XX Fluid Mechanics Conference KKMP2012,
Gliwice, 17-20 September 2012

FAST LATTICE BOLTZMANN ALGORITH FOR HYBRID ARCHITECTURE

Łukasz Łaniewski-Wołłk[1], Wojciech Regulski[1]
[1] Institute of Aeronautics and Applied Mechanics, Warsaw, Poland
E-mail: llaniewski@meil.pw.edu.pl

Key words: CUDA, Lattice Boltzmann, R, MPI

We present a highly efficient Lattice Boltzmann (LB) code called CLB predestined to computation on multiple-GPU architecture.

In recent years computation on Graphics Processing Units (GPU) has gained broad recognition. The Compute Unified Device Architecture (CUDA) standard released by NVIDIA eased the integration of computation on GPU with standard C/C++ code architecture. We combine this standard with MPI communication protocol for fast and parallel algorithms for hybrid architectures. Such architecture consists of network-connected computational nodes equipped with CUDA compatible graphics cards.

Lattice Boltzmann (LB) schemes [1] can be efficiently addressed in multi-thread architectures like CUDA. Computations in grid nodes are completely independent and communication is a separate step. In CLB we implemented a Bhatnagar-Gross-Krook (BGK) and Multiple Relaxation Time (MRT)[2] collision model for two- and three-dimensional lattices (so-called D2Q9 and D3Q19 lattices).

Since LB is an explicit time-stepping scheme, high optimization of iteration computation is needed. Speed of CUDA implementation is mostly dependent on memory usage. Several memory levels are present on a GPU, all with very limited capacities. Three main issues corresponding to three types of memory had to be addressed:

– Each multiprocessor uses finite number of registers. Up to 512 threads are running on a single multiprocessor at one time and each thread has to use a small number of registers. This means that the code cannot use to many local variables.

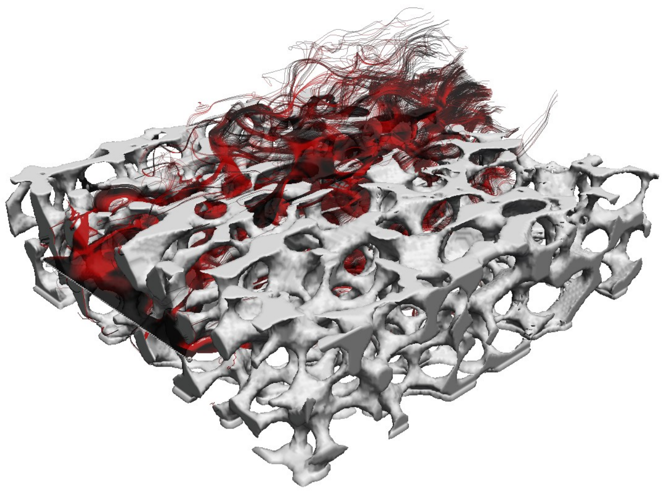

Figure 1: Streamlines in 10 ppi ceramic foam

- Shared memory tables are the only possibilities of communication between threads. This type of memory is divided into banks that can be simultaneously used by up to 32 threads. Reading of writing to the same bank from several threads causes bank conflicts, which results in lags. The code has to avoid these bank conflicts.
- The last type of memory is the global memory. It is the only memory accessible by the external CPU code. It is slowest to use, so it is of utter most importance to optimize the access to it. To make the access faster, Nvidia GPU uses coalescence. This mechanism combines several 32 bit reads/writes into a one read/write which is optimal for the memory bus (e.g. 128 bit). The coalescence is most efficient if we access consecutive elements of the memory, and even more efficient if the access is aligned (the memory address is divisible by the access length). In the code, before writing to global memory, we use shared tables to exchange data between threads. This allows the threads to write to consecutive and aligned addresses

To address all this issues in CLB we developed a special implementation methodology. Nearly 70% of the code is computer-generated. The generator makes all computations that can be made before the compilation. As conditional structures slow down CUDA parallelism, the generator also makes case-specific code. The main advantages of such approach can be seen in the communication step of the scheme. We can generate specific optimized code for each direction of communication. In each direction we have to check different conditions and use different coalescence approaches. Not only is the code more efficient, but can be regenerated for different LB models (multi-speed, multi-phase).

For code generation we use specially designed tools for R programming language. R is a open source free software based on award-winning S language (Bell Laboratories). It is broadly used in statistics, applied mathematics, and engineering. It surpasses the capabilities of S in both speed and language structure. Two main tools were used in the code generation: *Rtemplate* for text operations and source file generation, and *PolyVector* for vectorized symbol polynomial algebra. Both tools were developed by the Author.

At the last stage we use standard C preprocessing to compile CLB for specific architectures. This way the CLB can work on different GPU architectures, memory access schemes, and also can be cross-compiled for CPU only.

We present the implementation details and example usage for calculating pressure drops on ceramic foams (see figure 1). We compare the speeds on several architectures with and without GPU.

The work was carried out with the support of the "HPC Infrastructure for Grand Challenges of Science and Engineering" Project, co-financed by the European Regional Development Fund under the Innovative Economy Operational Programme.

References:

[1] Succi S., The lattice Boltzmann equation for fluid dynamics and beyond. Oxford University Press, 2001
[2] D. d'Humières, I. Ginzburg, M. Krafczyk, P.Lallemand, and L.-S. Luo, Multiple-relaxation-time lattice Boltzmann models in three-dimensions, Phil. Trans. R. Soc. A 360, pp. 437-451, 2002