Porting Multiscale Fluid Model EULAG to Modern Heterogeneous Architectures

Bogdan Rosa, Miłosz Ciżnicki, Krzysztof A. Rojek, Damian K. Wójcik, Piotr K. Smolarkiewicz, and Roman Wyrzykowski

***Abstract*—The goal of this study is to adapt the multiscale fluid solver EULAG [Prusa *et al*., Computers & Fluids, vol 37, 2008] to the future GPU-based high-performance computing platforms. The EULAG model has a proven record of successful applications in a range of environmental fluid dynamics, and excellent efficiency and scalability on conventional supercomputer architectures. Currently, the model is being implemented as a new dynamical core of COSMO (Consortium for Small-scale Modeling) weather prediction framework. The EULAG code combines features of a stencil and point wise computations. Its communication scheme consists of both halo exchange subroutines and global reduction functions. Within the project, two main modules of EULAG, namely the multidimensional positive definite advection transport algorithm, MPDATA, and the variational generalized conjugate residual, GCR, elliptic pressure solver are analyzed and optimized. Relevant techniques have been chosen and applied to accelerate code execution on modern GPU architectures: stencil decomposition, block decomposition (with weighting analysis between computation and communication), reduction of intercache communication by partitioning of cores into independent teams, cache reusing and vectorization.**

Testing and validation of the new GPU implementation have been carried out based on modeling decaying turbulence of a homogeneous incompressible fluid in a triply-periodic cube. Simulations performed using the standard version of EULAG and its new GPU implementation give similar solutions. Preliminary results of the parallel performance of the new implementation show a promising increase in terms of computational efficiency.

***Index Terms*—Anelastic model, EULAG, hybrid architectures, parallel computing.**

I. Introduction

In recent years there has been growing interest in employing heterogeneous and hybrid supercomputing architectures for modeling complex physical processes. Especially promising application for the new (CPUs/GPUs) architectures is computational fluid dynamics (CFD) and particularly the numerical weather prediction (NWP). Adaptation of conventional CFD codes to modern supercomputing architectures offers a unique opportunity for modeling complex physical flows with accuracy greater than ever before. The new parallel computers based on multi- and many-core processors (CPUs) or graphics processing units (GPUs) enable to increase computational efficiency and reduce energy consumption. Consequently, more computational resources, i.e. processing units and memory can be employed. This in turn allows increasing complexity of the models, so that more details which may affect the evolution of the system can be captured. To be able to run the traditional codes efficiently on the new hybrid platforms it is necessary to redesign their structures. Nowadays, several research centers around the world are involved in various projects aimed at adapting weather forecasting models to future high-performance computing platforms. One of such priority projects "Performance on Massively Parallel Architectures," POMPA, has been launched in 2010 by the COSMO [1] consortium. The goal of the POMPA project is to develop a prototype implementation of the current COSMO NWP model for modern GPU and CPU based computing hardware. To-date results show a large potential of the new implementations in terms of reduction of time-to-solution. It is worth noting that the hardware costs for running the redesigned model, both on CPU-based and GPU-based machines, are significantly lower. The newly developed dynamical core is robust and capable of running COSMO-7 (horizontal resolution 7 km) and COSMO-2 (horizontal resolution 2.2 km); grid resolution ~2 km is nowadays common in regional weather forecast models used in operation.

Another example of a successful GPU implementation is the Weather Research and Forecasting (WRF) Model [2]. WRF is a mesoscale numerical weather prediction system designed for both atmospheric research and for operational forecasting. Porting the microphysics module to GPU allowed obtaining a 10-fold increase in its computational performance. The microphysics module is an important and computationally demanding component of the WRF model. The module represents only 1% of the total source code but its adaptation to GPU resulted in a 20% increase in the computational performance of the entire model. Adapting of selected modules of WRF to GPU allowed to achieve tremendous speedups in modeling weather forecast, ocean dynamic and tsunami waves [3].

The adaptation of traditional NWP codes to the new machines based on GPUs allows increasing numerical efficiency and enables to take full advantage of the available computational resources. This offers a unique opportunity to develop simulations with finer grid resolutions and computational domains larger than ever before. Refined grid

Manuscript received February 25, 2014; revised April 25, 2014. This work was supported by the Polish National Science Centre under grant no. UMO-2011/03/B/ST6/03500.

B. Rosa and D. K. Wójcik are with the Institute of Meteorology and Water Management - National Research Institute, Podleśna 61 Street, 01-673 Warsaw, Poland (e-mail: bogdan.rosa@imgw.pl).

P. K. Smolarkiewicz is with the European Centre for Medium-Range Weather Forecasts, Reading, RG2 9AX, UK.

M. Ciżnicki is with the Poznan Supercomputing and Networking Center, Noskowskiego 10 Street, 61-704 Poznań, Poland.

K. A. Rojek and R. Wyrzykowski are with the Czestochowa University of Technology, Dabrowskiego 69 Street, 42-201 Czestochowa, Poland.

resolutions in simulation of meso- and large-scale atmospheric flows for NWP and climate studies may have profound impact on improving the reliability of prognoses. This is because at the convective scales of O(1) km, flows are highly turbulent and contain a significant amount of energy [4]. In order to explicitly resolve or even admit convective processes, grid spacing has to be fine enough. Due to computational constraints these processes are grossly underresolved in today's simulations. The new methods and algorithms at work on modern architectures should allow to dispense with a large part of convective parameterizations in global models and improve numerical weather forecasts.

