Using Blue Gene/P and GPUs to Accelerate Computations in the EULAG Model

Roman Wyrzykowski, Krzysztof Rojek, and Łukasz Szustak

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

Abstract. EULAG (Eulerian/semi-Lagrangian fluid solver) is an established computational model developed by the group headed by Piotr K. Smolarkiewicz for simulating thermo-fluid flows across a wide range of scales and physical scenarios. This paper presents perspectives of the EULAG parallelization based on the MPI, OpenMP, and OpenCL standards. We focus on development of computational kernels of the EULAG model. They consist of the most time-consuming calculations of the model, which are: laplacian algorithm (laplc) and multidimensional positive definite advection transport algorithm (MPDATA).

The first challenge of our work was parallelization of the laplc subroutine using MPI across nodes and OpenMP within nodes, on the BlueGene/P supercomputer located in the Bulgarian Supercomputing Center. The second challenge was to accelerate computations of the Eulag model using modern GPUs. We discuss the scalability issue for the OpenCL implementation of the linear part of MPDATA on ATI Radeon HD 5870 GPU with AMD Phenom II X4 CPU, and NVIDIA Tesla C1060 GPU with AMD Phenom II X4 CPU.

1 Introduction

Eulerian/semi-Lagrangian fluid solver (EULAG) [3] is an established computational model for simulating thermo-fluid flows across a wide range of scales and physical scenarios. Its important features are: nonoscillatory integration algorithms, robust elliptic solver, and generalized coordinate formulation enabling grid adaptivity technology. The EULAG model is an ideal tool to perform numerical experiments in a virtual laboratory with time-dependent adaptive meshes and within complex, and even time-dependent model geometries.

In this work, we focus on parallelization of two computational kernels of the EULAG model. They consist of the most time-consuming calculations of the model, which are: laplacian algorithm (laplc) [9] and multidimensional positive definite advection transport algorithm (MPDATA). While the starting point of our development was BlueGene/P - the IBM supercomputer with an innovative massive parallel architecture [4], we focus finally on GPUs which nowadays become [2] extremely promising multi-core architectures for a wide range of general-purpose applications demanding high-intensive numerical computations.

I. Lirkov, S. Margenov, and J. Waśniewski (Eds.): LSSC 2011, LNCS 7116, pp. 670–677, 2012.

In our research we used the BlueGene/P machine, which is located in the Bulgarian Supercomputing Center. This supercomputer consist of two racks, that include 2048 PowerPC 450 processors, which gives 8192 cores. Each node contains 2 GB RAM (4 TB RAM for two racks). This configuration supports the single-precision peak performance of 27.85 Tflops.

Current GPUs are highly efficient, multi-core processors, which have the computing power of several Tflops. GPUs offer a fast, inexpensive solution, but understanding the parallel trade-offs is crucial. These architectures allow for creating many thousands of threads, which has a significant influence on performance of parallel codes. Data transfers between GPU memory and RAM are strongly limited by the PCI Express bandwidth. Available software such as OpenCL and CUDA facilitate the implementation of general-purpose computation on GPUs using high-level programming languages and tools [2,6].

The material of this paper is organized as follows. In Section 2, architectures of BlueGene/P and two graphics cards from leading vendors NVIDIA and AMD are presented. Section 3 introduces computational kernels of the EULAG model, while Section 4 is devoted to OpenCL, an emerging parallel programming standard for multicore architectures. The key issues of parallelizing the computational kernels of the EULAG model on the target architectures are discussed in Section 5, while results of numerical experiments are presented in Section 6. Section 7 gives conclusions, and outlines further work.

2 Architecture Overview

Our research is based on two kind of architectures. The first one is the BlueGene/P supercomputer, while the second one are GPUs. We focus only on features of these architecture which are used in our work.

