Journal of Computational Physics 235 (2013) 241–257

Contents lists available at SciVerse ScienceDirect

Journal of Computational Physics

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

Direct numerical simulation of turbulence using GPU accelerated supercomputers

Ali Khajeh-Saeed, J. Blair Perot [*]

Theoretical and Computational Fluid Dynamics Laboratory, University of Massachusetts, Amherst, MA 01003, USA

A R T I C L E   I N F O

Article history:
Received 25 November 2011
Received in revised form 28 September 2012
Accepted 26 October 2012
Available online 22 November 2012

Keywords:
Direct numerical simulation
Multi-GPU
Turbulence

A B S T R A C T

Direct numerical simulations of turbulence are optimized for up to 192 graphics processors. The results from two large GPU clusters are compared to the performance of corresponding CPU clusters. A number of important algorithm changes are necessary to access the full computational power of graphics processors and these adaptations are discussed. It is shown that the handling of subdomain communication becomes even more critical when using GPU based supercomputers. The potential for overlap of MPI communication with GPU computation is analyzed and then optimized. Detailed timings reveal that the internal calculations are now so efficient that the operations related to MPI communication are the primary scaling bottleneck at all but the very largest problem sizes that can fit on the hardware. This work gives a glimpse of the CFD performance issues will dominate many hardware platform in the near future.

1. Introduction

The direct numerical simulation (DNS) of turbulence is a computationally intensive scientific problem that can benefit significantly from improvements in computational hardware performance. Graphics processors (GPUs) are special purpose hardware that are designed for interactive gaming, not for large scientific computations. Nevertheless, GPUs are reasonably well suited for many tasks that appear in scientific computing, and it is of some interest to examine their potential impact on computationally intensive algorithms such as the simulation of turbulent flow.

The fundamental advantage of GPUs over classic CPUs comes from the entirely different design of the memory subsystem. GPUs are designed to display millions of pixels at a rate of at least 60 times per second. The hardware is therefore directly designed to stream large sets of data in, do a few calculations on the data, and then stream the data out (often to the screen). To do this, GPUs use fast memory (DDR5) whereas most CPUs still use DDR3. They also use many duplicate channels to memory (16 at a time) rather than the typical two channels for a quad core CPU. And finally these processors do not use cache approaches to try and hide the memory latency as CPUs do. Caches are designed to speed up email, operating system tasks, and word processing. But caches are not typically useful for enhancing game performance or scientific computations. The long memory streams encountered in graphics applications (or PDE solvers) defeat caches.

The memory streams in a direct numerical solution of turbulence, such as a pressure or velocity field, are on the order of 1 billion bytes each. We would like to efficiently stream this data in, do a few computations and return the same field but at the next time level. Because processor speeds have been increasing over the last decade but memory speeds have not, all CFD simulations (and in fact almost all PDE solution techniques) are memory bound. This means, that on modern computers the computations are not important to the performance of the algorithm. The critical factor is the ability to read data in, and

[*] Corresponding author.
E-mail address: perot@ecs.umass.edu (J. Blair Perot).

0021-9991/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jcp.2012.10.050

write results out. GPU memory subsystems do this an order of magnitude more quickly than CPU memory subsystems based on caches. Perhaps not surprisingly, the GPU memory subsystem functions (and can be optimized by the user) remarkably similarly to the memory subsystem of the original Cray supercomputer vector processors.

Early examples of CFD calculations on GPUs are discussed in [1–3]. However, the GPU architectures have been evolving rapidly since that time in many different areas [4–7]. More recent CFD implementations that involve the CUDA programming paradigm are discussed in Elsen et al. [8] and Corrigan et al. [9]. Micikevicius [10] applied 3D finite difference computation using CUDA on 4 GPUs and achieved linear speedup for up to four GPUs. Rossinelli et al. [11] describe a 2D simulation using a vortex particle method on the GPU that achieves a speedup of 25. Jacobsen et al. [12] discretized the Navier–Stokes equations on a uniform Cartesian staggered grid with a second-order central difference scheme and achieved a 11$\times$ speedup with one GPU compared with two quad-core CPUs on the same node. They also achieved a 130$\times$ speedup for 128 GPUs compared to two quad-core CPUs (8 cores). In this work, a speedup of 20$\times$ is found for 192 GPUs vs. 192 CPU cores. The large reduction in the calculation time when using GPUs makes the MPI communication time, and methods for hiding this communication time, paramount in the following discussion.

