In-Situ Statistical Analysis of Autotune Simulation Data using Graphical Processing Units

Niloo Ranjan

Jibonananda Sanyal

Joshua New

Table of Contents

Figures and Tables

ABSTRACT

Developing accurate building energy simulation models to assist energy efficiency at speed and scale is one of the research goals of the Whole-Building and Community Integration group, which is a part of Building Technologies Research and Integration Center (BTRIC) at Oak Ridge National Laboratory (ORNL). The aim of the Autotune project is to speed up the automated calibration of building energy models to match measured utility or sensor data. The workflow of this project takes input parameters and runs EnergyPlus simulations on Oak Ridge Leadership Computing Facility's (OLCF) computing resources such as Titan, the world's second fastest supercomputer. Multiple simulations run in parallel on nodes having 16 processors each and a Graphics Processing Unit (GPU). Each node produces a 5.7 GB output file comprising 256 files from 64 simulations. Four types of output data covering monthly, daily, hourly, and 15-minute time steps for each annual simulation is produced. A total of 270TB+ of data has been produced. In this project, the simulation data is statistically analyzed in-situ using GPUs while annual simulations are being computed on the traditional processors. Titan, with its recent addition of 18,688 Compute Unified Device Architecture (CUDA) capable NVIDIA GPUs, has greatly extended its capability for massively parallel data processing. CUDA is used along with C/MPI to calculate statistical metrics such as sum, mean, variance, and standard deviation leveraging GPU acceleration. The workflow developed in this project produces statistical summaries of the data which reduces by multiple orders of magnitude the time and amount of data that needs to be stored. These statistical capabilities are anticipated to be useful for sensitivity analysis of EnergyPlus simulations.

I. INTRODUCTION

Automated calibrated building energy models are important for making energy efficient residential and commercial buildings. The purpose of the Autotune project is to quickly modify the automated building models to match measured usage data such as utility bills, sub-meter, and sensor data. The workflow of this project takes input parameters and runs many EnergyPlus simulations on supercomputers. This robust and automated Autotune approach significantly reduces the cost of building energy models[1, 2,5,12]. By leveraging Titan's new NVIDIA Kepler K20 GPU on each node, the capability for massively parallel data processing[10] is leveraged to process sensitivity data while in memory and reduce the amount of data that must be written to disk.

