A Parallel Algorithm to Process Harmonic Progression Series Using OpenMP

[1,]Rinki Kaur, [2,]Sanjay Kumar, [3,]V. K. Patle
[1,2,3,]*School of Studies in Computer science & IT*
Pt. Ravishankar Shukla University, Raipur (Chhattisgarh) 492010 India

***Abstract*— In Present days, multicore systems have become popular as it provides parallelism and hence less delay, but multicore systems provide only hardware parallelism. In order to achieve best result we should use software parallelism also.**

To achieve software parallelism there are many programming models like OpenMP, MPI etc. In this work we have used OpenMP for dividing Harmonic Progression Series into different threads and running on quad core processor. We have found the optimum number of threads for best result.

Keywords*— *Parallelism, Multithreading, Multi-core Processor, OpenMP, Parallel Programming

I. INTRODUCTION

One of the recent innovations in computer engineering has been the development of multi-core processors, which are composed of two or more independent cores in a single physical package. Today, many processors, including digital signal processor (DSP), mobile, graphics, and general purpose

Central processing units (CPUs) have a multi-core design, driven by the demand of higher performance. Major CPU vendors have changed strategy away from increasing the raw clock rate to adding on-chip support for multi-threading by increase in the number of cores. Dual-and quad-core processors are now commonplace [1].

Parallel programming or parallel computing is an alternative towards the traditional serial computing which instead of only allowing a single instruction to be executed once at a time, it simultaneously executes multiple instruction at once using multiple computational resources [2],[3],[4]. High-end computing is one area that sought tremendous amount of processing powers which most computer systems are having difficulties to accomplish. This is where parallel programming plays its role. From solving complex mathematical equations to assisting scientist in

research, parallel programming has proven the world it's worth [2], [3], [4]. Parallel computing on shared memory multi-processors has become an effective method to solve large scale scientific and engineering computational problems. Both MPI and shared memory are available for data communication between processors on shared memory platforms [5]. There are three main models for parallel programming multi-core architectures. These models are the message-passing paradigm (MPI), shared memory programming model, and Partitioned Global Address Space (PGAS) programming model [6], [7]. The shared memory programming model allows a simpler programming of parallel application, as the control of the data location is not required.

OpenMP is the most widely used solution for shared memory programming, as it allows an easy development of parallel applications through compiler directives. Moreover, it is becoming more important as the number of cores per system increases.

The OpenMP Application Programming Interface (API) was developed to enable portable shared memory parallel programming. It aims to support the parallelization of applications from many disciplines. Moreover, its creators intended to provide an approach that was relatively easy to learned as well as apply. The API is designed to permit an incremental approach to parallelizing an existing code, in which portions of a program are parallelized, possibly in successive steps. In this paper performance evaluation of multi-core processor are discussed on the basis of three parameters (scalability, average execution time for each thread in each terms, CPU utilization for each thread in each term) by using OMP, which gives multithreading environment.

This paper aims to analyze the performance of multi-core architecture on the basis of the sum of Harmonic Progression implemented on various thread processing via the parallel programming paradigm. This paper has been organized as follows. In section 2, we briefly describe the multi-core processor and multithreading. In Section 3 the methodology is described in detail. In Section4 elaborates the results and analysis of the finding. Finally, section 5 is the conclusion & future work.

II. MULTI-CORE PROCESSOR AND MULTITHREADING

Multi-core Processor employing multithreading are being extensively used now a days.

A. *Multi –core processor*

A multi-core processor is a single computing component with two or more independent actual central processing units called "cores", which are the units that read and execute program instructions. The instruction are ordinary CPU instructions such as add , move data, and branch, but the multiple cores can run multiple instructions at the same time, increasing overall speed for programs amenable to parallel computing. A dual-core processor is a multi-core processor with two independent cores. A quad-core processor is a multi-core processor with four independent cores, an octo-core processor or octa-core processor contains eight cores, and a deca-core processor contains ten cores [8].

B. *Multithreading*

A thread is a basic unit of execution. A single thread executes a series of application instructions, following a single path of logic through the application. All applications have at least one thread [12]. Multi-Threading is the ability of a CPU to execute several threads apparently at the same time. CPUs are very fast at executing instructions. Modern PCs can execute nearly a billion instructions every second. Instead of running the same program for one second, the CPU will run one program for perhaps a few hundred microseconds then switch to another and run it for a short while and so on.

