1-14-2009

Rapid-Response Urban CFD Simulations Using a GPU Computing Paradigm on Desktop Supercomputers

Inanc Senocak
Boise State University

Julien C. Thibault
Boise State University

Matthew Caylor
Boise State University

J19.2 **Rapid-response Urban CFD Simulations using a GPU Computing Paradigm on Desktop Supercomputers**

Inanc Senocak*, Julien Thibault and Matthew Caylor
Boise State University, Boise, ID

1. INTRODUCTION

In the event of chemical or biological (CB) agent attacks or accidents, first-responders need hazard prediction data to launch effective emergency response action. Accurate and timely knowledge of the wind fields in urban areas is critically important to identify and project the extent of CB agent dispersion to determine the hazard-zone. In their 2008 report (GAO-08-180), U.S. Government Accountability Office has reported that first responders are limited in their ability to detect and model hazardous releases in urban environments. The current set of modeling tools for contaminant dispersion in urban environments rely on empirical assumptions with diagnostic equations (Wang et al. 2003, Williams et al. 2004). The main advantage of these models is their relatively fast turn-around times, although their predictive capabilities can be limited. As part of the Joint Effects Model (JEM), funded by the Department of Defense, urban transport and dispersion models have been evaluated for their rapid-response capabilities. As discussed in Heagy et al. (2007), majority of the urban transport and dispersion models considered in the evaluation study fell short of satisfying the JEM key performance parameter of maximum 10-minutes run-time on a desktop computer, and the models that were able to satisfy the performance parameter were employed at low resolutions.

CFD models have been applied to urban environments (Hanna et al. 2006) Thus far, first-principles based computational fluid dynamics (CFD) models have been considered impractical for deployment in emergency response operations because of their slow computational turn around times. Loosely speaking, a general purpose CFD code may not give the fastest computational turn-around time because of the software complexity that arises from implementing various methods and models to address a wide range of problems in different disciplines. CFD codes can benefit from a specific implementation for urban environments (Burrows et al. 2004). Recently, it has been shown that urban CFD simulations can be accelerated significantly by a careful implementation of numerical methods on Cartesian grids. (Gowardhan et al. 2007; Gowardhan 2008). It should be noted a CFD simulation does not necessarily translate to better predictions. Contaminant dispersion predictions depend highly on initial conditions of wind speed and direction. Equally important, the choice of turbulence model and proper mesh resolutions near solid boundaries can have an impact on the predictions.

There is a strong need to develop rapid-response CFD-based contaminant transport and dispersion models for first responders. Given the recent breakthroughs in both hardware and software for high performance computing (Owens et al. 2008), it is believed that with a careful selection of numerical methods and parallel computing strategies, a CFD solution of wind fields and CB agent dispersion within complex urban environments can be delivered within a time frame that benefits the first responder.

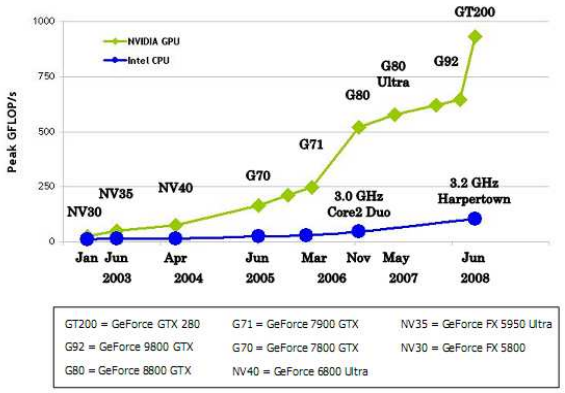


Figure 1: Evolution of floating-point performance for Intel CPUs and NVIDIA GPUs (courtesy of NVIDIA).

Graphics Processing Units (GPU) that are traditionally designed for graphics rendering have emerged as massively-parallel "co-processors" to the Central Processing Unit (CPU) (Owens et al. 2008). Figure 1 depicts the growing gap in peak performance, measured in floating point operations per second (FLOPS) between GPU and CPU over the last five years. Currently, NVIDIA GPUs outperform Intel CPUs on floating point performance and memory bandwidth, both by a factor of roughly ten (NVIDIA, 2008a). Small-footprint desktop supercomputers with hundreds of stream processors that can deliver teraflops theoretical peak performance at the price of conventional workstations have been realized. Figure 2 presents two desktop supercomputers with different GPU and CPU configurations.