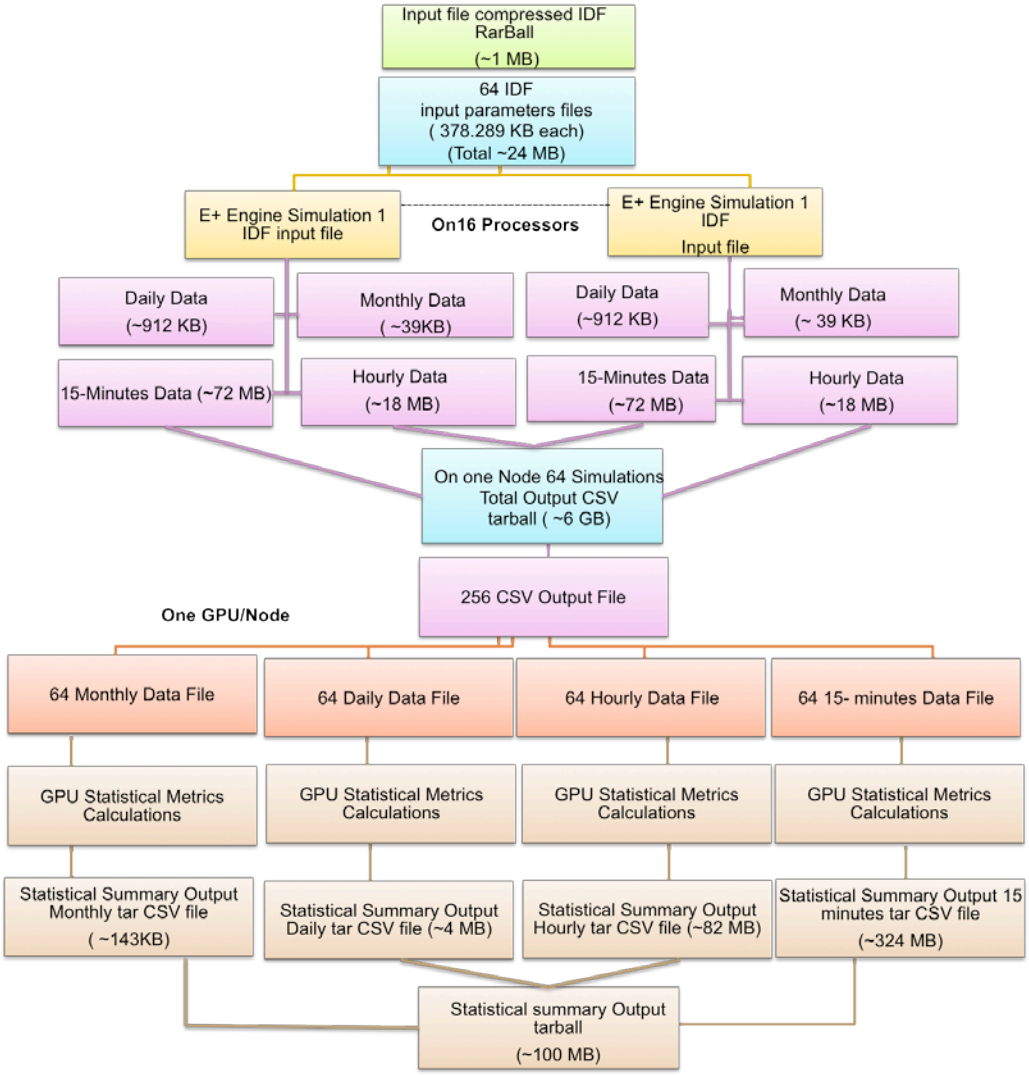

Figure 1. Simulation Workflow with In-situ Statistical analysis of output simulation data

The size of the output data reduces from ~6 GB to ~100 MB with the Statistical summary of all one data type. This workflow shows the simulation process running on one node. Currently, the simulations are running on 8,192 nodes and has potential to run on 16,384 nodes.

A. Graphical Processing Units (GPUs)

GPUs are combinations of thousands of smaller but efficient many-core co-processors that have the capability to accelerate high performance computing. General-Purpose computing on Graphical Processing Units (GPGPU) allows algorithmic execution via many programming

interfaces and can be used to accelerate many scientific and engineering applications. A traditional Central Processing Unit (CPU) often executes the serial portion of the program while the GPU calculates the parallel part of the application using the principle of same instruction on multiple data (SIMD). GPU computing is best for data-parallel computation, such as operations on matrices and vectors, where elements of the data set are independent of each other and can be computed simultaneously[9].

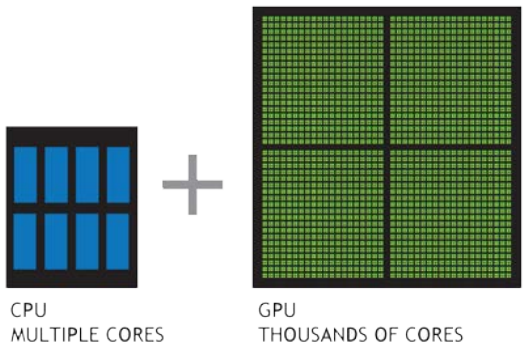

Figure 2. Architecture of CPU and GPU. CPUs and GPUs have different architectures with GPUs containing many cores compared to CPUs. The combination of CPUs and GPUs can greatly accelerate an algorithm process[9].

