SAMS: A Self-Adaptive Mapping Scheme to Assist Page Allocation for DRAM Energy Efficiency

Shao-Chiang Tsai, Keng-Hao Yang, Paul Jendra, and Tien-Fu Chen
National Chiao Tung University, Hsinchu, Taiwan
Email: {sctsai.cs01g, ansallen.cs00g, pjulu.cs02}@nctu.edu.tw, tfchen@cs.nctu.edu.tw

Abstract—As we enter multi-core era, static address mapping schemes used in conventional memory systems may lead to poor utilization of DRAM and cause unnecessary power consumption. Scheduling often helps to reduce dynamic power consumption, yet it is still difficult to allow long periods where DRAM can remain in power down/sleep mode. In this paper, we skew memory traffic by dynamically allocating pages to apposite ranks without harming system performance based on the workload behavior provided, as determined by a monitor. Our experimental results show that our mechanism can reduce power consumption by 6.1% in comparison to the conventional memory mapping scheme.

Index Terms—bank-level parallelism, DRAM, energy efficiency, memory mapping

I. INTRODUCTION

Due to its high density and low cost-per-bit, Dynamic Random Access Memory (DRAM) is widely used. The capacity of DRAM is much larger than cache, and it consumes a significant fraction of the power in modern systems. As the use of mobile systems has increased rapidly, power management policies are now an important issue. Consequently, a good power management policy can improve the overall energy efficiency. There are three major functions that consume power in DRAM - the background, activation and pre-charge, and refresh. The majority of the power consumption is caused by the background since DRAM consistently consumes power, whether servicing requests or not. Dynamic power-activate and pre-charge, which required when data is read from a row buffer or written to an array, is the second largest fraction of DRAM power consumption. Finally, a refresh is required periodically in order to maintain data integrity. Typically, a row must be refreshed every 64ms when operating under 85°C.

In order to minimize the background power consumption, DRAM manufacturers provide various power modes. DRAM chips operate in full power mode while servicing requests, and can only enter the power down mode when all banks within a rank are idle. A rank is therefore the smallest granularity to allow a power.

Self-refresh is a deeper power down mode which aggressively reduces power consumption yet suffers from a higher penalty to return to the full power mode. Several power management policies were proposed to deal with this problem. Among the proposed methods, some exploit a cache replacement policy, while some use prediction or scheduling to prolong a rank’s idle time. However, there is a tradeoff between power reduction and performance degradation. It is difficult to find a long period of time for a certain rank to be idle, so entering a power down mode too frequently may increase the time spent to power-up a rank.

Dynamic power consumption depends on row buffer hit rate. A row buffer conflict occurs when a sequence of requests access the same bank but different rows, causing a lot of row misses and requiring additional activates and pre-charges. An address remapping scheme has been proposed in previous works [1], [2] to mitigate this situation. However, as we enter the era of multi-core devices, the benefit of static address remapping schemes is limited since the address of each request made to the memory is randomly shuffled and is therefore unpredictable. Without any modification to conventional DRAM, the only way to decrease the activation and pre-charge power is to increase the row buffer hit rate. Other methods as proposed in [3], exploit the existence of a local row buffer by adding a latch to each sub-array, not only reducing the significant number of conflicts, but also increasing the row buffer hit rate.

DRAM cells store data on a capacitor, yet the charge leaks over time until eventually the stored data cannot be sensed correctly. To guarantee data integrity, the rows must be refreshed every 64ms. The refresh wastes energy since DRAM refreshes a row irrespective of whether or not the data will be used in the near future. The other downside of refreshing is that whenever a refresh command is issued, all accesses to the data array must be blocked since refresh operation has the highest priority. This can potentially cause performance degradation. Some researchers have exploited the variations between strong and weak cells, so rows can be refreshed at different rates [4], [5]. Consequently, the refresh interval should not always be designed for the worst case scenario.

In a conventional memory mapping scheme, page allocation does not take the rank and bank information
into consideration. The generated physical address is cut into different columns, such as channel: row: bank: rank: column. Each of these columns determines where the request should go. However, this address mapping scheme may not utilize bank-level parallelism and may cause row buffer conflicts that are otherwise avoidable. In comparison, without a rank-aware page allocation, it is difficult to determine an appropriate time for a rank to enter the power down mode. If a rank enters the power down mode, it may not stay in the idle state for long, thus, the cost of the latency cannot offset energy efficiency.

