Energy-Efficient Performance-Aware Fair Memory Access Scheduling on Multicore Platform (EEPAF)

Aastha Modgil and Vivek Kumar Sehgal
Department of CSE and IT, Jaypee University of Information Technology Waknaghat, Solan 173234, H.P.,India.
aastha.modgil90@gmail.com

Abstract—**In current scenario, energy consumption, performance and capacity of the main memory system are key factors that affect the design of a computing system. 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 energy consumption and performance can be improved by reducing 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. This paper proposes a memory access scheduling scheme, EEPAF, for reducing the energy consumption and improving the performance of main memory. EEPAF, prioritizes reads over writes, exploits row buffer hits, increases bank level parallelism, implement delayed write drain policy 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.**

Index Terms—**Energy Efficiency; Memory Access Scheduler SDRAM; Thread Fairness.**

I. Introduction

Nowadays, computing devices demand has extended battery life, while other systems like embedded system also require reduced power consumption as most of the applications in embedded systems are memory intensive. Memory access constitutes a significant part of overall application's energy consumption [1]-[2]. From environmental and economic aspect also, desktop systems should be energy efficient.

Main memory constitutes a major part in overall system's power consumption. In [3], authors have reported that in mid-range IBM eServer Machine, main memory contributed 40% of the total system's power consumption. The major contribution of main memory in total system's power consumption motivates researchers to find ways to reduce its energy consumption. In a modern main memory system JEDEC-style Dual Data Rate Synchronous Dynamic Random Access Memory (SDRAM) is used. DRAM's power consumption can be divided into two parts, *i.e.* ,active power consumption and standby power consumption.By minimizing active power consumption or standby power consumption, DRAM's power consumption can be reduced.DRAM's active power consumption is due to memory accesses and it can be reduced by minimizing the number of operations required to serve the memory accesses or by reducing read-write switching. On the other hand, the power consumption made by DRAM cell when it is in idle state constitutes its standby power consumption. A DRAM cell is periodically refreshed even when it is idle to maintain the data stored in it. DRAM's standby power consumption can be reduced by employing strategies like frequency scaling, power down, self-refresh mode., etc. Hence, systematized reordering of memory accesses helps to achieve the goal of minimizing active power consumption.

In addition to energy consumption, issue of fairness among threads should also be addressed while scheduling memory accesses. In a multicore environment simultaneously running threads compete with each other for off-chip memory bandwidth, as main memory is the key resource shared among all running threads. The contention among threads may lead to thread interference. Several prior studies [4]-[7]have revealed that thread interference can result in degraded system performance and fairness. This problem accelerates more with the increase in the number of cores [5]. In embedded systems most of the applications are memory intensive. Memory intensive applications generate more frequent memory accesses compared to workload intensive applications and hence tend to block the reorder buffer head. It may lead to starvation like situation for requests generated from workload intensive applications. Unfortunately, conventional scheduler does not consider the issue of thread fairness while scheduling memory commands rather they focuses on improving data throughput of the memory sub-system, *e.g.*, [8]. Hence, they do not work well in multicore environment. To solve the problem of unfairness among threads and a wise main memory access scheduler is required that fairly provides opportunity for all applications to access main memory resource.

In this paper, we propose a memory access scheduler that reorders the memory accesses and schedules them to achieve increased performance and improved energy consumption by employing policies like delayed write drain,selecting reads over writes and prioritizing row hits. In addition to improved energy consumption and performance proposed scheduler also ensures fairness among simultaneously running threads and increases bank level parallelism.In this work, we evaluate proposed scheduler against four previously proposed memory schedulers across a wide variety of workloads in terms of power consumption, energy-delay product, total execution time and fairness among threads.

The rest of the paper is organized under following sections, Section II, gives brief introduction about DRAM architecture, memory access scheduling and DRAM power model. Section III, elaborates pertinent details about the proposed scheduling mechanism. In Section IV, system configuration and performance metrics used for simulation and evaluation are described. Section V, highlight the evaluated results. In

Section VI, we conclude the paper and provide future scope.

