Accelerating Breadth First Search on GPU-BOX

Takuji Mitsuishi
Keio University
3-14-1 Hiyoshi, Yokohama,
223-8522, Japan
mits@am.ics.keio.ac.jp

Shimpei Nomura
Keio University
3-14-1 Hiyoshi, Yokohama,
223-8522, Japan
nomura@am.ics.keio.ac.jp

Jun Suzuki
NEC Corporation
1753, Shimonumabe,
Nakahara-ku, Kawasaki,
Kanagawa 211-8666, Japan
j-suzuki@ax.jp.nec.com

Yuki Hayashi
NEC Corporation
1753, Shimonumabe,
Nakahara-ku, Kawasaki,
Kanagawa 211-8666, Japan
y-hayashi@kv.jp.nec.com

Masaki Kan
NEC Corporation
1753, Shimonumabe,
Nakahara-ku, Kawasaki,
Kanagawa 211-8666, Japan
kan@bq.jp.nec.com

Hideharu Amano
Keio University
3-14-1 Hiyoshi, Yokohama,
223-8522, Japan
asap@am.ics.keio.ac.jp

The graph analysis has been applied in various fields related to big-data processing and actively researched in recent years. For processing a larger scale of graph, parallel computing with multi-GPU system is paid attention as an economical solution. Here, an efficient parallel method is proposed to solve a typical graph analysis, Breadth First Search (BFS) for multi-GPU systems. Our target system is GPU-BOX, a prototype of multi-GPU system using ExpEther which is a virtualization technology based on PCI Express and Ethernet. Although many vertices between GPUs must be exchanged to run BFS on multi-GPU system, GPU-BOX provides only small communication performance because of using Ethernet. Our parallel algorithm for BFS is designed so as to reduce the traffic between GPUs as possible. The proposed method reduced 30–40% traffic between GPUs and improved the traditional parallel method by 10%.

Keywords

GPU, Cluster, ExpEther, Graph Algorithm, Scalability

1. INTRODUCTION

Graph analysis is a key processing in big data analysis. It is frequently utilized in social networks, micro-blogs, protein-protein interactions, and the connectivity of the Web. In such fields, large target graphs sometimes require a large computation cost [15]. As a cost efficient solution, multi-GPU systems which have been widely utilized in scientific computing have been researched recently. Graph500, a benchmark with a graph analysis, was adopted as a target application program for ranking supercomputers.

However, although supercomputers in which nodes providing a number of GPUs are connected with powerful interconnection networks like Infiniband achieve a high performance, they are not suitable for data-centers which work mainly for non-numeric computing considering their cost. As a cost-effective alternative, we have proposed a multi-GPU system connecting with ExpEther [14]. ExpEther is a virtualization technique for extending PCI Express used inside a PC to Ethernet, and gives programmers a flat view of network connecting various types of devices located at distant places. By using ExpEther, various number of GPUs can be connected, and programmers can treat them as if they were connected with a PCI Express bus of a node. Various number of GPUs can be attached to a data-center depending on the performance requirement.

However, a problem of multi-GPU system using ExpEther is relatively poor communication performance between multiple GPUs. Although it allows direct data communication between all GPUs, packets must be transferred on Gbit Ethernet through two bridges; PCI Express to Ethernet and Ethernet to PCI Express. The latency is stretched compared with supercomputers connected with the powerful SANs (System Area Networks) like Infiniband. Some Breadth First Search (BFS) algorithms which make the best use of a single or multiple GPUs have been proposed [11] [10]. Unfortunately, they require a large amount of communication which might degrade performance on multi-GPU systems with ExpEther. Since communication amount is increased as the size of graph and the number of GPUs, the communication will become bottleneck when a large sized graph is analyzed by a large number of GPUs in the future.

Here, a parallel BFS algorithm which reduces the communication between GPUs as possible is proposed, implemented and evaluated on a multi-GPU system with ExpEther called GPU-BOX. The proposed algorithm minimizes the communication of vertices between GPUs by setting flags only on the active vertices.

The rest of paper is organized as follows: First, our target multi-GPU system is introduced in 2 as a cost efficient method to provide multiple GPUs in a datacenter. Then, the Breath First Search is introduced in 3, focusing on parallel processing of a level-synchronized BFS. After a related BFS algorithm is explained in section 4, we propose our BSF algorithm in 5.

