Performance Analysis for Stencil-Based 3D MPDATA Algorithm on GPU Architecture

Krzysztof Rojek(✉), Lukasz Szustak, and Roman Wyrzykowski

Czestochowa University of Technology, Dabrowskiego 73,
42-201 Czestochowa, Poland
{krojek,lszustak,roman}@icis.pcz.pl

Abstract. EULAG (Eulerian/semi-Lagrangian fluid solver) is an established computational model for simulating thermo-fluid flows across a wide range of scales and physical scenarios. The multidimensional positive defined advection transport algorithm (MPDATA) is among the most time-consuming components of EULAG.

The main aim of our work is to design an efficient adaptation of the MPDATA algorithm to the NVIDIA GPU Kepler architecture. We focus on analysis of resources usage in the GPU platform and its influence on performance results. In this paper, a performance model is proposed, which ensures a comprehensive analysis of the resource consumption including registers, shared, global and texture memories. The performance model allows us to identify bottlenecks of the algorithm, and shows directions of optimizations.

The group of the most common bottlenecks is considered in this work. They include data transfers between host memory and GPU global memory, GPU global memory and shared memory, as well as latencies and serialization of instructions, and GPU occupancy. We put the emphasis on providing a fixed memory access pattern, padding, reducing divergent branches and instructions latencies, as well as organizing computation in the MPDATA algorithm in order to provide efficient shared memory and register file reusing.

Keywords: GPGPU · CUDA · EULAG · Stencil · MPDATA · Geophysical flows · Parallel programming

1 Introduction

The multidimensional positive definite advection transport algorithm(MPDATA) is among the most time-consuming calculations of the EULAG model [2,9]. In our previous works [8,10,11] we proposed two decompositions of 2D MPDATA computations, which provide adaptation to CPU and GPU architectures separately. The achieved performance results showed the possibility of achieving high performance both on CPU and GPU platforms.

In the paper [12], we developed a hybrid CPU-GPU version of 2D MPDATA, to fully utilize all the available computing resources by spreading computations across the entire machine. It is the starting point for our current work.

R. Wyrzykowski et al. (Eds.): PPAM 2013, Part I, LNCS 8384, pp. 145–154, 2014.
DOI: 10.1007/978-3-642-55224-3_15,

In this paper, we focus on parallelization of the 3D MPDATA algorithm, and analysis of resources usage in the GPU platform and its influence on the performance. We detect the bottlenecks and develop the method of efficient distribution of computation across CUDA kernels. Proposed method is based on analysis of memory transactions between GPU global and shared memory.

2 Related Works

Reorganizing stencil calculations to take full advantage of memory hierarchies has been the subject of much investigation over the years.

Modern processor architectures tends to be inherently unbalanced concerning the relation of theoretical peak performance versus memory bandwidth. To reveal performance constraints for MPDATA running on hybrid architectures, we will follow the simple methodology presented in [4], where attainable performance is estimated based on flop and byte ratio.

Memory optimizations for stencil computations have principally focused on different decomposition strategies, like space and blocking techniques [3], that attempt to exploit locality by performing operations on data blocks of a suitable size before moving on to the next block.

The issue of adapting the EULAG model to GPU accelerators was discussed in [5], where the PGI Accelerator compiler was used for the automatic parallelization of selected parts of EULAG on NVIDIA GPUs, including the 2D MPDATA algorithm. However, disadvantage of this approach is relaying entirely on the automatic parallelization, without any efforts to guide the parallelization process taking into account characteristics of target architectures.

In the paper [6], a 3.5D-blocking algorithm that performs 2.5D-spatial blocking of the input grid into on-chip memory for GPUs was discussed. We also employ 2.5D blocking technique to increase data locality, but we propose alternative solution for memory-bounded kernels, which is based on minimizing the number of global memory transactions, rather than applying 3.5D-blocking.

The quite large set of techniques of CUDA optimizations including data parallelism, threads deployment and the GPU memory hierarchy was discussed in [1]. In this work, the authors manually evaluated the best configurations of 2D stencil computations. We offer model-based solution, which automatic configures the code, making our solution more portable.

3 Kepler NVIDIA Architecture

The NVIDIA GTX TITAN GPU [7] is based on the Kepler architecture, and includes 14 streaming multiprocessors (SMX), each consisting of 64 double precision units (DP units) with 48 KB of shared memory and 16 KB of L1 cache. It gives a total number of 896 DP units with the clock rate of 870 MHz. It provides the peak performance of 1.5 TFlop/s in a double precision. This graphics accelerator card includes 6 GB of global memory with the peak bandwidth of 288 GB/s. All the accesses to the global memory go through the L2 cache of size

1.5 MB. This GPU supports two modes of access to data: 32-bit access mode and 64-bit access mode. The number of load/store unit per SMX is 32, so it gives a possibility to load/store 256 bits per clock cycle per SMX.

