Power Management for Main Memory with Access Latency Control

Matthias Eiblmaier
University of Tennessee
Department of Electrical Engineering and Computer Science
Knoxville, TN 37996
+1 865 235 3415
meiblmai@utk.edu

Rukun Mao
University of Tennessee
Department of Electrical Engineering and Computer Science
Knoxville, TN 37996
+1 865 898 6652
rmao@utk.edu

Xiaorui Wang
University of Tennessee
Department of Electrical Engineering and Computer Science
Knoxville, TN 37996
+1 865 974 0627
xwang@eecs.utk.edu

ABSTRACT

Main memory, as an essential component of modern computer systems, has increased its power consumption significantly over the last 10 to 20 years. Energy consumed by memory can account for as much as 40% of an entire computer system. To address this challenge, major memory manufactures have developed memory chips with multiple power states. Putting memory chips into low power modes when they are not (or less frequently) being used is an effective way to save power. As a tradeoff, the time the chip needs to respond to an access may increase. It is therefore crucial to choose optimal timing for power mode switching depending on the current workload. In this paper, we present a power management algorithm to save power while guaranteeing the desired memory access latency. Our algorithm is designed based on feedback control theory to achieve theoretical analysis and guarantees. By performing extensive simulations based on a practical memory model, we demonstrate that our algorithm can improve memory power efficiency up to 19.3%. The stability of the designed controller has also been examined in our simulations.

Categories and Subject Descriptors
C.4 [**Performance of System**]; D.4 [**Operating Systems**]

General Terms
Performance, Algorithms, Management, Experimentation, Measurement.

Keywords
Power management, feedback control, main memory.

1. INTRODUCTION

Energy efficiency is becoming increasingly important in various computer systems, such as embedded systems. For example, mobile devices need to reduce power consumption to extend battery lifetime. While various power management algorithms have been developed for microprocessors, main memory has not received enough attention with regards to energy optimization. The power consumption of memory devices has significantly increased in recent years. For example, the power consumed by main memory can reach up to 50% of the power consumption of the whole system for some pocket computers. Thus, memory system energy efficiency is important and main memory is becoming a crucial target for energy optimization.

On the other hand, memory system performance can also become the bottleneck in the performance improvement of an entire computer system. Performance of modern computer systems has been greatly improved because of the fast development in silicon process technology. For example, the performance of processors has doubled approximately every two years. However, improvement in processor performance does not lead to a proportional performance increase of computer systems for all types of applications. The reason lies in the fact that computer system performance is fundamentally constrained by the interaction between the processor and memory. As a result, power conservation at the cost of significant memory system performance loss is not appropriate. Memory performance must be guaranteed while achieving power savings, which is the key research goal in power management for main memory.

To address memory power efficiency, the memory industry has produced memory chips with multiple power modes rather than a single power mode. Those power modes can be used to offer various power management features [1]. One example is Direct Rambus DRAM (RDRAM) technology, which has four power states: active, standby, nap and power-down, with power consumption in decreasing order and access latency in increasing order. All transactions have to be executed in the active state. Resynchronizing the memory chip to active state from a low power state may result in additional latency. Lower power states may need longer time for resynchronization.

Various hardware and software policies for controlling DRAM power state switching have recently been introduced. There are mainly two categories: static policies and dynamic policies. Static schemes place all DRAM chips in a single power state. DRAM chips must first transit to the active state in order to respond to an access. Only when there are no outstanding requests, the device can return to certain low-power states. In dynamic policies, a certain timing threshold is introduced which allows the chip to power down during control epochs when the chip's idling time is longer than the threshold. Consequently, the chip is able to achieve the desired performance and energy efficiency. Common approaches to dynamic power control use the time between accesses to a chip as a metric for transitioning to lower power states. If a chip is not accessed for a threshold amount of time, it transitions to the next lower power state. Clearly, the threshold values are critical parameters in this approach. If they are too large, the approach may result in more time spent in a higher power mode and thus lower power efficiency; if they are too small, the approach may result in more transitions into lower power modes than desired, and subsequently degrade the performance of the system.

In this paper we propose a novel power management approach for main memory with access latency control. We assume that the user has certain deadlines for applications which are his/her primary concern. Within the range imposed by the deadline, we try to tune the thresholds for power saving modes to slow down the speed of the memory for reduced energy consumption.

Specifically, this paper makes the following contributions:

- We present a practical feedback method for controlling the memory latency by dynamically manipulating the thresholds.
- We analyze the threshold's influence on memory's power consumption through practical memory power modeling.
- We implement two controllers, one based on a common heuristic and the other based on feedback control theory for memory power feedback control for comparative studies.
- We carry out extensive simulations to test the memory power management algorithm and analyze the results.

The remainder of this paper is organized as follows: The next section provides a brief introduction about existing related work on memory power management. Section 3 presents our approach for memory power control system design, while Section 4 explores the practical memory power model and describes the implementation of the power management controller. In Section 5, experimental results are presented and analyzed. Finally, conclusions are given in Section 6.

2. RELATED WORK

There are some architectural-level studies that examined the impact of software structure on power consumption [15]. There are also other architecture studies that focused on the memory hierarchy [16]. Device-level power management includes work on scheduling for low power modes. Common methods that appeared in many power management solutions are the identification and prediction of the length of idle time in the activity patterns of a component.