B. CUDA

CUDA is a general purpose GPU programming model that supports the simultaneous CPU and GPU execution of a program. It has support for many programming languages such as C/C++, FORTRAN, DirectCompute, and OpenACC. It has Application Programming Interfaces (APIs) and libraries for many operations. The libraries include CUBLAS for basic linear algebra applications, CUSPARSE for operations on sparse matrix, CUDA math library for basic math functions, CURAND for random number generation, and CUFFT for Fast-Fourier transformation operations[7].

C. GPU Accelerated Libraries

Several open sources, high-performance, GPU-accelerated libraries are available for general purpose parallel computing and were evaluated in the context of sensitivity analysis for simulation data.

1. Thrust

Thrust is a Standard Template Library (STL) for GPU programming. It is an open source library and also available as a part of NVIDIA cudatoolkit. It supports operating systems such as Linux, Windows, Mac OSX. It is interoperable with CUDA C, OpenMP, and Intel's thread building blocks (TBB). It provides the data parallel functions for scan, search, search by key, count, merge, reorder, prefix-sum, set, sort, and transform[8].

2. CUDPP

CUDPP is an open source CUDA data parallel primitive library. It is compatible with NVIDIA CUDA 3.0 or better and requires cudatoolkit to be installed. It supports the CentOS Linux, Windows 7, and Mac OS X operating systems. It has an interface for the CUDA C/C++ programming language and the main algorithms include sort, stream compaction, scan, prefix-sum, and parallel reduction[3].

3. CUBLAS

CUBLAS is a library of CUDA basic linear algebra subroutines. It is a part of NVIDIA cudatoolkit and works with NVIDIA CUDA 4.0 or later. It supports Linux, Windows, and Mac OS X operating systems. It has an interface for the CUDA C/C++ programming language. It has all 152 standard basic linear algebra subroutines (BLAS). These routines include 3 levels of BLAS operations: level-1 is called BLAS1 and has functions for scalar and vector operations,

BLAS2 perform matrix-vector operations, and BLAS3 performs matrix-matrix operations. All three levels of functions perform min, max, sum, copy, dot product, norm (Euclidean norm of the vector), scal, swap, multiplication, and rank calculation operations[6].

4. MAGMA

MAGMA is a library for matrix algebra on GPU and multicore architectures. It is an ongoing open source project managed by University of Tennessee (UT). The newest version, as of the time of this writing, is MAGMA 1.4 Beta 2, which was released in June 2013. It works with NVIDIA GPUs and supports operating system platforms such as Linux, Windows, and Mac OS X. Its programming interface includes CUDA C/C++, Fortran, Matlab, Python, and OpenCl. MAGMA includes a CPU and GPU interface for BLAS routines. The CPU interface takes input and produce result in CPU memory, and GPU interface takes input and produce result in GPU memory. It has 80+ hybrid algorithms and total of 320+ routines for basic linear algebra routines[13].

	Thrust	CUDPP	cuBLAS	MAGMA
Description	C++ like Standard Template library	CUDA Data Parallel Primitives Library	CUDA Basic Linear Algebra Subroutines	Matrix Algebra on GPU and Multicore architectures
Availability / Latest Version / date	Included in nVIDIA CUDAtoolkit / Thrust v1.7 / July 2013	Open source (New BSD License)/ CUDPP 2.0 / August 2011	Included in nVIDIA CUDAtoolkit / CUDA 5.0 / October 2012	Open Source(UT)/ MAGMA 1.4 Beta2 / June 2013
Requirement / Operating System (OS) support / Programming Language support	nVIDIA CUDA 4.0 or better / Linux,Windows, Mac OS X / CUDA C/C++, OpenMP, TBB (Threading Building Blocks, Intel)	nVIDIA CUDA 3.0 or better/ Linux,Windows, Mac OS X / CUDA, C/C++	nVIDIA CUDA 4.0 or better / Linux,Windows, Mac OS X / CUDA, C/C++	nVIDIA CUDA(support for NVIDIA Kepler GPUs)/ Linux,Windows, Mac OSX/ C/C++,Fortran, Matlab,Python, OpenCl
Main Subroutines	Scan, search, search by key, count, merge, reorder, prefix-sum, set, sort, transform	Sort, stream compaction, scan, prefix-sum, parallel reduction	min, max, sum, copy, dot product, norm(Euclidean norm of the vector), scal, swap, multiplication, rank	80+ hybrid algorithms (total of 320+ routines), linear and least squares solvers, and all of basic linear algebra subroutines