4 3D MPDATA Overview

Our research includes Multidimensional Positive Definite Advection Transport Algorithm (MPDATA), which is one of the main part of the EULAG geophysical model EULAG (EUlerian/semi-LAGrangian) can be used to simulate: weather prediction; ocean currents; areas of turbulence; urban flows; gravity wave dynamics; micrometeorology; cloud microphysics and dynamics.

The MPDATA algorithm belongs to the group of nonoscillatory forward in time algorithms [9]. The 3D MPDATA is based on the first-order-accurate advection equation:

$$\frac{\partial \Psi}{\partial t} = -\frac{\partial}{\partial x}(u\Psi) - \frac{\partial}{\partial y}(v\Psi) - \frac{\partial}{\partial z}(w\Psi), \tag{1}$$

where x, y and z are space coordinates, t is time, $u, v, w = const$ are flow velocities, and Ψ is a nonnegative scalar field. Equation (1) is approximated according to the donor-cell scheme, which for the $(n+1)$-th time step ($n = 0, 1, 2, \ldots$) gives the following equation:

$$\begin{aligned} \Psi^*_{i,j,k} = \Psi^n_{i,j,k} - & \left[F(\Psi^n_{i,j,k}, \Psi^n_{i+1,j,k}, U_{i+1/2,j,k}) - F(\Psi^n_{i-1,j,k}, \Psi^n_{i,j,k}, U_{i-1/2,j,k})\right] \\ - & \left[F(\Psi^n_{i,j,k}, \Psi^n_{i,j+1,k}, V_{i,j+1/2,k}) - F(\Psi^n_{i,j-1,k}, \Psi^n_{i,j,k}, V_{i,j-1/2,k})\right] \\ - & \left[F(\Psi^n_{i,j,k}, \Psi^n_{i,j,k+1}, W_{i,j,k+1/2}) - F(\Psi^n_{i,j,k-1}, \Psi^n_{i,j,k}, W_{i,j,k-1/2})\right]. \end{aligned} \tag{2}$$

Here the function F is defined in terms of the local Courant number U:

$$F(\Psi_L, \Psi_R, U) \equiv [U]^+ \Psi_L + [U]^- \Psi_R, \tag{3}$$

$$U \equiv \frac{u\delta t}{\delta x};\ [U]^+ \equiv 0,5(U + |U|);\ [U]^- \equiv 0,5(U - |U|). \tag{4}$$

The same definition is true for the local Courant numbers V and W.

The first-order-accurate advection equation can be approximated to the second-order in δx, δy and δt, defining the advection-diffusion equation. Such transformation is widely described in literature. For the full description of the main important aspects of the second order equation of MPDATA, the reader is referred to [9].

The 3D MPDATA algorithm consists of 17 stencils that are processed by CUDA kernels on the GPU. Figure 1 shows the mechanism of kernel processing. We employ widely used method of 2.5D blocking [6], where two dimensional CUDA blocks are responsible for computing XY planes of matrices. The loop inside kernel is used to traverse the grid in the Z dimension. Since, the MPDATA algorithm requires to store 3 XY planes at the same time, we use queue of planes placed in registers and shared memory, which firstly copies data from GPU global memory to registers, and then moves data between registers and shared memory. This method allows us to increases data locality significantly.

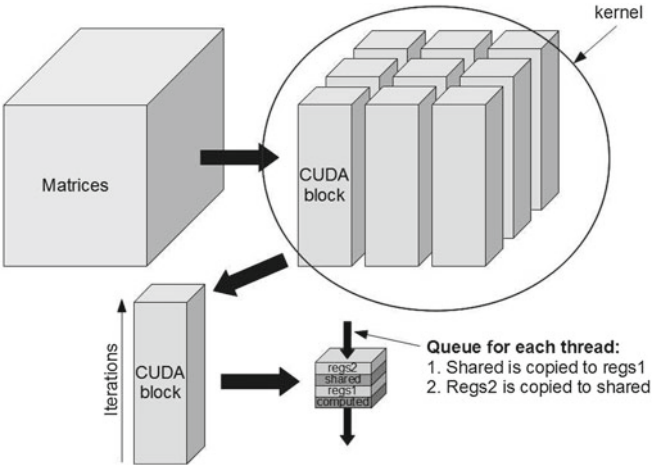

Fig. 1. Kernel processing

5 Analysis of 3D MPDATA with NVIDIA Visual Profiler

The starting point of our considerations is when the 17 stencils are distributed across 6 CUDA kernels. Our analysis begins with detection of bottlenecks of the algorithm. We examine the following potential bottlenecks:

- data transfers between GPU global memory and host memory;
- instructions latency (stall analysis);
- arithmetic, logic, and shared memory operations;
- configuration of the algorithm taking into account size of CUDA block and GPU occupancy.

