Heterogeneous Memory Management for 3D-DRAM and external DRAM with QoS

Le-Nguyen Tran, Fadi J. Kurdahi, Ahmed M. Eltawil
EECS, University of California Irvine
Irvine, CA, USA
Email: {ltran15,kurdahi,aeltawil}@uci.edu

Houman Homayoun
George Mason University
Fairfax, Virginia, USA
Email: hhomayou@eng.ucsd.edu

***Abstract*—This paper presents an innovative memory management approach to utilize both 3D-DRAM and external DRAM (ex-DRAM). Our approach dynamically allocates and relocates memory blocks between the 3D-DRAM and the ex-DRAM to exploit the high memory bandwidth and the low memory latency of the 3D-DRAM as well as the high capacity and the low cost of the ex-DRAM. Our simulation shows that in workloads that are not memory intensive, our memory management technique transfers all active memory blocks to the 3D-DRAM which runs faster than the ex-DRAM. In memory intensive workloads, our memory management technique utilizes both the 3D-DRAM and the ex-DRAM to increase the memory bandwidth to alleviate bandwidth congestion. Our approach supports Quality of Service (QoS) for "latency sensitive", "bandwidth sensitive", and "insensitive" applications. To improve the performance and satisfy a certain level of QoS, memory blocks of different application types are allocated differently. Compared to the scratchpad memory management mechanism, the average memory access latency of our approach decreases by 19% and 23%, while performance improves by up to 5% and 12% in single threaded benchmarks and multi-threaded benchmarks respectively. Moreover, using our approach, applications do not need to manage memory explicitly like in the scratchpad case. Our memory block relocation comes with negligible performance overhead, particularly for applications which have high spatial memory locality.**

I. Introduction

Microprocessor architecture has entered the multicore and many-core era to overcome the exponentially growing power consumption problem. One major obstacle of the multi-core architecture is the "memory wall". Because the memory bandwidth is quite limited and is shared by all cores, it typically creates a congestion bottleneck. Moreover, memory latency will also be affected if the bandwidth becomes congested. According to the well-known "queuing theory" latency will increase exponentially when the bandwidth utilization is increased. Currently, the memory latency is already as high as hundreds of clock cycles and highly impacts the performance of the entire system. If memory latency continues to increase, the benefit of multi-core processor architecture will be diminished. In addition to performance, DRAM power is also of great concern especially for mobile systems and servers. To improve DRAM performance, architects have been increasing the number of memory modules and their densities. However, these approaches increase power consumption and therefore operating temperature of DRAM to an extent that existing DRAM modules now operate at $95\,^{\circ}\mathrm{C}$ under some workloads[13].

Prior research [1,9,10,11,12,22] has shown that 3D-DRAM provides significant advantages in terms of performance while enabling energy-efficient computing. 3D-DRAM has a number of superior characteristics namely high bandwidth, low latency, and low power consumption. For example, a single Hybrid Memory Cube (HMC) [15,20] can provide more than $15\times$ the performance of a DDR3 module while utilizing 70% less energy per bit than conventional DDR3 DRAM technologies. Although 3D-DRAM accelerates the thermal challenges because of the higher thermal resistivity resulting from vertical stacking, innovative cooling methods [21] may be used to solve this issue.

Although 3D-DRAM has a number of advantages, access to ex-DRAM is indispensable in many applications. In general, the capacity of the 3D-DRAM chip is not as high as the capacity of the external DRAM and it is also not cost efficient to manufacture at the current time. As a result, our approach combines the 3D-DRAM and the ex-DRAM to create a heterogeneous memory to achieve high bandwidth, low latency, high capacity, and low cost systems.

In this paper, a memory management mechanism utilizing both 3D-DRAM and ex-DRAM as a heterogeneous memory system is proposed. Our memory management mechanism integrates the OS virtual to physical address translation so that applications use 3D-DRAM and/or ex-DRAM transparently. In other words, applications do not need to manage the memory explicitly (like scratchpad memory mechanism). To improve system performance, applications should utilize the 3D-DRAM which has small memory latency and high memory bandwidth. Therefore, our memory management mechanism allocates memory blocks so that many actively access memory blocks are in the 3D-DRAM. Because applications change their memory access pattern dynamically, our approach provides a memory relocation mechanism to relocate memory blocks between the 3D-DRAM and the ex-DRAM. To relocate memory blocks, a monitoring unit is used to measure the memory utilization rate of both the 3D-DRAM and the ex-DRAM periodically. Due to the strong correlation between the memory latency and the memory utilization rate (queuing theory), measuring the memory utilization rate can be used to estimate memory latency. On one hand, if the 3D-DRAM memory utilization rate is low which is the case when latency is small and bandwidth is available, the relocation mechanism moves some ex-DRAM memory blocks to the 3D-DRAM. On the other hand, if the 3D-DRAM memory utilization rate is very high (which is the case when it is congested) and latency is high and ex-DRAM memory utilization rate is low, the relocation mechanism moves some 3D-DRAM memory blocks to the ex-DRAM. Moreover, applications are classified into three types namely "latency sensitive", "bandwidth sensitive", and "insensitive" with priority. Our memory management mechanism not only provides Quality of Service (QoS) for application types [2] but also reserves the 3D-DRAM which is superior to the ex-DRAM for applications having higher priority.

