A Performance Conserving Approach for Reducing Memory Power Consumption in Multi-Core Systems*

Juan Fang[†], Jiajia Lu[‡] and Mengxuan Wang[§]

*Faculty of Information Technology,
Beijing University of Technology,
Beijing 100124, P. R. China*

[†]fangjuan@bjut.edu.cn

[‡]lu_jiajia@emails.bjut.edu.cn

[§]mengxuanw13@emails.bjut.edu.cn

Hui Zhao

*Computer Science and Engineering Department,
University of North Texas, Texas, USA
hui.zhao@unt.edu*

Received 28 March 2016

Accepted 26 July 2018

Published 29 August 2018

With more cores integrated into a single chip and the fast growth of main memory capacity, the DRAM memory design faces ever increasing challenges. Previous studies have shown that DRAM can consume up to 40% of the system power, which makes DRAM a major factor constraining the whole system's growth in performance. Moreover, memory accesses from different applications are usually interleaved and interfere with each other, which further exacerbates the situation in memory system management. Therefore, reducing memory power consumption has become an urgent problem to be solved in both academia and industry. In this paper, we first proposed a novel strategy called Dynamic Bank Partitioning (DBP), which allocates banks to different applications based on their memory access characteristics. DBP not only effectively eliminates the interference among applications, but also fully takes advantage of bank level parallelism. Secondly, to further reduce power consumption, we propose an adaptive method to dynamically select an optimal page policy for each bank according to the characteristics of memory accesses that each bank receives. Our experimental results show that our strategy not only improves the system performance but also reduces the memory power consumption at the same time. Our proposed scheme can reduce memory power consumption up to 21.2% (10% on average across all workloads) and improve the performance to some extent. In

*This paper was recommended by Regional Editor Piero Malcovati.

[†]Corresponding author.

This is an Open Access article published by World Scientific Publishing Company. It is distributed under the terms of the Creative Commons Attribution 4.0 (CC-BY) License. Further distribution of this work is permitted, provided the original work is properly cited.

the case that workloads are built with mixed applications, our scheme reduces the power consumption by 14% on average and improves the performance up to 12.5% (3% on average).

Keywords: Dynamic bank partitioning; power consumption; multi-core memory systems.

1. Introduction

With the widespread use of chip multiprocessors and rapid growth of the I/O speed, multiple applications running in parallel have increasing demand in the accessing speed and the capacity of the memory system. Coupled with the *memory wall* problem, it is imperative to optimize the design of DRAM memory system. To improve the system performance, manufactures tend to increase chip integration density and memory bandwidth. However, such practice leads to even higher memory power consumption. It is shown in a recent study that memory system power consumption is close to or even greater than that of the processor, which is about 19~41% of the whole system power.¹ Many prior schemes have been proposed to reduce the processor power consumption and now the focus has been shifted to reducing main memory power. However, the challenge is that we want to reduce the effect of a power reduction scheme on the performance to the minimum.

Modern memory systems mainly take advantage of the spatial locality to increase bandwidth and reduce the access latency, in order to reduce the memory power consumption. However, with the increasing number of cores on chip, memory accesses from different cores interfere with each other. Such interference ruins the original characteristics of an application's memory accesses and leads to low spatial locality, greatly reducing system performance and increasing power consumption.² Experiments in Ref. 3 show that when the LBM benchmark was running alone, the hit rate could reach up to 98% and its memory accesses exhibited large amount of special locality. However, when four LBM benchmarks were run together, the hit rate dropped to 50%. This shows that we cannot simply base on the spatial locality of an application running alone to design strategies to reduce power consumption. Shah *et al.* discuss the use of timing analysis to limit the interference between applications in shared resources.^{3,4} Bank partitioning^{5,6} maps memory accesses from different cores to different banks, which isolates the memory access streams from different cores and effectively eliminates the interference. In bank partitioning, one core can only access its specific banks and the number of banks that each core owns is equal. The bank partitioning limits the number of banks a core can access, because it takes no consideration of the different characteristics of memory access streams from different cores.

