A Quantitative Analysis of Performance Benefits of 3D Die Stacking on Mobile and Embedded SoC

Dongki Kim, Sungjoo Yoo, Sunggu Lee, *Jung Ho Ahn, **Hyunuk Jung
Department of Electronic and Electrical Engineering, POSTECH
*Graduate School of Convergence Science and Technology, Seoul National University
**System LSI Division, Samsung Electronics

ABSTRACT

3D stacked DRAM improves peak memory performance. However, its effective performance is often limited by the constraints of row-to-row activation delay (tRRD), four active bank window (tFAW), etc. In this paper, we present a quantitative analysis of the performance impact of such constraints. In order to resolve the problem, we propose balancing the budget of DRAM row activation across DRAM channels. In the proposed method, an inter-memory controller coordinator receives the current demand of row activation from memory controllers and re-distributes the budget to the memory controllers in order to improve DRAM performance. Experimental results show that sharing the budget of row activation between memory channels can give average 4.72% improvement in the utilization of 3D stacked DRAM.

1. Introduction

3D stacked DRAM is expected to become a practical solution to resolve the memory wall problem [1][2][3][4]. It provides the vertically connected SoC with high memory bandwidth via many inter-die wires (via face-to-face and/or TSVs). Thus, multiple memory channels[1] are available from a single DRAM die or a set of DRAM dies. Figure 1 illustrates multiple channels enabled by 3D stacked memory. Figure 1 (a) shows the case where a single DRAM die provides four channels while Figure 1 (b) shows the case where multiple DRAM dies are stacked and they share the four channels.

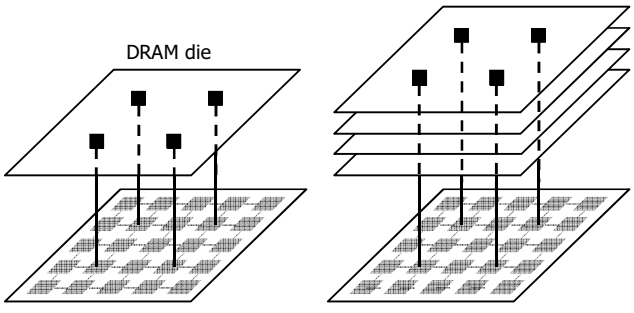

Figure 1 3D stacked DRAM

[1] A memory channel corresponds to the memory interface of conventional DRAM. It consists of address, data and control signals and works as an independent path to access DRAM.

978-3-9810801-7-9/DATE11/©2011 EDAA

3D stacking of DRAM increases peak memory bandwidth. However, increasing effective memory bandwidth requires resolving practical issues. In this paper, we identify a significant problem in increasing effective memory bandwidth which is caused by DRAM timing constraints.

Figure 2 illustrates two timing constraints in DRAM: tRRD and tFAW. tRRD is the minimum latency (e.g., 7.5ns) between two RAS (row activation) commands while tFAW (four active bank window) limits the number of RAS commands within the time interval of tFAW to four. Both are mostly constrained by IR drops [5] and address decoding latency [6]. Especially, RAS commands for row activation is power hungry [7][8][9]. Figure 2 illustrates a current profile in DRAM [10]. Four consecutive RAS commands (spaced with tRRD) are issued within tFAW. The figure shows a current profile where RAS commands create current spikes. Such current drawing incurs supply voltage drop affecting the latency of neighbor blocks which share the same supply voltage. Such voltage drop can cause timing-related errors in timing critical circuits in DRAM, e.g., data sensing errors in sense amplifiers [11].

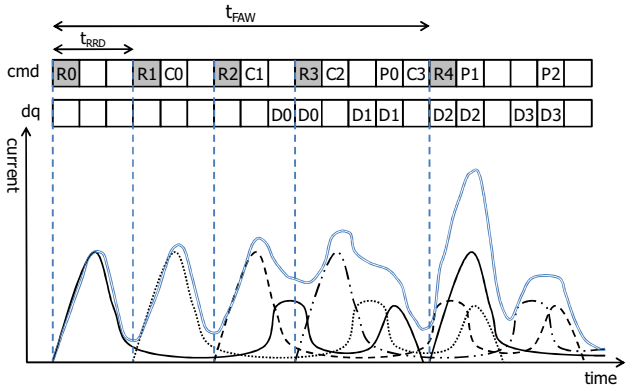

Figure 2 Current profile in DRAM accesses

Note that the constraints of tRRD and tFAW are imposed on the entire DRAM die since the power network is often designed as a single entity. Typically, there are two core and peripheral power networks on a DRAM die [6][11]. Currently, a 3D stacked DRAM die is designed as an extension of existing DRAM die by adding new circuitry to increase the number of memory channels, e.g., I/O circuits for 4 channels, and by regrouping internal memory tiles to give more number of banks, e.g., 4 banks per channel [3]. Thus, if such modifications are made without significantly improving the capability of power network, the existing peak power constraints