Finally, the evaluation results are shown in 6 and 7 concludes this paper.

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2014), Sendai, Japan, June 9-11, 2014.

2. A MULTI-GPU SYSTEM WITH EXPETHER

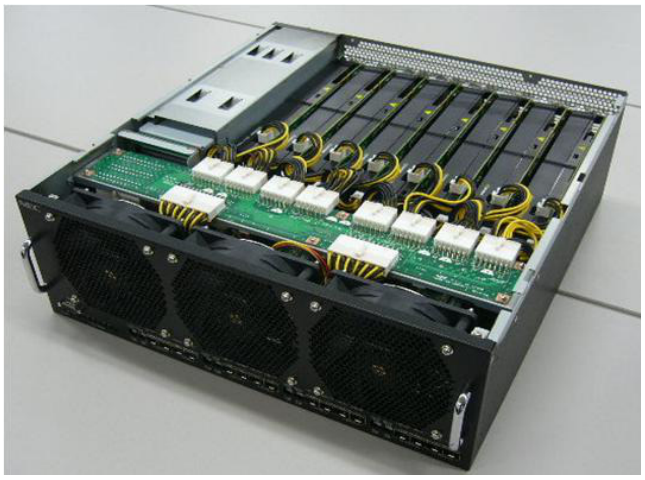


Figure 1: GPU-BOX

ExpEther [14] is an Ethernet based virtualization technique, which was developed by NEC [12]. It extends PCI Express which is high performance I/O network but limited to small area around the cabinet to much wider area by using Ethernet. Various types of I/O devices on distant location can be connected as if they were inside the cabinet.

Now, a prototype multi-GPU system with ExpEther called GPU-BOX is available. As shown in Figure 2, a ExpEther NIC is connected to each PCI Express port of GPU. In the NIC, PEB (PCI Express-to-Ethernet Bridge) is provided. It encapsulates Transaction Layer Packet (TLP) used in PCI Express into Ethernet frame, and decapsulates it for extending PCI Express to Ethernet. Multiple GPUs are connected with a common Ethernet switch. A delay-based protocol which supports repeat and congestion control is utilized instead of TPC, the loss-based protocol. It also employs Go-back-N flow-control instead of Selective-Repeat. By using such a performance centric protocol for small area communication, ExpEther can support larger bandwidth and smaller latency than common Ethernet even with common cables and switches for Ethernet. A host can be connected through ExpEther NIC, and all GPUs can be treated as if they were connected to the PCI Express of the host.

Since ExpEther gives programmer a flat view of multi-GPUs, the programming complexity is much reduced. Programmers do not have to care about the communication between hosts which provide a number of GPUs.

Although the proposed system enables to treat a lot of GPUs located in the distant places, as the first step of research, here, multiple GPUs are implemented in a single cabinet called GPU-BOX. As shown in Figure 1, GPU-BOX provides multiple PCI Express slots with ExpEther NIC and power supply. The target GPU-BOX in this paper has eight slots. Through the ExpEther NIC implemented with an FPGA, two 10Gb Ethernet ports (EFP+) are provided for each slot, thus, 20Gbps bandwidth is available in total. The power supply of the target GPU-BOX is up to 3000W which is enough to connect eight GPUs. [13]

3. BREADTH FIRST SEARCH

Breadth-first search (BFS) is a typical graph analysis algorithm which visits every vertex in all levels of the graph from a source vertex. Here, each vertex is labeled by the parent number, and the result of the search is represented by the label.

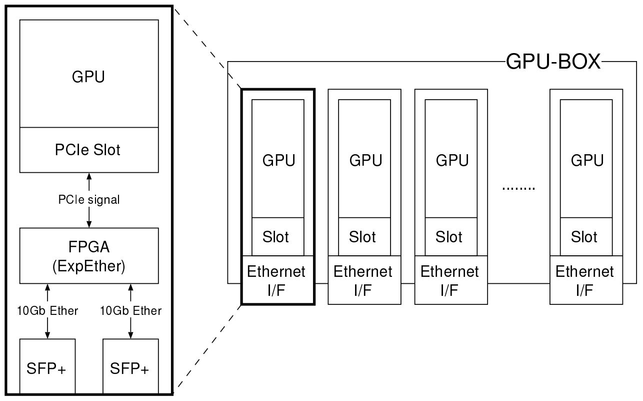
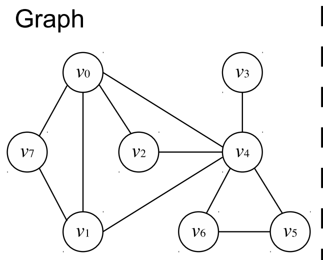