Zheng *et al.* pointed out that page policy had a significant impact on memory power consumption and the optimal page policy was application-dependent.⁷ Single page policy is not applicable in multi-core system, because the characteristics of memory accesses from different cores are different. The open page policy is suitable for banks whose memory access streams are intensive, which can effectively employ spatial locality to reduce the delay and operation power. The close page policy is

more effective on banks whose memory requests are not frequent, which would make the row buffer idle immediately after the column access and greatly increase the probability of the rank entering the power down mode to reduce background power. In multi-core systems, dynamic bank partitioning keeps the original characteristics of each core, which provides the opportunity for exploiting the spatial locality to reduce memory power consumption.

Combining bank partitioning and page policy, we introduce a new scheme called dynamic bank partitioning (DBP) and adaptive page policy. DBP allocates the optimal number of banks to each core, which not only limits the interference among applications effectively, but also fully takes advantage of bank level parallelism. Adaptive page policy dynamically selects an optimal page policy for each bank according to the characteristics of memory accesses that each bank receives.

The rest of this paper is organized as follows: Sec. 2 provides background information of the DRAM memory systems. Section 3 presents our proposed schemes in detail. Section 4 describes the evaluation methodology and Sec. 5 provides the results. Section 6 reviews the related work and we conclude this paper in Sec. 7.

2. Background

2.1. The DRAM system

Modern DRAMs consist of one or more memory channels. Each channel has independent memory address space, data and instruction buses. A single channel contains one or more ranks and each rank has multiple banks. A bank is the basic structure of the memory system, which is a two-dimensional array that contains rows and columns.⁸ Each bank has a row buffer and a data array. The target row must be loaded to the row buffer in advance. There are three major steps required when accessing a data element:

- (1) *precharge*: write the row buffer's data back and make the row buffer idle, which is coupled with row activate step.
- (2) *row activate*: according to the row address, activate the target row and put the data of the target row into the row buffer. Before activating the target row, we must ensure that the row buffer is in idle state. Otherwise, we need to precharge the row first.
- (3) *column access*: according to the column address, read or write the data from the row buffer.

2.2. DRAM power model

DRAM power consumption is divided into four categories: background power, the operation power, the read/write power and I/O power. Regardless of memory access or not, background power is consumed all the time. If all the banks of the rank are idle, the rank can enter into power-down mode to save background power. The background power is the smallest in power-down mode. When the bank performs row

activate or precharge, it consumes operation power. The read/write power is consumed when we read or write data during the column accesses. The I/O power is consumed by the transactions and terminations.⁹ This paper mainly studies background power and operation power.

2.3. Page management policy

In modern DRAM systems, each bank has a row buffer. In a bank, only one row can be in row active state. If the target row has already been loaded into the row buffer when we access the DRAM, then only column access is necessary, which is called a *row hit*. Otherwise, it is called a *row conflict*. In the case of a *row conflict*, we have to first precharge the opened row, activate the target row, and then perform the column access. The latency of the *row conflict* caused is $2 \sim 5$ times than that of *row hit*.¹⁰

The spatial locality of memory access streams can be measured by the row hit ratio. There are two kinds of page policies for the row buffer: the open page policy and close page policy. The difference between the two page policies is the time to precharge. The open page policy does not precharge the row in the row buffer until a row conflict happens or some other operation such as refresh. On the contrary, close page policy precharges a bank immediately after a column access, regardless of whether the next access is row hit or row conflict. So the open page policy works better for high memory intensive applications with high spatial locality to reduce the number of active rows and precharge, which reduces operation power. The close page policy is often employed on low intensive applications, which not only reduces the background power but also increases the opportunity for the rank entering into the low power mode.

2.4. Bank level parallelism

The length of a memory command queue is limited. If adjacent DRAM accesses to different banks can be executed in parallel, the latency can be overlapped. Multiple independent memory access streams to different banks run in parallel, called Bank Level Parallelism (BLP), which can hide the latency by pipelining memory accesses. For high memory intensive applications with low spatial locality, assigning the open page policy to such applications would result in more row activate and precharge operations. If we run such applications in parallel, the latency would be greatly reduced. For low memory intensive applications whose memory access streams are rare and the latency between two adjacent accesses is long, running such applications in parallel has no obvious benefit. Applying BLP with discretion can reduce the latency and improve the system performance.