2.1 Architecture of Blue Gene/P

Each node of BlueGene/P [4] is a single application-specific integrated circuit (ASIC) with four IBM PowerPC 450 (PPC450) embedded 32-bit processor cores, arranged as an SMP. Each core contains L1 cache of a 32 KB instruction cache, and a 32 KB data cache. A dual-pipeline floating-point unit (FPU) is attached to each PPC450 core. It supports two simultaneous double-precision floating-point calculations in SIMD fashion. The dual-pipeline FPUs can simultaneously execute two fused multiply-add instructions per machine cycle, each of which is counted as 2 flops. Thus, each processor unit (PPC450 and FPU) has a peak performance of 4 flops per machine cycle, and the BPC chip with quadruple processor units rates at the peak performance of 16 flops per cycle (850 MHz), or 13.6 Gflops. A BG/P compute chip integrates PPC450 cores with L2 and L3 cache, memory controllers, and external 10 Gb/s network interfaces.

2.2 Architecture of GPUs

Our research is focused on NVIDIA Tesla C1060 and ATI Radeon HD 5870.

Architecture of NVIDIA Tesla C1060. NVIDIA Tesla C1060 [7] includes 10 Thread Processing Clusters (TPC). Every TPC contains 3 compute units. Each compute unit consists of 8 processing elements, and 16KB of local memory. It gives a total number of 240 available processing elements with a clock rate of 1296 MHz. It provides a peak performance of $240 * 1.296 * 2 = 622\ Gflops$ in single precision. This graphics accelerator card includes 4 GB of global memory with the peak bandwidth of 102.4 GB/s.

Architecture of ATI Radeon HD 5870. ATI Radeon HD 5870 [1] includes 20 compute units. Each compute unit consists of 16 processing elements, and 32KB of local memory. Each of the processing element is built of 5 streaming processors. It gives a total number of 1600 available streaming processors with a clock rate of 850 MHz, and provides the peak performance of $1600*0.850*2 = 2720\ Gflops$ in single precision. This accelerator card includes only 1 GB of global memory with the peak bandwidth of 153.6 GB/s.

3 The Scope of Our Research on the EULAG Model

The most time-consuming calculations of the EULAG model are two algorithms: laplacian algorithm (laplc) and multidimensional positive definite advection transport algorithm (MPDATA). To identify the most intensive computing we used TAU and gprof profilers.

3.1 Laplacian Algorithm

The laplc routine is summarized in three stages [3,9]. The first one is responsible for computing pressure derivatives:

$$p_i^k = \frac{p_{i+1}^k - p_{i-1}^k}{2} \cdot \delta k, \tag{1}$$

where p_i^k is a pressure in the i-th point of a mesh, k determines the dimension of the mesh, in which i is incremented, and δk is a distance between points in the mesh. The second stage is based on the following equation:

$$f_i^k = -p_i^k \cdot c_i, \tag{2}$$

where f is an interior pressure force, and c_i is a coefficient function. The last stage is responsible for computing the laplacian function, which is finally expressed by the following equation:

$$r_i = -\sum_k \frac{f_{i+1}^k - f_{i-1}^k}{2 \cdot c_i} \cdot \delta k. \tag{3}$$

3.2 Linear Version of MPDATA

The multidimensional positive definite advection transport algorithm [8] is based on the following equation:

$$\Psi_i^{n+1} = \Psi_i^n - \frac{\delta t}{\nu_i} \sum_{j=1}^{l(i)} F_j^{\perp} S_j, \tag{4}$$

where Ψ is a nondiffusive scalar field, S_j refers both to the face itself and its surface area, ν_i is the volume of the cell containing vertex i, while $F_j^{\perp}$ is interpreted as the mean normal flux through the cell face S_j averaged over temporal increment δt.

