VOL. 10, NO. 16, SEPTEMBER 2015 ISSN 1819-6608

ARPN Journal of Engineering and Applied Sciences

©2006-2015 Asian Research Publishing Network (ARPN). All rights reserved.

www.arpnjournals.com

PERFORMANCE IMPROVEMENT IN DATA SEARCHING AND SORTING USING MULTI-CORE

Venkata Siva Prasad Ch., Ravi S. and Karthikeyan V.

Department of Electronics & Communication Engineering, Dr.M.G.R. Educational and Research Institute University, Chennai, India

E-Mail: siva6677@gmail.com

ABSTRACT

Recently multi-core processors have become more popular due to performance, and efficient processing of multiple tasks simultaneously using concurrent and parallel languages like openMP programming The design of parallel algorithm and performance measurement is a major issue on multi-core environment and multi-core modules used in Searching and sorting of data in unsorted database. Multi-core based searching and sorting (of serial and parallel algorithms) can reduce the execution time considerably compared to single core. In the proposed work searching and sorting is done for numbers as well as words in a large database with comparison of both single and multicore implementation. Multi-core offers explicit support for executing multiple threads in parallel and thus reduces the time. The results for number searching and sorting, word searching and sorting is presented and also speed up achieved using multi-core over single core is reported. Hardware implementation is done on Gizmo board (dual core).

Keywords: searching, sorting, AMD, multi-core, openMP, parallel processor.

INTRODUCTION

Advancement in technology demands processing of large data (ex: bio-informatics, web applications etc.) reaching peta bytes. Large databases are common in Telecom fields, social media sites, medical filed etc. In large databases the issues are not only with the volume of data but on the database speed and on the hardware associated with it. Also, the method must be suited for various data types like text, audio, video and image. Sorting is a process of arranging the list of items in a order. The existing methods for sorting have a problem of quadratic time complexity when the size of the data is very huge and this requires high amount of memory and stand-alone resources. Existing algorithms for searching includes serial, binary searching, etc. The performance of a searching algorithm is evaluated using the following metrics.

(i) Average, worst case and best possible time. The execution time for searching and sorting is reduced when workload is divided and given to multithreads. i.e. use of multi-cores. The focus of this paper is on efficiency improvements that are specific to large database searching and sorting. We use serial and parallel algorithms for sorting and searching as a test bed to explore parallelization schemes that may possibly apply without significant changes to other divide-and conquer methods the parallelization is well-studied in theory is known of how to implement parallelization in scheduling of the listed data in multi-core on mainstream architectures (such as standalone), by means of mainstream shared memory. The paper is organized as follows: Section 2 presents a brief introduction to Multi-core systems; section 3 gives the concept of data searching and sorting using openMP. Section 4 shows the hardware details, and implementation of algorithms for searching and sorting for numbers and words and in section 5 comparisons between single and multi-core performance is presented along with the speedup achieved.

MULTICORE SYSTEMS AND OPENMP

The tasks can be dynamically scheduled for execution based on the mutual dependencies and on the computational resources available. The dynamic runtime system efficiently schedules the implemented kernels across the processing units and ensures the data dependencies are not violated. In the central processing unit (CPU), independent processing units are called cores and the multi-core processor is one which utilizes more than one core. Multi-core processors plays a vital role in parallel processing as the cores share workload and perform load balancing in the running time. The core can be with one thread or more thread each. The software running in the processor is a key factor in deciding the performance of multi-core processor. Since workload is shared equally among the processors, execution time is minimized and results in better performance. The main advantage of multi-core systems over single core systems is time consumption. But there are some drawbacks in multi-core processor. For example in a sorting application the task is divided into number of threads and each thread is running in separate cores. In this case for each thread there is a need for separate set of computational resources. Alternately, single core systems are optimal over the multi-core systems. Additionally, the amount of heat radiation can increase with the number of cores. The above mentioned drawbacks need to be considered as a trade off to faster execution time.

Multiple cores to remove deadlock

Multi-core module can overcome deadlock situations using scheduling types. Deadlock can occur due to process running in single core and use of more cores in an optimal manner can relieve
Deadlock can occur in

- **Mutual exclusion:** only one process at a time can use a resource.
- **Hold and wait:** A process holding at least one resource which is waiting to acquire additional resources held by other processes
- **No pre-emption:** A resource can be released only voluntarily by the process holding it, after that process has completed its task.
- **Circular wait:** there exists a set {P0, P1, …, P0} of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P2, …, Pn–1 is waiting for a resource that is held by Pn, and Pn is waiting for a resource that is held by P0.

