I J C T A, 9(4), 2016, pp. 1869-1879
© International Science Press

An Optimal Solution to Improve the Performance of Embedded System Based on Simple Applications

K. Lakshmi Narayanan[1], S. Ravi[2] and M. Anand[3]

ABSTRACT

Superior performance is the basic requirement of any real time electronic components these days. Speed, Power consumption and heat dissipation are the major challenges of modern embedded application. In this paper we develop a two different applications namely searching and sorting, and implemented it in a Gizmo sphere G Series APU in an OpenMP programming language meanwhile the optimization technique is affinity switching for improving application's performance. The results of Multicore embedded processors are compared with traditional single core processor. Experimental results show Multicore processors show an accelerated performance than the traditional single core processor.

Keywords: Open MP (Open Multicore Programming); Multicore Processor; Searching; Sorting; Multithreading.

1. INTRODUCTION

Gordan.E.Moore, 1965 "The number of transistors that can be placed on an integrated circuit is increasing exponentially, doubling approximately every 18 months". So memory size in chip will also double every two years. But now the single core processors have reached its threshold in performance. Overheating and power consumption has become a critical issue in a single core processor, since performance per watt is considered to be the main objective of the recent technology development. This leads to a new trend of Multi-core Processor.

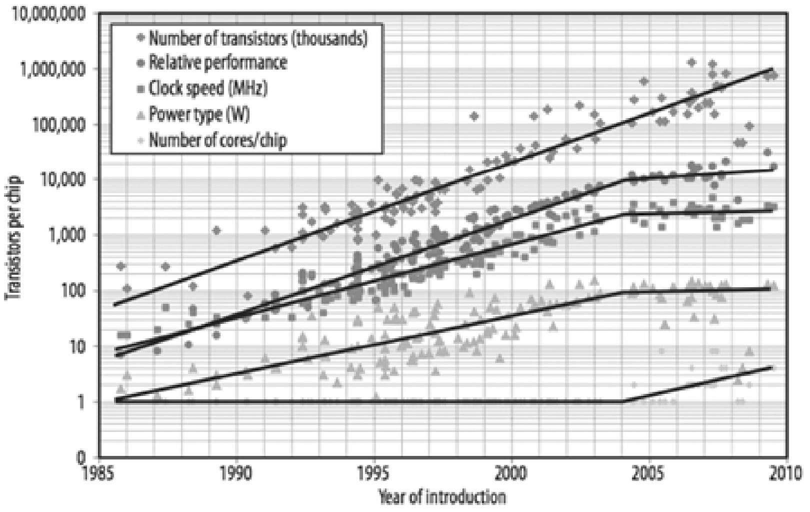

Figure 1: Moore's Law and Multicore Era

[1] Research Scholar, ECE Department, St.Peter's University, Chennai, India, *Email: kyelyen@gmail.com*

[2] Professor and Head, ECE Department, Dr. M.G.R. Educational and Research Institute University, Chennai, India.

[3] Professor, ECE Department, Dr. M.G.R. Educational and Research Institute University, Chennai, India.

1.1. Evolution of Multicore Processors

1.1.1. Multi CPU

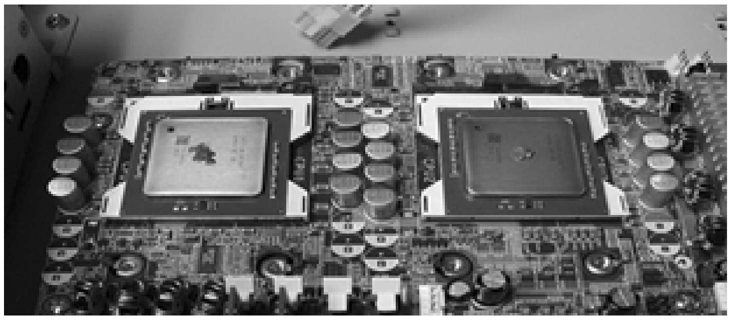
Fig 2: System with 2 CPU's

Initially the computer era started with a single CPU when the complexity of the application started increasing a single CPU cannot accommodate those applications so people tried adding additional power to the computer by adding additional CPUs. But the requirement for this also increased as it requires more than one CPU socket. Additional hardware are also required to connect those sockets connected in the mother board to the RAM. On solving this as multiple CPUs connected together to perform a task the speed got increased, but in the cost of more power consumption.

Nowadays it is very hard to see this type of multi CPUs in the regular home PC, unless it is a supercomputer or a server which requires more processing power.

1.1.2. Hyper threading

When Intel tried to solve the power consumption problem it came up with an excellent solution that is hyper threading. A single CPU has a Single Core, but with single core with hyper-threading duplicated the single core as two cores and makes it pretend as a two logical CPU (cores). The CPU Remains a single core CPU, while the OS sees as two cores for a CPU. Hyper threading allows the CPU cores to share the resources, and can help in speed up the system.