The aim of this study is to develop effective methods and algorithms for adapting multiscale model EULAG [5] for heterogeneous and hybrid supercomputing architectures. EULAG belongs to a class of numerical models for low Mach number flows in geo- and astrophysics. The dynamical core of EULAG is based on the non-hydrostatic Euler equations, either fully compressible or anelastic. The model employs generalized curvilinear coordinate description [6], finite-volume non-oscillatory transport algorithm MPDATA [7] and the advanced elliptic solver GCR [8]. Since 2008, EULAG is a candidate for the dynamical core of a very-high resolution NWP model of the COSMO consortium. The dynamical core of EULAG has considerable advantages concerning conservation properties. Moreover, modeling of atmospheric flows with EULAG does not impose severe constraints on the maximal allowable steepness of the surface orography.

In this paper we present our recent efforts aimed at adapting two main modules of EULAG, namely advection algorithm MPDATA and iterative elliptic pressure solver GCR to GPU based supercomputers. We describe the crucial detail of different strategies used to accelerate code execution, namely stencil decomposition, block decomposition (with weighting analysis between computation and communication), reduction of inter-cache communication by partitioning of cores into independent teams, cache reusing and vectorization.

The remainder of the paper is organized as follows. Section II describes the key details of the EULAG code. In Section III main concepts of the new MPDATA implementation and some preliminary results of performance tests are presented. Analogously, Section IV contains the discussion of the new GCR implementation. Validation of the new GPU implantation is presented in Section V. Key conclusions are summarized in Section VI.

II. Methodology

Porting the EULAG solver to modern GPU architectures is a highly complex task; therefore, the entire process had to be divided into several stages. Essential programming work was preceded by a number of preliminary tasks. The initial efforts were focused on extracting the main driver with MPDATA advection and the GCR elliptic solver subroutines from the main code. We used the latest version of the EULAG in which parallelization scheme is based on the three-dimensional spatial MPI domain decomposition [9]. Special options and extensions of no interest to this project were removed from the code. The call tree has been reformulated to better expose mathematical structure of the code. Further, careful analysis of the structure of memory references has been done. We studied different strategies for overlapping computations and interprocesses communication.

EULAG employs pure MPI programming model for parallelization between all cores. The computational grid is divided in all three dimensions and each MPI process advances the solution in its subdomain. The computational domain is decomposed evenly so that MPI processes have the same number of grid points and the same computational load. All-to-all communications are required for computing global reduction operations, whereas point-to-point communications exchange halo regions between the nearest neighbors in X, Y and Z dimensions. The data arrays are explicitly dimensioned to contain a subgrid of a total array corresponding to the entire model grid, plus an extra space for a copy of the neighboring processors' boundary cells, commonly referred as "hallo cells". Each subgrid is then assigned to only one processor, though the halo regions may vary through the code. To minimize communication cost, the blocks of data corresponding to halo regions are exchanged partially and only when needed.

III. Adaptation of Advection Transport Algorithm MPDATA to GPU Architectures

In this Section, we describe our new approach that allows efficiently distribute computational tasks of MPDATA across GPU resources. The proposed technique is based on stencil computations and is an extension of the concept described in previous studies [10]-[13]. In this paper, we assume prior knowledge about OpenCL programming and terminology. For an informative description of the crucial aspects of GPU programming, the reader is referred to reference [14].

The MPDATA algorithm is a set of 17 stencils [15], where each stencil may depend on one or more others. MPDATA requires loading 5 input matrices and returns only one. We assume that the size of the computational grid is $n \times m \times l$. In previous work [11], it has been shown that MPDATA is strongly memory-bounded. The main bottleneck in adaptation of MPDATA to GPU architecture resides in memory traffic between global and local GPU memory. Our idea of adaptation is based on an appropriate distribution of stencils onto GPU kernels in order to minimize the number of GPU memory transaction between local and global memory. For this purpose we propose a method, where a different number of GPU kernels is considered. In each configuration, a single kernel processes a different number of stencils. For each configuration, we estimate a number of GPU memory transactions and then select the configuration, where the number of memory transaction is minimized.

A. GPU Kernel Processing

Each GPU kernel is processed by $n \times m$ work-items (GPU threads), that are grouped into work-groups. Each work-item is responsible for computing one element of data grid. Each work group contains $g1 \times g2$ work-items, so the total number

of work-groups is $n/g1 \times m/g2$. Fig. 1 illustrates a single data chunk of MPDATA matrices, which is processed by a single work-group. In this paper, we focus on the analysis of coalesced and uncoalesced memory areas of the data chunk.

To increase data locality within work-groups, we employ widely used method of 2.5D blocking [16] in which two dimensional work-groups are responsible for computing $g1 \times g2$ data chunks. The loop inside kernel is used to traverse the grid in the l dimension. Since, the MPDATA algorithm requires to store at most $3 \times (g1 \times g2)$ data chunks at the same time, we use a queue of data chunks placed in registers and local memory.

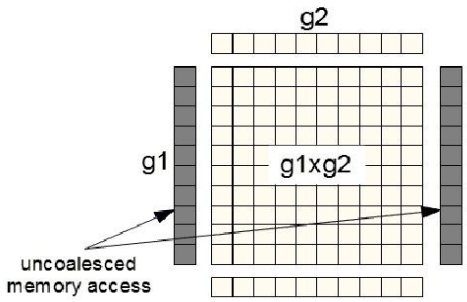

Fig. 1. A single data chunk of MPDATA matrix with its halo areas processed by a single work-group.