still hold for the multiple channels of 3D stacked DRAM.

In the industrial design [3], as an example, the number of memory channels in one DRAM die is increased from 1 to 4 for 3D stacking. However, the same peak power constraints as those of the traditional single channel DRAM, e.g., tRRD=7.5ns, tFAW=37.5ns, are still applied to the four channels. Thus, a single DRAM die cannot fully utilize the increased number of memory channels, i.e., increased memory bandwidth. The problem becomes significant especially when multiple DRAM rows in different banks need to be activated concurrently, which is a typical method to increase DRAM performance via bank parallelism [12][13]. In [3], in order to resolve this problem, multiple DRAM dies are utilized and each memory channel is shared by the multiple DRAM dies as shown in Figure 1 (b). In this configuration, multiple dies on the same channel are accessed in an interleaved manner thereby overcoming the constraints imposed on each of DRAM dies separately.

The usage of multiple 3D stacked DRAM dies only to resolve the above problem as in [3] is expensive, especially in mobile and embedded devices, e.g., smart phone, due to the increased DRAM cost (due to multiple dies) and reduced yield in 3D stacking of multiple dies. Another possibility to resolve the problem is to modify the DRAM architecture, e.g., smaller row sizes [9], which is also costly since DRAM is a commodity and sensitive to cost. In our work, we propose a method to improve the performance of 3D stacked DRAM without redesigning DRAM nor utilizing multiple DRAM dies.

The proposed solution improves performance via load balancing of row activation demands among multiple DRAM channels. It exploits the slack in required row activations between the multiple channels of the same DRAM die and re-distributes the budget of row activations across the channels thereby enabling more row activation on the channels requiring more bank parallelism. To do that, memory controllers cooperate with each other to share their budget of row activations with each other, which enables us to fully utilize the entire row activation capability of a single DRAM die thereby increasing performance.

This paper is organized as follows. Section 2 reviews related work. Section 3 gives the preliminary. Section 4 explains our idea. Section 5 presents our method of inter-memory controller cooperation. Section 6 reports experimental results. Section 7 concludes the paper.

2. Related Work

3D stacking of DRAM with SoC dies gives two major benefits: small form factor and higher memory bandwidth. Face-to-face stacking is already commercialized [14] while TSV stacking is being actively studied [2][3][4]. In order to improve the performance of 3D stacked DRAM, conventional methods of memory access scheduling can be applied to each of multiple channels [12][15][16][17][18][19].

Recently, several studies have been presented for memory access scheduling for multiple DRAM channels (some for conventional usage of multiple DRAMs and others for 3D stacked DRAM). In [20], multiple DRAM channels are utilized to reduce DRAM access latency and memory access scheduling is performed to serve critical word first. In [21], Abts et al. show that the locations of memory controllers affect memory performance, especially in the case of mesh-based many-core architecture. In [22], Kim et al. present a method of fair memory access scheduling by adjusting the per-thread priorities of memory accesses among memory controllers.

In [23], Kwon et al. address a performance problem which is caused by uncoordinated data transfers from memory controllers (for 3D stacked DRAM) to the same destination. They present a concept of inter-memory controller arbitration where, in case of possible conflict at the destination, memory controllers select a winner to send data to the destination thereby avoiding such conflicts. In [24], the authors present a concept called transaction ID renaming (which is based on reorder buffers at network entry) in order to resolve a performance problem caused by the in-order requirements imposed on multiple traffic streams from one master to multiple memory channels. In [25], the authors improve the solution in [24] by removing the reorder buffer at network entry and, instead, exploiting buffer resources in network-on-chip for the reorder buffer purpose. In [26], Kim, et al. address a problem called network congestion-induced memory blocking in the context of 3D stacked memory and present a method of network congestion-aware memory access scheduling.

