IOSR Journal of Engineering (IOSRJEN) **www.iosrjen.org**
ISSN (e): 2250-3021, ISSN (p): 2278-8719
Vol. 04, Issue 03 (March. 2014), ||V1|| PP 19-22

Performance Evaluation of Matrix Multiplication Using Mix Mode Optimization Techniques And Open MP For Multi-Core Processors

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 application areas like Sonar Systems, Relational Database Management System and other applications like algebra etc. Matrix Multiplication is quite difficult when it tends to infinity. In this paper we study and evaluate the execution time of simple matrix multiplication and optimized matrix multiplication with OpenMP on multi-core machine with same set of data values. Optimization techniques reduces space requirement and ensures fast execution. OpenMP is a very well known standard that exploits parallelism in shared memory architecture. The evaluation is based on simple execution of the algorithm that uses single thread for computation whereas the one with optimization techniques and OpenMP with multi-threads.

Keywords: *- Matrix Multiplication, Optimization techniques, Open MP, 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 Multi-Core [2][3] machine supports parallelism. An Optimized code with OpenMP[1] increases the performance. 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 Optimizing techniques and OpenMP is given, in reference to its declaration and use in the program. Then we will move towards various optimizing techniques for Matrix Multiplication in reference of Multi-core Architectures [14], followed by related work. Our main concern is the experiment and the time saved in processing of the Matrix Multiplication.

II. OPTIMIZATION TECHNIQUES

In this section we will introduce an effective combination of optimization techniques for Matrix Multiplication. In the study we will work on the combination of induction variable substitution and loop interchange which has been used and proven beneficial for in our case. Induction variable substitution works on the variables whose values forms arithmetic progression and is generally expressed as a function of loop index. It reduces number of operations inside the loop or in other words we can say it reduces the memory access without any extra cost. Secondly loop interchange is used for automatic parallelization of loops. Loop interchange is not beneficial in the case of serial loops.

III. 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. OpenMP is independent of Platform and Operating System. OpenMP is not the part of C language. The OpenMP pragma annotation expressing the parallel loop ”for” used in Matrix Multiplication algorithm is given as #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.

IV. PARALLEL MATRIX COMPUTATION

Before jumping to Parallelization [5][6][7][8][9] for Matrix-Multiplication directly, We must understand the mechanism of matrix multiplication in brief. When two matrices are multiplied, the result will again be a matrix. If the matrix product is somewhat like C=A * B, then the number of A's columns needs to be equal to B's number of rows. 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 subscript 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.

V. 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. To some extent Optimizing techniques individually are being used for speed ups execution and reducing memory requirements for the different tasks of very small sizes only, on simple machines.

VI. EXPERIMENT

In this paper we will run matrix multiplication [12] algorithm for various sizes ranging from 500*500 to 2000*2000 on 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 for matrix multiplication with OpenMP and mixmode of various optimizing techniques that includes loop interchange, for all combination of the loops used in the algorithm under study and induction variable substitution. We are having 3 loops, so combination for the 3 loops may be given by 3!=3*2*1 that is equals to 6. If IJK are the loops then combinations should be IJK, IKJ, JIK, JKI, KIJ and KIJ. Again we will try to read the run-time in seconds for all combinations. The time header file of C is very helpful for computation of execution time for sample code. The core loop computation for matrix multiplication code is given below:

```
#include<iostream>
#include<time.h>
#include<omp.h>
using namespace std;
int main(){…….

clock_t t1,t2,t;
……….
t1=clock();
#pragma omp for schedule(static,chunk);
for(i=0;i<n;i++){
for(j=0;j<n;j++){
c[i][j]=0;
for(k=0;k<n;k++){
c[i][j]+=a[i][k]*b[k][j];
   }
   }
   }
t2=clock();
t=t2-t1;
cout<<"time taken "<<t;
return 0;
}
```

In the above code clock function has been used for getting time, for which time header file has been added as a preprocessor directive. The time t received will be divided by the suitable value to get the time in seconds. Similarly OpenMP for chunk's reading and multiplication is being used.

VII. PERFORMANCE RESULTS:-

In the fig.: VII.1 execution time for algorithm after implementing all possible combinations of loop interchange and substitution has been studied. Execution time has been taken in seconds and the various sizes have been also taken. The Fig.:VII.2 compares the average of the values taken for all combinations with the result of simple execution of Matrix Multiplication.

	For loop IJK	For loop IJK	For loop JIK	For loop JKI	For loop KIJ	For loop KJI
500	1.17	1.34	1.18	1.21	1.16	1.11
700	3.45	3.63	3.59	3.63	3.69	3.36
1000	11.99	12.01	11.93	12.08	11.97	11.93
1200	18.69	19.73	19.42	19.41	19.10	19.42
2000	87.55	88.89	86.67	86.53	87.86	88.69

Figure VII.1

	Av. of optum.	For loop IJK(plane)
500	**1.20**	1.54
700	**3.56**	4.27
1000	**11.99**	12.6
1200	**19.30**	24.4
2000	**87.70**	156

Figure VII.2

The use of OpenMP with the mixmode optimizing technique accelerates the computation of matrix multiplication as shown in fig.: VII.3. In the fig.: VII.3 computation time for running the algorithm with and without OpenMP plus optimizing techniques are taken along y-axis and various sizes of Matrix is taken along X-axis. It is very clear that the time saved in execution is large and it increases with the increase in the size of matrix. Ultimately we can say that for a matrix of 2000 X 2000 we can save half up to half of the time by using OpenMP and mixmode optimizing techniques. 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.

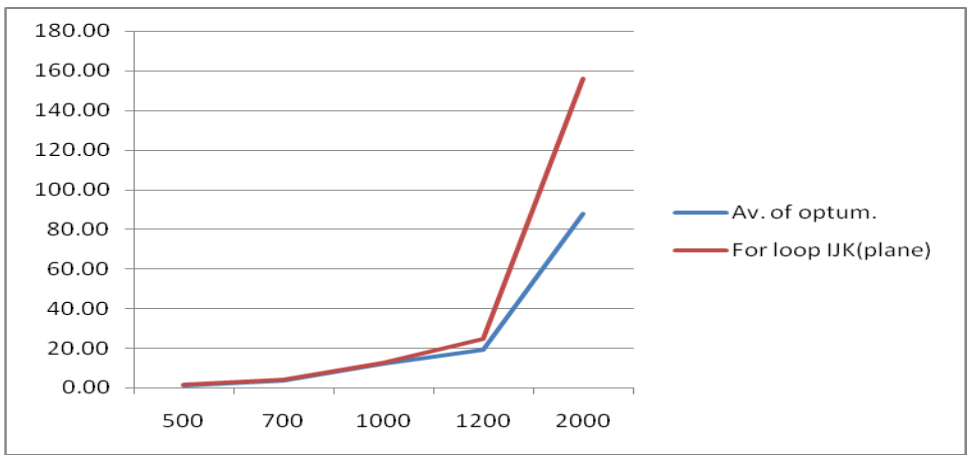

Figure VII.3

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 plus optimizing techniques and without OpenMP increases. As far as the matrix multiplication is concerned some new hybrid technique should be developed for better results that can exploit the system at L1, L2 and L3 cache level.

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 p[arallel 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