Achieving Efficient Packet-based Memory System by Exploiting Correlation of Memory Requests

Tianyue Lu[†§], Licheng Chen[†§], and Mingyu Chen[†]
[†]State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
[§]University of Chinese Academy of Sciences
{lutianyue, chenlicheng, cmy}@ict.ac.cn

***Abstract*—Packet-based interface is a trend for future memory system to alleviate memory capacity and bandwidth bottlenecks. On the other hand fine-grained memory access has been proven to efficiently reduce memory power. However leveraging both these two technologies will result in high packet overhead, because previous implementations of packet-based interface all adopt a simple design that a single packet is dedicated to a single request (SPSR). In this paper, we propose three optimizations to overcome the problem by exploiting correlations of memory requests. First, we propose a novel single packet multiple requests (SPMR) interface that encapsulates multiple requests into a packet to share packet header and tail. Second, we propose an adaptive address compression mechanism within a packet by adopting a base-difference algorithm. Third, we propose a mechanism to merge multiple memory requests with continuous access addresses into a single request before packing. By this way, the granularity constraint of cache line size is broken to enable efficiently row buffer scheduling. The experimental results show that, for certain memory-intensive workloads, the optimizations can effectively reduce packet overhead by about 53.9% and improve system performance by about 63.6% in average.**

I. Introduction

With the trend that increasing number of cores will be integrated into a processor chip, combined with significantly increasing working-set of applications (e.g. Big Data Processing), DRAM memory system will confront serious capacity and bandwidth pressure. However the conventional synchronous parallel DDRx (e.g. 3 or 4) bus interface, which is used to communicate between processor cores and DRAM memory channels, is failed to scale. It is due to a large number of interface pin counts (e.g. 240 processor pins for a DDR3 DRAM channel) and limited data rate on parallel bus. The memory capacity and bandwidth have become the main bottlenecks in chip-multi-processor (CMP) system.

Recently, emerging memory systems with asynchronous packet-based interface have been proposed to alleviate processor-pin limitations, which are capable of providing high capacity and high bandwidth (e.g. BOB [17], HMC [5]). In such interface the memory controller is divided into two parts: on-chip controller and off-chip controller. The on-chip controller communicates with the off-chip controller over a faster and narrower serial link bus with high level packet protocol, thus it can significantly reduce pin-counts for each memory channel and support more memory channels to achieve higher capacity. Furthermore, the serial link bus can work at a higher frequency than conventional DDRx bus (e.g. as fast as CPU clock rate) to provide sufficient memory bandwidth.

Meanwhile, memory power has become a severe problem. It has been reported that DRAM memory system contributes up to about 40% of the total system power in large systems [23, 25]. Thus how to improve memory power efficiency has become an important consideration. Previous work has shown that conventional coarse-grained memory accesses, which always read or write a cache block data (e.g. 64B), are very inefficient in terms of power and bandwidth when spatial locality is low [29, 30]. Since it often brings back a large portion of useless data from memory. Memory system that supports fine-grained memory access is an attractive alternative approach. It usually adopts sub-ranked memory which groups DRAM devices (in a wide 64-bit rank) into multiple narrow sub-ranks (e.g. 8-bit, 16-bit, 32-bit), and each sub-rank can be accessed independently. For a fine-grained memory request, it only needs to activate and access part of the devices (within a sub-rank), therefore it can significantly reduce memory power. In addition, only really useful data is returned instead of a whole cache block . Thus it can also improve memory bus efficiency.

The two trends expose the demand on future memory system to support both packet-based interface and fine-grained access. However, communicating with packet-protocol for fine-grained access is not a free lunch. Besides payload (request or data), extra data such as destination identifier, packet meta-data (e.g. size), or redundant data for integrity (e.g. CRC), are needed to add as packet header or packet tail. These are named as **packet overhead** in this paper. In previous implementations such as BOB [17], HMC [5], they all adopt the simplest design that a single packet is dedicated to a single request, which is named as **SPSR (Single-Packet-Single-Request)**. However the SPSR approach will result in heavy **packet overhead**. This is because that in fine-grained access, much smaller size of data (e.g. 8 bytes) is read from or written to memory devices, which is served as payload in a packet. As the payload decreases, the packet overhead is increasing. The memory bus efficiency is also decreased because a large portion of bandwidth is contributed to transferring extra packet-protocol data. Take HMC [5] for example,it adopts SPSR and the packet header and tail has fixed size in each packet, which are both 8-byte (64-bits), despite of what size of the payload data. HMC

This work is partially supported by the National Basic Research Program of China (973 Program) under a grant number 2011CB302502, the Strategic Priority Research Program under a grant number XDA06010401, the National Natural Science Foundation of China (NSFC) under grant numbers 60925009, 61221062, 61331008, and the Huawei Research Program under a grant number YBCB2011030.