Table 1. GPU Accelerated Libraries comparison

II. METHOD

CUDA and the C programming language were used to write a program that generates a statistical summary of simulation output. The statistical summary includes sum, mean, and Standard deviation of the 64 *.csv output files each for 4 types of simulation data: Monthly, Daily, Hourly, and 15-minute resolution. The EnergyPlus simulation engine runs and stores the *.csv files in RAMDisk. The program created for this project then reads one file at a time into a matrix. The data is then transferred to the GPU in order to calculate the running sum, mean,

number of elements in the data set, and sum of squares of difference between the data and the mean. After processing all 64 files of a type, it calculates the variance by dividing the sum of squares by the number of elements. The standard deviation is computed by taking the square root of the variance. The program then generates output *.csv files with statistical summaries and stores compressed files in the output directory provided by the workflow.

The algorithm used for statistical metrics calculation on GPU is as follow[11]:

1. Number of elements =0;
2. Sum =0;
3. Mean =0;
4. M2 =0;
5. For each new data in the file
    a. Updated sum = sum so far + new data;
    b. Updated number =number so far +1;
    c. Delta = (new data – mean so far);
    d. Updated Mean = mean so far + (new data – mean so far) / updated number;
    e. Updated M2 = M2 so far + delta * (new data – updated mean);
6. Variance = M2/(total number);
7. Standard deviation = sqrt(Variance);

Inside the GPU function thread working in parallel evaluates each statistical metric for each cell.

We calculated performance metrics such as time, effective bandwidth and computational throughput of the GPU function for statistical analysis. The formula used to calculate these GPU metrics are:

Effective Bandwidth[4] = $(R_B + W_B) / (t * 10^9)$

where R_B is the number of bytes read per kernel, and

W_B is the number of bytes written per kernel

Computational throughput[4] (GFLOP/s) =

number of floating point operation in the functions * number of elements / (GPU time in seconds * 10^9)

