Vol.2014-ARC-213 No.10
Vol.2014-HPC-147 No.10
2014/12/9

Efficient Utilization of Multi-level Memory System for Stencil Computation (Unrefereed Workshop Manuscript)

Tianqi Xu[†1†2] Guanghao Jin [†1†2] Toshio Endo [†1†2] Satoshi Matsuoka [†1†2†3]

This paper is to efficiently use the multi-level memory system for stencil computation to enable Tera-Scale computation by single GPU. We build a performance model to explain the relationship between different memories and propose a new algorithm to reduce the communication cost between memories and efficiently use the capacity of memories. We evaluated 7 point stencil computation on the multi-level memory system which includes GPU memory, CPU memory and SSD. The evaluation on the real system shows that our algorithm enables the computation on the 23 times bigger domain than GPU memory capacity as well as achieves 5.5 times higher performance than other optimization methods.

1. Introduction

In many supercomputer systems, it uses multi-level memories to efficiently use the memory space to contain and compute big size simulations. For example, each node of TSUBAME2.5 has *Graphic Processing Unit* (GPU) memory, CPU memory and *Solid State Disk* (SSD) in TSUBAME2.5. In some nodes of TSUBAME2.5, the GPU memory size is 6GB, CPU memory size is 96GB and SSD size is 128GB [14]. GPU has been proved to achieve high performance in some applications like stencil computation. To get a high performance in GPU based supercomputers, common way sends data to GPU side to compute. GPU memory should contain the whole data during the computation which limits the size of applications that can be computed by GPU in common way case. As the capacity of CPU memory or SSD is much bigger than that of GPU memory, common way cannot efficiently use the memory space.

This paper uses stencil computation as an application study case. Stencil computation is one of the base kernels in many scientific and engineering simulations [1]-[3]. When using stencil computation in those simulations, the computation of each point depends on the value of nearby points at each time step. Then, it updates the whole domain for multiple time steps. In some stencil based simulations, it needs to compute bigger domains. Bigger domain means bigger computational area or higher accuracy which is important to simulations like weather forecast.

In 2-level memory system which includes GPU and CPU memory, there are many methods to efficiently use CPU and GPU memory to compute big domain of simulations. The first one is naive method [4]. It separates the domain into sub-domains. Then, it copies each of them to GPU side to compute and copy the result back to CPU side. In stencil case, the sub-domain that can be computed at next time step become smaller as there is data dependence between neighbor points. Because of the data dependence, naive method causes frequent communication problem if it wants to get correct result of each sub-domain. So, Temporal blocking method [5]-[10] copies bigger initial which is bigger than the original sub-domain to GPU side. So, it can compute more time steps on the GPU side while getting the correct result of each sub-domain. As it needs to copy and compute bigger initial, it may cause redundant communication or computation cost.

There is further optimization method for temporal blocking method which is called TBM method [11]-[12] in 2-level memory system. It saves some result on the GPU side when computing current sub-domain. Then, it reuses that result when computing next sub-domain to solve redundancy problem of temporal blocking method.

In this paper, we propose the optimization methods that efficiently use 3-level memory system to compute big domain while maintaining high performance in stencil case. We first evaluate our methods on single node of TSUBAME2.5. To compute bigger domains, we evaluate the optimization method on the other 3-level memory system which has bigger size SSD to evaluate performance in stencil case. The result shows that our optimization method can compute 23 times bigger domain than the GPU memory capacity, and achieve 5.5 times higher performance in 7 point case than other optimization methods.

2. Background

In this section, we introduce some backgrounds for further explanation.

2.1 Stencil computation

Stencil computation is widely applied in scientific and engineering simulations. When it computes each point of the domain, it needs the value of nearby points. We give 7-point stencil example. When computing each point in 7 point stencil case, it needs the value of itself and nearby 6 points. Then, it updates each point of the domain to continue the computation. At next time step, each point of the domain also needs the value of itself and nearby 6 points. If any of the 7 point has not been updated, it cannot compute the point for next time step.