In this paper, we propose an address suggestion method, called the Self-Adaptive Mapping Scheme (SAMS), to increase the rank idle time with negligible performance loss by suggesting an OS suitable for an actual DRAM chip to allocate a new page. SAMS allocates pages based on the workload behavior provided by a monitor. The key idea is to utilize some ranks while others are allowed a higher possibility of entering the power down mode. The unbalanced page allocation may increase row buffer conflicts, so we must increase bank-level parallelism to compensate for performance degradation. To keep a rank in the power down mode longer, we also proposed another method called Self-Adaptive Mapping Scheme with Write Delay policy (SAMS-WD). In comparison to a read, the write operation is not performance-critical since a CPU does not have to wait for the data to be returned. If a write command is sent to a rank which is in the power down mode, we can buffer the data temporarily until the rank returns to the full power mode.

Our contributions are as follows:

We propose SAMS, an address mapping scheme which can assist the OS with deciding page allocations to power down ranks at a higher possibility with negligible performance loss.

SAMS-WD, a scheme that when combined with a write delay, can increase the time that a rank remains in the power down mode.

SAMS-WD can save 7.3% of the background power consumption, 9.1% of activate and pre-charge power, and 6.1% of the total power, when compared to a base case without a power down mode. Only ~1% performance overhead on average is observed.

The rest of the paper is organized as follows. Section 2 provides background information of three major functions that consumes power in DRAM. Section 3 describes a system overview, SAMS, and an enhancement approach SAMS-WD in detail. Our experimental methodology and results follow in Section 4 and Section 5, respectively. We also provide related work in Section 6 and conclude in Section 7.

II. BACKGROUND

DRAM power consumption can be divided into three categories: background, activate and pre-charge, and refresh. A rank must operate in the full power mode in order to service a request. When all banks in a rank return to the idle state and no commands are pending on the rank, the rank can enter the power down mode. DRAM manufacturers provide three power down modes for both DDR2 and DDR3: (i) fast-exit, (ii) slow-exit, and (iii) self-refresh. A self-refresh does not consume any background power, only refresh power, yet the penalty is significantly higher than to return to the full power mode. Both the fast-exit and slow-exit modes consume background power over time, yet have smaller penalties. All of these modes must power up a rank whenever it receives a read or write command.

To access data from DRAM, a row must be opened in order to load data into the row buffer. When accessing a row which is neither active nor idle, the bank needs to pre-charge first and then activate. If the row is already activated, the data could be served immediately. This is called row buffer hit. Successful accesses to the same row increase the row hit rate, not only reducing additional activation and pre-charge power consumption, but also improving the performance. However, successive accesses to the same bank but different rows causes a row miss, called a row buffer conflict. A row miss deteriorates performance and significantly increases the activation and pre-charge power consumption. To mitigate this effect, First Ready-First Come First Search (FR-FCFS) is applied by giving the row hit command the highest priority to issue even when other commands come first. Although scheduling is one way to solve this issue, most of the activation and pre-charge commands are difficult to reduce.

Due to the nature of DRAM cells, they lose charge from the capacitor over time. To maintain data integrity, rows must be restored or refreshed in regular intervals. We call this interval tREFI, typically 64 ms for commodity DRAM. The rank is the smallest granularity to operate a refresh operation; whenever a refresh command arrives in the command queue, the rank must return to the full power mode and all requests must be blocked until finishing the refresh operation. As the capacity of the DRAM increases, refresh operations will cause more severe performance degradation and power consumption.

III. SELF-ADAPTIVE MAPPING SCHEME

A. System Overview

In conventional page allocation schemes, pages are allocated/mapped by the OS without any hardware information (i.e., the state of target ranks and banks on a DRAM chip are ignored). As a result, uneven distribution of pages causes additional conflicts, activate/pre-charge commands, and access latency. The overall performance is therefore decreased due to increased power consumption, resulting in poor energy efficiency. This work proposes a method called Self-Adaptive Mapping Scheme (SAMS), to eliminate this problem. The primary idea is to change the conventional page allocation process by suggesting a rank and a few banks (at least one) to the OS using the memory controller. In other words, the memory controller will determine a rank and a bank (at least one) to assist the OS when it must decide on page allocation.
As illustrated in Fig. 1, the left part shows the path of the traditional page allocation scheme, which picks up a physical address from a free pages management (e.g. free_area) for a new page and uses the same address to access the DRAM without considering the usage or current state of the DRAM chip. This work (SAMS), shown in the right part, will suggest a rank and a few banks (at least one) for candidates dependent on the hints from the monitor after receiving a hint request from the OS. The OS will then allocate the new page into the rank and banks (selecting one) and send the address to the memory controller as a normal access. Hence, the memory controller can manage page placement while balancing the loading of banks, decreasing row buffer conflicts, saving more power, and achieving superior energy efficiency.

