Vol.2014-HPC-143 No.33
2014/3/4

Multi-level Temporal Blocking for Stencil Computation for Memory Hierarchy on TSUBAME2.5

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

The domain of the stencil computation is limited by the memory capacity of GPUs on a GPU cluster. As the domain grows to cope with higher accuracy requirements, more GPUs need to be employed to extend the memory capacity. In this paper, we propose new methods which apply temporal blocking method to device memory and registers of a set of GPUs to allow computations on the domain that is bigger than the memory capacity of GPUs while maintaining high performance on TSUBAME2.5. We also analyze the parameters and performance differences between TSUBAME2.0 and TSUBAME2.5 to apply our methods to wide range GPU clusters.

1. Introduction

Stencil computation is one of the base kernels in various scientific and engineering simulations [1], [2]. 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 simulations, it needs to compute bigger domains. Bigger domain means bigger computational area or higher accuracy which is important to simulations like weather forecast.

The common way [2], [3] that uses GPU cluster to compute bigger computational domain is to separate the domain into sub-domains. Then, it employs multiple GPUs to compute the sub-domains. In that case, it assigns each sub-domain to each GPU. So, the domain size is limited by the memory capacity of GPUs. To efficiently use the GPUs, it should enable the computation on the domain that is bigger than the memory capacity of GPUs.

There are existing methods that enable the computation on the bigger domains. The first one is naive method [4]. It separates the domain into sub-domains. It assigns some sub-domains to each GPU. Then, it copies each of them to GPU to compute and copy the result back. In stencil case, it can only compute 1 time step because of the boundary constraint. So, it causes frequent communication in stencil case. Temporal blocking method [5], [6], [7], [8], [9], [10] can solve the frequent communication problem. It also separates the domain into sub-domains and assigns some sub-domains to each GPU. When it copies each of them to GPU side, it copies more ghost boundaries with the sub-domain to compute multiple time steps on GPU side. So, it can reduce the communication cost by reducing communication times. But, the more ghost boundaries cause redundant cost which degrades the performance.

In our previous papers [11], [12], we proposed the optimization methods that use the buffer on GPU to avoid the redundant cost. We also applied the optimization methods to multi-GPU case. The evaluation on TSUBAME2.0 shows that our optimization methods achieve higher performance than existing optimization methods.

In this paper, we propose the optimization methods that efficiently use the registers of GPU to improve the performance in kernel level. Furthermore, we analyze the differences of performance and parameters on the TSUBAME2.0 and TSUBAME2.5 to apply those optimization methods on wide range GPU clusters.

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 as one of the base kernels. When it computes each point, it needs the value of nearby points. We give two examples of stencil computation. When computing each point in 7 point stencil case, it needs the value of itself and nearby 6 points. When computing each point in 19 point stencil case, it needs the value of itself and nearby 18 points. In the 3×3×3 area, 19 point stencil uses more nearby points than 7 point stencil.

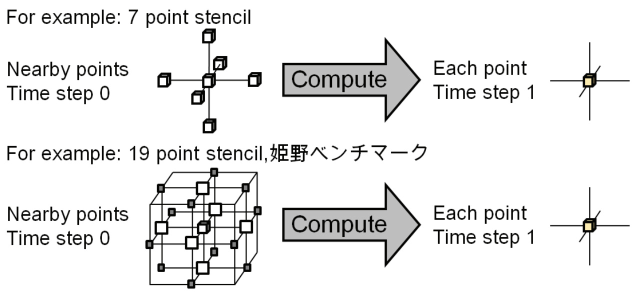
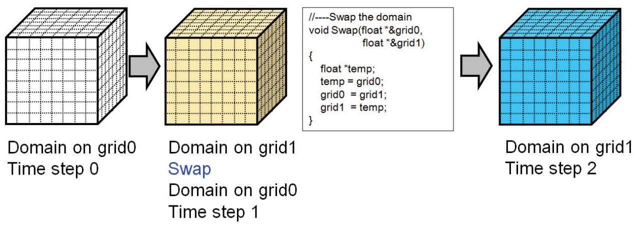


Fig.1. 7 point, 19 point stencil.

To efficiently compute the whole domain on GPU, it uses double buffering methods to read initial and save result of the domain.



```
//----Swap the domain
void Swap(float *&grid0,
          float *&grid1)
{
   float *temp;
   temp = grid0;
   grid0  = grid1;
   grid1  = temp;
}
```

Fig.2. Double buffering method

†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

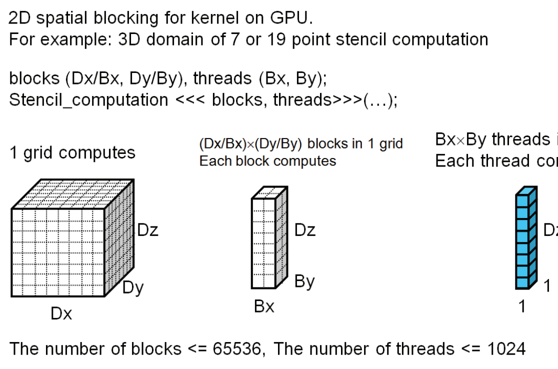
It allocates two grids. One grid read initial of the domain while the other one save the result of the domain. Then, it swaps the two grids to continue the computation. As double buffering method uses two grids, it consumes two times space on GPU side. To simplify the explanation, we will not identify which grid contains the domain. Double buffering is the method that how to read the initial and save the result of 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. Next section, we will introduce how to implement the stencil computation by using CUDA program model on GPU.

