New Astronomy 15 (2010) 581–589

Contents lists available at ScienceDirect

New Astronomy

journal homepage: www.elsevier.com/locate/newast

Adaptive mesh fluid simulations on GPU

Peng Wang [*], Tom Abel, Ralf Kaehler

Kavli Institute for Particle Astrophysics and Cosmology, SLAC National Accelerator Center and Stanford Physics Department, Menlo Park, CA 94025, United States

A R T I C L E   I N F O

Article history:
Received 26 May 2009
Received in revised form 8 August 2009
Accepted 8 October 2009
Available online 17 October 2009
Communicated by D.B. Sanders

Keywords:
Multi-scale methods
Finite volume methods
Hydrodynamics
Magnetohydrodynamics and plasma

A B S T R A C T

We describe an implementation of compressible inviscid fluid solvers with block-structured adaptive mesh refinement on Graphics Processing Units using NVIDIA's CUDA. We show that a class of high resolution shock capturing schemes can be mapped naturally on this architecture. Using the method of lines approach with the second order total variation diminishing Runge–Kutta time integration scheme, piecewise linear reconstruction, and a Harten–Lax–van Leer Riemann solver, we achieve an overall speedup of approximately 10 times faster execution on one graphics card as compared to a single core on the host computer. We attain this speedup in uniform grid runs as well as in problems with deep AMR hierarchies. Our framework can readily be applied to more general systems of conservation laws and extended to higher order shock capturing schemes. This is shown directly by an implementation of a magneto-hydrodynamic solver and comparing its performance to the pure hydrodynamic case. Finally, we also combined our CUDA parallel scheme with MPI to make the code run on GPU clusters. Close to ideal speedup is observed on up to four GPUs.

1. Introduction

Graphics Processing Units (GPUs) are specialized for math-intensive highly parallel computation, thus more transistors are devoted to data processing rather than data caching and flow control like in CPU. So the potential tremendous performance of general non-graphics computations on GPUs has recently motivated a lot of research activities on general-purpose GPU (GPGPU) computing (see e.g. Owens et al., 2007, for a review).

NVIDIA introduced the Tesla unified graphics and computing architecture in November 2006. The Tesla architecture is built around a scalable array of multithreaded streaming multiprocessors (SM). A SM consists of eight streaming processor (SP) cores. The Tesla SM uses a new processor architecture called single-instruction, multiple-thread (SIMT). The SIMT unit creates, manages and executes up to 768 concurrent threads in hardware with zero scheduling overhead. The SM also implements barrier synchronization intrinsic with a single instruction. The fast barrier synchronization, together with lightweight thread creation and zero-overhead thread scheduling support very fine-grained parallelism allowing thousands and even millions of threads to be invoked in kernel calls to achieve highly scalable parallel programming.

High performance supercomputing has been important in modern astrophysical research since it became available. Simulations allow astronomers to perform "experiments" on astronomical objects, collide stars, galaxies, or model the entire visible Universe; all situations are clearly impossible to recreate in a terrestrial laboratory. Studying the formation of stars, black holes and galaxies in the Universe is particularly challenging computationally. Their formation involves the nonlinear interplay of a range of physical processes including gravity, turbulence, magnetic field, shocks, radiation, chemistry, etc. Those questions motivated the astrophysical community to develop robust and efficient fluid codes with all the relevant physics.

Studies involving astrophysical fluid dynamics in general are benefitting tremendously from using spatial and temporal adaptive mesh refinement (AMR). This is especially so in the studies of structure formation. For example, the radius of a star is eight orders of magnitude smaller than the size of a molecular cloud. A uniform grid code is hopeless. On the other hand, the AMR technique has been demonstrated to work well in resolving the large dynamical range involved in those problems (e.g. Abel et al., 2002; Wang and Abel, 2009).

The mapping of computational fluid algorithms to GPU however is still at an early stage of development. Harris et al. (2003) performed cloud simulations using Stam's method (Stam, 1999). This method is also used by Liu et al. (2004) for 3D flow calculations. Using finite difference methods, Brandvik and Pullan (2008) solved uniform grid 3D Euler equations, Elsen et al. (2008) solved 3D Euler equations on a multi-block meshes and Zink (2008) solved Einstein's equation with uniform grid. As far as we are aware of, this work is the first on mapping an adaptive mesh finite volume solver to GPU.

[*] Corresponding author. Tel.: +1 650 926 4699; fax: +1 650 926 5566.
E-mail address: pengwang234@gmail.com (P. Wang).

1384-1076/$ - see front matter © 2009 Published by Elsevier B.V.
doi:10.1016/j.newast.2009.10.002

2. CUDA

In 2007, NVIDIA released CUDA for GPU computing as a language extension to C (see NVIDIA, 2009 for a comprehensive introduction to CUDA programming). CUDA makes GPU computing application development much easier and more efficient than earlier attempts to GPGPU using various shading languages which need to translate the computation to a graphics language.