3. Preliminary: DRAM Operation

Figure 3 illustrates a simplified DRAM architecture consisting of four banks. A bank consists of rows (often called pages) whose sizes are typically 1KB or 2KB. When an access request (read or write) arrives at the memory controller, the address of request is utilized to select the corresponding bank and row as the two arrows show in Figure 3. The memory controller sends a RAS (row activation, in short ACT) command and the entire contents of the row are copied from the data array to the row buffer (one per bank) which takes a latency of tRCD (e.g., 12.5ns). Then, the memory controller issues a CAS command to access (read or write) the corresponding data (shaded rectangle in the figure) in the row buffer, which takes a latency of CL (CAS latency), e.g., 12.5ns. After accesses to the row buffer are finished, the bitlines of the accessed bank need to be precharged for subsequent row accesses. Thus, the memory controller sends a precharge (PRE) command which takes a latency of tRP. The PRE command can be overlapped with read data transfer from the corresponding row buffer while it needs to be issued only after write data transfer. Note that the memory controller issues RAS, i.e., ACT commands only when the constraints of tRRD and tFAW are satisfied.

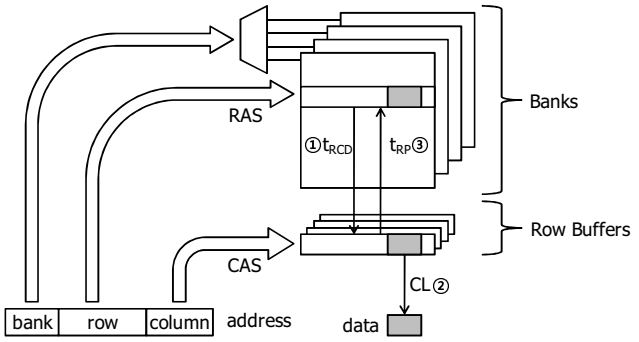

Figure 3 DRAM architecture and operations

4. Basic Idea

Figure 4 illustrates a scenario where two memory channels are accessed concurrently. Each memory controller (one for each channel) has four requests. Each request is denoted with 'R#B#' where R and B represent a row and a bank, respectively. In this case, in order to satisfy the constraints of tRRD and tFAW, each memory channel can issue ACT commands at even or odd period of tRRD while both satisfying tFAW. For instance, memory controller MC0 can issue an ACT command at every even period of tRRD. Thus, it can issue ACT commands twice per tFAW period.

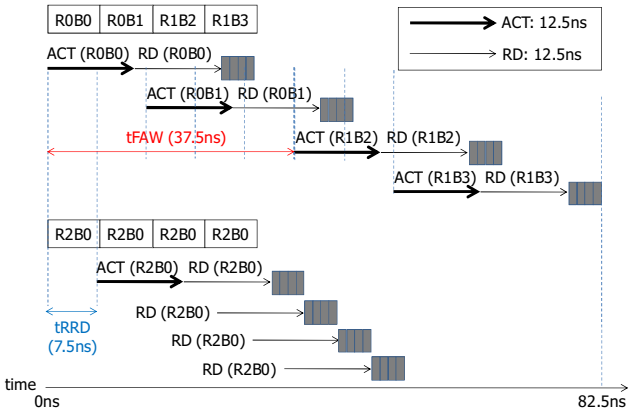

Figure 4 Baseline case

The scenario in Figure 4 does not exploit the full potential in row activation since only three ACT commands are issued during tFAW. To be specific, at the 2nd odd period of tRRD, memory controller MC1 does not have ACT command to issue while MC0 has ready ACT commands. In the scenario, due to the uniform budgeting of row activation and the load imbalance of row activation between the two channels, only three ACT commands can be issued thereby increasing the latency of all the eight requests.