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 simulations. In November 2006, NVIDIA introduced CUDA™, 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] is an efficient way to perform stencil computation on GPU. 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.


2D spatial blocking for kernel on GPU.
For example: 3D domain of 7 or 19 point stencil computation

```
blocks (Dx/Bx, Dy/By), threads (Bx, By);
Stencil_computation <<< blocks, threads>>>(…);
```

The number of blocks <= 65536, The number of threads <= 1024

Fig.3. 2D spatial blocking method

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 upper figure.

2.3 TSUBAME system

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 node which have different system specifications. The nodes are interconnected by dual QDR InfinitBand network with a full bisection-bandwidth fat-tree topology. There are 1408 thin nodes, 24 medium nodes and 10 fat nodes. Each thin node has 54GB host memory. The nodes mainly consist of two Intel Xeon Westreme-EP 2.9 GHz CPUs and three NVIDIA GPUs. Each GPU of TSUBAME2.0 has 3GB device memory.

Recently, TSUBME2.0 is upgraded to TSUBAME2.5. TSUBAME2.5 upgraded TSUBAME2.0 by exchanging the GPU series of Tesla M2050 to Tesla K20X to improve the throughput in the fall of 2013. The CPUs and the connections between nodes and inside nodes remain same. The Tesla M2050 is Fermi and Tesla K20X is Kepler in NVIDIA product series.

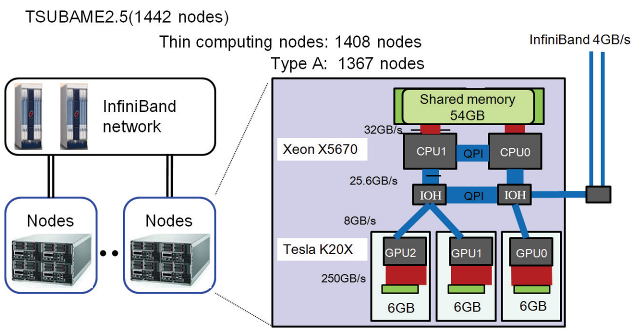


Fig.4. TSUBAME2.5 architecture

In Fermi, the shared memory and the L1 cache share the same physical on-chip storage, and a split of 48 KB shared memory / 16 KB L1 cache. Kepler continues this pattern and introduces an additional setting of 32 KB shared memory / 32 KB L1 cache, the use of which may benefit L1 hit rate in kernels that need more than 16 KB but less than 48 KB of shared memory per multiprocessor. L1 caching in Kepler GPUs is reserved only for local memory accesses, such as register spills and stack data. In Kepler GPUs, global loads are only cached in L2 cache. In Fermi GPUs, the global loads are cached in L1 and L2 caches which mean more data locality in caches to improve data hit rate. Kepler has more computing units which are called SMs and has bigger L2 cache. So, Kepler ensures performance improvement in most application cases.

3. Related works

In this section, we introduce some related works. Temporal blocking method is to reduce the communication cost between CPU and GPU. Temporal blocking method also can efficiently use the registers of GPU to improve the performance of kernels.

3.1 Temporal blocking method

Leonardo Mattes [5], [6] proposed optimization method to overcome the GPU memory limitation by overlapping sub-domains. 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.

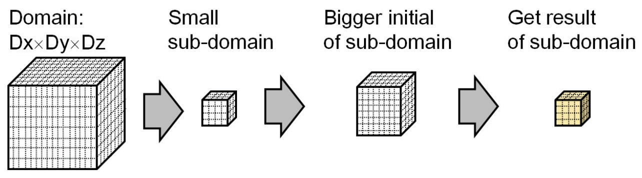


Fig.5. Temporal blocking method to reduce communication cost

His work can avoid communication cost between CPU and GPU and can overcome the memory limitation of GPU. Yet, as he mentioned, his method has to initialize a bigger initial for sub-domain which cause more communication and computation. We call his method as temporal blocking method and the more communication and computation is called as redundancy problem. It is important to solve the redundancy problem to improve the performance.

3.2 Temporal blocking method for registers of GPU

In Wai Teng Tang's paper [15], he explores a different data access strategy to improve the performance of stencil kernels. In particular, he proposed a temporal blocking method to reuse the data on the registers. With the appropriate tuning, his method is able to obtain optimal parameters on different GPU platforms, and is able to achieve better performance than prior methods.