The structure of this paper is as follows. In section II, we introduce some related works. In section III, we present our heterogeneous memory architecture. Then we introduce and analyze our QoS-aware memory management mechanism in section IV. The dynamic memory allocation and relocation mechanisms are presented in section V. In section VI, we present the simulation results. Finally, conclusions are drawn in section VII.

978-1-4673-3030-5/13/$31.00 ©2013 IEEE

II. Related work

3D-DRAM is a state-of-art technology which has been proposed recently to lower the power consumption of the memory subsystem [16]. Several design specifications of 3D DRAM have been standardized recently. For instance, JEDEC has already created the specification for Wide IO DRAM targeting low power memory [17]. In this case, 3D-DRAM is planned to be used as the main memory or combined with LPDDR2 DRAM as the scratchpad memory. As discussed previously, because the capacity of 3D-DRAM is limited, this technique can only be beneficial for some special applications. Moreover, developing software for scratchpad memory is a challenging problem.

3D-DRAM is proposed to be used as the main memory in [4]. The key idea is to remove the L2 cache so that many simple processor cores can be integrated into the same die. The proposed approach then uses 3D-DRAM to provide very high memory bandwidth for the cores. This architecture is targeting high multi-threading server applications.

3D-DRAM is also proposed to be used as cache and main memory in [5]. In this paper, the authors realized that the latencies of large L2 SRAM caches are high, mainly due to the large access latency of H-tree. The authors proposed to use TSV to interconnect the processor cores and the caches. This will help the 3D-DRAM cache to be as fast as the SRAM cache. However, as presented in [6], the performance improvement of using 3D-DRAM as the Last Level Cache (LLC) is not comparable with the performance improvement of using the heterogeneous memory system. Therefore, in this paper, our approach is compared with the scratchpad memory method (using both 3D-DRAM and ex-DRAM) instead of using 3D-DRAM as the LLC.

The related work which is closest to our proposed system is presented in [6]. This research also utilizes both 3D-DRAM and ex-DRAM as a heterogeneous memory system and implements a memory migration to exchange memory blocks between them. Compared with our work, this approach does not provide QoS for various application types. Moreover, the memory migration mechanism is very complex and comes with a large overhead. It requires the size of memory pages to be large (4MB) to reduce the total number of memory pages to control. Because many Operating Systems (OS) use small memory block sizes (4KB), the authors have to add one more address translation layer which translates the physical address to the real memory address. The latency of this address translation is two clock cycles, which affects all memory requests. In our work, we propose a simple approach utilizing small memory block size (4KB) which can integrate into the OS virtual address to physical address translation and does not incur any latency overhead for memory accesses. Moreover, for memory block assignment our approach is adapted based on the application type to optimize the performance of the whole system.

In contemporary systems, it is common that many applications run simultaneously, with different requirements for memory bandwidth and memory access latency[19]. Therefore, the performance of the system can be improved if a QoS mechanism is provided. Differentiating application types to provide QoS for homogeneous memory systems is presented in [7]. Our approach is different as it targets heterogeneous memory systems, a more challenging problem in today's complex architecture.

III. Heterogeneous Memory architecture

Our baseline architecture is shown in Fig. 1. There are two distinct memory channels: one connecting to 3D-DRAM and the other connecting to ex-DRAM. 3D-DRAM has two independent memory controllers. Each memory controller has three queues for

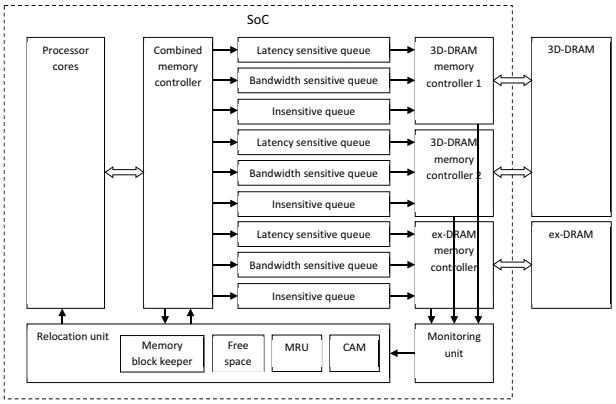

Fig. 1. Baseline architecture.

memory requests of different application types. The memory requests from the processor cores are pushed into the combined memory controller. Then the combined memory controller will translate virtual to physical address, specify the corresponding controller and queue, and then push the request to the queue. To implement a memory relocation mechanism, a monitoring unit measures the memory utilization rate of the memory controllers. When the monitoring unit detects that the system does not operate under the optimum performance condition (for instance the memory bandwidth is congested), it signals the relocation unit. The relocation unit relocates a single memory block between the ex-DRAM and 3D-DRAM.

