# Improving the DRAM Access Efficiency for Matrix Multiplication on Multicore Accelerators

Sheng Ma, Yang Guo, Shenggang Chen, Libo Huang, Zhiying Wang
State Key Laboratory of High Performance Computing
College of Computer, National University of Defense Technology
Changsha, Hunan, 410073, China
Email: {masheng, guoyang, shgchen, libohuang, zywang}@nudt.edu.cn

***Abstract*—The parallelization of matrix multiplication on multicore accelerators divides a matrix into several partitions. The existing design deploys an independent DMA transfer for each core to access its own partition from DRAM. This design has poor memory access efficiency, since memory access streams of multiple concurrent DMA transfers interfere with each other. We propose Distributed-DMA (D-DMA), which invokes one transfer to serve all cores. D-DMA accesses data in a row-major manner to efficiently exploit inter-partition locality to improve the DRAM access efficiency. Compared with a baseline design, D-DMA improves the bandwidth by 84.8% and reduces DRAM energy consumption by 43.1% for micro-benchmarks. It achieves higher performance for the GEMM benchmark. With much lower hardware cost, D-DMA significantly outperforms an out-of-order memory controller.**

## I. Introduction

To boost performance and energy efficiency, computer architects have designed numerous scientific computing accelerators, such as IBM CELL [1], Intel Xeon Phi [2] and GPGPUs [3]. An essential issue for these accelerators is providing efficient matrix multiplication implementations, since matrix multiplication forms the core of many crucial algorithms, including solvers of linear systems, least square problems, and singular and eigenvalue computations [4], [5].

Currently, the most widely used and highest effective implementation is the GotoBLAS approach [4]. This approach divides a matrix into several data partitions for parallelization across multiple cores [5]. Each core accesses its own data partitions from DRAM. To free processing units from data movements, many accelerators, such as IBM CELL [1], digital signal processors (DSPs) [6] and some GPGPUs [7], access data with a direct memory access (DMA) controller; each core invokes a DMA transfer to move its own data partitions [8], [9]. This design has poor memory access efficiency, since memory access streams of multiple concurrent DMA transfers interfere with each other and deteriorate DRAM page hit rates. Out-of-order memory controllers [10], [11] may recover some of the lost page access locality, but their effectiveness is constrained by the limited scheduling buffer size and the arrival interval (often large) of requests.

This work is supported by the National Natural Science Foundation of China (No. 61672526, 61572508, 61872374, 61472435), Research Project of NUDT (ZK17-03-06) and Science and Technology Innovation project of Hunan Province (2018RS3083).

Instead of recovering locality at memory controllers, we exploit locality with the DMA controller, inspired by observing a favorable feature for DRAM addresses of data partitions. Since data partitions are divided from one matrix, the same row of adjacent partitions has continuous addresses, and there is significant inter-partition locality. We propose a novel design, Distributed-DMA (D-DMA), to exploit inter-partition locality to improve the DRAM access efficiency. Unlike the traditional design which invokes one DMA transfer for each core, D-DMA invokes one transfer to serve all cores. It accesses data in a row-major manner; it first moves the first row of multiple partitions and then moves the second row of multiple partitions, and so on. Since the same row of adjacent partitions has continuous addresses, this significantly improves the DRAM access efficiency.

Experimental results show that, compared with a baseline design, D-DMA achieves an 84.8% higher bandwidth and reduces the DRAM energy consumption by 43.1% for micro-benchmarks. It provides much higher fairness than the baseline design. Its average GEMM performance gain is 31.6%. With much lower hardware cost, D-DMA outperforms an out-of-order memory controller. In summary, we make following contributions:

- Observe that the traditional DMA design for matrix transposition has poor memory access efficiency.
- Propose D-DMA to efficiently exploit the significant inter-partition locality in matrix multiplication.
- Compared with existing designs, D-DMA significantly improves the performance and energy-efficiency.

The rest of this paper is organized as follows. Section II first describes the research background, including the accelerator architecture and the data movement in matrix multiplication, and then presents the research motivation. Section III describes the proposed Distributed-DMA. Section IV conducts the evaluation. Section V discusses the related work, and Section VI concludes this paper.