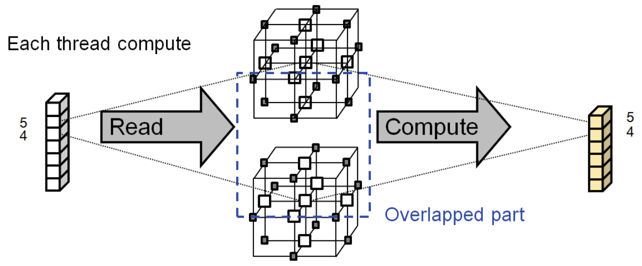


Fig.6. Overlapped reading between two points on each thread.

As we can see in the upper figure, it needs to read overlapped part when computing sequential points by each thread. His method tries to save overlapped part on the register of each thread and reuse when computing next point as below:

```
Example of 7 point stencil of floating computation:
__global__   void kernel (grid0, grid1,…)
{
   …
   int jx = blockDim.x * blockIdx.x + threadIdx.x;
   int jy = blockDim.y * blockIdx.y + threadIdx.y;
   float    tempo,tempc,tempb,tempt;
   …
   for(jz = 0;jz < Dz; jz++)
   {
     ji = Dx*Dy*jz + Dx*jy + jx;
     je = ji+1;    jw = ji-1;    jn = ji+Dx;    js =ji-Dx;
     tempt    = grid0[ji];
     grid1[ji]= tempo+cc*tempc+ct*tempt+ cb*tempb;
     tempb    = tempc;
     tempc    = tempt;
     tempo = ce*grid0[je] + cw*grid0[jw]
           + cn*grid0[jn] + cs*grid0[js];
   }
   …
}
```

As we can see, it only needs to allocate additional 4 points on the register of each thread. Then, it can reuse those points to contain the overlapped part and continue the computation. It can easily improve the performance. But, it should allocate related number of points for different stencil which also consumes the space of registers.

4. Supporting large domain on single node

In this chapter, we first focus on the optimization methods on single node of GPU cluster. We only use 1 GPU on each node. We select 7 point and 19 point on 3D domains of floating computation to implement and explain the optimization methods. We assume the points of the domain are stored in X, Y, Z order. To simplify the implementation, we only use 1D decomposition method (decompose the domain by Z dimension) in single node case. To simplify the explanation, we set number to each XY-plane of whole domain as below:

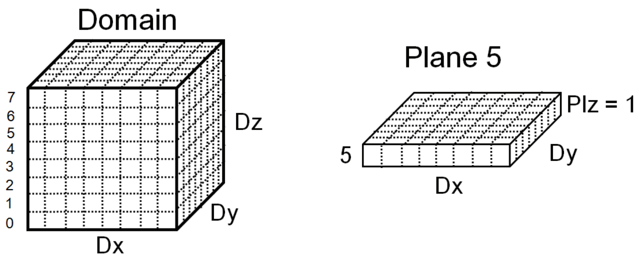


Fig.7. Set number to each XY-plane of whole domain.

Here, Dx, Dy, Dz are the size of X, Y, Z dimension. Plz is the dimension size of each XY-plane.

4.1 Existing method: 1D-N, 1D-T

1D-N is the 1D decomposition version of naive method. 1D-T is the 1D decomposition version of temporal blocking method.

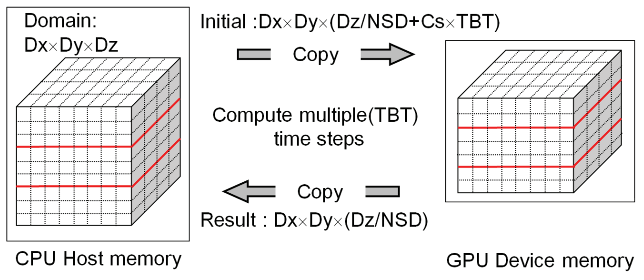


Fig.8. The process of 1D-N and 1D-T

As the upper figure shows, 1D-N or 1D-T separates the domain into sub-domains. Then, it copies each sub-domain to the GPU side. Here, TBT stands for temporal blocking times, NSD stands for the number of sub-domains, Cs stands for the data dependence of boundary. In 7 point or 19 point stencil case, Cs equals to 2.

In 1D-N case, TBT equals to 1. So, it copies sub-domain with ghost boundary to GPU side and compute 1 time step. Then, it copies the result back. It only consumes Dx×Dy×(Dz/NSD+Cs)×2 space on the GPU side, but it causes frequent communication between CPU and GPU.

In 1D-T case, TBT can be bigger than 1. It copies sub-domain with more ghost boundaries to GPU side and compute multiple time steps. Then, it copies the result back. It can reduce the communication cost between CPU and GPU. But, it consumes more space which equals to Dx×Dy×(Dz/NSD+Cs×TBT)×2 on GPU side. Also, the more ghost boundaries may cause redundant communication and computation cost.

4.2 Previous contribution: 1D-TBM

In this section, we introduce the previous contribution that can reduce the communication cost and solve the redundancy problem of 1D-T.

