International Journal of Advanced Computer Technology (IJACT)

ISSN:2319-7900

Parallel Algorithm Performance Analysis using OpenMP for Multicore Machines

Mustafa B[1], Waseem Ahmed.[2]
[1]Department of CSE,BIT,Mangalore, mbasthik@gmail.com
[2]Department of CSE, HKBKCE,Bangalore, waseem.pace@gmail.com

Abstract: Parallel programming is a language that allows us to explicitly indicate how different portions of the computation may be executed concurrently by different processors. Multi-core CPU's supports the parallel programming that fully exploits the performance and efficient processing of multiple tasks simultaneously. Unfortunately, writing parallel code is more complex than writing serial code. This is why the programmers used many techniques and programming models such as MPI, CUDA and OpenMP to adapt more performance. OpenMP programming model helps in creating multithreaded applications for the existing sequential programs. This paper presents the performance potential of the parallel programming model over sequential programming model using OpenMP. The experimental results show that a significant performance is achieved on multi-core system using parallel algorithm.

Keywords: multicore, openMP, parallel programming, speedup.

1. INTRODUCTION

Parallel computing is now considered as standard way for computational scientists and engineers to solve problems in areas as diverse as galactic evolution, climate modeling, aircraft design, and molecular dynamics. Parallel computer has roughly classified as Multi-Computer and Multiprocessor. Multi-core technology means having more than one core inside a single chip. This opens a way to the parallel computation, where multiple parts of a program are executed in parallel at same time. Thread-level parallelism could be a well-known strategy to improve processor performance. So, this results in multithreaded processors.

Multi-core offers explicit support for executing multiple threads in parallel and thus reduces the idle time. The factor motivated the design of parallel algorithm for multi-core system is the performance. The performance of parallel algorithm is sensitive to number of cores available in the system. One of the parameter to measure performance is execution time. In this paper we have considered parallelizing the Mergesort algorithm and the Floyd's algorithm. Rest of the paper is organized as follows: Section 2 giving detailed description of related work, section 3 explains programming in OpenMP, section 4 is about detailed implementation details of algorithms, section 5 focuses on result and analysis.

1.1 Parallel Computing

The main fact to 'parallelise'the program code, is to reduce the amount of time it takes to run. Consider the time it takes for a program to run (T) to be the number of instructions to be executed (I) multiplied by the average time it takes to complete the computation on each instruction (t_{av})

$$T = I \times t_{av}.$$

In this case, it will take a serial program approximately time T to run. If you wanted to decrease the run time for this program without changing the code, you would need to increase the speed of the processor doing the calculations. However, it is not viable to continue increasing the processor speed indefinitely because the power required to run the processor is also increasing. With the increase in power used, there is an equivalent increase in the amount of heat generated by the processor which is much harder for the heat sink to remove at a reasonable speed[11] . As a result of this, we have reached an era where the speeds of processors is not increasing significantly but the number of processors and cores included in a computer is increasing instead. For a parallel program, it will be possible to execute many of these instructions simultaneously. So, in an ideal world, the time to complete the program (Tp) will be the total time to execute all of the instructions in serial (T) divided by the number of processors you are using (Np)

$$Tp = \frac{T}{Np}$$

In reality, programs are rarely able to be run entirely in parallel with some sections still needing to be run in series. Consequently, the real time (Tr) to run a

ISSN:2319-7900

program in parallel will be somewhere in between Tp and T, ie Tp < Tr < T.

In the 1960's, Gene Amdahl determined the potential speed up of a parallel program, now known as Amdahl's Law. This law states that the maximum speed-up of the program is limited by the fraction of the code that can be parallelised

$$S + P = 1$$

$$\Rightarrow SU = \frac{1}{S}.$$

The serial fraction of the program (S) plus the parallel fraction of the program (P) are always equal to one. The speed-up (SU) is a factor of the original sequential runtime (T). So, if only 50% of the program can be parallelised, the remaining 50% is sequential and the speedup is

$$SU = \frac{1}{0.5} = 2,$$

ie, the code will run twice as fast.