In this approach, we first copy the data from GPU global memory to registers, and then, for each iteration across l dimension, we move the data between registers and local memory. This method is illustrated in Fig. 2.

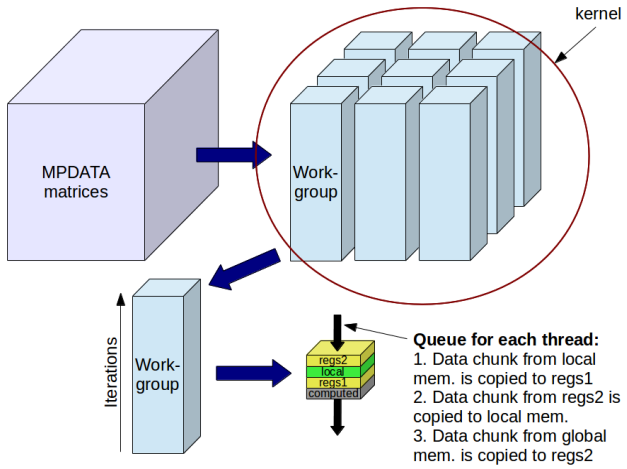

Fig. 2. GPU kernel processing.

B. Implementation Tuning

The main challenge in the efficient adaptation of MPDATA to GPU platforms is reduction of global memory usage and optimization of data transfer. In the new approach we address this problem through a comprehensive analysis of the data flow. Distribution of computational tasks is preceded by estimation of local memory utilization, sizes of halo areas (ghost zones), data dependencies between and within stencils. Based on such extensive analysis we are able to specify the most favorable number of GPU kernels and set an optimal distribution of stencils across kernels and the sizes of work-groups for each kernel. As a consequence, a load balancing is maintained and data communication is minimized and well structured.

Our analysis is based on the estimation of the number of GPU memory transactions (assuming 64-bits access mode) from global to local memory (G_f) and from local to global memory (G_t).

$$G_f(g1,g2) = ceil((g2 + j_p)/u_{ls}) \times g1 + ceil(g2/u_{ls}) \times (i_m + i_p) + g1 \times j_m, \quad (1)$$

$$G_t(g1,g2) = ceil(g2/u_{ls}) \times g1 \quad (2)$$

where $ceil\ (x)$ returns a rounded up value of x, u_{ls} is the number of load/store units for each compute unit. The halo area can have a different size for each side of the $g1 \times g2$ data chunk: i) $i_p \times g2$ from the top; ii) $i_m \times g2$ from the bottom; iii) $j_m \times g1$ on the left side, and finally iv) $j_p \times g1$ on the right side. The number of bytes that needs to be allocated in the local memory to store $g1 \times g2$ data chunk can be computed as follows

$$L_{mem}(g1,g2) = (g1 + i_m + i_p)(g2 + j_m + j_p) \times S_{el} \quad (3)$$

where S_{el} is size of the grid element in bytes.

The number of transactions to the GPU global memory can be greatly reduced by merging two stencils into one. To perform this task a special procedure has been developed. The procedure has a build-in set of conditions, including:

1) if merged stencils are independent, then the local memory usage and the number of transactions is accumulated from each stencil;
2) if the output matrix of the first stencil is among the input matrices of the second stencil, then the halo area of the first stencil is extended by the halo area of the second stencil, otherwise
3) The halo area is a maximum from the halo areas for each stencil.

This procedure is called repeatedly in order to build a set of possible configurations of kernels. Finally, the simple minimum algorithm estimates the best configuration of MPDATA, including the number of kernels, the mapping of stencils onto kernels, and the work-groups sizes.

C. Parallel Performance

To examine the scalability of the MPDATA algorithm in the stand-alone version, a number of numerical experiments have been performed. All tests presented in this paper were conducted on NVIDIA GTX TITAN GPU [17]. This GPU graphics card is based on the Kepler architecture and includes 14 streaming multiprocessors (SMX), each consist of 64 double precision units (DP units) with 48 KB of shared memory and 16 KB of L1 cache.

We compare parallel performance of the original CPU version of MPDATA with the new GPU implementation. The CPU used in the test is Intel Core i7-3770 with 3.4 GHz clock frequency. The input data were defined as an array of random values. The sizes of grid range from $16 \times 16 \times 16$ to $512 \times 512 \times 64$. The CPU tests have been performed for a sequential (one core) and parallel (4 cores) version of MPDATA.

The first test was conducted for 100 time steps. The performance results are shown in Table I. Our GPU implementation allows achieving speedup of about 5 over the

parallel CPU version and of about 15.7 over the sequential version.

TABLE I: EXECUTION TIME OF GPU AND CPU VERSIONS FOR 100 TIME STEPS

	CPU		GPU [s]
n ×m ×l	1 core [s]	4 cores [s]	
16 ×16 ×16	0.044	0.016	0.087
32 ×32 ×16	0.164	0.048	0.100
64 ×64 ×16	0.636	0.192	0.112
64 ×64 ×64	2.216	0.776	0.366
128 ×128 ×64	10.484	3.316	0.792
128 ×128 ×128	20.517	6.624	1.583
256 ×256 ×64	40.374	12.868	2.560
256 ×256 ×128	-	24.373	5.221
256 ×256 ×256	-	52.776	10.516
512 ×512 ×64	-	50.743	9.689

The speedups for all grid sizes are listed in Table II. The GPU version is profitable for mesh sizes greater than or equal to 64×64×16. Computations on smaller grids do not allow taking full advantage of GPU resources; hence the GPU performance is rather weak for these cases.