## II. Background and Motivation

Here, we first describe the architecture of the M2G accelerator, which is leveraged as an example platform to demonstrate the proposed D-DMA. Then, we analyze the data movement in matrix multiplication and present the research motivation.

978-3-9819263-2-3/DATE19/©2019 EDAA

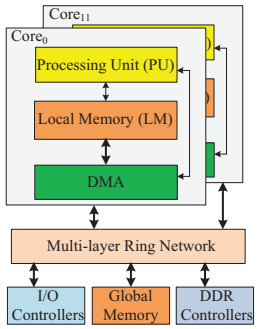



Fig. 1: The architecture of M2G accelerator.

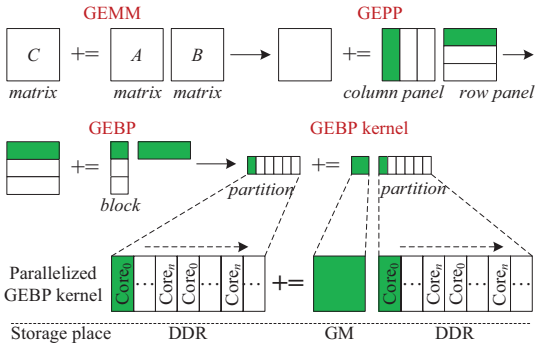



Fig. 2: The GotoBLAS approach and its parallelization.

### A. Accelerator Architecture

The M2G accelerator is a prototype accelerator explored for efficiently realizing high performance scientific computing. Its architecture is inspired by the idea of leveraging energy efficient DSPs for scientific computing [8]. As shown in Fig. 1, the architecture of M2G has some similarities to existing DSPs [6], [12] and accelerators [1]. It has 12 cores, which are connected with a multi-layer ring network-on-chip (NoC). The bit width of the ring NoC is 512 bits to boost the throughput. The core includes a processing unit (PU) to perform scalar control tasks and vector computations. Each PU integrates 50 double precision multiply accumulators (MACs) to offer high performance. Each core integrates a 768 KB local memory (LM) to store computation operands. A DMA controller is deployed for each core to move data among different memories. The accelerator deploys a 4 MB on-chip distributed global memory (GM). The GM can be also configured as a shared cache. The accelerator integrates four 64-bit DDR3-1600 controllers and achieves a 51.2 GB/s peak memory bandwidth. It also integrates several I/O controllers, including the PCIe, SRIO and etc.

### B. Data Movement in Matrix Multiplication

The general matrix multiplication (GEMM) computes $C$+=$AB$ for three matrices. Modern GEMM implementations apply the GotoBLAS approach [4] shown in Fig. 2. It consists of multiple nested loops to amortize data movements with computations. First, matrices $A$ and $B$ are divided into several column panels and row panels, respectively. GEMM is decomposed into several general panel-panel multiplications (GEPP). Next, the column panel of $A$ is divided into several blocks; GEPP is decomposed into multiple general block-panel multiplications (GEBP). Finally, the row panels of $B$ and $C$ are divided into several partitions; GEBP is broken into several GEBP-kernels. Multicore accelerators parallelize the GotoBLAS approach to fully use multiple cores. The GEBP kernel level is appropriate for parallelization since a panel typically consists of thousands of partitions. As shown in Fig. 2, many accelerators parallelize this level by periodically assigning partitions to different cores [5], [8], [13]. Our M2G accelerator parallelizes this level in a similar manner.

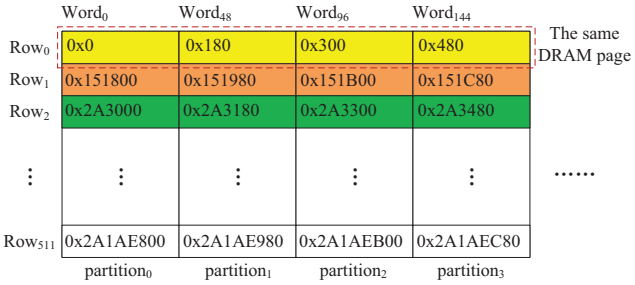
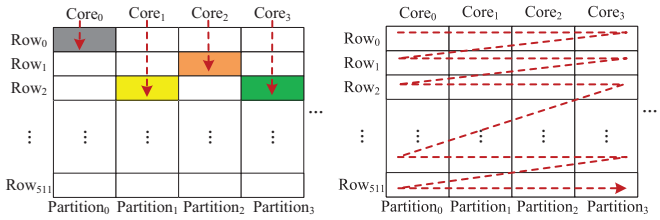



Fig. 3: The DRAM Addresses of data partitions. The panel matrix has 3600 partitions. The partition size is 512×48. Each box has 48 words. The number in each box is the starting address of these words.

