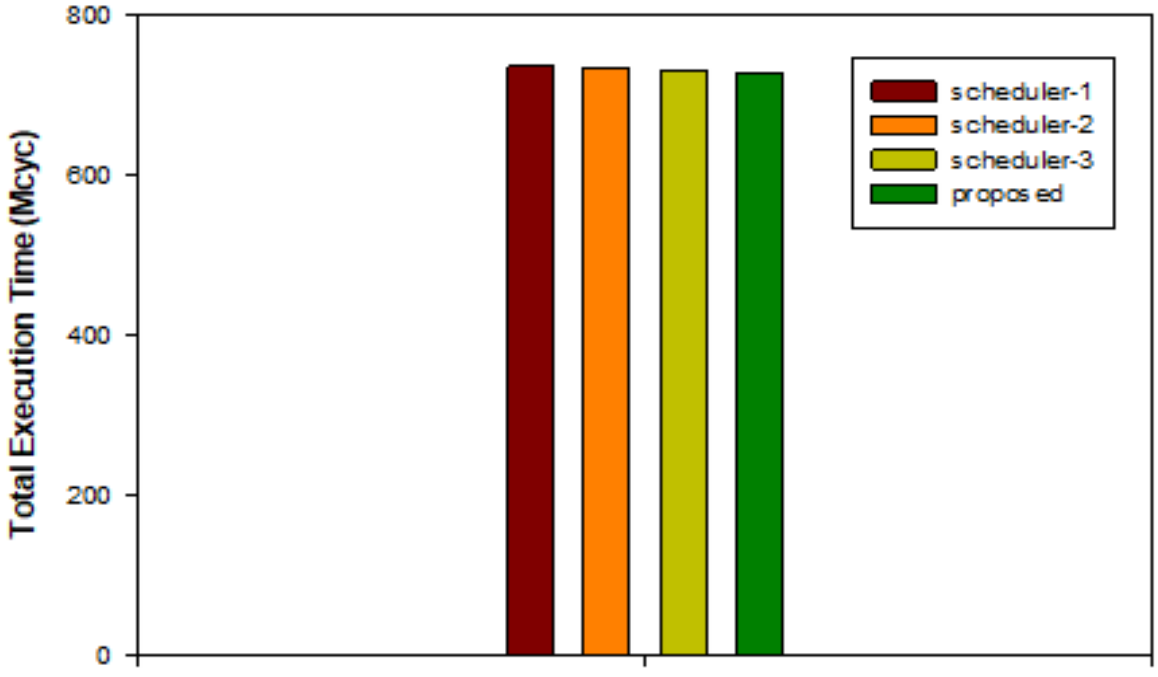

Figure 6.4 Comparison based on Total Execution Time

6.2.3 *Maximum Slowdown time*

Figure 6.5, reveals the results obtained in terms of maximum slowdown time. The simulation trend obtained presents that for maximum slowdown proposed scheduler is better than scheduler-1 and scheduler-3. With respect to scheduler-1, proposed scheduling approach has reduced maximum slowdown time by 0.21%, whereas in comparison to scheduler-3 a drop of

1.04% in terms of maximum slowdown time is observed. However, when compared to scheduler-2, 0.3% marginal increase in maximum slowdown time is observed, because scheduler-2 ensures fairness among threads, whereas in proposed scheduling policy fairness is not key concern.