978-3-9815370-2-4/DATE14/©2014 EDAA

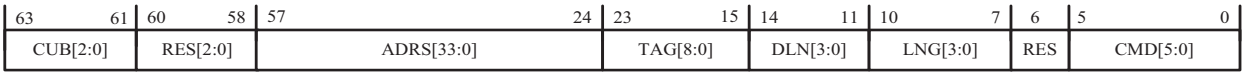

Fig. 1: The detailed request packet header layout in HMC [5], the size of it is 8-byte (or 64-bit).

can support multiple memory access granularity ranging from 16-bytes to 128-bytes. For a packet with the maximum 128-bytes data, the packet overhead (including memory address) is only 12.5%. However, for a packet with the minimum 16-bytes data, the packet overhead will increase up to 100% (as shown in Figure 2), which means that only half of the memory bandwidth is really used for transferring effective data.

To overcome the high packet overhead problem in SPSR, we propose a novel packet interface named **SPMR (Single-Packet-Multiple-Requests)** that supports to encapsulate multiple memory requests into a single packet. Memory requests are first buffered in the request queue of on-chip controller before they are packed. Then it will provide the opportunity for multiple memory requests that access the same destination memory module to be encapsulated into a single **request packet**. The SPMR mechanism is also appropriate for **response packet**. In SPMR, the packet header and packet tail are the same as in SPSR, but they are now shared by multiple requests in a packet. So the packet payload is the aggregation of all the requests (address and data). The SPMR can significantly reduce packet overhead and improve bus efficiency. Furthermore, SPMR provides the opportunity to exploit correlation of memory requests within a packet to reduce packet size, such as address compression, contiguous memory request merging.

Overall, we have made the following contributions:

- We propose a novel packet-interface named Single-Packet-Multiple-Requests to reduce packet overhead by about 53.9% and improve bandwidth efficiency compared with the conventional SPSR, SPMR can also improve system performance by about 63.6%.
- We further propose a self-adaptive compression mechanism for addresses of multiple requests within a packet, and we adopt a simple but efficient packing scheduling algorithm that memory requests with high locality and similarity are preferentially selected to be packed in a packet. The experimental result shows that addresses can be effectively compressed with a ratio of 1.95 on average.
- Finally, aiming at applications with good locality, we propose a mechanism to merging contiguous small granularity requests into a single large granularity request. It can further reduce address-field overhead (especially for read request packets), improve DRAM row buffer locality and improve system performance by 17.0%.

The rest of the paper is organized as follows: Section II introduces the background and summarizes the related work. Section III describes our optimizations on packet-based memory system to reduce packet overhead and improve memory power and bandwidth efficiency. We describes the experimental methodology in Section IV, and demonstrate the experimental results and discussion in Section V. Finally, Section VI gives the conclusion of this paper.

II. Background and Related work

A. Packet-based Interface Memory System

Packet-based interface is considered as a probable trend for future memory system, which has been shown to work well with different memory technologies, such as DDRx DRAM

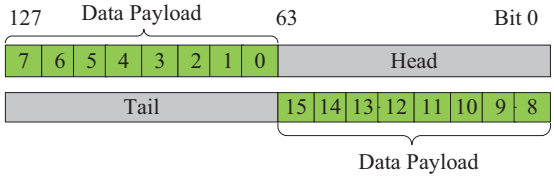

Fig. 2: The packet layout with 16B data payload, in which the packet overhead is 100% (for packet header and tail).

