Future Irregular Computing with Memory Accelerators

Noboru Tanabe
Toshiba Corporation
Kawasaki, Japan
noboru.tanabe@toshiba.co.jp

Junko Kogou, Sonoko Tomimori, Masami Takata, Kazuki Joe
Nara Women's University
Nara, Japan
{vega, tomimori-sonoko0826, takata, joe}@ics.nara-wu.ac.jp

***Abstract*—Effective memory bandwidth for irregular applications such as a CG (Conjugate Gradient) solver and graph processing for larger sparse matrices must be accelerated with lower power in the future. Since the total performance of such HPC (High Performance Computing) applications is limited by memory bandwidth, smart memory is a possible lower power accelerator than GPUs (Graphics Processing Units). In this paper, we propose a GPU based HPC system using memory accelerators with gather functions and a HMC (Hybrid Memory Cube) interface. We implemented CG solver for it. The memory accelerator converts indirect accesses, which are unsuitable for cache and device memory, into direct accesses using gather functions. This paper presents the performance of the proposed memory architecture with University of Florida Sparse Matrix Collection. The result shows 1.01 to 1.20 times acceleration by the memory accelerator against the texture cache, even in the case of small matrices that take advantage of texture cache effects. The ratio will dramatically increase when the gap of the cache capacity and the matrices size increases. The scalability of the proposed method is guaranteed by the scalable broadcast thorough interconnection network.**

Keywords-high performance computing; memory architecture; smart memory; irregular processing; CG solver

I. INTRODUCTION

The computation ability of vector processor based supercomputers can be substituted with COTS (Commercial Off-The-Shelf) CPUs or GPUs in many cases. The computation ability of a high-end GPU reaches to one Tera FLOPS (FLoating point Operation Per Second) to be widely used for various applications known as GPGPU (General Purpose computing on Graphics Processing Units). The peak performance of GPUs seems to continuously and steadily increase in FLOPS according to the Moore's law [1]. If such performance progress is not given with sufficient device memory bandwidth for the peak performance, the memory wall problem gets more serious year by year. Hierarchical memory systems, typified by cache memory, do not address the problem when data reusability is small. However, in the case of some applications such as solving a system of linear equations consisting of kernels of SpMV (sparse matrix-vector multiplication), data reusability cannot be exploited so much. When matrices are small, such as a benchmark collection for a sparse matrix [2], the problem is not critical because GPU's cache works well for the small size matrices. The larger the target sparse matrices are, the more frequent and inefficient accesses to the external memory the cache needs to make, thus degrading performance. When large enough sparse matrices are to get regular accesses, most accesses can be converted to coalesced accesses of GPUs so that each cache miss takes a cache line with possible data to be accessed later. On the other hand, when the same sparse matrices are to get irregular accesses because the cache line size or the shortest burst length of GDDR5(Graphics Double Data Rate 5) memory is 128 bytes, it is reported that a cache miss exhausts the memory bandwidth 16 times (double precision) or 32 times (single precision) [3]. The line size of the last level cache (L2 cache) on a new generation GPU is larger than that of texture cache on older GPUs. Therefore, effective bandwidth degradation for huge SpMV (i.e. in the situation with low cache hit rates) of the newer GPU will be larger than that of the older GPUs.

To solve the above problem, an extended large capacity functional memory (memory accelerator) system with PCI (Peripheral Component Interconnect) express based interface and a set of scalable SpMV algorithms for GPUs are proposed in [3]. Although the experiment results of the SpMV algorithms and the functional memory system show four times performance improvement at a maximum, the contribution of the algorithms and the functional memory for the performance improvement is not described separately. In the meantime, they show that the bottleneck of the proposed algorithms and the functional memory lies in the PCI express.

The main contributions of this paper are summarized below:

- We propose new interfaces for the functional memory on a future GPU cluster to avoid the PCI express bottleneck, where Hybrid Memory Cube ports are promising.
- We analyze the relation between the cache hit rate and the matrix size for an SpMV executed on two kinds of GPUs. A tendency of hit rate degradation of texture cache and L1 cache is observed when row vector size increases.
- We implement a CG (Conjugate Gradient) solver including SpMVs for the proposed memory system to evaluate the performance improvement against a cache based system. Since we use the same algorithms for the evaluation, the improvement of memory system is estimated separately.
- During the execution of the CG solver on the proposed memory system, we get its breakdowns to detect hidden problems.

 ISBN: 978-1-61208-272-1

- To solve the above new problems, we improve the implementation based on CUBLAS, which is a numerical calculation library by Nvidia.