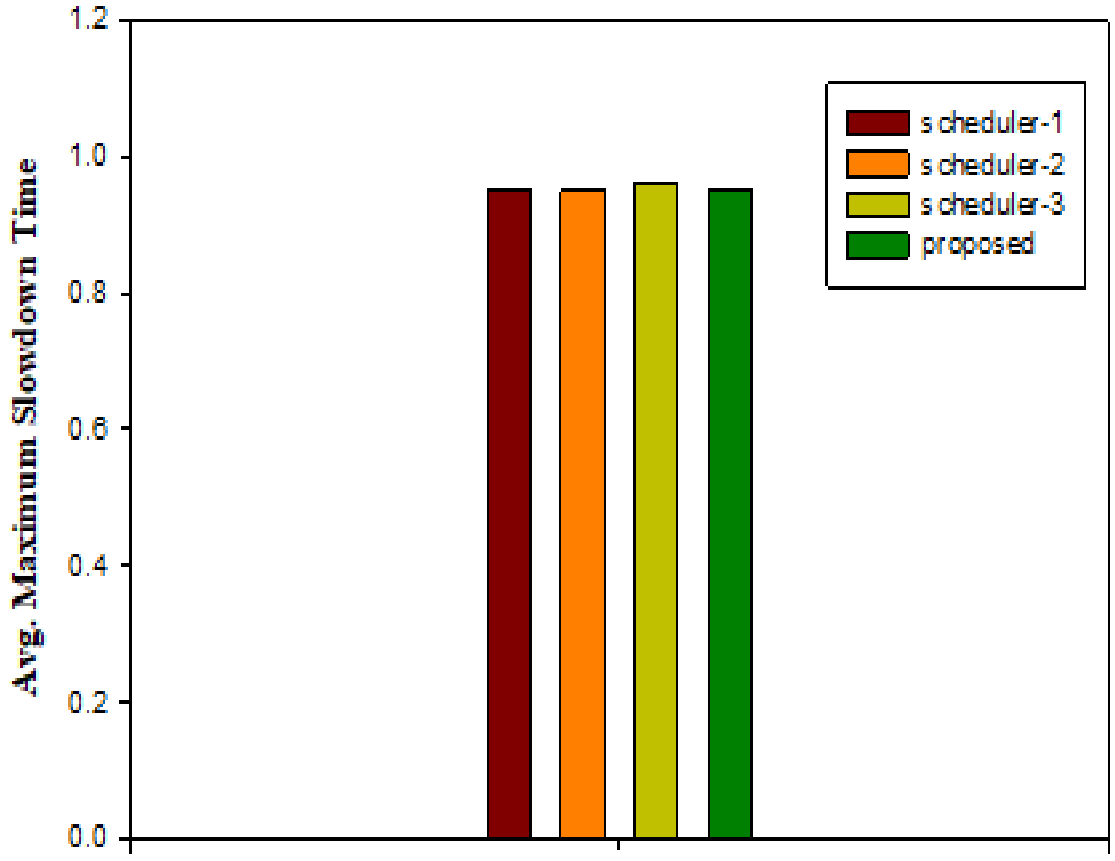

Figure 6.5 Comparison based on Average Maximum Slowdown Time

6.2.4 *Total Memory System Power consumption*

Proposed memory scheduling approach improves system's performance for all simulated workloads under considered memory configuration in comparison to simulated schedulers, *i.e.*, scheduler-1, scheduler-2 and scheduler-3. As shown in Figure 6.6, total memory system power consumption is marginally increased with respect to scheduler-2 and scheduler-3. Whereas, when compared to scheduler-1, 1.23% deduction in total memory system power consumption is observed. The increase in power consumption is due to constant stride prefetcher and global history buffer implementation. The increased components consume more power for prefetching operations which introduces marginally increased power consumption overhead. But we want to evaluate proposed scheduling policy in terms of its impact on both energy as well as performance (service latency) and this performance goal is captured by energy-delay

product and in terms of EDP proposed approach has shown best behaviour among all simulated environment.