TABLE II: SPEEDUP

	Speedup	
n ×m ×l	1 core/GPU	4 cores/GPU
16 ×16 ×16	0.51	0.18
32 ×32 ×16	1.64	0.48
64 ×64 ×16	5.68	1.71
64 ×64 ×64	7.15	2.12
128 ×128 ×64	13.24	4.19
128 ×128 ×128	13.15	4.18
256 ×256 ×64	15.77	5.03
256 ×256 ×128	-	4.67
256 ×256 ×256	-	5.02
512 ×512 ×64	-	5.24

Significantly better results of the code scalability have been obtained in the second test. The second test has been performed for 1000 time steps. The speedup results are illustrated in Fig. 3. In the longer simulation, we achieved speedup about 15 times comparing to the CPU parallel version and about 40 times comparing to the CPU sequential version.

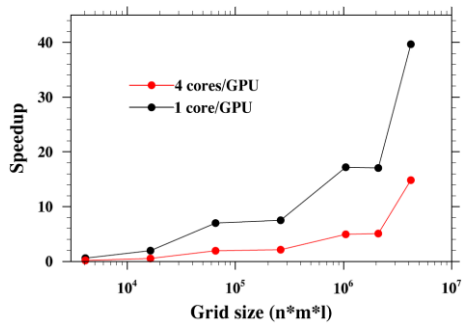

Fig. 3. Speedup of GPU version over CPU versions for 1000 time step.

IV. ADAPTATION OF ELLIPTIC SOLVER COMPUTATION TO GPU ARCHITECTURES

A. Main Objectives

Several different techniques for porting the GCR elliptic solver to hybrid architectures have been presented in previous study [18]. The proposed techniques rely on porting MPI-all code (which uses MPI exclusively) to the hybrid version MPI+OpenMP. The body of the elliptic solver consists of five major routines. The main routine advances the solution iteratively by calling other major computational routines. The elliptic solver invokes the collective communication to compute grid global values. The routines *prforc* and *divrhs* initialize the solver. The *prforc* evaluates the first guess of the updated velocity - combining the explicit part of the solution and the estimate of the generalized pressure gradient - while imposing the appropriate boundary condition. The *divrhs* evaluates the density weighted divergence of that velocity, and thus the initial residual error of the elliptic problem for pressure. The most computationally intensive routine of the GCR is *laplc* that evaluates iteratively generalized laplacian operator (a combination of divergence and gradient) acting on residual errors. An important part of the solver is the preconditioner *precon* that accelerates the convergence of the variational scheme. By performing direct matrix inversion in the vertical, it is especially useful for large-scale simulations on thin spherical shells with grids characterized by large anisotropy. The routine *precon* employs sequential Thomas algorithm to solve tridiagonal system of equations with the right hand side consisting of the horizontal divergence of the generalized horizontal gradient, evaluated by *nablaCnablaxy*, the second most computationally intensive routine of the elliptic solver. To sum up, the computational loops within the elliptic solver can be simply divided, with regard to the data access pattern, into three categories: reductions, implicit methods – the Thomas algorithm and explicit methods – the stencils.

B. Parallel Performance

TABLE III: AVERAGE VALUES PER GRID CELL

		precon	prforce	divrhs	laplac	GCR
CPU	Ops	18	27	15	33	297
	Bytes	255	289	170	604	4329
	CI	0.07	0.09	0.09	0.05	0.07
	Ops	22	27	17	32	291
GPU	Bytes	167	154	113	242	2443
	CI	0.13	0.17	0.15	0.13	0.13
CI GPU/CPU [%]		185	184	167	239	189

To evaluate numerical intensity of the new GPU implementation we made use of the standard metric for measuring computational performance. The metric is defined for each function as the ratio of the number of arithmetic operations to the number of required bytes. The number of bytes is computed for double precision quantities and takes into account both reading and writing operations. The constant values are not included in the metric, as they can be easily cached by the compiler in the registers.

Table III shows the computational intensities (CI) of the main GCR functions for both CPU and GPU code. We compare them with the computational intensity of Kepler GPU (http://www.nvidia.com/object/tesla-servers.html) to show which hardware resources affect the code efficiency more.

In the Kepler K20 accelerator the time needed for computing 8 double precision operations is equal to the read/write time of one byte of data from/to the global memory. Thus, all functions in the elliptic solver are bound by the global memory bandwidth. Each of them requires more than 10 bytes of data to compute one operation. Increasing the computational intensity can optimize the memory-bound functions. This, in turn, can be achieved by both reducing of the memory traffic and increasing the number of arithmetic operations. To address this issue we analyze the data dependencies between the computational loops. Based on the analysis we try to join them to increase the computational intensity. The next paragraphs describe methodology of optimization which has been applied to the selected functions of the elliptic solver.

