An Algorithm to Improve the Execution Time and Speedup for Complex Application

Rajesh Tiwari[1], Dr. Manisha Sharma[2], Dr. Kamal K. Mehta[3]

[1]*ResearchScholarBhilai Institute of Technology, Durg (C.G.), India*
[2]*Prof. Bhilai Institute of Technology, Durg (C.G.), India*
[3]*Prof. O. P. Jindal University Raigarh, (C.G.), India*
[1]*raj_tiwari_in@yahoo.com*

Abstract— **Modern graphics processing units (GPUs) are being widely adopted as application accelerators in HPC owing to their massive floating point compute power that can be leveraged by data parallel algorithms. Consequently, need for virtualization of GPU resources has grown rapidly to provide on demand efficient resource sharing. To solve the computation problem for large data , the necessity for high-performance calculation is growing day by day. Few common application where high-performance computing is used are Weather Forecasting, Quantum Physics, Climate Research, Heat Distribution Problem etc.. An architectural framework has been proposed by NVIDIA to join the power of GPUs with CPUs to improve the execution time and speedup. GPUs were previously used only for Graphics Application like Computer games, Multimedia and graphics but now GPU has been used for high-performance computation work. Finally the results of execution time and speedup with CPUand GPU are compared and find that the GPU gives the better result for bulk data.**

Index Terms—CPU, GPU, Shared Memory, SIMD.

1. INTRODUCTION

Parallel processing is the method of processing program instructions by dividing them into multiple small segments and executes that segments on multiple processors this results the minimum execution time. In the older version (Sequential) of computers, only one program can executed at a time. To solve the complex problem the sequential technique is not used, so the new technique to solve such problem is parallel technique. There are two types of program, one is computation intensive and the other is I/O intensive program. A computation intensive program consider only computation time and I/O intensive program consider only the time spend during the input and output. The interleaved execution of both (computation intensive and I/O intensive) programs together allowed in parallel processing. When the computer system starts an I/O operation, the system is in waiting state till the operation complete. During this time , the compute intensive program starts execution and the utilized the waiting time of the system. This cause in reduction of execution time and improve in speedup.

Heat distribution problem is widely used in mathematical modeling of various processes, phenomena, and systems. Heat distribution problem is the basis of many scientific and engineering calculations few of them are Computational Mathematics, Physics etc. . One of the fundamental building block for scientific computing is the

heat distribution problem and it is one of the most important approaches to understanding parallel programming in GPU [1][2].

The concurrent use of more than one processor to execute a program is an example of SIMD (single instruction stream and multiple data stream) process [3]. Generally, the parallel processing makes a program to execute quicker because of more CPUs are running [4] parallel. In practice, it is a lot difficult to divide a program in such a way that separate CPUs can execute different portions of the program without interfering with each other.

Heat Distribution Problem [5] is a problem where an area has known temperatures along each of its edges as shown in fig..1.1. Divide area into fine mesh of points, $h_{i,j}$as in equation (i). Temperature at an inside point taken to be average of temperatures of four neighboring points. Convenient to describe edges by points.Temperature of each point by iterating the equation:

$$\mathbf{h_{i,j}} = \frac{\mathbf{h_{i-1,j}+h_{i+1,j}+h_{i,j-1}+h_{i,j+1}}}{\mathbf{4}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \textbf{(i)}$$

$(0 < i < n, 0 < j < n)$ for a fixed number of iterations or until the difference between iterations less than some very small amount.