The parallelized computation consists of several epochs. During each epoch, each core accesses one of its allocated partitions from DRAM. To free processing units from data movements, several accelerators use DMA controllers to access data partitions [1], [7], [8]. The data access procedure must be optimized to improve performance. We analyze DRAM addresses for data partitions. Since these partitions are divided from one matrix, their DRAM addresses have a special feature: the addresses of the same row of adjacent partitions are continuous, while the addresses of adjacent rows are separated by a large gap since there are thousands of partitions. Use the M2G accelerator as an example. Based on its architecture, its partition size is set as 512×48; a partition has 512 rows and each row has 48 words. Fig. 3 shows the addresses of several data partitions. The same row of adjacent partitions has continuous addresses and is in the same DRAM page, while different rows are in different DRAM pages.

In traditional designs, each core uses a DMA transfer to access its own partition for one computation epoch. Since multiple cores have different status and their distances to memory controllers are different, it is quite possible that they access different rows simultaneously and interfere with each other. For example, in Fig. 4a, DMA transfers of $Core_0$, $Core_1$, $Core_2$ and $Core_3$ access $Row_0$, $Row_2$, $Row_1$ and $Row_2$, simultaneously. These rows are in different DRAM pages, which results in poor DRAM page hit rate and bandwidth.

Since the same row of adjacent partitions is in the same




(a) Traditional design. (b) Distributed-DMA.

Fig. 4: The data transfer of traditional design and Distributed-DMA. This figure uses 4 partitions as an example.

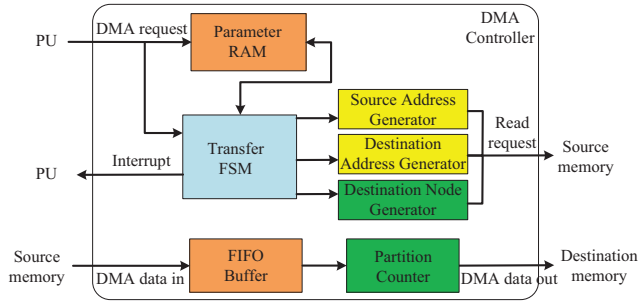



Fig. 5: The structure of a D-DMA controller.

DRAM page, there is significant inter-partition locality. We propose Distributed-DMA (D-DMA) to exploit inter-partition locality. As shown in Fig. 4b, during each computation epoch, D-DMA invokes one transfer to serve all cores. It accesses data in a row-major manner; it first moves the first row of multiple partitions belonging to the current computation epoch, and then moves the second row of these partitions, and so on. Since the same row of adjacent partitions is in the same DRAM page, this strongly improves the DRAM access efficiency.

## III. Distributed-DMA

Here, we first introduce the structure of a D-DMA controller. Then, we describe the transfer procedure of D-DMA.

### A. The Structure of a D-DMA Controller

The D-DMA controller is modified based on a canonical DMA controller [6], [12]. Fig. 5 shows its structure. The DMA controller receives transfer parameters from the PU, and stores them in the parameter RAM. These parameters typically include the initial source and destination addresses, the row and column count of data array.