The rest of this paper is organized as follows: In Section 2, we introduce related works. In Section 3, we explain the architecture of our proposed memory system. In Section 4, the target workload (i.e. CG solver) on the proposed architecture is explained. We present the performance evaluation results in Section 5. In Section 6, we conclude the work and present future considerations.

II. RELATED WORKS

Recently, there are many research results that SpMV on GPU is accelerated to take the advantage of larger memory bandwidth of GPU rather than CPU [3]-[12]. So far, some sparse matrix libraries are already open to the public, such as in Nvidia [13][14]. While most studies are about storage formats for sparse matrix [3]-[12], some studies make use of GPU's texture cache for high-speed access to column vectors of the sparse matrix [4]-[12]. Although GPU's texture memory is read-only memory from the GPU, it can be used for storing column vectors, which are reusable, by the GPU with dedicated functions such as tex1Dfetch and tex2D to access the column vectors. Since the texture memory is cached on the texture cache, accesses to the device memory would be reduced if appropriate disposition of non-zero elements of sparse matrices are given.

Latest GPUs (e.g., Fermi architecture GPUs in the case of Nvidia) contain general purpose L1 and L2 caches for global memory on the device memory as well as texture memory. Using such general purpose caches, the column vectors can be cached without any dedicated functions to reduce device memory accesses.

Among many studies for accelerating SpMV, there are very few studies for accessing huge sparse matrices from many GPUs. In such studies, the huge sparse matrices should be decomposed for each device memory on the GPUs. Unless smart decomposition methods are used, considerable numbers of fine grain random communications, which degrade the scalability, are generated. In [5], the reduction of inter-GPU communication by hyper graph partitioning is reported, but the efficiency heavily depends on the matrix shape and/or the number of GPUs.

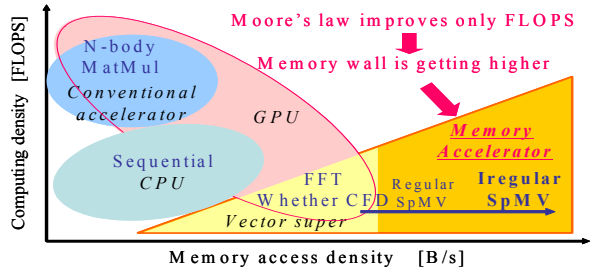
III. MEMORY ACCELERATOR

A. Basic concept

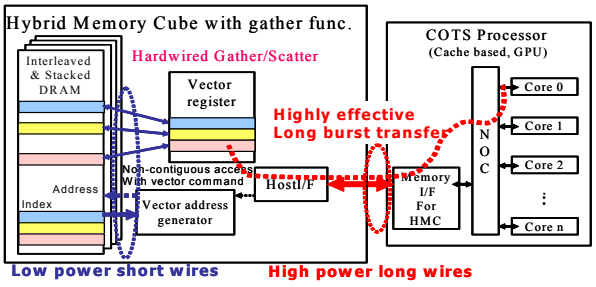
Figure 1(a) illustrates the mapping of applications and their suitable hardware accelerators categorized by the density of memory access and computation. The memory accelerator is hardware for the fast execution of memory bandwidth intensive applications such as irregular SpMVs that are difficult to be optimized for existing hardware accelerators. When a series of cache misses are issued, inefficient memory accesses where only four or eight bytes out of a 128 byte cache line are valid would be repeated. In such the case, the accesses to column vectors, which occupy one third of the total accesses, require the memory bandwidth 32 times for single precision and 16 times for double precision.

The memory accelerator has a memory controller with hardwired scatter/gather functions on the memory-side, i.e. between a block of external memory chips and a network-on-chip (NoC). The hardwired scatter/gather functions on the memory-side have been implemented in DIMMnet-2[15]. Figure 1(b) gives the concept of our proposed system including memory chips connected by many memory channels with a small number of wires for random memory accesses.

Recently, a memory subsystem that supports gather/ scatter capabilities is announced as a focus area [16] by IAA which is an organization for an Exa-FLOPS machine of the United States.



(a)



(b)

Figure 1. The concept of Memory accelerator: (a) Pourpose, (b) Proposed architecture.

B. Architectural improvement

In this paper, we propose an architectural improvement to the architecture of [3], where the interface of the memory accelerator is PCI express. Although it is reported in [3] that four times acceleration is observed with the combinatorial use of memory accelerator and pre-processing algorithms, it turned out that the bottleneck lies in the PCI express. As a short-term solution, we have adopted PCI express [3] as the host interface for the memory accelerator. For the middle and long term solutions, we are to adopt the GDDR5 device memory interface and the HMC (Hybrid Memory Cube) [17] interface, respectively. The current GDDR5 DRAM (Dynamic Random Access Memory) provides the bandwidth of 28GB/s.