II. Background

In this section we first briefly discusses DRAM main memory system. Then we provide an overview on previous memory access scheduling policies and DRAM power model.

A. *DRAM Architecture*

In DRAM main memory system, channels, ranks and banks are organized in hierarch manner. Channels are independent of each other and can be accessed in parallel. A channel is a collection of ranks (typically 1-4). All ranks within a channel shares common (channel's) data bus, command bus and address bus. A rank further consists of banks that can operate concurrently sharing a common data bus, command bus and address bus. Each bank is a collection of rows and columns arranged as a two-dimensional array. To service a memory access (read/write), firstly activates command is issued to bring the row containing data to sense amplifier. If consecutive memory access is also made to same row, then this scenario results in row hit. In this scenario, there is no need to issue the activate command first as requested row is already present in sense amplifier. Whereas if consecutive memory access is intended to some other row, then this situation leads to row miss or row conflict. Time to serve a memory request and energy consumed while serving the request depends on whether a memory access is leading to row hit or miss/conflict. Row hit memory accesses are served ~2-3x faster than a row miss/conflict accesses [9] and consumes lesser energy too. [10]-[13] have provided detailed information about DRAM operations.

B. *State-of-the-art Memory Access Scheduling*

The effect of memory access scheduling on the system's performance and energy consumption has been testified in many previously conducted studies [6]-[8], [14]-[18]. Numerous work has been done to improve the energy consumption and performance of the main memory system. In [8], proposed FRFCFS (First Ready First Come First Serve) memory scheduler that prioritized new row hits over pending row miss requests to achieve improved performance and energy consumption compared to the FCFS scheduling policy. In FRFCFS fairness constraint does not take care of while issuing commands. FCFS scheduler schedules commands in the order of their arrival time. A variant of FCFS policy schedules requests as per their arrival time in the ready queue. The ready queue is then sequentially scanned to find out request that can be served in the current cycle. Stall Time Fair Memory Scheduler proposed in [6], keeps track of all executing threads in terms of their maximum slowdown time. The threads experiencing maximum slowdown are prioritized over others. Scheduler PRWL proposed in [19], pre-issues some non-conflicting read commands during write mode and write command during read mode. In [20], researches have presented a memory scheduler (ATLAS) that prioritizes threads having least service time in previous epochs. In [21], the authors have proposed Row Locality Based Drain policy that allows read-write swapping only when all read hits or write hits are exhausted while executing in read mode or write mode respectively. PBFS (Priority Based Fair Scheduler) [22], addresses the issue of starvation by periodically accessing the behavior of thread's memory accesses. In [23], Fang et al. presented a scheduler named Thread-Fair Memory Request Reordering that gives preference to oldest request generated from each executing thread. In [24], authors investigated the role of memory access scheduling in resolving conflicts generated by multi-threaded workloads. In basic close page policy precharge command is issued immediately after serving the read/write command. In another variant of close-page policy, the scheduler issues precharge command to last serviced memory address on finding idle cycle.

C. *DRAM Power Model*

In this section, we describe the power model used to calculate memory system power consumption. DRAM power consumption can be divided into two components consumed by memory elements (core power consumption) [25], and power consumed while driving data into or out of the data bus (I/Opower consumption). Power consumed by memory elements, *i.e.*, core power consumption comprised of three main elements; i) Average power consumption when the memory is in idle state (base power consumption), is the sum of power consumed in standby mode and during refresh operation ii) Power consumption when DRAM is active (active power consumption) and iii) Power consumption while servicing read/write requests. The equations used for power modeling are based on Micron Memory System Power Technical Note [26] and Micron power calculator [27]. For better assimilation, $P_{(XX)}$ is used to denote power consumed by XX sub-component. Total power consumed by DRAM chip is calculated as

$$\begin{aligned} &P_{(DRAM)} = P_{(read)} + P_{(write)} + P_{(activate)} + P_{(background)} \\ &P_{(ter\min ate)} + P_{(refresh)} \end{aligned} \quad (1)$$

To calculate total memory system power consumption, DRAM chip power is multiplied by number of DRAM chip.