Multithreading is the approach of using multiple threads of execution to process different operations An operating system is able to take each of these threads and other multiplex them onto the same core or to run them in parallel on multiple cores that may exist. As such, multithreading is a common approach to parallel programming, whereby you can split a larger problem into multiple sub-problems and use multiple threads (i.e. multithreading) to process those sub-problems concurrently.

Multithreading is similar to multitasking, but enables the processing of multiple threads at one time, rather than multiple processes. Multithreading may occur within processes. Multithreading aims to increase utilization of a single core by using thread-level as well as instruction-level parallelism [9].

III. MATERIAL AND METHOD

A. *Material*

A program for the addition of Harmonic progression series is used. If a sequence is in Arithmetic Progression, then the sequence obtained by taking the reciprocal of every term in the sequence forms a Harmonic Progression. These programs are divided into subsection and each subsection is executed by the different threads for different number of terms.

B. *Harmonic Progression Series*

If a sequence is in AP, then the sequence obtained by taking the reciprocal of every term in the sequence forms a Harmonic Progression [10].

- That is if a, b, c, form an AP, then 1/a, 1/b, 1/c … form an HP.
- Let a, b, c form an HP. Then clearly, 1/a, 1/b, 1/c form an AP. Thus,
- Sum of Harmonic Progression Series is based on mathematical equation

$$\text{Sum} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \cdots \ldots \ldots \ldots \ldots \ldots + \frac{1}{n}$$

C. *Hardware and Software*

The Multi-core processor specifications used in this work are quad core and is described in following table.

Component	Description
# of processor core	4(Quad)
Processor	Intel(R)core™i3-3110M CPU @2.40GHz 2.40GHz
RAM	4.00GB RAM
System Type	64 bit

The software required to perform the parallel process are Linux (fedora 15) as a operating system and OpenMP parallel programming model and system monitor for to check the CPU utilization of the processor for performance measure.

D. *Method*

The parallel program for sum of Harmonic Progression series are written in OpenMP programming model. OpenMp is a implementation of multithreading, which allows to create system threads and hereby takes advantage of multi-core- architecture. OpenMP provides the thread level parallelism. OpenMP uses the fork-join model of parallel execution. All OpenMP programs begin as a single process: the **master thread.** A master thread forks a specified number of slave threads and a task is divided among them. The thread then runs concurrently, with the runtime environment allocating threads to different processors.

By default, each thread executes the parallelized section of code independently. Work-sharing construct can be used to divide a task among the threads so that each thread executes its allocated part of the code. Both task parallelism and data parallelism can be achieved using OpenMP in this way.

A parallel program for Harmonic Progression series are divided into two levels:

- According to Number of threads
- According to Number of terms

According to Number of threads: In these levels we use 2,4,8,16,32 threads. For each thread we increase the number of terms as 100, 1000, 10000, 100000. We make a parallel program for each number of threads, in which we use a **#pragma omp parallel** for the thread creation.

#pragma omp parallel is used to fork additional threads to carry out the work enclosed in the construct in parallel. The original process will be denoted as **master thread** with thread ID 0.

According to Number of terms: In these levels we have used the 100,1000,10000,100000 as a number of terms for each threads. For 2 threads we divide 100, 1000, 1000,100000 equally into 2 sections. For 4 threads we divide 100,1000,10000,100000 equally into 4 sections and also for the other threads. For this purpose we have used a work sharing construct in a program.

#pragam omp sections: Work sharing construct clauses sections assigning consecutive but independent code blocks to different threads. It contains a set of sections and informs that they should execute in parallel.

#pragam omp section: This informs that the code block that should be executed by a single thread and creation of section depends on the number of threads.

Syntax for section:

```
#pragma omp sections [clause ...]      newline
{#pragma omp section   newline
structured_block
#pragma omp section   newline

structured_block}
```

These levels provide a data level parallelism. This parallelism is achieved by splitting H.P. series into 100,1000,10000,100000 terms which corresponds to 2, 4, 6,8,16 and 32 threads used.

#pragma omp critical: A block in which only one thread may enter at a time.

a) Algorithm

The Parallel Algorithm for sum of Harmonic Progression Series is describing in this section.