Our approach is based on the stream processing [12] (Fig. 2) where each stream is responsible for computing a sequence of 3 instructions including: data transfer from host to GPU that occurs only once (before computations); execution the sequence of 6 kernels; data transfer from GPU to host memory (occurs after every time step). Since all streams are processed independently, the computation and data transfers can be overlapped. Table 1 shows the time consumption analysis of MPDATA for the 100 time steps and grid of size 392. Three streams are used in the simulation. The HTOD abbreviation means the data transfer from host to device, while the DTOH means data transfer in the opposite direction.

Based on this analysis, the data transfer takes relatively short time (about 18 % of all execution time). Stream processing decreases execution time by about 0.9 s, which is 2 times more than time of data transfer. We can simply conclude here, that data transfer between host and GPU is not a bottleneck of the MPDATA algorithm.

Now we focus on the analysis of computations. Our research include the most complex part of the MPDATA algorithm. Based on the NVIDIA Visual Profiler,

Fig. 2. Utilization of GPU resources by MPDATA

Table 1. Time consumption analysis of the 3D MPDATA algorithm

Operation	Time [s]	Ratio
HTOD	0.023	0.008
DTOH	0.453	0.172
Computation	3.051	1.16
Final time	2.631	1

we estimate two the most time consuming kernels, which are called kernels B and C. These kernels take about 57 % of the execution time. Each of this kernel has 5 input and 3 output matrices and is responsible for computing 3 stencils with 37 flops per each. The flop/B ratio for each kernel is $37 * 3/((5 + 3) * 8) = 1.73$. However, the minimum flop/B ratio required by NVIDIA GTX TITAN to achieve maximum performance is 5.2 [7]. The another conclusion is that the kernels are strongly memory-bounded!

The next analysis is devoted to the stall reasons analysis. Figure 3 shows the main reason of stalls for kernels B and C including execution dependency, data request, texture memory operations, synchronization, and instruction fetching. Based on the analysis, stalls are mostly caused by the execution dependency

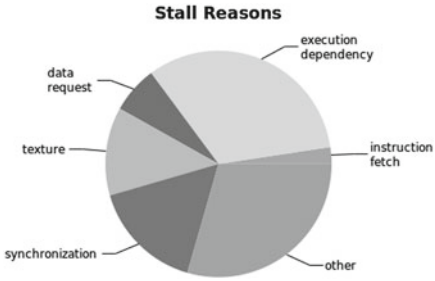

Fig. 3. Analysis of stall reasons for kernels B and C

(about 33 %). Such kind of stalls limits GPU utilization and results from the complex structure of the MPDATA algorithm. The execution dependencies can be hidden by increasing GPU occupancy. However, the kernels B and C use about 47 KB of shared memory for each CUDA block, executing only 768 active threads per SMX. It means, that the GPU occupancy is only 37.5 % for the both kernels. So the final conclusion is that the GPU utilization is limited by shared memory usage!

6 Performance Analysis Based on GPU Global Memory Transactions

We propose a performance analysis based on GPU global memory transactions. Such analysis is particularly helpful, when algorithm is memory-bounded. In our approach, the following scenarios are considered:

- distribution of computation across 2 kernels;
- compression of computation within 1 kernel.

The compression of computation increases hardware requirements for CUDA blocks, and decreases the GPU occupancy. Hence, the second scenario allows us to execute at most 512 active threads per SMX. It means, that the GPU occupancy is even lower than for the first scenario, and it is only 25 %.

