Coarse-Grained DRAM Power Management

Jin Hwan Park*, Sarah Wu* and Baback Izadi+
Department of Computer Science*
Department of Electrical and Computer Engineering+
State University of New York at New Paltz
New Paltz, NY 12561, U.S.A.

Abstract– *This paper presents an efficient system level power saving method for DRAM with multiple power modes. The proposed method is based on the power aware scheduling algorithm that controls DRAM modules in coarse grain in which the scheduler assigns appropriate power modes to memory banks at context switching time. The method controls the transition of multiple power modes, which is currently available technology, based on the history of gaining processor and memory bank usage of each process. The experimental results demonstrate the efficiency of the proposed schemes in multiprogramming environment.*

Keywords: low power, DRAM, scheduling, power mode transition

1. Introduction

Recent growing technology of mobile, handheld, and embedded systems requires increasing research efforts on developing low power management methods in several different aspects. Among those aspects, reducing energy consumption in the memory hierarch has been an important issue since the memory system is a major power consumption component of the system. Cost of memory is inexpensive now, but the cost of energy consumed by memory system is high for those battery dependable systems. Together with low power cache design techniques [8, 9, 15, 17] and power efficient on-chip memory system design techniques in the embedded systems that replace cache with small on-chip SRAM module [1, 2, 3], methods for reducing power in the DRAM have been proposed in the literature [4, 5, 9, 11, 16]. These efforts are based on the recent DRAM technology that makes DRAM module operate in different power modes. For instance, RAMBUS's RDRAM technology provides four different power modes such as active, standby, napping, and shut-down [12]. Of course each power mode consumes different amount of energy, and reactivation from a lower power mode to a higher power mode requires reactivation cost as additional cycles. Thus it is important to design appropriate power management scheme for utilizing these manufacturer provided technology to the applications including desk-top to embedded systems. In DRAM power mode management researches, both hardware and software based approaches have been proposed in the literature [4, 5, 11, 16]. In these works, memory bank partitioning method was used for utilizing the technological advantages of handling different power modes by turning down unused banks' power. Approaches for selecting appropriate power modes for partitioned memory banks are grouped into compiler based, hardware assisted, and operating system based methods. While each method has its own pros and cons, we focus on the operating system based method for reducing power of DRAM in this paper. More specifically, we develop an efficient scheme for handing multiple DRAM power modes for reducing energy consumption at context switching time that does not require additional hardware support which consumes extra energy. Since the approach is a coarse-grained, there exist not much overhead of the scheme; only the scheduler is responsible for the operation at the context switching time. Our study reflects full multiprogramming environment with simulated systems under the assumption of contiguous memory allocation without virtual address spaces. Since the scheme is generic which does not aim a specific commercial DRAM model nor a specific platform, it can be applied to a variety of architectures. Using the simulated system model, we trace the behavior of multiprogramming completely for utilizing the benefit of multiple power modes provided by manufacturers.

The remainder of this paper is organized as follows. Related work is reviewed briefly in Section 2. In Section 3, DRAM power modes and memory system structure for the experiment are described. The proposed scheme for reducing DRAM power dissipation is described in Section 4, followed by the experimental results in Section 5 and conclusion in Section 6.