The DMA controller consists of two parts. The first part sends read requests. It includes the transfer FSM, source and destination address generators. The transfer FSM controls the overall status. Once a DMA transfer finishes, it sends out an interrupt. The source and destination address generators calculate the current source and destination addresses, respectively. To allow out-of-order arrival of data replies, the read request carries both the current source and destination addresses. The second part processes data replies. It configures a FIFO buffer to synchronize the transfer between source and destination

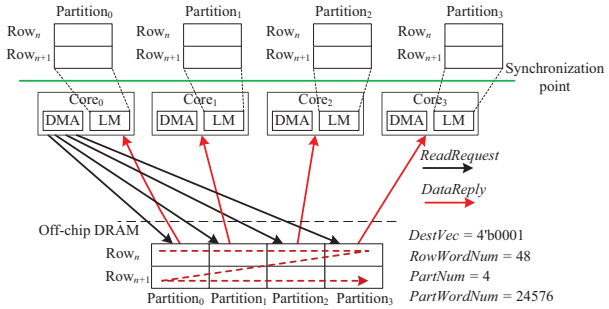



Fig. 6: The D-DMA transfer with 4 cores as an example.

memories. The received data reply is first stored into the FIFO buffer, and then sent out to the destination memory. The baseline DMA controller supports several types of point-to-point transfers, such as moving data from DRAM into the LM of a core or moving data from the LM of a core into DRAM. To support D-DMA, two modules are added. Since D-DMA moves data to the LM of several cores, a destination node generator is added to calculate the destination node for each read request. The other module is the partition counter, which counts the received data replies for a D-DMA transfer. We describe the working procedure of these two modules in further detail in Section III-B.

### B. The Transfer Procedure of D-DMA

As analyzed in Section II-B, to improve the DRAM access efficiency, D-DMA invokes one transfer to serve multiple nodes. These nodes are classified into two types. The master node invokes the transfer and sends read requests to access data for all nodes. The slave node receives and counts its own data replies. In the example shown in Fig. 6, the DMA controller of $\text{Core}_0$ is the master. It sends read requests to DRAM, and distributes data replies to all nodes. The DMA controllers of $\text{Core}_1$, $\text{Core}_2$ and $\text{Core}_3$ are slaves; they receive data of their partitions. The master sequentially accesses different rows of the partitions belonging to one computation epoch; it first reads $\text{Row}_n$ of the four partitions from DRAM, and then reads $\text{Row}_{n+1}$ of these partitions. Since the same row of these partitions is in the same DRAM page, D-DMA efficiently improves the DRAM access efficiency.

D-DMA induces additional transfer parameters. Three parameters, *DestVec*, *RowWordNum* and *PartNum*, are used by the destination node generator to calculate destination nodes. The *DestVec* uses one-hot encoding to indicate the destination node for the data reply of the current read request; the NoC will deliver the reply to the node encoded by the *DestVec*. The *RowWordNum* is the number of words for a row of a partition. When the number of words moved to the current destination equals to the *RowWordNum*, the *DestVec* shifts one bit to the left to denote the next destination. The *PartNum* is the number of partitions. When the shift count of *DestVec* equals to the *PartNum*, the *DestVec* is reloaded to its original value to read the next row from DRAM. Another parameter, *PartWordNum*, describes the number of words of each partition. The partition

counter counts received words. Once it equals to *PartWordNum*, the DMA controller sends out an interrupt.

Fig. 6 shows parameters for a D-DMA transfer in the M2G accelerator. The *DestVec* indicates that the current destination is $\text{Core}_0$. The *RowWordNum* denotes that each row of a partition has 48 words. After reading the first 48 words of $\text{Row}_n$ from DRAM to $\text{Core}_0$, the *DestVec* is shifted one bit to the left to indicate that the destination of the next 48 words is $\text{Core}_1$. This procedure continues until the last 48 words of $\text{Row}_n$ are moved to $\text{Core}_3$. Then, the *DestVec* is reloaded to its original value as 4'b0001; the master continually reads $\text{Row}_{n+1}$. The *PartWordNum* sets the number of words in one partition as 24576 (512×48). Once the number of received words equals to 24576, an interrupt is sent out.

All nodes need to be synchronized before configuring parameters for a D-DMA transfer. We analyze the synchronization overhead. There is one D-DMA transfer with one synchronization operation for each computation epoch. During each epoch, one core finishes the computation of one data partition; the length of one epoch depends on the computation latency of one data partition. This latency is typically hundreds of thousands of clock cycles. For example, it is more than 260,000 clock cycles in the M2G accelerator. There is one synchronization operation during such a long period of time; thus, the synchronization overhead is negligible.