1.1.3. Multi core Processor

In order to increase the performance of the single core processor manufacturers added additional cores to the CPU. A dual core processor has 2 individual physical cores and the OS sees this as two CPUs.

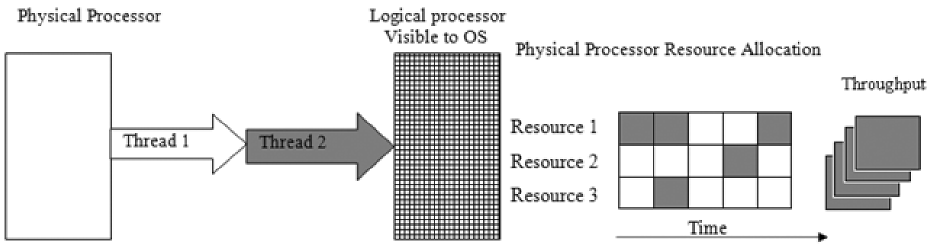

Figure 3: Single core with no hyper threading

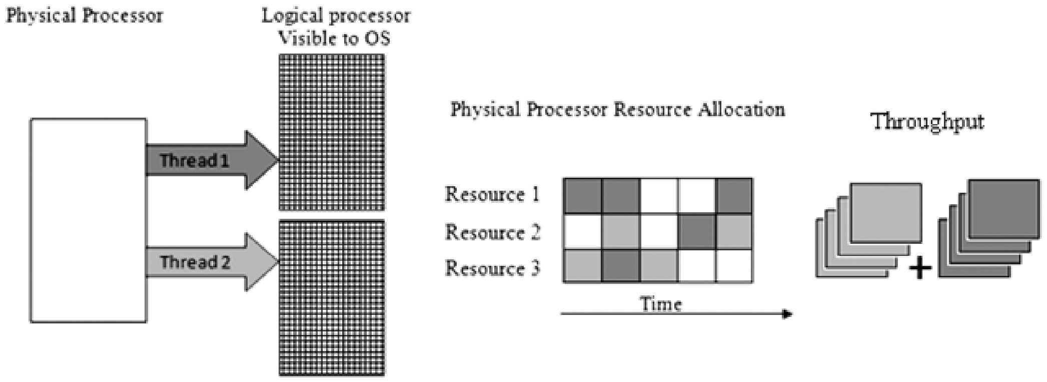

Figure 4: Single core with hyper threading

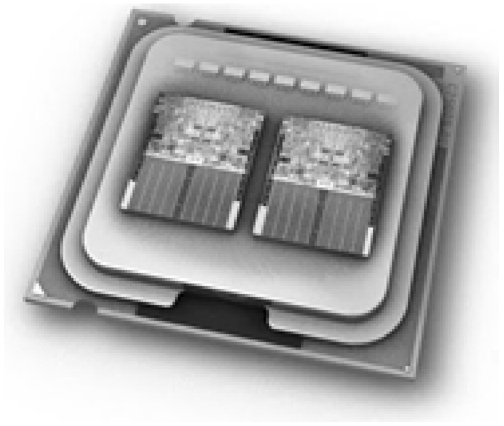

Figure 5: Multicore Processor

As each CPU can perform some operation at the same time computational speed up can be increased in the system. With hyper threading added to a dual core processor still more processing speed can be achieved as hyper threading on each core make a single core virtually two cores.

A **multi-core processor** is a single computing component with two or more independent actual central processing units (called C**ores**"), which are the units that read and execute program instructions and handle the power issues in single core.

Independent processor	=	A Core,
Multiple Cores	=	Better performance

Multicore Processor gave rise to Multicore programming which is said to be important leap in software development. Some of the parallel programming tools are:

- Pthreads
- Open MP

Pthreads exposes the full complexity of threads, OpenMP are much simpler and is used in this application.

2. OPEN-MP PROGRAMMING

2.1. Open Multicore Programming (Openmp)

For writing multi-thread applications in a shared memory environment the popular tool used is OpenMP (**Open Multi-Processing).** This OpenMP comprises of a set of compiler directives and runtime library routines

