A Fast Double Precision CFD Code using CUDA

Jonathan M. Cohen *, M. Jeroen Molemaker**

*NVIDIA Corporation, Santa Clara, CA 95050, USA
(e-mail: jocohen@nvidia.com)
**IGPP UCLA, Los Angeles, CA 90095, USA
(e-mail: nmolem@atmos.ucla.edu)

Abstract: We describe a second-order double precision finite volume Boussinesq code implemented using the CUDA platform. We perform detailed validation of the code on a variety of Rayleigh-Benard convection problems and show second order convergence. We obtain matching results with a Fortran code running on a high-end eight-core CPU. The CUDA-accelerated code achieves approximately an eight-time speedup for versus the Fortran code on identical problems. As a result, we are able to run a simulation with a grid of size $384^2 \times 192$ at 1.6 seconds per time step on a machine with a single GPU.

Keywords: CUDA, GPU Computing, Multicore, Rayleigh-Bénard convection.

1. INTRODUCTION

We investigate the use of massively multicore GPUs to accelerate a globally second-order accurate double precision CFD code. While supercomputing clusters allow for large problems to be solved in parallel, the trend towards increased parallelism on a single chip allows researchers to solve larger problems on a single machine than has previously been possible. Our code is implemented using the CUDA platform [1] and is designed to run on the NVIDIA GT200 architecture for GPU computing [3]. The NVIDIA Quadro FX5800 card consists of a single GT200 GPU with 240 cores and 4GB of memory. GT200 supports IEEE-compliant double precision math with peak throughput of 87 GFLOPS/sec. Our code is optimized to take advantage of the parallel GT200 architecture and is approximately 8 times faster than a comparable multithreaded code running on an 8-core dual-socket Intel Xeon E5420 at 2.5GHz. See Table 2 for a summary of relative performance.

2. NUMERICAL METHOD

We solve the incompressible Navier-Stokes equations using the Boussinesq approximation:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u}\cdot\nabla)\mathbf{u} + \nu\nabla^2\mathbf{u} - \nabla p + \alpha g T \mathbf{z}$$
$$\frac{\partial T}{\partial t} = -(\mathbf{u}\cdot\nabla)T + \kappa\nabla^2 T$$
$$\nabla\cdot\mathbf{u} = 0$$

where $\mathbf{u} = (u, v, w)$ is the fluid velocity field, T is the fluid temperature, p is the fluid pressure field, α is the coefficient of thermal expansion, g is the magnitude of gravity, ν is kinematic viscosity, and κ is thermal diffusivity.

We solve these equations on a staggered regular grid (Arakawa C-grid) using a second order finite volume discretization. The advection terms are discretized using centered differencing of the flux values, resulting in a discretely conservative second-order advection scheme. All other spatial terms are discretized with second-order centered differencing. We use a second-order Adams-Bashford method which we prime with a single forward-Euler step at the start of the numerical integration. Pressure is treated via a projection method, where pressure is calculated instantaneously at the end of the time step to enforce $\nabla\cdot\mathbf{u} = 0$. This requires solving a Poisson equation for pressure, for which we use a multigrid method.

While we have chosen simple discretizations for ease of implementation and validation, our code is designed to support a wide variety of higher-order discretizations and stencils. Our optimization strategies will apply to more complex higher order numerical methods as well.

3. IMPLEMENTATION

We have explored a number of GPU-specific optimization strategies. GT200 is designed to run tens of thousands of threads simultaneously, using the large amount of parallelism to hide latencies for off-chip memory reads and writes. GT200 has peak off-chip memory bandwidth of approximately 102 GB/sec, which corresponds to an optimal balance of just over 6 math operations per double precision value loaded from memory. Because most of our inner loops perform a small amount of math, performance of our code is mainly limited by memory bandwidth. Therefore we focused on optimizing our memory access patterns to take advantage of GT200's streaming memory architecture.

3.1 Access Pattern Optimization

On GT200, threads are grouped into batches of 32 called *warps* that execute in lockstep SIMD fashion. If threads in a warp read from the same cache line in the same cycle, these reads are batched into a single operation via a process known as *memory coalescing*. Coalescing operates at half-warp granularity, so uncoalesced loads and stores waste $15/16^{\text{ths}}$ of available memory bandwidth. Therefore the most important optimization for memory-bound applications is to arrange work so that threads in the same warp will access sequential memory locations at the same time.