Much of the related work on energy-driven optimization only considers the energy consumption of processors alone [2] [3] [4] [5]. In most embedded systems, the processor may only consume a limited amount of the total energy budget. Memory energy optimization between processor and memory are studied in [13] and [14].

Because of current increasing disparity in the operating frequency of modern processors and access latency to main memory, a lot of ongoing research effort is devoted to optimization of the DRAM memory controller. DRAM command and memory transaction ordering schemes have been studied by Ibrahim et. al. [6], Rixner et. al. [8], Cuppu et. al. [7], and Hur et. al. [9]. Address mapping schemes designed to minimize bank address conflicts have been studied by Zhang et. al. [10] [11] and Lin et. al. [12].

3. SYSTEM DESIGN

Many existing approaches have very sophisticated techniques to solve the energy versus time optimization problem. However, these approaches need high computing performance for the memory controller and therefore may cause unacceptable overhead. Some existing work tries to reach a certain power set point, which needs to be adapted at runtime, e.g., if workload increases. In this paper, we aim to achieve the minimum acceptable performance (i.e., access latency). The approach of this paper can therefore be summarized as: Saving energy if access latency allows.

The power management algorithm in this paper does not account for the interaction with the processor. However, its deadline can be dynamically adjusted according to optimal interaction with the CPU. The algorithm can be extended to handle multiple power-modes (with respective thresholds) and multiple tasks accessing different parts of the main memory. It can also run on top of other power saving mechanisms like cycle throttling.

3.1 Access Latency Control

The algorithm proposed in this paper is an epoch-based controller. It is invoked periodically during runtime and triggered by the number of memory cycles (i.e. every 10,000 cycles). Every time it is executed, it changes the threshold based on the difference between actual and desired memory access latencies. Its monitored variable is the access latency and its manipulated variable is the threshold.

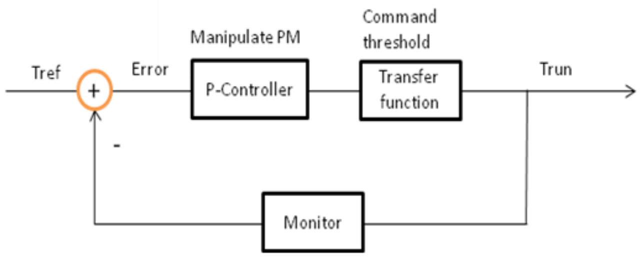


Figure 1: Threshold Feedback Control Loop

If the access latency is higher than the desired set-point, our algorithm will increase the threshold to decrease the number of power-down events happening per epoch. In order to calculate the set point per epoch, the user has to provide the controller with two crucial parameters:

- D_l: deadline of the application
- C_{est}: approximate number of memory cycles needed for completion

The first parameter is easily available since it can be given by an objective of a system or project. The second parameter, however, needs to be either empirically measured or pre-calculated by cycle prediction programs. Since we can divide every application into chunks (e.g. 1 second), it will be sufficient to first estimate the number of memory cycles per chunk. After the first completion of a chunk, one can then adapt the estimated number of memory cycles to the measured one. This method is, however, part of our future work and is not implemented in this paper.

When the information for both deadline and memory cycles is provided, one can calculate the desired access latency set point in the kth epoch with the following formula:

$$T_{set}(k) = \frac{C_{est}}{D_l} \cdot (C_{act}(k) - C_{pen} \cdot N_{pd}(k)) \qquad (1)$$

where C_{act} is the actual number of executed memory cycles. N_{pd} is the number of power down events. Both numbers are provided by the memory controller. C_{pen} is the number of cycles it costs to power up again into active mode from low power mode. These cycles do not contribute to the computation of the application and therefore need to be subtracted. C_{pen} is a constant whose value depends on which power-down mode the chip is currently in [18] [19]. Some example values are shown in Table1.

Table1: RDRAM Power State and Transition Values

Power State Transition	Power (mW)	Time (nS)
Active	$P_a = 300$	t_{acc}=60
Standby	$P_s = 180$	-
Nap	$P_n = 30$	-
Powerdwn	$P_p = 3$	-
Stby → Act	$P_{s\to a} = 240$	$T_{s\to a} = +6$
Nap → Act	$P_{n\to a} = 165$	$T_{n\to a} = +60$
Pdn → Act	$P_{p\to a} = 152$	$T_{p\to a} = +6000$

The actual access latency T_{act} is measured directly in every epoch and then compared with T_{set}. The difference between T_{act} and T_{est} is called slack and should be zero or positive.

3.2 Energy Control

The access latency controller is effective in achieving the desired timing constraints. However, it needs to be augmented with an energy controller to assure that the energy consumption is actually decreased. Note that energy consumption does not necessarily decrease monotonically for lower thresholds. For example, if the chip has to transit between power stages too often, energy consumption might be even worse than without power saving modes [20].