3. Our Proposals

3.1. Overview of our proposal

In this paper, our strategy mainly includes two parts. First of all, we propose DBP instead of equal bank partitioning, which assigns banks according to the memory

accesses characteristics of each application. DBP not only effectively eliminates the interference among applications, but also fully takes advantage of bank level parallelism. After eliminating the inter-application interference, the characteristic of memory access streams from each application is preserved. This provides new opportunity to use adaptive page policy to reduce the memory power consumption. Second, we propose an adaptive page policy to dynamically allocate an optimal page policy for each bank according to their memory characteristics. By integrating DBP with adaptive page policy, our work aims at reducing the memory power consumption and improving the system performance simultaneously.

3.2. Applications' access memory behavior analysis

Memory accesses from applications are notoriously difficult to predict,¹¹ but Sherwood *et al.* pointed out that programs can have considerably different behavior depending on which portion of execution is examined. More specifically, it has been shown that many programs execute as a series of phases, where each phase may be very different from the others, while still having a fairly homogeneous behavior within a phase.¹² Each application is run separately and the output shows the classification of the application based on the memory access behavior within each interval. Figure 1 shows the runtime grouping of four applications. From Fig. 1, it can be seen that an error occurs only when the application is regrouped, and it is re-corrected at the next time interval.

3.3. Dynamic bank partitioning

When multiple applications run in parallel, memory access streams from different applications interleave and interfere with each other. Because memory accesses from one application are randomly inserted into other applications' memory access queue, this may result in continuous row buffer hit being changed to row buffer conflicts,

Fig. 1. Runtime application grouping.

which would greatly compromise the advantage of spatial locality. Bank partitioning divides memory internal banks among cores by allocating different page colors to different cores, which isolates different applications' memory access streams and effectively eliminates the inter-application interference.

However, bank partitioning takes no consideration of different demands on bank amount from different applications. To some extent, it restricts available bank amount to an application and reduces the bank level parallelism. Therefore, we propose DBP to take full advantage of bank level parallelism.

DBP dynamically assigns different rules of partition to applications based on memory access's characteristics of each application. DBP does not distribute specialized banks to applications whose memory access streams are not frequent. This is because such applications just generate less memory access and do not interfere much with other applications. In addition, as the number of banks in the memory system is limited, we try to allocate more banks to applications which are very sensitive to bank-level parallelism. For applications whose memory access streams are rare, they can access all banks in the memory. On the contrary, for applications whose memory access streams are intensive and with high spatial locality, we isolate their memory access streams with other intensive applications to eliminate the interference, in order to preserve their original memory characteristics of applications and take advantage of spatial locality. We try our best to provide as many banks as possible to increase bank level parallelism for memory intensive applications with low spatial locality, which are relatively sensitive to bank level parallelism.

The detailed implementation of DBP is as follows:

First of all, we monitor all applications' memory access characteristics in each interval. Memory intensity and spatial locality are two major factors to describe memory characteristics. We use the number of memory accesses in the interval (accesses) to measure the memory intensity and use row buffer hit rate (hitRate) to weigh the spatial locality.

Then, at the beginning of the next interval, we divide applications into groups depending on the statistics we collected in the previous interval. The dividing rule is shown in Table 1. $Accesses_t$ and $hitRate_t$ in Fig. 3 are two thresholds to decide how to divide the group. First, we divided applications into high memory intensive (HMI) group and low memory intensive (LMI) group depending on memory accesses of each application. Second, we further classify memory intensive group into high line hit group (HLH) and low line hit (LLH) group based on the applications hitRate. For nonintensive group, because memory access streams of these applications are rare, there is no need to consider row buffer hit rate and we do not further divide non-intensive applications.

Finally, we make bank partitioning rule for each group. According to the previous step, the applications are divided into three groups. We specify a bank partitioning unit (BPU, $BPU = \frac{\text{the total number of banks}}{\text{the number of cores}}$).

Table 1. The rule of DBP classifying the application group.

Definition: accesses_i and hitRate_i are two thresholds N : the number of applications
Begin for ($i = 1$ to N) if $\text{accesses}_i < \text{accesses}_t$ Application i is assigned to the low memory intensive group (LMI) else if ($\text{hitRate}_i < \text{hitRate}_t$) Application i is assigned to low line hit group (LLH) else Application i is assigned to high line hit group (HLH) End

The first case, if all the applications in the group are memory-intensive applications. For applications in the HLH group, we allocate BPU colors to each core to isolate memory access streams from different cores, which effectively eliminates the interference among cores and provides new opportunity for open page policy. For applications in LLH group, the low spatial locality results in more row buffer conflicts, which significantly increases the memory latency. As a result, the memory access streams are blocked in the command queue and cannot receive a response for a long time. Some threads are even starved to death. In this situation, we allocate 2^* BPU colors to two applications in LLH group to fully exploit the bank level parallelism. The parallelism between memory banks can hide the latency by pipelining memory accesses. So for such applications, parallelism helps in reducing memory latency.

