A Three Layered Framework for Annual Indoor Airflow CFD Simulation (Part I)

Yue Wang[1,*], Ali Malkawi[1], Yun Kyu Yi[1], Ning Feng[1]

[1]T.C. Chan Center for Building Simulation and Energy Studies, University of Pennsylvania, PA, 19104, USA

ABSTRACT

Computational fluid dynamics (CFD) is one of the branches of fluid mechanics that uses numerical methods and algorithms to solve and analyze problems that involve fluid flows. Fast flow simulations are needed for some applications in building industry, such as the conceptual design of indoor environment, or coupled with energy simulation to provide deep analysis on the performance of the buildings. Such applications need a similar level of accuracy as traditional CFD simulation, since they require conceptual or semi-accurate distributions of the flow within a short computing time. Year round simulation is needed instead of two or three extreme cases, to help the designer investigate the problem clearly. However, CFD computation is usually time consuming with an inability to simulate real-time indoor air movement. This research will provide a three layered framework to optimize CFD to cover annual hourly simulation. Most of the literature will be reviewed based on the framework. Currently, many researchers are concentrating on hardware utilization to make simulation much faster so in this paper, latest researches using this methodology will be thoroughly reviewed.

KEYWORDS

CFD, FFD, GPGPU, Machine Learning, Annual Hourly Simulation, Building Simulation

INTRODUCTION

Annual hourly simulation is important in many applications in the building simulation field. The building envelope changes its control parameters according to the outdoor condition and internal load, and both change rapidly throughout a year. The amount of solar energy received is the main cause of the daily and yearly temperature variations. These temperature variations also create forces that drive the atmosphere in its endless motions, which also changes the outdoor humidity. Indoor air control system relies on outdoor temperature and humidity values and calculates the enthalpy from it, and thus makes the next hour's control strategies. Moreover, occupants' activity will also be changed according to the daily and weekly schedule. As a result, the hour to hour the

[*] Corresponding author email: yuleopen@gmail.com

indoor condition is entirely different. This is the reason why most simulation algorithms in building simulation, like energy simulation and sun radiation simulation, were all done annually.

Computational Fluid Dynamics (CFD) is one of the branches of fluid mechanics that uses numerical methods and algorithms to solve and analyze problems that involve fluid flows. CFD is widely used in building simulation studies. Computers are used to perform the millions of calculations required to simulate the interaction of liquids and gases with surfaces defined by boundary conditions.

Fast indoor airflow simulations are necessary for designing building emergency management systems, the preliminary design of sustainable buildings, and modeling real-time indoor environment control. The simulation should also be informative since the airflow motion, temperature distribution, and contaminant concentration is important. However, CFD computation is usually time consuming and unable to simulate annual indoor air movement. Thus only one or two extreme cases (like the hottest hour in the summer and coldest hour in the winter) will be simulated by CFD in most of the design analysis.

This creates many problems. Extreme cases will give extreme results. Most realistic hourly results throughout the year will reside in much closer boundaries. Thus most assumptions and conclusions drawn from those extreme cases of simulations will be exaggerated. Moreover, while doing coupling simulation with other types of simulation (for instance, energy simulation or particle simulation) with hourly time step, a few extreme cases can hardly suffice. For instance, Faculteit Bouwkunde(2005) suggests that CFD should be performed hourly when doing coupling simulation with energy simulation.

This leads to an important and urgent problem in the field. Given a geometry with yearly boundary conditions, it is important to speed up the traditional simulation strategies to cover annual simulation (8760 static state case) in a fairly reasonable amount of time and acceptable precision. This research provides a way to utilize various methods and algorithms in order to speed up the simulation, and gives a tentative solution to make annual simulations possible.

A THREE LAYER FRAMEWORK FOR THE CFD SIMULATION

The traditional CFD workflow for solving annual hourly simulation will naively work as this:

- Iterate all through annual hourly boundary conditions, and solve them one by one.
- For each of the simulation cases, CFD will turn the boundary conditions into an equation set.
- When the final equations set are formulated, an iterative solver will be used to solve the equation set.

In this traditional simulation strategy, there are three obvious different layers, the application level, the model level, and the mathematical level from top to bottom. Since this workflow will take months or even years to simulate an annual hourly case, certainly something should be changed in order to make it more efficient. Most research conducted only focus on one level and one can gain some speed by digging one level deeper and deeper.