[17], 3D-Stacking DRAM [5], and Phase Change Memory [20]. Within this interface the conventional integrated memory controller is divided into on-chip controller and off-chip controller. The on-chip controller is simplified and decoupled from memory scheduling, thus it has no need to take care of low level device timing constraints any more. After it receives a memory request from CPU core, the request is simply encapsulated into a **request packet** as payload (contains address, request type etc.). Then the **request packet** is relayed to the off-chip controller over a narrower and faster link bus. The off-chip controller is the intermediate logic located between on-chip controller and DRAM memory channels, and it is responsible to schedule memory requests to DRAM devices meanwhile taking care of timing constraints. For a memory read request, after it is served, a **response packet** which contains read-return data will be sent back to the on-chip controller, and finally the data will be returned back to the requested CPU core.

A packet usually contains three parts: packet header, data payload and packet tail. The packet header usually includes command, addressing information, and some other control fields. Figure 1 shows the detailed request packet header layout in HMC [5], and the size of it is fixed 8-byte. The *CUB* and *ADRS* field is addressing information, which indicates HMC identifier (for multi-HMC chain) and request address within an HMC respectively. The *TAG* field represents the tag number which identifies this request uniquely. The *DLN* and *LNG* field represents the packet length in FLITs (1 FLIT is 128 bits). And the *CMD* field indicates the memory command (e.g. read, write) and the access granularity (16B to 128B). The packet tail usually includes flow and link-retry control fields, along with some redundant information (e.g. CRC). Figure 2 shows the packet layout with 16B data payload in HMC, the packet is transferred within two flits, the first flit contains the packet header and the low half of data, while the second flit contains the high half of data and the packet tail. Thus we can see that the packet overhead (packet header and tail) is 100%.

Recently, several works on packet-based interface memory system have been proposed. Buffer-On-Board (BOB) [17] memory system is proposed for regular JEDEC-standardized DDRx DRAM, such as Intel SMB [3], IBM Power 795 memory system [21] and our recently work MIMS [16]. In BOB, A simple controller communicates with a BOB controller over a narrower and faster link bus with packet protocol. Hybrid Memory Cube (HMC) [5] has been proposed to work with 3D-stacking DRAM dies, in which a logic die communicates with the memory controller over multiple 16-lane, full-duplex serialized links (with high bandwidth SerDes I/O interface).

Ham et al. [20] extend packet-based interface memory system and propose disintegrating memory controllers to support heterogeneous command protocols (DRAM and PCM). Fang et al. [18] propose UniMA to enable universal inter-operability between processors and memory modules by adopting a unified communication interface. Udipi et al. [27] propose a novel packet based interface with silicon-photonic and 3D-stacking DRAM memory. Previous commercial product Fully-Buffered-DIMM (FBDIMM) [19] also adopts packet interface among AMBs, which were organized as a daisy chain. However, to the best of our knowledge, all of these previous works adopted SPSR approach, which would result in high packet overhead when working with fine grained memory access.

B. Fine Grained Memory Access

Conventional coarse-grained memory access which always reads or writes a cache block data (e.g. 64-byte) would waste memory power and bandwidth when spatial locality is poor. Thus fine grained memory access which only accesses partial really useful data within a cache block was proposed. AGMS [29] augments the virtual memory interface to configure preferable access granularity for each page based on page access profiling, while DGMS [30] adopts hardware prediction control mechanism to dynamically adapt memory access granularity, and it has shown that DGMS can reduce DRAM power by about 13% and reduce DRAM traffic by about 44%. Both of them adopt sub-ranked memory which groups DRAM devices into multiple independent narrow sub-ranks to support fine grained access. Sub-ranked memory can improve power efficiency and leveraged memory level parallelism. Many works have contributed to it, such as Rambus's Micro-threading [28], Mini-rank [32], Multi-core DIMM [10, 11], Convey's S/G DIMM [15]. Zhang et al. [31] propose Heterogeneous Multi-Channel to balance the performance and power consumption of the DRAM system by grouping physical DRAM devices into logical sub-ranks with different data bus width. Skinflint DRAM system [22] minimizes DRAM write power by selectively only accessing DRAM chips that with really modified sub-block data.