The number of processors performing the parallel fraction of the work can be introduced into the equation and then the speed-up becomes

$$SU = \frac{1}{\frac{P}{N} + S},$$

where S and P are the serial and parallel fractions respectively and N is the number of processors.

2. RELATED WORK

There are few works done to parallelize the serial algorithms using different parallel architectures. In [1], author has computed the value of Pi and solved linear equations to improve performance by reducing execution time and also shown the execution time of both serial and parallel algorithm for computation of Pi value and for the solution of system of linear equations.

Performance analysis of matrix multiplication algorithms[2] done to prove parallel implementations performs better than the sequential algorithm that is executed on Intel Pentium CPU G630 which has dual cores and also on Intel i7 processor. Also shows that, as the number of cores increases, the computation time taken by an algorithm is reduced.

In [3], author has presented the execution time of both serial and parallel execution of naive and Strassen's algorithm for matrix multiplication. They conclude that though Strassen's algorithm consumes a lot more memory than serial algorithm, but the performance is much better than the traditional matrix multiplication algorithm due to its reduced operations.

Shared memory systems can also vary widely but they all have the ability for each processor to access all memory as a global address space. An example of a shared memory system is a single desktop. The advantage of using a shared memory parallelization is that it is relatively simple to make existing serial codes parallel. There are a few disadvantages which include the possibility that multiple cores and processors accessing the shared memory simultaneously could cause a bottleneck which will slow down a program. Also, adding more processors does not increase the amount of memory available which could be a problem.

3. IMPLEMENTATION DETAILS

One of the useful things about OpenMP is that it allows the users the option of using the same source code both with OpenMP compliant compilers and normal compilers. This is achieved by making the OpenMP directives and commands hidden to regular compilers.

The OpenMP standard was formulated as an API for writing portable, multithreaded applications. It started as Fortran-based standard, but later grew to include C and C++. The OpenMP programming model provides set of compiler pragmas. Many loops can be threaded by just inserting a single loop above right to the loop. Implementation of OpenMP determines how many threads to use and how best to manage it. Instead of adding lots of code for creating parallel program, the programmer just need to tell the OpenMP which loop should be threaded. In ordered to understand the concept of OpenMP, it is necessary to know the concept of parallel programming. Parallel processing is done by more than one processor in parallel computing systems. Some the advantage of OpenMP includes: good performance, portable, requires very little programming effort and allows the program to be parallelized incrementally.

3.1 Objective

We have implemented parallel algorithm using OpenMP with the hope that they will run faster than their sequential counterparts. A sequential program, executing on a single processor can only perform one computation at a time, whereas the parallel program executed in parallel and divides up perfectly among the multi-processors. Main Objective of this approach is to increase the performance (speedup) which is inversely proportional to execution time.

3.2 Overview of Proposed Work

We describe the Mergesort and Floyd's algorithm parallely by using OpenMP to achive the good performance by reducing execution time on multi-core. We tested the execution time of the algorithm on dual core and the quad core and measured their performance as shown in the Figure 1.

ISSN:2319-7900

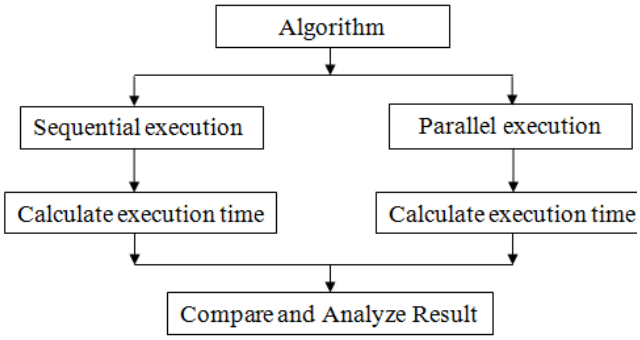

Figure 1: Overview of Proposed work

3.3Working Modules

Being able to analyze the execution time exhibited by a parallel program can help to understand the barriers to higher performance and predict how much improvement can be realized by increasing the number of processors. The parallel algorithms described below.