The approximation begins with specifying fluxes $F_j^{\perp}$:

$$F_j^{\perp} = 0.5(v_j^{\perp} + |v_j^{\perp}|)\Psi_i^n + 0.5(v_j^{\perp} - |v_j^{\perp}|)\Psi_j^n, \tag{5}$$

where advective normal velocity ($v_j^{\perp}$) is evaluated at the face S_j and assumes the following form:

$$v_j^{\perp} = \boldsymbol{S_j} \cdot 0.5[\boldsymbol{v_i} + \boldsymbol{v_j}]. \tag{6}$$

4 OpenCL: Emerging Standard for Multicore Architectures

Open Computing Language (OpenCL) is an open, royalty-free standard for parallel programming of heterogeneous computing platforms including CPUs, GPUs, and other processors like Cell/B.E [6,5,10]. It defines the host API for coordinating parallel computation across heterogeneous processors, and a programming language. OpenCL allows for creating portable code across different devices and architectures. The OpenCL allows applications to use OpenCL platform as a single heterogeneous parallel computer system.

The OpenCL platform consists of CPU (host) and one or more graphics cards (compute devices). Compute devices execute functions called kernels. Kernels are instanced as work-items that are grouped in work-groups. Work-items are executed as SIMD or SPMD on processing elements. Work-items executing a kernel have access to distinct memory regions.

5 Accelerating Computations in the EULAG Model

There are different methods of parallelization and parallel programming, which can be used depending on a target computer architecture. In this work, we utilize commonly adopted standards of parallel programing: MPI and OpenMP for BlueGene/P, as well as the emerging standard OpenCL for GPUs.

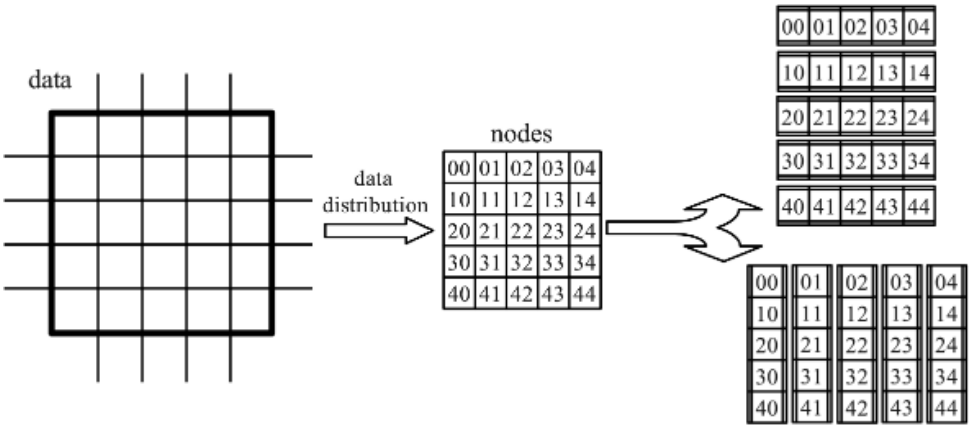

Fig. 1. Data distribution across array of nodes

5.1 Accelerating Computations in Laplc Kernel

One of the most efficient methods of parallelization is to use the message passing (MPI standard) across nodes and multithreading (OpenMP) within each node. In case of the laplc kernel, the first step is parallelization of the algorithm using MPI across a 2D array of nodes, having in mind to provide a load balancing. Fig. 1 shows data distribution across the array of nodes. We can extract two different ways of data distribution due to data dependencies. The first way is based on data dependencies across rows, while the second one is based on data dependencies across columns. To avoid communication between nodes, additional computations are required. Every data chunk is extended by additional rows or columns. Each additional row or column is the same as the row or column in the nearest neighbor chunk, respectively.

The second step is to add the parallelization with OpenMP applied for the laplc routine. This parallelization is implemented using 4 threads within one node. The main challenges for this step are:

1. avoiding reallocation of memory with every call of the routine;
2. possibility of creating OpenMP threads out of this routine;
3. providing shared access to memory by creating threads after memory allocation.