Whenever the timing controller allows setting the threshold lower, the energy controller will check whether power can be actually saved. This is done by an open-loop model which allows the absence of monitoring, hence no measuring devices or energy calculation algorithms are needed.

$$P(k) = P_{active}\left(C_{act}(k) - C_{pd}(k)\right)$$
$$+P_{pd}\left(C_{pd}(k)\right) + P_{change}\left(C_{pen} \cdot N_{pd}(k)\right) \qquad (2)$$

As one can see in equation (2), the power consumption is dependent on the number of power-downs which occurred in the kth epoch. If this number is too high the last term of the equation will cause the whole energy consumption to increase. Since this model is not very accurate (there are many system and application dependent effects like inter-arrival time and workload [19]), it just serves as an estimator of the tendency. The controller will store the energy consumed in the last epoch P(k-1) and compare it with the current one P(k). If the threshold was increased from the last epoch to the current epoch, and the difference P(k) - P(k-1) is positive, then the energy versus threshold (see section 4) curve is in the monotonically increasing range. Therefore, it is energetically effective to power down. The same is true when the threshold was decreased in the last period and it resulted in a decreasing energy consumption (P(k)-P(k-1) is negative).

3.3 Integration

In the scope of this paper, we have provided both an Ad-hoc controller and a P-controller to set the thresholds. In section 4 we will discuss their advantages and disadvantages. Both controllers are relatively lean (20-30 lines of code and no loops) and can be integrated in either the memory controller itself or in the kernel of the operating system.

A challenge for future work is to design a controller which keeps track of every chip and the respective applications accessing it. For simplicity we designed our controller to run one application on one chip.

4. SYSTEM MODELING AND CONTROL ALGORITHM

Main memory subsystem is comparatively complicated due to its various access status and corresponding power consumption in each access status. One practical method to approach this problem is to set up a simulator which takes as many parameters from practical RAM chips as possible into consideration, then feed the created model with a great amount of characteristic input and indentify memory system's overall behavior rather than focusing on detail interaction among different parameters and states.

4.1 System Model

Memory is treated as a black box with three interface variables about which we are concerned the most: power-down threshold, energy consumption, and run-time.

Our system is based on two basic models:

- The time consumed is monotonically decreasing with increasing threshold.
- Energy consumption is monotonically increasing with increasing threshold for a certain range of thresholds.

To verify this we set up a simulation environment called XEEMU. XEEMU is a configurable runtime and power simulator for the Intel(c) XScale(c) architecture. In contrast to other simulators, it targets one specific architecture and also conducts the cycle-accurate simulation of an SDRAM subsystem. Its main features include: cycle-accurate SDRAM simulation, accurate power simulation, high simulation speed and accurate simulation of voltage and frequency scaling [17].

For workload, we use a typical application program named JPEG TRANSFORMATION, which converts colorful image to a black and white version. This workload is used to test memory system and identify relationships between variables such as runtime-threshold and energy-threshold. Two images are converted: Butterfly.jpg and Lena.jpg. The threshold is tuned and tested in the range between 1 and 100, with an increase of 1 for every run. For each threshold, the corresponding runtime (i.e., access latency) is shown in Figure 2.

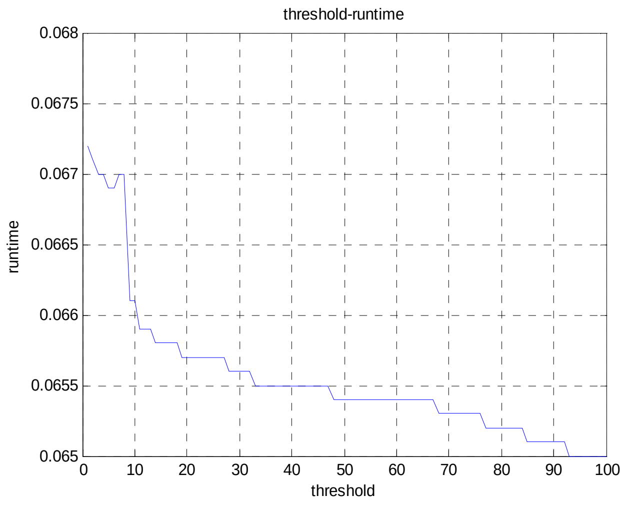


Figure 2: Runtime monotonically decreasing with Threshold increasing.

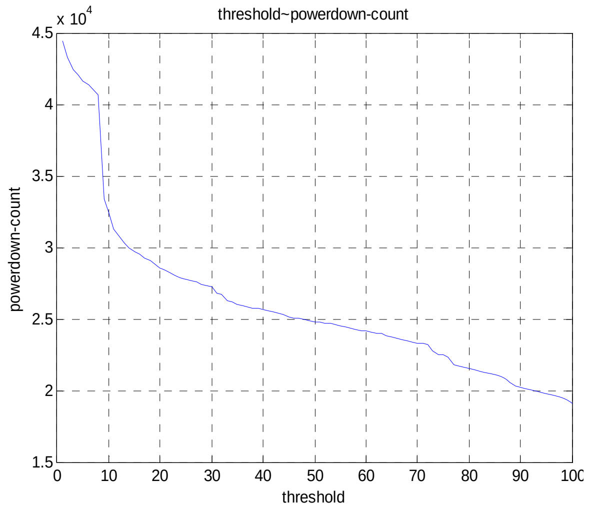


Figure 3: Relationship between Threshold and Number of Power-downs

As stated previously, despite the fact that memory saves power when it stays in a low-power state, it consumes more energy then in normal operation during the process of switching power modes. The relationship between threshold and number of power-downs (Figure 3) verifies the reasonable prediction that larger thresholds result in a lesser chance of switching into power-down mode. When the threshold is set to a small value (<20) there is consequently frequent power mode switching. The power saved when changing into a power saving mode will be consumed by the significant amount of power needed by the high number of mode changes.