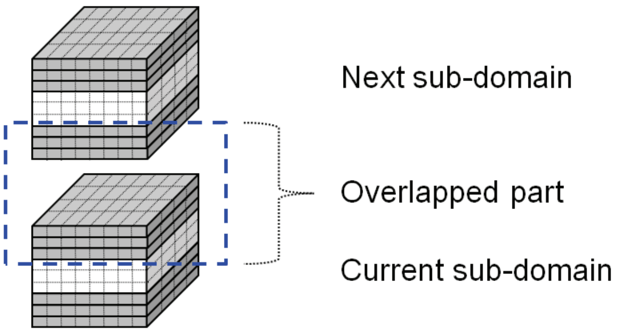


Fig.9. Overlapped part between two sub-domains.

In 1D-T case, we found that there is overlapped part between sequential sub-domains as the upper figure shows. To solve the redundancy problem, we try to save the overlapped part and reuse when computing next sub-domain.

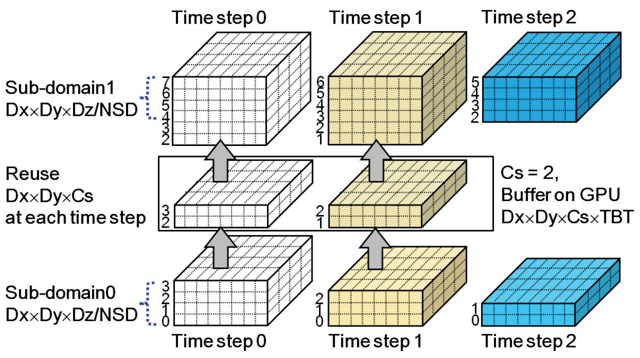


Fig.10. The process of 1D-TBM

When computing sub-domain0 on GPU side, it saves Cs = 2 XY-planes to buffer on GPU at each time step as the upper figure shows. When computing sub-domain1, it reuses the XY-planes (size equals to Dx×Dy×Cs) at each time step. It should allocate Dx×Dy×Cs×TBT size of buffer on the GPU side. We call this method as buffer-copy method.

When it saves the result to the grid, it also shifts the result as the upper figure shows. So, the space for each grid equals to Dx×Dy×(Dz/NSD+Cs). We call this as memory-saving method. It can save Dx×Dy×(Cs×TBT-Cs) space than 1D-T method. We call 1D-T method with buffer-copy method and memory-saving method as 1D-TBM method.

4.3 Discussion

In this section, we discuss the performance model of 1D-T and 1D-TBM. We build the model by counting the amount of communication and computation as below:

1D-T, Minimize T_{TBT}×Iteration/TBT
T_{TBT}
$=T_{C2G}$(domain)$+T_{C2G}$(ghost boundaries)$+T_{G2C}$(domain)
$+ T_{computation}$ (domain)$+T_{computation}$(ghost boundaries)

1D-TBM, Minimize T_{TBT}×Iteration/TBT
$T_{TBT}=T_{C2G}$(domain)$+T_{G2C}$(domain)$+T_{computation}$ (domain)

To achieve higher performance, it should try to minimize T_{TBT}×Iteration/TBT. We evaluate the actual time on the single node of TSUBAME2.5 to compare with the model. The domain is 640×640×640; the number of sub-domains is 2.

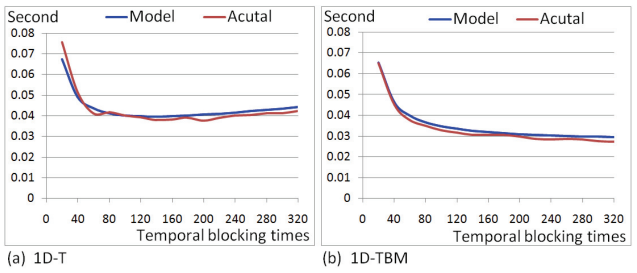


Fig.11. The model and actual time on TSUBAME2.5

As we can see in the upper figure, the models are similar to the actual evaluations which can be used to decide the parameters. As the temporal blocking times grow, T_{TBT}×Iteration/TBT degrades in 1D-TBM case. It shows that it can get higher performance as the temporal blocking times grow. To implement 1D-TBM method, it should allocate 2 grids and 1 buffer. We can get max TBT in 1D-TBM case as below:

Dx×Dy×(Dz/NSD+Cs)×2+ Dx×Dy×Cs×TBT
<=GPU memory capacity
TBT <= Dz/NSD, Get max TBT.

4.4 Contribution: 1D-TBMR

In this section, we propose our optimization method which is named as 1D-TBMR. As we introduced in the upper section, temporal blocking method is used to reduce the communication cost between CPU and GPU. To improve the performance of the kernel, it applies the temporal blocking method for the registers of GPU to 1D-TBM method. So, we can get 1D-TBMR method. We give the example of the kernel of 7 point stencil in 1D-TBMR method case as below:

Example of 7 point stencil of floating computation:

```
__global__   void   kernel(grid0, grid1,…)
{
   …
   int jx = blockDim.x * blockIdx.x + threadIdx.x;
   int jy = blockDim.y * blockIdx.y + threadIdx.y;
   float   tempo,tempc,tempb,tempt;
   for(jz = 0;jz < Dz; jz++)
   {
      ji = Dx*Dy*jz + Dx*jy + jx;
      je = ji+1;    jw = ji-1;    jn = ji+Dx;    js =ji-Dx;
      tempt    = grid0[ji];
      grid1[jt]= tempo+cc*tempc+ct*tempt+ cb*tempb;
      tempb    = tempc;
      tempc    = tempt;
      tempo = ce*grid0[je] + cw*grid0[jw]
                  + cn*grid0[jn] + cs*grid0[js];
   }
   …
}
```

As the upper example shows, it improves the performance of the kernel by saving some points to the register of each thread and reusing them. Also, we can see that memory-saving method saves the result to grid1[jt] instead of grid1[ji]. So, it can shift the result to save space and does not affect the performance of the kernel. So, we can give the whole process of 1D-TBMR method.