### B. Monitor

In order to determine the candidate rank and bank, this work proposes a monitor-based page allocation scheme to help operating system to decide where the page should be placed. The monitor is composed of some registers to record each core access behavior and the utilization of each bank. We categorize these registers in two parts: (i) rank-core register and (ii) bank utilization register. All registers are set to 0 initially.

The rank-core register is used for recording core access behavior in each epoch. Some cores may access intensively during the epoch and some may not. We can exploit the variation between core access behaviors to increase idle time by clustering pages in a certain rank. For example, assume there are 4 cores running a benchmark, core 1 request intensively during this epoch and some may not. We can exploit the variation between core access behaviors to increase idle time by clustering pages in a certain rank.

Whenever a rank(R) receives a request from a core(C), our monitor will update the information by giving a credit to rank(R)-core(C) while other rank(R’) -core(C) will minus 1 as long as the register has not reached to 0. Rank-core registers will be reset to 0 every 100 thousand cycles, this may help operating system has more flexibility to allocate pages on different ranks.

The bank utilization register is used for compensating the performance degradation caused by centralizing pages on a certain rank. To fully utilize bank-level parallelism, we need to know which banks are under high utilization and which are not. When a bank receives a request, the monitor will increase 1 to that bank while decrease 1 for the rest of the banks. Banks with a different number inform how busy they are, so we can choose a bank with the lowest utilization. Unlike rank-core registers, updating information of banks is done locally since the accesses from different ranks are totally independent. For example, in the case of a four-bit register, the flow of updating information on the monitor is shown in Fig. 2. Assume a transaction, T0, is a request from core 2 to rank 0 and bank 3, we give a credit to rank(0)-core(2) while minus 1 on the rest of the rank(1)-core(2) register. Next, the bank utilization registers need to be updated as well. In this case, the bank utilization register of rank(0)-bank(3) will be increased from 10 to 11 while the rest will be decreased by 1. The update of T1 and T2 will be the same as we mentioned above. The flow of updating information will not change even receiving a request to a rank which is currently in power down mode.

![Figure 1. Overview of the self-adaptive mapping scheme](image)

![Figure 2. Architecture of the monitor](image)
buffer conflict may arise. This in turn will cause the probability of bank conflicts to increase. When applying our mechanism the utilization of each bank can be made to almost equal.

In order to serve these commands, the memory controller must power up the rank and therefore reduce ranks idle time. This phenomenon causes more power consumption and increases the average memory access latency.

To further prolong the rank idle time without harming the system performance, we can delay write commands since they are not performance-critical. We propose a mechanism, SAMS-WD, to reduce background power aggressively. The idea is to delay some write commands, if the target rank is in the low power mode. However, because of latency overhead, hardware overhead, and energy efficiency, we still cannot delay as many write commands as we want. Hence, we propose two configurations, SAMS-WD2 and SAMS-WD4, to analyze the benefits and impacts on DRAM.

The longer a rank can stay in idle time, the more likely pages will over-centralize in one certain rank. This is because our Candidate algorithm will give a rank, which is in full power mode, a higher priority. For example, if a rank is in the power down mode, SAMS-WD2 will not power up the rank unless it receives a read command or over two write commands are pending in the command queue. While SAMS-WD4 delays up to four write commands and further reduces more power consumption, the latency will also increase if too many commands are pending in the command queue.

SAMS-WD is a combination of SAMS and WD; the simple write buffering scheme is slightly different in our mechanism. Previous work has buffered write data until the buffer is full or until a read transition is received, but this may decrease the efficiency. If a rank is in the power down mode and has a pending request, other ranks are more likely to be in a high utilization state. Even if we only place write requests in a pending state, the performance will be degraded severely if five or more commands are pending in the command queue. Apart from this, row buffer conflicts may increase as well due to the overly intensive requests to a rank.