Without loss of generality, similar to [14] we assume in this paper that the bandwidth of the 3D-DRAM is two times higher than the bandwidth of the ex-DRAM and use two memory controllers for the 3D-DRAM. Because the latency of 3D-DRAM is smaller, we assume that the read latency of the ex-DRAM and the 3D-DRAM are 12 and 8 memory clock cycles respectively. The write recovery latency of the 3D-DRAM is also 4 clock cycles smaller than the write recovery latency of the ex-DRAM. The other latency parameters are the same.

In addition to controlling memory requests, memory controllers also support QoS for different application types. Applications are classified into three types, namely "latency sensitive", "bandwidth sensitive" and "insensitive". Application classification needs to be implemented by programmers in advance. "Latency sensitive" is the type of applications whose performances are highly affected by long memory latency. Many general purpose applications belong to this type [3]. "Bandwidth sensitive" describes those applications whose performances are slightly affected by long memory latency but highly affected by limited memory bandwidth. Many multimedia applications belong to this type [3]. Finally, "insensitive" applications' performances are slightly affected by both long latency and limited bandwidth. They are often not memory intensive applications. Each memory controller has tree queues to support three application types. To provide QoS, the memory controllers have a special priority mechanism which will be presented in the next section.

The monitoring unit measures the memory utilization rates of the memory controllers for our relocation mechanism. Because the latency and bandwidth utilization rate are highly correlated, measuring the memory bandwidth utilization rate can estimate the memory latency. Measuring the memory bandwidth utilization is simple. There is one counter attached to each memory controller. During a certain period of time, the counter measures the number of clock cycles in which the controller is used. These values correspond to the bandwidth utilization. If our system operates under the optimum performance condition (for example the memory access latency is

small and the memory bandwidth is sufficient), no memory block relocation is needed. Otherwise, the monitoring unit will signal the relocation unit to relocate memory blocks.

Inside the relocation unit, Most Recently Used (MRU) policy is employed to select the memory blocks to be relocated. MRU registers record the MRU addresses for three application types and both the 3D-DRAM and the ex-DRAM. The values of MRU registers are valid only for a certain period of time (MRU lifetime which start from the time the register with the MRU address is updated). At the end of MRU lifetime, the MRU address is invalidated. Note that within an MRU lifetime if the MRU address changes, the MRU register is updated with the new MRU address.

To transfer the MRU block to the 3D-DRAM or the ex-DRAM free spaces are needed. With the help of the OS, our free space registers store the addresses of one free space memory block of the 3D-DRAM and one free space memory block of the ex-DRAM. If there is no more free space in the ex-DRAM and memory blocks need to be transferred to the ex-DRAM, some blocks are moved from the ex-DRAM to the hard drive. If there is no more free space in the 3D-DRAM and memory blocks need to be transferred to the 3D-DRAM, some memory blocks which have not been accessed recently are moved from the 3D-DRAM to the ex-DRAM. Content Addressable Memory (CAM) is used to specify the "un-accessed" blocks. The mechanism is as follows. Each memory block of the 3D-DRAM corresponds to one bit in the CAM unit. When one 3D-DRAM block is accessed, its bit in the CAM unit is set. After a certain period of time, for example 1M cycles, the CAM array is checked to find "L", for example 64, un-accessed blocks whose CAM bit is "0". Then their addresses are recorded to a smaller CAM. After that the big CAM array is reset. Similarly, the small CAM array bit is set when its corresponding memory block is accessed. To find the un-accessed memory blocks the small CAM array is checked for value "0" .

Because our approach supports QoS, a priority mechanism for different application types is implemented. The order of priority from high to low is: "latency sensitive", "bandwidth sensitive", and "insensitive". Therefore, when one memory block needs to be moved from the ex-DRAM to the 3D-DRAM, and when there is no free space and no un-accessed block in the 3D-DRAM, the ex-DRAM block is swapped with a 3D-DRAM memory block whose priority is lower. The memory block keeper unit stores the addresses of one "bandwidth sensitive" block and one "insensitive" block of 3D-DRAM for memory relocation.

When memory blocks are relocated, the TLB is updated for address translation. That is done by hardware and the OS. When memory blocks are relocated or swapped between 3D-DRAM and ex-DRAM cache data which uses a physical address (physical index or physical tag) is also updated. Note that for caches, which use a physical tag, only the tag needs to be updated. On the other hand, for caches, that use a physical index, the cache data needs to be evicted. Updating the tag and/or evicting cache data is implemented by hardware.

IV. QoS-Aware Memory Management

In this section our QoS-aware memory management policy is explained. Our QoS-aware memory management policy optimizes the system to improve performance and reduce power.