C. Preconditioner Optimization

The preconditioner employs the sequential Thomas algorithm [19] to solve tridiagonal systems of equations. There are two different implementations of the algorithm that depend on the parallelization method or, equivalently on the domain decomposition scheme. The standard version is dedicated for the 2D horizontal decomposition. For implementations that use the 3D domain decomposition, the version based on recurrence doubling approach is more suitable. The detailed description of the recurrence doubling version can be found in [20]. Equation (4) shows the structure of the 2D horizontal decomposition

$$\text{for } k = 3...l,\ j = 1...mp,\ i = 1...np$$

$$f(i,j,k) = \left(p33(i,j,k-1) \times f(i,j,k-2) + r(i,j,k)\right) \times dni(i,j,k)$$

$$\text{for } k = l-2...1,\ j = 1..mp,\ i = 1...np \qquad (4)$$

$$p(i,j,k) = e(i,j,k) \times p(i,j,k+2) + f(i,j,k)$$

where the *np* and *mp* values are sub-domain sizes in the horizontal direction. The data dependency in both loops implies that this decomposition is more suitable for use on a single graphic card. The numerical scheme of our implementation is shown in (5).

$$fsm(1) = freg1 = f(1);\ fsm(2) = freg0 = f(2)$$

$$\text{for } k = 3...l$$

$$freg2 = freg1;\ freg1 = freg0$$

$$fsm(k) = freg0 = \left(p33(k-1) \times f(k\text{-}2) + r(k)\right) \times dni(k)$$

$$preg1 = p(l);\ preg0 = p(l-1) \qquad (5)$$

$$\text{for } k = l-2...1$$

$$preg2 = preg1;\ preg1 = preg0$$

$$p(k) = preg0 = e(k) \times preg2 + fsm(k)$$

To minimize the global memory traffic two loops that compute the *f* and *p* array, have to be joined into one kernel. The kernel is a function executed by each thread on the vertical column. The computation of the *f* array is done by using registers to eliminate global memory access in the *k*-2 direction. The shared memory could be used to cache access in the *k-2* direction and still the registers would provide lower access latency. Next, the *f* array is saved in the shared memory. To compute the *p* array the *f* array is read from the shared memory and similarly, the usage of registers eliminates the *k*+2 global memory access pattern. Those optimizations improved the computational intensity about 40% from 0.07 to 0.1, mainly by reducing the number of required bytes.

D. Stencil Optimization

Here we demonstrate the new methodology for stencil computation taking as an example the horizontal Laplacian. The stencil is a function that independently updates each cell in a computational grid. The update is defined by a pattern that indicates which neighbors of the cell take part in the computation. The horizontal Laplacian contains four computation loops defined as the stencils. The stencils have access only to one neighboring cell on both sides for the *i*-th and *j*-th directions, see Fig. 4a for details. In particular situations when the required data on domain boundaries are not available, the boundaries have to be handled specifically. We joined all computational loops into one kernel to improve the computational intensity, see Fig. 4b.

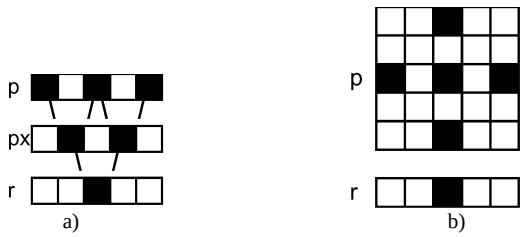

Fig. 4. a) Two stencils computing *px* (partial derivative of *p* in *x* direction) and *r* (partial derivative of *px* in *x* direction) in the *i*-th direction. The same applies to *j*-th direction. b) The stencil pattern in the *i*-th and *j*-th directions after transformation.

This transformation creates a larger stencil with the new access pattern. In this new pattern each cell accesses a distant neighbor that is one cell away from the currently updated position. The distant neighbors, similarly to stencils before transformation, have to be accessed on both sides for the *i*-th and *j*-th directions. We use 2.5 D spatial blocking technique proposed in [16] to efficiently cache the neighbor cells in the shared memory and reduce the global memory traffic. That is, we read the horizontal sub-planes, defined by the *i*-th and *j*-th directions, to shared memory and iterate through the *k*-th direction. The additional data, called margin, is read to the shared memory in order to compute properly cells on the sub-plane boundaries. Thus, some cells are read more than once and their number depends on the sub-plane dimensions. These dimensions are mainly constrained by the size of the shared memory. We minimize the number of the global memory transactions by changing the dimensions of the sub-planes. The memory transactions are defined at the

granularity of cache lines. The sub-plane dimensions that are close in shape to square limit the margin size. On the other hand the sub-plane dimensions have influence on the number of so-called coalesced memory transactions. The coalesced memory transactions define cache lines fully loaded with useful data. The higher size of the sub-plane dimension on the longest direction, in which following cells are placed in consecutive memory addresses, the larger number of the coalesced memory transactions. Therefore, we try to find the sub-plane dimensions that optimize both goals. All described optimizations improved the computational intensity from 0.06 to 0.22 and reduced the number of moved bytes to/from global memory per cell by a half.

E. Computational Intensity of the New Implementation

Parallel performance of the new GPU-implementation have been tested using dual socket Intel Xeon E5-2670 CPU with 16 cores clocked at 2.6 GHz and Kepler K20 GPU. To compare the performance of CPU and GPU processors solely, the data transfer time between RAM and GPU is not included in case of the GPU timings. It is reasonable to assume that all data are available on the global memory as we transfer the full data set to CPU after every one hundred time steps.