The second case, if all applications are of low memory intensity, we also allocate BPU colors to each core to isolate memory access streams from different cores. Because for such applications, their memory access streams are rare and the interval between two adjacent accesses is larger, there is no need to parallel execute memory accesses.

The third case, the group has both HMI applications and LMI applications. Applications in LMI group can evenly access all the banks in the memory system. We do not allocate special banks to LMI group because of their rare memory access and weak aggression to memory-intensive applications. Furthermore, the amount of banks is limited and we try to provide more banks to applications which are sensitive to bank-level parallelism. In order to isolate memory access streams with other intensive applications to eliminate the inter-application interference, we allocate BPU colors to each application in HLH group. We allocate 2^* BPU colors to two applications in LLH group and the parallelism between memory banks can hide the latency by pipelining memory accesses.

We apply our strategy periodically at fixed-length time intervals. At the beginning of the next interval, we classify the group according to the historical data collected in the previous interval. Then we assign the different bank partitioning

Table 2. The rule of DBP.

Definition:	HMI: high_memory_intensive benchmark LMI: low_memory_intensive benchmark HLH: high_memory_intensive and high_line_hit benchmark LLH: high_memory_intensive and low_line_hit benchmark N: the number of applications in simulation BPU: bank partitioning unit MI: the ratio of HMI in the workload
Begin	if MI is 0% Allocate an equal number to each core else if $0\% < MI < 100\%$ all the LMI in the workload can evenly access all banks Allocate BPU colors to each core in HLH Allocate 2*BPU colors to each core in LLH else if $MI = 100\%$ Allocate BPU colors to each core in HLH Allocate 2*BPU colors to each core in LLH
End	

rules to different groups to make full use of bank-level parallelism and spatial locality to optimize the memory system. The rule of DBP is shown in Table 2.

3.4. Adaptive page policy

The close page policy precharges a bank immediately after a column access, and does not care whether the next access is row hit or not. When the next access arrives, the row buffer is closed. So the close page policy is suitable for low memory LMI intensive applications or HMI applications with low spatial locality. For such applications, the close page policy can greatly reduce power consumption and access latency. The page open policy does not precharge the bank until a conflict happens or refreshes. So this policy has an obvious effect on HMI applications with high spatial locality, which greatly reduces the number of row activate and precharge operations, thereby reducing memory power consumption and latency.

On the basis of steps 1 and 2 in the DBP, we dynamically allocate optimal page policy to different groups.

In the first case, if all applications in the group are very memory intensive, we assign open page policy to applications in HLH group whose major power consumption comes from operation power. Because continuous row hits make the number of row activate and precharge operations less, the two commands are the main source of operation power. For applications in LLH group, the close page policy works better. For these applications, the open page policy would significantly increase the number of row conflict, which results in more latency and power consumption.

In the second case, if all applications in the group are not memory intensive whose major power consumption comes from background power, close page policy is the

better choice. For such applications, the memory access streams are rare and the interval of two adjacent accesses is long, so there is no need to keep the row buffer open waiting for the next access. Furthermore, the close page policy increases the opportunity for the rank to enter the power down mode, which consumes the lowest amount of background power.

In the third case, the group includes both high intensive and low intensive applications. We assign open page policy to applications in HMI group and close page policy to other groups.

DBP not only effectively eliminates inter-application interference, but also meets the different needs of bank level parallelism. At the same time, the adaptive page policy dynamically allocates the optimal page policy for each bank according to their memory access characteristics and further reduces memory power consumption.

4. Evaluation Methodology

4.1. Simulation setup

We use both Gem5¹³ and DRAMSim2¹⁴ simulators in our experimental environment to evaluate our proposed schemes. Gem5 models various components of a computer system and can simulate parallel applications. DRAMSim2 can accurately simulate the DRAM memory system. Through several experiments, we set the value of the time interval to 100 K memory cycles, $accesses_t$ to 200 and $hitRate_t$ to 0.5. Table 3 summarizes the simulation parameters. From the reverse analysis of the experimental results, when the time interval is less than 100 K, the information collected during the current time interval may be too local and unrepresentative, increasing the default prediction error rate. When the time interval is greater than 100 K, the change of the application's access behavior cannot be quickly adapted. Once the default prediction is wrong, the division within the current time interval is not appropriate and may even have the opposite effect.