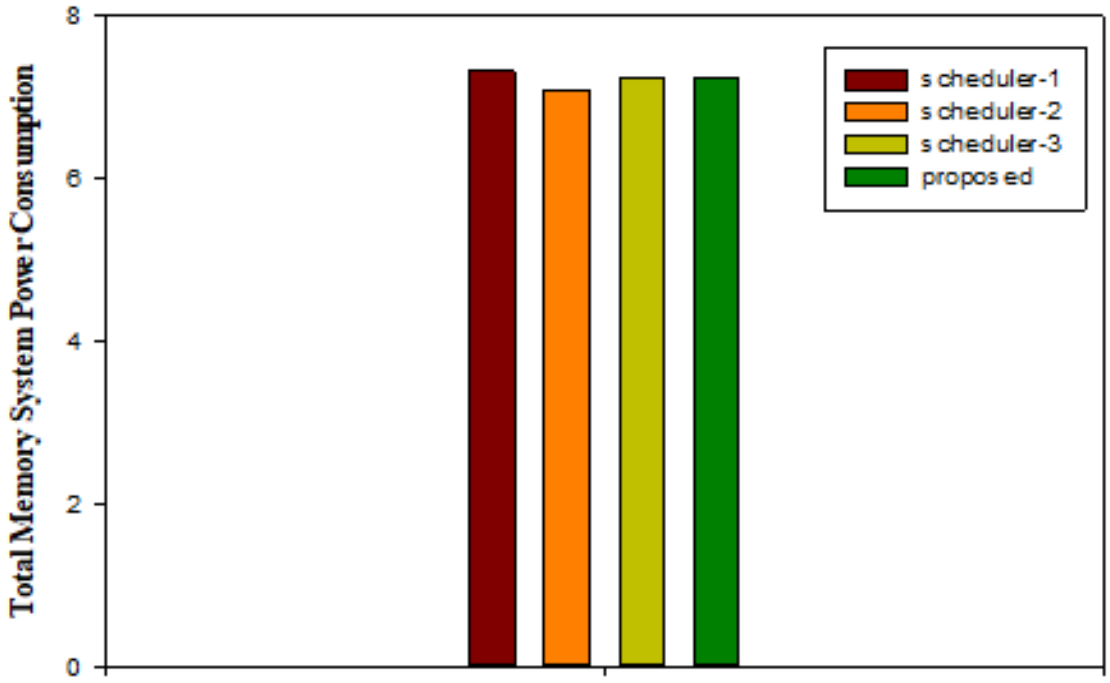

Figure 6.6 Comparison based on Total Memory System Power Consumption

6.2.5 *Conclusion*

The scheduler proposed in this chapter utilizes delayed write drain policy to prioritize read requests over writes, uses prefetching based precharge and activate operations and parallelizes write operations with refresh operation to compensate idle memory cycle time. Reads are preferred over write requests to achieve performance benefits as read affects overall performance of a computing system significantly than write operations. For prefetching constant stride prefetching and global history buffer based prefetcher is used. Whereas to implement parallelized write operations with reads in addition to read and write states, two more controller states are added. By parallelizing writes with refresh operation, memory cycle that earlier wasted for refresh operation in conventional scheduling approach, is now utilized for servicing memory write requests, which further helps to achieve reduced execution time for any program. Prefetching helps to reduce number of operations required to complete execution of a program. Reduced number of operations causes reduced execution time which may further benefit in terms of energy delay product. But extra components introduced in terms of prefetcher causes comparatively increased power consumption. But the increase in power consumption is marginal which do affect the energy consumption and performance of a system.

Proposed scheduler reduces total execution time by 1.22%, 0.95% and 0.6% in comparison to scheduler-1, scheduler-2 and scheduler-3, respectively. In terms of energy delay product, 3.27%, 1.33%, 1.33% reduction is observed in comparison to scheduler-1, scheduler-2 and scheduler-3.We conclude that proposed scheduler is able to provide reduced execution time, and performance benefit in terms of energy consumption and service latency (EDP) while showing slight increment in power consumption.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

This chapter concludes the various topics covered in this thesis and future prospects of proposed scheduling policies. In this thesis, we discuss novel memory scheduling policies that further extend the state-of-art. Main memory is a major contributor in overall system's energy consumption. To rationalize main memory's energy consumption and to improve its performance memory scheduling has received significant attention. Through the policies we have proposed in this research work, we draw attention to scheduling policies that are capable of maximizing performance benefits and restraining the power consumption while meeting the timing intricacies of the DRAM devices.

7.1 Summary

We conclude work done in this dissertation as follows

- In chapter 3, proposed scheduling policy tries to maximize row buffer hits and fairness among simultaneously executing threads. Proposed scheduling policy is facilitated with features like prioritizing reads over write requests, preferring row hit memory requests over other memory accesses, improved bank level parallelism and ensuring fair execution environment by scheduling requests that block reorder buffer head. The proposed scheduling policy not only ensures improved performance and reduced energy consumption but also increased bank level parallelism and thread fairness. The conducted comparative analysis of proposed scheduling policy reveals that proposed scheduler improves energy-delay product by 4.76% in comparison to PBFS. The results found after simulation supports the fact that proposed scheduler provides more fair simulation environment as it decreases maximum slowdown time observed by executing threads by 1.63% when compared to PBFS and by 4.03% when compared to close page policy. The simulation trend observed for total execution time supports the fact that it decreases total execution time by 1.47% when compared to PBFS and by 4.82% with respect to close page policy. The results obtained supports that proposed scheduling policy improves row hits and fairness among threads during execution.
- The scheduling policy proposed in chapter 4 tries to reduce read-write switching of command and data bus. First, read-write switching is reduced by serving read requests

and write requests in bursts. Then by issuing issuable precharge and activate commands when in drain mode and at the end of it. One more factor incorporated to reduce read-write switching is by delaying entering in write mode when serving reads and read queue is empty and memory traffic is not heavy. By delaying entering in write drain mode not only read-write switching is curtailed but also reads are prioritized over write requests. The proposed scheduling policy is then evaluated in terms of total execution time, energy-delay product and maximum slowdown time to analyse the impact of proposed scheduling policy on memory system's performance and power consumption. The conducted comparative analysis reveals the fact that that proposed scheduler reduces total execution time by 0.6%, maximum slowdown time by 0.59% and energy-delay product by 4.2% when compared to PBFS.