A. Queuing theory observation

In this section we study the impact of memory bandwidth utilization on memory access latency based on the well-known queuing theory. Fig. 2 shows our simulation results for the total memory access latency as a function of bandwidth utilization and for a number of requesting cores (active cores that are running a thread). The result indicates that when the memory throughput is small (throughput

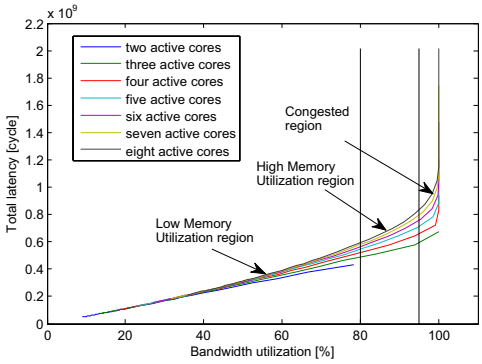

Fig. 2. Relation between the total memory access latency and the memory bandwidth utilization.

= utilization x Max available bandwidth), the total memory access latency is quite linear. This is in fact due to the negligible queuing effect, which results in a somewhat constant average memory latency. On the other hand, when the bandwidth utilization is almost 100%, the queuing effect is very high and therefore the average memory latency increases significantly. At the same bandwidth utilization rate, as the number of requesting cores increases the memory access latency becomes larger.

Based on the results shown in Fig. 2. the memory bandwidth utilization rate is divided into three regions, namely "low memory utilization region" (in brief LMU-region), "high memory utilization region" (HMU-region), and "congested region" (C-region) (shown in Fig. 2). LMU-region represents a case where the queuing effect is small. In this region, increasing the memory bandwidth utilization does not noticeably change the average memory access latency. Therefore, working in this region is preferable, because one can increase the memory throughput with only slightly impacting the memory latency. In the HMU-region, the queuing effect is high. Compared to the LMU-region, in the HMU-region a slight increase in the memory throughput is accompanied by a large rise in the average memory latency. Working in this region is acceptable because the memory throughput can still increase at the cost of high average memory access latency. In the C-region the memory bandwidth is congested. Working in this region is undesirable because slightly increasing the memory throughput would significantly impact the memory access latency. In our simulation, 80% and 95% memory bandwidth utilization are chosen as the thresholds between the LMU-region, the HMU-region, and the C-region, respectively.

B. Application classification and priority

As mentioned previously, applications are classified into three types, namely "latency sensitive", "bandwidth sensitive", and "insensitive" applications. We now consider the priority of these applications. It is clear that "insensitive" applications should have low priority because long memory latency and limited memory bandwidth only slightly affect their performance. Moreover, the "latency sensitive" applications are often general purpose applications and the "bandwidth sensitive" applications are often multi-media applications. We assume that the general purpose applications are more important than the multi-media applications (multi-media applications can run at smaller memory bandwidth with low quality). Thus, the priorities of the "latency sensitive", "bandwidth sensitive", and "insensitive" applications are very high, high, and low respectively.

Our approach implements the priority mechanism for memory request handling and memory allocation/relocation. As shown in Fig. 1, each memory controller has three queues for three application type requests. Normally, the "first come first served" mechanism

is implemented for the memory controllers. However, when two or three memory requests coming from different queues are available simultaneously, the request coming from the "latency sensitive" queue will be processed for the first "M" times. Then round-robin technique is used to grant requests coming from "bandwidth sensitive" and "insensitive" queues. Once the controller grants the first request coming from the "bandwidth sensitive" queue, it processes up to "N" consecutive requests from the same queue. For the memory allocation/relocation priority mechanism, the 3D-DRAM is reserved for applications having higher priority. For example, on one hand, when the 3D-DRAM operates under the LMU-region, memory blocks of high priority applications are considered moving from the ex-DRAM to the 3D-DRAM. On the other hand, when the 3D-DRAM operates under the C-region, memory blocks of low priority applications are considered moving from the 3D-DRAM to the ex-DRAM.

V. Memory allocation/relocation mechanisms

Memory allocation/relocation mechanism is the key point of our approach to optimize the operation of our system. As mentioned previously, our memory management mechanism integrates with the OS virtual to physical address translation to manage the heterogeneous memory system so that applications can access memory transparently. To improve the performance of the system, applications should utilize the 3D-DRAM to benefit from reduced memory access latency and high bandwidth. When new memory blocks are allocated, our approach attempts to utilize the 3D-DRAM first. If the 3D-DRAM controllers operate under the LMU-region and the 3D-DRAM has free space, the new blocks are allocated to the 3D-DRAM. Otherwise, they are allocated to the ex-DRAM. The monitoring unit periodically measures the memory utilization rate of the 3D-DRAM and ex-DRAM controllers for the memory relocation mechanism. If the 3D-DRAM controllers operates under LMU-region, one ex-DRAM MRU memory block is transferred to the 3D-DRAM. When there are more than one ex-DRAM MRU blocks available, the one having highest priority (for example "latency sensitive" MRU block) will be selected. If the 3D-DRAM controllers operate under HMU-region, the 3D-DRAM is reserved for applications having high priority only. Our approach swaps the 3D-DRAM MRU memory block having low priority and the ex-DRAM MRU memory block having higher priority.