### IV. METHODOLOGY

#### TABLE I. EVALUATION SYSTEM CONFIGURATIONS

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>DRAM controller</th>
<th>Row page policy</th>
<th>Scheduling policy</th>
<th>Depth of command queue</th>
<th>DRAM (DDR3)</th>
<th>Frequency</th>
<th>Row buffer size</th>
<th>Rank</th>
<th>Scheduling policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>2.0 GHz</td>
<td></td>
<td>open page</td>
<td>FR-FCFS</td>
<td>32</td>
<td>4 GB</td>
<td>200 MHz</td>
<td>2 KB</td>
<td>8</td>
<td>FR-FCFS</td>
</tr>
</tbody>
</table>

In this work, we use Multi2Sim [6], DRAMSim2 [7], and SPEC CPU2006 [8] to evaluate our mechanism. We run each simulation at 2GHz clock frequency with a private L1 cache (64 KB), a shared L2 cache (2MB), and a 2GB DDR3 main memory. The configurations of
DRAM are shown in Table I, such as frequency, scheduling policy, depth of command queue, number of ranks (banks).

In order to analyze our methods, we create eight workloads by mixing four benchmarks of SPEC CPU2006 into one workload instead of the entire suite. The combination of each mixed workload is according to [9], as shown in Table II. We simulate each benchmark until it reaches 800 million instructions.

### TABLE II. WORKLOAD MIXED BY SPEC CPU2006

<table>
<thead>
<tr>
<th>Name</th>
<th>Benchmarks in Mix</th>
</tr>
</thead>
<tbody>
<tr>
<td>mix0</td>
<td>xalancbmk</td>
</tr>
<tr>
<td>mix1</td>
<td>bzip2</td>
</tr>
<tr>
<td>mix2</td>
<td>soplex</td>
</tr>
<tr>
<td>mix3</td>
<td>zeusmp</td>
</tr>
<tr>
<td>mix4</td>
<td>astar</td>
</tr>
<tr>
<td>mix5</td>
<td>mcf</td>
</tr>
<tr>
<td>mix6</td>
<td>lbm</td>
</tr>
<tr>
<td>mix7</td>
<td>games</td>
</tr>
</tbody>
</table>

### V. EXPERIMENT

In this work, the conflict can be omitted about 68% compared to the conventional mapping scheme on average, as shown in Fig. 4. Since this mapping approach scattered the pages on different banks, it will enhance bank level parallelism. It does not only improve the bank conflict which is not solved exactly by the static mapping mechanism, but also avoid over centralized access to certain banks. In the other words, the loading of each bank is more balanced than the conventional design.

The power improvement opportunity for SAMS is majorly consists of two parts, background power and activate/pre-charge power. The impact of activate/pre-charge power relies on bank-level parallelism.

![Figure 4. Improvement of row buffer conflict](image)

Because the accesses are dispersed to different banks, i.e. higher bank parallelism, it can slightly improve the row buffer hit rate. As the hit rate increased, it implies that activate and pre-charge command could be saved. The equal utilization of each bank can reduce background power as well. On the other hand, the impact of background power depends on how much cycles the ranks are in power down mode. However, once a rank is in power down mode, it is unable to request an access from the memory queue. In this case, the scheduler must delay the access and power-up the rank; it will cause higher average memory access time. Therefore, we analyze this work in latency, power, energy and Energy Delay Product (EDP) by comparing the standard page allocation, Permutation [10], and three configurations: SAMS (without any write delay), SAMS-WD2 (needs an extra 1 bit to be a control flag of each rank), and SAMS-WD4 (needs extra 2 bits of each rank).

As illustrated in Fig. 5, the total power consumption of SAMS, SAMS-WD2, and SAMS-WD4 can be decreased by 2%, 5%, and 6% on average, respectively. Permutation [10] is good at a single program system, but it does not gain significant improvement (both power consumption and access latency) in a multi-program system.

![Figure 5. Power consumption (normalized)](image)

![Figure 6. Average access latency](image)

As described above, higher bank-level parallelism can improve access latency due to promoted row buffer hit and reduced row buffer conflict. Hence, the average access latency is also improved by SAMS not only because of its higher bank level parallelism, but also it accomplishes a better loading balance. Since SAMS-WD delays some write commands (2 or 4) to increase the time that a rank remains in the power down mode, it would increase latency overhead by 0.8% (delay 2 commands) and 1.6% (delay 4 commands) on average, as shown in Fig. 6. There are two workloads, mix1 and mix7, that present worse access latency, because the behavior of benchmark, hmmer, which consists of very centralized accesses (both reads and writes). Since SAMS-WD delays some write commands to make some ranks stay in power down mode; once the centralized requests come in the command queue, all of the requests which target to a power-down rank, will have to wait until the memory controller wake up the target rank. Hence, the average access latency will be increased. As a result, energy improvements of SAMS, SAMS-WD2, and SAMS-WD4 are 4%, 4.6%, and 4.8% (shown in Fig. 7).