†1 東京工業大学
Tokyo Institute of Technology
†2 JST-CREST
†3 国立情報学研究所
National Institute of Informatics

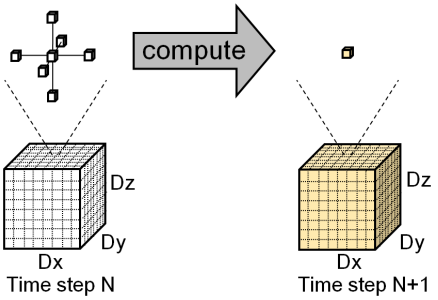

Fig.1. 7 point stencil.

To efficiently compute the domain on GPU side, it uses double buffering method [2] to read initial and save result of the domain.

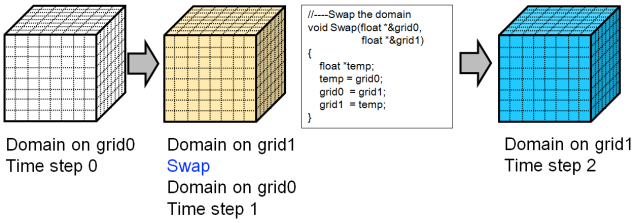

Fig.2. Double buffering method

It allocates two grids on the GPU side. One grid read initial of the domain while the other one save the result of the domain. Then, it swaps the two grids to continue the computation. As double buffering method uses two grids, it consumes two times space on GPU side. To simplify the explanation, we will not identify which grid contains the domain. If the domain is divided into sub-domains, each point of the boundary needs adjacent points which may belong to the other sub-domains. We call these adjacent points on the other sub-domains as ghost boundary [11].

2.2 GPU and CUDA program model

Graphics Processing Units for *general-purpose computation* (GPGPU) is proved to be a high-performance computing device to accelerate a wide variety of scientific and engineering applications [4], [6], [11], [17]-[18]. In November 2006, NVIDIA introduced CUDA™[13], a general purpose parallel computing architecture – with a new parallel programming model and instruction set architecture – that leverages the parallel compute engine in NVIDIA GPUs to solve many complex computational problems in a more efficient way than on a CPU [13]. CUDA comes with a software environment that allows developers to use C as a high-level programming language. Other languages or application programming interfaces are supported, such as CUDA FORTRAN, OpenCL, and DirectCompute.

2D spatial blocking [2], [11] is an efficient way to perform stencil computation by GPU kernel. Since the computation involves a large number of memory accesses, it should reduce access to the global memory. Thus, it should efficiently reuse the data on registers. Moreover, it should invoke sufficiently larger number of threads than that of physical CUDA cores in order to hide latency of memory accesses.

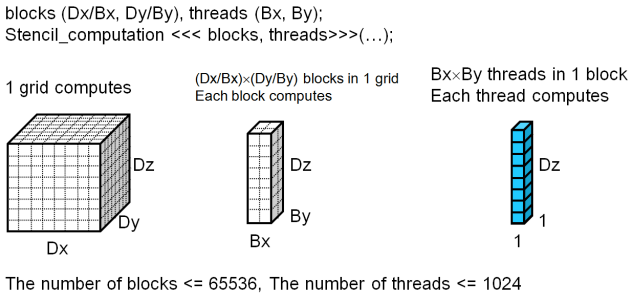

Fig.3. 2D spatial blocking method by GPU kernel

To fulfill those requirements, it designed a kernel function that calculates the points of a computational domain (Dx, Dy, Dz) for a single time step. The kernel function is invoked on a GPU with (Dx/Bx, Dy/By) blocks, each of which has (Bx, By) threads. Bx×By should no more than the number of threads that a single block can contain. It is better to set Bx more than By to improve data locality. It divides the given domain into pieces of size of (Bx, By, Dz) as shown in the Figure 3.

2.3 Multi-level memory system

In this section we introduce two systems which have multi-level memories. Both of them are consist of GPU memory, CPU memory and SSD.