Our simulation results show that swapping memory blocks improves the 3D-DRAM memory access latency. As shown in Fig. 2, at the same memory bandwidth utilization, the memory access latency can decrease if there are fewer active cores. Swapping memory blocks for different application types allows the 3D-DRAM to be used only for high priority applications. Hence, the number of active cores accessing the 3D-DRAM decreases.Finally, If the 3D-DRAM controllers operates under C-region, our approach first swaps the 3D-DRAM MRU memory block having low priority and the ex-DRAM MRU memory block having high priority. Then if the ex-DRAM controller operates under LMU-region, one 3D-DRAM MRU memory block is transferred to the ex-DRAM. When there are multiple 3D-DRAM MRU blocks available, the one having lowest priority is selected. Fig. 3 shows the flowchart of our allocation/relocation mechanism.

For memory relocation, free space is needed. As mentioned previously, memory blocks can be moved from the ex-DRAM to the hard drive to free up space. For the 3D-DRAM if there is no free space,one memory block can be dumped from the 3D-DRAM to the ex-DRAM. Initially, the CAM is used to check for the un-accessed blocks. Then the memory blocks of the low priority applications can be dumped for the high ones.

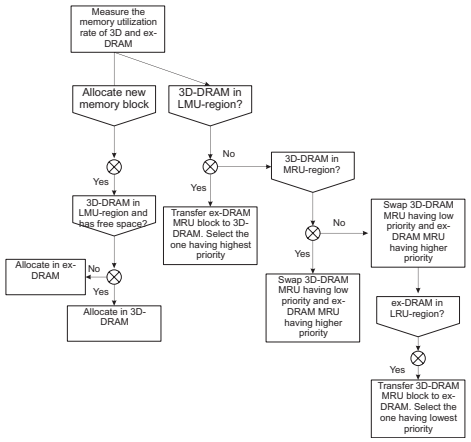

Fig. 3. Flowchart of the memory relocation mechanism.

VI. Simulation results

To prove our concept the SMTSIM simulator [18] is used. The ex-DRAM is modelled based on Micron DDR3-1600 specification. For the 3D-DRAM, we shorten the read and write latency to two thirds of the ex-DRAM's latencies and double its bandwidth. There are two independent memory controllers that access interleaving memory space of the 3D-DRAM. In the processor model, we use infinite length for the queues of the memory controllers. Therefore, our model is not accurate when the memory bandwidth is congested (in this case the number of memory requests inside the queues is very large). As a result, we do not get simulation results in the congested cases. To simulate with different bandwidth, the ex-DRAM bus is set to 64, 128, and 256 bits (3D-DRAM bus is always double) (cases in Fig. 4,5,6,7,8,9 imply the ex-DRAM bus wide). We run benchmark simulation for our heterogeneous memory system using our memory management mechanism, scratchpad memory mechanism (3D-DRAM scratchpad memory + ex-DRAM), in the best case (access all memory inside 3D-DRAM) and the worst case (access all memory inside ex-DRAM).

Based on the SPEC2006, we select 13 benchmarks as the workload for our simulation. The workloads are "lbm_06", "swim", "lucas", "applu", "bwave_06", "lib_quantum_06", "mgrid", "gcc_06_typeck", "art_470", "vpr_route", "gap", "bzip2_source", and "galgel". They are listed in order of memory bandwidth utilization decrement. We assume that all of these benchmarks are "latency sensitive" applications. We run simulation in different scenarios. First of all, we simulate each benchmark as single thread applications. Then we simulate 2 and 4 benchmarks simultaneously. These simulations show the performance of the multi-threaded applications. Table 1 summarizes our simulation benchmark workload. Fig. 4 and Fig. 5 shows the average latencies of these cases. As we can see our memory management mechanism provides the equivalent average latency compared to the best case of the scratchpad memory mechanism. That is understandable because our memory management mechanism brings all accessed memory blocks to 3D-DRAM when its memory controllers operate under the LMU-region. Compared to the worst case of the scratchpad memory mechanism, the average latency of our approach decreases 19% and 23% for the single threaded and multi-threaded benchmarks respectively. Fig. 6 and Fig. 7 show the weighted speedup of our memory management compared to the worst case of the scratchpad memory mechanism (performances of our memory management mechanism and the best case of scratchpad memory mechanism are equivalent). There are four simulation cases