The Deadlock due to circular wait is shown in Figure-1(a), along with the process dependency shown in Table-1 from Table-1, the wait among the processes P_A, P_B, P_C and P_D are (P_A, P_B, P_C, P_D, P_A). to overcome the deadlock, a new core to handle process P_C alternately (instead of P_B) is use as shown in Figure-1(b) along with the new dependency Table-2.

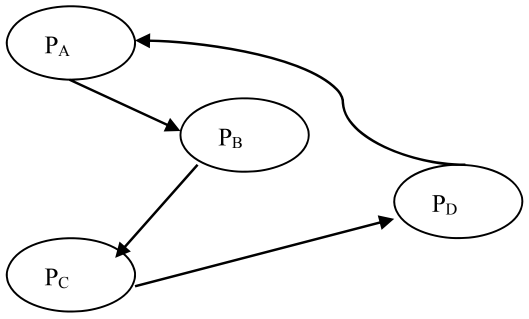

Figure-1(a).

Table-1.

Process	Parent	Child
A	D	B
B	A	C
C	B	D
D	C	A

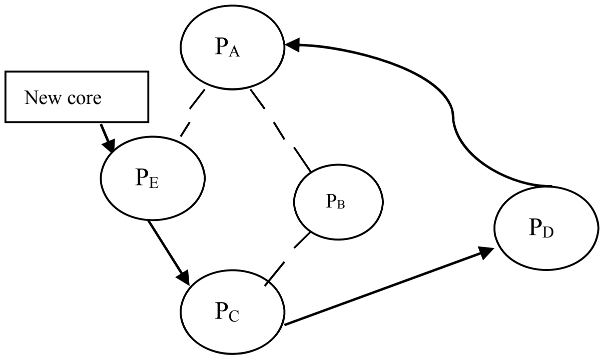

Figure 1(b).

Table-2.

Process	Parent	Child
A	D	E(NEW CORE)
E	A	C
C	E	D
D	C	A

THREAD AFFINITY

The Thread affinity has been studied in shared memory with various views introduced new affinity level system calls. AMD and INTEL compilers are allow programmer to control thread binding by following modules. Thread affinity cache between processor to processor can have a dramatic effect on the application speed. *Thread affinity* restricts execution of certain threads (virtual execution units) to a subset of the physical processing units in a multiprocessor.

Thread affinity is supported on Windows OS systems and versions of Linux OS systems that have kernel support for thread affinity. The compiler's OpenMP runtime library has the ability to bind OpenMP threads to physical processing units.

The total number of processing elements on the machine is referred to as the number of *OS thread contexts.*

Each processing element is referred to as an Operating System processor, or *OS proc.*

Each OS processor has a unique integer identifier associated with it, called an *OS proc ID.*

The term *package* refers to a single or multi-core processor chip.

low affinity - when no state is retained in the core's cache at each new scheduling data

high affinity - when the benchmark's state is almost entirely retained in the cache at each new scheduling data.

OpenMP

The speed of execution of a task will be improved if all the available cores are utilized as per load that is task is divided and allocated to number of cores and this is the concept called multithreading. OpenMP has been very successful in exploiting structured parallelism in applications. A thread is a single sequential flow of control within a program. OpenMP simplifies parallel application development by hiding many of the details of thread management and communication. It is suitable for different processor architectures and a number of operating systems. In openMP based multithreading concept the tasks is divided and send to various threads that simultaneously. Each thread executes the allotted sub tasks individually. Once all the slave threads have completed its execution they are joined. OpenMP is easy to program and mostly used for parallelizing the serial and parallel programs. The master thread assigns tasks unto worker threads. Afterwards, they execute the task in parallel using the multiple cores of a processor. The model of openMP is shown in Figure-2.

- The OpenMP supports loop level scheduling. This defines how loop iterations are assigned to each participating thread. Those scheduling modules have run in the single core and multi core processor based applications Scheduling types include:

Static: Each thread is assigned a chunk of iterations in fixed fashion (round robin).

Guided: Given Iterations are divided into pieces that successively decrease exponentially, with chunk being the smallest size. This is a form of load balancing.

Dynamic: Each thread is initialized with a chunk of threads, then as each thread completes its iterations, it gets assigned the next set of iterations.