SSD	HP 572071-B21*2 (partly HP 572073-B21*2)
Capacity	120GB(partly 240GB)
RAID	RAID 0
Read Speed	230 MB/s, 460MB/s (RAID 0)
Write Speed	180 MB/s, 360MB/s (RAID 0)

Table1: SSD specification in TSUBAME thin node

TSUBAME supercomputer [14] is developed by Global Scientific Information and Computing Center (GSIC) at Tokyo Institute of Technology. It consists of thin, medium and fat computing nodes which have different system specifications. There are 1408 thin nodes, 24 medium nodes and 10 fat nodes. The nodes mainly consist of two Intel Xeon Westreme-EP 2.9 GHz CPUs and three NVIDIA K20 GPUs. Each GPU of TSUBAME2.5 has 6GB device memory. The CPU side and GPU side are connected by PCI-Express which bandwidth is 8GB/s. Table shows the SSD detail in TSUBAME thin node [14], . SATA is used to connect CPU and SSD in TSUBAME thin node. The bandwidth between CPU side and flash SSD is about 3GB/s, however the actual speed is limited by SSD, the I/O bandwidth of TSUBAME Thin node SSD is around 0.4GB/s,

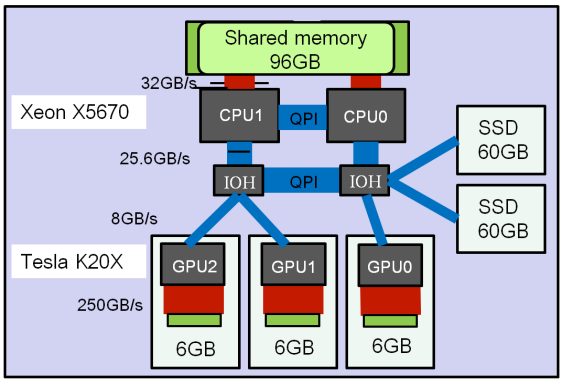

Fig.4. TSUBAME2.5 architecture

The second system also uses K20 GPU and memory size is 6GB. The CPU memory size is 64GB. The size of SSD is 1.2TB. It uses PCI-Express 2.0 x4 electrical x8 physical to connect CPU side and GPU side, bandwidth is read 1.5 GB/s, write 1.3GB/s.

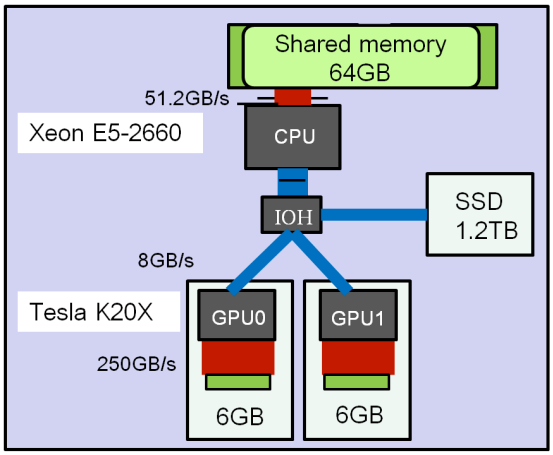

Fig.5. Bigger SSD system

2.4 Page Cache and Page Writeback

Page cache is a mechanism of cache recent disk file in memory. The disk access is extremely lower compared to today's CPU speed, even the SSD. Page cache enables kernel to fulfill the subsequent read request on the same data from memory, without repeated access to disk.

Today's Linux kernel supports page cache mechanism to burst read performance. Linux page cache is dynamic in size, grows larger and larger as more and more I/O requests are issued, consuming any free memory. So more free memory means more data can be buffered and can achieve higher read performance. I/O operation like POSIX read, and read operation in mmap use page cache by default.

Linux kernel has another mechanism called page writeback, to burst write performance. When processes issue a write request, kernel copies data into a buffer, and processes can return to computation without waiting data be finally write back to disk. Subsequent write will update the buffer, read will read from the buffer, to ensure data consistency. I/O operations like POSIX write and write operation in mmap use page writeback by default.

3. Related works

In this section, we introduce some related works. To enable computation, it initializes the domain on the bigger memory (like CPU memory). Then, it separates the domain to sub-domains and sends each sub-domain to smaller memory (like GPU memory) side to compute. Temporal blocking method is to reduce the communication cost between the memories (like CPU and GPU).

3.1 Temporal blocking method in 2-level memory system

Leonardo Mattes [5-6] introduces temporal blocking method to enable big domain computation while reducing the communication cost in 2-level memory system that includes CPU and GPU memory. His solution is to divide the domain into small sub-domains. Then, it copies the initial that is bigger than the sub-domain to GPU side. On the GPU side, it can compute more time steps to reduce the communication times between CPU and GPU.