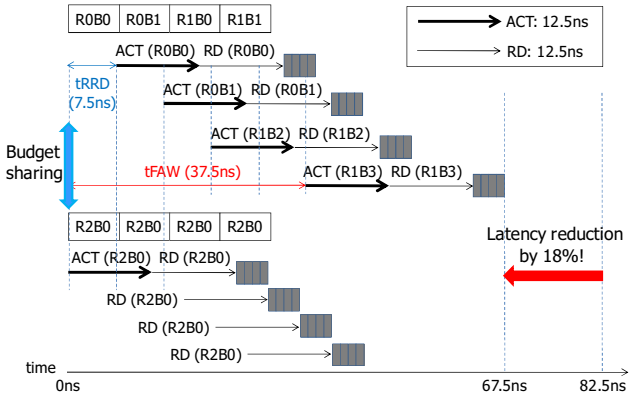

Figure 5 Inter-memory controller cooperation

The limitation of the scenario in Figure 4 is due to the fact there is no method to consider the load imbalance of row activation. In order to take into account the load imbalance, we need to collect the information on the status of memory channels and adjust the budget of row activations on each memory controller according to the collected information. Figure 5 illustrates our idea. In this scenario, the status of memory channel is collected at time 0 and the budget of row activation on each channel is adjusted according to the information. Assume that MC0 now has three ACT commands per tFAW as its budget while MC1 has one. MC0 can serve more ACTs during tRRD than the case in Figure 4 thereby reducing the total latency of eight requests by 18% in this case. The performance of MC1 is not affected by the reduced budget since it requires only one ACT command during tFAW. As shown in Figure 5, the information of load imbalance in row activation can be exploited to achieve further performance improvement.

5. Inter-Memory Controller Cooperation

5.1 Proposed Architecture

Figure 6 illustrates the proposed architecture. It consists of two memory controllers and one coordinator (located at the center in the figure). The memory controller consists of two parts: monitor and memory scheduler. Periodically, the monitor of memory controller sends the number of required ACT commands to the coordinator. Based on the information of required row activation from each memory controller, the coordinator allocates the budget of row activation to each memory controller (details in Section 5.2) and sends the information of budget allocation to each memory controller.

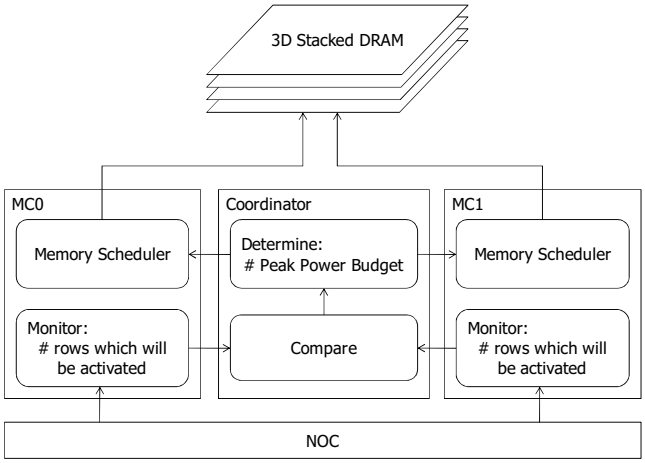

Figure 6 Inter-memory controller coordinator

If a MC receives its budget of row activation, then it needs to follow it until the budget is updated in the future. Note that the MC can determine its own CAS command schedule as far as the given budget of row activation is satisfied. Thus, under the given budget, it applies its own scheduling policy (e.g., FR-FCFS [15], STFM [18], PARBS [19], etc.) for performance improvement.

5.2 Budget Allocation Method

In this subsection, we assume four-channel 3D stacked DRAM. In this case, there are five possible cases of allocating the budget of row activation as shown in Table 1.

Table 1 Budget allocation cases

Case	A	B	C	D	E
Budget	4,0,0,0	3,1,0,0	2,2,0,0	2,1,1,0	1,1,1,1

Given the information of required row activation load (# of ACT commands to be issued by each memory controller), the coordinator chooses the most suitable budget allocation among the above five cases. In a trivial case, if only one channel has ACT commands to be issued while the others are idle (case A), then all the available budget, i.e., 4 ACT commands per tFAW, is allocated to the channel. When each of all the four channels has at least one ACT command to be issued (case E), then all the channels have the same uniform budget, i.e., one ACT command per channel. Case B is applied when one channel has one ACT command to be issued and another has more than one ACT commands to be issued while the other two channels are idle. Case C occurs when two channels each

have (more than) one ACT commands to be issued while the other two are idle. In case D, two channels have each one ACT command to be issued, one channel has more than one ACT commands to be issued while the other channel is idle.

The coordinator adjusts the budget allocation whenever any of memory channels changes its required row activation level. In such a case, the newly available portion of row activation budget, if any, can be re-distributed to memory channels requiring more row activations.