Run-time: Scheduling is deferred until run time. The schedule type and chunk size can be chosen by setting the environment variable OMP_SCHEDULE.

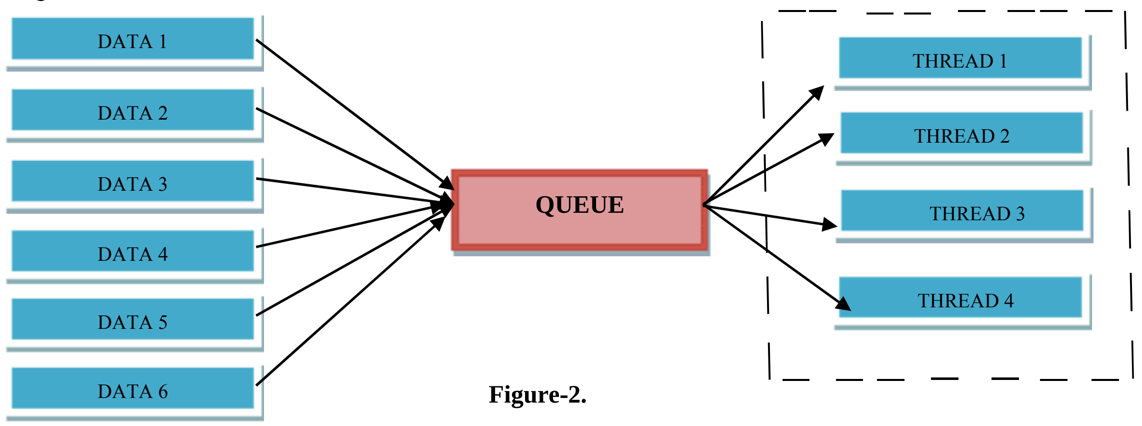

Figure-2.

Creating an OpenMP program with scheduling

- The OpenMP's directives for creating the program and speceify which instructions to execute in parallel and how to distribute them among the threads in the processing time.
- The first and second step in creating parallel program using OpenMP from a serial threading (single core) is to identify the parallelism it contains and then to express, using OpenMP, the parallelism that has been identified.
- Data can be search and public or private in the OpenMP memory model.
- When that data is private it is visible to one thread only, when data is public it is global and visible to all threads.
- OpenMP divides tasks into threads; a thread is the smallest unit of a processing that can be scheduled by an operating system. The master thread assigns tasks unto worker threads. Afterwards, they execute the task in parallel using the multiple cores of a processor.

OpenMP API's

In this paper searching and sorting applications on multicore using openMP is done Those modules of API's are used in the OpenMP Module. The openMP API's are shown in the Listed Table-3 and Table-4.

Searching

Table-3.

#pragma omp parallel	calls the default number of threads to execute the program
#pragma omp for	allocate the number of iterations to each thread
pragma omp for firstprivate(b, f1)	will make b and f1 as private to each thread with initial values
omp_get_wtime()	it will get that moment of time

Sorting

Table-4.

#pragma omp parallel	Calls the default number of threads to execute the program
#pragma omp sections	It tells the compiler that code is divided into sections which will start with "#pragma omp section" statement. Sections are executed by different threads simultaneously and only 1 thread is allowed per thread.
omp_get_wtime()	It will get that moment of time

IMPLEMENTATION AND PERFORMANCE ANALYSIS

That programming for both serial threading (single-core) and parallel threading (Multi-core) module is implemented on the Linux operating system (OS). The flowchart for serial process is shown in Figure-3. The searching/sorting is done for numeric and alphabets and the execution time is calculated for both single and multi-core.

Thread handling in running time

The thread handling in running process to compile the input from the multithread with openMP processor has follow the Example of the data reallocation on the multithread to threads in running of the total data has been shared on the requiring the data limit.

FOR EXAMPLE:

N=1000 and num of threads =4
Then
→ N/num=250

Out of 1000 elements each thread (0, 1, 2, 3) will execute 250 elements each, thereby the execution time is reduced its shown in Table-5. The flow chart for parallel implementation is shown in Figure-4.