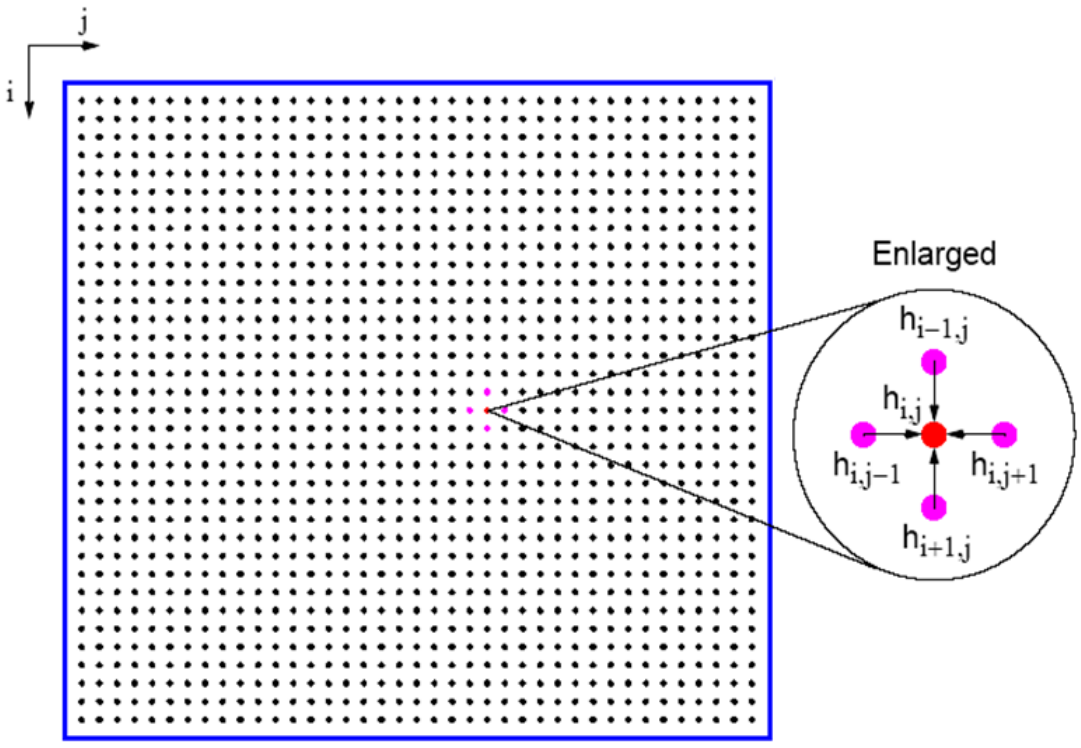

Fig. 1.1 : Heat Distribution Problem

A parallel computation engine is used in GPUs to carries out the complex computational problem in less time than it would have if same problem would have been executing on a single CPU[6][7]. GPUs have been previously utilized mainly for playing games or the application where large graphics resolutions are required. Now

GPU stepped into the fields that need high-performance computation. Fields such as Medical Image, Weather forecasting, and System of linear equations are some fields, where the systems require the high-performance computation to use the possible power of GPUs by which system solve the existing and current problems.

CUDA is a library provided by NVIDIA, it provides extended functionalities in C language by adding CUDA specific functions. This paper shows the different optimization techniques of heat distribution using CPU, heat distribution on GPU using Shared memory which increases floating portion for optimizing a N*N size metal plate.

Compute Unified Device Architecture is a library provided by NVIDIA to execute processes in parallel manner [8]. This is an application programming interface (API) to help communication between device and user. There are CUDA specific functions or methods defined which meant to run on CUDA library only. These are used along with C and C++ programming language. To convert a single processor specific program into CUDA capable programs the programmer needs to modify it accordingly. The CUDA program is generally divided into two parts: the main program executes in the CPU, whereas the parallel portion of the program is executed in GPU. This GPU part is called by the main program and data is sent to GPU for execution where the instructions are executed on the given data, after the calculation result is sent back to CPU [9].

GPU (Graphics Processing Unit) was primarily developed to fulfill the need of algorithms used in computer graphics. It has hundreds of cores which are able to execute multiple threads simultaneously. Later it was proposed that this technology can be useful for non-graphic process also if one can divide a single process into multiple threads and distribute them to multiple processors, the overall computation time can be reduced drastically. There are several types of memory present in the GPU [14][15][16] like device memory, shared memory, constant cache, texture cache, and registers [10][11][12][13]. To manipulate data in this memory and to use the multiple cores to their programmers must write the CUDA programs very carefully.

2. PROBLEM IDENTIFICATION

The main resources of a computer system are memory and processor. Memory and processor both plays an important role in high-performance computing, when large amount of data sets used as input. These data sets requires large amount of memory . A single system is not able to fulfill the memory requirements. So multiprocessor or multicomputer systems are used. Multiprocessor system uses the concepts of shared memory and multicomputer uses the concepts of distributed memory (Non – shared).

When large amount of data sets used as input, calculation was not done in proper way, it takes garbage value. The main reason of the problem is cache storage organization and defect caused by mapping of elements of matrix on to single cache set instead of using the entire cache set. Over all degrade the performances of machine which increased execution time instead actual execution time. This paper presents a heat distribution problem on the GPU and CPU and comparing the execution time with the use of NVIDIA GeForce GT 525M machine.