*Corresponding author address: Boise State University
Department of Mechanical & Biomedical Engineering,
Boise, ID 83725-2075, e-mail: senocak@boisestate.edu

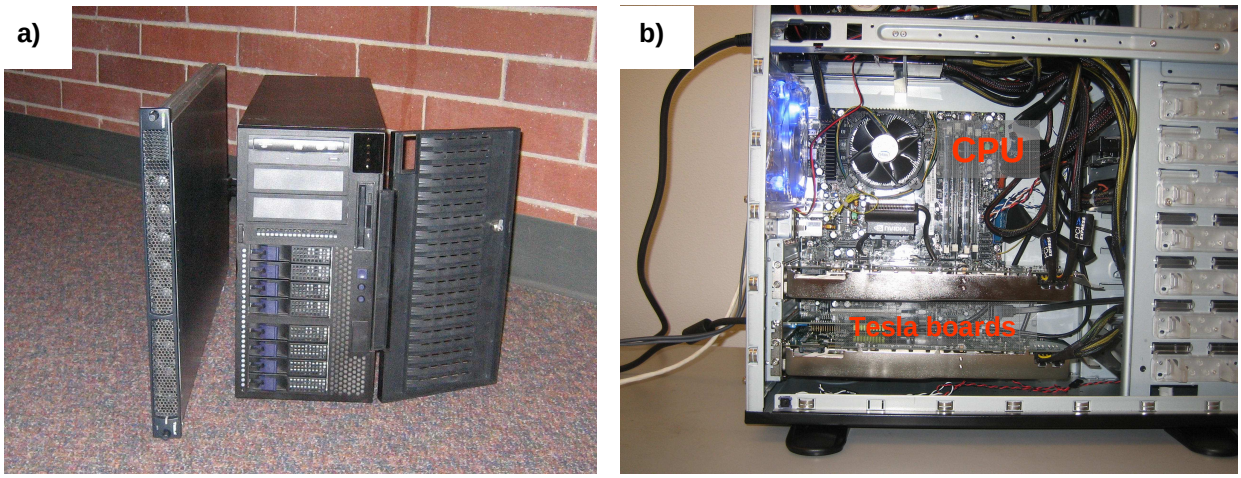


Figure 2: Desktop supercomputers for GPU computing research at Boise State University **a)** Quad-GPU platform: NVIDIA S870 Tesla server connected to a high performance workstation with 8 dual-core AMD Opteron (8216) 2.4MHz. Total of 512 streaming cores with 6 GB device (GPU) memory. **b)** Dual-GPU platform: 2 NVIDIA C870 Tesla boards connected to a Intel Core 2 Duo (E8400) 3.0 GHz. Total of 256 cores with 3GB device (GPU) memory.

Until recently, using the GPUs for general-purpose computation was a complicated exercise. A good knowledge of graphics programming was required, because GPU's old fixed-function pipeline did not allow complex operations (Owens et al. 2007). GPUs have evolved into a programmable engine, supported by new programming models trying to find the right balance between low access to the hardware and high-level programmability (Owens et al. 2008). Brook programming model, released in 2004 by Stanford University, offered one of the first development platforms for general purpose GPU (GPGPU) programming (Buck et al. 2004). NVIDIA recently released a more advanced programming model for its own line of GPUs: Compute Unified Device Architecture (CUDA). With CUDA, NVIDIA offers a common architecture and programming model for its own line of GPUs. The C-based application programming interface (API) of CUDA enables data-parallelism through the use of shared memory, but also computation parallelism thanks to the introduction of the thread and grid concepts (NVIDIA 2008a).

Prior to the introduction of the CUDA and Brook programming models, several Navier-Stokes solvers have been implemented for the GPU. Harris (2003) implemented a 3D solver to create a physically-based cloud simulation using the Cg programming language from NVIDIA. Due to its relative potential for easy parallelization, the Lattice-Boltzman method (LBM) has also been implemented in different studies addressing complex geometries. In Li et al. (2005), GPU implementation of LBM resulted in speedup of 15× relative to the CPU implementation. Fan et al. (2004) implemented the LBM on a GPU cluster to calculate winds and contaminant dispersion in urban areas. A speedup of 4.6× relative to a CPU cluster was achieved in their study, which demonstrates that GPU clusters can serve as an efficient platform for scientific computing. In Willemsen et al. (2007) and Pardyjak et al. (2007) a simple Lagrangian dispersion model with prescribed wind fields was implemented on the GPU. A substantial speedup relative to the existing CPU implementation was demonstrated.