We propose that the memory accelerator is implemented with 3D stacking as an HMC to integrate its hardwired scatter/gather functions inside the logic base chip of the HMC as shown in Fig. 1(b). Since each host interface of the

 ISBN: 978-1-61208-272-1

HMC provides 160 to 320GB/s bandwidth, the bottleneck problem would be resolved. The HMC with gather functions can randomly and effectively access many narrow memory banks using short low powered wires. It transmits just effective data on long off-chip wires. Therefore, the proposed memory system architecture must be low power and have high performance. Since the gather function can be implemented on a logic base of the HMC, the additional cost for the proposed architecture must be considerably smaller than that of conventional vector supercomputers. This improved memory can be a candidate for a new accelerator of future power constrained HPC platforms.

The cost of the proposed architecture with HMC interface is very low, since HMC essentially has a logic base chip which can easily have proposed gather functions. HMC is seen to be a low cost commonly used memory in the future. If the proposed logic is listed in the standard of future HMC, the hardware cost of the proposed architecture can be negligible.

IV. TARGET WORKLOAD FOR THE PROPOSED SYSTEM

In this paper, we use a CG solver for the evaluation of the proposed system as one of the target workloads. SpMV is the main part of the CG solver. In [3], an inter GPU communication reduction method for SpMV with keeping high scalability against the matrix shape and the number of GPUs is presented. The approach in [3] presents one of the best SpMVs from comprehensive standpoint considering scalability, load balancing and memory access efficiency.

As shown in Figure 2, an SpMV can be divided into multiplications of a set of row vectors and a column vector. Since there is no data dependence among the multiplications, it is quite easy to decompose the sparse matrix for each GPU by row vector according to the memory capacity constraint. The SpMV algorithm of [3] guarantees the scalability to remove the inter node communications by the strategy shown in Figure 2. In the case of the proposed architecture for accelerating CG, the resultant column vectors must be written back to the memory accelerator. In the case of multiple memory accelerators, the resultant column vectors must be multicasted to each memory accelerator. This multicast can be implemented so that it does not depend on the number of memory accelerators by using appropriate interconnection networks. Furthermore, the application of streaming would overlap the execution of the SpMV and multicast data transfer of resultant column vectors.

Furthermore, [3] proposes some pre-processing algorithms shown in Figure 3 to accelerate the performance of SpMVs. The pre-processes are padding, folding, and transposition to get better disposition of non-zero elements of the sparse matrix to GPUs. The use of the pre-processes in combination with the proposed hardware gives a maximum performance improvement of four times compared to [7]. In this paper, we follow [3] to use the SpMV algorithms and the hardware except the host interface of the GPUs.

In the CG solver, two parameters are generated from inner product operations to dense matrices, and using the two parameters each column vector of a sparse matrix is updated to apply SpMVs. The operation number of the dense matrices inner product is proportional to the number of unknowns of the simultaneous equations (the row number of the coefficient matrix), and the number of SpMVs is proportional to the number of non-zero elements of the coefficient matrix. It depends exactly on the disposition of non-zero elements. In general, the more non-zero elements per row the sparse matrix has, the more computation for the SpMVs the total operations contain. The less non-zero elements per row, the more computation there is for dense matrix inner product. Especially in the latter case, if the dense matrix inner product is not performed at the GPU but sent to the host to be calculated, it seems to have a risk to move the bottleneck to the host as well as data transfer overhead. Therefore, all the calculation should be performed in the GPU.

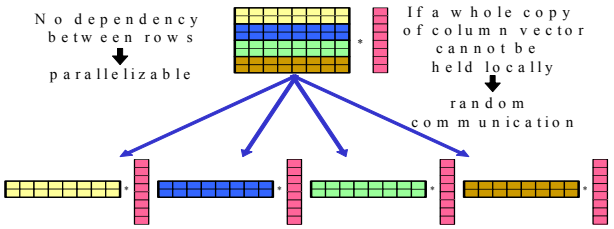


Figure 2. Strategy of scalable Sparse Matrix-Vector Multiplication in [3].