TABLE IV: PERFORMANCE RESULTS – TEST 1

Test 1					
n × m × l	CPU - 16 cores		GPU		Speedup
	[ms]	[GFLOPS]	[ms]	[GFLOPS]	
16 × 16 × 16	0.4	2.31	3	0.28	0.13
32 × 32 × 32	1.2	6.15	3.6	1.86	0.33
64 × 64 × 64	11.6	5.09	7.3	7.34	1.59
128 × 128 × 128	113	4.18	36.5	11.75	3.10
256 × 256 × 256	902	4.19	264	12.99	3.42
512 × 256 × 256	1741	4.34	506	13.56	3.44

TABLE V: PERFORMANCE RESULTS – TEST 2

Test 2					
n × m × l	CPU - 16 cores		GPU		Speedup
	[ms]	[GFLOPS]	[ms]	[GFLOPS]	
64×32×16	2.6	10.07	14	2.12	0.19
128×64×32	23.5	8.92	28	8.46	0.84
256×128×64	382.3	4.38	136	13.94	2.81
512×256×128	3016.5	4.32	967	15.69	3.21
1024×256×128	6053.4	4.43	1871	16.21	3.24

Table IV and Table V show the performance of the CPU and GPU implementations obtained in two different numerical tests.

The test 1 refers to the flow in a cube with triply periodic boundaries. The test 2 is a simulation of the flow on a sphere with non-periodic boundaries in the vertical. Additionally, the test 2 utilizes the preconditioner in the horizontal direction. Both tests were conducted with only one GCR iteration. It turned out that, the GPU implementation is about 3 times faster than the CPU implementation. The speedup grows with the increase of domain sizes as the kernels utilize GPU resources more efficiently. The GPU code achieves 72% of the peak performance for the test 1 and up to 86% for the test 2. The peak performance is defined as the product of the measured global memory bandwidth and the computational intensity. The measured global memory bandwidth for Kepler K20 is 144GB/s. The peak performance corresponds to the ideal situation in which data transfer and computations perfectly overlap. This occurs only if there are no memory latency effects and when the bandwidth of the global memory is completely saturated.

V. 3D INCOMPRESSIBLE TURBULENT FLOW; VALIDATION TEST CASE

The new GPU implementation of the GCR solver is validated using a standard benchmark test case for incompressible flow solvers. We simulate decaying turbulence of a homogeneous incompressible fluid. Here, only simplified setup proposed by Taylor and Green [21] is considered. This problem was originally used to illustrate processes of grinding down of large eddies into smaller ones. The initial conditions for the velocity are slightly modified compared to the original work [21], namely

$$\begin{aligned} u &= \sin(ax)\cos(by)\cos(cz) \\ v &= \cos(ax)\sin(by)\cos(cz) \\ w &= \sin(ax)\cos)by) \end{aligned} \tag{6}$$

where $a = 2/(n-1)dx$, $b = 2/(m-1)dy$, $c = 2/(k-1)dz$. Here, m, n and k are integers whereas dx, dy and dz are grid spacings in all three directions. Our computational domain is a triply-periodic cube of length 2π. Grid points are uniformly distributed in each spatial direction. The size of the computational grid is 128^3. There is no external forcing so the flow is driven by the turbulent energy cascade. We compare results from simulations performed with two complete versions of EULAG.

The first simulation has been performed using the traditional CPU architecture and the standard version of EULAG. To perform the second simulation we used the new GPU implementation of the GCR solver and graphic card NVIDIA Kepler K20.

First, we compare 2D velocity flow field obtained in both

simulations. Fig. 5 shows the vertical component of velocity in the bottom horizontal plane. There is good agreement between solutions computed with the standard version of EULAG and the new GPU implementation. The black and red lines precisely overlap.

To compare the results from simulations in the entire domain the three-dimensional visualization of the vorticity flow field has been prepared. The results are shown in Fig. 6. Again, we confirm the perfect agreement between simulations performed on two different hardware platforms.