## IV. Evaluation

We implement the proposed D-DMA with synthesizable RTL Verilog and integrate it into the M2G accelerator. We evaluate designs with the Cadence NC-Verilog simulator on the RTL simulation environment of M2G accelerator. We experiment with micro-benchmarks and GEMM benchmarks. The micro-benchmarks move data from DRAM to the LM. It includes the baseline evaluation and sensitivity studies. The baseline evaluation is conducted with 12 cores, and each core accesses a 512×48 partition from the DRAM to its LM. The address layout of adjacent partitions is similar to that shown in Fig. 3. The sensitivities study varies the core count and partition size for further insights. The evaluation with GEMM benchmark is conducted on the fabricated accelerator chip to reduce the running time. We experiment with an in-order and an out-of-order memory access scheduler. The in-order scheduler executes column access commands according to the arrival order of memory requests. To improve performance, it allows out-of-order execution of precharge and activate commands once their timing constraints are satisfied. The out-of-order memory scheduler is the most representative FR-FCFS scheduler [10]. We leverage the Micron DDR3 power calculator [14] to estimate the DRAM power consumption, including the background power, RD/WR/Term power and activate power. The background power includes the standby power and refresh power. The RD/WR/Term power includes the power of read and write commands and the I/O termination power. The activate power includes the power of activate and precharge commands. The DRAM parameters are fetched from the Micron MT41J512M8 DRAM chip [15].

### A. Baseline Evaluation on Micro-benchmarks

Fig. 7 shows evaluation results for micro-benchmarks. P2P is the traditional design which uses point-to-point transfers with an in-order memory scheduler. P2P&OM is point-to-point transfers with an out-of-order memory scheduler. D-DMA is D-DMA with an in-order memory scheduler. D-DMA&OM is D-DMA with an out-of-order memory scheduler.

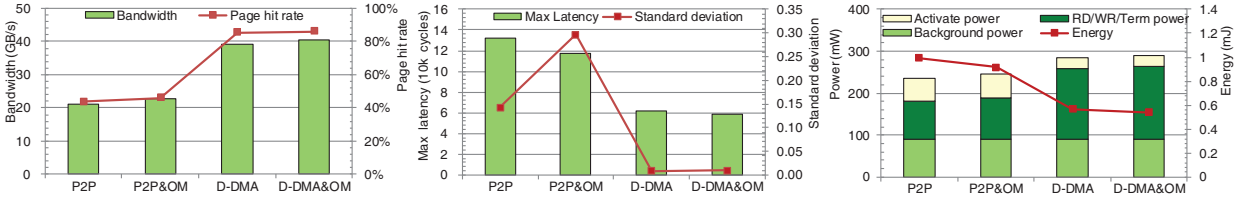
Fig. 7a shows the total bandwidth of all cores and the average page hit rate of all memory controllers. D-DMA and D-DMA&OM see bandwidth gains against P2P and P2P&OM. The gains of D-DMA and D-DMA&OM against P2P are 84.8% and 91.0%, respectively. The page hit rate improvements bring these gains. D-DMA efficiently exploits inter-partition locality and improves the page hit rate from 43.7% for P2P to 85.3% for D-DMA and 86.1% for D-DMA&OM. P2P&OM and D-DMA&OM get higher bandwidths than P2P and D-DMA, respectively. These gains benefit from out-of-order schedulers, which prioritize page hit requests and improve the bandwidth. Yet, the gains are much lower than these of D-DMA. The gain of P2P&OM vs. P2P is only 8.5%, while that is 84.8% for D-DMA vs. P2P. The effectiveness of out-of-order scheduler is constrained by limited scheduling buffers and arrival intervals of requests.

