Journal of Physics: Conference Series **759** (2016) 012100 doi:10.1088/1742-6596/759/1/012100

High performance FDTD algorithm for GPGPU supercomputers

Andrey Zakirov, Vadim Levchenko, Anastasia Perepelkina, Yasunari Zempo

Keldysh Institute of Applied Mathematics, RAS, Miusskaya sq., 4, Moscow, 125047, Russia
Hosei University, Faculty of Computer and Information Sciences, 3-7-2 Kajino-cho, Koganei-shi, Tokyo, Japan

E-mail: mogmi@narod.ru

Abstract. An implementation of FDTD method for solution of optical and other electrodynamic problems of high computational cost is described. The implementation is based on the LRnLA algorithm DiamondTorre, which is developed specifically for GPGPU hardware. The specifics of the DiamondTorre algorithms for staggered grid (Yee cell) and many-GPU devices are shown. The algorithm is implemented in the software for real physics calculation. The software performance is estimated through algorithms parameters and computer model. The real performance is tested on one GPU device, as well as on the many-GPU cluster. The performance of up to $0.65 \cdot 10^{12}$ cell updates per second for 3D domain with $0.3 \cdot 10^{12}$ Yee cells total is achieved.

1. Introduction

The Finite Difference Time Domain (FDTD) method [1] is implemented in numerous codes for simulation of electrodynamics. The method is easily extended for simulation of other wave processes, such as acoustic and elasticity waves [2]. This paper deals with an original approach to the numerical solution of the evolutionary Cauchy problem, the Locally Recursive non-Locally Asynchronous (LRnLA) algorithms [3]. The difference between the LRnLA method and conventional methods is that the optimization of computations deals not only with layerwise computation, but traces data dependencies in a 4D space-time domain [4]. The algorithms are implemented in codes for various physics modelling [5, 6, 7, 8, 9].

The increase in performance is possible by avoiding stepwise synchronization. This is the idea of alternative algorithms for stencil computations (trapezoids, time-slicing and time-skewing [10, 11, 12, 13, 14, 15]). This approach is not wide spread, but the basics may be found in works by other authors. The so-called loop tiling and loop skewing methods result in similar algorithms for a simple domain geometry [16, 17, 18, 19]. The research on loop blocking led to creation of the cache-aware and cache-oblivious algorithms [20]. These were used for stencil computations of partial derivative equations [21, 15]. In a 1D simulation these methods lead to trapezoidal and diamond blockings of space. The generalizations to 2D and 3D are possible.

Among these approaches the LRnLA method has the following advantages. The approach takes account for the complexity of modern computers. The space-time optimization account for all parallel levels, all levels of the memory subsystem. The theory is built on the model of the available computer and allows a priori quantitative estimates of the performance of the supposed

method implementation. The theory applies to any physics simulation with local dependencies, any amount of dimensions.

Some cache-oblivious algorithms coincide with the LRnLA algorithms for lower dimensions.

1.1. DiamondTorre algorithm and its CUDA implementation

The DiamondTorre algorithm is described as a subdivision of 4D space-time region in shapes. DiamondTorre size is defined by two parameters: the size of the base `DTS`, and its height `TH`. More detailed explanation of the algorithm, its parameters and implementation for wave equation on 1 GPU can be found in [4].

The correspondence of the DiamondTile to the components of staggered grid in X—Y projection is illustrated on fig. 2. This is defined as the basic element for DiamondTile algorithms for the FDTD scheme for Maxwell equations with the 4th order of approximation and its size is `DTS=1` by definition. It consists of the two (E-field and B-field) diamonds that are shifted along the time axis (by half of the time step) and along one coordinate axis (by 1.5 spatial step — the half width of the chosen scheme stencil). DiamondTorre consists of `TH` DiamondTile pairs, shifted in a similar manner against each other. The DiamondTorre algorithm is a process of making calculations for all points in the described 4D shape. In the GPGPU implementation, DiamondTorre is a CUDA-kernel. DiamondTorre's with the same Y-axis position are processed asynchronously by CUDA-blocks. Inside, cells with different Z-coordinates are processed by different CUDA-threads.

The DiamondTorre base size `DTS`=1 is optimal in this case. The higher `DTS` leads to higher operational intensity. But with `DTS`=2 significant performance drops are confirmed, since the data on one DiamondTile exceeds the GPU register file size, or the instruction count in one DiamondTorre kernel exceeds the instruction cache size. The register size is taken as 256 KB. If we choose the number of grid points along Z axis as $Nz = 384$ (double precision) or $Nz = 768$ (single precision), the limit of registers per CUDA-thread is estimated as $256K/(768 * 4) = 85$. This roughly corresponds to the amount of data for `DTS`=1. Also, in this case, one CUDA-kernel corresponding to DiamondTorre algorithm contains ~ 1500 instructions. One instruction takes ~ 8 B, which brings us very close to the instruction cache limit (estimated as 16 KB).