- The scheduler proposed in chapter 5, improves memory access latency of memory commands by prefetching upcoming memory requests and issuing precharge and activate commands accordingly. The precharge command is issued to rows that are not to be used by upcoming requests while activate command is used to bring the rows in sense amplifier that are to be used by upcoming memory requests. Prefetching is constant stride based and global history buffer based. In addition to prefetching delayed write drain policy is incorporated to further enhance system's performance by prioritizing read requests. By evaluating the results obtained after simulating proposed scheduler it is found that proposed scheduling approach is able to reduce total execution time by 0.7% in comparison to scheduler-1 and by 26.8% in comparison to scheduler-2. In addition to reducing execution time, proposed scheduler is able to achieve performance benefit depicted by performance metric energy delay product.
- The scheduler proposed in chapter 6, reduces main memory's energy and performance overhead introduced due to DRAM refreshes. Proposed scheduler parallelizes memory write operation with memory refreshes in addition to constant stride prefetching and global history buffer based close page policy. Only writes are overlapped with refresh operation because writes are not critical for system's performance as reads are. So, we can delay issuing writes to banks that are not going to be refreshed in upcoming refresh cycle and servicing them when refresh operation is performed on other banks. Proposed approach is able to achieve 1.23% power consumption reduction and 3.27% curtailed energy consumption when compared to scheduler-1.

7.2 Future Scope

In this section we elaborate that how emerging trends may lead to various applications of memory scheduling strategies.

7.2.1 Scheduling for Heterogeneous Platform

Advancement in technology have facilitated the researchers to architect computing system in which different types of blocks lie on same die. For instance, AMD's fusion series of APU's (Accelerated Processing Units) integrates a GPU along with CPU having multiple cores. Integration of GPU and CPU has led researchers to think about memory architecture in more detail. In Fusion series, the GPU as well as CPU shares a common memory controller. Both GPU and CPU have different requirements in terms of memory. GPU(s) are more memory latency tolerant than CPUs while requiring more bandwidth. A memory scheduling policy should be able to satisfy the needs of both clients. *i.e.*, C.P.U. as well as G.P.U. We plan to investigate the applicability of proposed scheduling policy in such heterogeneous environment where C.P.U. and G.P.U. work together contending for same memory resource and having different expectation from memory system.

7.2.2 Scheduling for Mobile Devices

A lot of emphasis have been laid on memory controller's design with respect to large servers. However, mobile devices are growing very fast and hence they require equal attention for efficient performance and energy consumption. With respect to mobile devices, first memory access scheduler should be simple in order to save chip area, second scheduler should be more energy efficient than performance oriented. So, memory scheduling techniques should focus on reducing energy consumption but performance drop should not be large.

7.2.3 Scheduling for Emerging Non Volatile Memory Technologies

Due to increased demand of large capacity of main memory system as well as restrained energy consumption, researchers these days are exploring the possibility of using non-volatile memory technologies such as Phase Change Memory, memristor etc. as an replacement to DRAM. To benefits of both memory technologies, i.e., DRAM as well as non-volatile memory technology researchers are opting hybrid main memory system also. So, a memory scheduling policy should be able to satisfy future memory trends also. In order to facilitate so, proposed scheduling policies can be analyzed to check their impact in hybrid (DRAM + non-volatile memory based) main memory system.

7.2.4 Scheduling for Hybrid Memory Cubes

Hybrid memory cube [103] is a memory technology in which multiple layers of DRAM are stacked on a logic layer. The HMC device is connected to processors. The problem in using HMC is distribution of work responsibilities between memory controllers lying on chip as well as one on the logic layer of HMC. To solve this dilemma one approach is implementation of transaction scheduler in logic layer of HMC. This enables on-chip memory controller to decide thread priorities and command scheduler to decide which policies are to be implemented for HMC DRAM layers. Scheduling policies should be able to support such computing environment. Work can further be extended to support HMC memory technology.

