Fast Parallelized Flow Simulations on Graphic Processing Units

Wangda Zuo and Qingyan Chen

National Air Transportation Center of Excellence for Research in the Intermodal Transport Environment (RITE), School of Mechanical Engineering, Purdue University, USA

Corresponding email: yanchen@purdue.edu

ABSTRACT

To design a comfortable and safe indoor environment, it is crucial to know the distributions of air velocity, air temperature and contaminant concentrations in the indoor space. The distributions can be obtained by computer simulations, such as Computational Fluid Dynamics (CFD). However, the CFD requires intensive computing efforts to solve the Navier-Stokes equations, energy equation and species concentration equations. If the computing domain is as large as a building, the computing time on a single Central Processor Unit (CPU) will become too long for most design applications. Parallelized computing on multiprocessor-computers can reduce the computing time, but the costs of such computers and their maintenance is prohibitively high for many users. Hence, it is essential to seek an alternative to speed up the CFD calculations and to reduce the computing costs. This investigation performed flow simulations in parallel on a graphics process unit (GPU). A GPU consists of hundreds of processors, which offers a great power for parallel computing. The price of a GPU is less than $1,000 and it has almost no maintenance cost. A Fast Fluid Dynamics (FFD) model was implemented on the GPU. The FFD solves the Navier-Stokes equations as the CFD does. By scarifying some accuracy, the FFD can be about 50 times faster than the CFD with the same numerical settings. Our results show that the FFD simulations on the GPU can provide the same results as those on the CPU, but the GPU version is up to 30 times faster than the CPU version. As a result, it is possible to perform real-time simulations of airflow in a moderate size building by using the FFD on the GPU. Meanwhile, this study indicates that the GPU can be also used to accelerate other scientific computing, such as CFD simulations.

INTRODUCTION

To design sustainable buildings that can provide a comfortable and healthy indoor environment, it is essential to know air distributions in buildings. CFD is an important tool to study the air distributions [1]. By solving Navier-Stokes equations, the CFD can provide detailed and accurate flow information [2]. However, when the simulated flow domains are large or the flow is complex, the computing time needed for such a flow by the CFD on a single processor computer will be very long [3].

A traditional method to accelerate the flow simulation is to use multi-processor computers [4, 5], but the cost is very high. It is not only expensive to purchase the hardware and software but also to find a suitable space for installing and cooling the computers [6]. In addition, the operation and maintenance can also be costly [7]. Hence, these multi-processor computers are too expensive to be widely used in building simulations.

Acceleration of the flow simulation speed can be achieved through the improvements on both the flow models and computing hardware. Our early effort [8] applied a FFD technique to solve the governing equations in the flow models. The FFD can run about 50 times faster than the CFD with the same numerical setting. Although the FFD is not as accurate as the CFD, it can provide almost the same flow information as the CFD does.

This investigation focused on the improvement of computing hardware by performing FFD simulations of room airflow on a GPU. The GPU is the core of a computer graphic card that has hundreds of processors and costs around $500. The GPU can be easily installed into a personal computer and does not need maintenance.

METHODS

The FFD model used in our investigation was proposed by Stam [9]. The FFD solves the continuity equation and Navier-Stokes equations for an unsteady, incompressible flow. It firstly solves the diffusion term by Gauss-Seidel method; then the advection term by a semi-Lagrangian approach [10]. Finally, it ensures the mass conservation by a pressure-correction projection method [11].

To implement the FFD model on the GPU, this investigation used a GPU Programming language named Computer Unified Data Architecture (CUDA) [12]. CUDA is an extended C language that allows users to manipulate the GPU without knowing the details of the hardware. If a flow simulation code is written in C, it can be executed on GPU by re-writing the parallel computing part using CUDA. This feature can save a huge amount of time on code development.

CUDA divides a GPU into three levels (Figure 1). The highest level is called “grid”. Each grid consists of multiple “blocks”, and every block has many “threads”. A thread is the basic computing unit of the GPU. Mathematic and logic operations are performed on the threads. The current implementation assigns one mesh point to one thread and the dimension of a block is fixed as 16×16 threads. Since there is only one grid in our implementation, the number of blocks can be decided according to the mesh points used in the simulation.

Figure 2 shows the program structure. The implementation used the CPU to read, initialize and write data. The solver, which needed to be parallelized, was implemented on the GPU.

RESULTS

This study was to evaluate the performance of the FFD on GPU. The investigation first compared the results obtained by the GPU version with those by the CPU version. Then the computing speed on the GPU would be compared with that on the CPU.