We also implemented the following two policies to evaluate our proposal:

- (1) FRFCFS&OPP: integrating First Ready, First Come, First Serve scheduling with Open Page Policy.
- (2) DBP&APP: integrating DBP with Adaptive Page Policy.

Table 3. Simulated system parameters.

Parameter	Configuration
Processor	4 cores, 3 GHz, x86 out-of-order, two-level cache
L1 caches	Inst Cache: 32 KB, 64 B line, 4-way, LRU Data Cache: 32 KB, 64 B line, 4-way, LRU
L2 caches	Shared Cache: 8 MB, 64 B line, 16-way, LRU
Memory	4 GB, one channel, 2-ranks/channel, 8-banks/rank

Table 4. Hardware overhead.

Function		Size
Access-counter	memory accesses per interval	$N_{\text{core}} * \log_2 \text{access}_{\text{max}}$
HitRate-counter	memory hitRate per interval	$N_{\text{core}} * \log_2 \text{hitRate}_{\text{max}}$
Bank pre-address	bank address of previous memory access	$N_{\text{ch}} * N_{\text{rank}} * N_{\text{bank}} * \log_2 N_{\text{bank}}$
Bank access	bank access per interval	$N_{\text{ch}} * N_{\text{rank}} * N_{\text{bank}} * \log_2 N_{\text{bank}}$

The results are normalized to FRFCFS&OPP. In order to concisely analyze the experimental results, we call the categories in which the proportion of HMI applications are 75%, 50%, 25%, respectively, as mixed group.

4.2. Hardware support

Two counters are set for each bank, and the number of accesses received by the bank and the last accessed bank address in each interval are recorded. we also need two counters to record the accesses and hitRate for each application. The major hardware storage cost incurred to profile application memory access behavior is shown in Table 4.

4.3. Evaluation metrics

Power consumption and system throughput are two key factors in the system. As shown in Eq. (1), DRAM power is composed of Background.Power, ACT_PRE.Power, Burst.Power and Refresh.Power.

$$\begin{aligned} \text{TOTAL_POWER} = & \text{Background_Power} + \text{ACT_PRE_Power} \\ & + \text{Burst_Power} + \text{Refresh_Power}. \end{aligned} \quad (1)$$

As shown in Eq. (2), we evaluate the overall throughput of the system using Weighted Speedup as in Ref. 5. $\text{IPC}_{\text{alone}}$ and $\text{IPC}_{\text{shared}}$ are the IPCs of an application when it is run alone and in a mix, respectively. The number of applications running on the system is given by N .

$$\text{WeightedSpeedup} = \sum_{i=1}^n \frac{\text{IPC}_i^{\text{shared}}}{\text{IPC}_i^{\text{alone}}}. \quad (2)$$

4.4. Workload construction

We use multi-programmed workloads consisting of benchmarks from the SPEC CPU2006¹⁵ in our experiments. We first warm up the system for 100 million instructions, and then simulate each workload until one of the four applications completes 200 million instructions. Each processor core is single-threaded and runs one program. In order to clearly show the experiment result, the workloads in simulation are divided into five categories. The proportion of HMI applications in each

Table 5. Multi-programmed workloads.

Workload	Benchmark	Proportion (%)
1	401.bzip2-410.bwaves-437.leslie3d-458.sjeng	100
2	410.bwaves-436.cactusADM-481.wrf-462.libquantum	100
3	458.sjeng-470.lbm-445.gobmk-473.astar	100
4	401.bzip2-410.bwaves-470.lbm-444.namd	75
5	462.libquantum-445.gobmk-437.leslie3d-450.soplex	75
6	481.wrf-473.astar-445.gobmk-400.perlbench	75
7	450.soplex-459.GemsFDTD-436.cactusADM-437.leslie3d	50
8	401.bzip2-445.gobmk-444.namd-429.mcf	50
9	458.sjeng-470.lbm-444.namd-450.soplex	50
10	470.lbm-481.wrf-464.h264ref-454.calculix	50
11	470.lbm-400.perlbench-456.hmmmer-444.namd	25
12	429.mcf-416.gamess-435.gromacs-470.lbm	25
13	401.bzip2-400.perlbench-454.calculix-416.gamess	25
14	400.perlbench-429.mcf-444.namd-456.hmmmer	0
15	400.perlbench-416.gamess-435.gromacs-444.namd	0
16	435.gromacs-450.soplex-400.perlbench-416.gamess	0

category are 100%, 75%, 50%, 25%, 0%, respectively. The detailed workloads are shown in Table 5.