The execution time of parallel task depends on the maximum latency of all cores. The variation of latencies of different cores reflects the fairness provided by designs. Fig. 7b shows the maximum value and standard deviation of latencies of all cores. D-DMA achieves lower maximum latencies than P2P. Compared with bandwidth gains, maximum latency gains of D-DMA are larger. For example, the bandwidth gain of D-DMA vs. P2P is 84.8%, while the maximum latency gain is 112.6%. Since different cores use independent point-to-point transfers, P2P latencies exhibit a higher variation and raise the maximum latency. P2P's standard deviation is 16.5 and 13.8 times higher than D-DMA and D-DMA&OM, respectively. P2P provides much poorer fairness. Both P2P&OM and D-DMA&OM have larger standard deviations than P2P and D-DMA, respectively. The out-of-order scheduler unfairly penalizes the requests from some cores.

As shown in Fig. 7c, D-DMA induces more DRAM power consumption than P2P. D-DMA improves the bandwidth, and executes more DRAM read commands and increases the RD/WR/Term power. Yet, D-DMA improves page hit rates; it executes less DRAM activate/precharge commands and reduces the activate power. These factors cause D-DMA consumes 21.0% higher power than P2P. Similarly, P2P&OM and D-DMA&OM consume more power than P2P and D-DMA, respectively. D-DMA's energy efficiency is significant since it reduces transfer latencies. D-DMA's energy is 43.1% lower than P2P. Out-of-order schedulers also improve the energy efficiency. P2P&OM's energy is 7.6% lower than P2P.

### B. Sensitivity Study

Individual system or application may have different core counts and partition sizes from the baseline. We explore vari-




(a) Bandwidth and page hit rate. (b) Maximum latency and variation. (c) Power and energy.

Fig. 7: The evaluation results of micro-benchmarks.

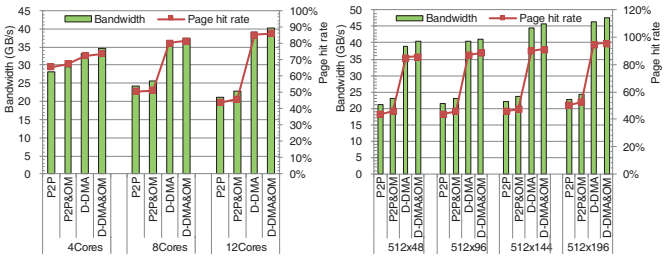
ations for further insights. Except for the analyzed parameter, other parameters are the same as the baseline.

**Core counts.** Different systems may configure different numbers of cores. Fig. 8a evaluates architectures with four and eight cores. The page hit rate and bandwidth of D-DMA increase with more cores. More cores transfer more partitions; this increases the amount of data provided by one DRAM page, and raises the available locality in the address layout. D-DMA efficiently exploits locality. In contrast, P2P cannot effectively exploit locality; its page hit rate and bandwidth decrease with more cores. P2P is limited by the interference among multiple concurrent transfers; this interference exacerbates with more cores. The efficiency of out-of-order scheduler slightly increases with more cores. The bandwidth of P2P&OM is 5.3% higher than P2P with four cores. With 12 cores, this gain is 8.5%. More cores deteriorate the interference, and offer more opportunities for out-of-order schedulers to optimize the performance. Since D-DMA performs better and P2P performs worse with more cores, the gap between them increases with more cores. For example, with four cores, the bandwidth of D-DMA is 18.4% higher than P2P. This gap increases to 84.8% with 12 cores.

**Column counts.** The partition size is decided based on the GEMM implementation in a particular architecture. Other applications or architectures may choose different partition sizes. Fig. 8b changes the column count of the partition. The page hit rate and bandwidth of P2P and D-DMA increase with larger column counts. Mores columns increase the amount of data accessed from one DRAM page, and improve available locality. Higher available locality improves the performance of all designs. D-DMA more efficiently exploits locality; its bandwidth gain is more significant than P2P. When the column count increases from 48 to 196, D-DMA's bandwidth is improved by 19.2%, while P2P's bandwidth is improved by only 4.2%. P2P is strongly limited by the interference among multiple concurrent DMA transfers. The performance gap between D-DMA and P2P increases with larger column counts. When the column count is 48, the bandwidth of D-DMA is 84.8% higher than P2P. When the column count is 196, this gain is 110.2%.

### C. The GEMM Evaluation

We evaluate performance with the GEMM benchmark. We parallelize the GEBP kernel level as analyzed in Section II-B. Before the computation, the source operands are fetched from




(a) Core count. (b) Partition size.