TABLE I
SIMULATION BENCHMARK WORKLOAD

Single thread workload			
1T0	lbm_06	1T7	gcc_06_typeck
1T1	swim	1T8	art_470
1T2	lucas	1T9	vpr_route
1T3	applu	1T10	gap
1T4	bwave_06	1T11	bzip2_source
1T5	lib_quantum_06	1T12	galgel
1T6	mgrid		
Two thread workload			
2T0	lbm_06+swim	2T6	mgrid+gcc_06_typeck
2T1	swim+lucas	2T7	gcc_06_typeck+art_470
2T2	lucas+applu	2T8	art_470+vpr_route
2T3	applu+bwave_06	2T9	vpr_route+gap
2T4	bwave_06+lib_quantum_06	2T10	gap+bzip2_source
2T5	lib_quantum_06+mgrid	2T11	bzip2_source+galgel
Four thread workload			
4T0	lbm_06+swim+lucas+applu		
4T1	swim+lucas+applu+bwave_06		
4T2	lucas+applu+bwave_06+lib_quantum_06		
4T3	applu+bwave_06+lib_quantum_06+mgrid		
4T4	bwave_06+lib_quantum_06+mgrid+gcc_06_typeck		
4T5	lib_quantum_06+mgrid+gcc_06_typeck+art_470		
4T6	mgrid+gcc_06_typeck+art_470+vpr_route		
4T7	gcc_06_typeck+art_470+vpr_route+gap		
4T8	art_470+vpr_route+gap+bzip2_source		
4T9	vpr_route+gap+bzip2_source+galgel		

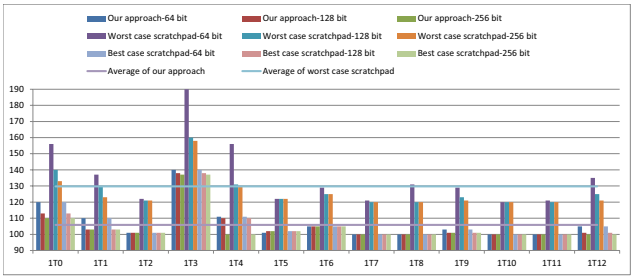

Fig. 4. Average latencies in clock cycles of single threaded benchmarks.

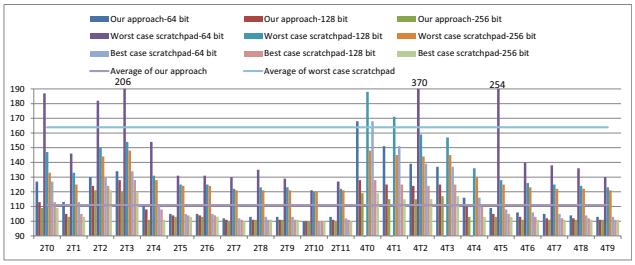

Fig. 5. Average latencies in clock cycles of multi-threaded benchmarks.

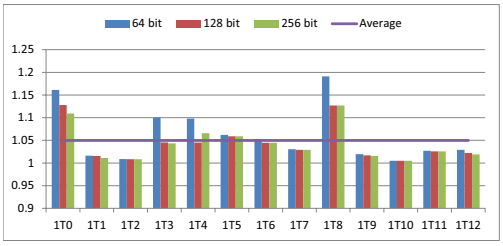

Fig. 6. Weighted speedup of our approach compared to the worst case of scratchpad memory for single threaded benchmark.

in which the memory bandwidth is congested (not shown in the figures). Compared to the worst case of the scratchpad memory mechanism, the average of the performance improvement of our approach is 5% and 12% for the single threaded and multi-threaded benchmarks respectively.

To verify our QoS mechanism and test the performance of the proposed memory mechanism in the high memory workload scenario, we run single thread benchmarks with a blocker. The blocker is the "bandwidth sensitive" application accessing consecutive memory addresses at a constant rate. The blocker occupies 63% memory bandwidth of the 3D-DRAM. In this case, our mechanism is compared to

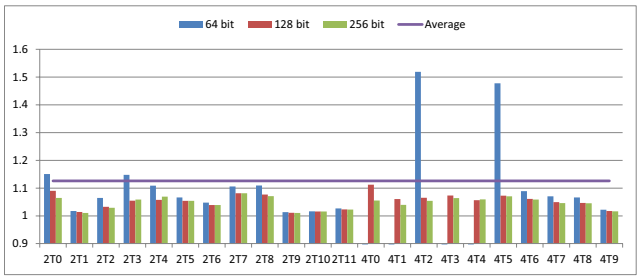

Fig. 7. Weighted speedup of our approach compared to the worst case of scratchpad memory for multi-threaded benchmark.

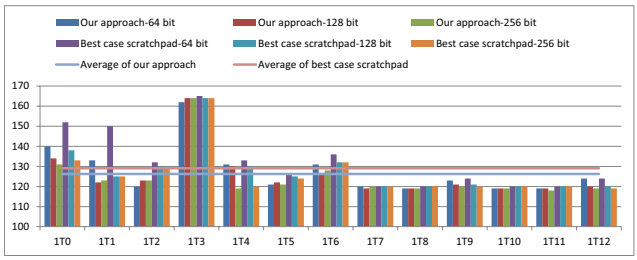

Fig. 8. Average latencies in clock cycles of single threaded benchmarks with blocker.