III. RESULTS

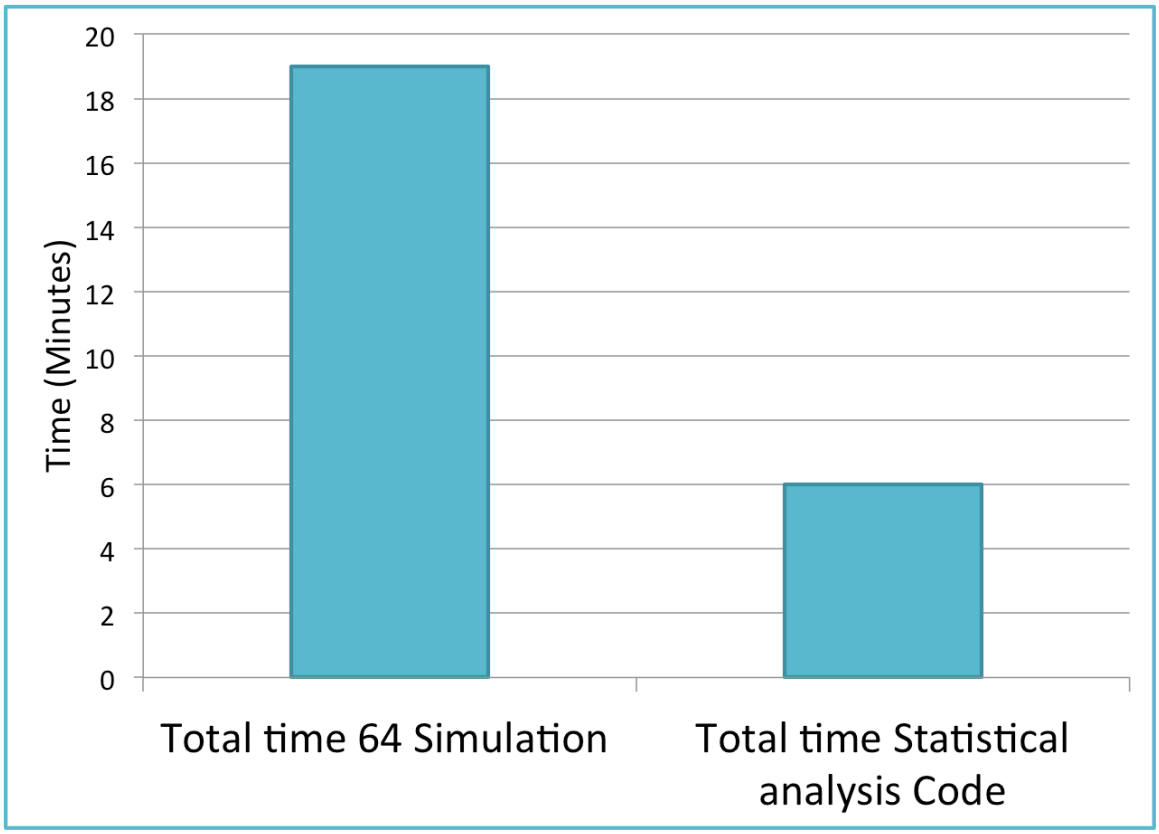

Figure 3. Time for simulation run with GPU statistical analysis

It takes ~19 minutes to complete all 64 simulations running on 16 processors in parallel. The GPU statistical analysis takes ~6 minutes to generate the statistical summary of 256 simulation output files using one processor on one node.