REFERENCES

1. C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. Keller, “Energy management for commercial servers,” IEEE Computer, 36(12):39–48, 2003.
2. JEDEC, “JEDEC solid state technology association." http://www.jedec.org.
3. JEDEC Solid State Technology Association, “DDR3 SDRAM specification,” Tech. Rep. JESD79-3E, Arlington, 2010.
4. JEDEC Solid State Technology Association, “DDR4 SDRAM specification,” Tech. Rep. JESD79-4, Arlington, 2012.
5. Micron, “DDR4 - Advantages of Migrating from DDR3," 2013.
6. JEDEC, “Migrating to LPDDR3", 2012.
7. JEDEC, “Wide IO SDR Standard - JESD229", 2010.
8. A. Carroll and G. Heiser, “An analysis of power consumption in a smartphone,” In Proc. USENIXATC 2010.
9. O. Vargas, “Achieve minimum power consumption in mobile memory subsystems," March 2006, EE Times Asia.
10. C. H. van Berkel, “Multi-core for mobile phones," In Proc. DATE 2009.
11. T. Vogelsang, “Understanding the energy consumption of dynamic random access memories," In Proc. MICRO 2010.
12. H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," In Proc. ICAC 2011.
13. I. Hur and C. Lin, “A comprehensive approach to DRAM power management," In Proc. HPCA 2008.
14. E. Cooper-Balis and B. Jacob, “Fine-grained activation for power reduction in DRAM," IEEE Micro, vol. 30, May 2010.
15. S. Min, H. Javaid, and S. Parameswaran, “XDRA: Exploration and optimization of last-level cache for energy reduction in DDR DRAMs," In Proc. DAC 2013.
16. H. Huang, K. G. Shin, C. Lefurgy, and T. Keller, “Improving energy efficiency by making DRAM less randomly accessed," In Proc. ISLPED 2005.
17. K. T. Malladi, I. Shaeer, L. Gopalakrishnan, D. Lo, B. C. Lee, and M. Horowitz, “Rethinking DRAM power modes for energy proportionality," In Proc. MICRO 2012.
18. K. Fang and Z. Zhu, “Conservative row activation to improve memory power efficiency," In Proc. ICS 2013.

19. N. Aggarwal, J. Cantin, M. Lipasti, and J. Smith, “Power-efficient DRAM speculation," In Proc. HPCA 2008.
20. K. K.W. Chang, D. Lee, Z. Chishti, C. Wilkerson, A. Alameldeen, Y. Kim, and O. Mutlu, “Improving DRAM performance by parallelizing refreshes with accesses," In Proc. HPCA 2014.
21. M. Xie, D. Tong, Y. Feng, K. Huang, and X. Cheng, “Page policy control with memory partitioning for DRAM performance and power efficiency," In Proc. ISLPED 2013.
22. X. Li, G. Jia, C. Wang, X. Zhou, and Z. Zhu1, “A scheduling of periodically active rank of DRAM to optimize power efficiency," In Proc. Highly-Reliable Power-Efficient Embedded Designs 2013.
23. S. Mittal, “A survey of architectural techniques for DRAM power management," In Int. J. High Perform. Syst. Archit., vol. 4, December 2012.
24. S. Kim, S. Kim, and Y. Lee, “DRAM power-aware rank scheduling," In Proc. ISLPED 2012.
25. C. Ma and S. Chen, “A DRAM precharge policy based on address analysis," In Proc. DSD 2007.
26. M. Awasthi, D. W. Nellans, R. Balasubramonian, and A. Davis, “Prediction based DRAM row-buffer management in the many-core era," In Proc. PACT 2011.
27. ITRS, “International technology roadmap for semiconductors, 2009 edition.” [Online], Available: http://www.itrs.net/Links/2009ITRS/Home2009.htm
28. O. Mutlu and T. Moscibroda, “Stall-time fair memory access scheduling for chip multiprocessors,” in Proc. 40th Annu. IEEE/ ACM Int. Symp. Microarchit., 2007, pp. 146–16.
29. O. Mutlu and T. Moscibroda, “Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems,” in Proc. 35th Annu. Int. Symp. Comput. Archit., 2008, pp. 63–74.
30. S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, “Memory access scheduling,” in Proc. 27th Annual Int. Symposium Computer Architecture, 2000, pp. 128–138.
31. O. Mutlu, and T. Moscibroda, Microsoft Technology Licensing LLC, “Parallelism-aware memory request scheduling in shared memory controllers”, U.S. Patent 9,588,810, 2017.
32. J. Ahn, S. Yoo, O. Mutlu, and K. Choi, “PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture,” in ISCA, 2015.