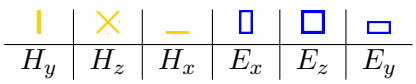

Figure 1. Field designations

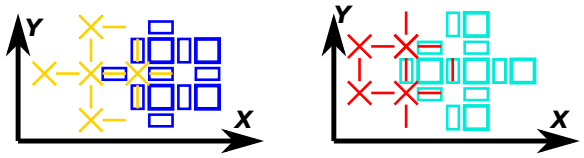

Figure 2. The field components in one DiamondTile. Two options.

If we assume that one DiamondTile is loaded into the GPU register, to calculate one B-field diamond three E-field diamonds need to be loaded. The resulting B-fields need to be saved. It concludes one half of cell updates in a Diamond, for the other half three B-field diamonds are loaded and one B-field diamond is calculated and saved. An estimate of data throughput can be made. With `TH` $\to \infty$, on average, 3 Yee cell data are to be loaded, and 1 Yee cell data should be saved per one Yee cell update, $4 \times 6 \times 8 = 192$ B total. Operation count for one cell update is about 110 Flop. Operational intensity is obtained as their ratio, 0.57 Flop/Byte.

We use roofline [22] model to estimate the efficiency of the algorithms. For any contemporary GPU our problem is memory-bound. The theoretical performance is estimated as $P/192$ Yee cell updates per second (Ys), where P is GDDR5 memory bandwidth, 192 is the necessary data throughput. On NVidia Tesla K20 ($P = 224 \cdot 10^9$ bytes per second) this equals $1.167 \cdot 10^9$ Ys.

1.2. Calculation window

The advantage of DiamondTorre algorithm is the high performance for large problems in which field data do not fit in the device memory. It is easily achieved by updating data in a "calculation window", which moves along x-axis in negative direction. Data load and save to/from the global RAM are performed asynchronously with computations. Only the data for one following group of asynchronous DiamondTorres are loaded. The data of DiamondTorres which are no longer necessary for the current `TH` update are saved and deleted from device memory.

The performance does not decline if the computation time of DiamondTorre is longer than the time that is necessary for memory copy to/from the device. The computation time increases linearly with `TH`, and the copy time is constant. So with high enough `TH` host-device transfers are concealed.

1.3. Small scale performance tests

The described algorithms are implemented in code, which features all the required methods for real physics computations: the FDTD simulation is performed in a 3D spatial domain with the 4th order accurate scheme; Perfectly Matched Layer (PML) absorbing boundary conditions, Total Field/Scattered Field (TFSF) wave source and complex material equations according to Drude, Drude-Lorenz model are used.

Small scale performance tests were conducted to find the optimal algorithm parameters. We measure performance in Yee cell updates per second (Ys). This number is usually of 10^9 order, so the main unit is GYs.

Fig. 3 shows the performance results for a problem size of $(600 \times (3 \times blocks) \times 384)$ with varied "blocks" parameter and varied `TH`. For DiamondTorres lined up along Y axis, one CUDA-block performs computation for one DiamondTorre. The tests were conducted on Tesla K20x. It has 14 streaming multiprocessors (SMs). If the number of involved SMs per device is less than 14, the performance is limited by by GDDR5 access latency. The latency is about 500 clock cycles (about 500 ns). From the Little's law [23], to cover this latency, at least 10^4 transactions are necessary.