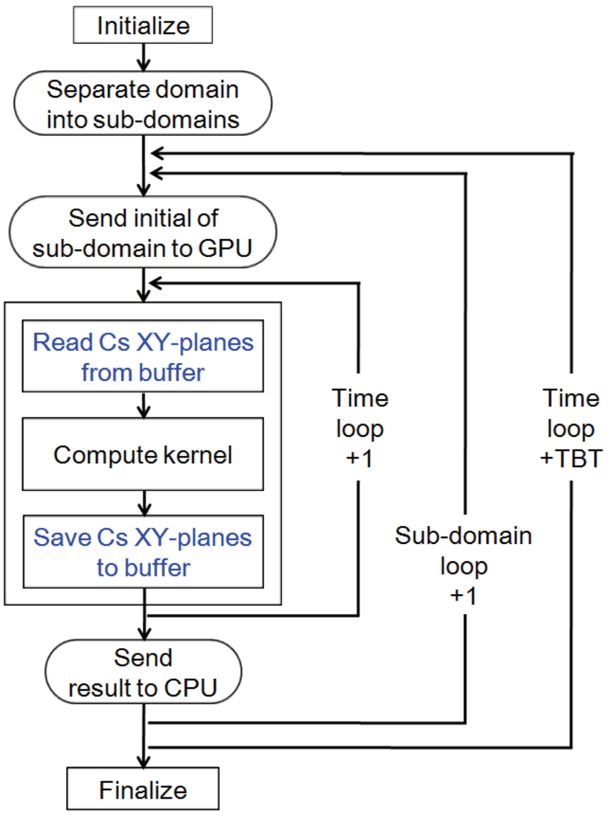

Initialize
Separate domain into sub-domains
Send initial of sub-domain to GPU
Read Cs XY-planes from buffer
Compute kernel
Save Cs XY-planes to buffer
Time loop +1
Time loop +TBT
Sub-domain loop +1
Send result to CPU
Finalize

Fig.12. The process of 1D-TBMR

As it only improves the performance of kernel of 1D-TBM, the parameters are decided by the same way as section 4.3 introduced.

4.5 Contribution: 1D-P-TBMR

The temporal blocking times are limited as section 4.3 explains. To improve the performance, 1D-P-TBMR method tries to overlap the communication with the computation. It first allocates 2 more grids on the GPU side. One grid contains the result of former sub-domain; the other grid accepts the initial of the next sub-domain. Both of them perform the communication when computing current sub-domain as the below figure shows.

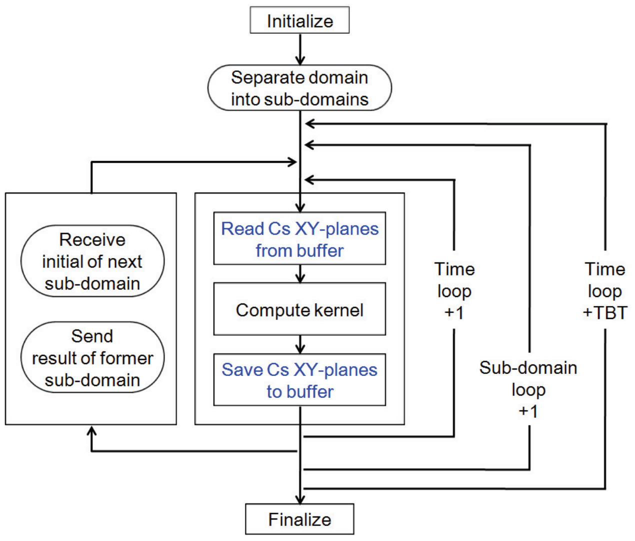

Initialize
Separate domain into sub-domains
Receive initial of next sub-domain
Send result of former sub-domain
Read Cs XY-planes from buffer
Compute kernel
Save Cs XY-planes to buffer
Time loop +1
Time loop +TBT
Sub-domain loop +1
Finalize

Fig.13. The process of 1D-P-TBMR

5. Supporting large domain on multiple nodes

In this chapter, we introduce how to apply optimization methods to multiple nodes. We also select 7 point and 19 point stencil of floating computation. We also assume the points of the domain are stored in X, Y, Z order.

5.1 Previous contribution: Decomposition method

As we introduced in the upper chapter, our optimization methods achieves scalability in one dimension. In 7 point or 19 point case, it computes 3D domain. To achieve scalability in 3D domain, it should decompose the domain by other two dimensions among the nodes. Then, it separates each sub-domain into smaller parts and applies the 1D optimization methods between sub-domain and parts.