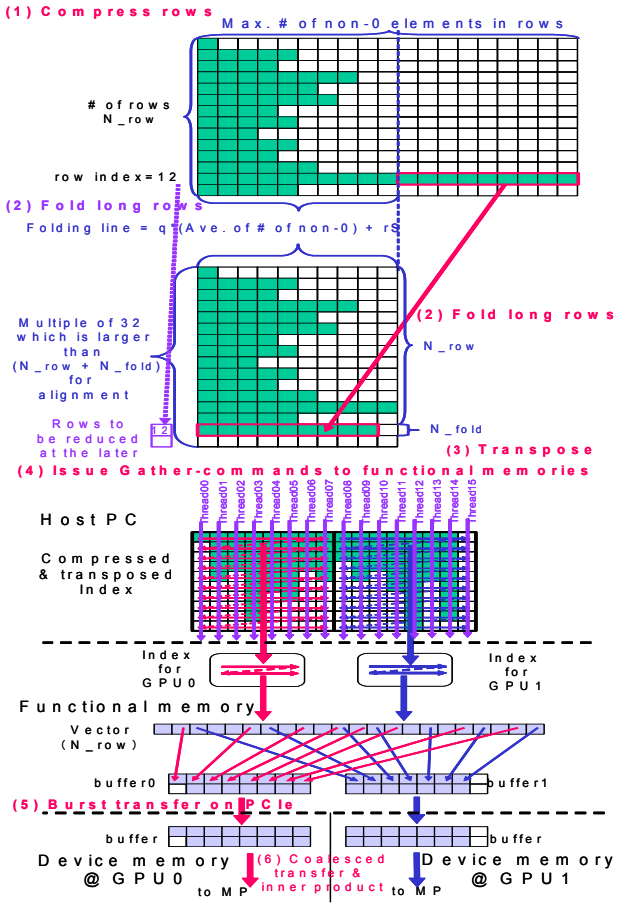


Figure 3. Flow of SpMV in [3].

V. EVALUATION

A. Evaluation environment and test matrices

The evaluation environment is illustrated in Table I (Tesla C1060) and Table II (Tesla C2050). Table III shows the test matrices. The test matrices are chosen from University of Florida Sparse Matrix Collection [2]. Although these sparse matrix problems are insufficient for our goal, namely not "so large problems that each vector cannot be stored in GPU device memory", we assume that the test matrices have the same characteristics to the too large vectors distributed among GPUs of the proposed system for parallel execution. In previous work [8], the same kinds of matrices were chosen for the performance evaluation of SpMV. In this paper, we chose the sparse matrices in [8]. Note that the chosen test matrices are not large enough. The size of the largest vector to be multiplied is so small (6.3MB) as to be insignificant compared with the device memory size. Therefore, we can say that the evaluation experiments are performed in a condition that cache effect is stronger (the condition that the cache-based previous work is profitable) than the finally supposed case (the condition that the problem is too large). Actually our method does not need cache effect so much.

TABLE I. EVALUATION ENVIRONMENT (C1060)

CPU	Intel®[a] Core(TM) i7 CPU920 @ 2.67GHz
GPU	Nvidia Tesla C1060 (# of core=240, 4GB, Memory bandwidth103GB/s)
Host I/F	PCI express x16 Gen2 (bandwidth 8GB/s)
OS	Fedora10
CUDA	Cuda3.0

TABLE II. EVALUATION ENVIRONMENT (C2050)

CPU	Intel® Xeon®[a] CPU X5670 @ 2.93GHz
GPU	Nvidia Tesla C2050 (# of core=448, 3GB, Memory bandwidth144GB/s)
Host I/F	PCI express x16 Gen2 (bandwidth 8GB/s)
OS	Red Hat Enterprise Linux Client release 5.5
CUDA	Cuda3.2
ECC	Off

a. Intel, Intel Core, and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.

TABLE III. TEST MATRICES

Name	# of rows	Non-0 elements			
		Total	*Ave.*	*Max*	σ
Na5	5,832	155,731	26	185	35.7
msc10848	10,848	620,313	57	300	49.4
thermal2	147,900	3,489,300	23	27	6.9
hood	220,542	5,494,489	24	51	13.3
F1	343,791	13,590,452	39	306	20.0
ldoor	952,203	23,737,339	24	49	12.9
G3_circuiit	1,585,478	4,623,152	2	4	2.2

When we solve a large problem for the same kind of applications, it is expected that the number of non-zero elements per row of the sparse matrices is considered as a constant for most cases. When a large matrix is decomposed for multiple GPUs by row based on the strategy described in Figure 2, the memory accesses to the matrix issued by each GPU does not change so much when the matrix size is small. Since the indirect referenced column vector size increases when the problem size is large, the cache hit rate of existing cache based systems deteriorates. In the meantime, in the case of the proposed method, such performance degradation does not occur when the column vector size increases because the proposed method is based on a vector processor architecture. Furthermore, the strategy described in Figure 2 does not include any point-to-point communication, namely it is scalable. In short, the effect of the proposed method to the performance gain for small matrices to be evaluated for a GPU gives the lower limit of the performance gain for large matrices to be evaluated for GPU clusters.