Comparison of the FFD results obtained by CPU and GPU

This investigation used the FFD to calculate two airflows relevant to indoor environment: flow in a lid-driven cavity and fully developed flow in a plane channel. The simulations used the exactly same meshes and numerical settings were performed on GPU and CPU, respectively.

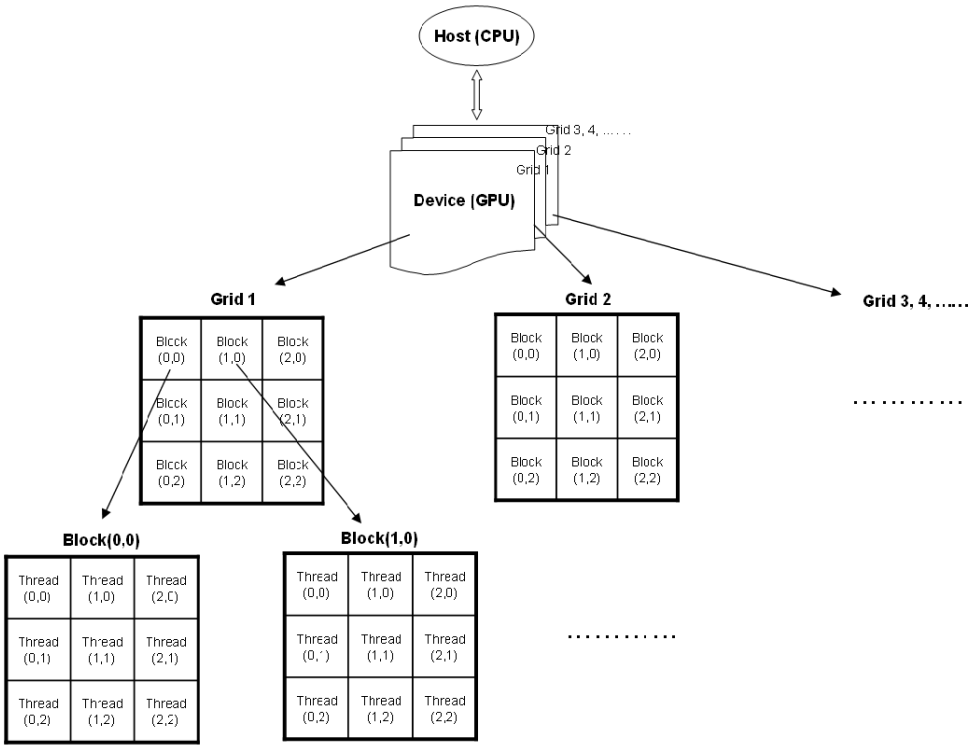


Figure 1 The schematic of parallel computing on CUDA

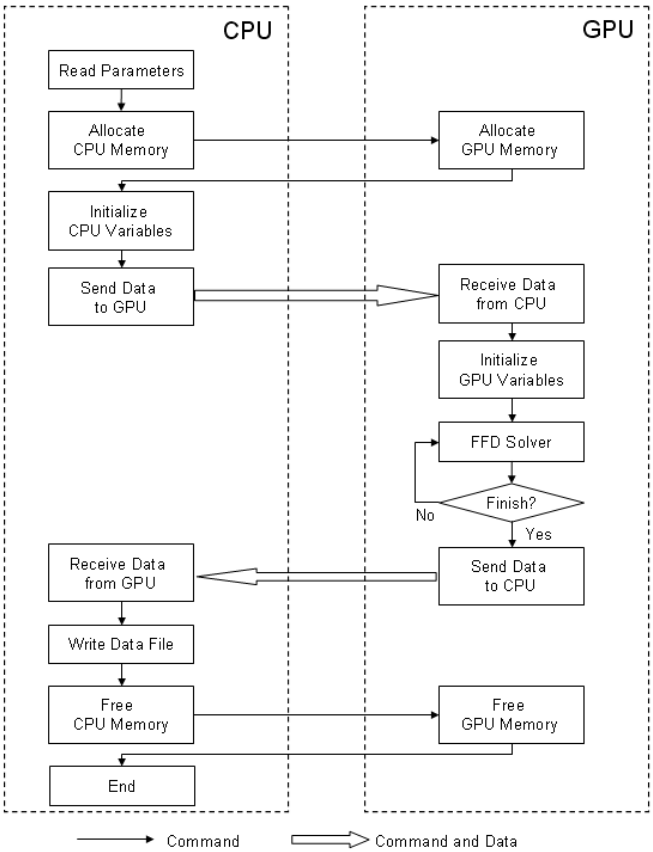
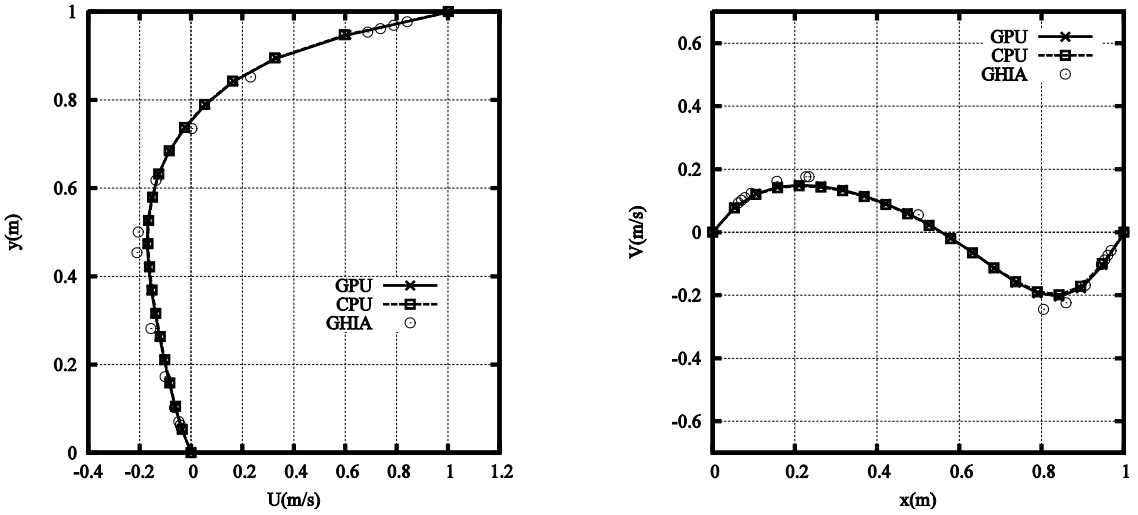


Figure 2.The schematic for implementing the FFD on the GPU

The flow in a lid-driven cavity was a classical case for numerical validation. The flow looks like the one with air supply from a jet near the ceiling in a room. Based on the lid velocity of U = 1 m/s, cavity length of L = 1m, and kinematic viscosity of ν = 0.01 m^2/s, the Reynolds number of the flow studied was 100. A mesh with 32 × 32 grid points was applied. The reference data was the high quality CFD results obtained by Ghia et al [13]. As shown in Figure 3, the GPU version could predict the same velocity profiles as the CPU version. Furthermore, the FFD results were close to the reference data. Although this is a simple case, it proves that a GPU could be used for numerical simulations as a CPU. The FFD model worked as beautiful as the CFD model for this particular case.



(a) (b)