2. Related Work

Efforts for reducing energy consumption by selectively turning down memory banks' power are found in the aspects of compiler based, hardware assisted, operating system based. Delaluz, et. al. [5] proposed a compiler-based method for reducing power on DRAM modules (banks) by determining idle periods of each memory module at compile time. Though this method does not have resynchronization cost overhead since it completely predicts the memory bank usage at compile time for the target data, it also has several drawbacks. The scheme is conservative since not all information are available/analyzable at compile time, only arrays and loop constructs are involved in the practice, and only single programming case was considered. Thus the approach handles very limited situations and it assumed the direct mapping of address to the physical memory (i.e., without virtual memory space). Udayakumaran, et al. [16] proposed a compiler-based scheme in which data allocation and memory partitioning are performed jointly. It uses application specific partitioning of the scratchpad memory based on the temporal locality. Self-monitored run time approaches that need hardware support were proposed in [5]. The idea is that the memory system automatically transitions to lower power modes based on the information captured by the supporting hardware. Three proposed schemes were based on adaptive threshold value (ATP), fixed threshold value (CTP), and history of inter-access time (HBP). Among them, the history-based approach showed the best performance. In fact, in ATP, no direct transition of power modes are allowed, but only through a higher mode to the one lower mode serially are allowed. The ATP method needs extra hardware for making the adaptive decision that consumes extra energy, and yields the resynchronization cost with mis-prediction. The CTP method needs a counter for each memory bank, and the transition method is the same as the one in the ATP; i.e., serial from a higher to one lower, but it showed better performance than the compiler-based approach proposed in [5]. The HBP method can transit power mode of a bank directly from active to a lower power mode, by simply assuming that the next inter-access time is the same as the previous inter-access time. Since the compiler based approaches have certain limitations and hardware oriented approaches need additional energy consumptions on those supporting hardware for checking bank usage, alternative methods based on the operating system has started appearing in the literature. In fact, there have been many operating system based power reduction schemes for processors and I/O devices, but very few works has focused on utilizing multiple DRAM power modes for reducing energy consumed by memory system. Lebeck, et al. [11] proposed a scheme for reducing DRAM energy by power aware page allocation algorithm. Underlying idea is that an application's pages are allocated into a minimum number of chips (banks) and unused banks are in low powered modes during the execution of the application. The work considered only single programming case and there was no consideration of multiprogramming situations. Delaluz, et al. [4] proposed a scheduler-based approach of turning down the power of unused memory banks. The method uses a bank usage table that managed by the operating system, and resets all banks' power modes at the context switching time. It is a prediction method with mis-prediction risks since the setting is based only on the one previous time's bank usage. These operating system based methods can be used with hardware methods if hardware cost (additional energy consumption) is affordable.

3. DRAM Power Modes and Memory System Structure

Partitioning the memory into smaller sized components can reduce the power consumption significantly. In this paper, we assume the memory system consists of smaller modules in rows (banks) and columns. For the simplicity, we further assume that each bank consists of one DRAM module that is operated in four different operation modes; i.e., active (attention), standby, napping, and power-down modes. Figure-1 illustrates the memory system structure under these assumptions, and Figure-2 illustrates the DRAM power mode transitions and costs found in the data sheet from RMBUS [12]. In this paper, we use normalized values for measuring energy consumption and

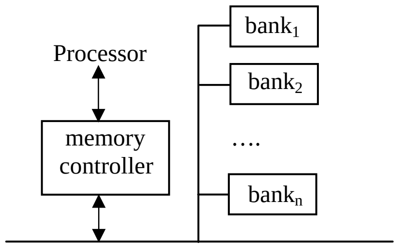
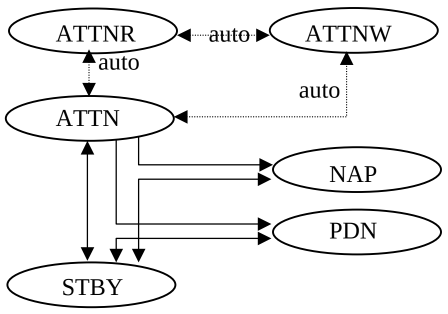


Figure-1. Memory system structure

resynchronization costs. Thus the values shown in Figure-2(b) are normalized to constants x and c. In fact, 8MB RDRAM module consumes 3.57nJ per cycle in active mode in 3.3V, 2.5ns clock cycle time technology. In the figure, refined six power modes are illustrated, but only four modes are distinguishable because once in ATTN mode, the device will automatically transition to the ATTNW and ATTNR modes as it receives WR and RD commands. Thus in this paper, we use term “active” for representing the three attention modes shown in the figure.