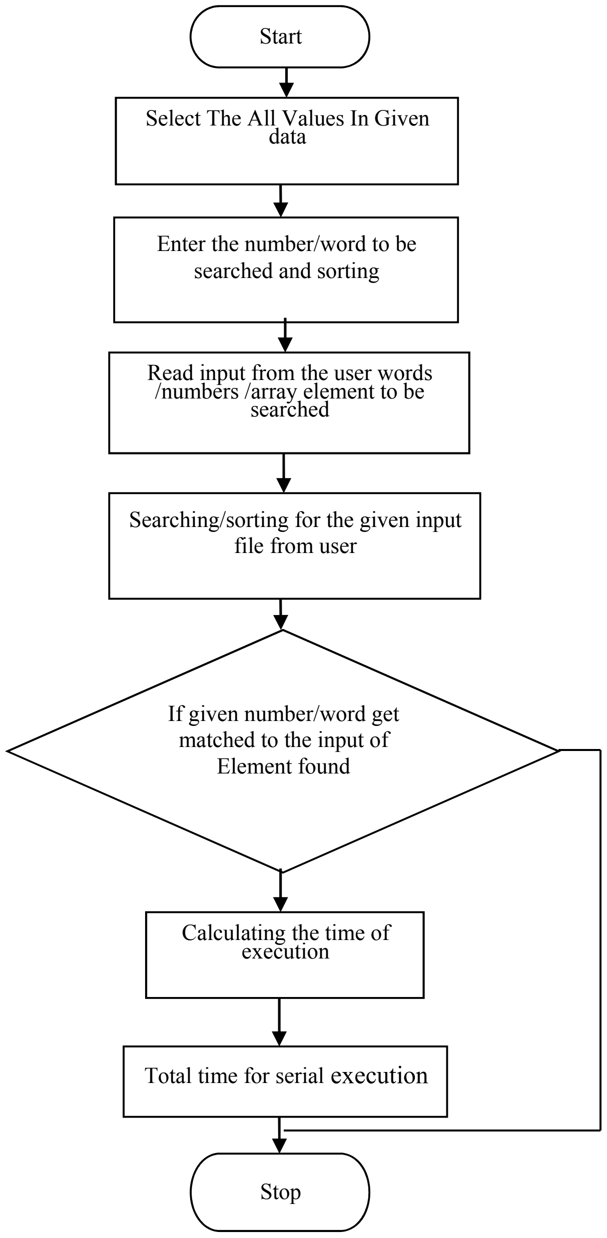
Start
Select The All Values In Given data
Enter the number/word to be searched and sorting
Read input from the user words /numbers /array element to be searched
Searching/sorting for the given input file from user
If given number/word get matched to the input of Element found
Calculating the time of execution
Total time for serial execution
Stop

Figure-3 Flow chart for serial process.

VOL. 10, NO. 16, SEPTEMBER 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences
©2006-2015 Asian Research Publishing Network (ARPN). All rights reserved.

Table-5.

Thread 0	Thread 1	Thread 2	Thread 3
X=0*250+1 = 1 P_1=1*250 = 250	X=1*250+1=251 P_2=2*250 = 500	X=2*250+1=501 P_3=3*250 = 750	X=3*250+1=751 P_4=4*250 = 1000