6. Experiments

6.1 Experimental Methodology

Figure 7 shows a 7x7 mesh architecture used in our experiments. Each tile has a Tensilica LX2 core or a memory controller. The coordinator (denoted with 'C' in the figure) is located at the center tile together with a LX2 core. Memory controllers and the coordinator are connected via dedicated connections. We utilize 4-channel 3D stacked DRAM. The bandwidth of each channel is 3.2GBps corresponding to 32b DDR2-800. Table 2 gives the summary of architectural details.

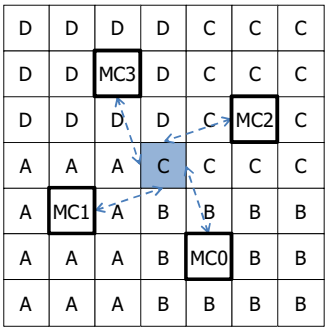

Figure 7 A 7x7 mesh architecture

Table 2 Architectural parameters

Component	Details
CPU core	Tensilica LX2 (7 stage pipeline), 32b address, 64b data, 4-way 16KB I/D, 400MHz
NoC	Input buffered 5-port router, 4 pipeline stages with lookahead XY routing, 4 VCs per port, 8 64b flits per VC, 400MHz
Memory controller	FR-FCFS policy [15], express path (1+CL latency in case of no previous request), open row scheme
Memory channel	32b DDR2 800, CL/t_{RP}/t_{RCD}=15ns/15ns/15ns, tRRD=10ns, tFAW=45ns

Each core accesses the nearest DRAM channel. For instance, all cores denoted with 'A' access memory controller MC1 in Figure 7. We utilize two types of memory traffic: synthetic traffics and benchmark programs. Table 3 shows four cases of synthetic traffics. They are different in terms of row buffer hit rate. In cases A, B and C, one or two memory channels receive traffics with high row buffer hit rate while the other memory channels receive low row buffer hit traffics. In case D, all the traffics have low row buffer hit rate thereby requiring high levels of row activation load at each channel.

Table 3 Row buffer hit rate in synthetic cases

	Case A	Case B	Case C	Case D
MC0	High	High	Low	Low
MC1	High	High	High	Low
MC2	Low	Low	Low	Low
MC3	High	Low	Low	Low

Table 4 shows five cases of benchmark programs. The five cases are different from each other in terms of memory traffic load and row buffer hit rate. We utilize benchmark programs from SPEC2000, 2006, and MiBench. We obtain the trace of memory accesses from the core by running the commercial LX2 core simulator and capturing L1 miss request traces. We apply the traces to our simulation model of 7x7 mesh architecture.

Table 4 Benchmark program cases

	Case 1			Case 2		
	Prog.	Load	Hit rate	Prog.	Load	Hit rate
MC0	ammp	High	High	ammp	High	High
MC1	mcf	High	Low	mcf	High	Low
MC2	equake	Low	High	fft	Low	Low
MC3	susan	Low	High	bzip2	Low	Low
	Case 3			Case 4		
	Prog.	Load	Hit rate	Prog.	Load	Hit rate
MC0	qsort	Med	High	ammp	High	High
MC1	dijk.	Med	High	mcf	High	Low
MC2	parser	Med	Low	vortex	High	High
MC3	jpeg	Med	Low	art	High	High
	Case 5					
	Prog.	Load	Hit rate			
MC0	susan	Low	High			
MC1	bzip2	Low	Low			
MC2	equake	Low	High			
MC3	fft	Low	Low			

6.2 Experiments

Figures 8 and 9 show the memory utilization and average memory access latency of the synthetic cases. Each case has four sets of bars, each set is for memory controllers 0 (left), 1, 2, and 3 (right). The proposed method gives 0%~8.79% and 0%~14.46% improvement in memory utilization and average memory access latency, respectively. Considering the traffic patterns in Table 3, most of performance improvement comes from the memory channels having low row buffer hit rates. It is because such memory channels have high demand of row activations. Thus, they benefit from the budget sharing.

