IOSR Journal of Engineering (IOSRJEN) **www.iosrjen.org**
ISSN (e): 2250-3021, ISSN (p): 2278-8719
Vol. 04, Issue 01 (January. 2014), ||V3|| PP 56-59

Performance Evaluation Of Matrix Multiplication Using OpenMP For Single Dual and Multi-Core Machines

Yogesh Singh Rathore [1], Dharminder Kumar [2]
[1] *Department of Computer Science &Engineering, Mewar University, Gangrar, Chittorgarh, Rajasthan, INDIA.*
[2] *Department of Computer Science &Engineering Guru Jambheshwar University of Science & Technology, Hisar, Haryana, INDIA.*

Abstract : **-** Matrix Multiplication is one of the most commonly used algorithm in many applications including operations on Relations in Relational Database System. In this paper we study and evaluate the execution time of matrix multiplication on a single, dual and multi-core processor with same set of processors having OpenMP(Open Multi-Processing) libraries for C-Language. OpenMP is a very well known standard that exploits parallelism in shared memory architecture. The evaluation is based on execution of the algorithm uses single thread for computation whereas the one with OpenMP uses multi-threads.

Keywords: **-** *Matrix Multiplication, Single Core Machine, OpenMP, Multi-Threads.*

I. INTRODUCTION

A Core of a machine is basically a unit that reads and executes a program instruction of a fixed length called Word of that machine. It is generally of 8, 16, 32, 64 or variable length chunks of bits. The instructions of a program tells the operation to be performed, it can be reading of data from key board, displaying of results on display device, fetching of data from file or writing the output in a file. A single core machine executes one instruction at a time but with the use of OpenMP[1] a multi-core[2][3] machine's processors of Multi-Core Architectures[14] can be used for parallel processing with the help of threads concept. OpenMP has Directives, Environment variables and Run-Time Environment. However the OpenMP API (Application Programming Interface) does not guarantees the improvement in performance. In this paper an introduction of OpenMP is given, in reference to its declaration and use in the program. Then we will move towards the concept of forks and joins in reference of threads and multithreads for multicore architectures for a given equation of Matrix Multiplication followed by related work. Our main emphasis is on the experiment and the saved in processing the Matrix Multiplication.

II. OPENMP

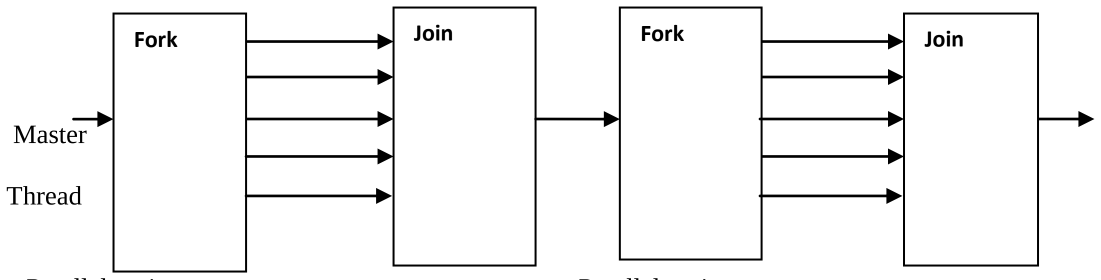
Open MP was born in the 1990s with the objective of bringing a standard to a different directive languages defined by community of various vendors. It supports various characteristics necessary for parallelism in a program. OpenMP is based on the insertion of directives in the sequential source code that give hints to run time library about the existent parallelism in the algorithm in the study. Open- MP is independent of Platform and operating system. The OpenMP pragma annotation expressing the parallel loop "for" used in Matrix Multiplication algorithm is given below:

```
#pragma omp parallel for
```

Version 3.0[4] of OpenMP includes a functional model that fills a gap with regard the ways of expressing the parallelism in the algorithm in study. With the help of new OpenMP directives the programmer can locate the units of independent jobs, leaving the judgment to how and when to execute them to the run-time system. This gives the programmers a way of expressing patterns of concurrency that do not match the work sharing constructs defined in the OpenMP2.5 specification.