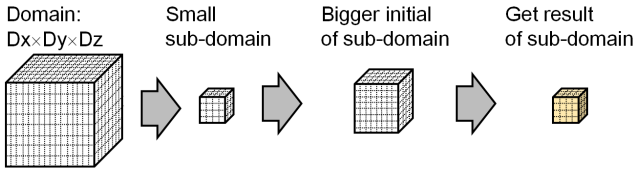

Fig.6. Temporal blocking method to reduce communication cost

His work can avoid communication cost between CPU and GPU. But, his method has to send a bigger initial of sub-domain which causes more communication and computation. It is important to solve this redundancy problem to improve the performance.

In Midorikawa's work [15], he also applies temporal blocking method to efficiently use the CPU memory and SSD to enable big domain computation while reducing the communication cost between CPU and SSD.

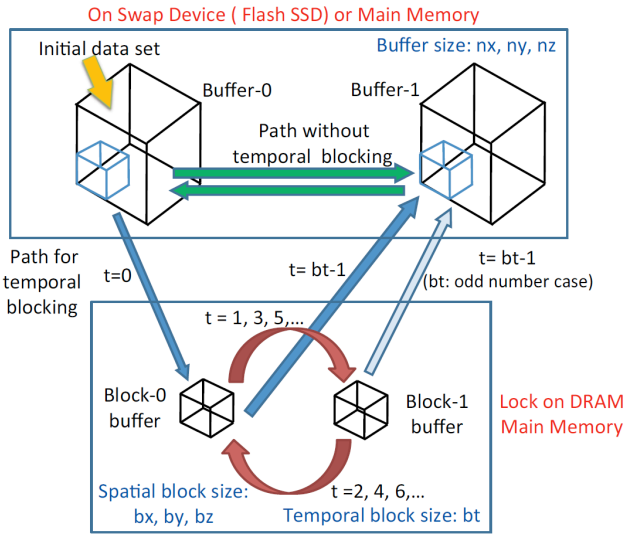

Fig.7. Temporal blocking method between CPU and SSD

In his work, he uses two buffers on SSD side and assigns them to two block buffers on CPU side. He uses tree kinds of methods to enable the computation on big domains that are bigger than memory capacity of CPU.

The first method is called as swap method which allocates swap space to contain some part of the big domain on the SSD side. The second method is mmap method which maps sub-domain to CPU side. The third one is aio method which can parallel communication with computation. We select mmap method to implement optimization methods in this paper as it is easier to implement.

3.2 Optimization methods in 2-level memory system

To solve the redundancy problem of temporal blocking method, Jin [11-12] proposes optimization methods for temporal blocking method. The main idea of this method is to save some XY-planes of current sub-domain and reuse the XY-planes when computing next sub-domain. To simplify the explanation, it abstracts each XY-plane of the domain as a square in Figure 8.

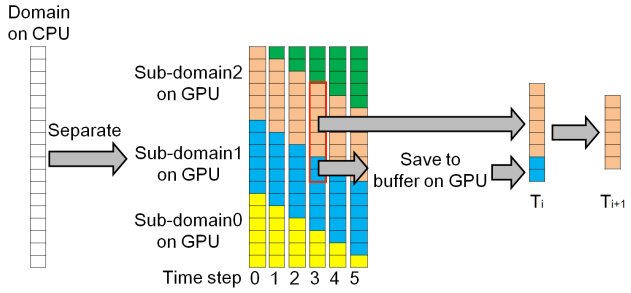

Fig.8. 1D-TBM method

It uses 1D decomposition method to separates the domain to sub-domains. In his work, 1D-TBM method saves the reused XY-planes on the GPU side.

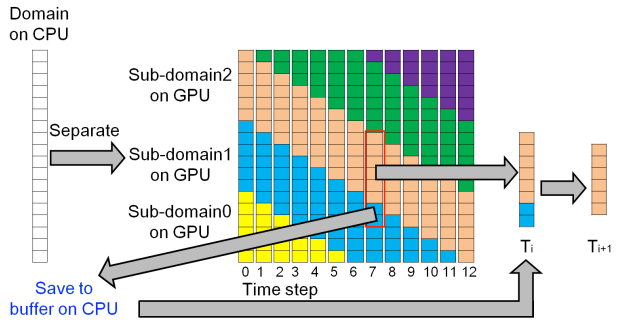

Fig.9. 1D-C-ETBM method

1D-C-ETBM method saves the reused part on the CPU side as Figure 9 shows. Then, it overlaps the communication of reused XY-planes with the computation of the sub-domain. It also allocates additional sub-domains on the GPU side to enable more temporal blocking times.