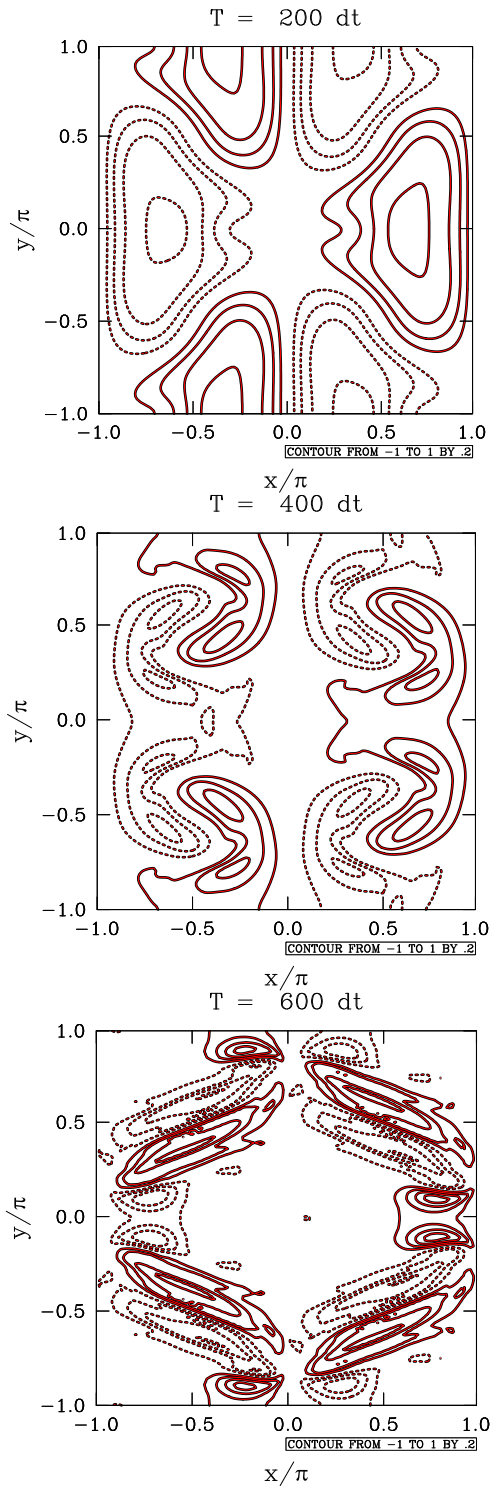

Fig. 5. Time evolution of the vertical component of velocity, displayed in the bottom (z = -π) horizontal cross section through the domain. Black lines correspond to simulations with standard CPU version of EULAG. Red contours represent solution from the new GPU-implementation. The dashed lines indicate negative values.

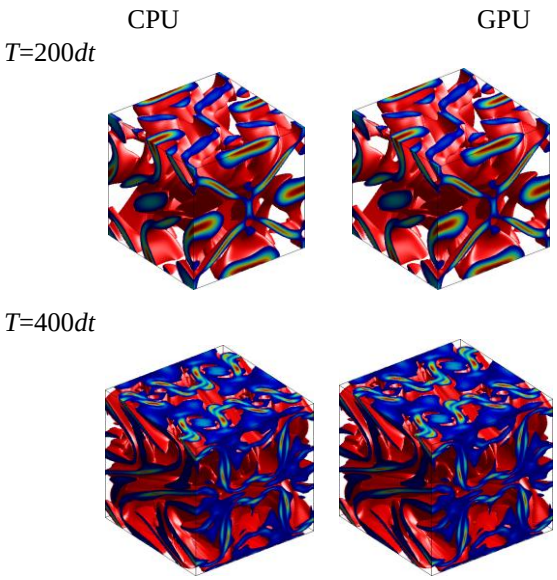

Fig. 6. Isosurfaces of vorticity magnitude from simulation at grid 128³. On the left side are results from simulations performed with the traditional (CPU) version of EULAG. On the right side are results from the new GPU implementation. The top panels present the flow after 200 time steps. The two bottom panels show the vorticity field after 400 time steps.

VI. Conclusion

In this paper, we reported on our efforts in adapting multiscale model EULAG to modern GPU-based architectures. Two main modules of EULAG, namely advection algorithm MPDATA and iterative elliptic pressure solver GCR have been redesigned and the new organization of computations has been implemented. The new implementation performs better than the conventional code, and can take full advantage of modern heterogeneous architectures. The scalability tests were performed using two different graphic cards. It was found that the speedup grows with the domain size as the kernels can utilize GPU resources more efficiently.

Porting the multiscale model EULAG to modern architectures opens bright perspectives for further progress in fundamental research and in applied fields that are closely related to computational fluid dynamics. It is expected that improved performance will allow reproducing more faithfully meteorological processes occurring in the real atmosphere. The new development allows performing simulations in a larger domain and thereby extends the range of scales what in turn may results in a more reliable operational weather forecast.

Acknowledgment

The authors would like to express gratitude to Piotr Kopta (PSNC), Michał Kulczewski (PSNC), Krzysztof Kurowski (PSNC), Zbigniew P. Piotrowski (IMWM-NRI) and Łukasz G. Szustak (CzUT) for their scientific programming, valuable comments and suggestions.

References

[1] The Consortium for Small-scale Modeling. [Online]. Available: http://www.cosmo-model.org.
[2] The Weather Research and Forecasting Model. [Online]. Available: http://www.wrf-model.org/index.php