33. D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu, “Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case,” in HPCA, 2015.
34. O. Mutlu and L. Subramanian, “Research Problems and Opportunities in Memory Systems,” SUPERFRI, 2014.
35. P. Nair, C.-C. Chou, and M. K. Qureshi, “A Case for Refresh Pausing in DRAM Memory Systems,” in HPCA, 2013.
36. Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, “A case for exploiting subarray-level parallelism (SALP) in DRAM,” in Proc. ISCA, 2012.
37. L. Subramanian et al., “The blacklisting memory scheduler: Achieving high performance and fairness at low cost,” in ICCD, 2014.
38. K. K. Chang, “Understanding and Improving the Latency of DRAM-Based Memory Systems,” Ph.D. dissertation, Carnegie Mellon Univ., 2017.
39. D. T. Wang, “Modern Dram Memory Systems: Performance Analysis and a High Performance, Power-Constrained Dram Scheduling Algorithm”, Ph.D. dissertation, Univ. of Maryland, 2005.
40. A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. Jouppi, “Rethinking DRAM design and organization for energy-constrained multi-cores,” in Proc. ISCA, 2010.
41. J. Stuecheli, D. Kaseridis, D. Daly, H. Hunter, and L. John, “The Virtual Write Queue: Coordinating DRAM and Last-Level Cache Policies,” In Proceedings of ISCA, 2010.
42. O. Mutlu and T. Moscibroda, “Stall-time fair memory access scheduling for chip multiprocessors,” in Proceedings of the 40th International Symposium on Microarchitecture, pp. 208–222, Dec. 2007.
43. B. Jacob, S. W. Ng, and D. T. Wang, “Memory Systems - Cache, DRAM, Disk”, 2008.
44. B. Fanning. Method for Dynamically Adjusting a Memory System Paging Policy, 2003. United States Patent, Number 6604186-B1.
45. O. Kahn and J. Wilcox, “Method for Dynamically Adjusting a Memory Page Closing Policy”, United States Patent, Number 6799241-B2, 2004.
46. T. Rokicki, “Method and Computer System for Speculatively Closing Pages in Memory”, 2002. United States Patent, Number 6389514-B1.
47. V. Stankovic and N. Milenkovic, “DRAM Controller with a Close-Page Predictor”, In Proceedings of EUROCON, 2005.

48. Y. Xu, A. Agarwal, and B. Davis, “Prediction in dynamic sdram controller policies”, In Proceedings of SAMOS, 2009.
49. T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder, “Discovering and Exploiting Program Phases”, In IEEE Micro, volume 23, pages 84–93, 2003.
50. I. Bhati, M.-T. Chang, Z. Chishti, S.-L. Lu, and B. Jacob, “DRAM Refresh Mechanisms, Penalties, and Trade-Offs,” in IEEE Transactions on Computers, 2015.
51. S. Khan, D. Lee, Y. Kim, A. R. Alameldeen, C. Wilkerson, and O. Mutlu, “The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study,” in SIGMETRICS, 2014.
52. J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, “An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms,” in ISCA, 2013.
53. J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, “RAIDR: Retention-Aware Intelligent DRAM Refresh,” in ISCA, 2012.
54. M. Patel, J. S. Kim, and O. Mutlu, “The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions,” in ISCA, 2017.
55. JEDEC, “DDR3 SDRAM Standard - JESD79-3E", 2010.
56. J. W. Janzen, “Calculating Memory System Power for DDR”, Micron Technology Inc., TN-46-03, October 2003.
57. Micron Technology Inc., “Calculating Memory System Power for DDR3,” Technical Note, TN-41-01, 2007.
58. Micron System Power Calculator. http://goo.gl/4dzK6.
59. B. K. Mathew, S. A. McKee, J. B. Carter, and A. Davis, “Design of a parallel vector access unit for sdram memory systems,” in Proceedings of the Sixth International Symposium on High-Performance Computer Architecture, pp. 39 –48, Jan 2000.
60. S. A. Moyar, “Access ordering and effective memory bandwidth,” Technical Report TR CS-93-18, April 1993.
61. S. A. McKee and W. A. Wulf, “Access ordering and memory-conscious cache utilization,” in Proceedings of the First International Symposium on High-Performance Computer Architecture, pp. 253 –262, Jan 1995.
62. S. I. Hong, S. A. McKee, M. H. Salinas, R. H. Klenke, J. H. Aylor, and W. A.Wulf, “Access order and effective bandwidth for streams on a direct rambus memory,” in

Proceedings of the Fifth International Symposium on High-Performance Computer Architecture, pp. 80 –89, Jan 1999.