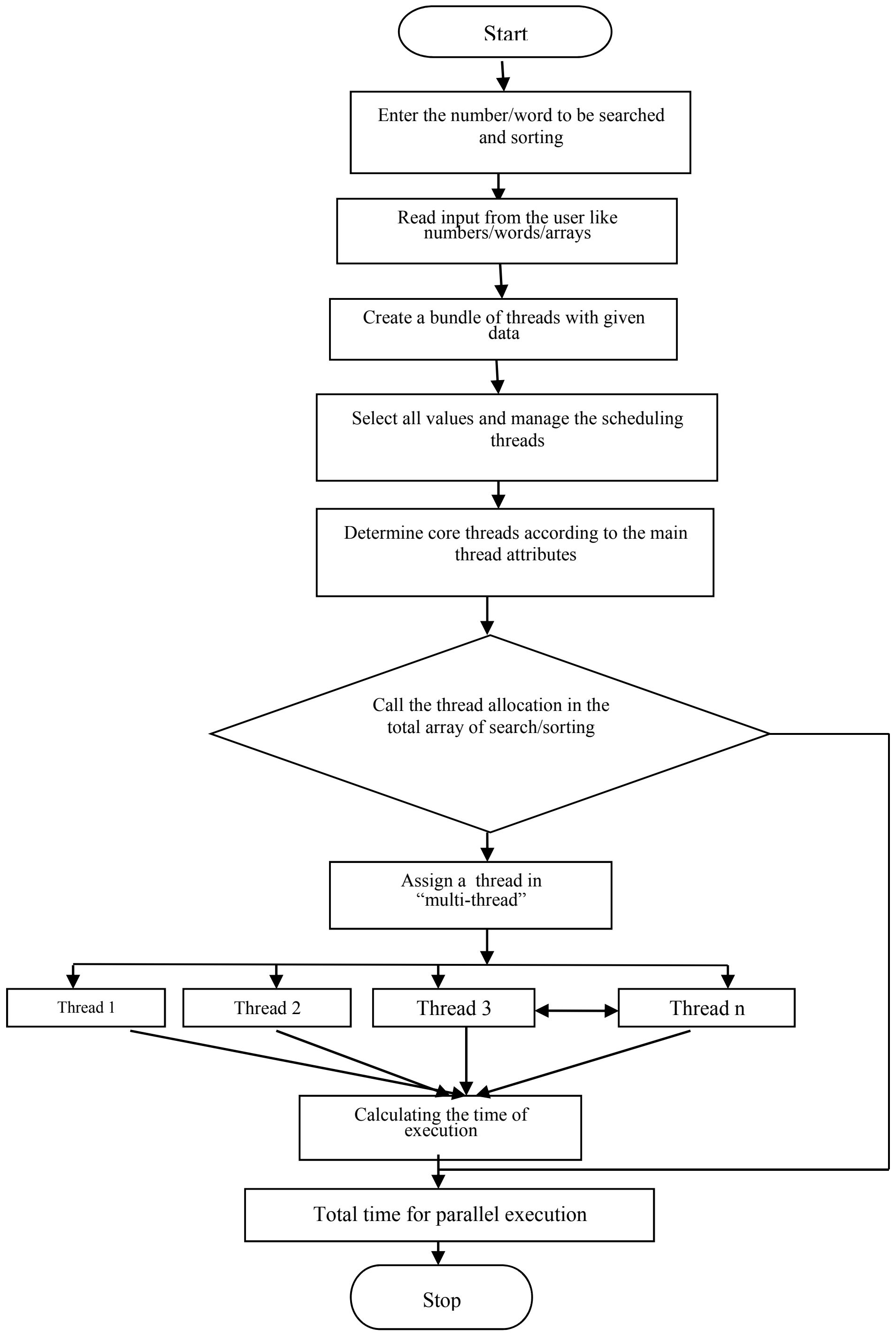
Start
Enter the number/word to be searched and sorting
Read input from the user like numbers/words/arrays
Create a bundle of threads with given data
Select all values and manage the scheduling threads
Determine core threads according to the main thread attributes
Call the thread allocation in the total array of search/sorting
Assign a thread in "multi-thread"
Thread 1
Thread 2
Thread 3
Thread n
Calculating the time of execution
Total time for parallel execution
Stop

Figure-4. Flow chart for parallel process.

Experimental setup

The AMD Embedded G-Series SOC platform is a high-performance, low-power System-on-Chip (SOC) design, featured with enterprise-class error-correction code (ECC) memory support, The AMD G-Series SOC achieves superior performance per watt in the low-power x 86 microprocessor classes of products when running multiple industry standard benchmarks. This helps enable the delivery of an exceptional HD multimedia experience and provides a heterogeneous computing platform for parallel processing. AMD Embedded G-Series SOCs build on the strength of the AMD G-Series APU architecture to provide an array of performance and power options and enhanced multimedia capabilities via a single scalable architecture This improvement is made possible by the seamless single chip integration of CPU, GPU and I/O controller.

The explorer board: A companion board for Gizmo, the Explorer expansion I/O board allows for even greater experimentation and exploration opportunities. This two-layer board connects to Gizmo via the low-speed connector and provides an alpha-numeric keypad, a micro-display, and a sea of holes for prototyping and customization.