III. Optimizations on Packet-based Memory System

A. SPMR: Single Packet Multiple Requests

As shown in the section I, SPSR approach with fine grained memory access will result in high packet overhead. Thus in this paper we propose a new packet interface SPMR to reduce the transferring size for extra packet-protocol bits. To support encapsulating multiple requests into a single packet, a new adaptive packet header layout is introduced based on HMC's packet header. As shown in Figure 3, a packet header now has a universal header field with multiple *ADDR* and *GRAN* fields. The universal header is the same as shown in Figure 1 which includes *CUB*, *TAG*, *LNG*, *DLN* and *CMD* fields. The *ADDR* field is now no longer dedicated to a single packet, thus it is decoupled from the universal packet header. Instead, multiple *ADDR* fields along with multiple *GRAN* fields are integrated into a packet, and each *ADDR* and *GRAN* represents the memory address and the granularity of each memory request respectively. The packet tail is remained the same as in a SPSR packet, because it has nothing correlated with requests information. It is worth noting that multiple requests that access the same memory module (with a dedicated off-chip controller) and with the same type of operation can be encapsulated into a SPMR packet, and they can share the same memory module identifier (e.g. *CUB*) field and the same *CMD* filed in the universal header.

Thus, the total size of a SPMR packet with N memory requests is:

$$PKT\_SZ\_SPMR(N) = UNI\_HEADER\_SZ + N * (ADDR\_SZ + GRAN\_SZ) \quad (1)$$

Where *PKT_SZ_SPMR(N)* represents the total size of a SPMR packet, *UNI_HEADER_SZ* represents the size of a universal header, *N* represents the number of requests, *ADDR_SZ* and *GRAN_SZ* represent the size of address and granularity field of a memory request respectively.

For comparison, with SPSR, N memory requests need N separate packets, thus the total size is:

$$PKT\_SZ\_SPSR(N) = N * PKT\_SZ = N * UNI\_HEADER\_SZ + N * (ADDR\_SZ + GRAN\_SZ) \quad (2)$$

Where *PKT_SZ* represents the size of a single SPSR packet which contains a universal header and a dedicated memory request (including address and granularity field).

Thus the SPMR with N requests can save up to *(N-1)*UNI_HEADER_SZ* space, and it can reduce packet overhead and improve memory bandwidth efficiency.

SPMR can also work with response packet in the same way, multiple return-data from the same memory module can be selected to encapsulate into a single response packet, with a separate *DATA* and *GRAN* field for each memory request, thus multiple return-data can share the same universal header.

Figure 4 shows the architecture of discrete memory controllers that support SPMR packet interface. There are separate read queue and write queue in the on-chip controller, which are used to buffer read requests and write requests respectively. A packing scheduler is added to select multiple requests with the same destination memory module and the same type in a batch. Then the selected requests are sent to the packet generator and encapsulated into a SPMR request packet there. This is done by generating a corresponding universal header, multiple address and granularity fields for requests and a packet tail. Request packets are sent to the off-chip controller through SerDes module over a serial link bus. The packet decoder in the off-chip controller is enhanced to support SPMR packets: it firstly decodes the universal header to get the size of the packet (which indicates the number of requests) and the request type; then it can retrieve the address and granularity information for each request. All decoded memory requests are put in the request queue and scheduled by the command (CMD) scheduler into DDRx commands. The DDRx commands are finally scheduled to the DDRx interface which supports Sub-ranked memory to access destination DRAM devices.

Decoding a SPMR packet might introduce extra processing latency if it retrieves multiple requests in serial. Two optimizations can be adopted to reduce the decoding latency: 1) multiple requests can be decoded in **parallel**, since the size of the *ADDR* and *GRAN* field is fixed, the offset of each request in a packet can be efficiently calculated in advance; 2) a SPMR packet can be decoded before it is received completely in a pipeline manner: decoding requests from the previous flit can be pipelined with receiving the next flit.

Fig. 3: The new request packet header layout of SPMR with multiple memory requests, each request needs a separate *ADDR* and *GRAN* field.

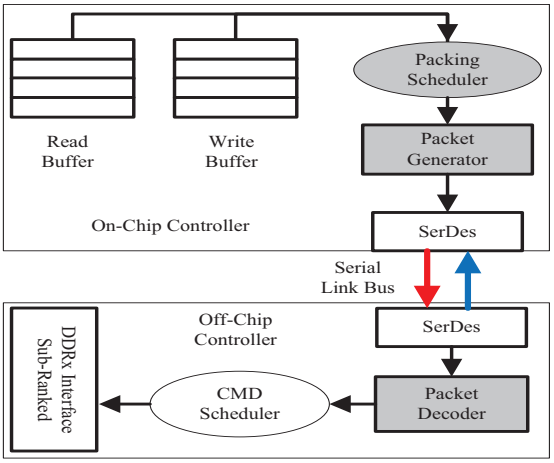

Fig. 4: The architecture of discrete memory controllers that support SPMR packet interface.