3. SPECIFICATION

The testing platform requirements are as follows:-

Hardware specification:-

Intel (R) core (TM) i3-2350M CPU @ 2.30 GHz

System memory:- 4GB(installed memory)

Testing platform specifications:-

Operating System: -	windows7 (32-bit operating system)
Software used: -	Microsoft visual studio 2010
Language used: -	CUDA C
Version of CUDA: -	CUDA Toolkit 6.5

4. METHODOLOGY

In this paper the temperature of each points are stored in matrix form in file and this file has been used as input. The algorithm [17] is as given below: -

Step 1. Input the file of heat distribution of different size to CPU

Step 2. The time recorder starts (t_s)

Step 3. CPU sends heat distribution data to GPU

Step 4. GPU receives the data and operation

Step 5. GPU distributes these among threads with scatter function.

Step 6. GPU performs their operations in parallel

Step 7. GPU collects the processed data with gather function.

Step 8. GPU returns the processed data to CPU

Step 9. CPU collects the processed data and produce the Result

Step 10. The time recorder stops (t_e)

The total elapsed time includes the computation time (t_{comp}) as well as total communication time (t_{comm}) which is calculated by equation (ii) as:

$$\text{Elapsed time} = t_e - t_s \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\text{(ii)}$$

Here communication time is the time to spend in communicationof data and computation time is the time to spend in calculation of data.

5. RESULT and CONCLUSION

Execution Times

Calculating the speedup requires the collection of the execution times during the operation of the experiments as mentioned above. The user is allowed to run the benchmarks and make a note of the execution times. The user specifies varying types and amounts of the input and is responsible for executing the benchmark initially. Finally, since the speedup studied is fixed size relative speedup, the execution times thus obtained by the user are noted for fixed and notable size of the input data and studied for the features required. The execution times are calculated for the serial implementation of the dwarf, the parallel implementation of the dwarf on one node and the execution times of upto8-nodes. The execution time is typically measured in seconds, and is calculated from the moment the parallel threads are created and after the moment all the threads are finished. For the serial implementation the unit of

measurement is the same and the execution time is calculated from the moment the computational task begins and the moment the task ends.

Table 5.1: Execution times of Laplace Heat Distribution

Programming Model	Execution Time			
	1- node	2- nodes	4- nodes	8- nodes
MPI	105.3247	66.3870	37.1451	19.8941
OpenMP	116.4337	69.7985	38.4351	20.3857

The execution times for the Laplace heat distribution for the two models are shown in the table 5.1. The execution times of the Laplace heat distribution are gathered by computing the algorithm on the matrices(plate size) of size 2048 with a tolerance value of 0.02 and a relaxation factor of 0.5. And of all the models the MPI model is observed to have better execution times which is 19.894 on 8-nodes and the size of the chunk partition is specified by the programmer.

Speedup

Table 5.2: Speedup of Laplace Heat Distribution

Programming Model	Execution Time		
	2- nodes	4- nodes	8- nodes
MPI	1.5865	2.8354	5.2943
OpenMP	1.6681	3.0294	5.7115

Table 5.2 depicts the speedup achieved by above programming models for Laplace heat distribution. Problems based on local communications however incurs less overhead also the speedup achieved is greater than those with global communications. The speedup achieved in most of the cases is slightly less than or equal to the linear because of the less overheads involved due to communications between threads. As the number of threads increases the overhead associated with the communications also increases by very less amount.

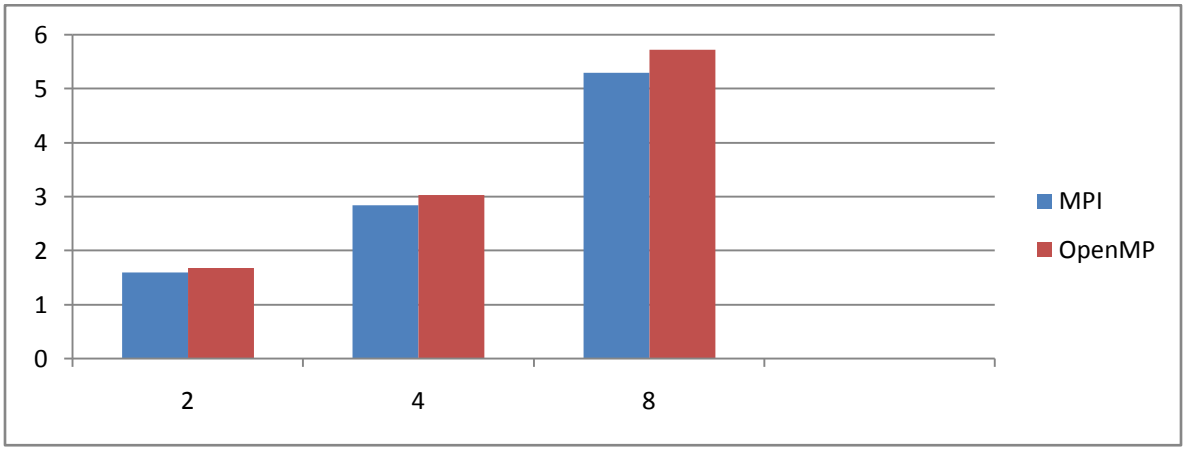

Fig. 5.1 : Comparison Chart between MPI &OpenMP

The speedup for the Laplace heat distribution is achieved high speedup for OpenMP, which is shown in the fig. 5.1 which is 5.7115 on 8-nodes.