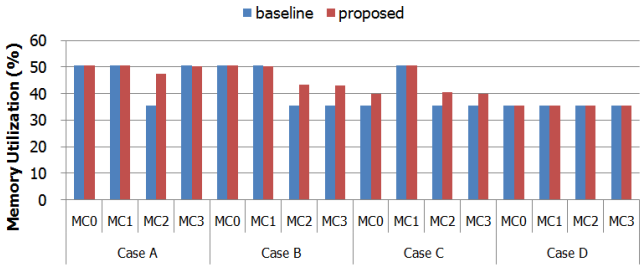

Figure 8 Memory utilization in synthetic cases

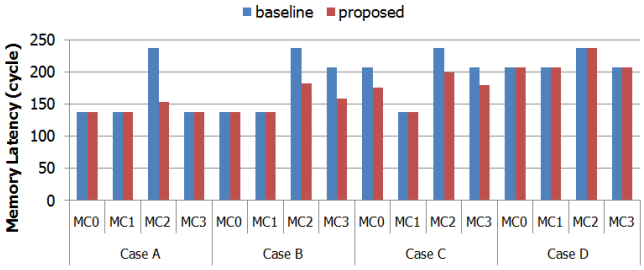

Figure 9 Latency in synthetic cases

Figures 10 and 11 show the benchmark cases. The proposed method offers average 4.72% and 8.31% improvement in average memory utilization and latency, respectively. As in the synthetic cases, the performance gain mostly comes from the memory channels with low row buffer hit rates (i.e., high row activation demand). The gain becomes prominent when the memory load is low. It is because the cases of low memory load can have more chances of having idle memory channels which allows the budget sharing.

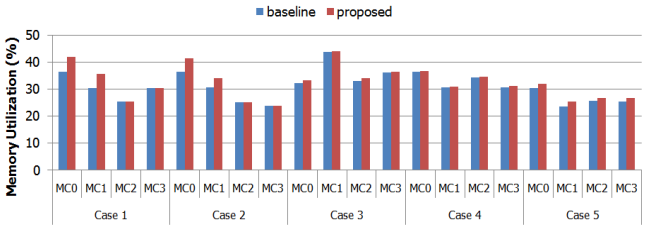

Figure 10 Memory utilization in benchmark cases

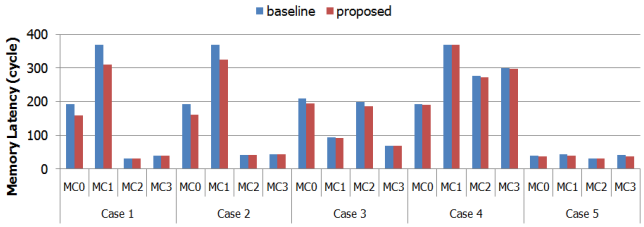

Figure 11 Latency in benchmark cases

Figure 12 show the effect of wire delay between the coordinator and memory controllers in Case 1. We vary the delay from 1 to 17 cycles. As the delay is increased, the memory utilization gain drops. It is because as the wire delay is increased, each memory controller needs to wait for more delay to receive the budget and perform row activations based on the updated budget.

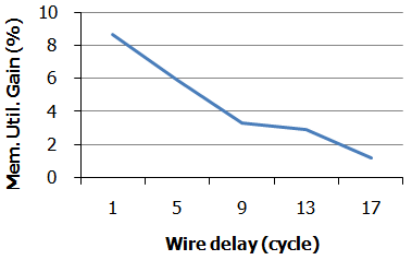

Figure 12 Memory utilization gain v.s. wire delay

Table 5 shows the average period that the coordinator updates the budget allocation for the five cases of benchmark programs.

Table 5 Average period (in cycles) of budget update

	Case 1	Case 2	Case 3	Case 4	Case 5
Average	99	93	89	82	61

6.3 Discussion

Our current solution utilizes a centralized coordinator which can suffer from large wire delay as shown in Figure 12. In our future work, we will work on distributed solutions to mitigate the limitation of long wire delay. Utilizing additional power TSVs between DRAM and SoC can mitigate the current problem that the same timing constraints of tRRD and tFAW are applied to multi-channel 3D stacked DRAM [5]. In such a case, another possibility will be thermal issues incurred by frequent executions of power-consuming row activation, which can require performance throttling to limit again total row activations. Such a scenario needs to be investigated for widespread adoptions of 3D stacked DRAM in mobile devices as well as in high-end devices since mobile devices are vulnerable to thermal issues due to the lack of active cooling facilities. Reconsidering DRAM architecture in terms of power consumption, e.g., smaller row size [9], will also be useful for both performance and power efficiency though DRAM manufacturers often try to avoid such architectural changes due to possible low area efficiency [6].