Figure 3. The comparison of the calculated velocity profiles by the FFD on the CPU and GPU with the reference data [13]. (a) Horizontal velocity at x = 0.5 L (b) Vertical velocity at y = 0.5 L

The second flow studied was a fully developed flow in a plane channel. This flow is similar to that in a corridor of a building. The Reynolds number of the flow studied was 2800, based on the mean bulk velocity U_b and the half channel height, H. A mesh with 65 × 33 grid points was adopted by the FFD simulations. The Direct Numerical Simulation (DNS) data from Mansour et al. [14] was selected as a reference. Figure 4 compares the predicted velocity profiles by the FFD on CPU and GPU, respectively. The FFD version on GPU could also produce the same velocity profile as the version on CPU, although both the predictions gave more laminar like profiles when compared with the DNS data. Obviously, the disagreement was caused by the FFD method, not the GPU.

The above two cases show that the FFD code on the GPU produced the same results as the FFD on the CPU. Thus, the GPU could be used for flow simulations.

Comparison of the computing speed on CPU and GPU

To compare the FFD simulation speed on the GPU with that on the CPU, this study used the computing time used for the lid-driven cavity flow. The simulations were performed on a HP workstation with an Intel XeonTM CPU and an NVIDIA GTX 8800 GPU. The data was for 100 time steps. The mesh size was varied to study its impact on the computing time.