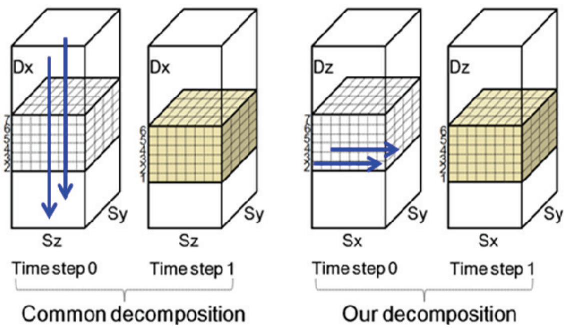


Fig.14. Common decomposition VS our decomposition

Sx, Sy, Sz is the size of each dimension in the upper figure. The common 2D decomposition method is to separate the domain by Y and Z dimensions. As it saves the domain points in X, Y, Z order, 2D decomposition by Y and Z dimension can benefit the implementation and boundary exchange in the common way case.

When using 1D optimization method between sub-domain and parts, the computational part on the GPU is changed at each time step. To contain the points in continuous space on GPU side, it is better to separate the domain by X, Y dimension among the nodes.

5.2 Previous contribution: apply optimization method

As the upper section explains, our 2D decomposition method separates the whole domain into sub-domains by X, Y dimension. Then, we use 1D-decompsition to separate the sub-domain into parts by Z dimension as the below figure shows. Then, it can apply 1D optimization methods between sub-domain and parts. So, we can get optimization methods for multiple nodes by combining 2D decomposition method with 1D optimization methods: 2D-1D-N, 2D-1D-T, 2D-1D-TBM.

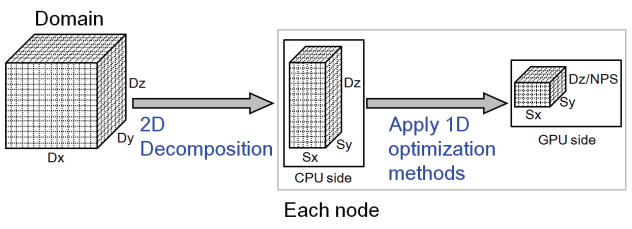


Fig.15. Apply 1D optimization method

Here, NPS is the number of parts. We set the number of nodes as Nx×Ny. Then, the size of each sub-domain is (Dx/Nx)×(Dy/Ny)×Dz; the size of each part is (Dx/Nx)×(Dy/Ny)×(Dz/NPS).

5.3 Contribution: 2D-1D-TBMR

In this section, we propose our optimization method 2D-1D-TBMR. We give the example of the kernel of 7 point stencil as below:

```
Example of 7 point stencil of floating computation:
__global__   void   kernel(grid0, grid1,⋯)
{
    int jx = blockDim.x * blockIdx.x + threadIdx.x;
    int jy = blockDim.y * blockIdx.y + threadIdx.y;
    float    tempo,tempc,tempb,tempt;
    for(jz = zstart;jz < zend; jz++)
    {
        ji = Sx*Sy*jz + Sx*jy + jx;
        je = ji+1;    jw = ji-1;    jn = ji+Sx;    js =ji-Sx;
        tempt    = grid0[ji];
        grid1[jt]= tempo+cc*tempc+ct*tempt+ cb*tempb;
        tempb    = tempc;
        tempc    = tempt;
        tempo = ce*grid0[je] + cw*grid0[jw]
                + cn*grid0[jn] + cs*grid0[js];
    }
}
```

It decomposes the whole domain by X, Y dimension. So, it adjusts the position of the points that are computed by the kernel as the upper shows. It also uses memory-saving method in the kernel. Then, we can give the process of 2D-1D-TBMR method as below:

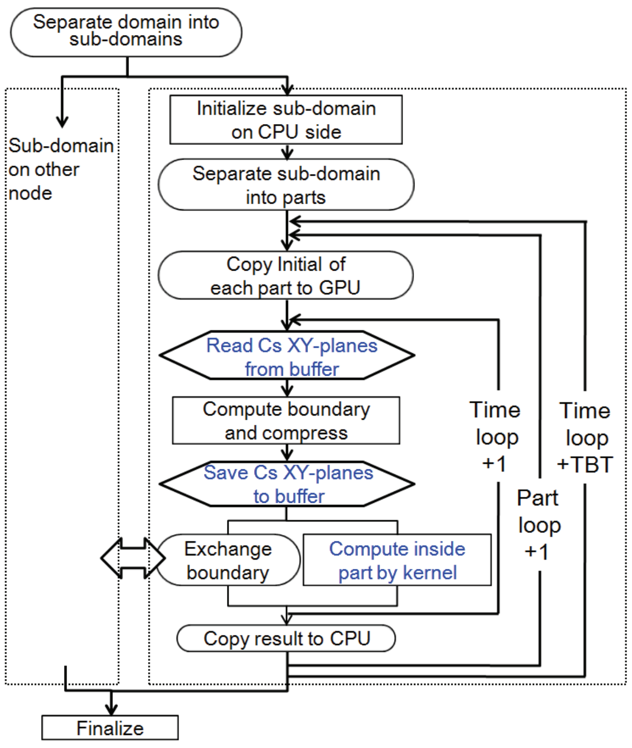

Separate domain into sub-domains
Sub-domain on other node
Initialize sub-domain on CPU side
Separate sub-domain into parts
Copy Initial of each part to GPU
Read Cs XY-planes from buffer
Compute boundary and compress
Save Cs XY-planes to buffer
Exchange boundary
Compute inside part by kernel
Copy result to CPU
Time loop +1
Time loop +TBT
Part loop +1
Finalize