Performance improvement

The Architecture based Multi-core in-memory databases for modern machines can support extraordinarily high transaction rates for online transaction processing workloads. The amount of performance benefit an application will realize by using OpenMP depends entirely on the extent to which it can be parallelized. ***Amdahl's*** law specifies the maximum speed-up that can be expected by parallelizing portions of a serial program. Essentially, it states that the maximum speed up (*S*) of a program is

$$\mathbf{S = 1/ (1-F) + (F / N)}$$

where, F is the fraction of the total serial execution time taken by the portion of code that can be parallelized and N is the number of processors over which the parallel portion of the code runs. The metric that have been used to evaluate the performance of the parallel algorithm is the speedup. It is defined as ***Sp = T1 / TP***

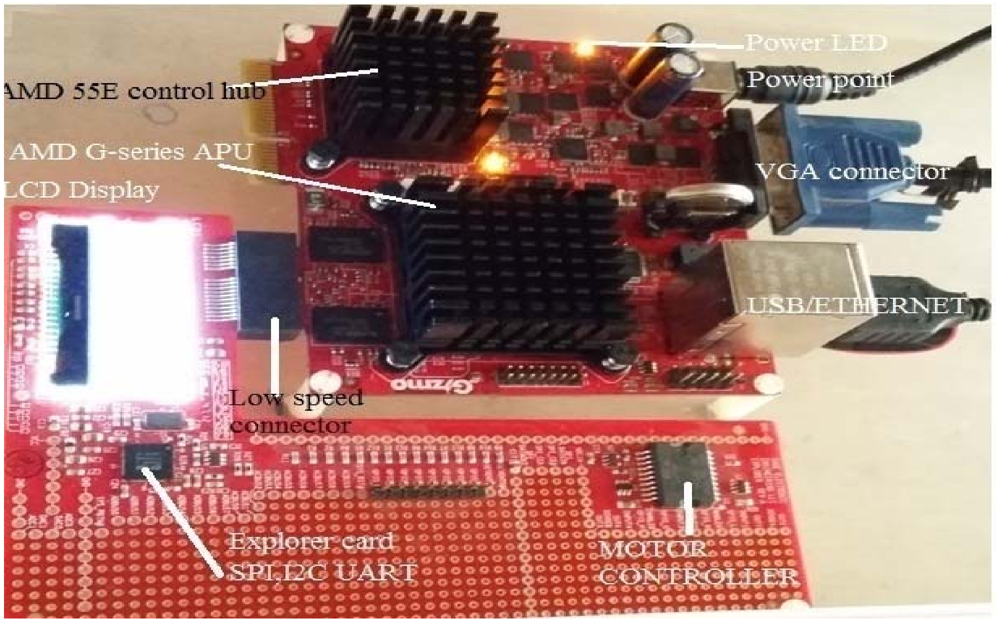

Figure-5. AMD GIZMO board.

A loop for checking the number of numbers in database is written. Then setting back pointer to beginning of the file, the parallel regions starts for parallel execution and appropriate constructor (For example 2 threads for dual core). Scanning and storing the numbers in an array from is done the file. The program is divided in two sections, one section is executed by 1 thread and second section is executed by another thread simultaneously. Each thread will search their sections for number. If anyone thread finds that number, immediately convergence is achieved and they both stop(both threads will be monitoring "Flag" continuously if any one thread defects number then flag will be set 0), Output statement will written to the output file(("fredlog.txt "), with time taken for the execution in the log file(fredlog.txt).

Alternately the same program serially, and we need to set the" MULTICORE" as "0" can be executed. It executes all the same but with only one core or thread for sorting.

EXPERIMENTAL RESULTS AND DISCUSSIONS

There are two version of algorithm: serial and parallel. The programs are executed on AMD Gizmo@Core2-Duo processor machine. We analyzed the performance using results and finally derived the conclusions. The linux compiler fopenmp under compilations and executions. The linux C++ compiler supports multithreaded parallelism with */Qopenmp* flag. In this experiment the execution times of both the serial (single core) and parallel (multicore) algorithms have been recorded to measure the performance (speedup) of parallel algorithm against serial.

In dual-core processing the program is divided into two sections or threads in dual core processor, one section is executed by 1 thread and another section is executed by another thread simultaneously. Each thread will sort their section's names and store in another array. Once both threads complete their work (# pragma omp barrier). The outputs are merged .The same program can be executed serially, for that we need to set the" MULTICORE" as "1". It executes all the same but there will only two core or two threads will sort whole database. The results obtained for number searching is presented in Table-4 and Figure-6. The results obtained for number searching is presented in Table-5 and Figure-7. The results obtained for number searching is presented in Table-6 and Figure-8. The results obtained for number searching is presented in Table-7 and Figure-9. The result has shown in milli seconds (ms). The overall improvement achieved using multicore over single core in searching and sorting is presented in Figure-10.