The experiment is repeated, but this time we measure the energy consumption and compare it with our simple model (section 3.2). The thresholds are increased with a step size of 20 for each run in the range of 0 to 1600. We measure the number of actual cycles (C_{act}) and of power-down events (N_{pd}). Using this data with equation (2) results in a curve like the one displayed in Figure 4. Note that the jumps in the range between 0 and 200 are caused by several system effects (row misses, etc.) which are not investigated further. The energy consumption in this area is too high and/or unpredictable for our controller to operate.

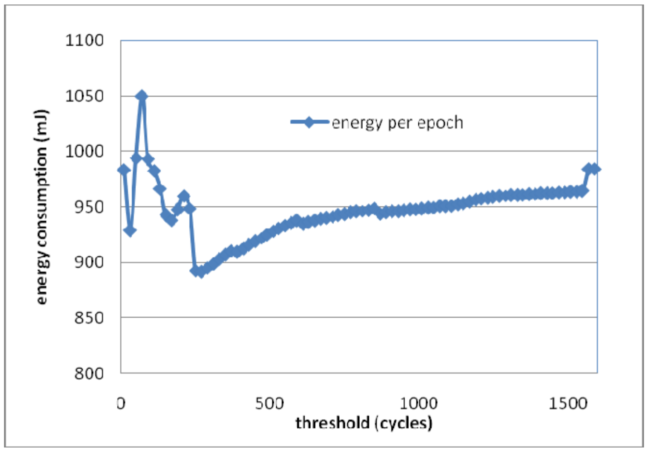


Figure 4: Theoretical Energy consumed with various Thresholds.

The simulator uses its own model to calculate the power consumed by main memory. In Figure 5 the results are displayed for the same threshold range and step-size.

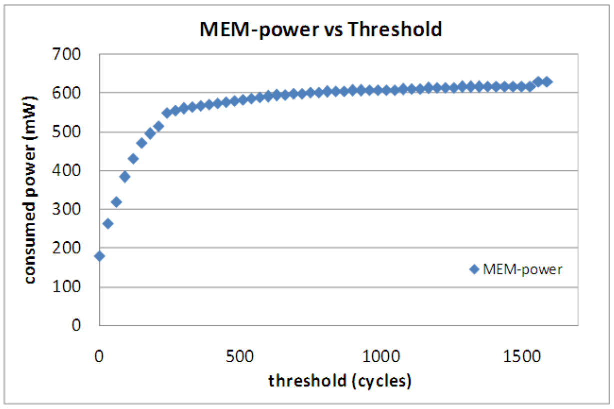


Figure 5: Power consumed with various Thresholds based on actual Memory System Model

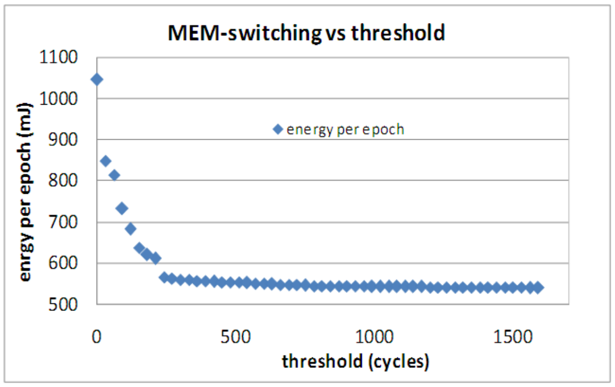


Figure 6: Power Consumption caused by Power Mode Switching.

The power per epoch alone is misleading regarding the total energy consumption. The power consumption needs to be multiplied with the respective runtime to gain the energy consumption. If we take these two sources (Figure 5 and Figure 6) into account, one can see that the energy consumption is only monotonically increasing with increasing thresholds for a certain range of thresholds (>200 cycles). Note that for large access inter-arrival times, the energy consumption might be decreasing with decreasing thresholds even for very low or zero thresholds [18].

From the above system modeling process, we verified feasibility of two basic assumptions: The relationship between runtime and threshold, as well as energy consumption and threshold through a practical memory system model. Theoretically there exists a trade-off between power saved by putting memory into power-down and extra power needed for power mode switching. But if memory access is augmented by optimal methods, such as instructions throttling and access predication, then, with a smaller threshold, it is still possible to save more energy for memory subsystem. Note that, for simplicity, we do not take effects like instruction throttling etc. into account for our energy model.