GT200 has two small on-chip caches: a read-only L1 cache called the *texture cache*, and a read/write software managed cache called *shared memory*. With thousand of simultaneous active threads, on-chip caches are beneficial only if threads scheduled to the same processor access the same cache lines at the same time. Therefore optimizing for cache performance is very similar to optimizing for memory coalescing. The texture cache can be thought of as a "bandwidth aggregator" because it is designed to aggregate memory requests over several cycles so that coalescing will be more efficient. For some routines, especially finite difference stencils, we found the texture cache to yield a performance improvement of up to 1.4x. In many cases, however, it produced no benefit over coalescing alone. Because the shared memory cache is software managed, we found that the cost of the logic required to use it effectively typically was not offset by the bandwidth savings.

3.2 Congruent Padding

Another optimization we found to be important was a concept we term "congruent padding." By congruent, we mean that for all pairs of indices (i, j, k), the offset in bytes between the memory location of element (0, 0, 0) and element (i, j, k) should be the same for all grids. Figure 1 demonstrates congruent padding in two dimensions. Because GT200 runs most efficiently with a large numbers of threads, we assign one thread per computational cell (i, j, k). Since all threads translate from grid indices to memory locations in parallel, index translations must be recalculated at every grid cell. Congruent padding amortizes the cost of index translation over several grids by calculating the offset from location (0, 0, 0) to (i, j, k) once per thread, and then adding this offset to the base pointer for each grid. In our code, this optimization reduces the number of instructions of a typical GPU routine by 10-15% and reduces the register usage by 0-10%. Minimizing the number of per-thread registers allows for more threads to be active at once. Having more active threads covers memory latencies more effectively, which improves throughput [3].

(-1,2) 18	**(0,2)** 19	**(1,2)** 20	**(2,2)** 21	22	23
(-1,1) 12	**(0,1)** 13	**(1,1)** 14	**(2,1)** 15	16	17
(-1,0) 6	**(0,0)** 7	**(1,0)** 8	**(2,0)** 9	10	11
(-1,-1) 0	**(0,-1)** 1	**(1,-1)** 2	**(2,-1)** 3	4	5

(a) A 2x2 grid with 1 row of ghost cells on all sides.

18	**(0,2)** 19	**(1,2)** 20	**(2,2)** 21	**(3,2)** 22	23
12	**(0,1)** 13	**(1,1)** 14	**(2,1)** 15	**(3,1)** 16	17
6	**(0,0)** 7	**(1,0)** 8	**(2,0)** 9	**(3,0)** 10	11
0	1	2	3	4	5

(b) A 4x3 grid with no ghost cells.

Fig 1: An example of congruent padding in 2 dimensions. All grids have the same physical layout in memory, even though they may have different logical dimensions. Computational cells are white, ghost cells are light gray, and unused padding is dark gray.

Resolution	Full Slip		No Slip	
	Value	Difference	Value	Difference
$16 \times 8 \times 16$	659.69	-	1674.29	-
$32 \times 16 \times 32$	658.05	1.64	1699.25	24.96
$64 \times 32 \times 64$	657.65	0.40	1705.59	6.34
$128 \times 64 \times 128$	657.54	0.11	1707.22	1.63
∞	657.51	-	1707.76	-

Table 1: Calculated critical Rayleigh values for full-slip (aspect ratio $\sqrt{2} : .5 : 1$) and no-slip (aspect ratio $\pi : .5 : 3.11$) boundaries at different resolutions. The columns labeled Difference show the differences in values between subsequent grid resolutions. The reduction of this error by a factor of 4 for each doubling of resolution shows the globally second order convergence character of the numerical discretizations. The last row shows the Richardson extrapolated values, which match theory.