ATTNR: Attention read mode
ATTNW: Attention write mode
ATTN: Attention mode
STBY: Standby mode
NAP: Nap mode
PDN: Power-down mode

(a) Power mode transition direction

Power mode	Power consumption	Resynchronization cost
ATTN	1x nJ	0c ns
STBY	0.23x nJ	1c ns
NAP	0.089x nJ	15c ns
PDN	0.0014x nJ	4,500c ns

(b) Energy consumption and resynchronization cost

Figure-2. RDRAM power modes

We consider the following power mode transition directions in this paper; active to standby, active to nap, and active to power-down directly, and the resynchronization cost from each of these low power modes to active mode are shown in Figure-2(b). We ignore all other directions and costs in this paper.

In the ideal case in which the resynchronization starts in advance with no latency overhead, formula for measuring the energy consumed in a bank during its idle time (in a lower power mode) is

$$(T_{idle} - T_{resyn})\ E_{low} + T_{resyn}\ E_{act}$$

where, T_{idle} is the bank’s idle time, T_{resyn} is the resynchronization cost, E_{low} is the unit energy consumed in the lower power mode, and E_{act} is the unit energy consumed in the active power mode. The term $T_{resyn}*E_{act}$ represents the energy needed during the resynchronization. Based on the above formula, the gain of using a lower power mode during the idle time in each bank is:

$$T_{idle}\ E_{act} - [(T_{idle} - T_{resyn})\ E_{low} + T_{resyn}\ E_{act}\ ]$$

The term $T_{idle}*E_{act}$ represents the energy consumption without using multiple power modes; i.e., all modules are in active continuously. Thus when $T_{idle} > T_{resyn}$, we have the energy gain:

$$(T_{idle} - T_{resyn})\ (E_{act} - E_{low}) \qquad (1)$$

4. Context Switching Time Power Management

The main idea of controlling DRAM power at context switching time is that the memory array is divided into smaller banks and unused banks are power downed during the entire period of the current time quantum (or, until the processor is preempted based on the event). Thus the scheme is coarse grained; i.e., during the current time quantum (or, until CPU preemption) no power transitions occur. Since we can implement the scheme with only operating system’s modules, this approach does not require special hardware support that consumes extra energy. A simple way of utilizing the above idea is using the contiguous memory allocation to processes. In fact, the compiler-based approaches allocate contiguous data memory for arrays into the same bank as much as possible [5], power aware paging scheme [11] allocates frames for each process into contiguous frames that are physically closed so that pages are distributed to as less banks as possible. In our experiment with a simulated system, processes are loaded into contiguous memory spaces (with best-fit) that are divided into banks, and at each context switching time, unused banks (banks except the ones assigned to the running process) are power downed. At loading time, the operating system checks which banks are allocated to the process, and keep the information in the PCB of each process. Following three possible cases are considered in the experiment; multiple processes are allocated to a bank, only one process is allocated to a bank, and a process is allocated to multiple banks (i.e., it exceeds the boundary of a bank). Thus, it is not difficult to figure out which banks will be unused during the new time quantum until the CPU is preempted due to the quantum expiration or some event occurrence (e.g., I/O). When the processor is preempted, banks that are allocated to the running process (switching out process) have choices of selecting lower power modes based on the selection algorithm. The idea used in the selection algorithm is that the scheduler checks the process waiting time history (i.e., how frequently it has been scheduled to CPU) that is kept in PCB, and selects an appropriate lower power mode to each idle bank. The selection algorithm is described in section 4.2.

4.1 Energy Gain

The simplest scheme for saving power of DRAM banks is turning down unused bank's power to the lower power mode if there exist only two power modes; i.e., active and low. But the consideration is whether the turning down to the lower mode actually yields the gain or not. In the case of the switched out process regains CPU within very short period of time, turning down to the lower mode yields more energy consumption because of the resynchronization cost. For the ideal case in which the resynchronization starts as early as possible for eliminating the latency overhead (i.e., extended time), energy gain is computed using formula (1) in Section 3. Since our proposed scheme is coarse-grained and thus it does not start the reactivation at any time we should use a different formula for computing the energy gain of using a lower power mode during the bank's idle time. For the general case without considering the early resynchronization, the energy gain is

$$T_{idle}\,E_{act} - (\,T_{idle}\,E_{low} + T_{resyn}\,E_{act})$$

with the overhead of extended time with the ratio of

$$(T_{idle} + T_{resyn}) / T_{idle}$$

Thus, when $T_{idle} > T_{resyn}(E_{act}/(E_{act}-E_{low}))$, we have the energy gain:

$$T_{idle}\,(E_{act} - E_{low}) - T_{resyn}\,E_{act} \qquad (2)$$

Figure-3 illustrates the energy consumption during the bank's idle time in two cases such as the ideal case and the case with the latency overhead. During the resynchronization, energy for the active mode is consumed as illustrated in Figure-3(b).