Table-6. Results of number searching.

Number to search	**Single core** (Ms)	**Multi core** (Ms)
40	0.06631	0.0013
50	0.06747	0.0025
60	0.06852	0.0031
70	0.06931	0.0040
80	0.06986	0.0047
90	0.07008	0.0051

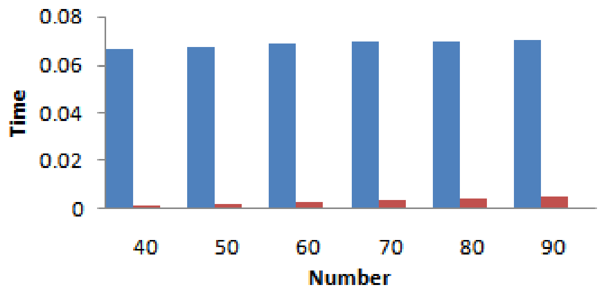

Figure-6. Number searching.

Table-7. Results of number sorting.

Array size	**Single core** (Ms)	**Multi core** (Ms)
10000	1.1208	0.4179
15000	2.0751	0.8754
25000	5.3759	2.4767
30000	8.9471	3.5523
50000	17.613	8.3173

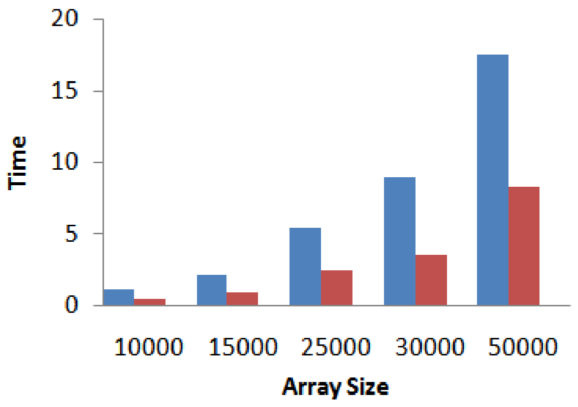

Figure-7. Number sorting.

Table-8. Results of word searching.

Query word	**Single core** (Ms)	**Multi core** (Ms)
Time	0.001863	0.00036
Apple	0.002014	0.00039
University	0.002501	0.00042
Mango	0.001963	0.00038
Communication	0.002795	0.00045
Electronics	0.001972	0.00051

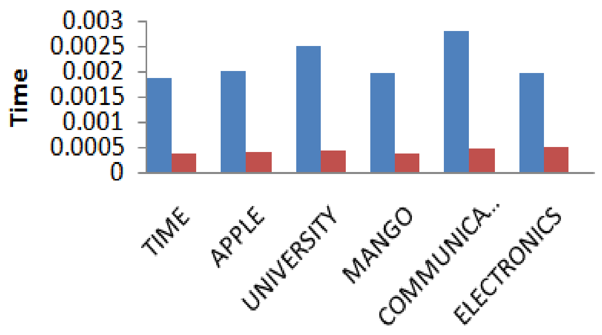

Figure-8. Word searching.

Table-9. Results of word sorting.

Array size	**Single core** (Ms)	**Multi core** (Ms)
9902	4.7731	2.021
9915	4.8585	2.037
9978	4.8971	2.108
9997	5.0374	2.284
9821	4.9484	1.986

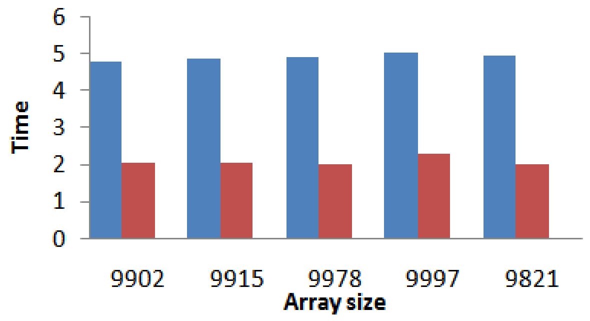

Figure-9. Word sorting.