III. Proposed Mechanism

In this section we provide details of our proposed memory access scheduling policy. The key features of the EEPAF policy area) preferred read requests overwrites ii) prioritized row buffer hits iii) delayed write drain iv) thread fairness and v) bank level parallelism.

The proposed scheduling policy gives preference to memory read requests over write requests as memory reads significantly affects the system's performance. When the processor generates a memory read request, it stalls its execution while waiting for reply from memory (content requested). The processor stops its execution till generated read request is serviced by the memory sub-system. This halt in execution results in increased execution time for an application. Whereas memory write request does not stall the processor. Hence, in our proposed scheduler we prioritize read requests overwrites, resulting in decreased halting time of processor that further lead to decreased execution time of application. Decreased execution time may help in reducing the energy consumption of the system.

Another feature that we have employed in our proposed scheduling algorithm is prioritized row buffer hit requests over other requests. A row buffer hit is a condition in which the address which is being addressed is already present in the sense amplifier, so in order to perform read or write on specified memory address only column-read or column-write is required, respectively. Whereas in the row buffer miss situation column read or column write command should be preceded by precharge command and activate command. The least number of commands are required to be issued in row hit situation and hence the least amount of energy is consumed. The dynamic energy consumed to perform column read command (to read data from cell) on a DRAM memory cell is given by:

$$E_{RD} = (I_{\mathrm{DD4R}} - I_{DD3N}) * V * T_{data} \tag{2}$$

where: I_{DD4R} denotes current withdrawn to perform column read and 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).

$$T_{data} = M * T_{burst} \tag{3}$$

T_{burst} represents a data transfer latency and is given by (4).

$$T_{burst} = BL * {t_{clk}}/{2} \tag{4}$$

Along with E_{RD} additional dynamic energy (E_{DQ}) is also expanded to read data out from DRAM cell, given by (5).

$$E_{DQ} = P_{DQ(R)} * (N_{DQ} + N_{DQS}) * T_{data} \tag{5}$$

where, $P_{DQ(R)}$ represents the power consumed per pin while extracting output [28]. $N_{DQ(R)}$ 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 (6).

$$E_{WR} = (I_{DD4W} - I_{DD3N}) * V * T_{data} \tag{6}$$

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

While writing data, write termination energy is also spent to write (7).

$$E_{term} = P_{DQ(W)} (N_{DQ} + N_{DQS} + N_{DM}) * T_{data} \tag{7}$$

In Equation (7), N_{DM} represents the number of data mask pins and $N_{DQ(M)}$ denotes power per pin during write termination.

Equation (8) and (9) gives dynamic energy consumption during read miss and write miss.

$$E_{DRAM(readmiss)} = E_{DD0} + E_{RD} + E_{DQ} \tag{8}$$

$$E_{DRAM(writemiss)} = E_{DD0} + E_{WR} + E_{term} \tag{9}$$

where,

$$\begin{aligned} E_{DD0} = (I_{DD0} - {1}/{t_{rc}} \, (I_{DD3N} + t_{RAS} + I_{DD2N} * \\ (t_{rc} - t_{RAS}) * V * t_{rc} \end{aligned} \tag{10}$$

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

Dynamic energy spent in a 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 (11) and (12).

$$E_{DRAM(readhit)} = E_{RD} + E_{DQ} \tag{11}$$

$$E_{DRAM(writehit)} = E_{WR} + E_{term} \tag{12}$$

By analyzing the Equation (8), (9), (11) and (12), it is clearly revealed that row buffer hits require a lesser number of operations to access the desired page.

EEPAF also employ delayed write drain policy to further prioritize read request and to exploit row buffer hit. In conventional scheduling policies once all read requests are served, *i.e.* ,read queue gets empty, the scheduler enters into write drain mode. In proposing 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. So, by extending read drain mode EEPAF prioritizes read requests and further more read hits can be achieved. Delayed write drain policy enhanced the scheduler's ability to reduce energy consumption and performance.