At the beginning of our analysis we need to estimate the cost of access to matrix for each scenario. We assume, that CUDA block is of size $g_1 \times g_2$, matrices are processed according with Fig. 1, halo areas are of size 1, and are placed from the four sides of CUDA block (Fig. 4). The number of elements, that need to be transferred from GPU global memory to shared memory or register files is given by the following formula:

$$S_{el} = g_1 * g_2 + 2 * g_2 + 2 * g_1. \tag{5}$$

Taking into account 64-bits access mode, which can be simply enabled on Kepler NVIDIA architecture by calling $cudaDeviceSetSharedMemConfig()$

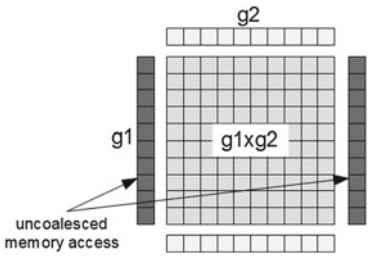

Fig. 4. XY plane of CUDA block with its halo areas

routine with $cudaSharedMemBankSizeEightByte$ parameter, we can estimate the number of required transactions to transfer a single CUDA block:

$$S_{tr} = g_1 * top(g_2/32) + 2 * top(g_2/32) + 2 * g_1, \tag{6}$$

where $top(x)$ returns rounded up value of x. In this approach, addresses of vertical halo areas are not coalesced.

Table 2 shows the cost of access to matrix for the first scenario. In this analysis, the plane is of size 392×256. Transactions overhead is ratio between required number of transactions to transfer matrix and the naive number of transactions required to transfer matrix assuming unlimited size of shared memory (without halo area). The naive number of transactions for plane of size 392×256 is 3136. Based on our analysis, the minimum number of transactions for the first scenario is $3136 * 185.2\,\% = 5808$. This analysis also allows us to estimate the most suitable size of CUDA block, which is 6×128.

Table 2. Analysis of GPU global memory transactions: first scenario

g1	g2	Blocks per plane	Transactions per block	Transactions per plane	Transactions overhead [%]
6	128	132	44	5808	185.2
5	128	158	38	6004	191.45
3	256	131	46	6026	192.16
4	128	196	32	6272	200
8	96	147	46	6762	215.63
3	128	262	26	6812	217.22
12	64	132	52	6864	218.88
7	96	168	41	6888	219.64

A similar analysis is made for the second scenario and the results of this analysis are shown in Table 3. The best configuration of CUDA block is 4×64, while the transactions overhead is 250 %.

Finally, we estimate the cost of access to all matrices for the both scenarios. Figure 5 shows flow diagram for kernels B and C. There are 5 input matrices for the kernel B and 6 input matrices for the kernel C. Additionally, there are 2 output matrices per each kernel. Transactions overhead for each matrix is 185.2 %, so the total cost of access to all matrices is $(5+6+2+2)*1.852 = 27.78$. The flow diagram for the second scenario is shown in Fig. 6. Here we have 5 input and 3 output matrices. Transactions overhead is 250 %. So, the total cost of access to all matrices is $(5 + 3) * 2.5 = 20$.

Table 4 shows the summary of MPDATA analysis, taking into account considered scenarios. Based on our analysis, it is expected to achieve about 1.39 speedup using the second scenario over the first scenario. So the conclusion is that, we should compress kernels B and C into a single kernel.

Table 3. Analysis of GPU global memory transactions: second scenario

g1	g2	Blocks per plane	Transactions per block	Transactions per plane	Transactions overhead [%]
4	64	392	20	7840	250
2	128	392	20	7840	250
3	64	524	16	8384	267.35
2	96	588	16	9408	300
2	64	784	12	9408	300
8	32	392	26	10192	325
1	256	392	26	10192	325
7	32	448	23	10304	328.57

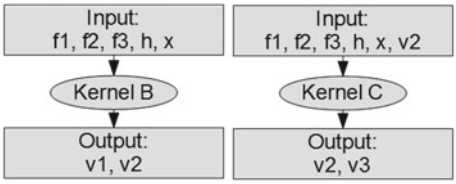

Fig. 5. Flow diagram for kernels B and C

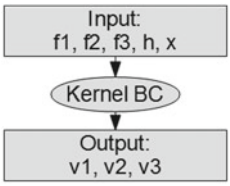

Fig. 6. Flow diagram for BC kernel

Table 4. Summary of MPDATA analysis

	Kernels B and C	Kernel BC	Ratio
Occupancy	37.50 %	25.00 %	1.5
Access overhead	185.20 %	250.00 %	0.74
# of matrices	15	8	1.85
Total cost of access	27.78	20	1.39

7 Performance Results

Table 5 presents performance results for both scenarios. In our tests we used a single NVIDIA GTX TITAN GPU with Intel Core i7–3770 CPU. The MPDATA algorithm was tested for the grid of size $392 \times 256 \times 64$. The achieved results are far from the peak performance due to the complexity of the algorithm, strong instructions and data dependencies, and shared memory size limitations.