5. Result Analysis

5.1. DRAM power consumption

As shown in Fig. 2, we first analyze the impact of our mechanism on DRAM power consumption in detail.

For workload 1 to 3, their power consumption is mainly operation power. The DBP&APP greatly reduces DRAM power consumption by 10% on average

Fig. 2. DRAM power consumption.

(up to 16%). There are two reasons for the reduction. First, for applications in HLH group, the DBP isolates memory access streams from different cores and effectively eliminates the interference among cores. The open page policy reduces the number of row activate and precharge operations. Secondly, for applications in LLH group, DBP improves the bank level parallelism which hides the latency by pipelining memory accesses. Furthermore, the close page policy reduces the number of row conflicts.

For workload 4 to 13, we reduce DRAM power consumption by 11% on average (up to 21.2%). Such workloads have both HMI applications and LMI applications. There are three reasons for the reduction. First, the LMI applications have the highest priority and save the banks for others. Second, for applications in HLH group, the DBP isolates memory access streams from different cores and effectively eliminates the interference among cores. The open page policy reduces the number of row activate and precharge operations. Third, for applications in LLH group, DBP improves the bank level parallelism. Furthermore, the close page policy reduces the background power by reducing the number of row conflicts.

For the workload 14 to 16, there is no obvious effect on the DRAM power. We reduce DRAM power consumption by 3% on average. This is because the memory accesses are rare and the interval of two adjacent accesses is relatively long. The close page policy makes the row buffer idle immediately after the column access, which reduces the background power to some extent.

5.2. System throughput

Figure 3 shows the performance impact of our proposal. For the workload 1 to 10, the proportion of HMI applications in each workload is relatively high and the system performance increases by 3% on average (up to 12%). By reducing the number of row activates and precharges, the adaptive page policy reduces the latency of two adjacent memory accesses for applications in HLH group. Our policy provides more banks to enlarge the bank level parallelism for applications in LLH group. For

Fig. 3. System throughput.

workload 11 to 16, the proportion of LMI applications in each workload is relatively high and the system performance has no obvious improvement. For such applications, the memory accesses are rare, so they are not sensitive to bank level parallelism and spatial locality.

6. Related Work

6.1. Cache partitioning

Chen *et al.* do a lot of work on dynamic cache partitioning on real-time MPSoCs.^{16–18} They present a reconfigurable cache architecture which supports dynamic cache partitioning at hardware level and a framework that can exploit cache management for real-time MPSoCs. The proposed reconfigurable cache allows cores to dynamically allocate cache resource with minimal timing overhead while guaranteeing strict cache isolation among the real-time tasks. The cache management framework automatically determines time-triggered schedule and cache configuration for each task to minimize cache misses while guarantee the real-time constraints. However, these works are all aimed at improving the performance of shared cache.

6.2. Memory partitioning

Cache partition is employed to partition shared cache for concurrent running threads, which can eliminate the interference between multithreads and hence reduce conflicts at cache level.^{19–21} Meanwhile, other resources such as memory controller, rank and bank are also shared by multi-core. Thus, memory partitioning has been a popular research in recent years.

Muralidhara *et al.* put forward application-aware memory channel partitioning (MCP), which maps the data of applications that are likely to severely interfere with each other to different memory channels based on applications' characteristics.²² However, in reality the number of threads in the system is usually more than the number of channels, so some threads must be assigned to the same channel, which cannot essentially eliminate the inter-thread interference.

Sub-rank Refs. 5, 23–25 is to make a rank split into multiple small ranks, such as break a 64 bit rank into two 32 bit sub ranks or four 16 bit sub ranks, in order to improve the bank level parallelism. But this is at the cost of bandwidth and data transferring time, there is no obvious performance improvement. Clearly, Sub-rank needs to modify the conventional DRAM structure. In comparison, our proposed schemes can improve bank utilization without modifying the DRAM structure and achieve the goal of reducing power consumption and improving performance.