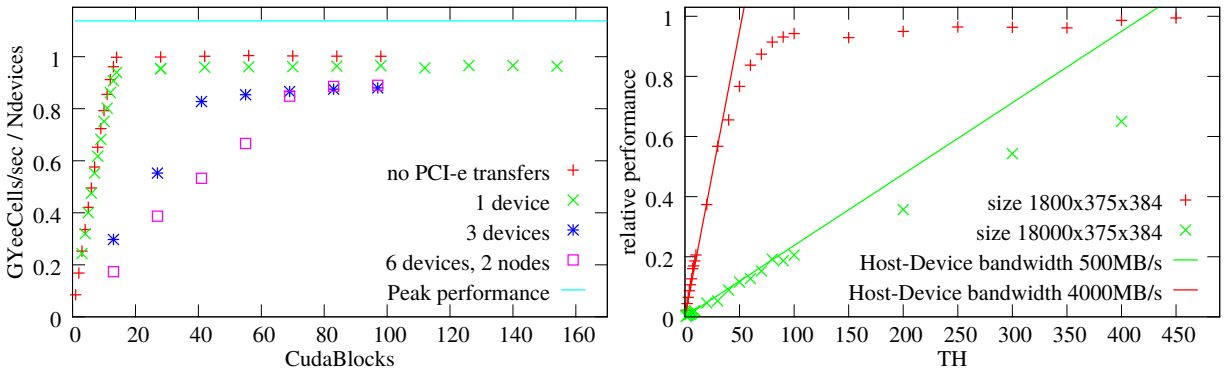

Figure 3. Performance results for varied amount of blocks. Domain size is $600 \times (3 \cdot block) \times 384$ (left). Performance results for varied TH (right). The second problem size is 10 times bigger, so it does not fit into node memory.

The register file should be enough to keep the data of all necessary diamonds. So, for the Kepler architecture, the maximum vector size (Nz, also equal to the amount of involved CUDA-threads) is equal to 384 (double precision). This is why the data throughput is not utilized completely when all 14 SMs are involved ($\sim 14 \cdot 384 \approx 6000$ transactions). It is the main reason why the performance is only 90% from the peak one. It becomes more than 10^9 GYs for

sufficiently large `TH`. `TH` is inevitably smaller near region boundaries, so the performance for real problems is by few percent lower.

For low `TH` (fig. 3) the performance is limited by the PCI-express bandwidth. With the increase of `TH` the shift to limitation by GDDR5 bandwidth is confirmed. Its smoothness is explained by the cut-off of DiamondTorres at the edges of the domain. The optimal sufficient `TH` is 100, as can be seen on the graph. This value is used in subsequent tests, if not stated otherwise. A similar behavior of the dependency of performance rate on `TH` is observed when the data is stored on a disk (SSD in this case). The optimal `TH` becomes significantly larger (more than 500).

1.4. Multi-device and multi-node parallelism

All DiamondTorre's standing side-by-side along Y axis are asynchronous. So they may be processed by different devices inside one node, or by different nodes. Judging by the previous tests (fig. 3) the data transfers between 3 devices on one node may be concealed completely if the amount of the involved CUDA-blocks is more than 42 on each device. For the devices installed on different nodes the duration of data transfer is approximately twice bigger. In this case data transfers may be concealed completely if the amount of the involved CUDA-blocks is more than 70 on each device.

Additionally, concurrency on X axis is possible. In this case the data are subdivided in blocks in X axis and distributed between nodes. The data on adjacent nodes overlap by the calculation window size: NW points of X axis. On the i-th device the data are updated from the $(n + i \cdot TH)$-th step to the $(n + i \cdot TH + \mathrm{TH})$-th step, where n is an integer number (the time step on which the data of the leftmost node exists on).

2. Parallel Scaling Results on TSUBAME2.5

The parallel performance of the code has been tested on TSUBAME2.5 supercomputer. Its specifications, that are important for this study, are as follows. The amount of available node per one run is up to 300 according to the usage conditions (1408 in total). Each node has 3 NVIDIA Tesla K20x GPGPUs installed. Their total GDDR5 memory is $3 \times 5.625 = 16.875$ GB, with 208 GB data throughput each. Each node has at least 54 GB, up to 96 GB on several ones. Only a little above 40 GB from it is available for the computation. Devices are connected with PCI-e 2.0 with 4 GB/sec throughput in each direction. Each node has 120 GB SSD memory. Nodes are connected by Infiniband QDR interconnect with up to 4 GB/sec throughput.

2.1. Weak scaling

For the weak scaling the domain is scaled proportionally to the amount of nodes used. Both X and Y axis parallelism were tested. Three series were performed (fig. 4):

(i) Y axis scaling. Data transfers may be concealed (112 CUDA-blocks on each device).

(ii) Y axis scaling. Data transfers limit the performance (42 CUDA-blocks on each device).

(iii) X axis scaling. Each device contains 1890 Yee cells. All 3 devices are involved on each node, performing parallel computation in Y direction.

In the first case, the parallel efficiency is above 99% as expected. The maximum achieved performance is $0.65 \cdot 10^{12}$ Ys for one computation on domain with $300 \cdot 10^9$ grid points (10 TB data, 256 nodes). In the second case the performance becomes lower when the number of nodes rises from 1 to 2. This is because the Infiniband data throughput is lower than data throughput between devices on one node. The following increase in the number of nodes does not lower the performance. In the third case parallel scaling is close to ideal, but still less than the one for the first series. The main decrease occurs between 1 and 2 nodes. It is caused by a slight imbalance in node utilization.