[3] Weather, Atmospheric, Ocean Modeling, and Space Sciences. [Online]. Available: http://www.nvidia.com/object/weather.html
[4] W. C. Skamarock, "Evaluating mesoscale NWP models using kinetic energy spectra," *Month. Weather Rev.*, vol. 132, pp. 3019–3032, December 2004.
[5] J. M. Prusa, P. K. Smolarkiewicz, and A. A. Wyszogrodzki, "EULAG, a computational model for multiscale flows," *Comp. Fluids*, vol. 37, no. 9, pp. 1193–1207, 2008.
[6] P. K. Smolarkiewicz, C. Kühnlein, and N. P. Wedi, "A consistent framework for discrete integrations of soundproof and compressible PDEs of atmospheric dynamics," *J. Comput. Phys.*, vol. 263, pp. 185–205, 15 April 2014.
[7] P. K. Smolarkiewicz, "Multidimensional positive definite advection transport algorithm: an overview," *Int. J. Numer. Meth. Fluids.*, vol. 50, no. 10, pp. 1123–1144, 2006.
[8] P. K. Smolarkiewicz and J. Szmelter, "A nonhydrostatic unstructured-mesh soundproof model for simulation of internal gravity waves," *Acta Geophysica*, vol. 59, no. 6, pp. 1109–1134, 2011.
[9] Z. P. Piotrowski, A. A. Wyszogrodzki, and P. K. Smolarkiewicz, "Towards petascale simulation of atmospheric circulations with soundproof equations," *Acta Geophys.*, vol. 59, no. 6, pp. 1294–1311, 2011.
[10] L. Szustak and K. Rojek, "Parallelization of eulag model on multicore architectures with GPU accelerator," *Lect. Notes in Comp. Sci.*, vol. 7204, pp. 391–400, 2012.
[11] K. Rojek, L. Szustak, and R. Wyrzykowski, "Performance analysis for stencilbased 3D MPDATA algorithm on GPU architecture," *Lect. Notes in Comp. Sci.*, vol. 8384, pp. 145-154, 2014.
[12] L. Szustak, R. Wyrzykowski, and K. Rojek, "Using blue gene/p and gpus to accelerate computations in the eulag model," *Lect. Notes in Comp. Sci.*, vol. 7116, pp. 662–670, 2012.
[13] R. Wyrzykowski, L. Szustak, K. Rojek, and A. Tomas, "Towards efficient decomposition and parallelization of mpdata on hybrid CPU-GPU cluster," *Lect. Notes in Comp. Sci.*, vol. 8353, pp. 450-460, 2014.
[14] A. Munshi, B. R. Gaster, T. G. Mattson, J. Fung, and D. Ginsburg, *OpenCL - Programming Guide*, Boston, MA: Addison-Wesley, 2011, ch. 1, pp. 3-37.
[15] A. Schafer and D. Fey, "High performance stencil code algorithms for GPGPUs," *Computer Science*, vol. 4, pp. 2027-2036, 2011.
[16] A. Nguyen, N. Satish, J. Chhugani, K. Changkyu, and P. Dubey, "3.5-D blocking optimization for stencil computations on modern CPUs and GPUs," in *Proc. 2010 ACM/IEEE Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1-13.
[17] NVIDIA Kepler computer architecture. [Online]. Available: http://www.nvidia.com/object/nvidia-kepler.html.
[18] M. Ciznicki, P. Kopta, M. Kulczewski, K. Kurowski, and P. Gepner, "Elliptic solver performance evaluation on modern hardware architectures," *Lect. Notes in Comp. Sci.*, vol. 8384, pp. 155-165, 2014.
[19] J. Strikwerda, *Finite Difference Schemes and Partial Differential Equations*, Society for Industrial and Applied Mathematics, 2004, ch. 3.5, pp. 88-91.
[20] A. Wyszogrodzki, Z. P. Piotrowski, and W. W. Grabowski, "Parallel implementation and scalability of cloud resolving EULAG model," *Lect. Notes in Comp. Sci.*, vol. 7204, pp. 252–261, 2012.
[21] G. I. Taylor and A. E. Green, "Mechanism of the production of small eddies from large ones," in *Proc. the Royal Society of London. Series A, Mathematical and Physical Sciences*, vol. 158, no. 895, pp. 499-521, February 1937.

Bogdan Rosa received his M.Sc. degree in physics from the University of Warsaw in 2000, followed by the Ph.D. in 2005. Afterwards, he spent 3 years as a postdoctoral fellow at the University of Delaware, Department of Mechnical Engineering, where he was involved in developing computational tools to study collision rates and growth of droplets in atmospheric clouds. Since 2009, Dr. Rosa is working at the Institute of Meteorology and Water Management - National Research Institute. His current projects include adaptation of the numerical model EULAG into a German/Swiss weather prediction model COSMO.

Miłosz Ciżnicki has received M.Sc. in computer science from Poznan University of Technology in 2011. He is employed at Poznan Supercomputing and Networking Center affiliated to the Institute of Bioorganic Chemistry of the Polish Academy of Sciences. His research is focused on new hardware architectures, high performance computing, visualization and image compression.

Krzysztof A. Rojek received his M.Sc. in computer science from the Czestochowa University of Technology in 2008 and his PhD in 2012. During this period, his doctoral research focused on adaptation of high performance computing to the parallel processors architectures including Cell Broadband Engine and GPUs. Since 2012, Dr. Rojek is employed at Czestochowa University of Technology. His current work focused on automated performance tuning and adaptation of the EULAG numerical model to the graphics processor architectures.

Damian K. Wójcik received M.Sc. degree in computer science in 2009 and B.Sc. degree in physics in 2010 from the University of Warsaw. He is employed at the Institute of Meteorology and Water Management - National Research Institute. His current work is focused on coupling of the numerical model EULAG with the German/Swiss weather prediction model COSMO.

Piotr K. Smolarkiewicz received M.Sc. and Ph.D. degrees from the University of Warsaw, Poland, in geophysics and physical sciences in 1973 and 1980, respectively. Afterwards, in 1981, he went to Boulder, Colorado, to join as a postdoctoral fellow the Advanced Study Program at the National Center for Atmospheric Research (NCAR). After the postdoctoral appointment, since 1983 he has been a scientist at NCAR, in the rank of Senior Scientist since 1994. In March 2013 he joined the Research Department of the European Centre for Medium-Range Weather Forecasts (ECMWF) as a consultant.

Roman Wyrzykowski received M.Sc. and Ph.D degrees from the Kiev Polytechnic Institute in computer science in 1982 and 1986, respectively. Since 1982, he is employed at the Czestochowa University of Technology, Poland, where currently he is the head of Department of Computer and Information Science, and Director of Metropolitan Area Network in Czestochowa. His fields of expertise are: parallel and distributed computing, mapping algorithms onto parallel architectures, cluster and cloud technologies with applications. Prof. Wyrzykowski is the coordinator of the project "Methods and algorithms for organization of computations in the class of anelastic numerical models for geophysical flows on modern computer architectures with realization in the EULAG model" funded by the National Science Centre.