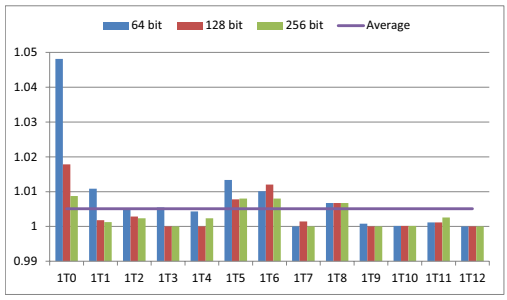

Fig. 9. Weighted speedup of our approach compared to the best case of scratchpad memory for single threaded benchmark with blocker.

the best case of the scratchpad memory mechanism only (the worst case of the scratchpad memory mechanism is congested). To see the difference between our memory management and the scratchpad mechanisms, we change the threshold values between the LMU-region, HMU-region, and C-region to 55% and 80% respectively. Fig. 8 and Fig.9 show the average latency and weighted speedup in these cases. As shown, the proposed mechanism offers smaller memory access latency. This is because our QoS mechanism transfers one part of the access memory of the blocker to the ex-DRAM (it has lower priority). However, the improvement is small. That is understandable because the difference of the average latencies in the LMU-region is small (for example the difference of latencies of 55% and 80% of memory utilization rate is small).

Our simulation results can be explained as follows. For single threaded and multi-threaded benchmarks, there are some benchmarks requiring high memory bandwidth like 1T0, 4T2. For these benchmarks, the latency decreases when the memory bus is extended and the improvement of our approach performance is also significantly high because of our high memory bandwidth and low latency of 3D-DRAM. Other benchmarks like 1T10, 2T10 do not require large memory bandwidth. As a result, extending memory bandwidth does not reduce memory latency significantly and the improvement of our approach performance comes from low latency of 3D-DRAM only. For the blocker simulation cases, our approach offer significantly higher performance than the best case of scratchpad memory

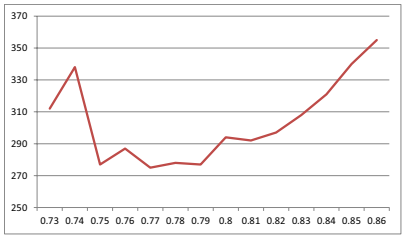

Fig. 10. Average latencies in clock cycles when the values of threshold 1 and 2 are adjusted (X-axis is threshold 1).

mechanism for benchmarks requiring high memory bandwidth like 1T0. The reason is that our approach move more blocker workload to the ex-DRAM. The performances of our approach and the best case of scratchpad memory mechanism are similar for benchmarks requiring low memory bandwidth like 1T10 because memory access latency changes slightly if the memory utilization rates are within the LMU-region. Moreover, because the blocker always occupies 63% memory bandwidth of the 3D-DRAM, extending memory bandwidth decreases memory access latency only for benchmarks requiring high memory bandwidth.

Finally, we run simulations to test how the proposed mechanism distributes the memory access workload to both the 3D-DRAM and the ex-DRAM in a high workload scenario. We simulate 5 threads: lbm_06+swim+lucas+applu+swim for the system using 32 bit ex-DRAM bus and 64 bit 3D-DRAM bus. In this case, the scratchpad memory mechanism is congested in both the best and the worst cases. Therefore, only the average latency of our mechanism is shown in Fig. 10. To adjust the distribution of the memory access workload, we change the threshold value between the LMU-region and HMU-region ($Threshold1$) from 0.73 to 0.86. The threshold value between the HMU-region and the C-region ($Threshold2$) is changed according to the following formula.

$$Threshold2 = \begin{cases} Threshold1 * 0.95/0.8 & \text{if } Threshold1 \leq 0.8, \\ 0.75 + Threshold1/4 & \text{if } Threshold1 > 0.8. \end{cases}$$

It is noted that selecting an optimum value for the thresholds is a challenging task for two reasons. First, the optimum threshold values depend highly on the memory access pattern of the applications. Second, we have to consider the memory block relocation overhead. Therefore, we currently use the static threshold values which are chosen heuristically. We plan to extend our algorithm to use dynamic threshold values in future research.

VII. Conclusions

This paper proposes an innovative dynamic memory management approach with QoS to exploit the high bandwidth low latency of the 3D-DRAM and high capacity of the ex-DRAM. The dynamic memory management approach measures the memory bandwidth utilization and balances the workload for the heterogeneous memory system. It is shown to be very effective in alleviating memory congestion for multi-core processor systems. The proposed system is very simple to implement. A hardware unit is used to monitor the memory state to generate interrupts only when relocation is needed. Moreover, our approach integrates our mechanism to the virtual-physical address translation so that there is no additional latency to determine the memory location. In addition, our QoS mechanism further improves the performance of the system. It allocates memory space adaptively depending on the application type. Simulation results prove that the performance of our approach is equivalent to or superior to the best case of the scratchpad memory mechanism in many cases. Compared to the worst case of the scratchpad memory mechanism, the average performance improvement of the proposed approach is 5% and 12% for the single threaded and multi-threaded benchmarks respectively. While the accuracy of the proposed model does not allow for congestion based simulation, it is clear that the performance of our mechanism is superior to the performance of the scratchpad memory mechanism in the congested cases because our approach can utilize both the 3D-DRAM and the ex-DRAM to have higher memory bandwidth.