B. Address compression

Further optimization focuses on the multiple *ADDR* fields which represent memory addresses of multiple requests in a SPMR packet. Due to the locality of memory address accessed by a process or a thread, high bits of many 48-bits addresses may be the same. We propose an address-compression technology that makes the *ADDR* domain shorter. Our compression only pays attention to the request packet here since the response packet does not have the *ADDR* domain.

As the high bits of addresses could be repeated, our address-compression algorithm uses the format of base address with difference address. The algorithm can be divided into three categories: full-bits base with difference (has symbol bit), high-bits base with difference (has not symbol bit), base number with difference (has symbol bit).

Full-bits base with difference algorithm selects memory requests with similar addresses and put them into one packet, and chooses one of them as the base address. This base address is written into *ADDR* domain with other difference addresses which are calculated from it. The new *ADDR* domain's format is at Figure 5. High-bits base with difference algorithm has the method similarly, just takes only the high-bits of address as the base address and removes the symbol bit of difference addresses to save the bits transferred.

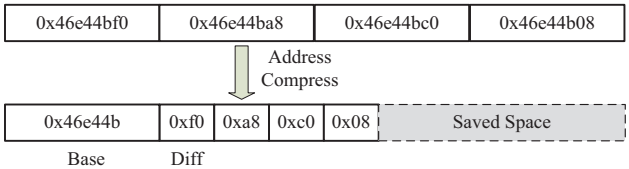

Fig. 5: The new ADRS Field of Base address and Difference address.

Base number with difference algorithm needs base address table in both requester and responder of the bus. A base address table keeps some full-bits addresses in it and one of the items in table is selected as the base address. This algorithm makes each address in the packet has its own base address and each base address is written into *ADDR* domain in the form of base number (this number is its position in the base address table) as shown in Figure 6. The base address table needs update when there is not an item matching the address of a new memory request so that extra transmission is occurred to write a new base address. But if the program has good locality, the base address table does not need update frequently and the tables are set to each thread because that the memory accesses from same thread may have good locality. This algorithm has better effect than the other two owing to saving of bits transferred as a result of the replacement of base number to base address.

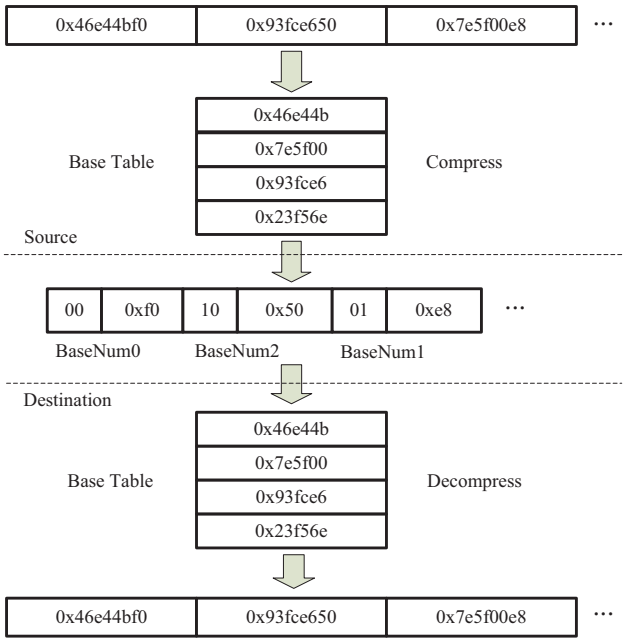

Fig. 6: The ADRS field of Base number and Difference address with synchronous base-address table approach.