- Mathematical level: In the mathematical layer, CFD will first try to discrete the equations in the model part into several linear equations, and try to use an iterative method to solve these systems.
- Model level: In the model layer, differential equations are used to describe the physics of the real world. In a traditional way, Navier-Stokes equations are used to solve the indoor air movement.
- Application level: Among the 8760 hourly cases, many cases are highly similar because the inputs do not change that much, thus reusing the calculated result as much as possible might help to reduce unneeded calculation by a large amount.

However, there are limitations in each level, and one can only get a certain speed in each level. So it is better to speed up all the three levels simultaneously, and try to combine the speedups in all layers together. There is a large amount of paper that contributes to one of the three layers; however, currently the biggest problem is to solve and validate these models for building simulation, and integrate all these layers together to maximize the simulation performance.

Also note all these three layer approaches should be designed like optional plugins. Each one is a standalone simulation optimization strategy, but it is possible to be integrated with other methods. In this way the framework can be general purposed as much as possible.

THE MATHEMATICAL LAYER

The traditional CFD program divides the space into grids and applies the Navier-Stokes equation, turbulent models, boundary conditions and so on to each grid, then solves the problem using Finite Difference Method, Finite Volume Method, or Finite Element Method. Then the program will try to solve the linear system using the conjugate gradient method. A lot of enhancements have been incorporated into the solving strategies of CFD codes recently. Frank Deserno added a strongly-implicit-procedure (SIP) solver to H.L. Stone's(1968). This method is suitable for solving systems of linear equations resulting from a discretisation of partial differential equations (PDEs). It is a widely used method in fluid mechanics and therefore of great importance. The results show 2 to 3 speed ups over the original sparse matrix iterative solver. Also, Johnannes Habich implemented the Bouzidi bounce-back with interpolation to meet the numerical requirements of continuous surfaces.

However, any improvement related to a pure numerical mathematical analysis view is quite limited. Those new solvers might solve certain matrix types faster, but they might perform even worse on other types of matrices since they rely heavily on the form of the matrix. Moreover, these methods would only give limited speed ups.

But with the modern advancement in the semi-conductors development, there are hopes that we can utilize the power of parallel computing to solve this bottleneck completely or at least partially. The CPU clock speed in consumer products increases every year followed by Moore's Law, so that it is possible to perform CFD more efficiently when new CPU chips are available. Moore's Law describes a long-term trend in the history of computing hardware, in which the number of transistors that can be placed inexpensively on an integrated circuit has doubled approximately every two years (Electronics Magazine 1965). For the past 30 years, the CPU transistor counts against dates of introduction closely following Moore's Law (Anon).

However, the exponential processor transistor growth predicted by Moore does not always translate into exponentially greater practical computing performance. For example, the higher transistor density in multi-core CPUs doesn't greatly increase speed on many consumer applications that are not parallelized. In light of the costs for manufacture, the performance per watt, and heat generation are concerned, consumer CPU products developed by AMD and Intel switched from speed improvements over one processor core, to multi core processors (Intel Inc., AMD Multi-core white Paper), which means, building a single thread application that runs on one CPU core will not see dramatic speed benefits in the following years. Major CPU vendors' focus switch from improving clock speed, to integrating more cores to the processors. This means that a non-highly paralleled CFD program will not benefit much from the recent hardware development achievements. Remarkable engineering achievements in graphics processing should also be taken into account when designing the real-time CFD program, or the computation resources in personal computers might be wasted in

vain. Moreover, traditional CFD's algorithms should still be dramatically improved to meet the performance requirements.

In recent years, the GPU (Graphics Processing Unit) has attracted attention for numerical computing. Its structure is highly parallelized and optimized to achieve high performance for image processing. GPU speed has improved dramatically over the past five years; the acceleration is much greater than CPU (Nvidia 2007). Several companies published their specifications and implementations of computational framework on their own GPUs. Compute Unified Device Architecture (CUDA) was developed by NVIDIA cooperation in 2007, followed by Stream SDK (formerly CTM, Close to Metal) (2009) from AMD and DirectCompute from Microsoft in their Windows Vista or Windows 7 (Anon A). In 2009, Apple Inc. developed a new technology called OpenCL which was ratified as a royalty-free open standard in December 2008 (Khronos Group). Mac OS X Snow Leopard, which contains a full implementation of OpenCL (Apple Developers Connection), was released in 2009. AMD and NVIDIA are closely following this step, releasing several OpenCL implementations in beta (AMD, NVIDIA).