In addition to rationalizing energy consumption, proposed scheduler also provides fairness among threads and bank level parallelism. In order to ensure fairness, EEPAF is based on following idea. 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 $T_{interf(own)}$.

$$T_{\mathrm{int}erf} = T_{\mathrm{int}erf(others)} + T_{\mathrm{int}erf(own)} \tag{13}$$

$T_{interf(others)}$ 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, $T_{interf(bank)}$.

$$T_{\mathrm{int}erf}(others) = T_{\mathrm{int}erf(bank)} + T_{\mathrm{int}erf(bus)} \tag{14}$$

Every read or write request is sent 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:

$$T_{bus} = {BL}/{2} \tag{15}$$

EEPAF reduces a thread's interference caused due to other threads. Memory intensive threads tend to access main memory sub system more frequently and hence block the reorder buffer head. Our scheduler prioritizes service requests generated from the reorder buffer so other requests (generated from load intensive threads) that were starved due to blocked reorder buffer gets equal opportunity to be served.

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.

Flow chart of our implemented scheduling policy is given in Figure 1.

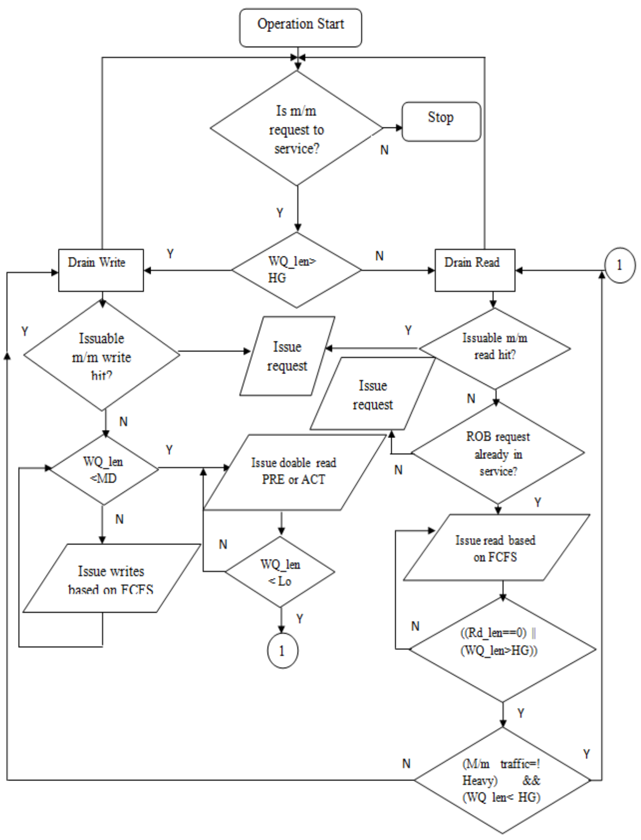
Operation Start
Is m/m request to service?
Stop
Drain Write
WQ_len> HG
Drain Read
1
Issuable m/m write hit?
Issue request
Issuable m/m read hit?
Issue request
ROB request already in service?
WQ_len <MD
Issue doable read PRE or ACT
Issue writes based on FCFS
WQ_len < Lo
1
Issue read based on FCFS
((Rd_len==0) || (WQ_len>HG))
(M/m traffic=! Heavy) && (WQ_len< HG)

Figure1: Flow Chart of EEPAF

IV. METHODOLOGY

In this section we 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.

A. *System Configuration and Workloads*

We build our proposed scheduler on simulator named USIMM [29], that issues device level memory command based on current memory status. To evaluate proposed scheduler experiments are run using two memory configurations, Table 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) [30] are used for simulation. Using before mentioned trace files ten different workload combinations are made and simulated for both memory combinations in the varied core environment.

B. *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 [31]. To measure unfairness among threads, maximum slowdown time performance metric is used [32]. 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 measure the thread's execution time.