Both of the optimization methods do not degrade computation accuracy as it stores some result of current sub-domain and reuse it when computing next sub-domain to solve redundancy problem. Therefore, other stencil forms can adopt this method. It only copies un-overlapped XY-planes to reduce communication cost and computes only un-overlapped XY-planes with reused XY-planes to reduce computation cost. 1D-TBM consumes more space on GPU side and 1D-C-ETBM methods consumes more space on CPU side.

4. Supporting large domain for 3-level memories

In this chapter, we introduce how to enable big domain computation on 3-level memory system for stencil computation, and we introduce our implementation of proposal system.

4.1 Decomposition methods

To enable bigger domain computation in 3-level memory system than in 2-level memory system, we initialize the domain on the SSD which has biggest capacity in 3-level memory system. We use two methods to implement the big domain decomposition. Both of the methods use 1D decomposition to separate domain or sub-domain.

The first decomposition method separates domain to sub-domains. Then, it copies sub-domain to CPU side. So, the CPU memory should contain the sub-domain. On the CPU side, it also separates sub-domain to parts and send each part to GPU side compute. So, the GPU memory should contain each part. We call this method as double decomposition method.

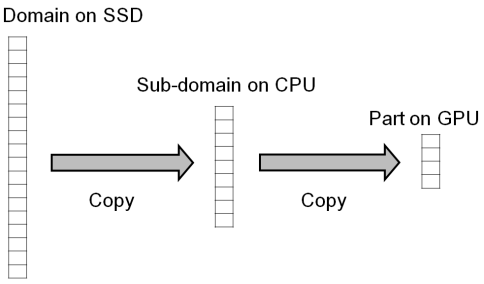

Fig.10. Double decomposition method

The second implementation method separates domain to sub-domains. Then, it sends each sub-domain to CPU side. Different from double decomposition method, we set sub-domain and part size are the same. So, sub-domain should also be contained by the GPU memory. We call this implementation method as single decomposition method.

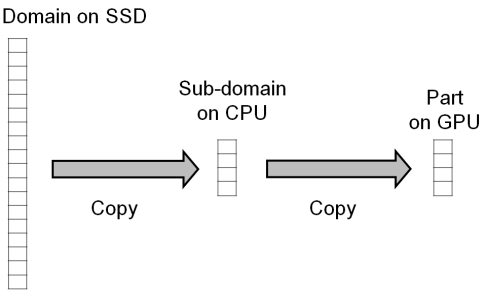

Fig.11. Single decompostion method

We combine these 2 decomposition methods with 1D-C-ETBM method that has been introduced in section 3.2. We set temporal blocking times between SSD and CPU equals to that between CPU and GPU. We call 1D-C-ETBM method with single or double decomposition methods as 1D-SC-ETBM or 1D-DC-ETBM method.

4.2 Performance model

In this section, we analyze the performance model of the optimization methods in 3-level memory system. We set TBS as the temporal blocking times. By the introduction of section 4.1, we can get the communication of the optimization methods at each TBS time steps.

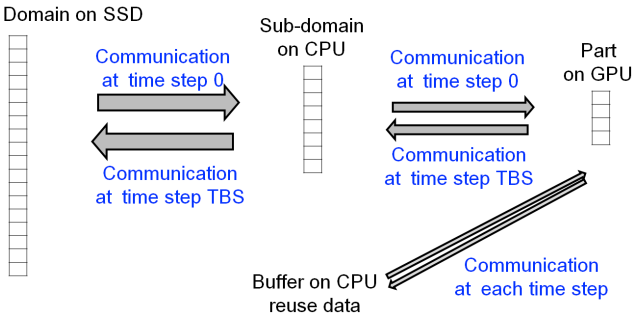

Fig.12. Communication of the optimization methods

We set T_{TBS} as the execution time of TBS time steps. $T_{C2G+G2C}$ is the communication time between GPU and CPU. $T_{C2S+S2C}$ is the communication time between CPU and SSD. $T_{Computation}$ is the computation time on GPU side. For memory capacity, we have the formulae for 1D-SC-ETBM and 1D-DC-ETBM as below:

Sizeof (Domain)×2 <= SSD capacity,
Sizeof (Sub-domain)×2 + Sizeof(Buffer)
<= CPU memory capacity,
Sizeof (Part)×2 <= GPU memory capacity, (1)

For the execution time, we can get below formula:

$T_{TBS} = T_{C2S+S2C}$(Sub-domains)+ $T_{C2G+G2C}$(Parts)
+ Max($T_{Computation}$ (Parts), $T_{C2G+G2C}$(Buffer)) (2)

For the performance we can get below formula:

Performance=Computation (Parts)/T_{TBS} (3)

The $T_{C2S+S2C}$(Sub-domains) and $T_{C2G+G2C}$(Parts) only depends on the size of Sub-domains and Parts. Both of them can be reduced as the TBS is increased. The communication cost between buffer and GPU depends on the size of reusing XY-planes at each time step. In the case that the computation cost of sub-domain is bigger than the communication cost of reusing XY-planes, the overhead cost of reusing XY-planes can be covered.

4.3 Implementation

In this section, we analyze the implementation of communication between SSD and GPU. As the section 4.2 introduced, the communication time between CPU and GPU can be presented as $T_{C2S+S2C}$(Sub-domains)+ $T_{C2G+G2C}$(Parts). In mmap method case, we separate the whole domain to sub-domains and map each sub-domain to index on CPU side in floating computation case as below function shows.

```
subdomain = reinterpret_cast<float*>(mmap(NULL,
Size*sizeof(float), PROT_READ | PROT_WRITE,
MAP_SHARED, File0, offset0));
```

Then, we send the initial from SSD to GPU and send result from GPU to SSD by the function of CUDA function as below.

```
cudaMemcpy(&subdomain[…],&part[…],…);
cudaMemcpy(&part[…],&subdomain[…],…);
```

So, there is no actual code for sending data from SSD to CPU or from CPU to SSD. As the process of mmap method, as we introduced in Section 2, In Linux system, the file cache accelerates many accesses to files on non volatile storage, Read and write data can be cached in free memory by Linux kernel, kernel can fulfill data access request on these data from memory, and thus hide the low bandwidth of disk(Section 2.4).

Such cache may cause data consistency problem while multiple machines access to a same shared storage. However, as our optimization methods only reads and writes un-overlapped sequential sub-domains on a single machine, there is no reading or writing of same data concurrently from other machines. So, there is no need to worry about data consistency, and reading from or writing to cache is correct in our optimization methods case.

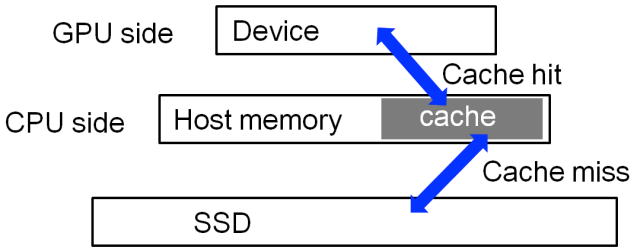

Fig.13. The process of the Linux cache

The communication between GPU memory and SSD can partially accelerated by the cache. So, the data access between GPU memory and SSD only happens when the required data is not in the cache. So, the single decomposition method and double decomposition method have same communication cost. As it easier to implement single decomposition method, we choose single decomposition method to implement optimization methods. For the execution time, we can get below formula:

$T_{TBS} = T_{S2G+G2S}$(Parts)
+ Max($T_{Computation}$ (Parts), $T_{C2G+G2C}$(Buffer)) (4)

We can get $T_{S2G+G2S}$(Parts) < $T_{S2C+C2G+G2C+C2S}$(Parts) by the effect of page cache. As our experiment, it can speed up 2 ~ 3 times in sequential reading or writing in stencil case in real systems. As we introduced, there is no same reading or writing from different requirement at the same time. So, the data on the cache correctly presents the data on the SSD.