The recent literature attests to the compute-potential of GPU computing with new programming models. Numerous studies have adopted the CUDA programming model computational problems in engineering and sciences at large (NVIDIA 2008b).

In the computational fluid dynamics (CFD) field, Tolke and Krafczyk (2008) implemented a 3D Lattice-Boltzman method for flow through a generic porous medium. They obtained a gain of up to two orders of magnitude with respect to the computational of an Intel Xeon 3.4GHz. Brandvik and Pullan (2008) mapped 2D and 3D Euler solvers to the GPU using BrookGPU and CUDA programming models. For the CUDA version of the 3D Euler solver, their computations on NVIDIA 8800GTX showed a speedup of 16× over the CPU, whereas the BrookGPU implementation of the 3D Euler solver showed a modest speedup of only 3× on the ATI 1950XT. Molemaker et al. (2008) developed a multi-grid method to solve the pressure Poisson equation. The CUDA implementation of the multi-grid pressure Poisson solver produced a speedup of 55× relative to a 2.2MHz AMD Opteron processor26.

We envision that a high-fidelity CFD simulation capability with a rapid computational turn-around time on small-footprint computing systems can transform emergency response and hazard zone prediction for contaminant dispersion in urban environments. In the following, we present the computational performance analysis of a Navier-Stokes solver code that we

develop specifically for small-footprint desktop platforms equipped with multiple GPUs.

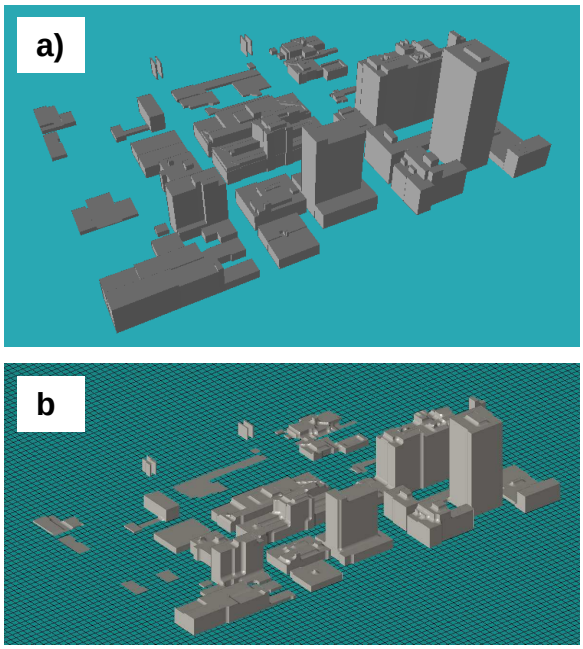


Figure 3: Mesh generation for urban environments. a) Visualization of the urban domain in ArcGIS b) Isosurface visualization of the building flags in Matlab, with a uniform mesh of 4m resolution in the horizontal.

2. MULTI-GPU IMPLEMENTATION

Multi-GPU parallel implementation of the present incompressible Navier-Stokes solver is explained in detail in Thibault and Senocak (2009). To the best of our knowledge, the work of Thibault and Senocak (2009) is the first CUDA implementation of a 3-D incompressible Navier-Stokes solver on multi-GPU desktop platforms. In this paper, we briefly summarize the main features of the code.

Second-order accurate central difference scheme is used to discretize the advection and diffusion terms of the Navier-Stokes equations on a uniform staggered grid. First-order accurate, explicit Euler scheme is used for the time derivative term. The projection algorithm (Chorin, 1968) is adopted to find a numerical solution to the Navier-Stokes equation for incompressible fluid flows. We use CUDA programming model of NVIDIA to implement the discretized form of the Navier-Stokes equations on desktop platforms with multiple GPUs. Communication among GPUs is enabled with POSIX threading.