To the base number with difference address-compression algorithm above, further improvement is feasible. First, to reduce the frequency of base address table's update, we propose self-adaptive algorithm. Self-adaptive algorithm means that at every time a base address in table is used, this base address is replaced automatically to the real address transferred this time so that when the memory request sequence is increased or decreased progressively, one specific base address is used for this sequence from beginning to end instead of a long sequence uses several base addresses which occupy many items in the base address table. Second is two-level base address table structure. Sometimes we find that the whole base address may not be rewritten entirely, only the low bits are covered. So we divide a base address into two parts of high bits and low bits and put them into first-level table and second-level table separately. To some programs that have not very good locality which bring many base address update, this structure can make the first-level base address table that keeps the high bits of base

ROB	2.7GHz, 256-entry, max fetch/retire per cycle: 4/2
L1 Cache	Private, 32KB, 4-way, 64B-block, 4-cycle hit
L2 Cache	Private, 256KB, 8-way, 64B-block, 10-cycle hit
L3 Cache	Shared, 1MB/core, 16-way, 64B-block, 40-cycle hit
Link Bus	2.7GHz, p2p, read/write bus width: 32/32
DRAM Parameters	
Memory	2 64-bit Channels, 2 Ranks/Channel, 8 devices/Rank, 8 sub-ranks/rank, x8-width (1 device) sub-rank
Device	DDR3-1333MHz, x8, 8 banks, 8 KB Row Buffer, 32768 Rows/bank, 1024 Columns/Row, BL=8, Time/power parameters from Micron SDRAM [1]

TABLE I: System Configurations.

address not been updated as often despite the second-level base address table still update frequently. It will save half the cost than replacing the whole address every time in average.

C. Merging Continuous Memory Requests

Address-compression technology has obvious effect on some applications, because these programs have good locality. Some of these programs not only have locality but also have continuity that program requests for a continuous large memory space in a short time. Because of the limit of cache line, a large memory space access is cut into a batch of 64 bytes memory operations.In a multi-core environment, a batch of continuous requests may be broken into many noncontinuous parts.Memory behavior of program is hidden in this case so that the burden of request scheduling is increased. We propose request-merging technology that increase the upper limit from 64 bytes to 4KB thus the original continuous memory access can be issued in a single request. Requests can be merged in reorder buffer when it finds the continuity among several requests in instruction window or be merged by compiler when compiler discovers several memory requests which have continuous addresses. No matter how the requests are merged, it will need a modification to CPU core. In this paper, we do not discuss this technology in detail and just evaluate its effect.

IV. Evaluation Methodology

In this section, we will introduce our simulator and benchmark using in our evaluation.

A. Simulator and workloads

In our evaluation, we use a memory simulator extended from DRAMSim2[26]. We modify the memory module with serial bus system and packet-based communication to simulate the structure and protocol of HMC. To estimate the performance of fine-grained memory access, we add the function of sub-rank to support the minimum granularity of 8 bytes. We use Pin[24] to collect the memory access traces and input these traces into a cache simulator to get granularity message of each memory access trace. At last, we input all traces with type of cache miss into memory simulator. We choose several multi-thread memory intensive applications from BFS in Graph500[9], PARSEC[14], Listrank[12], Pagerank[4], GUPS[7] and SSCA2[13]. To address compression and continuous requests merger, we also choose some programs with good locality from STREAM[8], perM, SPECCPU2006[2] and NAS[6].

B. System configurations

We test our work in different configurations. The same parts are shown in table I in detail.

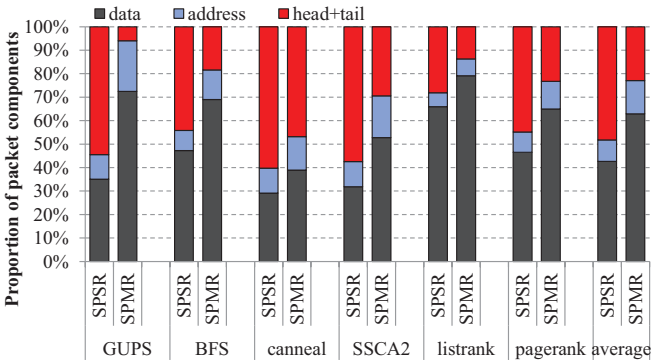

Fig. 7: The proportion of packet components with original SPSR and with our proposed SPMR.

Fig. 8: The performance and memory bandwidth efficiency improvement of combining multiple requests in one packet.