B. Implementation of CG solver

For the evaluation of the proposed architecture, we implement three types of CG solvers with different accesses to the column vectors in the kernel, as explained below. In either case, pre-processes described in the previous section are applied to perform SpMV so that we analyze the relation between the access types of vectors and acceleration efficiency.

1) Texture memory version: In this version, column vectors are stored in GPU texture memory so that Tex2D function is called to make use of texture cache. This version gives the criteria of performance to be compared with other versions, and the number of iterations to be converged in this version is given to (3) as described later.

2) Shared memory version: In this version, column vectors on device memory are accessed via shared memory. In the case of Fermi C2050, those accesses are accelerated by L1 and L2 caches.

3) Proposed architecture version: In this version, column vectors on device memory, which get disposition operations in advance (fairing and transposition), are accessed. Because of the disposition operations, the original indirect references are converted to direct references in the source code so that they are in the form of burst and coalesced access. We assume that we have enough memory bandwidth of memory accelerator.

Note that the proposed method assumes the use of the mixed precision iterative refinement algorithm [19]. Although the algorithm [19] mostly consists of a single precision CG solver, it provides rich convergence ability to be comparable with double precision operations. For this reason, we measure the execution speed of a single precision CG solver in this evaluation. In the case that there is a matrix not to be converged with single precision operations, we just measure the execution speed of the loop which includes the matrix for a fixed number of interactions.

C. Texture cache hit rate

We measure the texture cache hit rate of the texture memory version to be executed on a C1060 with profiling. CUDA 3.0 provides performance counters tex_cache_hit and tex_cache_miss. Figure 4 shows the relation between the

 ISBN: 978-1-61208-272-1

matrix size (the number of rows) and the texture hit rate. As the number of rows increases, the texture cache hit rates decreases because column vector accesses protrude the small texture cache. In Figure 4, we show the linear approximation by the black line, which shows a precipitously falling diagonal stroke from top left to bottom right. The linear approximation indicates that when the matrix size increases further, any cache effect is not expected.

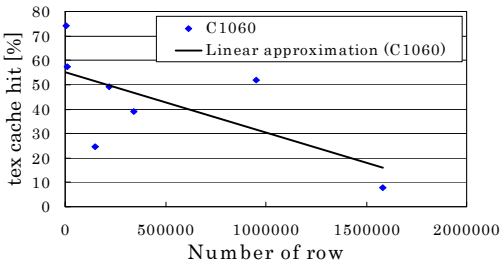


Figure 4. Matrix size and the texture hit rate.

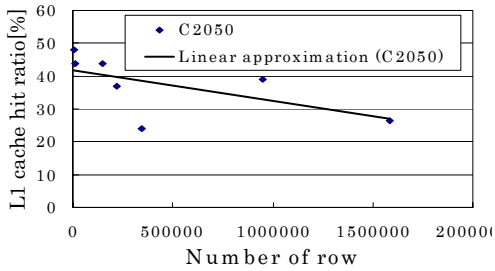


Figure 5. Matrix size and the L1 cache hit rate.

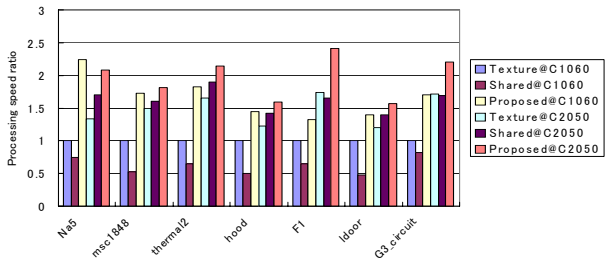


Figure 6. Processing speed ratio of SpMV.

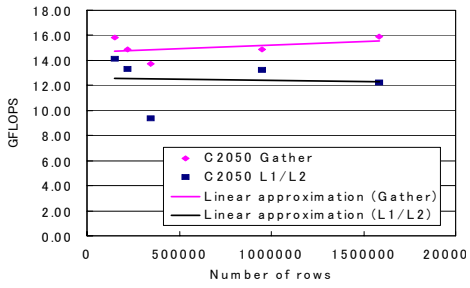


Figure 7. Matrix size and GFLOPS of SpMV.