The main steps of the projection algorithm (Chorin, 1968) are implemented with separate CUDA kernels, and a unique implementation that exploits the memory hierarchy of the CUDA programming model is suggested in Thibault and Senocak (2009). Kernels for the velocity predictor step and the solution of the pressure Poisson equation were implemented using the shared memory of the device, whereas a global memory implementation was pursued for the kernels that are responsible to calculate the divergence field and velocity corrections and to apply the boundary conditions. This unique combination resulted in factor of two speedup relative to global memory only implementation on the device (Thibault and Senocak, 2009)

The GPU computing hardware that is shown in Figure 2 was used in this study. A dual-CPU/dual-GPU platform was built in-house with an Intel Core 2 Duo (E8400) 3.0 GHz CPU, 4GB of host memory and two Tesla C870 boards. Each Tesla board provides 128 streaming processor cores and 1.5 GB of global device memory. A second platform with 8 AMD Opteron 2.4 GHz (8216) dual-core CPUs that is connected to a Tesla S870 server via two PCIe 16× slots provides a total four Tesla GPUs and 16 CPU cores. Each GPU board used in this study can deliver a theoretical peak performance of 512 GFLOPS, according to the manufacturer. These two high performance computing platforms with different GPU-CPU configurations were used to perform speedup and multi-GPU scaling analysis.

3. MESH GENERATION FOR URBAN DOMAINS

Mesh generation for urban domains can be quite tedious depending on the meshing strategy. An unstructured mesh strategy may lead to skewed cells in complex spaces between buildings, which may then cause numerical errors in the solution. A practical fast approach has been adopted in Burrows et al. (2004) and Gowardhan (2008) to generate computational meshes for urban domains. We adopt the same approach in this study.

An Environmental Systems Research Institute (ESRI) shape file containing the vertices of polygons and their heights is imported into the ArcGIS software, and the polygons extruded to their elevations. The result is a three dimensional block representation of a sample set of buildings. The region of interest for atmospheric transport and dispersion simulations is then captured within ArcGIS software and the subdomain is saved as a shape file. Figure 3a shows several blocks from the Oklahoma City domain.

The same shape file is then read into Matlab and the points are converted from UTM coordinates to latitude and longitude coordinates. A grid resolution is provided by the user and the polygons are rasterized to a two dimensional matrix with a resolution of dx and dy. The two dimensional matrix is then grown into a three dimensional matrix with a resolution of dx, dy and dz.. Cells within buildings are then flagged to be used in the CFD code to impose boundary conditions. Figure 3b shows the isosurface visualizaition of the flag value and grid distribution in the horizontal plane.