Merge Sort Algorithm

In merge sort there are three steps to be done, i.e. Divide, Conquer and Combine. Initially divide the given array consisting of n elements into two parts of n/2 elements each. Sort the left part and right part of the array recursively. Merge the sorted left part and right part to get a single sorted array. In this algorithm, the three steps-divide, conquer and combine are done in parallel.

Algorithm: *MergeSort(a,low,high)*

```
if(low<high)
        mid←(low+high)/2
#pragma omp parallel
        MergeSort(a,low,mid)
#pragma omp parallel
        MergeSort(a,mid+1,high,)
#pragma omp parallel
        Merge(a,low,mid,high)
```

Floyd's Algorithm

The Floyds algorithm is the solution for all-pairs-shortest path-problem. Here the shortest distance from all nodes to all other nodes has to find. The all-pairs-shortest-path-problem is to determine a matrix D such that D[i,j] contains the shortest distance from i to j. In this algorithm, each thread has given a chunk size which specifies number of iterations that thread executes.

Algorithm: *Floyd(n,cost,D)*

```
#pragma omp parallel for private(i,j,k),shared(cost)
        for k←0 to n-1 do
            for i←0 to n-1 do
                for j←0 to n-1 do
                    D[i,j]=min(D[i,j],D[i,k]+D[k,j])
                end for
            end for
        end for
    return

//copying D[i,j] to the cost[i,j]
        for i←0 to n-1 do
            for j←0 to n-1 do
                D[i,j]=cost[i,j]
            end for
        end for
```

4. EXPERIMENTAL RESULT

There are two algorithms and each has two versions: sequential and parallel. Both the programs are executed on intel@i5-3317U CPU which is a quad core machine. We analyzed the result and derived the conclusion. In both the experiment the execution times of both the sequential and parallel algorithms have been recorded to measure the performance (speedup) of parallel algorithm against sequential.

The data presented in Table 1 represents the execution time taken by the sequential and parallel programs for Mergesort algorithm and the data presented in Table 2 represents the execution time taken by the sequential and parallel programs for Floyds algorithm.

No. of Elements	Sequential Program in quad core (sec.)	Parallel Program in quad core (sec.)	Performance/ Speed up
500	0.001000	0.001055	0.947867
1000	0.003000	0.001301	2.305918
2000	0.004000	0.001503	2.661343
3000	0.006000	0.001800	3.333333
4000	0.007500	0.002805	2.673796
5000	0.009000	0.003505	2.567760
6000	0.011000	0.005001	2.199560
7000	0.013000	0.007006	1.855552
8000	0.015000	0.008001	1.874765
9000	0.017000	0.010001	1.699830
10000	0.018000	0.012010	1.498751

Table 1: Execution time for Merge Sort Algorithm

ISSN:2319-7900

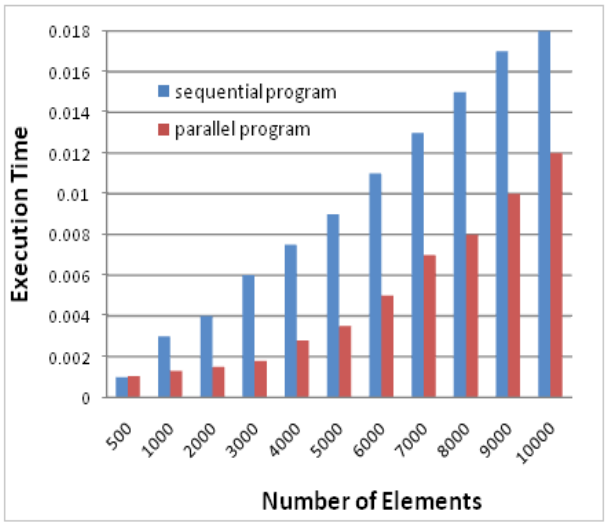

Figure 2: Execution time of sequential and parallel algorithm of Merge Sort.