63. Z. Zhu and Z. Zhang, "A performance comparison of dram memory system optimizations for smt processors," in Proceedings of the 11th International Symposium on High-Performance Computer Architecture, pp. 213 –224, 2005.
64. A. El-Moursy, R. Garg, and D. H. Albonesi, "Compatible phase co-scheduling on a cmp of multi-threaded processors," in International Parallel and Distributed Processing Symposium, 2006.
65. H. Zheng, J. Lin, Z. Zhang, and Z. Zhu, "Memory access scheduling schemes for systems with multi-core processors," in 37th International Conference on Parallel Processing, 2008.
66. O. Mutlu and T. Moscibroda, "Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared dram systems," in 35th Annual International Symposium on Computer Architecture, 2008.
67. Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "Atlas: A scalable and high-performance scheduling algorithm for multiple memory controllers," The 16th International Symposium on High-Performance Computer Architecture, 2010.
68. Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread cluster memory scheduling: Exploiting differences in memory access behaviour," The 43rd Annual International Symposium on Computer Architecture, 2010.
69. J. Stuecheli and D. Kaseridis, "Elastic refresh: Techniques to mitigate refresh penalties in high density memory", In MICRO, pages 375–384, Dec 2010.
70. J. Mukundan, H. Hunter, K.-h. Kim, J. Stuecheli, and J. Martinez, "Understanding and mitigating refresh overheads in high-density DDR4 DRAM systems", In International Symposium on Computer Architecture, pages 48–59, Jun 2013.
71. T. Hamamoto, S. Sugiura, and S. Sawada, "On the retention time distribution of dynamic random access memory (DRAM)", IEEE Transactions on Electron Devices, 45(6):1300–1309, 1998.
72. K. Kim and J. Lee, "A new investigation of data retention time in truly nanoscaled DRAMs", IEEE Electron Device Letters, 30(8):846–848, 2009.
73. T. Ohsawa, K. Kai, and K. Murakami, "Optimizing the DRAM refresh count for merged DRAM/logic LSIs", In International Symposium on Low Power Electronics and Design, pages 82–87, Aug 1998.

74. J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, “RAIDR: Retention-aware intelligent DRAM refresh”, In International Symposium on Computer Architecture, pages 1–12, June 2012.
75. R. Venkatesan, S. Herr, and E. Rotenberg, “Retention-aware placement in DRAM (RAPID): software methods for quasi-non-volatile DRAM,” In High-Performance Computer Architecture, pages 155–165, Feb 2006.
76. S. Baek, S. Cho, and R. Melhem, “Refresh now and then”, IEEE Transactions on Computers, 2013.
77. A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, “EnerJ: approximate data types for safe and general low-power computation,” In Programming Language Design and Implementation, pages 164–174, Jun 2011.
78. H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, “Architecture support for disciplined approximate programming”, In Architectural Support for Programming Languages and Operating Systems, pages 301–312, Mar 2012.
79. S. Liu, K. Pattabiraman, T. Moscibroda, and B. Zorn, “Flikker: Saving DRAM refresh-power through critical data partitioning”, In Architectural Support for Programming Languages and Operating Systems, pages 213–224, Mar 2011.
80. M. Ghosh and H. Lee, “Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3D die-stacked drams”, In MICRO, pages 134–145, Dec 2007.
81. P. Emma, W. Reohr, and M. Meterelliyoz, “Rethinking refresh: Increasing availability and reducing power in DRAM for cache applications”, IEEE Micro, 28(6):47–56, 2008.
82. A. Agrawal, P. Jain, A. Ansari, and J. Torrellas, “Refrint: Intelligent refresh to minimize power in on-chip multiprocessor cache hierarchies,” In High-Performance Computer Architecture, pages 400–411, Feb 2013.
83. C. Isen and L. John, “ESKIMO: Energy savings using semantic knowledge of inconsequential memory occupancy for DRAM subsystem”, In MICRO, pages 337–346, Dec 2009.
84. Micron Technology, Inc., “Calculating memory system power for DDR2”, Technical Note, 2007. http://www.micron.com.
85. T. Moscibroda and O. Mutlu, “Memory performance attacks: Denial of memory service in multi-core systems,” in Proc. 16th USENIX Security Symp. USENIX Security Symp., pp. 18:1– 18:18, 2007.