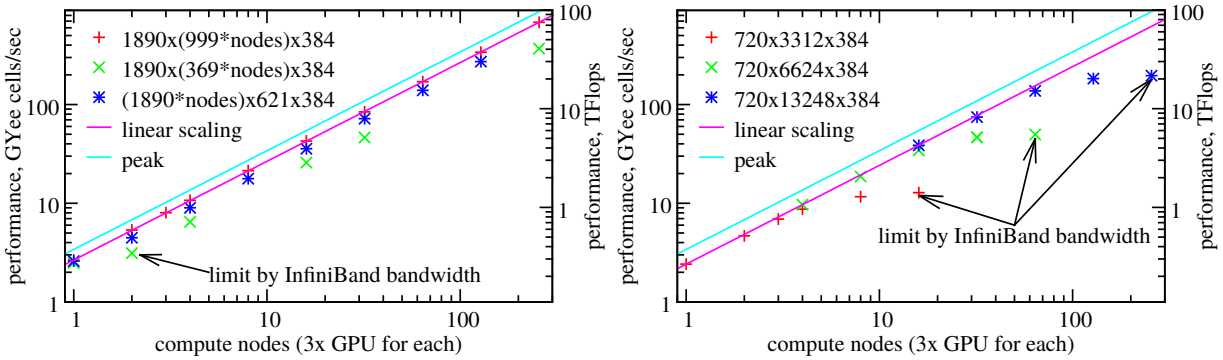

Figure 4. Weak scaling (left). Strong scaling (right)

2.2. Strong scaling

For the strong scaling we chose a fixed size domain. By increasing the amount of nodes, the domain is subdivided to more and more parts, that are to be processed concurrently. Three series were performed (fig. 4):

(i) $720 \times 3312 \times 384$ size domain is scaled on 1–32 nodes;

(ii) 4 times bigger domain ($720 \times 13248 \times 384$) is scaled on 4–64 nodes;

(iii) 16 times bigger domain ($720 \times 52992 \times 384$) is scaled on 16–256 nodes;

With the increase in amount of parallel nodes the performance decreases since the amount of CUDA-blocks per device becomes lower (see fig. 3). At the same time there is a limit on maximum Ny size per device, determined by device memory. It actually may be increased, but the Nx size and `TH` should be decreased at the same time. This would lower the general performance. By utilizing more nodes, domain size and `TH` may be increased again and performance rises.

This dependency on `TH` is shown on fig. 5. The problem size is $450 \times 62208 \times 128$ grid cells. For one node computation `TH` is equal to 15, and increases up to 150 for 8 nodes and higher. For low amount of nodes the speedup is better than linear. This is because `TH` may be optimized only when enough data is processed on each node.

Finally, strong scaling tests were performed on a problem with $38400 \times 363 \times 128$ grid points (fig. 5). On one node the performance is about 30% from the peak performance, since the size along Z axis is not optimal. It is not big enough to conceal the GDDR5 access latency, since the amount of simultaneous transactions is too low. But the acceleration is up to 40 times. Only with 128 nodes and above the data transfers are taking more time than computation, which leads to decrease in computation rate.

One node has little memory size (only about 3 times more than total device memory), and this becomes the reason for the acceleration limit. The increase in the available memory size should increase acceleration ability proportionally.

3. Conclusion

The work can be summarized as follows. The FDTD code has been developed, that allows simulation of real optical phenomena. The distinguishing feature of the code is the use of DiamondTorre LRnLA algorithm, which maximizes the performance on one device, and parallel efficiency for multi-GPU architectures. The software was tested on the TSUBAME2.5 supercomputer. The high computation rate is achieved (more than 1 billion Yee cell updates per second on one device). The problem size is not limited by device memory. The scaling for ~ 1000 devices becomes 1000 for the weak scaling, and ~ 100 for the strong scaling.