Several following papers were published related to utilizing the power of GPU to do FFD (which will be covered later) simulations using the GPGPU frameworks like CUDA (Andrew Corrigan et al, Wangda Zuo and Qingyan Chen). The performance benefit of GPGPU (General Purposed GPU) computing gives hope to solve CFD problems more efficiently. In 2010, Wangda Zuo optimistically wrote, ``It is possible to implement the CFD solver on the GPU by using a similar strategy. One can also expect that the speed of CFD simulations on the GPU should be faster than that on the CPU. For the CFD codes written in C language, the implementation will be relatively easy since only the parallel computing part needs to be rewritten in CUDA" (Wangda Zuo and Qingyan Chen). However, no research in the building simulation field has successfully achieved this goal in order to ship a production tool due to many technical difficulties. The major problem is that most mature CFD programs, like OpenFOAM (OpenCFD Ltd 2009), all have a large source code base, usually millions of lines of code, which has very advanced software architecture, and uses generic programming, object oriented programming wisely. This makes it extremely difficult to convert to OpenCL or CUDA code.

Surprisingly, benchmarking using profiling tools shows that one of the procedures, the conjugated gradient method, takes more than 90% of the overall running time with only 70 lines of code. Other procedures' running time can be neglected. By only optimizing the conjugated gradient code instead of the entire CFD code, great performance can still be achieved. We formally described this approached for the first time in (Yue Wang 2011) with demonstrated source code (Yue Wang 2010). This leads to a series of similar codes and papers released or published in year 2010 and 2011. SpeedIT released a CUDA version for OpenFOAM too (Anon B). Other similar

projects include the ofgpu project released by symscape in early 2011 (Anon C) and Xia Qingfeng's clFOAM released in mid-2011 (Anon D). All of these use the same method.

CONCLUSION

In this paper, we enumerate many different research methods conducted to make CFD much faster. There are three different approaches to speed up the simulation. In this paper, researches using the GPGPU method to improve CFD performance are reviewed, and it has been proved that the optimization of the linear solver can dramatically speed up the computation.

Based on the GPGPU calculation layer, two other layers will be employed to further speed up the CFD computation. The related literature will be reviewed in companion paper (Yue Wang 2012).

REFERENCES

Anonymity http://en.wikipedia.org/wiki/Moore\%27s_Law

Anonymity A http://msdn.microsoft.com/en-us/directx/default.aspx

Anonymity B http://vratis.com/speedITblog/2010/07/official-release/

Anonymity C http://www.symscape.com/gpu-openfoam

Anonymity D http://www.iesensor.com/blog/2011/06/02/281/

Faculteit Bouwkunde, External coupling between building energy simulation and computational fluid dynamics, PhD thesis, Technische Universiteit Eindhoven, 2005

Frank Deserno, Basic Optimisation Strategies for CFD-Codes.

H.L. Stone, Iterative solution of implicit approximations of multidimensional partial differential equations, SIAM J. Numerical Analysis, 5 (5), 1968.

Johannes Habich, Improving computational efficiency of lattice Boltzmann methods on complex geometries, Bachelor Thesis

Cramming more components onto integrated circuits, Electronics Magazine, 19 April 1965

Intel Inc., Intel Multi-Core Processor Architecture Development Backgrounder

AMD Multi-core White Paper

Nvidia. Nvidia CUDA compute unified device architecture, programming guide (version 1.1). Santa Clara, California: NVIDIA Corporation; 2007.

ATI Stream Computing - Technical Overview, 03/20/2009

Khronos Gourp, OpenCL 1.0 Specification

Apple Developers Connection, Technology Brief: OpenCL Taking the graphics processor beyond graphics.

AMD, OpenCL and the ATI Stream SDK v2.0

NVIDIA, NVIDIA OpenCL Programming Overview

Andrew Corrigan, Fernando Camelli, Rainald Lohner, Running Unstructured Grid Based CFD Solvers on Modern Graphics Hardware, 19th AIAA Computational Fluid Dynamics, June 22-25, San Antonio, Texas

Wangda Zuo, Qingyan Chen, Fast and informative flow simulations in a building by using fast fluid dynamics model on graphics processing unit, Building and Environment

OpenCFD Ltd, OpenFOAM 1.6 User Guide, 2009

Yue Wang, Ali Malkawi, Yun Yi, Implementing CFD (Computational Fluid Dynamics) in OpenCL for Building Simulation, 12th Conference of International Building Performance Simulation Association, 2011

Yue Wang, http://code.google.com/p/freefoam-bin/downloads/ 2010

Yue Wang, Ali Malkawi, Yun Yi, Ning Feng, A Three Layered Framework for Annual Indoor Airflow CFD Simulation (Part II), The 1st Asia conference of International Building Performance Simulation Association, 2012