Figure 2: Constitution of GPU-BOX



Adjacency matrix

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Compressed Sparse Row (CSR)

$$C = [1, 2, 4, 7, 0, 4, 7, 0, 4, 4, 0, 1, 2, 3, 5, 6, 4, 6, 4, 5, 0, 1]$$
$$R = [0, 4, 7, 9, 10, 16, 18, 20, 22]$$

Figure 3: CSR

Undirected graphs generated for Graph500 benchmark are used as target graphs. They are sparse graphs generated by a Kronecker generator like the Recursive MATrix (R-MAT) graph generation algorithm [2] [7] [9]. A generated graph is represented by using an adjacency matrix A, whose rows are corresponding to the adjacency lists A_i. The number of edges M in generated graphs is constant multiplication (default: 16) of the number of vertices N. The graphs are transformed into Compressed Sparse Row (CSR) sparse matrix format constructed of two arrays C and R [11]. The column-indices array C is a single array of M 64-bit integers that have all adjacency lists A_i $(0 \leq i < N)$ successively. The row-offsets array R is a single array of $N + 1$ 64-bit integers whose the entry $R[i]$ indicate the start of adjacency list A_i in C. Figure 3 shows an example. The left side represents the adjacency matrix, while the right side is corresponding to the compressed CSR form.

Next, a level-synchronized BFS which is suitable for parallel computing is introduced [5] [4]. Most parallel BFS algorithms are based on the level-synchronized BFS. It is consisted of the following three steps. Here, the output is represented by the parent vertices of the searched result rather than the distance between the source vertex.

Table 1: Level-synchronized BFS

BFS i	Current F	Next F	Label
0	2	–	-1, -1, 2, -1, -1, -1, -1, -1
1	2	0, 4	2, -1, 2, -1, 2, -1, -1, -1
2	0, 4	1, 3, 5, 6, 7	2, 0, 2, 4, 2, 4, 4, 0
3	1, 3, 5, 6, 7	–	2, 0, 2, 4, 2, 4, 4, 0

1. Add source vertex v_s into the current frontier, and give label v_s by its own vertex number s. We call this step BFS iteration 0 in this paper.

2. Add neighbors v_i which have not been visited in the current frontier into the next frontier. Give label v_i by its parent vertex number p_i.

3. If the next frontier is not empty, swap current frontier with next frontier then repeat step 2. Otherwise, finish searching and output labels.

The frontiers in the above steps are the sets of active vertices on that level, and they are used to judge the end of searching. We call the iteration of step 2 and step 3 BFS iteration. Assuming that we search the graph of Figure 3 from v_2 with the source vertex, three BFS steps are iterated as shown in Table 1.

Since the searching program is executed by using n GPUs, we have to divide the graph into n pieces. Each GPU processes N/n disjoint vertices and the corresponding adjacency lists in parallel. However, each GPU has to exchange vertices assigned into other GPUs in the frontier between all GPUs in every BFS iteration.

By using a lot of GPUs, while the computation of BFS can be accelerated, total amount of communication in the system is increased. It enlarges the ratio of communication time in the total execution time if the network with narrow bandwidth is used. In order to cope with this problem, a new method is proposed to reduce the number of vertices to be exchanged so as to reduce the total amount of communication.

4. RELATED WORK

Recently, several parallel algorithms have been proposed for a single or multi-GPU systems. Here, Mastrostefano's parallel BFS algorithm [10] for multi-GPU systems with multiple nodes is introduced as our base.

This algorithm is queue-based algorithms. The BFS iterations of this algorithms are consisting of the following three steps.