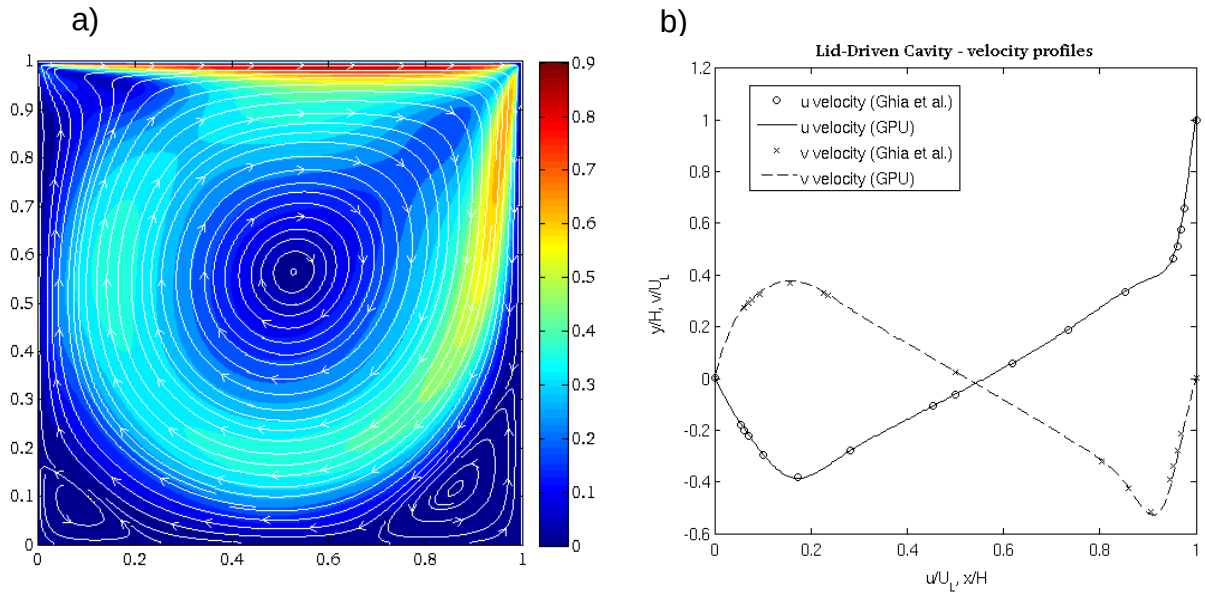


Figure 4: a) Distribution of velocity magnitude and streamlines at steady-state for Re=1000. Low velocity regions are represented in dark blue while high velocity regions are represented in red. b) Comparison of the multi-GPU implementation of our CFD code results with benchmark data given in Ghia et al. (1982).

4. RESULTS AND DISCUSSIONS

In every CFD code development effort care has to be taken to validate the correct implementation of the governing equations with proper boundary conditions. The lid-driven cavity problem (Ghia et al. 1982), in which the fluid inside the cavity is driven by motion of the lid, is a well-established benchmark case in the CFD field. We use the lid-driven cavity problem to validate our multi-GPU implementation.

Streamlines at steady-state are shown in Figure 4a. The flow structure inside a cavity for various Reynolds numbers is well established. For Re=1000, calculated based on the constant lid velocity and cavity height, a main circulation is observed at the core of the cavity, smaller recirculation zones at the bottom corners. The size of these corner vortices depends on the Reynolds number. At this Reynolds number, the flow is laminar and remains two-dimensional. Note that we adopt 3D computations to assess the computational performance of GPUs for large computational problems. Otherwise, the simulation can be performed by 2D computations.

Figure 4b shows the velocity field taken at the middle section in the vertical plane when steady-state condition is reached. The present results obtained from our multi-GPU CFD code are in excellent agreement with the results of Ghia et al. (1982).

Figure 5 summarizes our multi-GPU CFD code performance relative to the serial CPU version of our CFD code. We note that both the GPU and CPU versions of the CFD code adopt the same numerical methods. The serial code was written in C programming language and it was optimized to obtain fair comparisons in performance relative to the GPU version of our CFD code.

Using only a single CPU core, the serial CPU version of our CFD code takes 82,930 seconds on the Intel Core 2 Duo 3.0 GHz CPU and 218,580 seconds on AMD Opteron 2.4 GHz CPU to simulate the lid-driven cavity problem with a computational grid of 1024×32×1024 for 10,000 time steps.

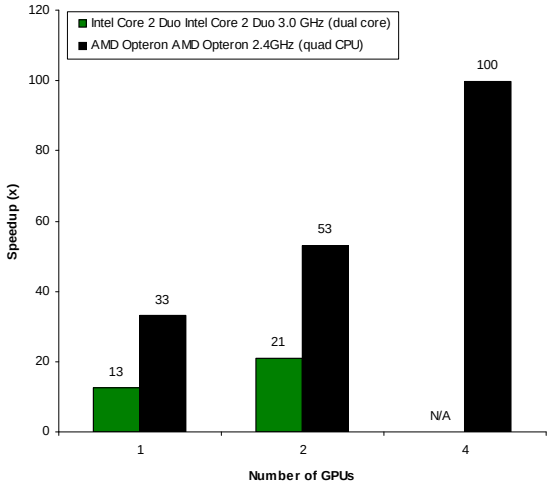
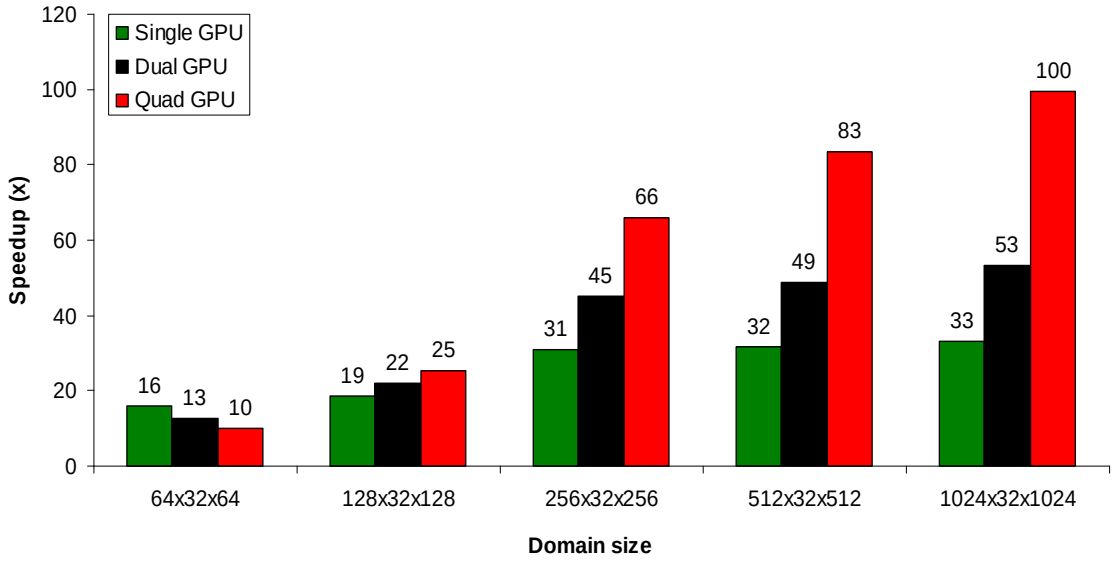