STEP1: Initialize the variables
STEP2: omp_set_num_threads ()
// [function that set the maximum thread count at run time.]
STEP3: Set the initial time using omp_get_wtime () function.
STEP4: #pragma omp parallel
// [syntax of the openmp]
// [create a team of threads that run the code block in parallel]
STEP5: #pragma omp sections
// [contains a set of sections and informs that they should execute in parallel]
STEP6: #pragma omp section
// [This informs that the code block that should be executed by a single thread and creation of section depend on the number of threads]
STEP7: #pragma omp critical
// [A block in which only one thread may enter at a time]
STEP8: Assign the initial value to SUM variable
STEP9: Initializes the counter
STEP10: Calculates the TERM and adds with SUM
STEP11: Increment the counter by 1
STEP12: omp_get_thread_num()
// [Runtime function to return a Thread –ID]
STEP13: [END of Parallel Region]
STEP14: Prints the value of SUM and Execution time
STEP15: STOP

b) Performance Metrics

There are different kinds of parameters for evaluating the performance of a system .These performance parameters are the execution time that are evaluate for each threads and CPU utilization .

Scalability of parallel algorithms: Increasing number of processor decreases efficiency with fixed problem size and increasing the amount of computation per processor increases efficiency with fixed machine size. It should possible to keep the efficiency fixed by increasing both the size of the problem and the number of processor simultaneously .Two Scalability metrics are used

1) Number of threads
2) Number of terms

Execution Time: Execution time is used to estimate the parallel execution time for each thread to well utilize the processor in solving the problem. For calculating Average Execution time for each thread we run parallel program of Harmonic progression Series three times for each number of terms.

CPU Utilization: CPU Utilization show the utilized percentage of the each core or CPU of the system**.**CPU Utilization is needed to monitor via the system monitor to determine whether or not it met the criteria intended.

IV. RESULT AND DISCUSSION

The Parallel results are discussed based on Average Execution Time and CPU Utilization factor using the algorithm in performing the algorithm.

A. Average Execution Time

After the execution of parallel program for the different number of threads and for the different number of terms we analyze the average execution time for each thread which is given in following tables.

Number of threads	Average Execution time(in second)
2	0.788442
4	0.513684
8	0.7770706
16	0.77374
32	0.7807707

1) The average execution time for each thread in 100 numbers of terms.

Table 1: Analyze data for 100 terms

Number of threads	Average Execution time (in second)
2	0.006704
4	0.001704
8	0.0020944
16	0.00366976
32	0.0037854

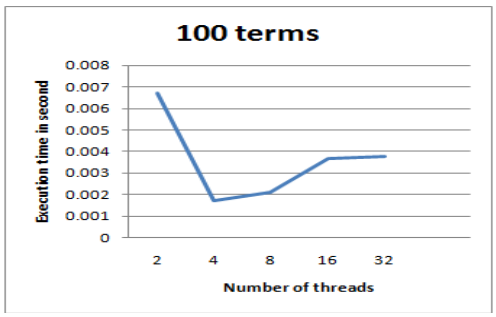

Figure 1: Average Execution time for 100 terms

2) The average execution time for each threads in 1000 number of terms.

Table 2: Analyze data for 1000 terms

Number of threads	Average Execution time (in second)
2	0.050578
4	0.0870763
8	0.034989
16	0.0357324
32	0.692207

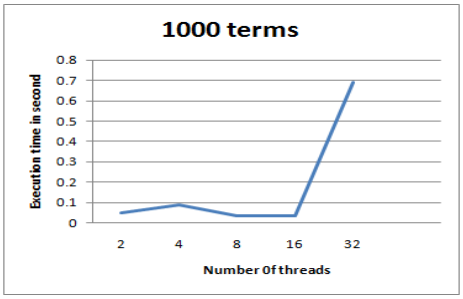

Figure 2: Average Execution time for 1000 terms

3) The average execution time for each threads in 10000 number of term.

Table 3: Analyze data for 10000 terms

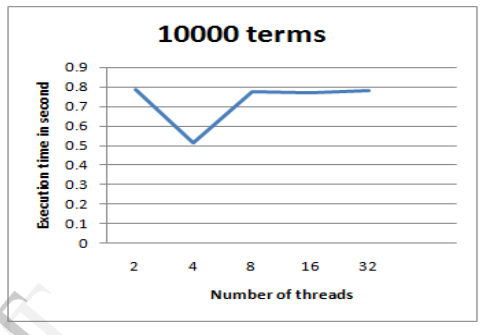

Figure 3: Average execution time for 10000 terms

4) The average execution time for each threads in 100000 number of term.