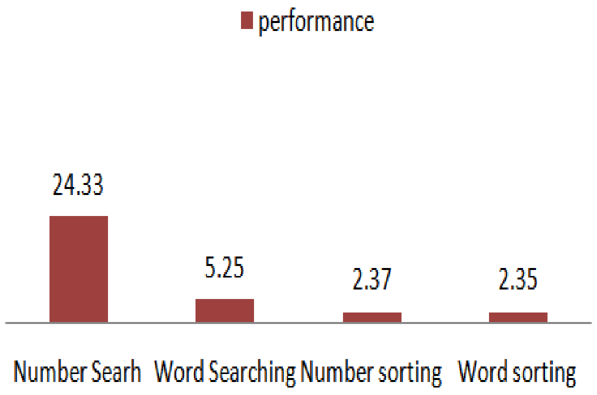

Figure-10. Speedup achieved using Multi-core.

CONCLUSIONS AND FUTURE ENHANCEMENT

In this paper, serial and parallel implementation of algorithm on a Dual core processor for data mining is presented The maximum speedup achieved with two cores in searching is 15 % and 2.4 % in sorting. This is almost twice the speed of the execution with serial algorithm compare with the parallel module (multi-core).This clearly indicates that as the number of cores increase, the computation time taken by a parallelism is also less. This analysis is done on a small data set. As the size of the data mining becomes large and as the number of cores increase, parallel programs written with OpenMP gives much better performance. The applications written in OpenMP can be further analysed. The implemented algorithm on core processor in a serial manner with standard datasets using different support counts on a dual core processor using OpenMP is in progress. From our proposed work we can ensure that parallel results would be better than serial once. Our aim of measuring the serial and parallel performance of a multi-core core processor with respect to time and performance comparison between them would get satisfied in future, using alternative constraints handling method, parallel algorithm and high performance computing paradigm a better speed up can be achieved

RERFERNCE

Sheela Kathavate1, N.K. Srinath. 2014. Efficiency of Parallel Algorithms on Multi CoreSystems Using OpenMP. International Journal of Advanced Research in Computer and Communication Engineering. 3(10).

M Rajasekhara Babu, M Khalid, Sachin Soni. 2011. Performance Analysis of Counting Sort Algorithm using various Parallel Programming Models. International Journal of Computer Science and Information Technologies. 2(5): 2284-2287.

Sanjay Kumar Sharma and Kusum Gupta. 2012. Performance Analysis of Parallel Algorithms on Multi-core System using OpenMP. International Journal of Computer Science, Engineering Information Technology. 2(5): 55-64 Journal of Research and Industry. 1(1): 30-35.

Chao-Chin Wu, Lien Fu Lai, Chao Tung Yang, Po Hsun Chiu. 2012. Using Hybrid MPI and OpenMP programming to optimize communication in parallel loop self-scheduling scheme for multicore PC clusters. The Journal of Supercomputing. 60(1): 31-61.

A. J. Umbarkar, M. S. Joshi, P. D. Sheth. 2015. OpenMP Dual Population Genetic Algorithm for Solving Constrained Optimization Problems. I.J. Information Engineering and Electronic Business. 1, 59-65.

Nilesh.S.Korde1, Prof.Shailendra.W.Shende2 IOSR. 2014. Parallel Implementation of Apriori Algorithm. Journal of Computer Science (IOSR-JCE) e-ISSN: 2278-0661, pp. 01-04.

Jiawei Han and MichelineKamber. 2006. Data Mining concepts and Techniques, 2nd Edn. Morgan Kaufmann Publishers, San Francisco.

Karthikeyan V and Ravi S. 2014. Efficient scheduler and multi threading for resource aware embedded system. Journal of Theoretical and Applied information technology. 67(3): 755-762.

Lan Xiaowen. 2014. Research on Multicore PC Parallel computation based on openMP", International Journal of Multimedia and Ubiquitous Engineering. 9(7): 131-140.

PranavKulkarni and Sumit Pathare. 2014. Performance Analysis of Parallel Algorithm over Sequential using OpenMP. IOSR Journal of Computer Engineering. 16(2): 58-62.

Wang L, Zhou L, Lu J and Yip J. 2009. An order-clique based approach for mining maximal collocations. Journal of Information Science. 179(19): 3370-3382.

Vaidehi M and T.R.Gopalakrishnan Nair. 2008. Multicore Applications in Real time systems. Journal of Research and Industry. 1(1): 30-35.

ZaidAbdi Alkareem Alyasseri, Kadhim Al-Attar and Mazin Nasser. 2014. Parallelize Bubble Sort Algorithm Using OpenMP. International Journal of Advanced Research in Computer Science and Software Engineering. 4(1): 103-110