V. Results and Discussion

A. Combine several requests in one packet

As shown in Figure 7, we find that the proportion of each part in packet has large difference before and after putting several memory requests in one packet. The proportion of necessary address and data becomes large as the proportion of packet header and packet tail becomes small. In average, the packet header and tail reduce by 53.9% which means that the efficiency of serial bus is improved. From the result shown in Figure 8, we can see that the improvement of bus system's efficiency brings 63.6% improvement of performance of whole memory system in average. The program with best effect(GUPS) has an improvement of 152% because of the great reduction of packet header and packet tail.

B. Address compression

In table II, we show the result of address compression with the best algorithm introduced above. We use the base address table to save a series of base address and transmit the base address number in packet. Although base address table use extra registers to save the table, this strategy brings the best effect of reducing the bits for transmission. The compress ratio shows this reduction that ADDR field in packet become short from 44.2% to 69.4% as before. Shorter packet length benefits the efficiency of serial bus reflected in bus utilization reduction. In this table, we also find the trend that address compression has better results on programs that have mainly small-grained memory access which infers the proportion of address is large in packet.

	Compress Ratio	Bus Utilization Reduction
streamcluster	1.61	7.93%
STREAM	2.26	3.71%
perM	2.01	4.36%
ScaleParC	2.26	12.25%
SPECCPU/437	1.44	4.93%
SPECCPU/458	2.14	7.46%

TABLE II: The compress ratio and memory bus utilization reduction of addresses compression in SPMR.

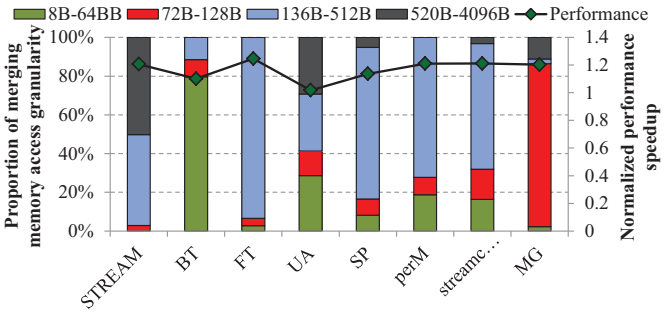

Fig. 9: The memory access granularity and performance improvement after merging continuous memory requests.

C. Merge continuous memory requests

Figure 9 shows the result of memory access granularity and performance improvement after merging continuous requests. We can find that some workloads have really good memory access locality (or continuity) so that a large number of large-grained memory requests are successfully generated after combining the continuous memory requests. The contention on memory controller is reduced by this combination and row buffer utilization is improved. Thus the overall performance of memory system can be improved. In Figure 9, we can find the average improvement is about 17.0%, while the max improvement is about 24.9% for FT workload.

VI. Conclusion

In this paper, we find that previous implementations of packet-based interface memory systems (referred as SPSR) will result in high packet overhead when working with fine-grained memory access, since a single packet is dedicated to a single memory request. Then we propose a novel **SPMR** mechanism that supports to encapsulate multiple memory requests into a packet. Since the packet header and tail are shared by multiple requests, the SPMR can efficiently reduce packet overhead. We also present an address compression mechanism that the shared addresses part among different memory requests is transmitted only once. For workloads with good locality, this compression mechanism can reduce the size of request packets. Finally, we propose another enhancement to merge several memory requests with continuous memory addresses into one memory request with larger granularity. By breaking the constraint of maximum granularity of 64 bytes, the utilization of row buffer on memory chip will be improved. As a result, our works improve the performance of memory system and reduce the energy of serial bus, especially works well on workloads with intensive memory access. SPMR opens new opportunities to exploit more correlations among subsequent memory request other than presented. Further improvement might be achieved by adding semantic information to the packet, which will be our future work.

References

[1] "Ddr3 sdram," http://download.micron.com/pdf/datasheets/dram/ddr3/2Gb_DDR3_SDRAM.pdf, 2006, Micron Technology, Inc.

[2] "Standard performance evaluation corporation," http://www.spec.org/cpu2006/, 2006.

[3] "Intel 7500/7510/7512 Scalable Memory Buffer," http://www.intel.la/content/dam/doc/datasheet/7500-7510-7512-scalable-memory-buffer-datasheet.pdf, 2011, datasheet, Intel Corporation.

[4] "Graphlab: Distributed graph-parallel api," http://docs.graphlab.org/index.html, 2012.