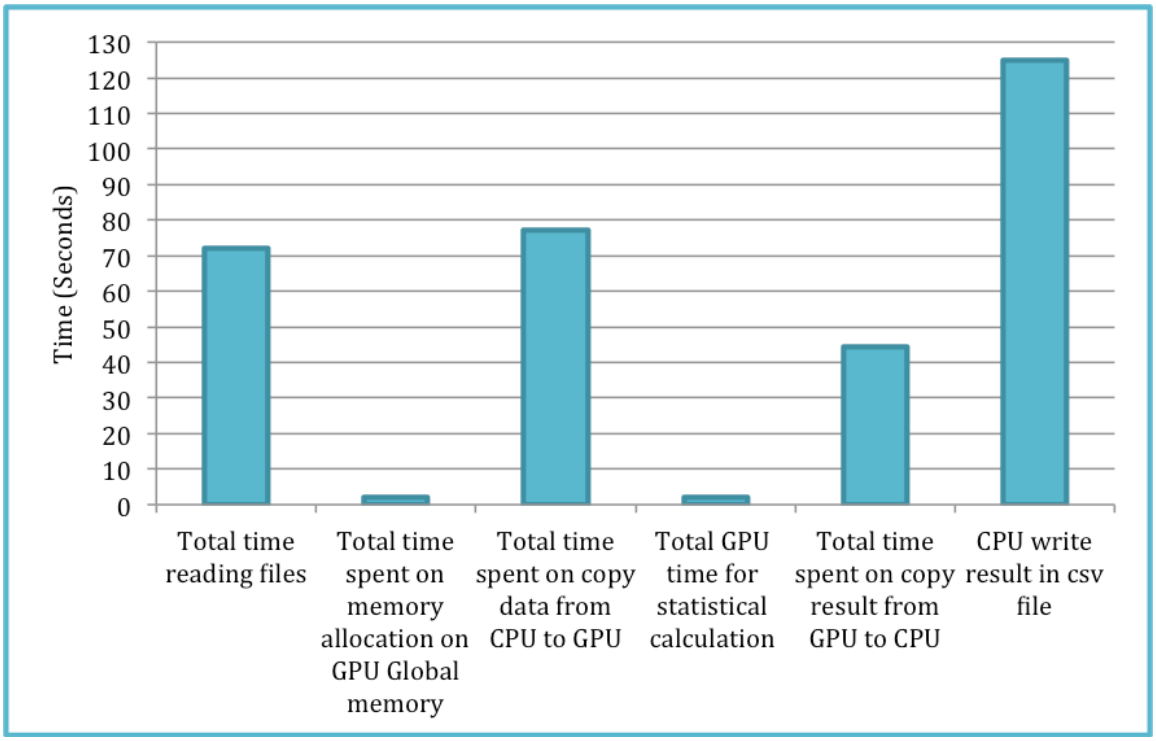

Figure 4**.** GPU Statistical Analysis Time Division. As can be seen in Figure 3, the time division of 6 minutes shows that it takes GPU analysis program to generate the statistical summary of all 256 simulation output files.

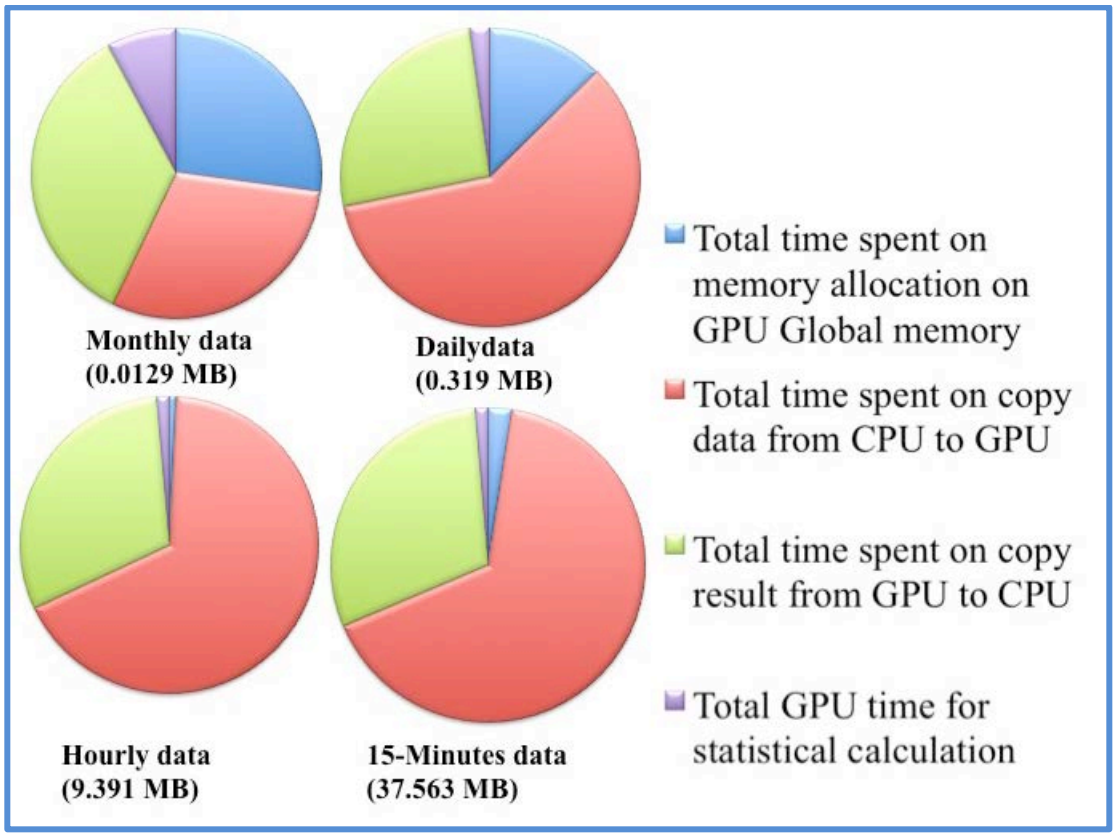

Figure 5. GPU time division for four data types

Figure 4 displays the GPU time division for processing each individual file type. As the data size increases more time is spent transferring data from CPU to GPU and from GPU to CPU. Therefore, a more efficient way of data storing on GPU such as use of shared memory or texture memory would help in reducing the time of the GPU analysis program.