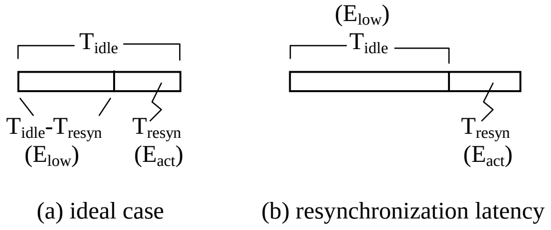


Figure-3. Power consumption during idle time of a bank

From formula (2), we can derive the conditions for converting to each of standby, nap, and power-down modes easily. For example with considering only active and nap modes and 2.5 ns clock cycle time, when a bank's idle time is bigger than 82.42 ns converting to nap mode has energy gain. The condition for checking whether converting to the lower power mode yields the gain or not heavily depends on the bank's future idle time. We describe the method of measuring each bank's idle time in section 4.2.

4.2 Selection from Multiple Power Modes

Based on the formula (2) in section 4.1, we can generalize the power mode selection scheme for the multiple power modes. For measuring each bank's future idle time, we use the accumulated average of each process's waiting time for predicting the period of the future idleness. In fact there have been proposed many prediction methods for saving energy of devices in the literature, but we use the accumulated average method for the simplicity in this paper. Algorithm-1 describes the multiple power mode selection scheme at context switching time. Measuring the accumulated average waiting time for processes is also described in the algorithm. Since we keep the information of allocated bank(s) to a process in the PCB, we can easily find the corresponding bank(s) for updating the power mode.

Algorithm-1.

```
Assume: In each PCB, wall clock times for CPU
        release/regain are kept.

for each process,
  {at context switching in time,
     {compute current wait time (Tw):
        Tw = Treg – Trel;
           //Treg and Trel represent CPU regain
           //and release times respectively
      update accumulated ave. wait time (Taw):
        Taw = [Taw * (n-1) + Tw] / n;
           //n represents number of times this
           //process has gained CPU so far
      }

   at context switching out time,
      {select a power mode from standby, nap,
        and power-down:
          find max of:
             $Taw (E_{act} – E_{stby}) – T_{resyn-stby} E_{act}$
             $Taw (E_{act} – E_{nap}) – T_{resyn-nap} E_{act}$
             $Taw (E_{act} – E_{pdn}) – T_{resyn-pdn} E_{act}$

          if (max > 0)
             set the corresponding bank's power
             mode to max's mode
          else
             no operation //keep active
       }
  }//for each process
```

In Algorithm-1, the accumulated average waiting time (Taw) for the process is used for the predicted idle time of the bank (T_{idle}). Thus Taw is an approximation of T_{idle}. Terms for the energy in the algorithm such as E_{act}, E_{stby}, E_{nap}, and E_{pdn} represent unit energy consumptions in active nap, and power-down modes respectively. Terms for the time such as $T_{resyn-stby}$, $T_{resync-nap}$, and $T_{resyn-pdn}$ represent resynchronization latencies (times) for standby, nap, and power-down modes respectively.

Since the algorithm checks PCB's of CPU releasing and gaining jobs only at the context switching time, the overhead of using the scheme is minimal, but it yields a remarkable amount of gain.