1. Each GPU expands neighbors in current frontier into an intermediate frontier.

2. Each GPU contracts the intermediate frontier.

3. Each GPU exchanges vertices in the contracted frontiers between all GPUs to make the next frontier.

"Expand" in step 1 means gathering all neighbors of vertices in the frontier. "Contract" in step 2 means removing redundant vertices from the frontier. Here, redundant vertices mean duplication in the frontiers. They are generated as multiple CUDA threads gathering neighbors at the same time, and different edges are connected to the same vertex.

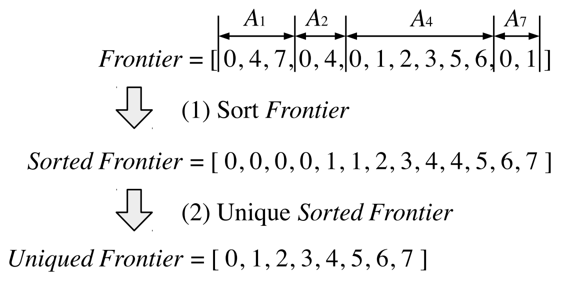

A_1
A_2
A_4
A_7
Frontier = [0, 4, 7, 0, 4, 0, 1, 2, 3, 5, 6, 0, 1]
(1) Sort *Frontier*
Sorted Frontier = [0, 0, 0, 0, 1, 1, 2, 3, 4, 4, 5, 6, 7]
(2) Unique *Sorted Frontier*
Uniqued Frontier = [0, 1, 2, 3, 4, 5, 6, 7]

Figure 4: Sort-Unique

In the common implementation, the computation order becomes $O(N^2 + M)$ in worst case similar to Harish's algorithm [8]. Most of computation is spent for gathering neighbors from the column-indices array C in the global memory. That is, it is corresponding to neighbor expansion process described above. However, the computation of neighbor expansion in this algorithm is suppressed to $O(N + M)$ by efficiently using the resource in GPUs.

In step 2 for frontier contraction, Mastrostefano removed all duplicates from intermediate frontier completely by using Sort-Unique method. Sort-Unique is a simple method. First, vertices in intermediate frontier are sorted by their vertex identifier, then removed all but the first vertex in the group of consecutive vertices with the same identifier, resulting a unique vertex. In this algorithm, "sort and unique" is performed by Thrust library [3] developed for GPU. Figure 4 shows an example. In this example, a graph shown in Figure 3 is assumed as a target and expanded $Current\ Frontier =$ [1, 2, 4, 7]. Then, expanded frontier is contracted. The subsets of $Uniqued\ Frontier$ in Figure 4 is exchanged between GPUs in the step 3.

Since a multi-GPU system with multiple nodes executes this algorithm, Mastrostefano's algorithm is written in the MPI program for communication of multiple nodes in order to exchange vertex data between GPUs.

5. PROPOSED METHOD

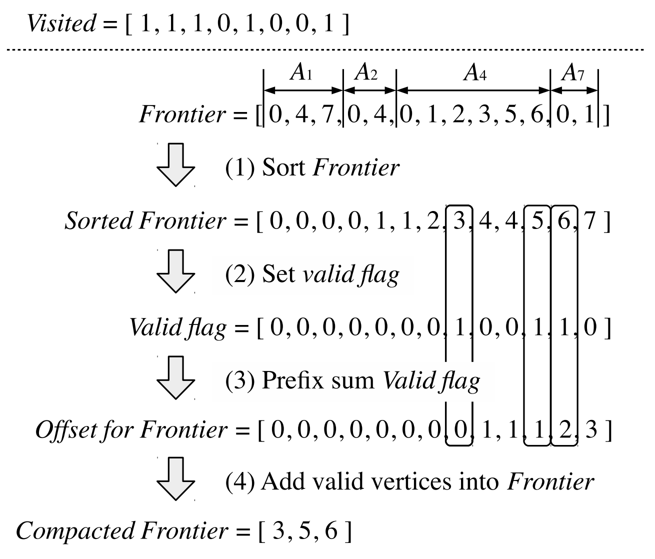
In order to make the best use of flexibility of ExpEther, the parallel BFS algorithm must be scalable. For keeping scalability with a relatively small bandwidth of ExpEther, the traffic between GPUs should be reduced as possible. We selected Mastrostefano's algorithm as our base, since it was designed for multi-GPU systems, and improved it so that the traffic between GPUs is small as possible.