Table 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

V. EVALUATION

We evaluated the 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 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.

A. *Memory System Power Consumption*

In terms of memory system power consumption, proposed scheduling policy outperforms all simulated policies for both memory configurations under multi-core environment. Here the 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 a simple mechanism and does not exhaust power to limit other factors. On analyzing the simulation trend we find that there is an 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. Figure 2, depicts the performance of EEPAF in comparison to all simulated scheduling policies for both memory configurations and varied core count.

B. *Energy Delay Product*

The results shown in Figure 3, for Energy Delay Product reveals that EEPAF performed best among all simulated memory access scheduling policies for both memory configurations. 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.

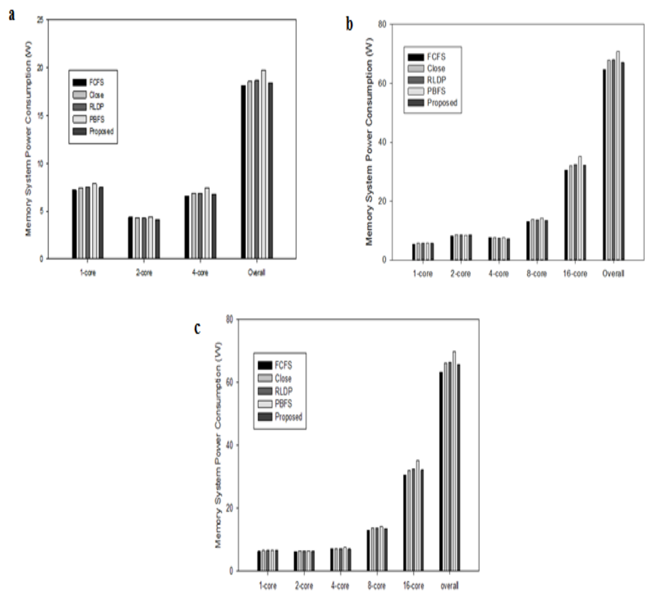

Figure 2: (a) Comparison based on Memory System Power Consumption using memory configuration-1, (b) Comparison based on Memory System Power Consumption using memory configuration-2, (c) Overall Memory System Power Consumption

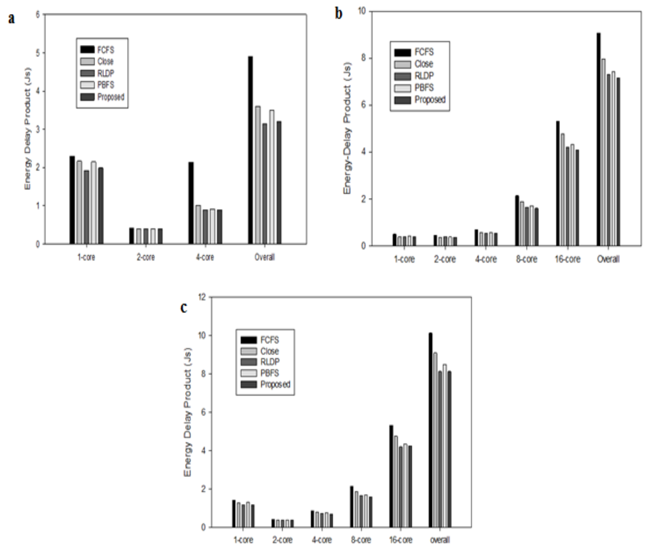

Figure 3: (a) Comparison based on Energy Delay Product using memory configuration-1, (b) Comparison based on Energy Delay Product using memory configuration-2, (c) Overall Memory System Power Consumption

C. *Total Execution Time*

The simulation trend seen in Figure 4, reveals that overall performance of EEPAF is better than PBFS, FCFS and close page policy. EEPAF 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, EEPAF shows 0.99% increase in execution time. But if we consider energy consumption, then EEPAF reduced energy consumption in comparison to RLDP.