Actually, among the matrices in this experiment, the largest one G3_circuit (1,585,478 rows) gets the cache hit rate of 7.74%, which reveals that the texture cache has spilt out. In the meantime, in G3_circuit the average number of non-zero elements per row is 2, and this means there are very few reusable data in each cache line, which makes the cache hit rate low, too.

D. General purpose cache hit rate

We measure the general purpose cache (L1) hit rate of the shared memory version to be executed on a C2050 with profiling. CUDA 3.0 provides performance counters L1_global_load_hit and L1_global_load_miss.

Figure 5 shows the relation between the matrix size (the number of rows) and the L1 cache hit rate, where the preference of L1 cache is LARGER (L1 cache is 48KB while shared memory is 16KB). As in the case of texture cache, the L1 cache hit rate decreases when the matrix size is large. G3_circuit gets the cache hit rate of 26.5% which is better than 7.74% on C1060. In the case of F1, the L1 cache hit rate is 23.9% which is lower than the texture cache hit rate. This phenomenon can be explained as follows. The L1 cache tries to keep any row vectors even if they do not have reusable data. Since F1 contains a large number of non-zero elements, F1 gets the lower cache hit rate as the result. To avoid this situation, when the row vector without reusable data is loaded, a special instruction, which skips L1 cache to directly load vector data, should be used.

E. SpMV execution times for three implementations

Figure 6 shows the processing speed ratio of each execution time of SpMV kernel to the execution time of the texture memory version on C1060. At a glance, it turns out that the proposed architecture version on both C1060 and C2050 is better than other implementations. Note that the effect of the additional hardware for the proposed architecture is very limited because the target matrices are so small. In other words, the proposed architecture provides more throughputs without any cache effect than the throughputs of GPUs with a certain amount of texture or L1 cache effect for relatively small matrices. Since the previous experiment shows that the use of larger matrices decreases the cache effect, the proposed architecture provides much more throughput for larger matrices compared with other implementations.

Considerable performance improvement is observed also in the shared memory version on C2050 compared with C1060 because of the improvement of device memory bandwidth and L1/L2 cache effect on C2050. This performance improvement seems to be given mainly by L2 cache effect. Since L2 cache is not so large compared with L1 cache, the performance improvement is limited when the target matrices are larger.

In [10], M. M. Baskaran et al. reports the execution times of F1 and ldoor with double precision SpMV in the JDS format on C2050. Our shared memory version on C2050 is 4.1 times and 2.74 times faster than in [10] for F1 and ldoor, respectively. Although our shared memory version is single precision that naturally makes two times

 ISBN: 978-1-61208-272-1

performance improvement because of the difference of the device memory bandwidth (double vs. single), the pre-processing (folding) of our shared memory version achieves more performance improvement than their implementation.

Figure 7 shows the relation between matrix size and GFLOPS of SpMV on C2050. The shape of the graph of GFLOPS values with L1/L2 is similar to that of the L1 cache hit rate (Figure 5). We observe a tendency of performance degradation for L1/L2 as the matrix size increases. On the contrary, we observe an improving tendency of matrix size for Gather. This result predicts that the performance ratio between Gather and L1/L2 cache will increase when the gap of cache capacity and solution vector size increases.

F. CG solver execution times for three implementations

When the SpMVs are fully accelerated on the GPU, the inner product operations performed on the host PC become a bottleneck. The inner product of dense matrices can be calculated in parallel, and seems to be accelerated using CUBLAS [18]. After the above consideration, we apply CUBLAS to the inner product operations and move most of the miscellaneous operations (except residual operations to be executed on the host PC) to the GPU. Figure 8(a) shows the speed ratio of the CG solver using CUBLAS applied to various matrices on C2050 with three implementations.

Each matrix is symmetric but some matrices are not positive definite. Furthermore, they are single float operations. So, the number of iterations is totally different by matrix, and some matrices cause the CG solver not to converge. Taking into account the above problems, we measure partial iterations to calculate the average execution time by iteration. Since the shared version program could not be executed for ldoor on our environment, the result is not available.

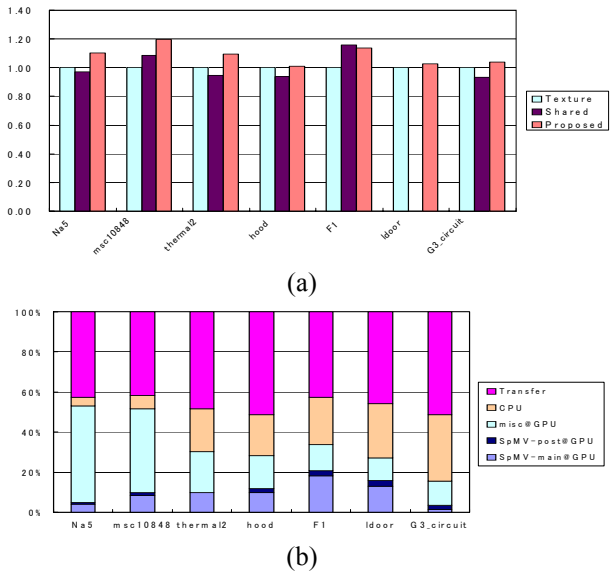


Figure 8. Performance of the CG solver using CUBLAS based inner products on C2050: (a) Speed ratio, (b) Breakdown.