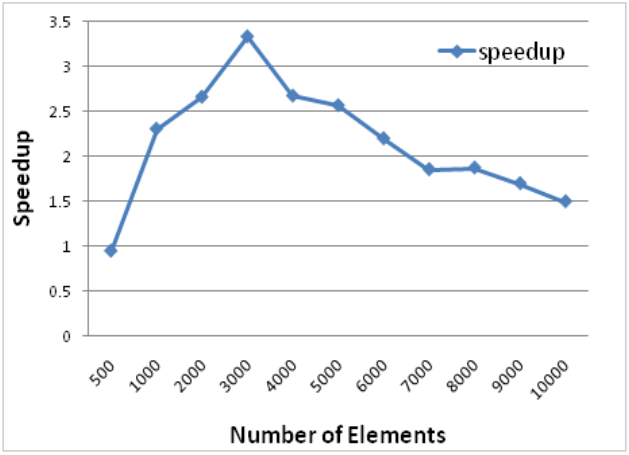

Figure 3: Speedup Graph for Merge Sort Algorithm

500	0.990000	0.543147	1.853422
600	1.710000	0.946513	1.806631
700	2.730000	1.483408	1.840356
800	4.080000	2.244393	1.817863
900	5.820000	3.179151	1.830677

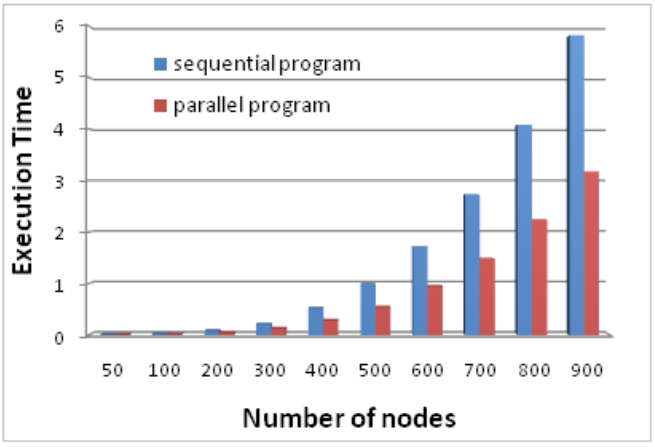

Figure 4: Execution time of sequential and parallel algorithm of Floyd's.

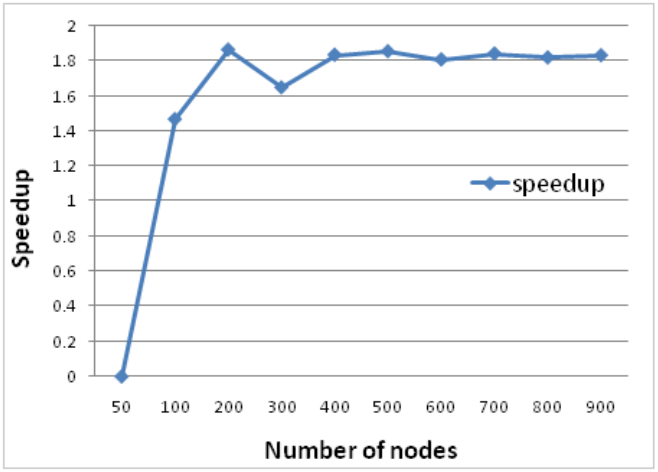

Figure 5: Speedup Graph for Floyd's Algorithm

Table 2: Execution time for Floyd's algorithm

No of nodes	Sequential Program in quad core (sec.)	Parallel Program in quad core (sec.)	Performance / Speed up
50	0.000000	0.005470	0
100	0.020000	0.013637	1.466598
200	0.080000	0.042952	1.862544
300	0.210000	0.127461	1.647562
400	0.520000	0.283614	1.833477

CONCLUSION

In this work we discussed how OpenMP programming techniques are beneficial to multi-core system. From our study we arrive at the following conclusions: (1) Performance is increased by parallelizing serial algorithm using OpenMP. (2) For multi-core system OpenMP provides a lot of performance increase and parallelization can be done with careful small changes. (3) The parallel algorithm is approximately twice faster than the sequential and the speedup is linear.