5. Experimental Results

The proposed scheme was tested on the simulated system. The simulation system model includes RISC based CPU simulator and traditional multiprogramming/time-sharing operating system that manages queues for holding ready processes and I/O blocked processes. CPU is preempted when the current time quantum expires or an I/O event occurs. The long-term scheduler (loader) selects jobs from job pool (we assume that all jobs arrived at time 0 and are available). In the system, memory is divided into 6 banks (one module per bank) and contiguous allocation with best-fit is used. By running 120 programs in the simulated system, we traced the behavior of the multiprogramming.

For the performance measurement, we tested following cases for demonstrating the efficiency of the proposed scheme:

do-nothing: all 6 banks are in active mode
active-nap: idle banks converts to nap mode only
active-standby-nap: idle banks converts to either standby mode or nap mode

In the later two cases, the scheme checks condition for the positive energy gain and keeps the active mode when the condition does not meet, as described in formula (2) and Algorithm-1. We excluded power-down mode from the experiment since the average execution time of testing programs was much shorter than the resynchronization cost of power-down mode. Following figures show the performance of the scheme using normalized values.

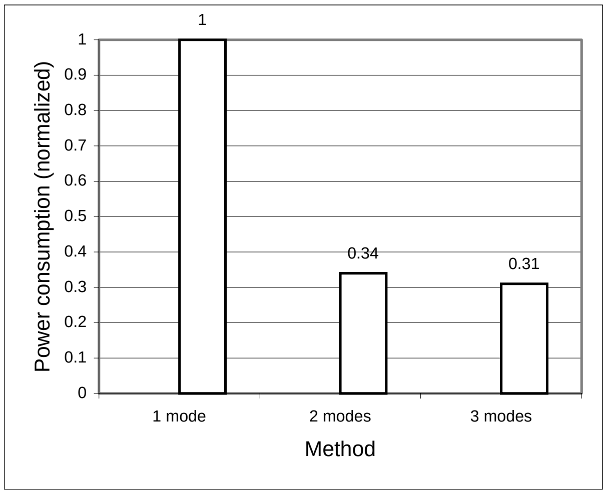


Figure-4. Power consumption v.s. methods

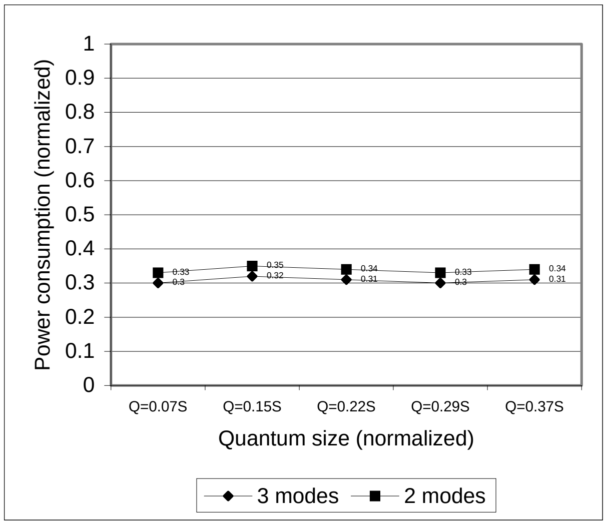


Figure-5. Power consumption v.s. time quantum

Figure-4 shows that using active-nap modes and using active-standby-nap modes save DRAM power 66% and 69% respectively. Thus it reflects the concept that using more power modes saves more energy. Figure-5 shows the results of applying varying size of the time quantum. Quantum values shown in the graph are normalized to the average execution time of processes (s). In our experiment, varying size of the time quantum did not significantly affect the performance of the scheme, and showed better performance of 3-modes method with all quantum values tested. Though our experiment was limited the results successfully demonstrated the benefit of using context switching time power management scheme that does not require any special hardware support. As other operating system based approaches, the proposed scheme can be used with hardware based approaches for gaining further energy saving.

6. Conclusion and Discussion

In this paper, we described an operating system based approach for saving DRAM power. The approach is coarse-grained since it operates only at context switching time without causing considerable overheads. Formulas for evaluating the condition of gaining power are developed and an algorithm for selecting appropriate power mode is designed based on that. For measuring each process's CPU accessing frequency, we used a simple accumulated average method that approximates the predicted idle time of the bank. The experimental results showed successful performance of the scheme in multiprogramming environment. Though our tests are limited yet, it demonstrated the benefit of using the scheme that can be used with other approaches such as hardware supported approaches shown in the literature.