The frequent communication between GPUs is caused by the frontier contraction (step 2) and exchange vertices (step 3). Thus, we focus on them without touching other parts (step 1).

5.1 Frontier contraction

Mastrostefano removed only duplicates in the intermediate frontier by Sort-Unique method in the step of frontier contraction. Our main idea is to remove the vertices which each GPU has visited locally as well as duplicates.

We propose the method which reduces traffic between GPUs by removing also the vertices which each GPU has visited locally. Vertices which all GPUs have visited globally cannot be removed since it might increase the traffic for exchanging the information obtained by visit between



Visited = [1, 1, 1, 0, 1, 0, 0, 1]
A1
A2
A4
A7
Frontier = [[0, 4, 7, 0, 4, 0, 1, 2, 3, 5, 6, 0, 1]]
(1) Sort Frontier
Sorted Frontier = [0, 0, 0, 0, 1, 1, 2, 3, 4, 4, 5, 6, 7]
(2) Set valid flag
Valid flag = [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0]
(3) Prefix sum Valid flag
Offset for Frontier = [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 2, 3]
(4) Add valid vertices into Frontier
Compacted Frontier = [3, 5, 6]

Figure 5: Proposed method

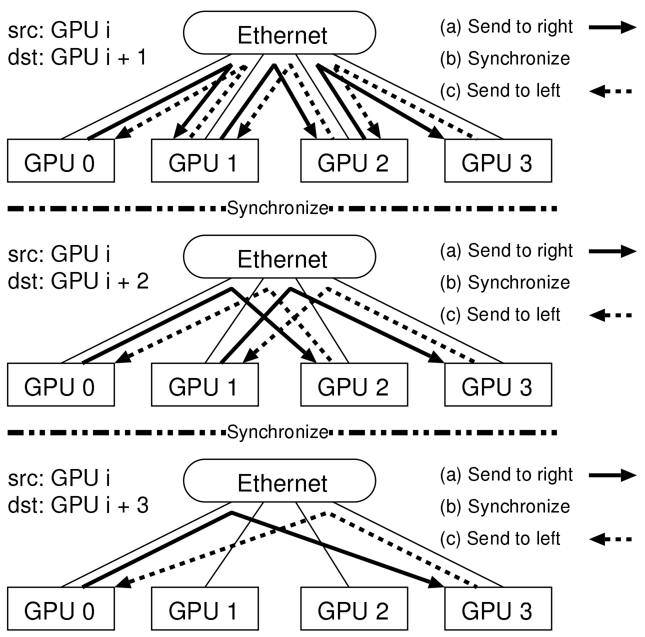


Figure 6: Communication between four GPUs

GPUs.

As Mastrostefano pointed out [10], in this method, all GPUs must save the information about visiting vertices in global memory. Since the information cannot be distributed for GPUs, the global memory is occupied by the visiting information when the number of graphs becomes large. On the other hand, if a large graph is treated by a log of GPUs, communication will occupy a large part of execution time.

Especially, in multi-GPU systems which use ExpEther, the communication is more likely to be a bottleneck because of its limited throughput between GPUs. Thus, we adopted to reduce the communication, while the visiting information requires a certain amount of global memory.

Figure 5 explains the proposed method. The same condition assumed for Sort-Unique is used in Figure 4.

(1) Similar to Sort-Unique, sort the vertices in $Frontier$.

(2) If the vertex in $Sorted\ Frontier$ is valid, that is not duplicated and not has been visited, set a $Valid\ flag$ for each vertex in $Sorted\ Frontier$.

(3) Perform a prefix sum with the $Valid\ flag$ to calculate offset for storing the valid vertices into $Compacted\ Frontier$.

(4) Store the valid vertices into $Compacted\ Frontier$ using $Valid\ flag$ and $Offset\ for\ Frontier$.

The visiting information of every vertex for each GPU is recorded in $Visited$ array on each GPU memory which contains N bytes. For easy implementation, visiting information is represented by the byte data rather than bit-vector. We can judge whether the vertex has been visited by $Visited$ array at (2). This method requires more amount of memory in each GPU for storing $Visited$ array than Sort-Unique method.