Table 4: Analyze data for 100000 terms

Number of threads	Average Execution time(in second)
2	8.27771
4	5.4542087
8	8.0280277
16	8.086431
32	8.156544

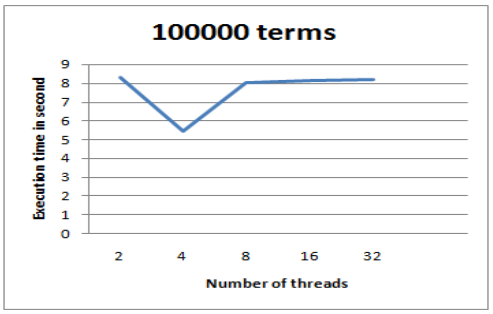

Figure 4: Average execution time for 100000 terms

Here we have implemented the addition of Harmonic Progression series 1+1/2+1/3+…….+1/100 in parallel environment at different number of threads likes 2, 4,8,16 and 32 threads. It is observe that in starting when we

increase number of thread from 2 to 4 execution time decreases, but as we increase number of threads above 4execution time starts increasing, instead of decreasing. 32 number of threads time taken is even more than the time taken by 2 threads. After observation by the graph it is found that:

- 4 is the optimum number of threads for 100, 10000,100000 cycles.
- 16 is the optimum number of threads for 1000 cycles.

5) The optimum number of threads for each number of terms

Number of terms	Optimum number of threads
100	**4**
1000	**4**
10000	**16**
100000	**4**

Here optimum number of threads for parallel execution of H.P. series is equal to the number of cores in the system where they implemented. Since we are running on quad core machine that is there are 4 cores available. At one time only 4 threads run parallel. Upon increasing no of thread beyond the certain limits, communication overhead occurs and therefore execution time increase that means we should parallelize a problem optimally**.**

B. CPU Utilization

The main objective is to fully utilize the CPU to its utmost potential. Therefore, CPU Utilization needs to be monitored via the system monitor to determine whether or not it met the criteria intended. Following figure describes the CPU utilization factor when executing the parallel algorithm on quad core processor respectively.

1) CPU Utilization in 1000000 terms for 2 threads

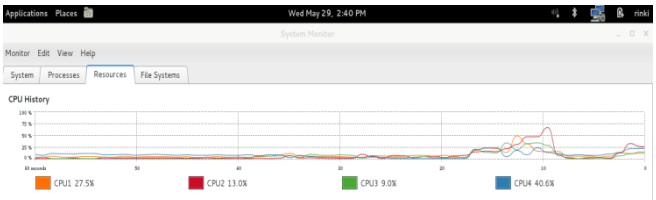

Figure 1: CPU Utilization in percentage for 2 threads

2) CPU Utilization in 1000000 terms for 4 threads

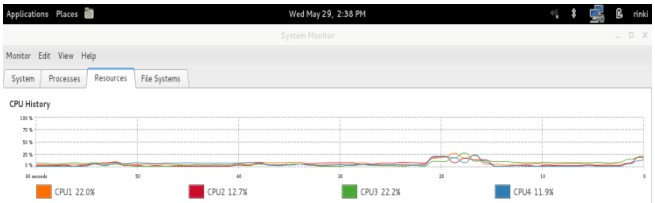

Figure 2: CPU Utilization in percentage for 4 threads

3) CPU Utilization in 1000000 terms for 8 threads

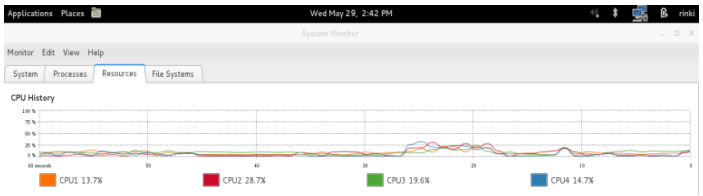

Figure 3: CPU Utilization in percentage for 8 threads

4) CPU Utilization in 1000000 terms for 16 threads

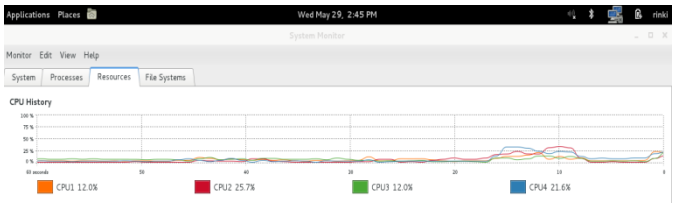

Figure 4: CPU Utilization in percentage for 16 threads

5) CPU Utilization in 1000000 terms for 32 threads

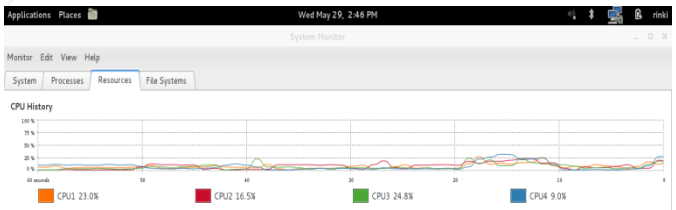

Figure 5: CPU Utilization in percentage for 32 threads

CPU utilization has shown the utilized percentage of each core or CPU of the system on the execution of our problem.