III. PARALLEL MATRIX COMPUTATION

Before jumping to Parallelization [5][6][7][8][9] for Matrix-Multiplication directly we must understand the concept of parallel computation in single-core machines. The parallelism in single core machines is achieved with the help of Forks and Joins as shown in figure-III.1. This can spread single threaded application into many sequences of instructions with different capacities.



Parallel region Parallel region

Figure: III.1 Fork Join Model

The term fork-join parallelism refers to a method of specifying parallel execution [13] of a program Where as the program flow diverges (forks) into two or more flows that can be executed concurrently and which comeback together (join) into a single flow when all the parallel works is complete. When two matrices are multiplied, the result will again be a matrix. If the matrix product is somewhat like A * B = C, then the number of A's columns needs to be equal to B's number of rows. General matrices A and B will then look like this:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & & a_{2m} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1r} \\ b_{21} & b_{22} & & b_{2r} \\ \vdots & & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mr} \end{bmatrix}$$

A is a $n * m$ matrix, that means A has n rows and m columns. B is a $m * r$ matrix, so B has m rows and r columns. m is the common thing for both matrices. The result matrix C will become a $n * r$ matrix. Each element of C is defined as the scalar product of a row vector of matrix A and a column vector of B is as follows:

$$A * B = C = \begin{bmatrix} \sum_{i=1}^{m} a_{1i} * b_{i1} & \sum_{i=1}^{m} a_{1i} * b_{i2} & \cdots & \sum_{i=1}^{m} a_{1i} * b_{ir} \\ \sum_{i=1}^{m} a_{2i} * b_{i1} & \sum_{i=1}^{m} a_{2i} * b_{i2} & & \sum_{i=1}^{m} a_{1i} * b_{ir} \\ \vdots & & \ddots & \vdots \\ \sum_{i=1}^{m} a_{ni} * b_{i1} & \sum_{i=1}^{m} a_{ni} * b_{i2} & \cdots & \sum_{i=1}^{m} a_{ni} * b_{ir} \end{bmatrix}$$

IV. RELATED WORK

A general purpose programming language designed for multi-threaded parallel programming is Clik[10]. Clik is a task based programming. In Clik, the programmer is responsible for exposing the application parallelism [11], identifying the sections of the code that can be executed in parallel. Tasks are started with the keyword "spawn" reserve word and sync reserve word is used to wait until all previously spawned have been completed. Clik supports recursively at task level and does not support automatic data dependence among the tasks. So data dependence has to be controlled by the programmer with the help of sync reserve word. At runtime the scheduler decides how to actually divide the work between processors. In recursive tasks Clik uses stealing approach for execution of tasks. Click supports only parallel tasks and Clicks++ supports parallel loops. OpenMP execution is opposite of that initially it supports parallel loops and the last version 3.0 supports parallel tasks.

V. EXPERIMENT

In this paper we will run matrix multiplication [12] algorithm for various sizes ranging from 500*500 to 3000*3000 on single, dual and multi-core processors and evaluate the mean of the run time in seconds for every set of various size of algorithms, Then we will implement the same process on the same sets of the sizes (as above) for matrix multiplication with OpenMP. Again we will try to read the run-time in seconds. The sample example for logic of matrix multiplication code is given below:

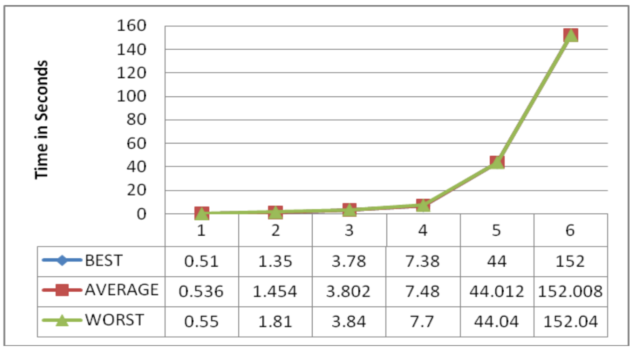
```
   I.  #include<omp.h>
  II.  #include<time.h>
 III.  {
  IV.  start=Clock();                        // start time
   V.  #pragma omp for schedule (static, chunk);   //reading of matrix A
  VI.  For(i=1;i<=n;i++)
```

```
VII.    For(j=1;j<=n;j++)
VIII.   Cin>>A[i][j];
IX.     #pragma omp for schedule (static, chunk);    //reading of matrix B
X.      For (i=1;i<=n;i++)
XI.     For (j=1;j<=n;j++)
XII.    Cin>>B[i][j];
XIII.   #pragma omp for schedule (static, chunk);    //multiplication of matrix A and matrix B
XIV.    For (i=1;i<=n;i++)
XV.     For (j=1;j<=n;j++)
XVI.    For (k=1;k<=n;k++)
XVII.   C[i][j]=C[i][j]+A[i][j]* B[i][j];
XVIII.  end=clock();                                 //end time
XIX.    T=end –start;
XX.     }
```

In the above code time function has been used for getting time, for which time header file has been added as a preprocessor directive. Similarly omp for chunk's reading and multiplication is being used.

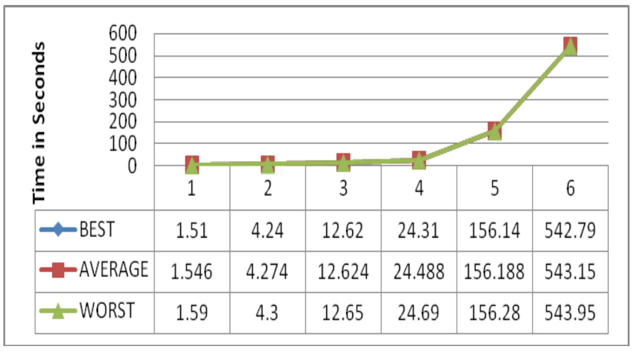
VI. PERFORMANCE RESULT

In the following fig.: V.1 and V.2 best, average and worst time has been taken along x-axis and time in seconds along y-axis. The use of OpenMP accelerates the computation of matrix multiplication as shown in fig.: V.1 and V.2, if compared. In the fig.: V.3 computation time for running the algorithm with and without OpenMP along x-axis and time, in seconds, for computation of algorithm has been taken along y-axis. Ultimately the time saved is considerably large larger size of matrix.



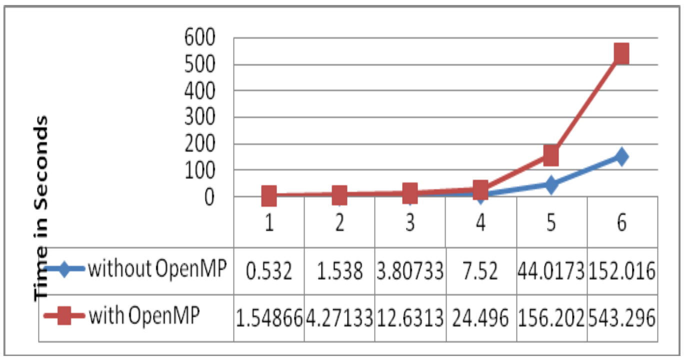
	1	2	3	4	5	6
BEST	0.51	1.35	3.78	7.38	44	152
AVERAGE	0.536	1.454	3.802	7.48	44.012	152.008
WORST	0.55	1.81	3.84	7.7	44.04	152.04

Figure: VI.1 Matrix Multiplication with OpenMP



	1	2	3	4	5	6
BEST	1.51	4.24	12.62	24.31	156.14	542.79
AVERAGE	1.546	4.274	12.624	24.488	156.188	543.15
WORST	1.59	4.3	12.65	24.69	156.28	543.95

Figure: VI.2 Matrix Multiplication without OpenMP



	1	2	3	4	5	6
◆ without OpenMP	0.532	1.538	3.80733	7.52	44.0173	152.016
■ with OpenMP	1.54866	4.27133	12.6313	24.496	156.202	543.296

Figure: VI.3 Matrix Multiplication with OpenMP

VII. EVALUATION AND ANALYSIS BASED ON PERFORMANCE RESULTS

With the use of OpenMP you can set any number of threads to an algorithm depending upon the no of cores, the machine is having. If you set a single thread to be on a multi-core machine still it performs better than the serial computation on a single core machine. If your number of threads is equal no of cores it is the best parallelism achieved. If you increase the number of threads more than the number of cores in a machine still the performance of execution of algorithm shows better results, but the computation is done in the combination of serial and parallel.