The proposed method can be implemented by four kernels corresponded to (1) – (4) in Figure 5. CUB library [1] which Merrill et al. developed to (1) sort and (3) prefix sum can be used. The CUB library has a benefit of stability of the performance compared with Thrust library. Also, it allows to allocate and free temporary memory for library explicitly unlike Thrust library. A $Valid\ flag$ is set when the vertex identifier is not equal to immediate left. Also, the flag is set when the vertex is the most left side element in $Sorted\ Frontier$. $Visited$ array is updated as shown in (4) of Figure 5.

When we implement (2) – (4) in Figure 5 by one kernel, we can write the $Valid\ flag$ data into registers, so that memory usage is reduced. However, this method requires synchronization between thread blocks in kernel, and introduces implementation difficulty. So, multiple kernels are used here.

5.2 Exchange vertices

Since multi-GPU systems with ExpEther can support a system consisting of multiple GPUs connected to a single host, vertices in each GPU can be gathered by Unified Virtual Addressing (UVA). UVA supported over CUDA 4.0 enables to use a unify address spaces for integrating memories of all CPUs and GPUs into a single address space virtually. We can copy data directly between two different GPU memory modules without copying data in CPU memory temporarily.

We apply Left-Right approach [6] for each GPU to communicate with all GPUs. Since multiple GPUs send and receive at the same time, we can achieve large aggregate throughput. Figure 6 shows the communication between four GPUs. GPU i communicates with GPU $i + j$ ($0 \leq i < \#\ of\ GPUs - 1$, where j is a variable for iteration ($1 \leq j < \#\ of\ GPUs$) by `cudaMemcpyPeerAsync`). Each GPU executes the following three steps for a peer GPU; (a) GPU i ($i \mid i + j < \#\ of\ GPUs$) sends data to GPU $i + j$, (b) synchronizes by stream, and (c) GPU $i + j$ sends data to GPU i, then synchronizes. Repeat this process $\#\ of\ GPUs - 1$ times.

6. EVALUATION

Table 2: Evaluation Environment

CPU	Intel Xeon E5-1650 @ 3.20GHz
GPU	NVIDIA Tesla C2050 ×4
Host Memory	16GB
OS	CentOS 6.3
Host Compiler	gcc4.4
CUDA	Toolkit 5.5
CUB library	v1.2.3
Network	10Gb Ethernet ×2

Table 3: The specification of NVIDIA Tesla C2050

CUDA cores	448
Processor Clock	1.15 GHz
Double Precision Performance	515 Gflops
Single Precision Performance	1.03 Tflops
Dedicated memory	3GB GDDR5
Memory Clock	1.5 GHz
Memory Bandwidth	144 GB/sec
System Interface	PCI Express ×16 Gen2

Here, the proposed BFS and Mastrostefano's BFS are implemented on GPU-BOX, a prototype of multi-GPU systems with ExpEther.

For implementing Mastrostefano's algorithm on GPU-BOX, the followings were changed for fair evaluation. First, for Sort-Unique, CUB library was used instead of the old Thrust library that had been used in Mastrostefano's algorithm. The second is that the frontier exchange was implemented with the same method as the proposed one. These modifications improved the performance of Mastrostefano's method for evaluating the effect of traffic reduction by the proposed method on GPU-BOX.

The specification of GPU-BOX used in the evaluation is shown in Table 2 and the specification of NVIDIA Tesla C2050 is shown in Figure 3. The target application is Graph500 benchmark [2] with $edgefactor$, which represents the number of edges in the graph, is fixed to be the default number: 16.

6.1 The reduction of traffic between GPUs

Figure 7 shows the total amount of communication between all GPUs when base BFS and proposed BFS are executed on GPU-BOX. The number of vertex N corresponding to the size of the graph is changed as $N = 2^{18}$, 2^{19}, 2^{20}. As expected, the amount of traffic is increased when the size of graph becomes large and the number of GPUs is increased. The proposed BFS achieved 30-40% reduction independent from the size of the graph, that is, the amount of reduced communication becomes large for the large target graph.