In comparison to energy efficiency, we use Energy Delay Product (EDP) to be the metric. As shown in Fig. 8, on average, even though SAMS-WD4 save more power than SAMS-WD2, SAMS-WD4 gain less energy...
efficiency (improves 3.2%) than SAMS-WD2 (improves 4%) due to its higher latency overhead.

Figure 7. Energy consumption

![Energy consumption graph]

Figure 8. Energy efficiency (EDP)

![Energy efficiency graph]

VI. RELATED WORK

Utilizing the low power mode of DRAM is a common way to reduce background power. A. Lebeck et al. [11] proposed a power-aware page allocation method by operating system and Z. Wang et al. [12] used compiler-based mechanisms to reduce power. A. M. Amin and Z. A. Chishti [13] proposed a rank-aware cache replacement policy and a write buffering scheme to increase some pre-chosen ranks idle time. They replaced LRU, which is the commonly used cache replacement policy, to pick a victim which can prolong prioritized ranks in the low power mode. However, the changed of replacement policy could also damage the system performance. Our technique differs from RAWB [13]; in order to cooperate with SAMS, SAMS-WD would not buffer write commands until the low power rank receives a read request. We aggressively power up the rank in order to optimize the trade-off between energy efficiency and performance degradation. H. Zheng et al. [14] broke the original rank into smaller granularity, so fewer chips would be involved in each access.

Row buffer conflicts cause additional activate and pre-charge operations. Z. Zhang et al. [10] analyzed the sources of row buffer conflicts, mainly from L2 conflict misses, L2 write-backs, and specific memory access pattern. They proposed a method called Permutation-based Page Interleaving Scheme to reduce row buffer conflicts by XOR bank index and the lowest few bits of last level cache tag index. Some works [15], [16] presented an interleaving scheme that swaps partial bits of L2 tag and partial bits of page offset, to improve the performance of some certain programs by finding the optimal number of bits to swap. J. Liu et al. [4] studied the trade-off between locality and memory level parallelism. They proposed a global space address mapping, which is, mapping a physical address space across multiple channels while a local address mapping defines how memory region mapped across the bank within the MC. The methodology is to determine where the memory channel index should be placed in the address, J. Shao and B. T. Davis [1] proposed a method that reverses the upper bits to increase row hit rate. All methods mentioned above only used a static mapping scheme; there is not much performance improvement under the multiprogramming workloads.

One naïve way to increase memory level parallelism is to add more channels or banks, but of course, expensive as well. Y. Kim et al. [3] proposed a novel method by exploiting the presence of a local row buffer in each bank sub-array to increase bank level parallelism. They add additional latches and designated bits to each local row buffer which not only can parallel each request but also can increase the locality. This could reduce activate and pre-charge command. Our mapping scheme is orthogonal to [3] and the improvement of the performance may gain more when our method is combined with [3] to distribute pages into different sub-array.

R. Panwar and P. A. Thomas [17] proposed a method by profiling some of the most frequently used instruction on cache to build a perfect hash table, but the capacity of DRAM is much bigger than cache so the method may not be practical when applying perfect hash to DRAM. Some works [1], [18] improved system performance by scheduling. C. J. Lee et al. [19] proposed BLP-Aware Prefetch Issue (BAPI) and BLP-Preserving Multi-Core Request Issue (BPMRI) to increase bank level parallelism in the presence of prefetching. Several papers [4], [5], [20], [21] have targeted on reducing refresh operation to save refresh power consumption with different arguments. These refresh methods can be classified into two groups, retention sensitive methods and state sensitive methods.

VI. RELATED WORK

Utilizing the low power mode of DRAM is a common way to reduce background power. A. Lebeck et al. [11] proposed a power-aware page allocation method by operating system and Z. Wang et al. [12] used compiler-based mechanisms to reduce power. A. M. Amin and Z. A. Chishti [13] proposed a rank-aware cache replacement policy and a write buffering scheme to increase some pre-chosen ranks idle time. They replaced LRU, which is the commonly used cache replacement policy, to pick a victim which can prolong prioritized ranks in the low power mode. However, the changed of replacement policy could also damage the system performance. Our technique differs from RAWB [13]; in order to cooperate with SAMS, SAMS-WD would not buffer write commands until the low power rank receives a read request. We aggressively power up the rank in order to optimize the trade-off between energy efficiency and performance degradation. H. Zheng et al. [14] broke the original rank into smaller granularity, so fewer chips would be involved in each access.