Fig. 8: Sensitivity studies on the core count and partition size.

the off-chip DRAM into the on-chip LM. After the computation, the results are stored from the on-chip LM into the off-chip DRAM. Since the DMA controllers work independently of computation units, we divide the LM into two parts to apply the ping-pong scheme for all designs. When one part of the LM performs computations, the other one performs data movements. During the computation of the current GEBP kernel, the updated $C$ partition of the last GEBP kernel is stored into the DRAM and the original $B$ and $C$ partitions of the next GEBP kernel are fetched from the DRAM. We set the size of $A$ matrix as 512×512. The row counts of $B$ and $C$ matrices are 512. We vary column sizes for $B$ and $C$ matrices to include 3600, 4800, 6000 and 7200 partitions.

Fig. 9 shows the speedup normalized to P2P. The ping-pong scheme make the execution time of one computation epoch depend on the larger of the computation latency and data movement latency. For all configurations, the movement latencies of P2P and P2P&OM are larger than computation latencies. Their performances are decided by data movements. P2P&OM optimizes memory access procedures by prioritizing ready requests. It outperforms P2P for all configurations. P2P&OM performs 8.7%, 6.7%, 8.7% and 7.4% better than P2P when the partition count of $B/C$ matrices is 3600, 4800, 6000 and 7200, respectively.

For all configurations, the data movement latencies of D-DMA and D-DMA&OM are less than the computation latencies. D-DMA and D-DMA&OM fully hide the data movements behind the computations; this causes them to have similar performance, even if D-DMA&OM reduces data movement latencies. D-DMA and D-DMA&OM outperform P2P and P2P&OM. The average performance improvement of D-DMA against P2P is 31.6% for all partition sizes. Since D-DMA's GEMM performance is decided by computation

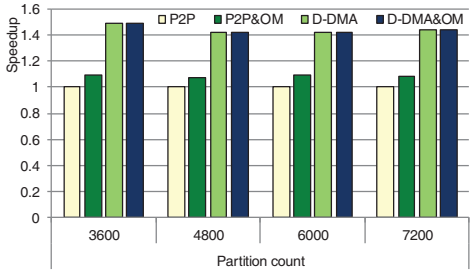



Fig. 9: The evaluation results of GEMM benchmarks.

latencies, its GEMM performance gain is less than its movement latency gain. For example, when the partition count is 3600, D-DMA's movement latency is 53.6% less than P2P, and its GEMM performance is 32.5% higher than P2P. Yet, D-DMA's optimization of data movements allows the integration of more powerful computation resources to further improve the performance. For example, more MAC units can be deployed to fully utilize the bandwidth advantage of D-DMA.

### D. Hardware Overheads

We synthesize the designs by Synopsys Design Compiler with a commercial 40 nm standard cell library. We analyze the area of the additional hardware for supporting D-DMA and out-of-order memory access scheduler. The area of the additional hardware for D-DMA is 23,586 $um^2$, which is less than 0.01% of the total area of the accelerator. The area of the additional hardware for an out-of-order memory scheduler is 122,905 $um^2$; this is more than five times of the D-DMA. The majority cost of out-of-order memory access scheduler comes from its scheduling buffers and logics.

## V. Related Work

Here, we review related work for DMA controller designs and matrix multiplication implementations.

**DMA controllers.** Several processors apply DMA controllers to move data between the off-chip and on-chip memory, including the CELL processor [1] and DSPs [6]. The DMA controllers of all these processors only support point-to-point transfers; each transfer serves only one core. To the best of our knowledge, D-DMA is first one to use one DMA transfer to serve multiple cores to improve the DRAM access efficiency. Recent research applies the DMA into GPUs [7]. Our D-DMA can work together with these designs. Some DMA controllers are used to move data between DRAM and I/O interfaces. The I/O traffic from these DMA controllers competes with CPU requests for the DRAM bandwidth. The Decoupled DMA isolates CPU requests and I/O traffic with a dual-data-port DRAM [16] to improve the performance for CPU requests and DMA transfers. Our research is different as the DMA transfer in our accelerator is for computation units instead of I/O interfaces. We optimize the DRAM access efficiency for DMA transfers without modifying the DRAM structure.