Figure 5: GPU code speedup relative to the serial CPU code for a domain of 1024×32×1024 computational nodes. Quad-GPU results are currently not available for the Intel Core 2 Duo platform, because we do not have the hardware available for the present studv.

a)



b)

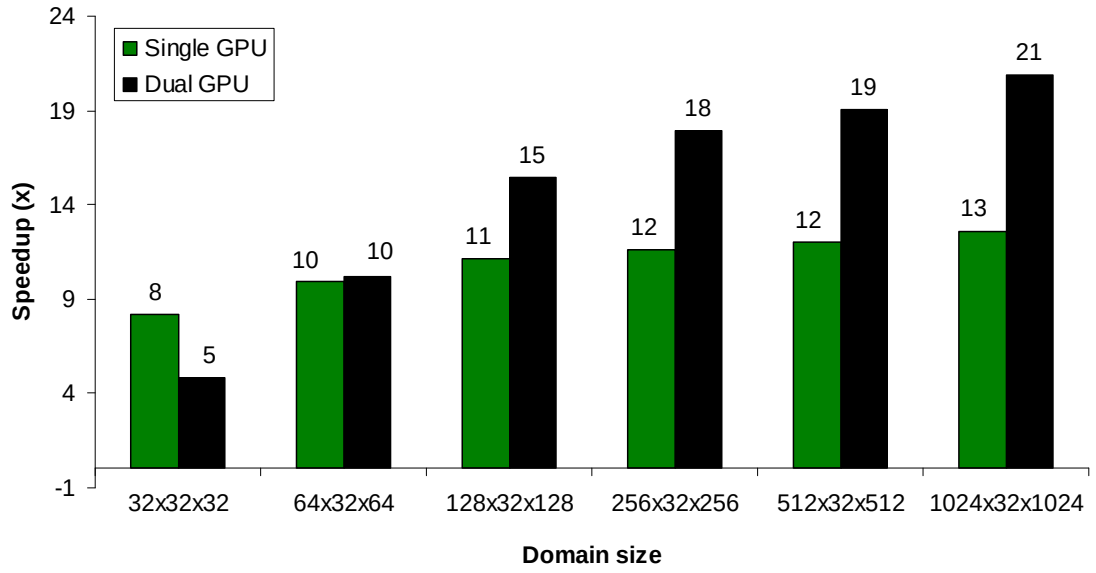


Figure 6: Single and multi-GPU speedup on the NVIDIA S870 Quad Tesla server relative to a single CPU core (AMD Opteron 2.4GHz) b) Intel Core 2 Duo 3.0GHz with dual NVIDIA C870 Tesla boards.

The serial CPU version of the CFD code runs faster on Intel Core 2 Duo CPU than on AMD Opteron CPU because of its larger L2 cache and its better clock frequency. On the other hand the execution time for the GPU code is barely dependent on the CPU clock speed. GPU performance was nearly the same on both the Intel and AMD platforms. As a result GPU performance relative to the CPU performance is better for the AMD Opteron 2.4 GHz platform as shown in Figure 5. On our Intel Core 2 Duo platform the GPU code performs 13 and 21 times faster than the CPU code with one and two GPUs, respectively. On the AMD Opteron 2.4 GHz platform the GPU code performs 33, 53 and 100 times faster using one, two and four GPUs respectively.