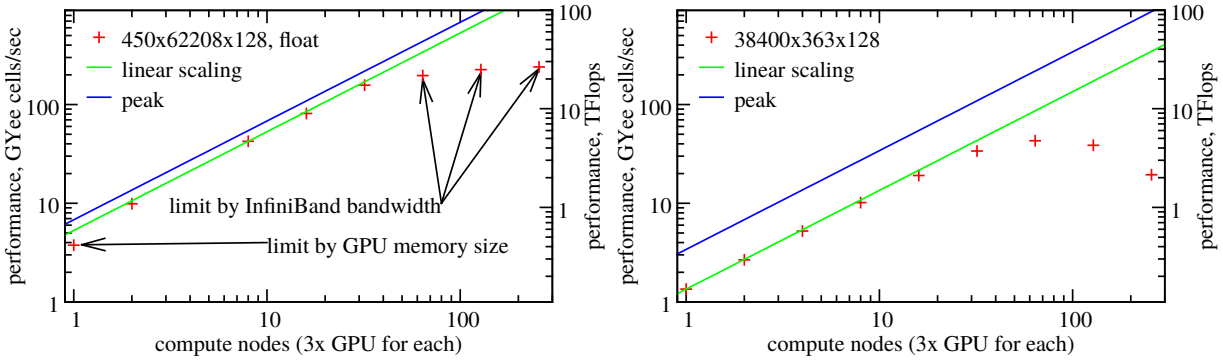

Figure 5. Strong scaling on Y axis with variable TH (left), on X axis (right)

The algorithm parameters (such as `TH` and problem size) allow not only qualitative, but also quantitative estimates of the performance and parallel scaling. Maximal achieved performance is $0.65 \cdot 10^{12}$ for 3D domain with $0.3 \cdot 10^{12}$ Yee cells total. For example, such size for wave optics problems corresponds to 1 cubic millimeter domain. This allows a significant breakthrough in computational nanooptics, by allowing the simulation in domains that were previously too big even for supercomputers. It may be used for simulation of complex optical devices.

The work is supported by Hosei International Fellowship grant, RFBR grant no. 14-01-31483.

References

[1] Taflove A and Hagness S C 2005 *Computational Electrodynamics: the Finite-Difference Time-Domain Method* 3rd ed (Norwood, MA: Artech House)
[2] Virieux J 1986 *Geophysics* **51** 889–901
[3] Levchenko V 2005 *J. of Inf. Tech. and Comp. Systems* **1** 68 (in Russian)
[4] Perepelkina A Y and Levchenko V D 2015 *Keldysh Institute Preprints* **18** 20
[5] Korneev B A and Levchenko V D 2015 em Procedia Computer Science **51** 1292-1302
[6] Perepelkina A Y, Goryachev I A and Levchenko V D 2013 *Journal of Physics: Conference Series* **441** 012014
[7] Perepelkina A, Goryachev I and Levchenko V 2014 *Journal of Physics: Conference Series* **510** 012042
[8] Zakirov A.V., Levchenko V.D. 2012 *Mathematical Models and Computer Simulations* **4** 2 pp 155–162.
[9] Kaplan S.A., Levtchenko V.D. et al. *Geoinformatika* 2011. **1** pp 49-55. (in Russian)
[10] Frigo M and Strumpen V 2007 *The Journal of Supercomputing* **39** 93–112
[11] Tang Y *et al.* 2011 *Proceedings of the Twenty-third Annual ACM SPAA* (NY, USA: ACM) pp 117–128
[12] Grosser T *et al.* 2013 *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units* GPGPU-6 (New York, NY, USA: ACM) pp 24–31
[13] Orozco D, Gao G 2009 Mapping the fdtd application to many-core chip architectures *Parallel Processing, 2009. ICPP '09* pp 309–316 ISSN 0190-3918
[14] McCalpin J, Wonnacott D 1999 Time skewing: A value-based approach to optimizing for memory locality Tech. rep.
[15] Strzodka R *et al.* 2011 *Proceedings of the International Conference on Parallel Processing* IEEE Computer Society p 571–581
[16] Wolf M, Lam M 1991 *Proceedings of the ACM SIGPLAN 1991* (New York, NY, USA: ACM) pp 30–44
[17] Wolfe M 1989 *Proceedings of the 1989 ACM/IEEE Conference on Supercomputing* Supercomputing '89 (New York, NY, USA: ACM) pp 655–664
[18] Wolfe M 1986 *International Journal of Parallel Programming* **15** 279–293
[19] Terrano A 1988 *Frontiers of Massively Parallel Computation, 1988. Proceedings* pp 227–229
[20] Prokop H 1999 Cache-oblivious algorithms
[21] Frigo M and Strumpen V 2005 *Proceedings of the 19th Annual International Conference on Supercomputing* pp 361–366
[22] Williams S, Waterman A and Patterson D A 2009 *Commun. ACM* **52** 65–76
[23] Little J D C 1961 *Operations Research* **9** 383–387