**GEMM implementations.** The matrix multiplication is one of the most important operations in scientific computing. Agarwal et al. [17] implement matrix multiplication with the double buffering scheme to overlap communication and computation. Cannon algorithm [18] is memory efficient for toroidal interconnected systems. The GotoBLAS approach [4] is a widely used GEMM implementation. Several libraries are designed based on this approach [19]. Smith et al. [5] parallelize this approach in many-threaded architectures by systemically exploit the concurrency available in several computation layers. The GEMM implementations on several processors, including Intel Xeon Phi [13], IBM PowerPC A2 [5], and DSPs [8] follow the GotoBLAS approach.

## VI. Conclusion

Optimizing the DRAM access efficiency is critical for matrix multiplication. We propose D-DMA to efficiently exploit inter-partition locality in matrix multiplication. Compared with the baseline design, D-DMA achieves much higher bandwidths and significantly reduces the DRAM energy consumption. It also provides much higher fairness. It improves the performance for the GEMM benchmarks. With much lower hardware cost, D-DMA performs significantly better than an out-of-order memory controller. Since the GEMM is critical for deep learning applications, future work will extend D-DMA into machine learning accelerators.

## References

[1] J. Kahle *et al.*, "Introduction to the cell multiprocessor," *IBM Journal of Research and Development*, vol. 49, no. 4.5, pp. 589–604, 2005.
[2] A. Sodani *et al.*, "Knights Landing: Second-Generation Intel Xeon Phi Product," *IEEE Micro*, vol. 36, no. 2, pp. 34–46, 2016.
[3] E. Lindholm *et al.*, "NVIDIA Tesla: A Unified Graphics and Computing Architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, 2008.
[4] K. Goto and R. Geijn, "Anatomy of high-performance matrix multiplication," *ACM Trans. Math. Softw.*, vol. 34, no. 3, pp. 1–25, 2008.
[5] T. M. Smith *et al.*, "Anatomy of High-Performance Many-Threaded Matrix Multiplication," in *IPDPS*, 2014.
[6] Texas Instruments (TI), "TMS320C6678 Multicore Fixed and Floating-Point Digital Signal Processor," Tech. Rep., March 2014.
[7] D. Jamshidi *et al.*, "D2MA: accelerating coarse-grained data transfer for GPUs," in *PACT*, 2014.
[8] F. D. Igual *et al.*, "Unleashing the high-performance and low-power of multi-core DSPs for general-purpose HPC," in *SC*, 2012.
[9] T. Chen *et al.*, "Optimizing the Use of Static Buffers for DMA on a CELL Chip," in *LCPC*, 2006.
[10] S. Rixner *et al.*, "Memory Access Scheduling," in *ISCA*, 2000.
[11] H. Usui *et al.*, "DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 4.
[12] M. Woh, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "Anysp: Anytime anywhere anyway signal processing," *IEEE Micro*, vol. 30, no. 1, pp. 81–91, Jan 2010.
[13] A. Heinecke *et al.*, "Design and Implementation of the Linpack Benchmark for Single and Multi-node Systems Based on Intel Xeon Phi Coprocessor," in *IPDPS*, 2013.
[14] Micron, "Calculating Memory System Power For DDR3," 2007.
[15] ——, "4Gb: x4, x8, x16 DDR3 SDRAM," 2014.
[16] D. Lee *et al.*, "Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM," in *PACT*, 2015.
[17] R. Agarwal *et al.*, "A high-performance matrix-multiplication algorithm on a distributed-memory parallel computer, using overlapped communication," *IBM J. Res. Dev.*, vol. 38, no. 6, pp. 673–681, Nov 1994.
[18] H.-J. Lee, J. P. Robertson, and J. A. B. Fortes, "Generalized Cannon's Algorithm for Parallel Matrix Multiplication," in *ICS*, 1997, pp. 44–51.
[19] F. G. Van Zee and R. A. van de Geijn, "BLIS: A Framework for Rapidly Instantiating BLAS Functionality," *ACM Trans. Math. Softw.*, vol. 41, no. 3, pp. 14:1–14:33, Jun 2015.