4. VALIDATION

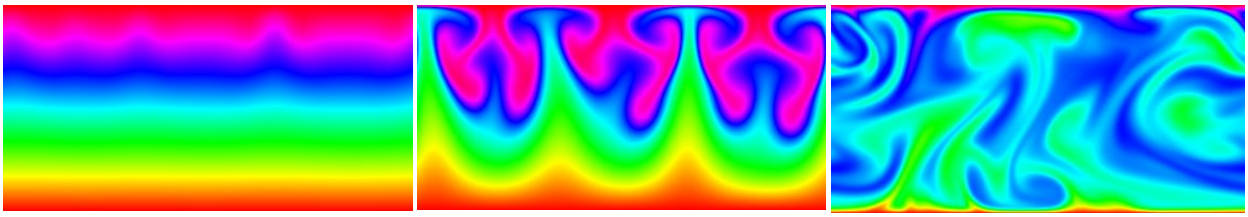
To demonstrate the potential of our GPU based code for scientific applications we have validated the code for a range of problems. We compared our results with an existing CPU based code written in Fortran [5] as well as against published analytical, numerical, and experimental results. Since our code implements the Boussinesq equations we choose to examine whether it can reproduce known solutions to different Rayleigh-Bénard convection problems in which a constant temperature difference ΔT is maintained between the top and bottom boundaries of the domain. The most basic result for Rayleigh-Bénard convection is the critical value of the dimensionless Rayleigh number $Ra = g\alpha\Delta T/\kappa\nu$. Below the critical value Ra_c the solution is motionless and heat flux between top and bottom is purely diffusive. When $Ra > Ra_c$ the diffusive solution becomes unstable to perturbations of arbitrarily small amplitude and a solution with non-trivial flow and enhanced vertical heat transport ensues.

We estimated Ra_c in our codes by calculating positive and negative growth rates of **u** for small perturbations around Ra_c and extrapolating to find the value of Ra for which the growth rate would be zero. Our GPU and CPU codes use identical numerical methods and therefore have matching values to several decimal places. Table 1 shows calculated Ra_c values from our codes. The third and fifth columns show the differences in Ra_c obtained for subsequent resolutions, highlighting the globally second order convergence rate of the numerical implementation. For the 2D problem with the aspect ratios chosen, analytical values are known [7] (we treat this as a 3D problem by choosing a

smaller aspect ratio and periodic boundaries in the y dimension). Using Richardson extrapolation, we obtain a value of Ra_c = 657.51 for the full-slip case, and Ra_c = 1707.76 for the no-slip case, both of which match the analytical results.

To test a fully 3D problem, we also studied the onset of convection in a cubic box with no-slip conditions on all sides and Dirichlet conditions for T on the side boundaries. We find a critical Rayleigh number Ra_c = 6755 for this case, matching the published experimental [2] and numerical [4, 6] values. To verify the nonlinear advection terms in the equations, we calculated the solution for a supercritical value of $Ra = 4.4 \times 10^4$. We then calculated the Nusselt number $Nu = \overline{(wT + \kappa T_z)}/(\kappa \Delta T)$. The Nusselt number is the ratio of vertical heat transport to diffusive heat transport across a 2D interface of a motionless solution. Nu also depends on the Prandtl number, $Pr = \nu/\kappa$. For $Ra = 4.4 \times 10^4$ and $Pr = 0.71$, Nu computed at the upper and lower boundaries is 2.05 (using both CPU and GPU codes) and exhibits global second order convergence when computed at increasing resolutions. This matches published results [6].

5. COMPARATIVE PERFORMANCE



(a) Perturbations appear in the stratifications.

(b) Instabilities form.

(c) Temperature mixes after the onset of turbulence.

Fig 2: False color plot of T at the $y = 0$ plane for a $384^2 \times 192$ resolution simulation with $Ra = 10^7$.

To generate a timing comparison, we ran an unsteady Rayleigh-Bénard convection problem on our GPU and CPU codes with $Ra = 10^7$ and $Pr = .71$. The simulation domain was set to [-1, 1] × [-1, 1] × [-.5, .5], with periodic boundary conditions in x and y, and no-slip boundaries in z. As shown in Figure 2, the flow starts out motionless until instabilities form, and then transitions to turbulence. To accelerate convergence of the multigrid solver for pressure, we reuse the solution from the previous time step as an initial guess. Consequently, the number of v-cycles required for convergence increases as the flow becomes less steady. In order to characterize our performance fairly, we only count the average time per step once the number of v-cycles per step has stabilized. Because of the different performance characteristics of the GPU and the CPU, we have chosen slightly different multigrid relaxation schemes for the two codes. The GPU code, which uses a red-black Gauss-Seidel point relaxer, requires 1 full multigrid step followed by 7 v-cycles. The CPU code uses a red-black line relaxer and requires 1 full multigrid step followed by 13 v-cycles.