5. Evaluation

5.1 Evaluation on single node of TSUBAME2.5

We evaluate the optimization methods on single node of TSUBAME2.5. The GPU memory size is 6GB; CPU memory size is 96GB; SSD size is 128GB.

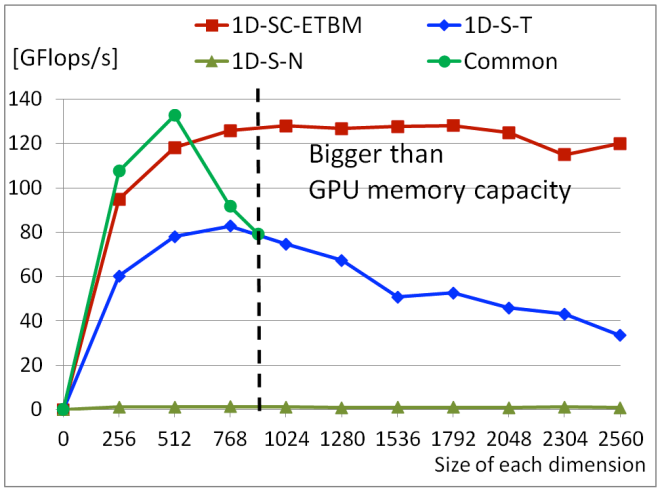

Fig.14. 7-point stencil, optimization method vs existing methods

As we can see in Figure 14, our optimization method has 2.2 times better performance than the existing methods. The optimization methods can compute 23 times bigger domain than the GPU memory capacity while maintaining high performance.

5.2 Evaluation on bigger SSD system

We evaluate the optimization methods on the system that has bigger SSD capacity. The GPU memory size is 6GB; CPU memory size is 64GB; SSD size is 1.2TB.

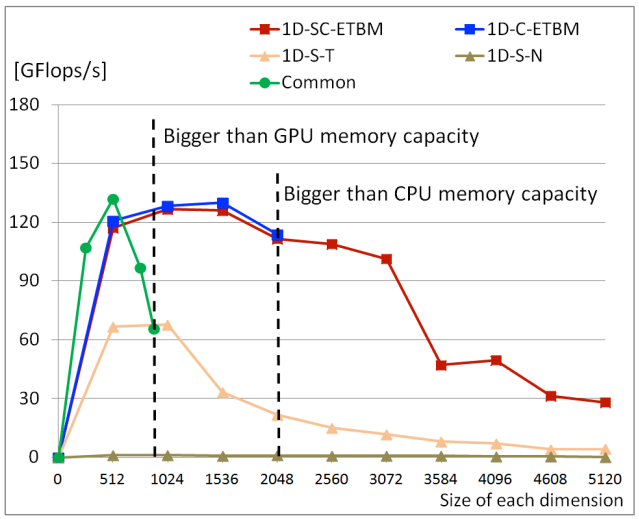

Fig.15. 19 point stencil, 2D-1D-TBMR vs others

As we can see in Figure 15, our optimization method has 5.5 times better performance than the existing methods. The optimization methods can compute 186 times bigger domain than the GPU memory capacity and 15.6 times bigger domain than the CPU memory capacity while maintaining high performance. The performance falls as the domain continues to grow. As it reusing 2 XY-planes in 7-point stencil case, the reusing part becomes bigger as the domain grows. So, the overhead by the communication grows which cannot be covered by the computation cost.

6. Conclusion

In this paper, we proposed a series of new optimization methods which enable the computations on the domain that is bigger than the memory capacity of GPU and CPU while maintaining high performance on different GPU cluster. The result shows that our optimization method achieves 5.5 times higher performance in 7 point case than other optimization methods. We also confirmed the decomposition methods in the case of using multi-level memory system and introduced the implement details.

As a future work, we will try to utilize our methods on multiple nodes of GPU cluster. Further research will focus on full utilization multi-level memory of GPU cluster and try to lower programming difficulty by using tools like Physis [16].

Acknowledgments This work was supported by "Software Technology that Deals with Deeper Memory Hierarchy in Post-peta scale Era", JST-CREST and JST, CREST (Research Area: Advanced Core Technologies for Big Data Integration).

Reference

[1] Christen, M. , Schenk, O., Yifeng Cui, “PATUS for Convenient High-Performance Stencils: Evaluation in Earthquake Simulations”, In Proceedings of IEEE/ACM International Conference on Supercomputing (SC12), pp. 1-10, 2012.