4.2 Control Algorithm

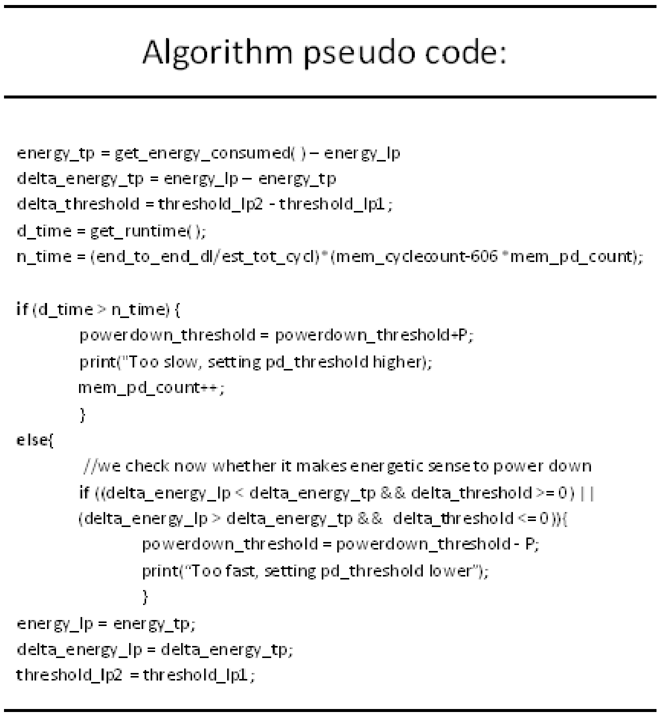
Based on these results, the control algorithm can be finally constructed.

4.2.1 Ad-Hoc Controller

As already stated in section 3, the control algorithm is periodically invoked. Every time it is executed, it compares actual runtime (i.e., access latency) with the set point. If the actual runtime is bigger than the set point, Ad-hoc controller raises the threshold to avoid time-expensive power-down events. The threshold is increased or decreased by a fixed step size which has proven to be most efficient between 10 and 100 cycles.

As one can see, the energy controller is only used when the time controller needs to lower the threshold. It compares the delta of the energy consumption to the delta of the threshold to assure that it is in the monotonically increasing range of the energy consumption:

$$\frac{\Delta Energy}{\Delta Threshold} > 0 \qquad (3)$$



Algorithm pseudo code:

```
energy_tp = get_energy_consumed( ) – energy_lp
delta_energy_tp = energy_lp – energy_tp
delta_threshold = threshold_lp2 - threshold_lp1 ;
d_time = get_runtime( );
n_time = (end_to_end_dl/est_tot_cycl)*(mem_cyclecount-606 *mem_pd_count);

if (d_time > n_time) {
        powerdown_threshold = powerdown_threshold+P;
        print("Too slow, setting pd_threshold higher);
        mem_pd_count++;
        }
else{
        //we check now whether it makes energetic sense to power down
        if ((delta_energy_lp < delta_energy_tp && delta_threshold >= 0 ) | |
        (delta_energy_lp > delta_energy_tp &&  delta_threshold <= 0 )){
                powerdown_threshold = powerdown_threshold - P;
                print("Too fast, setting pd_threshold lower");
                }
energy_lp = energy_tp;
delta_energy_lp = delta_energy_tp;
threshold_lp2 = threshold_lp1 ;
```

This slope might change drastically during runtime because the memory accesses of applications vary with time. Consequently there is no fixed border threshold and the controller adjusts dynamically to the workload.

4.2.2 P-Controller

The Ad-hoc controller can be improved with a P-Controller to allow faster and more accurate tuning to reach the desired threshold. With a controller implemented using well established control theory, system performance can be theoretically analyzed and therefore system characteristics such as accuracy and stability are guaranteed even in an uncertain application environment.

The P-controller uses a delta which is proportional to the error ($e = T_{set} - T_{act}$) instead of a fixed step size.

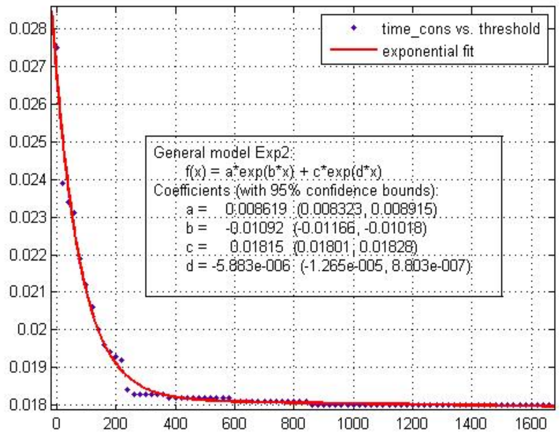
$$T(k) = T(k-1) + K \cdot e(k-1) \qquad (4)$$

To distinguish the correct value for K, which guarantees a fast and stable controller, one needs to find the transfer function of the system. Because of the high complexity and application dependence, a system identification based on empiric results is preferable to an analytical approach.

By running the benchmarks for different thresholds (similar to Section 4.1), we can try to find a function that describes the dependence from runtime to threshold best.