ISSN:2319-7900

The algorithms with small data set give good performance when executed by a sequential programming. But as data set increases performance of parallel execution increases over the sequential and it gives best results than sequential execution. Speedup is achieved by the ratio of sequential execution time and parallel execution time.

REFERENCES

[1] Sanjay Kumar Sharma, Dr. Kusum Gupta, "Performance Analysis of Parallel Algorithms on Multi-core System using OpenMP Programming Approaches", *International Journal of Computer Science, Engineering and Information Technology (IJCSEIT), Vol.2, No.5,* October 2012.

[2] Sheela Kathavate, N.K.Srikanth, "Efficient of Parallel Algorithms on Multi Core Systems Using OpenMP*", International Journal of Advanced Research in Computer and Communication EngineeringVol. 3, Issue 10*, October 2014.

[3]Vijayalakshmi Saravanan, Mohan Radhakrishnan, A.S.Basavesh, and D.P.Kothari, "A Comparative Study on Performance Benefits of Multi-core CPUs using OpenMP," *IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 1, No 2,* January 2012.

[4] Sodan, A.C, Machina J, Deshmeh A, Macnaughton K, "Parallelism via multithreaded and Multicore CPUs*", IEEE Computer Society,Volume: 43, issue: 3, pp. 24-32,* Mar. 2010.

[5] Pranav Kulkarni, Sumith Pathare, "Performance Analysis of Parallel Algorithms over Sequential using OpenMP", *IOSOR Journal of Computer Science, Volume6,* March-April 2014.

[6] Sushil Kumar Sah,andDinesh Naik, "Parallelizing Doolittle Algorithm Using TBB," IEEE 2014.

[7]DolittleDecomposition http://mathfaculty.fullerton.edu/mathews/2003/CholeskyMod

[8] Sanjay Kumar Sharma, Dr. Kusum Gupta, "Performance Analysis of Parallel Algorithms on Multi-core System using OpenMP Programming Approaches", International Journal of Computer Science, Engineering and Information Technology (IJCSEIT), Vol.2, No.5, October 2012.

[9] Matrix decomposition, Wikimedia Foundation,Inc, http://en.wikipedia.org/wiki/Matrix_decomposition.

[10] General-purpose computing on graphics processing units, Wiki-media Foundation, Inc, [11] An Introduction to Parallel Programming with OpenMP, Alina Kiessling

Biography

[1]**Mustafa Basthikodi** was born in Mangalore, India, in 1979. He received the B.E. degree in Computer Science and Engineering from the Mysore University, Mysore, India, in 2001, and the M.E. degree in Computer Science and Engineering from the Bangalore University, Bangalore in 2008. Currently Pursuing PhD in High Performance Computing and Embedded Systems from Visvesvaraya Technological University (VTU), Belgaum.

In 2001, he joined the Department of Computer Science & Engineering, PACE, Mangalore, as a Lecturer, and worked till 2006. From 2006 to 2008, He worked as Senior Lecturer in Department of Computer Science & Engineering in SJBIT, Bangalore. In 2008, He joined IBM as Senior Software Engineer and worked till 2010. Since 2010, He is working as Associate Professor and Head, Department of Computer Science & Engineering, in BIT, Mangalore.

He has published in various National and International Conferences. He has received few best technical paper awards and also Best performer award in industry. He has also worked as Technical Programme committee member for the various conferences. He is a Life Member and resource person for Computer Society of India. His subjects of Interest include High Performance Computing & Embedded Systems, Green & Cloud Computing, Compiler construction tools & technologies.

[2]**Dr.Waseem Ahmed** is currently a Professor in the Department of CSE at HKBK College of Engineering, Bangalore. Prior to this he has been served at different capacities in academic/work environments in the USA, UAE, Malaysia, Australia and India. He obtained his BE from RVCE, Bangalore, MS from the University of Houston, USA and PhD from the Curtin University of Technology, Perth, Western Australia. He has published extensively in various reputed International Journals and Conferences. He is a reviewer for various IEEE/ACM Transactions and magazines. His current research interests include heterogeneous computing in HPC and embedded Systems. He is a member of the IEEE.