6.3. The DRAM system power consumption

The system power consumption consists of many parts, reducing power consumption has always been an important task. Our previous work in Ref. 26 proposed a new tree

topology application mapping algorithm. The advantages of Kernighan–Lin algorithm and genetic algorithm are used to reduce overall communication costs and power consumption.

For DRAM power consumption, previous studies of Refs. 27 and 28 mainly focus on switching the rank to the low-power mode to save power. Rank is the minimum power consumption management unit. Only when all the banks in the rank are being idle, we can put the rank into low power mode. Thus, this method by changing the mode to reducing power consumption is applicable to the ranks whose memory access streams are rare.

Haih *et al.* proposed a technique to minimize short and unusable idle periods and create longer ones to reduce the memory power consumption.²⁹ They introduced the concepts of hot and cold ranks. Their method migrates frequently-accessed pages from cold ranks to hot ranks to prolong the idle time of cold rank. However, their strategy needs to modify the hardware design and was not easy to implement. Furthermore, their scheme increased the contention for hot ranks, leading to more power consumption.

7. Conclusion

In this paper, we propose schemes to reduce memory power consumption while maintaining satisfying performance. Our strategy takes into account both the system performance and memory power consumption at the same time. First, by applying different bank partitioning policies, DBP not only effectively eliminates the interference among threads, but also fully takes advantage of bank level parallelism. Second, we combine the DBP with the adaptive page policy to further improve the system performance and power efficiency.

The experiments were done in the simulator. Future work needs to take into account hardware statistics mechanisms on real machines. Hardik Shah *et al.* present a technique to measure the WCET of applications on multi-core architectures using existing tools for single-core architectures.³⁰ They insert a cache observation and a time-stamping module, observe the activity on the shared resource and save it in a trace. Future work can combine this technology to implement hardware statistics mechanisms.

Our experimental results show that our proposed schemes achieve varying results on different application groups. For workloads that memory-intensive applications relatively accounted for 100%, 75%, 50% of number of all applications, power consumption was reduced by 11.2% on average (up to 21.2%) and performance increased by 3% on average (max to 12.5%). For the workloads that memory-intensive applications relatively accounted for 25%, 0% of all applications, memory access streams were very infrequent. As a result, in this case, we did not observe obvious changes in performance. However, the power consumption was reduced by 5.3% average (up to 12.9%).

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant No. 61202076), along with other government sponsors. The authors would like to thank the reviewers for their efforts and for providing helpful suggestions that have led to several important improvements in our work. We would also like to thank all teachers and students in our laboratory for helpful discussions.

References

1. B. Diniz, D. Guedes, J. Meira and R. Bianchini, Limiting the power consumption of main memory, *ACM SIGARCH Computer Architecture News* (ACM, New York, 2007), pp. 290–301.
2. A. N. Udiipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis and N. P. Jouppi, Rethinking DRAM design and organization for energy-constrained multi-cores, *ACM SIGARCH Computer Architecture News* (2010), pp. 175–186.
3. H. Shah, A. Knoll and B. Akesson, Bounding SDRAM interference: Detailed analysis vs. latency-rate analysis, *Design, Automation & Test in Europe Conference & Exhibition* (IEEE, 2013), pp. 308–313.
4. H. Shah, K. Huang and A. Knoll, The priority division arbiter for low WCET and high resource utilization in multi-core architectures, *The Int. Conf.*, October 2014, ACM, New York, pp. 247–256.
5. K. J. Min, D. Sunwoo and D. Sunwoo, Balancing DRAM locality and parallelism in shared memory CMP systems, *2012 IEEE 18th International Symp. High Performance Computer Architecture (HPCA)*, February 2012, New Orleans, LA, USA, pp. 1–12.
6. Liu *et al.*, A software memory partition approach for eliminating bank-level interference in multicore systems, *Proc. of the 21st International Conference on Parallel Architectures and Compilation Techniques*, ACM, USA, 2012, pp. 367–376.
7. H. Zheng and Z. Zhu, Power and performance trade-offs in contemporary dram system designs for multicore processors, *IEEE Trans. Comput.* **59** (2010) 1033–1046.
8. T. Ikeda and K. Kise, Application aware DRAM bank partitioning in CMP, *2013 International Conf. Parallel and Distributed Systems (ICPADS)*, December 2013, IEEE, Seoul, pp. 349–356.
9. Micron Technology, Inc. TN-41-01: Calculating Memory System Power for DDR3, Technical Report, (2007).
10. H. Park *et al.*, Regularities considered harmful: Forcing randomness to memory accesses to reduce row buffer conflicts for multi-core, multi-bank systems, *ACM SIGPLAN Notices*, Vol. 48, ACM, USA, 2013, pp. 181–192.
11. H. Shah, K. Huang and A. Knoll, Timing anomalies in multi-core architectures due to the interference on the shared resources, *Design Automation Conf.* IEEE, Singapore, 2014, pp. 708–713.
12. T. Sherwood, S. Sair and B. Calder, Phase tracking and prediction, *2003 Proc. Int. Symp. Computer Architecture*, June 2003, IEEE, San Diego, USA, pp. 336–347.
13. Binkert *et al.*, The gem5 simulator, *ACM SIGARCH Computer Architecture News* **39** (2011) 1–7.
14. P. Rosenfeld, E. Cooper-Balis and B. Jacob, DRAMSim2: A cycle accurate memory system simulator, in *IEEE Computer Architecture Letters*, Vol. 10, No. 1, Jan.–June, 2011, pp. 16–19.