Table 5. Performance results for both scenarios

Kernel	Mflops per scenario	Time per scenario [ms]	Performance [Gflop/s]	Speedup
B and C	963.4	15.47	62.29	1
BC	847.8	10.47	80.95	1.48

Our method of stencils distribution across CUDA kernels allows for increasing the MPDATA performance by about 1.48 times. Such a speedup is a little higher than we expected due to the fact that compression of kernels brings some additional advantages, which were not taken into account in our analysis. The main reason of a higher speedup is possibility of applying the common subexpression elimination to reduce the number of MPDATA instructions. It allows us to reduce the number of operation from 963.4 Mflops to 847.8 Mflops.

8 Conclusions and Future Work

The proposed methods allow for estimating "the best" number of kernels, as well as easy selection of CUDA block size for each kernel. The compression of stencils into kernels and improvement of GPU occupancy are mutual excluded. However, the analysis of GPU global memory transactions allows us to find the compromise between these two kinds of optimizations. Moreover, the compression of kernels permits for decreasing the amount of computation. The proposed approach to kernel processing with queues of data placed in registers and shared memory increases the data locality significantly. The performance of kernels in our approach is limited by the number of memory transactions and latency of arithmetic operations. The GPU utilization is mostly limited by the size of shared memory.

Our parallelization of the EULAG model is still under development. The future work will focus on expansion of the implementation across a cluster of CPU-GPU nodes. The particular attention will be paid to implementation of MPDATA using OpenCL in order to ensure the code portability across different devices, as well as development of autotunig mechanisms aiming at providing performance portability.

Acknowledgments. This work was partly supported by the Polish National Science Centre under grant no. UMO-2011/03/B/ST6/03500.

References

1. Cecilia, J.M., García, J.M., Ujaldón, M.: Cuda 2D stencil computations for the Jacobi method. In: Jónasson, K. (ed.) PARA 2010, Part I. LNCS, vol. 7133, pp. 173–183. Springer, Heidelberg (2012)

2. Ciznicki, M., Kopta, P., Kulczewski, M., Kurowski, K., Gepner, P.: Elliptic solver performance evaluation on modern hardware architectures. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2013, Part I. LNCS, vol. 8384, pp. 155–165. Springer, Heidelberg (2014)
3. de la Cruz, R., Araya-Polo, M., Cela, J.M.: Introducing the semi-stencil algorithm. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2009, Part I. LNCS, vol. 6067, pp. 496–506. Springer, Heidelberg (2010)
4. Hager, A., Wellein, G.: Introduction to High Performance Computing for Science and Engineers. CRC Press, Boca Raton (2011)
5. Kurowski, K., Kulczewski, M., Dobski, M.: Parallel and GPU based strategies for selected CFD and climate modeling models. Environ. Sci. Eng. **3**, 735–747 (2011)
6. Nguyen, A., Satish, N., Chhugani, J., Changkyu, K., Dubey, P.: 3.5-D blocking optimization for stencil computations on modern CPUs and GPUs. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–13 (2010)
7. NVIDIA Kepler Compute Architecture. http://www.nvidia.com/object/nvidia-kepler.html
8. Rojek, K., Szustak, L.: Parallelization of EULAG model on multicore architectures with GPU accelerators. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Waśniewski, J. (eds.) PPAM 2011, Part II. LNCS, vol. 7204, pp. 391–400. Springer, Heidelberg (2012)
9. Smolarkiewicz, P.: Multidimensional positive definite advection transport algorithm: an overview. Int. J. Numer. Meth. Fluids **50**, 1123–1144 (2006)
10. Szustak, L., Rojek, K., Gepner, P.: Using Intel Xeon Phi coprocessor to accelerate computations in MPDATA algorithm. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2013, Part I. LNCS, vol. 8384, pp. 582–592. Springer, Heidelberg (2014)
11. Wyrzykowski, R., Rojek, K., Szustak, L.: Using Blue Gene/P and GPUs to accelerate computations in the EULAG model. In: Lirkov, I., Margenov, S., Waśniewski, J. (eds.) LSSC 2011. LNCS, vol. 7116, pp. 670–677. Springer, Heidelberg (2012)
12. Wyrzykowski, R., Szustak, L., Rojek, K., Tomas, A.: Towards efficient decomposition and parallelization of MPDATA on hybrid CPU-GPU cluster. In: LSSC 2013. LNCS (in print)