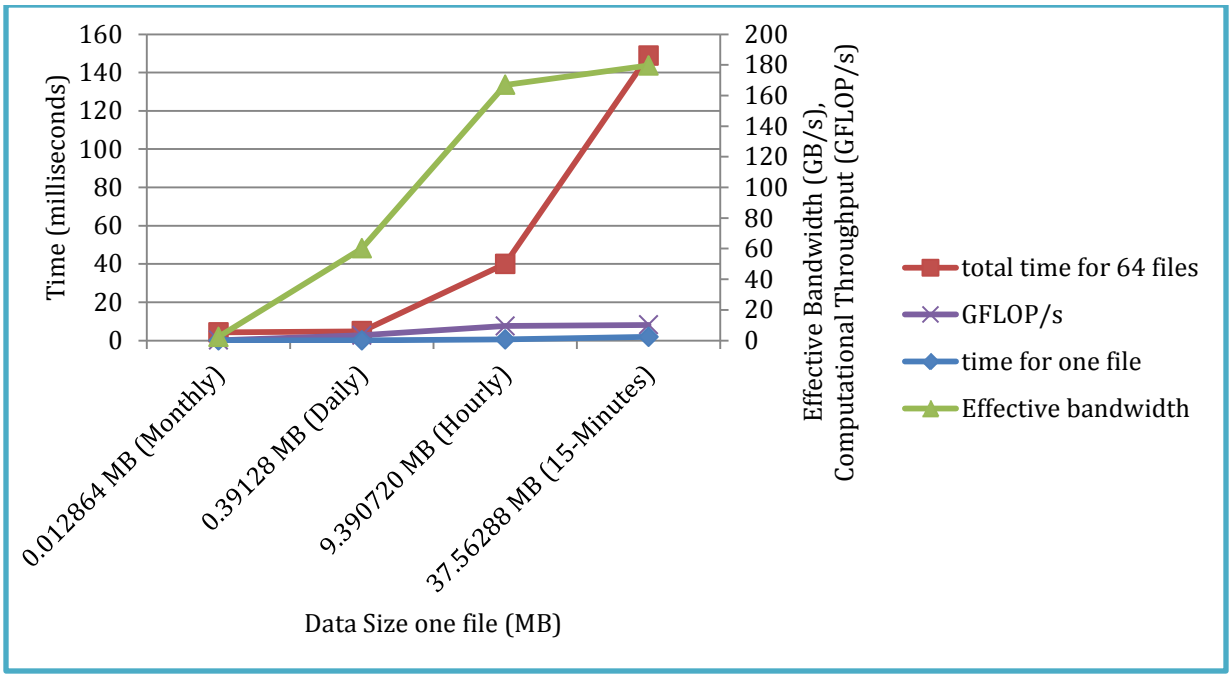

Figure 6. GPU Performance Metrics for Statistical Summary Generation. Theoretical Maximum Bandwidth = 250 GB/s. Peak Computational Throughput (double precision) = 1.31 TFlop/s

IV. CONCLUSION

An accelerated Autotune approach for calibration of building energy model will reduce the cost of making energy efficient building. The EnergyPlus simulations workflow is being speedup by using Message Parsing Interface (MPI) to run multiple simulations in parallel. The MPI first distributes the simulation process on the multiple nodes and then further on all 16 processors on each node. The workflow developed in this project for the statistical analysis of uses CUDA C GPU programming method to speed up the analysis process. The developed workflow generates the statistical summaries of the simulation data. The size of the simulation data produced on one that needs to be stored reduced from ~6 GB to ~100 MB. This statistical analysis could be used for sensitivity analysis of EnergyPlus simulations.