7. Conclusion

In this paper, we introduced a performance problem in accessing 3D stacked DRAM and proposed a method of sharing the budget of row activation across DRAM channels. The inter-memory controller coordinator receives periodically the demand of row activation from memory controllers and re-distributes the budget of row activation in order to improve memory performance. Experimental results show that the proposed method gives average 4.72% and 8.31% improvement in average memory utilization and latency, respectively.

8. Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2010-0007909 and 2010-0015336). This work was supported by IC Design Education Center (IDEC).

9. References

[1] S. Borkar, "Thousand-Core Chips - A Technology Perspective," Proc. DAC, 2007.

[2] G. H. Loh, "3D-Stacked Memory Architectures for Multi-Core Processors," Proc. ISCA, 2008.

[3] H. Lee, "3D Stacked Memory Design," 3D IC Workshop, Daejeon, Korea, Jan. 2010.

[4] Tezzaron Semiconductor, http://www.tezzaron.com/

[5] U. Kang, *et al*., "8 Gb 3-D DDR3 DRAM Using Through-Silicon-Via Technology," IEEE Journal of Solid-State Circuits, vol. 45, issue 1, pp. 111-119, Jan. 2010.

[6] T. Vogelsang, "Understanding the Energy Consumption of Dynamic Random Access Memories," Proc. MICRO 2010.

[7] D. Lee, S. Yoo, K. Choi, "Entry Control in Network-on-Chip for Memory Power Reduction," Proc. ISLPED, 2008.

[8] J. Ahn, *et al*., "Multicore DIMM: an Energy Efficient Memory Module with Independently Controlled DRAMs," Computer Architecture Letters, Vol.8, No. 1, October 2008.

[9] ISCA2010, HP Labs. A. N. Udipi, et al, "Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores," Proc. ISCA, 2010.

[10] Micron Tech. Inc., "TN-47-04: Calculating Memory System Power for DDR2," available at http://download.micron.com/

[11] B. Jacobs, S. Ng and D. Wang, Memory Systems: Cache, DRAM, Disk, Morgan Kaufmann, 2007.

[12] B. Akesson, K. Goossens, and M. Ringhofer, "Predator: a Predictable SDRAM Memory Controller," Proc. CODES+ISSS, 2007.

[13] Sonics Inc., "Sonics MemMax DRAM Access Scheduler," available at http://www.sonicsinc.com/.

[14] T. Ezaki, *et al*., "A 160Gb/s Interface Design Configuration for Multichip LSI," Proc. ISSCC, 2004.

[15] S. Rixner, *et al*., "Memory Access Scheduling," Proc. ISCA, 2000.

[16] I. Hur and C. Lin, "Adaptive History-based Memory Schedulers," Proc. MICRO, 2004.

[17] S. Heithecker and R. Ernst, "Traffic Shaping for an FPGA based SDRAM Controller with Complex QoS Requirements," Proc. DAC, 2005.

[18] O. Mutlu and T. Moscibroda, "Stall-time Fair Memory Access Scheduling for Chip Multiprocessor," Proc. MICRO, 2007.

[19] O. Mutlu and T. Moscibroda, "Parallelism-Aware Memory Access Scheduling," Proc. ISCA, 2008.

[20] Z. Zhu, Z. Zhang and X. Zhang, "Fine-grain Priority Scheduling on Multi-channel Memory Systems," Proc. HPCA, 2002.

[21] D. Abts, *et al*., "Achieving Predictable Performance through Better Memory Controller Placement in Many-Core CMPs," Proc. ISCA, 2009.

[22] Y. Kim, *et al*., "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers," Proc. HPCA, 2010.

[23] W. Kwon, *et al*., "An Open-Loop Flow Control Scheme Based on the Accurate Global Information of On-Chip Communication," Proc. DATE, 2008.

[24] W. Kwon, *et al*., "A Practical Approach of Memory Access Parallelization to Exploit Multiple Off-chip DDR Memories," Proc. DAC, 2008.

[25] W. Kwon and S. Yoo, "In-Network Reorder Buffer To Improve NoC Performance While Resolving the In-Order Requirement Problem," Proc. DATE, 2009.

[26] D. Kim, S. Yoo and S. Lee, "A Network Congestion-Aware Memory Controller," Proc. NOCS, 2010.