Figure 6 shows computational speedup with respect to different problem sizes. On the AMD Opteron platform (Figure 6a), depending on the problem size, the quad-GPU performance varies from 10× to 100× relative to the serial CPU version of the CFD code. On the Intel Core 2 Duo platform (Figure

6b), the dual-GPU performance varies from 5× to 21×. The speedup numbers are impressive for large problem size, because the arithmetic intensity on each GPU increases with problem size, and the time spent on data communication with other GPUs compared to the time spent on computation becomes relatively shorter.

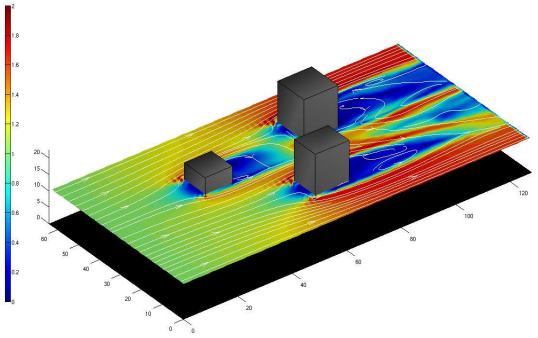


Figure 7: Preliminary simulation of low Reynolds numbers flow through an idealized urban domain simulation using the GPU code. Snapshot of velocity magnitude and streamlines are shown.

We have implemented new kernels in the GPU version of our CFD code to impose boundary conditions on building surfaces. The implementation takes advantage of the flagging approach to detect buildings in the computational domain. Figure 7 shows preliminary results from a low Reynolds number flow around three buildings with different heights. The snapshot of the velocity magnitude and streamlines shows that the current implementation can capture the buildings wakes reasonably. Our future work will focus on extending the current capability to complex urban domains as shown in Figure 3 and validate the computational results with experimental data.

5. CONCLUSIONS

We have presented a computational performance analysis of a Navier-Stokes solver for incompressible flows. The solver adopts NVIDIA's CUDA programming model to implement the discretized form of the governing equations on desktop supercomputers with multiple GPUs. The solver has been extended to address flow around complex urban geometry.

Overall, we have accelerated the numerical solution of incompressible fluid flow equations by a factor of 100 using the NIVIDIA S870 Tesla server with four GPUs. The speedup number is measured relative to the serial CPU version of our CFD code that was executed using a single core of an AMD Opteron 2.4 GHz processor. With respect to a single core of an Intel Core 2 Duo 3.0 GHz processor, we have achieved a speedup of 13 and 21 with single and dual GPU (NVIDIA Tesla C870) platforms, respectively. Same numerical methods were adopted in both the CPU and GPU versions of the CFD code. We have observed that multi-GPU scaling and speedup results improve with increasing computational problem size, suggesting that computationally "big" transport and dispersion problems in urban environments can be tackled with GPU clusters with multiple GPUs in each node.

Finally, our results suggest that multi-GPU desktop supercomputers can accelerate CFD simulations of transport and dispersion in urban environments substantially. It is envisioned that desktop supercomputers can serve as a cost-effective on-demand computing platform to arm the first responders with effective rapid-response simulation tools.

ACKNOWLEDGMENTS

The authors thank Drs. Massimiliano Fatica, Patrick Legresley, David Luebke from NVIDIA and Timothy J. Barth from NASA Ames Research Center for helpful discussions. Thanks are extended to Marty Lukes and Luke Hindman of Boise State University for their help on building our desktop supercomputer and NVIDIA Corporation and Micron Technology, Inc. for hardware donations. This work is partially funded by NASA Idaho EPSCoR Research Initiation grant.

REFERENCES

Brandvik, T. and G. Pullan, 2008: Acceleration of a 3D Euler solver using commodity graphics hardware," 46th AIAA Aerospace Sciences Meeting and Exhibit, Reno NV.

Buck, I., T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, 2004: Brook for GPUs: Stream computing on graphics hardware. *ACM Transactions on Graphics*, **23**, 777-786.