ACKNOWLEDGEMENTS

I would like to acknowledge everyone whose help and support enabled me to complete this SULI internship program.

I am grateful to the U.S. Department of Energy, Office of Science, and Office of Workforce Development for Teachers and Scientists (WDTS) who support Summer Undergraduate Internship (SULI) program and gave me a great opportunity to work with the experienced scientists in the field.

I am deeply thankful to my mentors Dr. Joshua New and Dr. Jibonananda Sanyal whose immense guidance, encouragement, and support allowed me to complete this summer internship project. They provided guidance where needed but encouraged independent solutions to unexpected problems. The skills gained here will definitely help me to become successful in my future career.

I would also like to thank the administration of ORNL, ORAU, ORISE, SULI program, and BTRIC division for their assistance throughout the internship project.

REFERENCES

[1] Edwards, Richard E. (2013). "Automating Large-Scale Simulation Calibration to Real-World Sensor Data." Doctoral Committee: Lynne E. Parker (advisor), Joshua R. New, Michael Berry, Hamparsum Bozdogan, and Husheng Li. A Dissertation presented for the Doctor

of Philosophy Degree in *Archives of The University of Tennessee*, Knoxville, TN, May, 2013.

[2]Garrett, Aaron, New, Joshua R., and Chandler, Theodore (2013). "Evolutionary Tuning of Building Models to Monthly Electrical Consumption." Technical paper DE-13-008. In *Proceedings of the ASHRAE Annual Conference*, Denver, CO, June 22-26, 2013.

[3]Mark Harris "CUDPP Documentation," *CUDPP 2.0 (CUDA Data Parallel Primitives Library,)* http://www.gpgpu.org/static/developer/cudpp/rel/cudpp_1.1/html/ (7 August 2013).

[4]Mark Harris, "How to Implement Performance Metrics in CUDA C/C++," *nVIDIA Developer Zone*, November 2012, https://developer.nvidia.com/content/how-implement-performance-metrics-cuda-cc (6 August 2013)

[5]New, Joshua R., Sanyal, Jibonananda, Bhandari, Mahabir S., Shrestha, Som S. (2012). "Autotune E+ Building Energy Models." In *Proceedings of the 5th National SimBuild of IBPSA-USA*, International Building Performance Simulation Association (IBPSA), Aug. 1-3, 2012.

[6]nVIDIA, "CUBLAS Library," "*User Guide,"* July 2013, http://docs.nvidia.com/cuda/pdf/CUBLAS_Library.pdf (6August 2013)

[7]nVIDIA, "CUDA C Programming Guide," *Design Guide*, July 2013, http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (6 August 2013).

[8]nVIDIA, "Thrust Quick Start Guide," *Thrust*, July 2013, https://developer.nvidia.com/thrust (6 August 2013).

[9]nVIDIA, "What is GPU Computing?," *High Performance Computing*, January 2013, http://www.nvidia.com/object/what-is-gpu-computing.html (6 August 2013).

[10]Oak Ridge Leadership Computing Facility (OLCF), "Titan Overview," *Titan Cray XK7*

https://www.olcf.ornl.gov/computing-resources/titan-cray-xk7/ (6 August 2013)

[11]"Online Algorithm," *Algorithms for calculating* variance, August 2013, http://en.wikipedia.org/wiki/Algorithms_for_calculating_variance (6 August 2013)

[12]Sanyal, Jibonananda and New, Joshua R. (2013). "Supercomputer Assisted Generation of Machine Learning Agents for the Calibration of Building Energy Models." In *Proceedings of the Extreme Science and Engineering Discovery Environment (XSEDE13) Conference and selected to be featured in Lightning Talks*, San Diego, CA, July 22-25, 2013.

[13]Tomov S., Nath R., Du P., Dongarra J., "MAGMA Users' Guide," *MAGMA Users Documentation*, November 2009, http://icl.cs.utk.edu/projectsfiles/magma/docs/magma-v02.pdf (6 August 2013).