2. Implementation

2.1. Hardware

Results for two different NSF GPU supercomputers are discussed in this work. Forge has replaced Lincoln at NCSA. It contains 288 Tesla M2070 NVIDIA Fermi GPUs that each have 6 GB DDR5 memory. Forge's 36 servers each hold two AMD Opteron Magny-Cours 6136 with 2.4 GHz eight-core CPUs (16 CPU cores per node) with 3 GB of RAM per core. Each server is also connected to 8 Tesla processors via PCI-e Gen2 X16 slots. The Forge results were compiled using Red Hat Enterprise Linux 6 (Linux 2.6.32) and the GNU compiler [13]. Another GPU cluster, Keeneland Initial Delivery (KID) resides at the National Institute for Computational Science (NICS) and will become an NSF XSEDE resource in late 2012. KID has 360 Tesla M2070 NVIDIA Fermi GPUs that each have 6 GB DDR5 memory. KID's 120 servers each hold two 6-core Intel Xeon (Westmere-EP) 2.93 GHz (11.72 GFlops) and 2 GB of DDR3 RAM per CPU core. Each server is connected to 3 Tesla processors via PCI-e Gen2 X16 slots. Keeneland nodes are connected by an x8 InfiniBand QDR (single rail) network [14]. In both Forge and Keeneland, error-correcting code (ECC) is on and the memory bandwidth of the GPU's is therefore reduced by roughly 12%.

The GPU performance of these machines will be compared with calculations using the CPUs on those machines. In addition, we will also perform tests on Orion, which is an in-house GPU machine containing 4 NVIDIA 295 cards (8 GPUs). On Orion, each GPU has 0.9 GB of memory and a theoretical bandwidth of 112 GBs. The connection between the GPUs on Orion uses an 8$\times$ PCI-e Gen2 connection (4 GB/s) and for simplicity, the communication still uses the MPI protocol even though this is a shared CPU memory machine. Also we replaced the first and second GPUs with GTX 480 and Tesla C2070 cards in order to run some cases with these newer GPUs.

The low-level CFD algorithm structure is dictated by two key features of the GPU hardware. First, the GPUs read/write memory is an order of magnitude faster when the memory is read linearly with stride 1. Random reads/writes or long strides are comparatively slow on a GPU. In addition, each multi-processor on the GPU has some very fast on-chip memory (shared memory) which serves essentially as an addressable program-supervised cache. CFD, like most three-dimensional PDE solution applications, requires considerable random memory accesses (even when using structured meshes) for sparse matrix–vector multiplies. Roughly 90% of these slow random memory accesses can be eliminated by: (1) linearly reading large chunks of data into the shared-memory space, which is fast for all accesses, (2) operating on the data in the shared-memory, and then (3) writing the processed data back to the main GPU memory (global memory) linearly. This optimization is the key to obtaining the roughly 20x speedup of the GPU over a CPU.

2.2. Software

The solution method uses a three-step, low-storage Runge–Kutta scheme [15] for time advancement that is second-order accurate in time. This scheme is stable for eigenvalues on the imaginary axis less than 2, which implies $CFL < 2$ for advective stability. The simulations always use a maximum $CFL < 1$. The diffusive terms are advanced with the trapezoidal method for each Runge–Kutta substep, and the pressure is solved using a classical fully-discrete fractional step method [16], although an exact fractional step method [17] is also possible. The solution of the elliptic pressure Poisson equation dominates the solution time (90%), so further details of this portion of the code are presented in Section 3, and the timing results in Section 4 also focus on this part of the algorithm. The Poisson solution in this work uses a preconditioned conjugate gradient method. This Poisson solver allows arbitrary boundary conditions (and physically relevant flow initialization) to be computed. FFT methods for the solution of the pressure, which are common in turbulence simulations, are restricted to fully periodic domains or extremely simple wall geometries.

For the spatial discretization, a second order Cartesian staggered-mesh scheme is used. This not only conserves mass and momentum to machine precision, but because it is a type of discrete calculus method [18] it also conserves vorticity (or circulation) and kinetic energy in the absence of viscosity. As a result, there is no artificial viscosity/diffusion in this method except that induced by the time-stepping scheme [19]. In addition, the staggered mesh discretization is free from pressure