CUDA's parallelization model is based on abstraction of the GeForce 8-series hardware. It allows programmer to define kernels which can be executed in parallel by many threads on GPU. Threads are organized into 1D, 2D or 3D thread blocks while blocks are organized into 1D or 2D grids. Each thread can access its thread and block indices by two built-in variables threadIdx and blockIdx. Fig. 1 shows how this CUDA threads/blocks/grids hierarchy map to the GPU hardware. As discussed above, a 8-series GPU has an array of SMs. Each SM is composed of eight SP. In CUDA, each thread is executed on a single SP while each block is executed on a single SM. The whole grid of blocks is mapped to the GPU. At current generation of architecture, only a single kernel can run on the GPU at a given time. Because a block is mapped to a single SM, synchronization is possible within a block. Currently, global synchronization between blocks is impossible.

The SM's SIMT unit creates, manages, schedules and executes threads in groups of 32 parallel threads called warps. The threads in a warp always execute a common instruction at a time, but different warps execute independently. As a result, different warps can execute on different branches. This is an enormous improvement for branching code compared to previous-generation GPUs as the 32-thread warps are much narrower than the SIMD (single-instruction multiple-data) width of prior GPUs. However, if threads of a warp diverge via a conditional branch, different execution path have to be serialized, increaing the total number of instructions executed for this warp. This means that if a code branches within a warp, then the total number of instructions to get the job done will be increased and as a result the execution time will be increased. So branching inside a warp should still be minimized to achieve good efficiency. However, there is no penalty if different warps branch to different pathes. Thus, the ideal situation is to organize the branching conditions such that each warp follows the same path.

CUDA exposes the hardware memory hierarchy by allowing threads to access data from multiple memory spaces. All threads have access to the same global memory. Each thread block has a shared memory visible to all threads of the block and with the same lifetime as the block. Each thread has a private local memory and a set of registers. There are also two additional read-only memories accessible by all threads: the constant and texture memory.

The shared memory is much smaller than global memory, typically 16 kB, but it is on-chip so it has very high register-level

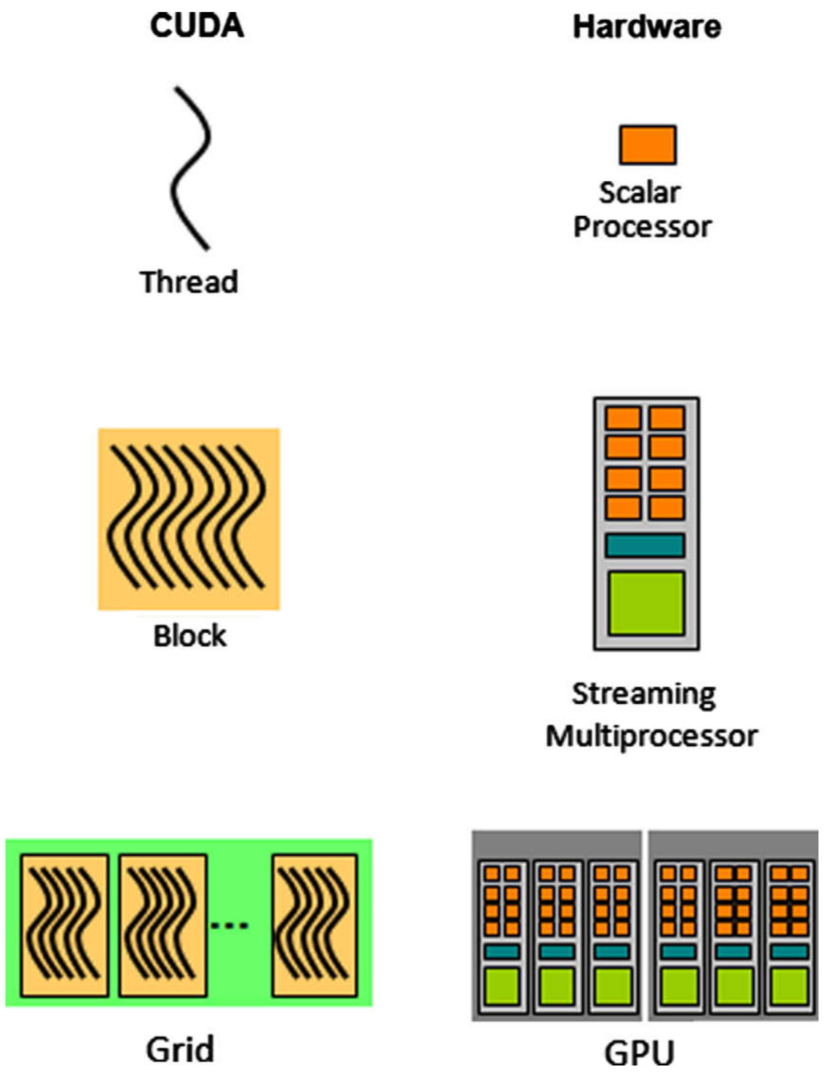

Fig. 1. The mapping of CUDA thread hierarchy to GPU hardware.