D. *Maximum Slowdown Time*

For maximum slowdown time performance metric, EEPAF showed best performance among all simulated memory access scheduling policies for both memory configurations (configuration-1 and configuration-2) under varied core environment. By limiting maximum slowdown time, EEPAF managed to reduce un-fairness among simultaneously running threads in multicore platform.

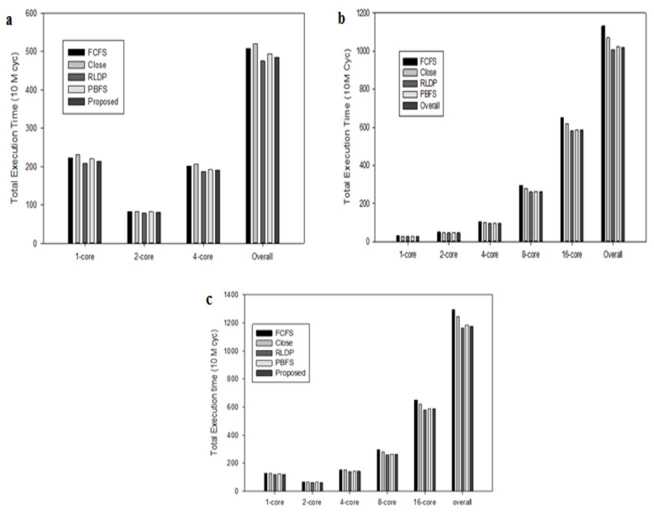

Figure 4: (a) Comparison based on Total Execution Time using memory configuration-1, (b) Comparison based on Total Execution Time using memory configuration-2, (c) Overall Total Execution Time

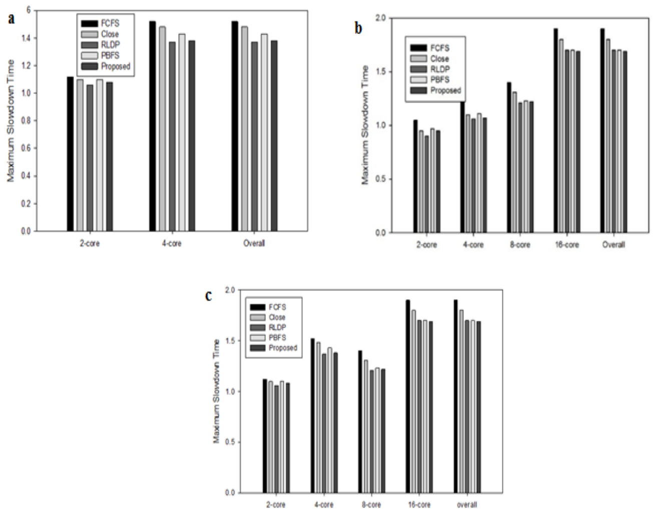

Figure 5: (a)Comparison based on Maximum Slowdown Time using memory configuration-1, (b)Comparison based on Maximum Slowdown Time using memory configuration-2, (c) Overall Maximum Slowdown Time

VI. CONCLUSION

We introduce energy-efficient performance aware fair memory scheduler, EEPAF. The detailed analysis conducted across a wide variety of workloads in a varied core environment reveals that EEPAF significantly reduces energy consumption and improves performance of the 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 reducing the number of operations required to service a memory request. Reduction in number of operations is achieved by maximizing row hits. Whereas, thread's execution time is reduced by i) reducing the 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 the system while reducing energy consumption because it considers both quantities, *i.e*., power and execution time while scheduling commands. We conclude that EEPAF can be an effective and efficient memory access scheduling strategy for multicore systems. 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.

REFERENCES

[1] K. Barr and K. Asanovic, "Energy aware lossless data compression," in 1st Int. Conf. Mobile Systems, Applications, and Services (MobiSys'03), San Francisco, CA, May 2003.

[2] 8 Kim, H., Vijaykrishnan, N., Kandemir, M., Brockmeyer, E., Catthoor, F., and Irwin, M.J.: 'Estimating influence of data layout optimizations on SDRAM energy consumption'. Proc. ISLPED, Aug. 2003, pp. 40–43.