VIII. CONCLUSIONS AND FUTURE WORK

If the input size of an algorithm in study increases the difference between the time taken for execution of an algorithm with OpenMP and without OpenMP increases. The best speed up for the algorithm is 3.573 times with OpenMP that is near to 3.6 times, for various sizes of inputs. Optimization of OpenMP can be done and better results can be obtained. As far as the matrix multiplication is concerned some loop optimization techniques with OpenMP can be added to get optimized output or good speedup.

REFERENCES

[1] COMPUNITY. The community of OpenMP users, researchers, tool developers and provider website. http://www.compunity.org/, 2006.

[2] L. Hammond, B. A. Nayfeh, and K. Olukotun,"A single-chip multiprocessor," Computer, Vol.30, no.9, pp. 79-85, 1997.

[3] A. Jerraya, H. Tenhunen, and W. Wolf,"Guest editors' introduction: Multiprocessor systems-on-chip," Computer, vol38, no. 7. pp. 36-40, July 2005.

[4] E. Ayguade. N. Copty. A. Duran, J. Hoeflinger, Y. Lin, and G. Zhang. A proposal for task parallelism in OpenMP. In proceedings of the 3rd international workshop on OpenMP, June 2006.

[5] H. Zhong, S. A. Lieberman, and S. A. Mahlke, "Extending multicore Architecture to exploit hybrid parallelism in single thread applications," hpca, vol. o, pp. 25-36, 2007.

[6] W. J. Dally and S. Lacy, "VLSI architecture: Past, present, and future," in ARVLSI '99: Proceedings of the 20th Anniversary Conference on Advanced research in VLSI. Washington, DC, USA: IEEE Computer Society, 1999, p.232.

[7] S. Kumar, C. J. Hughes, and A. Nguyen, "Carbon: architectural support for fine-grained parallelism on chip multiprocessors," in ISCA '07: Proceedings of the 34th annual international symposium on computer architecture. New York, NY, USA: ACM, 2007, pp. 162-173.

[8] K. Bousias. N. Hasasneh, and C. Jesshope,"instruction level parallelism through micro threading- a scalable approach to chip multiprocessors," Comput. J., vol. 49, no. 2, pp. 211-233, 2006.

[9] A. Rodrigues. R. Murphy, P. Kogge, and K Underwood, "Characterising a new class of threads in scientific applications for high end super computers," in ICS '04: proceedings of the 18th annual international conference on super computing. New York, NY, USA: ACM,2004, pp. 164-174.

[10] M. Frigo, C. E. Leiserson, and K. H. Randall. The implementation of the Clik-5 multithreaded language. SIGPLAN Notices, 33(5):212-223, 1998.

[11] A. Buluc, J.R. Gilbert," Challenges and advances in parallel sparce Matrix-Matrix Multiplication", in proceedings of37th international conference on parallel processing ICPP'08, Portland, Sept 2008.

[12] P. Alonso, R. Reddy, A. Lastovetsky, "Experimental study of six different implementations of parallel Matrix Multiplications on Hetrogenous Computational Clusters of Multi-core processors" in proceedings of parallel, Distributed and Network Based processing(PDP), Pisa, Feb. 2010.

[13] S. Ohshima, K. Kise, T. Katagiri and T. Yuba, "Parallel Processing of Matrix Multiplication in a CPU and GPU Heterogeneous Environment", High Performance Computing for Computational Sciences-VECPAR, 2006.

[14] P. F. Gorder, Multi-Core processors for science and engineering, computing in Science and Engg. 9(2)(2007)3-7.doi:http:dx.doi.org.10.1109/MCSE. 2007.35