Fig.16. The process of 2D-1D-P-TBMR

It should exchange the boundary of the part to continue the computation at each time step. To reduce the communication cost of exchanging boundary, it overlaps the communication with the inside computation at each time step.

6. Evaluation

6.1 1D-TBMR vs other methods on single node

We evaluated the optimization methods for 7 point stencil on single node of TSUBAME2.5. The domain is from 240×240×240 to 2160×2160×2160 of floating computation. As we can see in the below figure, 1D-P-TBMR method gets 2.63 times higher performance than 1D-T method.

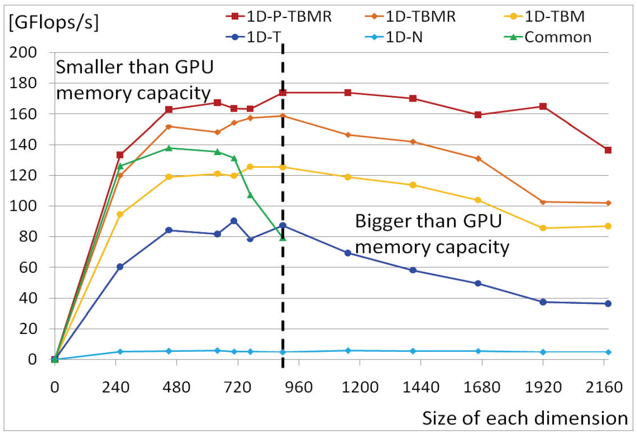


Fig.17. 1D-P-TBMR vs others

6.2 2D-1D-TBMR vs other methods on multiple nodes

We evaluated the 2D-1D-TBMR, 2D-1D-TBM, 2D-1D-T, 2D-1D-N method for 7 point stencil on multiple nodes of TSUBAME2.5. The domain is 5120×5120×2560 of floating computation. The number of nodes is from 16 to 256.

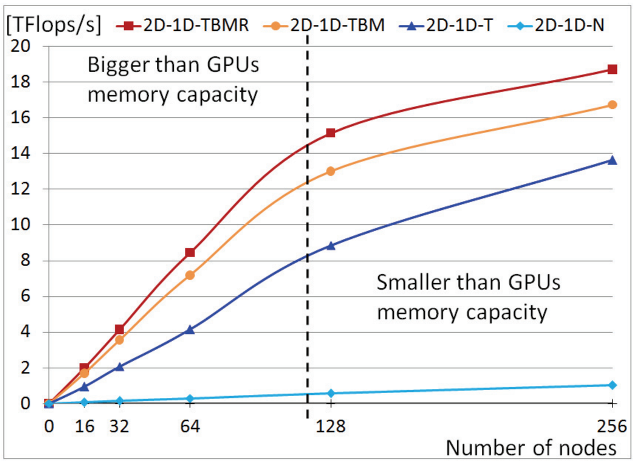


Fig.18. 7 point stencil, 2D-1D-TBMR vs others

As we can see in the upper figure, 2D-1D-TBMR method gets 1.59 times higher performance than 2D-1D-TBM (previous) and 1.84 times higher performance than 2D-1D-T method.

We evaluated the 2D-1D-TBMR, 2D-1D-TBM, 2D-1D-T, 2D-1D-N method for 19 point stencil on multiple nodes of TSUBAME2.5.

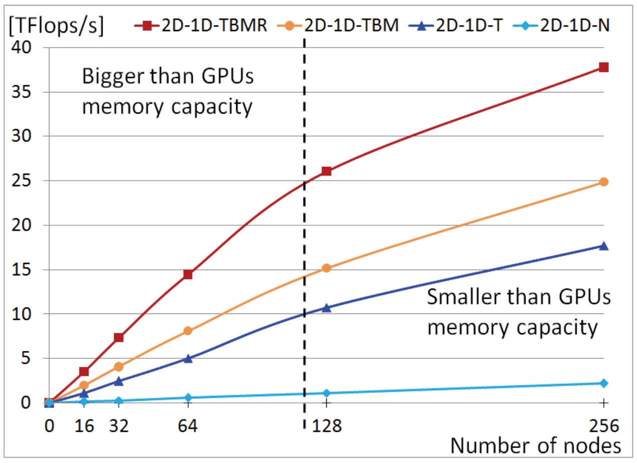


Fig.19. 19 point stencil, 2D-1D-TBMR vs others

In 19 point stencil case, 2D-1D-TBMR method gets 1.71 times higher performance than 2D-1D-TBM (previous) and 2.68 times higher performance than 2D-1D-T. As 19 point stencil reads more nearby points, it is more important to use registers to reduce access time.