In parallel execution each core of the system are equally utilized .We can say that multicore system is fully utilized by parallel execution of the program.

C. Scalability

In this paper, we divide H.P. series into number of threads like 2, 4, 6, 8, 16, and 32 to perform the sum of Harmonic Progression series into different number of terms. It shows the scalability with respect to number of threads and terms.

V. CONCLUSION & FUTURE WORK

In this work effect of parallelization on execution time was studied, addition of harmonic series was done, by using threads. Thread is an independent smallest unit of

processing that can be scheduled by operating system. The problem was divided into increasing number of threads and terms. It was found that upon increasing the number of threads (parallelization) time is increasing. This is probability because of small configuration of micro computers which include desktop; laptop ,another reason may be use of OpenMP which has very limited facility of parallelization. Small time also spends in overhead communication. It may be possible that on mini and mainframe computer along parallel operating system and parallel compiler upon increasing parallelization time may reduce. We conclude that for best performance when number of threads should be equal to the number of cores.

If any programs are running on quad core processor without thread then it is as good as running a program on single core computer. To make use of quad core computer he should used 4 threads in suitable environment like OpenMP or any other equivalent.

VI. FUTURE WORK

The future work can be computation may be made for arithmetic progression series, geometric progression series also. This may be extended to binomial series, sequential series integration series another type of complex series like sine series, cosine series etc.

REFERENCES

[1] Greg Slabaugh, Richard Boyes, Xiaoyun Yang, "**Multicore Image Processing with OpenMP"**

[2] B. Barney, *Introduction to ParallelComputing.* Retrieved from LawrenceLivermore National Laboratory: https://computing.llnl.gov/tutorials/parallel_comp/, 2010..

[3] C.Lin And L.Snyder, "Principles Of Parallel Programming", Pearson International, 2009.

[4] Noor Elaiza Abdul Khalid, Siti Arpah Ahmad, Noorhayati Mohamed Noor,Ahmad Firdaus Ahmad Fadzil, Mohd Nasir Taib, **"Analysis Of Parallel Multicore Performance On Sobel Edge Detector"** Isbn: 978-1-61804-019-0.

[5] Yong Luo, "**Shared Memory Vs. Message Passing: The Comops Benchmark Experiment"** 1060-3425/98 1998 Ieee.

[6] D. A. Mallón, G. L. Taboada, C. Teijeiro, J. Touriño, B. B. Fraguela, A. Gómez, R. Doallo And J. C. Mouriño " Performance Evaluation Of Mpi, Upc And Openmp On Multicore Architectures" Europvm/Mpi Lncs 5759, Springer Berlin Heidelberg Pp.174-184 , 2009.

[7] Alaa Ismail, El-Nashar,"Parallel Performance Of Mpi Sorting Algorithms On Dual–Core Processor Windows-Based Systems" International Journal Of Distributed And Parallel Systems (Ijdps) Vol.2, No.3, May 2011.

[8] Https://En.Wikipedia.Org/Wiki/Multi-Core_Processor.As Retrieve On Date 28/06/13.

[9]Http://Cplus.About.Com/Od/Glossar1/G/Multithreading.Htm. As Retrieve On Date 28/06/13.

[10]Hamacher, Vranesic,And Zaki "Computer Org Http://Ltcconline.Net/Greenl/Courses/103b/Seqseries/Seqser.Htmanization" Tmh.

11] Ashish Kumar,K Sudipta Achary, Motahar Reza "A Parallel Algorithm To Compute Shortest Path Between Two Node In A Graph Using Openmp",National Conference On High Performance Computing &Simulation(Nchpcs)18th To 19th Jan 2013,Isbn:978-93-82208-55-6.

[12]Http://Msdn.Microsoft.Com/En-s/Library/Ff649143.Aspx. Retrive on 05/07/2013.