The result (Figure 7) is the sum of two exponential functions.

$$T(k) = a \cdot e^{bx} + c \cdot e^{dx} \qquad (5)$$



```
General model Exp2:
     f(x) = a*exp(b*x) + c*exp(d*x)
Coefficients (with 95% confidence bounds):
     a =    0.008619  (0.008323, 0.008915)
     b =    -0.01092  (-0.01166, -0.01018)
     c =     0.01815  (0.01801, 0.01828)
     d = -5.883e-006  (-1.265e-005, 8.803e-007)
```

Figure 7: Measured consumed time

The Z-Transform of the transfer function can then be calculated by:

$$T(Z) = \frac{za}{z-e^b} + \frac{zc}{z-e^d} \qquad (6)$$

Using this transfer function in combination with a P-Controller in a closed-loop model allows us to calculate the optimal system gain K. The detailed steps are skipped due to space limitations. Based on the numerical calculation performed in Matlab and empirical tuning, we found that a value of K between 100,000 and 1,000,000 can result in a stable and efficient controller (see section 5). The controller has less oscillation if it does not enter the highly non-linear range for very low thresholds (<100).

5. EXPERIMENTAL RESULTS

The experiments are performed in the XEEMU simulation environment, which has been described in Section 4. For our setup there are two benchmarks available: BZIP and JPEG-TRANSFORMATION. Both benchmarks have a highly altering access frequency and therefore guarantee a realistic environment.

5.1 Properties and Behavior of Controller

For the first experiment, we investigate the controller's behavior with time. For this purpose, one benchmark is started with a given deadline (180ms). As one can see in Figure 8, the controller realizes the excessive time and therefore decreases the threshold to 0. When the workload changes, the controller increases the threshold in order to enforce its timing constraints. The controller adjusts the threshold smoothly to the slightly alternating workload around 200 cycles to the end of the run.

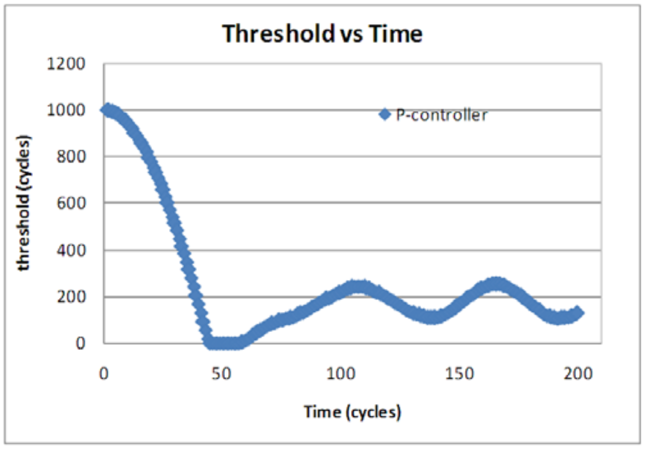


Figure 8: P-controller manipulating threshold with time

Note that the timing constraint is also held with the exact result of 180ms. This is true for all reasonable deadlines, i.e. not shorter then the fastest time without power down mode.

5.2 Performance and Energy Consumption

In this subsection, the power saving improvement with memory power management controller is evaluated. Two benchmarks are used: one is JPEG-TRANSFORMATION and the other is BZIP, a packing program. Several controllers are compared: *W/O PD* has no power-down state at all. *Fixed Thre* uses fixed thresholds of reasonable size (100, 500, 1000). *Ad-hoc Ctrl* represents the Ad-hoc Controller with two different timing set points (0.0185 sec and 0.0200 sec). The same timing set points are used for the P-Controller *P-Ctrl*.

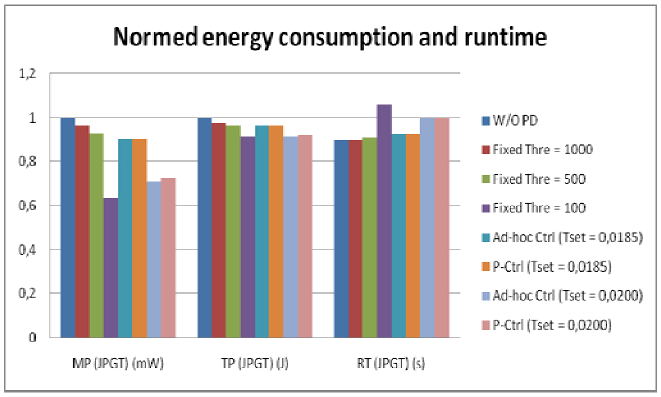


Figure 9: Memory Power Control saves Power (MP-Memory Power, TP-Total Power, RT - Runtime)

Table 2: Power Consumption and Runtime (JPEG-TRAN benchmark) for different types of controllers

	MP (JPGT) (mW)	TP (JPGT) (J)	RT (JPGT) (s)	T_{set} (sec)
W/O PD	629,718	0,0194	0,0180	-
Fixed Thre = 1000	608,7791	0,0190	0,0180	-
Fixed Thre = 500	583,603	0,0187	0,0182	-
Fixed Thre = 100	400,6131	0,0177	0,0212	-
Ad-hoc Ctrl	569,9273	0,0187	0,0185	0,0185
P-Ctrl	568,7966	0,0187	0,0185	0,0185
Ad-hoc Ctrl	446,8824	0,0177	0,0200	0,0200
P-Ctrl	456,9626	0,0179	0,0200	0,0200

As indicated by both Figure 9 and Table 2, runtime (RT) always stays within the constraint, which is set to 18ms and 20ms respectively. The controller can, as long as the timing set point is feasible (timing set point not lower then minimum runtime), reach the desired set point. The memory power dissipation (MP) and the energy consumption for the whole system (TP) are significantly reduced. In contrast the other mechanisms (no power down and fixed threshold) need either more energy, or, if the threshold is set too low, violate the timing constraints.