Figure 8 shows the summary of result by NVIDIA profiler of the execution time when graphs with $N = 2^{18}$, 2^{19}, 2^{20} vertices were searched 64 times with 2-4 GPUs. We can see from results of base BFS of Figure 8 that computation time increases about two times and communication time increases about 1.8 times as the size of graph becomes twice. Also, communication time increases more than twice as a GPU is added to the system. It can be seen from Figure 8 that we can reduce 30-40% of communication time by using the proposed method independent of the size of graph. This

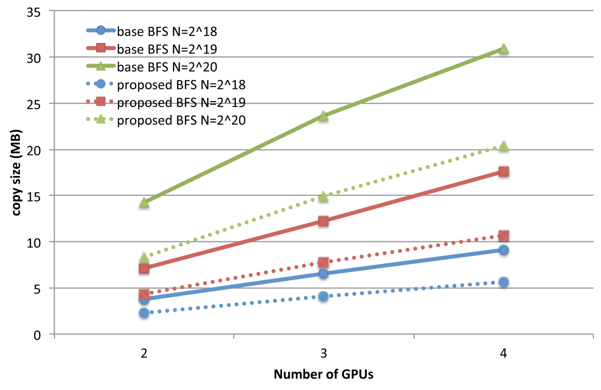


Figure 7: The amount of communication for a BFS

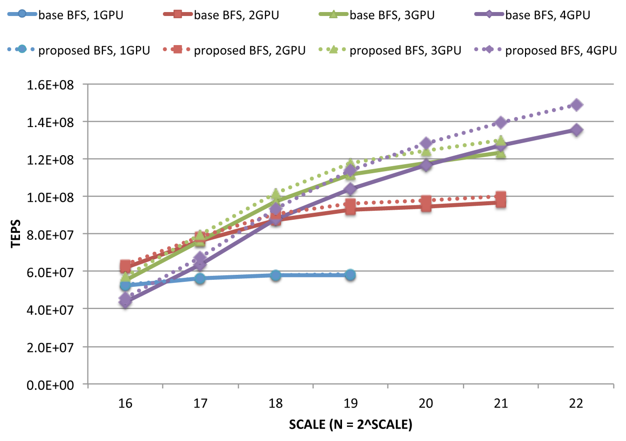


Figure 9: TEPS

fact is also shown in Figure 7. Although the reduction rate of communication time decreases 2-3% when the number of GPU is increased, the total amount of communication time becomes large. Thus, the effect of our method becomes large when the size of the system is increased more than four GPUs.

6.2 The performance of BFS algorithms

Figure 9 shows Traversed Edges Per Second (TEPS), a performance measure in the execution of two BFS algorithms with different number of GPUs. It shows that the proposed method is efficient when the number of GPUs becomes large by reduction of the traffic between GPUs. About 10% performance improvement was achieved.

The figure also shows that the grow of peak performance is degraded with the increasing number of GPUs. It comes from the fact that the size of graph cannot be increased by the limitation of the global memory when the number of GPUs is more than three.

This memory shortage was serious in the proposed method, since the array $Visited$ must be stored in the global memory and $Valid\ flag$ is also stored in the global memory instead of registers for easiness of implementation. Of course, this problem can be solved by using recent GPUs with more amount of memory, but improvement of the algorithm for memory usage reduction is our future work. For example, in