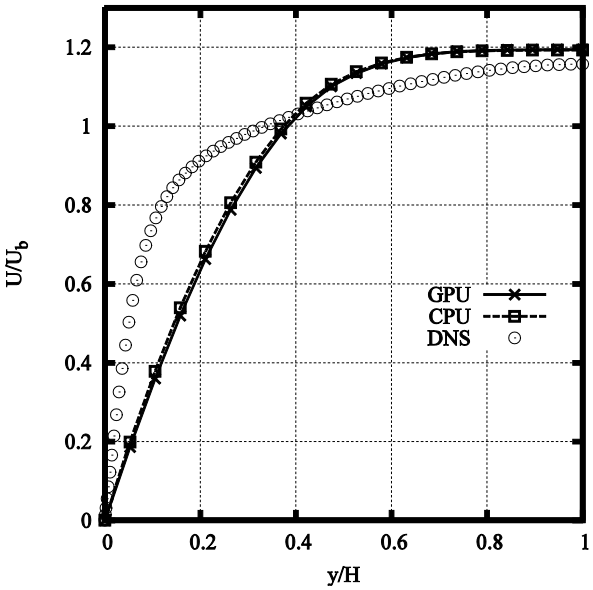


Figure 4. Comparison of the mean velocity profile in a full developed channel flow predicted by the FFD on CPU and GPU with the DNS data [14].

Figure 5 illustrates that the CPU computing time increased linearly with the mesh size. When the number of meshes was smaller than 3.6×10^3, the CPU version was faster than the GPU version. Since it took time to transfer data between CPU and GPU during the GPU simulation, this amount of time could be more significant than that saved in the parallel computing when the mesh size was small. Hence, the parallel computing on the GPU should be applied to cases with a large mesh size.

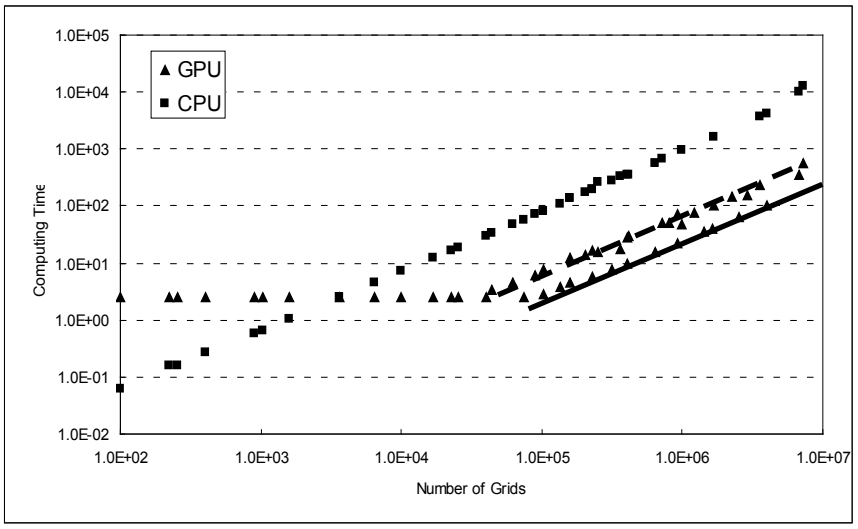


Figure 5. Comparison of the computing time used by the FFD on the GPU with that on the CPU

One can also notice that the GPU computing time was almost constant when the mesh size was less than 4×10^4. This is because the mesh size was not large enough for the GPU to fully use its capacity. When the mesh size was greater than 4×10^4, the GPU computing time increased along two paths. Those points on the solid line were for the cases with a mesh size in multiples of 256 and on the dashed line the mesh size cannot be divided exactly by 256. As mentioned previously, each mesh node was assigned to one thread and a block had 256 threads. If the mesh size was exactly the multiplication of 256, all the 256 threads of every

block were utilized. Thus, the working load among blocks was equal. Otherwise, some of threads in the block were rendered idle and the working load between blocks was unequal. The imbalance of GPU working loads can have a severe penalty on the performance. For example, the case with 640 × 640 meshes that was multiplication of 256 took 9.977 s, but that with 639 × 639 took 28.875 s. Although the latter case had fewer meshes, its computing time increased by almost two times.

Nevertheless, the GPU version is still much faster than the CPU version even for the cases on the dash line. The GPU computing time for the cases on the dashed line was still about 10 times shorter than that of the CPU. The difference increased to around 30 times when a right amount of meshes was used (solid line).