15. SPEC CPU2006, <http://www.spec.org/spec2006>.
16. G. Chen et al., Shared L2 cache management in multicore real-time system, *2014 IEEE 22nd Annual International Symp. Field-Programmable Custom Computing Machines (FCCM)*, IEEE, Boston, 2014, pp. 170–170.
17. G. Chen et al., Automatic cache partitioning and time-triggered scheduling for real-time MPSoCs, *2014 Int. Conf. ReConFigurable Computing and FPGAs (ReConFig)*, IEEE, Mexico, 2014, pp. 1–8.
18. G. Chen et al., Reconfigurable cache for real-time MPSoCs: Scheduling and implementation, *Microprocess. Microsyst.* **42** (2016) 200–214.
19. G. E. Suh, L. Rudolph and S. Devadas, Dynamic partitioning of shared cache memory, *Journal of Supercomputing* **28**(1) (2004) 7–26.
20. J. Lin et al., Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems, *2008 IEEE 14th Int. Symp on High Performance Computer Architecture*, IEEE, USA, 2008, pp. 367–378.
21. M. K. Qureshi, Y. N. Patt, Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches, *Proc. of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE Computer Society, Orlando, 2006, pp. 423–432.
22. S. P. Muralidhara et al., Reducing memory interference in multicore systems via application-aware memory channel partitioning, *Proc. of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ACM, Brazil, 2011, pp. 374–385.
23. D. H. Yoon et al., The dynamic granularity memory system, *ACM SIGARCH Computer Architecture News*, Vol. 40, IEEE Computer Society, New York, 2012, pp. 548–559.
24. G. Zhang et al., Heterogeneous multi-channel: Fine-grained dram control for both system performance and power efficiency, *Proc. of the 49th Annual Design Automation Conference*, ACM, San Francisco, 2012, pp. 876–881.
25. H. Zheng et al., Mini-rank: Adaptive DRAM architecture for improving memory power efficiency, *2008 41st IEEE/ACM International Symposium on Microarchitecture. 2008. MICRO-41*, IEEE, Italy, 2008, pp. 210–221.
26. J. Fang et al., KL-GA: An application mapping algorithm for mesh-of-tree (MoT) architecture in network-on-chip design, *J. Supercomput.* **71** (2015) 4056–4071.
27. V. Delaluz et al., Scheduler-based DRAM energy management, *Proc. of the 39th annual Design Automation Conference*, ACM, New Orleans, 2002, pp. 697–702.
28. C. S. Bae and T. Jamel, Energy-aware memory management through database buffer management, *Third Workshop on Energy-Efficient Design (WEED)* (2011).
29. H. Huang et al., Improving energy efficiency by making DRAM less randomly accessed, *Proc. 2005 International Symposium on Low Power Electronics and Design*, ACM, San Diego, 2005, pp. 393–398.
30. H. Shah, A. Coombes, A. Raabe, K. Huang and A. Knoll, Measurement based WCET analysis for multi-core architectures, *The International Conf.* (ACM, New York, NY, USA, 2014), pp. 257–266.