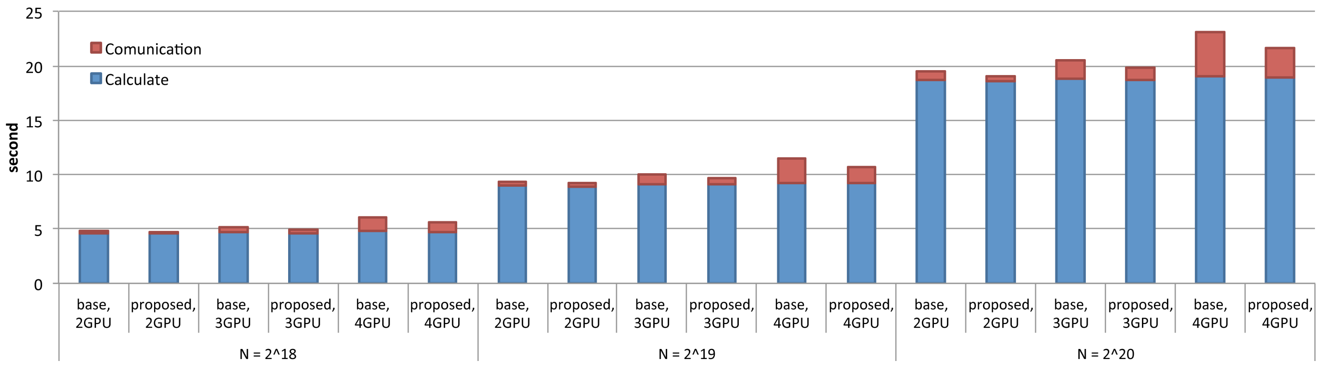


Figure 8: Execution time when 64 BFS

the current implementation the visiting information is represented by byte data, but using the bit-vector, the memory requirement can be one eighth. Application of such memory saving technique is our future work.

7. CONCLUSION

Multi-GPU systems using ExpEther, a virtualization techniques for extending host PCI but to Ethernet is a cost-efficient candidate to accelerate the big data processing in a data center.

In this paper, a parallel BFS algorithm which reduces the traffic between GPUs is proposed and evaluated on GPU-BOX, a prototype of multi-GPU systems with ExpEther, in order to avoid communication bottleneck which will be caused by growing graph size and the number of GPUs. The evaluation result shows that the proposed algorithm can reduce the traffic between GPUs by 30-40% when BFS in Graph500 benchmark is executed with 4 GPUs. Also, the performance was improved by 10% compared with Mastrostefano's algorithm.

Here, only 4 GPUs are evaluated because of the available resource limitation, since GPU-BOX is under development. Evaluation with a system with more number of GPUs is our future work. Also, reducing memory requirement in the implementation is another future work.

8. ADDITIONAL AUTHORS

9. REFERENCES

[1] CUB library. http://nvlabs.github.io/cub/index.html.

[2] Graph 500. http://www.graph500.org/.

[3] Thrust library. http://thrust.github.io/.

[4] V. Agarwal, F. Petrini, D. Pasetto, and D. A. Bader. Scalable graph exploration on multicore processors. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC'10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[5] D. A. Bader and K. Madduri. Designing multithreaded algorithms for breadth-first search and st-connectivity on the cray mta-2. In *Proc. The 35th International Conference on Parallel Processing (ICPP)*.

[6] L. Barnes. Multi-gpu programming, 2013. http://on-demand.gputechconf.com/gtc/2013/presentations/S3465-Multi-GPU-Programming.pdf.

[7] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-mat: A recursive model for graph mining. In *Computer Science Department. Paper 541*.

[8] P. Harish and P. J. Narayanan. Accelerating large graph algorithms on the gpu using cuda. In *Proceedings of the 14th International Conference on High Performance Computing*, HiPC'07, pages 197–208, Berlin, Heidelberg, 2007. Springer-Verlag.

[9] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani. Kronecker graphs: An approach to modeling networks. *J. Mach. Learn. Res.*, 11:985–1042, Mar. 2010.

[10] E. Mastrostefano. *Large Graphs on multi-GPUs*. PhD thesis, Spienza University of Roma, 2013.

[11] D. Merrill, M. Garland, and A. Grimshaw. Scalable gpu graph traversal. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '12, pages 117–128, New York, NY, USA, 2012. ACM.

[12] NEC Corporation. http://www.nec.co.jp.

[13] S. Nomura, T. Nakahama, J. Higuchi, J. Suzuki, T. Yoshikawa, and H. Amano. The multi-gpu system with expether. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, PDPTA'12, July 2012.

[14] J. Suzuki, Y. Hidaka, J. Higuchi, T. Yoshikawa, and A. Iwata. Expressether - ethernet-based virtualization technology for reconfigurable hardware platform. In *Proceedings of the 14th IEEE Symposium on High-Performance Interconnects*, HOTI '06, pages 45–51, Washington, DC, USA, 2006. IEEE Computer Society.

[15] T. Suzumura, K. Ueno, H. Sato, K. Fujisawa, and S. Matsuoka. Performance characteristics of graph500 on large-scale distributed environment. In *Proceedings of the 2011 IEEE International Symposium on Workload Characterization*, IISWC '11, pages 149–158, Washington, DC, USA, 2011. IEEE Computer Society.