5.2 Accelerating Computations in MPDATA Kernel

The main challenge of parallelization of MPDATA on GPUs is decomposition of a 3D MPDATA mesh (N by M by L) into a 2D grid (N by M) of threads. Fig. 2 shows a 2D grid decomposition into work-groups. Work-groups are mapped into a 2D grid of size RN by CM. A work-group is a collection of work-items of size n=N/RN by m=M/CM. Each work-item computes a single element of the 2D grid (L elements of 3D MPDATA mesh). To avoid communication between work-groups, additional work-items are required. Every work-group is extended by additional rows and columns of work-items. Each additional row and column is the same as the row and column in the nearest neighbor work-group.

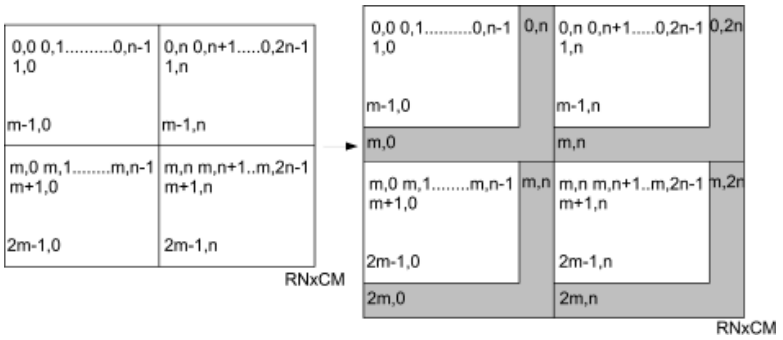

Fig. 2. MPDATA decomposition into work-groups

```
if(j<M && i<N)
      for(k=0; k<L; ++k)
        x(i, j, k)-=
          ( f1(i+1, j, k)-f1(i, j, k)+f2(i, j+1, k)-f2(i, j, k)
           +f3(i, j, k+1)-f3(i, j, k) )/h(i, j, k);
```

Fig. 3. The main part of MPDATA kernel

Fig. 3 presents the main part of the MPDATA kernel. Here $f1, f2, f3$ are computed using donnor-cell (upwind) scheme where $donnor = x - y \; for \; x > y$ and 0 otherwise.

6 Performance Results

6.1 Performance Analysis for Laplc

The parallelization of the laplc routine was based on the hybrid model (MPI and OpenMP). The simulations were performed for different data mesh sizes (from 1000x1000 to 9000x9000) using 4, 16, 25, and 100 nodes, that gives 16, 64, 100, and 400 threads, respectively. After introducing some additional computation, the algorithm does not require any communication mechanisms between nodes, so the algorithm is very scalable. The implementation provides a very good load balancing up to 400 threads using the hybrid model. For the mesh of size 9000x9000, the execution time of calculation for 1, 16, 64, 100, and 400 threads are 41.5, 4.57, 0.72, 0.34, and 0.068 seconds, respectively.

Table 1 shows performance results for the laplc routine, for the mesh of size 9000x9000. The speedups for 16, 64, 100, and 400 threads are 9.07, 57.56, 121.91, and 592.14, respectively. In this case, the efficiencies are 0.56, 0.89, 1.21, and 1.48, respectively. The key reason for the super-efficiency is dividing data among nodes into smaller chunks. It improves fitting data-elements into available caches, enhancing cache reuse.

Table 1. Performance results for laplc (mesh of size 9000 by 9000)

Number of threads	Execution time [s]	Speedup	Efficiency
1	41.45	-	-
16	4.57	9.07	0.56
64	0.72	57.56	0.89
100	0.34	121.91	1.21
400	0.07	592.14	1.48

Table 2. MPDATA performance results (mesh of size 90x90x1500)

Hardware	Kernel time [s]	Kernel speedup	Kernel+ data rec. time [s]	Kernel+ data rec. speedup	Kernel+ transfer time [s]	Kernel+ transfer speedup	Host-GPU bandwidth [GB/s]	Memory usage [MB]
CPU	0.75	1	-	-	-	-	-	514.016
NVIDIA Tesla	0.041	18.29	0.06	12.5	0.16	4.68	2.57092	584.543
ATI Radeon	0.039	19.23	0.08	9.38	0.27	2.78	1.35215	584.543

6.2 MPDATA Performance Analysis

The algorithm was tested on the NVIDIA Tesla C1060 card with Linux and ATI Radeon HD 5870 with Windows7. The achieved results are compared with a single-core implementation on the AMD Phenom(tm) II X4 955 processor with Linux.