The result is that the proposed architecture version works 1.01 to 1.20 times faster than texture memory version although the matrices are small enough to take advantage of the texture cache effect. These speed ratios are not attainable by the kernels shown in Figure 6. This is because operations other than SpMVs and inner products are executed on the host PC.

To investigate the performance effect of non-kernel operations, we measure other operations of the CG solver. Figure 8(b) shows the breakdown of the execution times by our proposed architecture using CUBLAS based inner products on C2050. It turns out that the post SpMV operations which are reductions of partial inner products of folded vectors and the main part of SpMVs do not take the major part of the processes, but the other calculations on the CPU and the data transfer between the host PC and the GPU dominate the total processes. This is one of the reasons why the speed-up ratio in Figure 8(a) is not so large. The second reason why the speed-up ratio in Figure 8(a) is not so large is that the workload matrix is too small for the L2 cache hit rate and the device memory bandwidth to dominate the performance. If workload matrices are large enough that these parameters dominate the performance, the speed-up ratio must be increased since the proposed memory system is based on the vector architecture whose performance does not decrease when the vector length is larger.

The amount of inner product operations depends on unknown variables of the linear equation. In the case of G3_circuit where the average number of non-zero elements per row is two, the amount of SpMVs is almost as same as inner product operations. This is the reason why miscellaneous operations dominate the execution time of G3_circuit. As the result of reducing CPU execution times, the data transfer between the host and the GPU comes to dominate the total execution time. If the complete optimization, namely all operations on a GPU such as Kepler by Nvidia, was achieved, the data transfer between the host and the GPU would be reduced and higher acceleration would be expected.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we evaluate texture and general purpose cache hit rates for conventional SpMVs. We confirm that relatively small matrices (several hundred Kilo Byte to several Mega Byte) taken from University of Florida Sparse Matrix Collection [2] generate cache hit rates of 10% to 70%, which are low. Larger matrices tend to degrade cache hit rates and FLOPS performance.

To keep high scalability of the number of GPUs, each GPU should contain a copy of the target column vector in its device memory. When the target is a mesh of 1,000 cubic, the column vector size becomes 8GB and exceeds the capacity of the device memory. In this case, the overhead of data transfer via PCI express would make the performance worse. The memory accelerator, which has gather functions and a huge capacity of memory optimized for short bursts, has been proposed in [3]. The memory accelerator can replace cache memory so that it treats huge problems that

 ISBN: 978-1-61208-272-1

matrices and column vectors are too large to be stored in the device memory.

The problem of the memory accelerator in [3] is that SpMV on GPUs reveals PCI express is the bottleneck. In this paper, we proposed that a memory accelerator is connected to the GDDR5 port or the HMC port of GPUs. In these cases, the bandwidth per port increases, and the total bandwidth additionally increases using multiple ports.

The previous evaluation could not explain the performance effects by replacing cache memory with the memory accelerator and by pre-processing algorithms separately. In this paper, we evaluate the performance effects on University of Florida Sparse Matrix Collection [2] with different memory systems including the memory accelerator. As a result, even in the case of small matrices where texture cache is effective, it turned out that our proposed architecture works 1.01 to 1.20 times faster than the texture cache based existing method. If workload matrices are large enough that the L2 cache hit rate and the device memory bandwidth dominate the performance, speed-up ratio will be greater since proposed memory system is based on vector architecture whose performance does not decrease when the vector length is larger. If the complete optimization, namely all operations on a GPU such as Kepler by Nvidia, was achieved, higher acceleration would be expected.

While the previous evaluation was just for SpMV, in this paper we evaluate the CG solver including SpMV. In the CG method execution, the full acceleration of SpMV on a GPU exposes the other processes of dense matrix inner product on a CPU and data transfer latency between the CPU and the GPU. Shifting some part of the miscellaneous processes to the GPU side, we observe considerable performance improvement.