Row buffer conflicts cause additional activate and pre-charge operations. Z. Zhang et al. [10] analyzed the sources of row buffer conflicts, mainly from L2 conflict misses, L2 write-backs, and specific memory access pattern. They proposed a method called Permutation-based Page Interleaving Scheme to reduce row buffer conflicts by XOR bank index and the lowest few bits of last level cache tag index. Some works [15], [16] presented an interleaving scheme that swaps partial bits of L2 tag and partial bits of page offset, to improve the performance of some certain programs by finding the optimal number of bits to swap. J. Liu et al. [4] studied the trade-off between locality and memory level parallelism. They proposed a global space address mapping, which is, mapping a physical address space across multiple channels while a local address mapping defines how memory region mapped across the bank within the MC. The methodology is to determine where the memory channel index should be placed in the address, J. Shao and B. T. Davis [1] proposed a method that reverses the upper bits to increase row hit rate. All methods mentioned above only used a static mapping scheme; there is not much performance improvement under the multiprogramming workloads.

One naïve way to increase memory level parallelism is to add more channels or banks, but of course, expensive as well. Y. Kim et al. [3] proposed a novel method by exploiting the presence of a local row buffer in each bank sub-array to increase bank level parallelism. They add additional latches and designated bits to each local row buffer which not only can parallel each request but also can increase the locality. This could reduce activate and pre-charge command. Our mapping scheme is orthogonal to [3] and the improvement of the performance may gain more when our method is combined with [3] to distribute pages into different sub-array.

R. Panwar and P. A. Thomas [17] proposed a method by profiling some of the most frequently used instruction on cache to build a perfect hash table, but the capacity of DRAM is much bigger than cache so the method may not be practical when applying perfect hash to DRAM. Some works [1], [18] improved system performance by scheduling. C. J. Lee et al. [19] proposed BLP-Aware Prefetch Issue (BAPI) and BLP-Preserving Multi-Core Request Issue (BPMRI) to increase bank level parallelism in the presence of prefetching. Several papers [4], [5], [20], [21] have targeted on reducing refresh operation to save refresh power consumption with different arguments. These refresh methods can be classified into two groups, retention sensitive methods and state sensitive methods.

Retention sensitive methods exploit variation between strong cells and weak cells so the row could be refreshed at different rates [4], [5]. State sensitive methods [20], [21] refer the state of rows to operate refreshes. The states represent some hint for the memory controller, such as accessing histories [20] or data properties [21].
power, and improve 4% energy efficiency with less than 1% of latency overhead on average.

REFERENCES


Shao-Chiang Tsai received the B.S. degree in computer science from National Chiao Tung University in 2012 and M.S. degree in computer science and engineering from NCTU, Hsinchu, Taiwan, in 2014. His current research interests are computer architectures, memory system, emerging memory technology, and embedded systems.

Keng-Hao Yang received the B.S. degree in computer science from National Chung Cheng University, Chia-yi, Taiwan, in 2011, where he is currently pursuing the Ph.D. degree in computer science at National Chiao Tung University, Taiwan. His current research interests are computer architectures, memory system, non-volatile memories, and embedded systems.

Paul Jendra received the B.S. degree in National Chiao Tung University, Taiwan, in 2011. He joined LITEON-IT Corp., Taiwan, as a design and quality engineer for two years. He is currently working toward the M.S. degree in the Institute of Computer Science and Engineering, National Chiao-Tung University, Hsinchu, Taiwan. His research interest is in memory hierarchy and design.

Tien-Fu Chen received the B.S. degree in Computer Science from National Taiwan University in 1983. He joined Wang Computer Ltd., Taiwan as a system software engineer for three years. From 1988 to 1993 he attended the University of Washington, receiving the M.S. degree and Ph.D. degrees in Computer Science and Engineering in 1991 and 1993 respectively. He is currently a Professor in the Department of Computer Science at National Chiao Tung University, Hsinchu, Taiwan. In the past, he had published several widely-cited papers on dynamic hardware prefetching algorithms and designs. In recent years, he has made contributions to processor design and SOC design methodology. His recent research interests include multithreading/multicore media processors, on-chip networks, and low-power architecture techniques as well as related software support tools and SoC design environment. His current research interests are computer architectures, system-on-chip design, and embedded systems.