REFERENCES

[1] L. Djinevski, S.Arsenovski,S. Ristov and M.Gusev, "Performance Drawbacks for Matrix Multiplication using Set Associative Cache in GPU devices," in MIPRO 2013, 20-24 May 2013, pp. 193-198.

[2] D. J. Sooknanan, A. Joshi, "GPU Computing Using CUDA in the Deployment ofSmart Grids," in SAI Computing Conference 2016, July 13-15, IEEE 2016, pp. 1260-1266.

[3] J. Sartori and R. Kumar, "Branch and Data Herding: Reducing Control and Memory Divergencefor Error-Tolerant GPU Applications," IEEE Transactions on Multimedia, Vol. 15, No. 2, February 2013, pp. 279-290.

[4] M. Shah and V. Patel, "An Efficient Sparse Matrix Multiplication for the skewed matrix on GPU," in 14th International Conference on High-Performance Computing and Communications, IEEE 2014, pp. 1301-1306.

[5] B. Wilkinson and M. Allen," Parallel Programming," 2nd edition of Pearson Education.

[6] X. Cui, Y. Chen, and H. Mei, "Improving Performance of Matrix Multiplication and FFT on GPU," in 15th International Conference on Parallel and Distributed Systems 2009, IEEE 2009, pp. 42-48.

[7] M. Shah, "Sparse Matrix Sparse Vector Multiplication -A Novel Approach," in 44th International Conference on Parallel Processing Workshops 2015, IEEE 2015, pp. 67-73.

[8] S.W. Ha and T.D. Han, "A Scalable Work-Efficient and Depth-Optimal Parallel Scan for the GPGPU Environment," IEEE Transactions on Parallel and Distributed Systems, Vol. 24, No. 12, December 2013, pp. 2324-2333.

[9] NVIDIA. https://developer.nvidia.com.

[10] S. H. Lo, C. R. Lee, Q. L. Kao, I. H. Chung, and Y. C. Chung," Improving GPU Memory Performance with Artificial Barrier Synchronization," IEEE Transactions on Parallel and Distributed Systems, 2013.

[11] M. Salim, A. O. Akkirman, M. Hidayetoglu, and L. Gurel, "Comparative Benchmarking: Matrix Multiplication on a Multicore Coprocessor and a GPU," in IEEE 2015, pp. 38-39.

[12] N. Q. Anh, R. Fan, Y. Wen, "Reducing Vector I/O for Faster GPU Sparse Matrix-Vector Multiplication," in 29th International Parallel and Distributed Processing Symposium, 2015, IEEE 2015, pp. 1043-1052.

[13] R.Eberhardtand M.Hoemmen,"Optimization of Block Sparse Matrix-Vector Multiplication on Shared-Memory Parallel Architectures," in International Parallel and Distributed Processing Symposium Workshops 2016, IEEE 2016, pp. 663-672.

[14] W. Liu and B. Vinter,"An Efficient GPU General Sparse Matrix-Matrix Multiplication for Irregular Data," in 28th International Parallel & Distributed Processing Symposium 2014, IEEE 2014, pp. 370-381.

[15] X. Zha and S.Sahni, "GPU-to-GPU and Host-to-Host Multi pattern String Matching on a GPU," IEEE Transaction On Computers, Vol. 62, No. 6, June 2013, pp. 1156-1169.

[16] A. Barberis, G. Danese, F. Leporati, A. Plaza, and E. Torti, "Real-Time Implementation of the Vertex Component Analysis Algorithm on GPUs," IEEE Geoscience and Remote Sensing Letters, Vol. 10, No. 2, March 2013, pp. 251-255.

[17] R. Tiwari, M. Sharma and K. K. Mehta, "Dynamic Load Balancing in Parallel Processing using MPI Environment to Improve System Performance " International journal of Advance Research in Computer Science and Software Engineering , Vol. 5, Issue 6, pp 730 – 734 , June 2015.