Discussion

This study implemented the FFD solver for flow simulations on the GPU. Since the FFD solves the same governing equations as the CFD, it is also possible to implement the CFD solver on the GPU by using a similar method. One can also expect that the speed of CFD simulations on the GPU should be faster than that on the CPU. For those CFD codes written in C language, the implementation will be relatively easy since only the parallel computing part needs to be re-written in CUDA.

The current GPU computing speed can be further accelerated by optimizing the implementation. For example, the dimensions of the GPU blocks can be flexible to adapt to different meshes. Meanwhile, many classical optimization techniques for paralleling computing are also good for the GPU computing. For instance, to read or write data from GPU memory is time consuming so that the processors are often rendered idle for data transmission. One approach to improve this situation is to reuse the data already on the GPU by calculating several neighboring mesh nodes with one thread.

CONCLUSION

This paper introduced an approach to conduct high performance and low cost parallel computing on a GPU for flow simulations. A FFD code has been implemented for flow simulations on a GPU. The implementation used CUDA language that is compatible to C language. By applying the code for flow in a lid driven cavity and a channel flow, this investigation found that the FFD code on the GPU produced the same results as that on the CPU. However, the FFD code may or may not generate the same results as a CFD code.

A flow simulation with the FFD code on GPU was 30 times faster than that on CPU when the mesh size was the multiplication of 256. If the mesh size cannot be exactly of the multiplication of 256, the simulation was still 10 times faster than that on CPU.

ACKNOWLEDGEMENT

This project was funded by U.S. Federal Aviation Administration (FAA) Office of Aerospace Medicine through the National Air Transportation Center of Excellence for Research in the Intermodal Transport Environment (RITE) Cooperative Agreement 04-C-ACE-PU-002. Although the FAA has sponsored this project, it neither endorses nor rejects the findings of this research. The presentation of this information is in the interest of invoking technical

community comment on the results and conclusions of the research.

REFERENCES

1. Nielsen, P V. 2004. *Computational fluid dynamics and room air movement.* Indoor Air, Vol. 14, pp 134-143.
2. Ferziger, J H and Peric, M, 2002. *Computational methods for fluid dynamics*. 3rd, rev. ed. Berlin, New York: Springer.
3. Lin, C, Horstman, R, Ahlers, M, et al. 2005. *Numerical simulation of airflow and airborne pathogen transport in aircraft cabins - Part 1: Numerical simulation of the flow field.* ASHRAE Transactions, Vol. 111.
4. Mazumdar, S and Chen, Q. 2008. *Influence of cabin conditions on placement and response of contaminant detection sensors in a commercial aircraft.* Journal of Environmental Monitoring, Vol. 10(1), pp 71-81.
5. Hasama, T, Kato, S, and Ooka, R. 2008. *Analysis of wind-induced inflow and outflow through a single opening using LES & DES.* Journal of Wind Engineering and Industrial Aerodynamics, Vol. 96(10-11), pp 1678-1691.
6. Feng, W and Hsu, C. 2004. *The Origin and Evolution of Green Destiny*. in *IEEE Cool Chips VII: An International Symposium on Low-Power and High-Speed Chips*. Yokohama, Japan.
7. Dongarra, J J. 1988. *The LINPACK Benchmark: An explanation*, in *Supercomputing*. Springer Berlin / Heidelberg. p. 456-474.
8. Zuo, W and Chen, Q. 2008. *Real-time or faster-than-real-time simulation of airflow in buildings.* Indoor Air, doi:10.1111/j.1600-0668.2008.00559.x, Vol.
9. Stam, J. 1999. *Stable Fluids*. in *26th International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'99)*. Los Angeles.
10. Courant, R, Isaacson, E, and Rees, M. 1952. *On the Solution of Nonlinear Hyperbolic Differential Equations by Finite Differences.* Communication on Pure and Applied Mathematics Vol. 5, pp 243–255.
11. Chorin, A J. 1967. *A Numerical Method for Solving Incompressible Viscous Flow Problems.* Journal of Computational Physics, Vol. 2(1), pp 12-26.
12. NVIDIA, 2007. *NVIDIA CUDA Compute Unified Device Architecture-- Programming Guide (Version 1.1)*. Santa Clara, California: NVIDIA Corporation.
13. Ghia, U, Ghia, K N, and Shin, C T. 1982. *High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method.* Journal of Computational Physics, Vol. 48(3), pp 387-411.
14. Mansour, N N, Kim, J, and Moin, P. 1988. *Reynolds-stress and dissipation-rate budgets in a turbulent channel flow.* Journal of Fluid Mechanics, Vol. 194, pp 15-44.