86. N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. Pugsley, A. Udipi, A. Shafiee, K. Sudan, M. Awasthi, and Z. Chishti, "USIMM: the Utah SImulated Memory Module," Technical report, University of Utah, 2012. UUCS-12-002.
87. C. Bienia, S. Kumar, J. P. Singh, and K. Li., "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in Proceedings of PACT, 2008.
88. K. Fang., N. Iliev, E. Noohi, S. Zhang, Z. Zhu, "Thread-Fair Memory Request Reordering," 3rd JILP Workshop on Computer Architecture Competitions(JWAC-3): Memory Scheduling Championship (MSC), July 2012.
89. Y. S. Moon, Y. Kwon, H.-S. Kim, D.-G. Kim, H. H. Lee and K. Park, "The Compact Memory Scheduling Maximizing Row Buffer Locality," In 3rd JILP Workshop on Computer Architecture Competitions: Memory Scheduling Championship, MSC, 2012.
90. R. Ausavarungnirun, K. K.-W. Chang, L. Subramanian, G. Loh, and O. Mutlu, "Staged memory scheduling: Achieving high performance and scalability in hetergenous systems," in Proc. ISCA, 2012.
91. Micron, "1Gb DDR2 SDRAM Component: MT47H128M8HQ-25", May 2007. http://download.micron.com/pdf/datasheets/dram/ddr2/1GbDDR2.pdf.
92. R. Gonzalez and M. Horowitz, "Energy Dissipation in General Purpose Processors," in proceedings of the IEEE Symposium on Low Power Electronics, Oct. 1995, pp. 12-3.
93. M. A. Bender, S. Chakrabarti, and S. Muthukrishnan, "Flow and stretch metrics for scheduling continuous job streams," in Proceedings of the ACM Symposium on Discrete Algorithms (SODA), 1998.
94. M.J. Khurshid, M. Chainani, A. Perugupalli and R. Srikumar, "Stride-and Global History-based DRAM Page Management," in 3rd JILP Workshop on Computer Architecture Competitions: Memory Scheduling Championship, MSC.
95. S. Rixner, "Memory controller optimizations for web servers", in Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture, IEEE Computer Society, pp.355–366, December 2004.
96. C. Natarajan, B. Christenson and F. Briggs, "A study of performance impact of memory controller features in multiprocessor server environment", WMPI-3, 2004.
97. S. Kim, A. Veidenbaum, "Stride-directed Prefetching for Secondary Caches", in proceedings of International Conference on Parallel Processor, pp. 314-321. IEEE, 1997.

98. K. Nesbit and J.E. Smith, “Data Cache Prefetching Using a Global History Buffer”, in proceedings of the 10th Annual International Symposium on High Performance Computer Architecture, pp. 144-154, July 2001.
99. Wind River Simics Full System Simulator http://www.windriver.com/products/simics/.
100. A. Modgil, V.K. Sehgal, and Nitin, “Energy-Efficient fairness-aware memory access scheduling”, in 2nd International Conference on Computers and Management (ICCM), RTU Kota, India, 28-29 December, 2016.
101. A. Modgil and V.K. Sehgal, “Energy-Efficient Performance-Aware Fair Memory Access Scheduling on Multicore Platform (EEPAF)", Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 9(3-6), pp. 61-66, 2017.
102. A. Modgil and V.K. Sehgal, “Improving the performance of Chip Multiprocessor by Delayed Write Drain and Prefetcher based Memory Scheduler”, in 2nd IEEE International Conference on Electronics, Communication and Aerospace Technology, Coimbatore, Tamilnadu, India, 29-31 March, 2018.
103. J. Jeddeloh and B. Keeth, “Hybrid Memory Cube – new DRAM architecture increases density and performance,” in Symposium on VLSI Technology, 2012.

LIST OF PUBLICATIONS

Conferences:

1. A. Modgil, Nitin, and V.K. Sehgal, “Understanding and Analyzing the Impact of Memory Controller’s Scheduling Policies on DRAM’s Energy and Performance”, in Procedia Computer Science, vol. 70, pp. 399-406, 2015.
2. A. Modgil, V.K. Sehgal, and Nitin, “Energy-Efficient fairness-aware memory access scheduling”, in 2nd International Conference on Computers and Management (ICCM), RTU Kota, India, 28-29 December, 2016.
3. A. Modgil and V.K. Sehgal, "Energy-Efficient Performance-Aware Fair Memory Access Scheduling on Multicore Platform (EEPAF)", in International Conference on Recent Innovation in Computer Science and Information Technology (RICSIT), Himachal Pradesh University, Shimla, India, 19 May, 2017.
4. A. Modgil and V.K. Sehgal, “Improving the performance of Chip Multiprocessor by Delayed Write Drain and Prefetcher based Memory Scheduler”, in 2nd IEEE International Conference on Electronics, Communication and Aerospace Technology, Coimbatore, Tamilnadu, India, pp. 1864-1869, March 2018.

Journals:

1. A. Modgil and V.K. Sehgal, “Energy-Efficient Performance-Aware Fair Memory Access Scheduling on Multicore Platform (EEPAF)", Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 9(3-6), pp. 61-66, 2017. (Scopus Index)
2. A. Modgil, V. Sehgal, and N. Chanderwal, "Energy-Efficient Fairness-Aware Memory Access Scheduling", International Journal of Services Technology and Management (IJSTM). (Scopus Index).

Communicated Papers:

3. A. Modgil and V. Sehgal, “Managing refresh penalty of DRAM based main memory system", in IEEE letter on electron devices.