References

[1] A. B. Kahng, and V. Srinivas, *Mobile System Considerations for SDRAM Interface Trends*. IEEE System Level Interconnect Prediction, 2011.

[2] T. Lin, K. Lee, and C. Jen, *Quality-aware memory controller for multi-media platform SoC*. IEEE Signal Processing Systems SIPS, 2003.

[3] Murphy R, *On the Effects of Memory Latency and Bandwidth on Super-computer Application Performance*. IEEE Workload Characterization IISWC, 2007.

[4] Taeho Kgil, Ali Saidi, Nathan Binkert, Steve Reinhardt, Krisztian Flautner, and Trevor Mudge, *PicoServer: Using 3D stacking technology to build energy efficient servers*. ACM Journal on Emerging Technologies in Computing Systems (JETC), 2008.

[5] Sun H., Liu J., Anigundi R., Zheng N., Lu J., Ken R., and Zhang T, *Design of 3D DRAM and Its Application in 3D Integrated Multi-Core Computing Systems*. IEEE Design & Test of Computer, 2009.

[6] X. Dong, Y. Xie, N. Muralimanohar, and N. P. Jouppi, *Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support*. Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, 2010.

[7] T. Lin, K. Lee, and C. Jen, *Quality-aware memory controller for multi-media platform SoC*. IEEE Signal Processing Systems SIPS, 2003.

[8] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller, *Energy management for commercial servers*. IEEE Computer, 2003.

[9] Christianto C. Liu, Ilya Ganusov, Martin Burtscher, and Sandip Tiwari, *Bridging the Processor-Memory Performance Gap with 3D IC Technology*. IEEE Design and Test of Computers, 2005.

[10] Qi Wu, Ken Rose, Jian-Qiang Lu, and Tong Zhang, *Impacts of Though-DRAM Vias in 3D Processor-DRAM Integrated Systems*. IEEE 3D System Integration, 2009.

[11] U. Kang, H. Chung, S. Heo, D. Park, H. Lee, J. Kim, S. Ahn, S. Cha, J. Ahn, D. Kwon, J. Lee, H. Joo, W. Kim, Member, IEEE, D. Jang, N. Kim, J. Choi, T. Chung, J. Yoo, J. Choi, C. Kim, Senior Member, IEEE, and Y. Jun, *8 Gb 3-D DDR3 DRAM Using Through-Silicon-Via Technology*. IEEE Journal of Solid-State Circuits, 2010.

[12] H. Sun, N. Zheng, J. Liu, R. S. Anigundi, J. Lu, K. Rose, and T. Zhang, *3D DRAM Design and Application to 3D Multicore Systems*. IEEE Design and Test Computers, 2009.

[13] Q. Zhu, X. Li, and Y. Wu, *Thermal management of high power memory module for server platforms*. ITHERM'08.

[14] D. Woo, N. Seong, D. Lewis, and H.-H. S. Lee, *An Optimized 3DStacked Memory Architecture by Exploiting Excessive, High-Density TSV Bandwidth* . Proceedings of the International Symposium on High Performance Computer Architecture, 2010.

[15] Joe Jeddeloh, and Brent Keeth, *Hybrid memory cube new DRAM architecture increases density and performance*. IEEE Symposium on VLSI Technology (VLSIT), 2012.

[16] Jie Meng, Daniel Rossell, and Ayse K. Coskun, *3D Systems with On-Chip DRAM for Enabling Low-Power High-Performance Computing*. IEEE High Performance Embedded Computing, 2011.

[17] JEDEC, *Wide I/O single data rate (WIDE I/O SDR)*. JEDEC standard JESD229, 2011.

[18] D.M. Tullsen, *Simulation and Modeling of a Simultaneous Multi-threading Processor*. the 22nd Annual Computer Measurement Group Conference, 1996.

[19] Benny Akesson and Kees Goossens, *SDRAM Controllers for Mixed Time-Criticality Systems*. CODES+ISSS, 2011.

[20] Online, *Hybrid Memory Cube*. http://www.hybridmemorycube.org/.

[21] A. Coskun, Jie Meng, D. Atienza, and M. M. Sabry, *Attaining Single-Chip, High-Performance Computing through 3D Systems with Active Cooling*. IEEE Micro, 2011.

[22] Houman Homayoun, Vasileios Kontorinis, Ta-Wei Lin, Amirali Shayan and Dean M. Tullsen, *Dynamically heterogeneous cores through 3D resource pooling*. International Symposium on High-Performance Computer Architecture, HPCA 2012.