The experiment is repeated for another benchmark (BZIP) and shows similar outcomes.

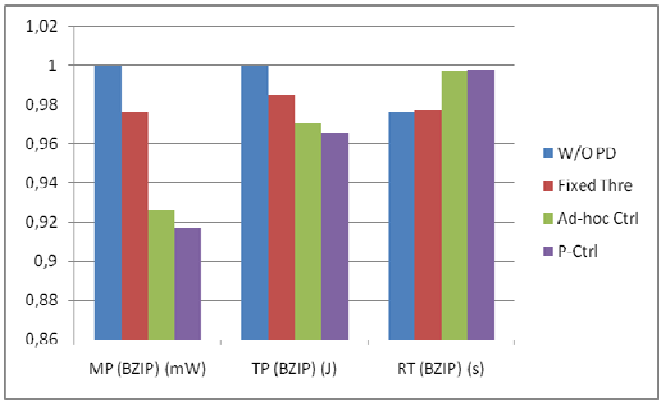


Figure 10: Memory Power Consumption and Runtime for BZIP Benchmark (MP-Memory Power, TP-Total Power, RT - Runtime)

Table 3: Power Consumption and Runtime (BZIP benchmark) for different types of controllers

	MP (Bzip) (mW)	TP (Bzip) (J)	RT (Bzip) (s)	T_{set} (sec)
W/O PD	815,2322	0,2191	0,1757	-
Fixed Thre	796,0312	0,2158	0,1759	-
Ad-hoc Ctrl	755,1279	0,2127	0,1795	0,18
P-Ctrl	747,5318	0,2115	0,1796	0,18

As indicated by both Figure 10 and Table 3, runtime always stays within the constraint, which is set to 180ms.

For both benchmarks, we see that with memory power feedback control, especially with the P-controller, power efficiency improvement can reach as high as 19.3%.

5.3 Properties and Performance of Controller

In this subsection, system stability is tested. For a specific runtime constraint we expect to set threshold within a small range rather than let it oscillate, which can cause instability of the whole system. For both the Ad-hoc controller and P-controller, runtime constraints are set to the same: 0.0200 seconds. The length of an epoch is selected based on the setup in [18] and is set to 10,000 cycles, although (not shown here) the epoch length is varied to prove stability between 10,000 and 1,000,000 with similar results. Figure 11 and Figure 12 clearly indicate that the P-controller is able to bring the system into stable state under this constraint, while the Ad-hoc controller oscillates more. Note that none of the controllers can reach a stable threshold due to the continuously altering access frequency. In addition, a minimum threshold is imposed for the P-controller to prevent the unlinear threshold from changing (Figure 12, minimum threshold allowed is 100).

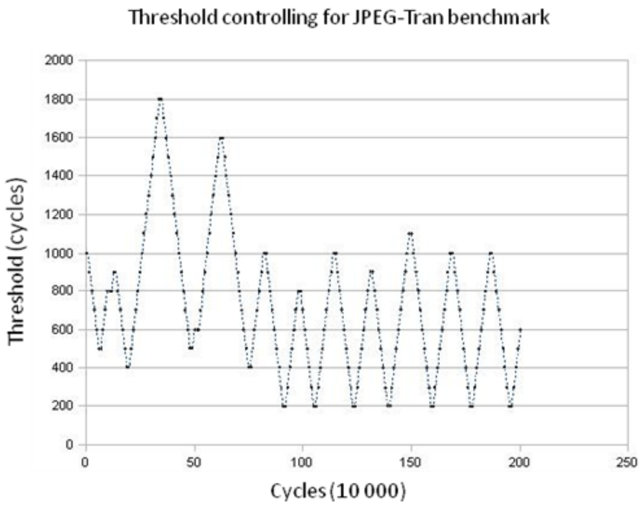


Figure 11: System Stability of Ad-hoc

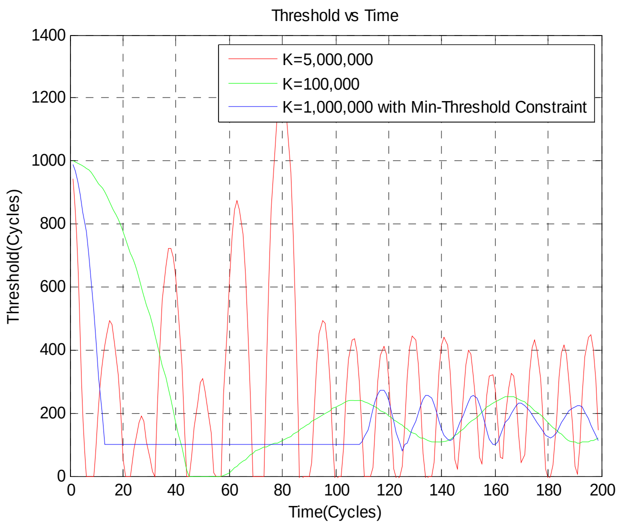


Figure 12: System Stability of P-Controller

6. CONCLUSION

It is well known that putting memory into a lower power mode can save power. However, how to use these available power states by choosing the optimal timing for power mode switching is a challenging task. In this paper, well-established feedback control theory is used to solve this problem.