[2] Takashi Shimokawabe, Takayuki Aoki, Tomohiro Takaki, Akinori Yamanaka, Akira Nukada, Toshio Endo, Naoya Maruyama, and Satoshi Matsuoka, “Peta-scale phase-field simulation for dendritic solidification on the TSUBAME 2.0 supercomputer,”In Proceedings of IEEE/ACM International Conference on Supercomputing (SC11), pp. 1-11, 2011.

[3] K. Datta, M. Murphy, V. Volkov, S. Williams,J. Carter, L. Oliker, D. Patterson, J. Shalf, and K. Yelick, “Stencil computation optimization and autotuning on state-of-the-art multicore architectures,”In Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC08), pp. 1–12, 2008.

[4] Toshio Endo and Satoshi Matsuoka, “Massive supercomputing coping with heterogeneity of modern accelerators,” In Proceedings of IEEE International Parallel & Distributed Processing Symposium (IPDPS 2008), 10pages, April 2008.

[5] Leonardo Mattes and Sergio Kofuji, “Overcoming the GPU memory limitation on FDTDthrough the use of overlappingsubgrids,” ICMMT, pp.1536 – 1539, 2010.

[6] Leonardo Mattes and Sergio Kofuji, “The use of overlapping subgrids to accelerate the FDTD on GPU devices,”Radar Conference, pp. 807 – 810, 2010.

[7] Takeshi Minami, Takeshi Iwashita, Yasuhito Takahashi, and Hiroshi Nakashima, “Cache-aware performance improvement of FDTD kernel,” IPSJ SIG Technical Report, vol.2010-HPC-124 No.5, 7pages, 2010.

[8] M. Wittmann, G. Hager, and G. Wellein, “Multicore-aware parallel temporal blocking of stencil codes for shared and distributed memory,” Workshop on Large-Scale Parallel Processing (LSPP10), in conjunction with IEEE IPDPS2010, 7pages, April 2010.

[9] Gerhard Wellein, Georg Hager, Thomas Zeiser, Markus Wittmann and Holger Fehske, “Efficient temporal blocking for stencil computations by multicore-aware wavefront parallelization,” Computer Software and Applications Conference, vol.1, pp. 579 – 586, 2009.

[10] Tomoki Kawamura,Naoya Maruyama, and Satoshi Matsuoka, “Performance model for automatic optimization of communication in data-parallel stencil computations,” IPSJ SIG Technical Report , vol.2012-HPC-135, 8pages, 2012

[11] Guanghao Jin, Toshio Endo and Satoshi Matsuoka, “A multi-level optimization method for stencil computation on the domain that is bigger than memory capacity of GPU,” AsHES/IPDPS 2013, 2013.

[12] Guanghao Jin, Toshio Endo, Satoshi Matsuoka. A Parallel Optimization Method for Stencil Computation on the Domain that is Bigger than Memory Capacity of GPUs . In Proceedings of IEEE Cluster 2013, Indianapolis, September 2013.

[13] NVIDIA CUDA C Programming Guide, Version 4.0, 2011.

[14] TSUBAME 2.5 http://tsubame.gsic.titech.ac.jp/en/

[15] Hiroko Midorikawa, Tan Hideyuki, Toshio Endo“An Evaluation of the Potential of Flash SSD as Large and Slow Memory for Stencil Computations”, IEEE The 12th International Conference on High Performance Computing & Simulation, HPC2014, 2014

[16] Naoya Maruyama, Tatsuo Nomura, Kento Sato, and Satoshi Matsuoka, “Physis: An implicitly-parallel programming model for stencil computing on large-scale GPU-accelerated supercomputers,” IEEE SC11,2011.

[17] Myungho Lee, Jin-hong Jeon , Joonsuk Kim and Joonhyun Song, “Scalable and Parallel Implementation of a Financial Application on a GPU: With Focus on Out-of-Core Case” Computer and Information Technology (CIT), 2010

[18] Bahri Haythem, Sayadi Fatma, Chouchene Marwa, Tourki Rached, “Contribution to the implementation of computer vision application on a GPU”, Control, Decision and Information Technologies (CoDIT), 2013 International Conference