Burrows, D., R. Keith, S. Diehl and E. Hendricks, 2004: A fast-running urban airflow model, 13th Conference on the Applications of Air Pollution Meteorology with the Air and Waste Management Assoc., Vancouver, BC, Canada, 23-28 August.

Chorin, A. J. 1968: Numerical Solution of the Navier-Stokes equations. *Mathematics of Computation*, Vol. **22**(104), 745-762.

Fan, Z., F. Qiu, A. Kaufman, and S. Yoakum-Stover, 2004: GPU cluster for high performance computing. Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, IEEE Computer Society, Washington, DC. p. 47.

Ghia, U., K.N. Ghia, and C. T. Shin, 1982: High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method," *J. Comp. Phys*. **48**, 387-411.

Gowardhan, A.A., E.R. Pardyjak, I. Senocak and M.J. Brown, 2007: A CFD based wind solver for a fast response dispersion model. Seventh Biennial Tri-Laboratory Engineering Conference, Albuquerque, New Mexico, May 7-10.

Gowardhan, A., 2008: Towards understanding flow and dispersion in urban areas using numerical tools, Ph.D Thesis, University of Utah, UT.

Hanna, S.R, M.J. Brown, F.E. Camelli, S.T. Chan, W.J. Coirier, O.R. Hansen, A.H. Huber, S. Kim, and R.M. Reynolds. 2006: Detailed simulations of atmospheric flow and dispersion in downtown Manhattan: An application of five computational fluid dynamics models. Bull. Amer. Meteor. Soc. **87** (12). 1713-1726.

Heagy, J., N. Platt, S. Warner and J. Urban, 2007: Joint Effects Model Urban IPT, Chemical Biological Information Systems Conference and Exhibition. Austin, Texas, 8-11 January.

Li, W., Z. Fan, X. Wei, and A. Kaufman, 2005: "GPU-based flow simulation with complex boundaries," GPU Gems 2, Addison-Wesley, Boston, MA, 747–764.

Molemaker, J., J. M. Cohen, S. Patel, and J. Noh, 2008: Low viscosity flow simulations for animation. Eurographics/ACM SIGGRAPH Symposium on Computer Animation, Eurographics Association, Aire-la-Ville, Switzerland.

NVIDIA 2008a: NVIDIA CUDA Compute Unified Device Architecture Programming Guide, Version 2.0

NVIDIA, 2008b: CUDA Zone, the resource for CUDA developers.
http://www.nvidia.com/object/cuda_home.html

Owens, J. D., D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell, 2007: A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum,* **26**, 80-113.

Owens, J.D., M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips 2008: GPU computing. Proceedings of the IEEE, **96**, IEEE Publishing, 879-899.

Pardyjak, E. R., B. Singh, A. Norgren, and P. Willemsen, 2007: Using video gaming technology to achieve low-cost speed up of emergency response urban dispersion simulations. Seventh Symposium on the Urban Environment, San Diego CA.

U.S. Government Accountability Office, 2008: Homeland security: First responders' ability to detect and model hazardous releases in urban areas is significantly limited. GAO-08-180.

Thibault J.C. and I. Senocak, 2009: CUDA Implementation of a Navier-Stokes Solver on Multi-GPU Desktop Platforms for Incompressible Flows. 47th AIAA Aerospace Sciences Meeting and Exhibit, Orlando FL. AIAA-2009-758.

Tolke, J. and M. Krafczyk 2008: TeraFLOP computing on a desktop PC with GPUs for 3D CFD," *International Journal of Computational Fluid Dynamics*, **22** (7), 443-456.

Wang, Y., J. J. Mercurio, C. C. Williamson, D. M. Garvey, and S. Chang, 2003: A high resolution, three-dimensional, computationally efficient, diagnostic wind model: Initial development report. U.S. Army Research Laboratory, Adelphi, MD. ARL-TR-3094.

Willemsen, P., A. Norgren, B. Singh and E.R. Pardyjak, 2007: Development of a new methodology for improving urban fast response Lagrangian dispersion simulation via parallelism on the graphics processing unit. Proceedings of the 11th International Conference on Harmonisation within Atmospheric Dispersion Modelling for Regulatory Purposes, Queen's College, University of Cambridge, United Kingdom, July 2-5.

Williams, M. D., M. J. Brown, B. Singh, and D. Boswell, 2004: QUIC-PLUME Theory Guide, Los Alamos National Laboratory, LA-UR-04-0561.