The MPDATA implementation distinguishes three stages. The first one is sending data, the second one is the kernel responsible for the computations, and the last one is receiving data. Table 2 shows performance results of the MPDATA routine for the mesh of size 90x90x1500. The kernel speedup for NVIDIA Tesla is 18.29, while for ATI Radeon is 19.23. The data transfers have a significant impact on the resulting execution time. Time of data transfers is several times larger than computation time. The achieved bandwidth of data transfer between host and global memory is 2.57 GB/s on NVIDIA Tesla and 1.35 GB/s on ATI Radeon, where the theoretical peak bandwidth is 4 GB/s. As a result, the speedup for NVIDIA Tesla and ATI has decreased to 4.68 and 2.78, respectively.

7 Conclusions and Further Work

The implemented parts of the EULAG model do not require any communication between threads, so the algorithms are very scalable. However, this approach requires some additional calculations for each thread. Our implementation provides a very good load balancing, when using the hybrid model on BlueGene/P, and OpenCL on GPUs.

The first challenge of our work was parallelization of the laplc code, using the hybrid model with MPI across nodes and OpenMP within nodes. This solution allows for a good usage of both shared and distributed-memory system resources,

concerning memory capacity, latency, and bandwidth. Also, the hybrid model allows for the adaptation of the EULAG code to other hierarchical architectures such as clusters of multi-core processors.

The second challenge was acceleration of MPDATA using modern GPUs. The code was adopted to two types of GPUs from two leading vendors. NVIDIA TESLA was tested with Linux operating system, while ATI Radeon used Windows7. On ATI we achieved a little better performance of kernel computing, but a worse bandwidth of data transfers than on NVIDIA. In both cases, the performance on GPU is higher than on CPU.

Our parallelization of the EULAG model is still under development. One of leading approaches is using the autotuning technique which allows for algorithm self-adapting to properties of a system architecture. The final result of our work will be adaptation of the EULAG model to heterogeneous clusters with CPUs and GPUs. In this case, the key challenge will be to find an optimal load balance between GPUs and CPUs.

References

1. AMD Corporation: ATI Radeon HD 5870 Feature Summary, http://www.amd.com/
2. Dokken, T., Hagen, T.R., Hjelmervik, J.M.: An Introduction to General-Purpose Computing on Programmable Graphics Hardware. In: Geometric Modelling, Numerical Simulation, and Optimization: Applied Mathematics at SINTEF, pp. 123–161. Springer, Heidelberg (2007)
3. Eulag Research Model for Geophysical Flows, http://www.eulag.com/
4. IBM Blue Gene Team: Overview of the IBM Blue Gene/P project. IBM Journal of Research and Development 52, 199–220 (2008)
5. Khronos OpenCL Working Group: The OpenCL C++ Wrapper API, http://www.khronos.org
6. Khronos OpenCL Working Group: The OpenCL Specification, http://www.khronos.org
7. Lindholm, E., Nickolls, J., Oberman, S., Montrym, J.: NVIDIA Tesla: A Unified Graphics and Computing Architecture. IEEE Micro 28, 39–55 (2008)
8. Smolarkiewicz, P., Szmelter, J.: MPDATA: An edge-based unstructured-grid formulation. Elsevier Journal of Computational Physics 206, 624–649 (2005)
9. Sviercoski, R., Winter, C., Warrick, A.: Analytical approximation for the generalized Laplace equation with step function coefficient. J. Appl. Math. 68, 1268–1281 (2008)
10. Tsuchiyama, R., Nakamura, N., Iizuka, T., Asahara, A., Miki, S.: The OpenCL Programming Book. Fixstars Corporation (2010)