Future study should focus on developing high performance prediction method for measuring future idle time of the bank and experimenting on real systems.

REFERENCES

[1] L. Benini, A. Macii, E. Macii, and M. Poncino, "Synthesis of Application-Specific Memories for Power Optimization in Embedded Systems," Proc. of 37th Design Automation Conference (DAC 2000), pp. 300-303, June 2000.

[2] Benini, A. Macii, and M. Poncino, "A Recursive Algorithm for Low-Power Memory Partitioning," Proc. of the 2000 Intl. Symposium on Low Power Electronics and Design, pp. 78-83, 2000.

[3] Y. Cao, H. Tomiyama, T. Okuma, and H. Yasuura, "Data Memory Design Considering Effective Bitwidth for Low-Energy Embedded Systems," Proc. of IEEE/ACM Intl. Symposium on System Synthesis (ISSS'2002), Oct. 2002.

[4] V. Delaluz, A. Sivasubramaniam, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "Scheduler-Based DRAM Energy Management," Proc. of 39th Design Automation Conference (DAC'2002), pp. 697-702, 2002.

[5] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin, "DRAM Energy Management Using Software and Hardware Directed Power Mode Control," Intl. Symposium on High Performance Computer Architecture (HPCA-7), pp. 159-169, Jan. 2001.

[6] T. D. Givargis, F. Vahid, and J. Henkel, "A Hybrid Approach for Core-Based System-Level Power Modeling," Proc. of the 2000 Conference on Asia and South Pacific Design Automation, pp. 141-146, 2000.

[7] P. Grun, N. Dutt, and A. Nicolau, "Memory Aware Compilation Through Accurate Timing Extraction," Proc. of 37th Design Automation Conference (DAC'2000), pp. 316-321, June 2000.

[8] K. Inoue, A. G. Moshnyaga, and K. Murakami, "Trends in High-Performance, Low-Power Cache Memory Architectures," IEICE Trans. on Electronics, Vol. E85-C, No. 2, pp. 304-314, Feb. 2002.

[9] M. J. Irwin and V. Narayanan, Low Power Design: From Soup to Nuts, ISCA'2000 Tutorial, 2000.

[10] M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and W. Ye, “Influence of Compiler Optimizations on System Power,” Proc. of 37th Design Automation Conference (DAC 2000), pp. 304-307, June 2000.

[11] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis, “Power Aware Page Allocation,” Proc. of 9th Intl. Conference on Architectural Support for Programming Languages and Operating Systems, pp. 105-116, Nov. 2000.

[12] Rambus Inc. http://www.rambus.com/

[13] M. Sekine and J. H. Park, “An Efficient Low Power Hybrid Memory System for Mobile Computers,” Proc. of the 2001 Intl. Conference on Parallel and Distributed Processing Techniques and Applications, pp. 979-985, June 2001.

[14] Y-H Lu, L. Benini, and G. D. Micheli, “Operating-System Directed Power Reduction,” Proc. of Intl. Symposium on Low Power Electronics and Design, pp. 37-42, July 2000.

[15] W-T Shiue and C. Chakrabarti, “Memory Exploration for Low Power, Embedded Systems,” Proc. of 36th Design and Automation Conference (DAC’99), June 1999.

[16] S. Udayakumaran, B. Narahari, and R. Simha, “Application-Specific Memory Partitioning for Low Power Consumption,” Proc. of Workshop on Compiler and Operating Systems for Low Power (COLP 2002), Sept. 2002.

[17] Y. Zhu and F. Mueller, “Preemption Handling and Scalability of Feedback DVS-EDF,” Proc. of Workshop on Compilers and Operating Systems for Low Power (COLP 2002), Sept. 2002.

[18] W-C Cheng and M. Pedram, “Power-Optimal Encoding for a DRAM Address Bus,” IEEE Trans. on VLSI Systems, Vol. 10, Issue 2, pp. 109-118, April 2002.