As a result, theoretical analysis of memory power control system and controller design becomes possible. A memory power control system is proposed, implemented, and explored. By performing extensive experiments with a practical memory model, feasibility of memory power management using feedback is verified. Both an Ad-hoc controller and a P-controller are implemented and evaluated, following feedback control system design approaches.

The main goal to save energy and meet the timing constraints has been achieved by the proposed algorithm. Our experimental results show that memory power efficiency has been improved up to 19.3% compared to common memory control power saving mechanisms. In the meantime, the timing constraints have been met for all benchmarks and timing set points.

7. REFERENCES

[1] A. Patrizio, Memory: The Overlooked Power Issue, InternetNews, 2007.

[2] X. Wang and M. Chen, Cluster-level Feedback Power Control for Performance Optimization, the 14th IEEE International Symposium on High-Performance Computer Architecture (HPCA-14), Salt Lake City, Utah, February 2008.

[3] Y. Wang, K. Ma, and X. Wang, Temperature-Constrained Power Control for Chip Multiprocessors with Online Model Estimation, the 36th International Symposium on Computer Architecture (ISCA-36), Austin, TX, June 2009.

[4] R. Teodorescu and J. Torrellas, Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors, the 35th International Symposium on Computer Architecture (ISCA-35), June 2008.

[5] C. Lefurgy, X. Wang, and M. Ware, Server-level Power Control, the 4th IEEE International Conference on Autonomic Computing (ICAC 2007), Jacksonville, Florida, June 2007.

[6] I. Hur and C. Lin A comprehensive Approach to DRAM Power Management, the 14th IEEE International Symposium on High-Performance Computer Architecture (HPCA-14), Salt Lake City, Utah, February 2008.

[7] V. Cuppu and B. Jacob, Organizational Design Trade-Offs at the DRAM, Memory Bus, and Memory Controller Level: Initial Results, Univ. of Maryland Systems and Comp. Arch. Group Technical Report UMD-SCA-TR-1992, Nov. 1999.

[8] S. Rixner, W Dally, U. Kapasi, P. Mattson, J. Owens, Memory Access Scheduling, Proceedings of the 27th International Symposium on Computer Architecture, Jun. 2000.

[9] I. Hur, C. Lin, Adaptive History-Based Memory Schedulers, Proceedings of the 37th International Symposium on Microarchitecture. Dec. 2004

[10] Z. Zhang, Z. Zhu, X. Zhang, Breaking Address Mapping Symmetry at Multi-levels of Memory Hierarchy to Reduce DRAM Row-buffer Conflicts, Journal of Instruction-level Parallelism (JILP), Vol. 3, October 2001.

[11] Z.Zhang, Z.Zhu, X.Zhang, A Permutation-based Page Interleaving Scheme to Reduce Row-buffer Conflicts and Exploit Data Locality, Proceedings of the 33rd Annual International Symposium on Microarchitecture (Micro-33), Monterey, California, December 10-12, 2000.

[12] W. Lin, S. Reinhardt, D. Burger, Reducing DRAM Latencies with an Integrated Memory Hierarchy Design, Proceedings of the 7th International Symposium on High-Performance Computer Architecture, Jan 2001.

[13] N. Vijaykrishnan, M.Kandemir, M.J.Irwin, H. S.Kim and W.Ye, Energy-driven Inegrated Hardware-software Opitmizations Using SimplePower, Proceedings of the 27th Annual International Symposium on Computer Architecture, pp95-106, June 2000.

[14] T. Li and L. K. John, Run-time Modeling and Estimation of Operating Power Consumption, Proceeding of the International Conference on Measurement and Modeling of Computer Systems, pp160-171, June 2003.

[15] J. da Silvia, F. Catthoor, D. Verkest, and H. de Man. Power Exploration for Dynamic Data Types Through Virtual Memory Management Refinement. In Proceedings of the International Symposium on Low Power Electronics and Design, pp 311-316, Aug. 1998.

[16] B. Nikolaos I. Hajj, C. Polychronopoulos, and G. Stamoulis. Architectural and Compiler Support for Energy Reduction in the Memory Hierarchy of High Performance Microprocessors. In Proceedings of the International Symposium on Low Power Electronics and Design, pp70-75, 1998.

[17] Home of XEEMU, http://www.inf.u-szeged.hu/xeemu/page_what_is.html

[18] X. Fan, C.S. Ellis, and A. R. Lebeck., "Memory Controller Policies for DRAM Power Management", In Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED), 2001.

[19] D. Lammers. IDF: Mobile Rambus spec unveiled. EETimes Online, February 1999. http://www.eetimes.com/story/OEG19990225S0016.

[20] X. Li, Z. Li, Y. Zhou, and S. Adve. Performance Directed Energy Management for Main Memory and Disks, ACM Transactions on Storage (TOS), Vol. 1, No 3, August 2005.