Table 2 shows the relative timing of the two codes for problem sizes up to $384^2 \times 192$, which is the largest that can fit on a single GPU with 4GB of memory. GPU times are for a single Quadro FX5800 running on a Core2-Duo E8500 at 3.17GHz. CPU times are for an 8-core dual socket Xeon E5420 at 2.5GHz, and the CPU code is multithreaded using OpenMP and MPI to take advantage of all 8 cores. These hardware configurations were chosen because they are roughly equivalent in terms of cost.

Our GPU timing results are significant for two reasons. First, at the high end of simulation size, $384^2 \times 192$, we have achieved performance on a single node comparable to that often achieved on a small cluster. Second, turbulent features can be observed at resolutions as low as $128^2 \times 64$. At this resolution, our code can calculate up 12 time steps per second, which enables interactive turbulent simulations on a commodity PC with a single GPU.

Resolution	GPU CUDA Code			CPU Fortan Code			**GPU Speedup**
	ms/Step	ms/Step/Node	Scaling	ms/Step	ms/Step/Node	Scaling	
$64^2 \times 32$	24	18.3e-5	-	47	37.0e-5	-	**2.0x**
$128^2 \times 64$	79	7.5e-5	0.41x	327	31.2e-5	0.84x	**5.3x**
$256^2 \times 128$	498	5.9e-5	0.79x	4070	48.5e-5	1.55x	**8.2x**
$384^2 \times 192$	1616	5.7e-5	0.97x	13670	48.3e-5	1.00x	**8.5x**

Table 2: Relative performance for the $Ra = 10^7$ problem. Time per step is calculated ignoring the initial run-up to turbulence since multigrid converges faster during this period. The column labelled Scaling indicates the change in time per step per computational node from one resolution to the next. Linear scaling in the number of computational nodes would therefore be 1.0x.

5. FUTURE WORK

Our results demonstrate that GPU-based codes can be a powerful tool for real scientific applications. We have demonstrated second order convergence in double precision on buoyancy-driven turbulence problems using a GPU. Using a single workstation that is equipped with a GPU, our code can integrate a non-trivial flow at moderately high resolutions up to 8 times faster than a high-end 8-core CPU. We intend to extend our work in several ways. First, we will implement higher-order methods in both space and time. Second, we are interested in numerical ocean and atmospheric models such as [8] that use logically regular grids, but are geometrically irregular. Third, we will explore multiple GPU configurations such as [9]. A single motherboard may have several PCI-express buses, each of which can connect to one or more GPUs. In addition to improving performance for large computing clusters, this has the potential to dramatically increase the resolution that people without access to clusters can achieve.

REFERENCES

1. NVIDIA Corporation (2008). *CUDA programming guide, version 2.0.*
2. Leong, W.H., Hollands, K.G.T., and Brunger, A.P. (1998). On a physically realizable benchmark problem in internal natural convection. *Int. J. Heat Mass Transfer* 41, 3817-3828.
3. Lindholm, E., Nickolls, J., Olberman, S., and Montrym, J. (2008). NVIDIA Tesla: A unified graphics and computing architecture. *IEEE Micro* 28(2), 39-55.
4. Mizushima, J. And Matsuda, O. (1997). Onset of 3d thermal convection in a cubic cavity. *J. Phys Soc. Japan* 66, 2237-2341.
5. Molemaker, M.J., McWilliams, J.C., and Capet, X. (2009). Balanced and unbalanced routes to dissipation in equilibrated Eady flow. *J. Fluid Mech,* In press.
6. Puigjaner, D., Herrero, J., Simo, C., and Giralt, F. (2008). Bifurcation analysis of steady Rayleigh-Bénard convection in a cubical cavity with conducting sidewalls, *J. Fluid Mech* 598, 393-427.
7. Reid, W.H. and Harris, D.L. (1958). Some further results on the Bénard problem, *Phys. Fluids* 1, 102-110.
8. Shchepetkin, A.F. and McWilliams, J.C. (2005). The regional oceanic modeling system (ROMS): a split-explicit, free-surface, topography-following-coordinate oceanic model, *Ocean Modelling* 9, 347-404.
9. Thibault, J.C. and Senocak, I (2009). CUDA implementation of a Navier-Stokes solver on multi-GPU desktop platforms for incompressible flows, *47th AIAA Aerospace Sciences Meeting*, Orlando, Florida, paper no: AIAA 2009-758.