[5] "Hybrid memory cube specification 1.0." Hybrid Memory Cube Consortium, 2012.

[6] "Nas parallel benchmarks," http://www.nas.nasa.gov/publications/npb.html, 2012.

[7] "Randomaccess – gups (giga updates per second)," https://icl.cs.utk.edu/projectsfiles/hpcc/RandomAccess/, 2012.

[8] "Stream: Sustainable memory bandwidth in high performance computers," http://www.cs.virginia.edu/stream/, 2012.

[9] "The graph500 list," http://www.graph500.org/, 2013.

[10] J. H. Ahn *et al.*, "Future scaling of processor-memory interfaces," in *SC 2009*, 2009, pp. 42:1–42:12.

[11] J. H. Ahn *et al.*, "Multicore dimm: an energy efficient memory module with independently controlled drams," *Computer Architecture Letters*, vol. 8, no. 1, pp. 5 –8, jan. 2009.

[12] D. A. Bader, G. Cong, and J. Feo, "On the architectural requirements for efficient execution of graph algorithms," in *ICPP 2005*, pp. 547–556.

[13] D. A. Bader and K. Madduri, "Design and implementation of the hpcs graph analysis benchmark on symmetric multiprocessors," in *HiPC 2005*, pp. 465–476.

[14] C. Bienia *et al.*, "The parsec benchmark suite: characterization and architectural implications," in *PACT 2008*, pp. 72–81.

[15] T. Brewer, "Instruction set innovations for the convey hc-1 computer," *Micro, IEEE*, vol. 30, no. 2, pp. 70 –79, march-april 2010.

[16] L. Chen *et al.*, "Mims: Towards a message interface based memory system," in *Technical Report: arXiv:1301.0051v1 Jan 2013*.

[17] E. Cooper-Balis, P. Rosenfeld, and B. Jacob, "Buffer-on-board memory systems," in *ISCA 2012*, pp. 392–403.

[18] K. Fang *et al.*, "Memory architecture for integrating emerging memory technologies," in *PACT 2011*, pp. 403 –412.

[19] B. Ganesh *et al.*, "Fully-buffered dimm memory architectures: Understanding mechanisms, overheads and scaling," in *HPCA 2007*, pp. 109 –120.

[20] T. J. Ham *et al.*, "Disintegrated control for energy-efficient and heterogeneous memory systems," in *HPCA 2013*, pp. 424–435.

[21] R. Kalla *et al.*, "Power7: Ibm's next-generation server processor," *Micro, IEEE*, vol. 30, no. 2, pp. 7 –15, 2010.

[22] Y. Lee *et al.*, "Skinflint dram system: Minimizing dram chip writes for low power," in *HPCA 2013*, pp. 25–34.

[23] K. Lim *et al.*, "Understanding and designing new server architectures for emerging warehouse-computing environments," in *ISCA 2008*, pp. 315–326.

[24] C.-K. Luk *et al.*, "Pin: building customized program analysis tools with dynamic instrumentation," in *PLDI 2005*, pp. 190–200.

[25] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: eliminating server idle power," in *ASPLOS 2009*, pp. 205–216.

[26] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *Computer Architecture Letters*, vol. 10, no. 1, pp. 16 –19, jan.-june 2011.

[27] A. N. Udipi *et al.*, "Combining memory and a controller with photonics through 3d-stacking to enable scalable and energy-efficient systems," in *ISCA 2011*, pp. 425–436.

[28] F. Ware and C. Hampel, "Improving power and data efficiency with threaded memory modules," in *ICCD 2006*, pp. 417 –424.

[29] D. H. Yoon, M. K. Jeong, and M. Erez, "Adaptive granularity memory systems: a tradeoff between storage efficiency and throughput," in *ISCA 2011*, pp. 295–306.

[30] D. H. Yoon *et al.*, "The dynamic granularity memory system," in *ISCA 2012*, pp. 548–559.

[31] G. Zhang *et al.*, "Heterogeneous multi-channel: fine-grained dram control for both system performance and power efficiency," in *DAC 2012*, pp. 876–881.

[32] H. Zheng *et al.*, "Mini-rank: Adaptive dram architecture for improving memory power efficiency," in *MICRO 2008*, pp. 210–221.