Our future work includes the implementation and evaluation of streaming that hides the write-back latency for the memory accelerator, the exact evaluation of L2 cache effect for larger matrices, the evaluation of the scalability of many GPUs and memory accelerators, and the design and evaluation of a memory accelerator with large capacity optimized with short bursts.

ACKNOWLEDGMENT

A part of this work is supported by the Ministry of Internal Affairs and Communications (Soumu-sho).

REFERENCES

[1] R. R. Schaller : Moore's law: past, present and future, Spectrum, IEEE , Vol.34, No.6, pp.52,59 (1997)

[2] Tim Davis : "The University of Florida Sparse Matrix Collection", http://www.cise.ufl.edu/research/sparse/matrices/ [retrieved March 2013]

[3] N. Tanabe, Y. Ogawa, M. Takata, K. Joe : Scaleable Sparse Matrix-Vector Multiplication with Functional Memory and GPUs, 19th Euromicro Conference on Parallel, Distributed and Network-Based Computing (PDP 2011), pp. 101-108 (2011)

[4] X. Yang, S. Parthasarathy, P. Sadayappan : Fast Sparse Matrix-Vector Multiplication on GPUs: Implications for Graph Mining, 37th International Conference on Very Large Data Bases (VLDB2011), pp.231-242 (2011)

[5] A. Cevahir, A. Nukada, S. Matsuoka : High performance conjugate gradient solver on multi-GPU clusters using hypergraph partitioning, Computer Science - Research and Development, Vol.25, No.1-2, pp.83-91 (2010)

[6] M. Ament, G. Knittel, D. Weiskopf, W. Straser : A Parallel Preconditioned Conjugate Gradient Solver for the Poisson Problem on a Multi-GPU Platform, 18th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2010), pp.583-592 (2010)

[7] A. Cevahir, A. Nukada, S. Matsuoka : An Efficient Conjugate Gradient Solver on Double Precision Multi-GPU Systems, Symposium on Advanced Computing Systems and Infrastructures (SACSIS2009), pp.353-360 (2009)

[8] Y. Kubota, D. Takahashi : Optimization of Sparse Matrix-Vector Multiplication by Auto Selecting Storage Schemes on GPU, Computational Science and Its Applications(ICCSA 2011), LNCS Vol. 6783, pp.547-561 (2011)

[9] N. Bell, M. Garland : Eficient Sparse Matrix-Vector Multiplication on CUDA, Nvida Technical Report NVR-2008-004 (2008)

[10] M. M. Baskaran, R. Bordawekar : Optimizing Sparse Matrix-Vector Multiplication on GPUs, IBM Research Report, RC24704 (2009)

[11] A. Monakov, A. Lokhmotov and A. Avetisyan : Automatically Tuning Sparse Matrix-Vector Multiplication for GPU Architectures, 5th International Conference on High Performance Embedded Architectures and Compilers (HiPEAC 2010), LNCS 5952, pp.111-125 (2010)

[12] F. Vazquez, G. Ortega, J. J. Fernandez and E. M. Garzon : Improving the Performance of the Sparse Matrix Vector Product with GPUs, 10th IEEE International Conference on Computer and Information Technology (CIT 2010), pp.1146-1151 (2010)

[13] Nvida : CUSPARSE User Guide, http://docs.nvidia.com/cuda/pdf/CUDA_CUSPARSE_Users_Guide.pdf [retrieved March 2013]

[14] - : cusp-library : Generic Parallel Algorithms for Sparse Matrix and Graph Computations, http://code.google.com/p/cusp-library/ [retrieved March 2013]

[15] N. Tanabe, M. Nakatake, H. Hakozaki, Y. Dohi, H. Nakajo, H. Amano : A New Memory Module for COTS-Based Personal Supercomputing, International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA2004), pp.40-48, DOI: 10.1109/IWIA.2004.10019 (2004)

[16] IAA: Focus area, http://iaa.sandia.gov/focus-areas/ [retrieved March 2013]

[17] Micron Technology, Inc. : Hybrid Memory Cube : Breakthrough DRAM Performance with a Fundamentally Re-Architected DRAM Subsystem, Hot Chips 23 (2011)

[18] Nvida : CUBLAS User Guide, http://docs.nvidia.com/cuda/pdf/CUDA_CUBLAS_Users_Guide.pdf [retrieved March 2013]

[19] A. Buttari, J. Dongarra, J. Kurzak, P. Luszczek, S. Tomov : Using Mixed Precision for Sparse Matrix Computations to Enhance the Performance while Achieving 64-bit Accuracy. ACM Transactions on Mathematical Software, Vol.34, No.4 (2008)

 ISBN: 978-1-61208-272-1