[3] 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.

[4] K. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith, "Fair queuing memory systems," in Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchit., 2006, pp. 208–222..

[5] 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., 2007, pp. 18:1– 18:18.

[6] 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.

[7] 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.

[8] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," in Proc. 27th Annu. Int. Symp. Comput. Archit., 2000, pp. 128–138.

[9] JEDEC, Standard No. 79-3. DDR3 SDRAM STANDARD, 2010.

[10] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A case for exploiting subarray-level parallelism (SALP) in DRAM," in Proc. 39th Annu. Int. Symp. Comput. Archit., 2012, pp. 368–379.

[11] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, "Tiered-latency DRAM: A low latency and low cost DRAM architecture," in Proc. IEEE 19th Int. Symp. High Perform. Comput. Archit., 2013, pp. 615–626..

[12] D. Lee, K. Yoongu, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu, "Adaptive-latency DRAM: Optimizing DRAM timing for the common-case," in Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit., 2015, pp. 489–50.

[13] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. Gibbons, M. Kozuch, and T. Mowry, "RowClone: Fast and efficient In-DRAM copy and initialization of bulk data," in Proc. 46th Annu. IEEE/ACM Int. Symp. Microarchit., 2013, pp. 185–197.

[14] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread cluster memory scheduling: Exploiting differences in memory access behavior," in Proc. MICRO, 2010.H. Simpson, Dumb Robots, 3rd ed., Springfield: UOS Press, 2004, pp.6-9.

[15] M. Bojnordi and E. Ipek, "PARDIS: A programmable memory controller for the DDRx interfacing standard," in Proceedings of ISCA, 2012.

[16] Aastha Modgil, Nitin and Vivek Kumar Sehgal,"Understanding and Analyzing the Impact of Memory Controller's Scheduling Policies on DRAM's Energy and Performance," in proceedings of the 4th ICECCS, 2015, vol. 70, pp.399-406

[17] J. Shao and B. T. Davis, "A burst scheduling access reordering mechanism", In HPCA-13, 2007

[18] I. Hur and C. Lin., "Adaptive history-based memory schedulers", in MICRO-37, 2004.

[19] Long Chen, Yanan Cao, Sarah Kabala and Parijat Shukla, "Pre-Read and Write-Leak Memory Scheduling Algorithm," in 3rd JILP Workshop on Computer Architecture Competitions: Memory Scheduling Championship, MSC, 2012.

[20] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers," in Proc. HPCA, 2010.

[21] Y.S. Moon, Y. Kwon, H.S. Kim, D. 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.

[22] Li, C., Wang, D., Wang, H., & Xue, Y. Priority Based Fair Scheduling: A Memory Scheduler Design for ChipMultiprocessor Systems. Tsinghua National Laboratory for Information Science and Technology.

[23] Fang, K., Iliev, N., Noohi, E., Zhang, S., Zhu, Z.; 2012.; "Thread-Fair Memory Request Reordering," 3rd JILP Workshop on Computer Architecture Competitions(JWAC-3): Memory Scheduling Championship (MSC), July 2012.

[24] Z. Zhu and Z. Zhang, "A performance comparison of dram memory system optimizations for smt processors," in Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA), pages 213 – 224, Feb. 2005.

[25] J. W. Janzen. TN-46-03 - Calculating Memory System Power for DDR. Micron Technology, Inc., October 2003.

[26] Micron Technology Inc. Calculating Memory System Power for DDR3 - Technical Note TN-41-01, 2007

[27] Micron System Power Calculator. http://goo.gl/4dzK6.

[28] Micron Technology, Inc., 2004. Calculating memory system power for DDR2. Technical Note, http://www.micron.com.

[29] 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.

[30] C. Bienia, S. Kumar, J. P. Singh, and K. Li., "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in Proceedings of PACT, 2008.

[31] 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.

[32] 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.