6.3 Evaluation on TSUBAME2.0 and TSUBAME2.5

We evaluated the 2D-1D-TBMR method for 7 point stencil on multiple nodes of TSUBAME2.5 and TSUBAME2.0. The domain is 5120×5120×2560 of floating computation.

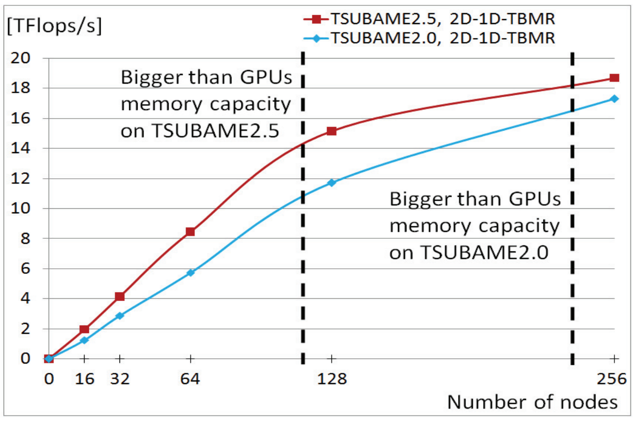


Fig.20. 7 point stencil, TSUBAME2.5 vs TSUBAME2.0

As we can see in the upper figure, it gets 1.37 times higher on TSUBAME2.5 than on TSUBAME2.0. Although it increases memory capacity of GPU by 2 times, it still needs to use optimization methods to compute bigger domain. It can compute 20 times bigger domain on TSUBME2.5 and 40 times bigger domain on TSUBAME2.0 than the common way. The domain size of the common way depends on the memory capacity of GPUs. The domain size of our optimization methods depends on the memory capacity of CPUs.

6.4 Parameters on TSUBAME2.0 and TSUBAME2.5

In multiple nodes case, it separates the domain into sub-domains and each sub-domain into parts. We set NPS is the number of parts; TBT is the temporal blocking times. We set TBT is smaller than the size Z dimension of each part (Dzp). To reduce the communication cost, we should get max TBT. So, we can get formula as below:

$Dzp = Dz / NPS,\ Cs = 2$

If use double buffering,

$$(Dx/Nx)\times(Dy/Nx)\times(Dzp+Cs)\times 2 + (Dx/Nx)\times(Dy/Ny)\times TBT\times Cs + (Dx+Dy)\times(Dzp)\times 2 \leqslant \text{GPU memory capacity} \quad (1)$$

$$TBT < Dzp, \quad \text{Get max TBT} \quad (2)$$

So, we can get the parameters in domain 5120×5120×2560 as the below figure shows. It can get max point 2 times earlier on TSUBAME2.5.

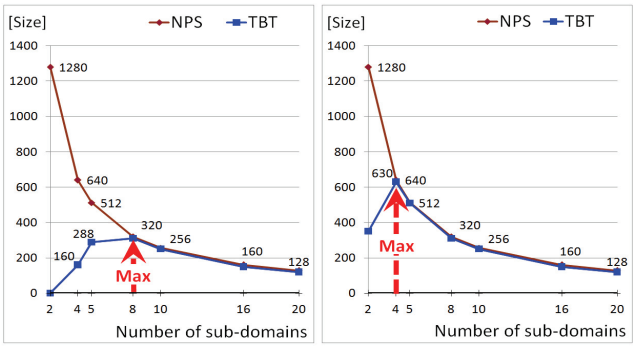


Fig.21. Parameters on TSUBAME2.5 vs on TSUBAME2.0

7. Conclusion

In this paper, we propose a series of new optimization methods which enable the computations on the domain that is bigger than the memory capacity of GPUs while maintaining high performance on TSUBAME GPU cluster. The result shows that our optimization method achieves 1.59 times higher performance in 7 point case and 1.71 times higher performance in 19 point case than other optimization methods on multiple nodes of TSUBAME2.5. Furthermore, our optimization method can compute 20 times bigger domain than the common way on TUSBAME2.5.

As a future work, we will try to lower programming difficulty by using tools like Physis [16]. To improve the performance and increase the memory capacity, we will also try to use 3 GPUs in each node.

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] TSUBAME2.5 User's Guide, 2013.

[15] Wai Teng Tang, Wen Jun Tan, Krishnamoorthy, R., Yi Wen Wong, Shyh-hao Kuo, Goh, R.S.M. , Turner, S.J., Weng-Fai Wong, "Optimizing and Auto-Tuning Iterative Stencil Loops for GPUs with the In-Plane Method ", Parallel & Distributed Processing (IPDPS), pp. 452 – 462, 2013.

[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.

Acknowledgments This work was partly supported by "Software Technology that Deals with Deeper Memory Hierarchy in Post-peta scale Era", JST-CREST.