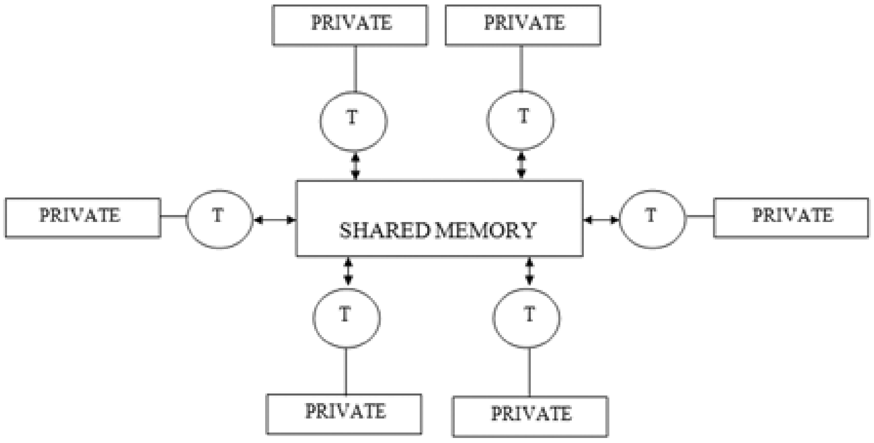

Figure 6: Shared Memory Architecture

In this architecture (figure 6) all the threads denoted by T have the access to the same globally shared memory. Data can be shared or private. If it is a shared data it will be accessible to all the threads, if it is private data it can be accessed only by the thread that owns it. The changes made in local data or private data cannot be seen by others. But if any change is made in global data or shared data it can be seen by all others. Data transfer is transparent to the programmer.[8]

OpenMP uses the fork-join model of parallel execution (figure 2). Initially all OpenMP program begins as a single process or the master thread. Master thread executes in serial mode until the parallel region construct is come across. Master thread generates a group of parallel threads (fork) that concurrently execute statements in the parallel region. After executing the statements in the parallel region, team threads synchronize and terminate (join) but master continues.

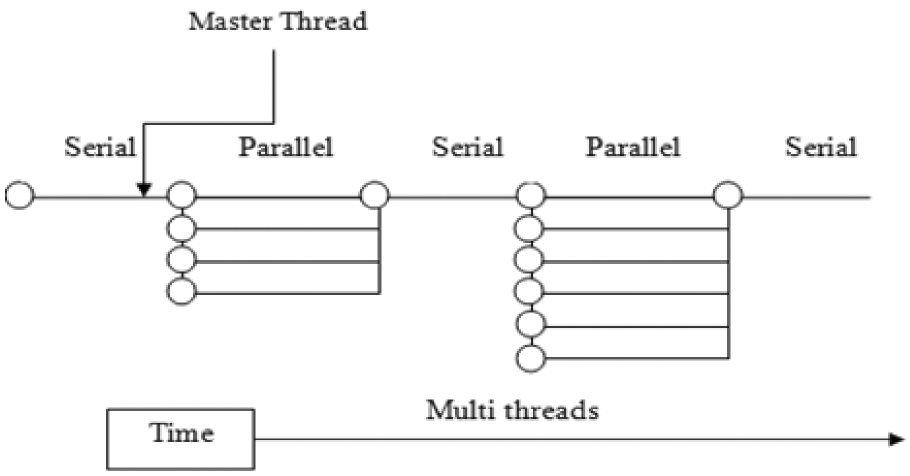

Figure 7: OpenMP Execution Model

2.2. OpenMP Constructs

The core elements of OpenMP are

1) *Parallel Control Structure:* It governs the flow of control in the program.
2) *Parallel Control Work Sharing:* It distributes the work among threads.
3) *Data Environment:* It specifies the variables as shared or private.
4) *Synchronization:* It coordinates the thread execution.
5) *Runtime functions and environmental variables:* OpenMP runtime library functions are included in the header <omp.h>. They include execution environment functions that can be used to control and query the parallel execution environment, and lock functions that can be used to synchronize access to data.

In C/C++, OpenMP uses #pragmas. The pragmas omp parallel is used to fork additional threads to carry out the work enclosed in the construct in parallel. The original thread will be denoted as master thread with thread ID 0.

2.3. How is OpenMP typically used?

OpenMP API Example: (Consider a sequential code)

```
Statement 1
Statement 2
Statement 3
```

Consider that we need to execute statement 1 and 3 sequentially & statement 2 in parallel.

OpenMP parallel code:

```
statement 1;
#pragma <specific OpenMP directive>
statement2;
statement3;
```

Now Statement 2 (may be) executed in parallel.
Statement 1 and 3 are executed sequentially.

OpenMP is usually used to parallelize loops, it find the most time consuming loop and Split them up between threads.

2.4. Gizmo Sphere Hardware Board

The application developed in OpenMP is implemented in a Gizmo Sphere board. The Gizmo board is a compact, low cost development board. It combines the power of supercomputer and I/O capabilities of a Microcontroller. Built upon the AMD G-series Accelerated Processing Unit (APU), the Gizmo board can deliver over 50GFLOPS at less than 10W.

The heart of the Gizmo board is AMD G-Series APU it contains two powerful x86 cores running at 1 GHz 1Mb of cache memory and radian HD graphics engine with 80 pipelines running at 280 MHZ. In addition to

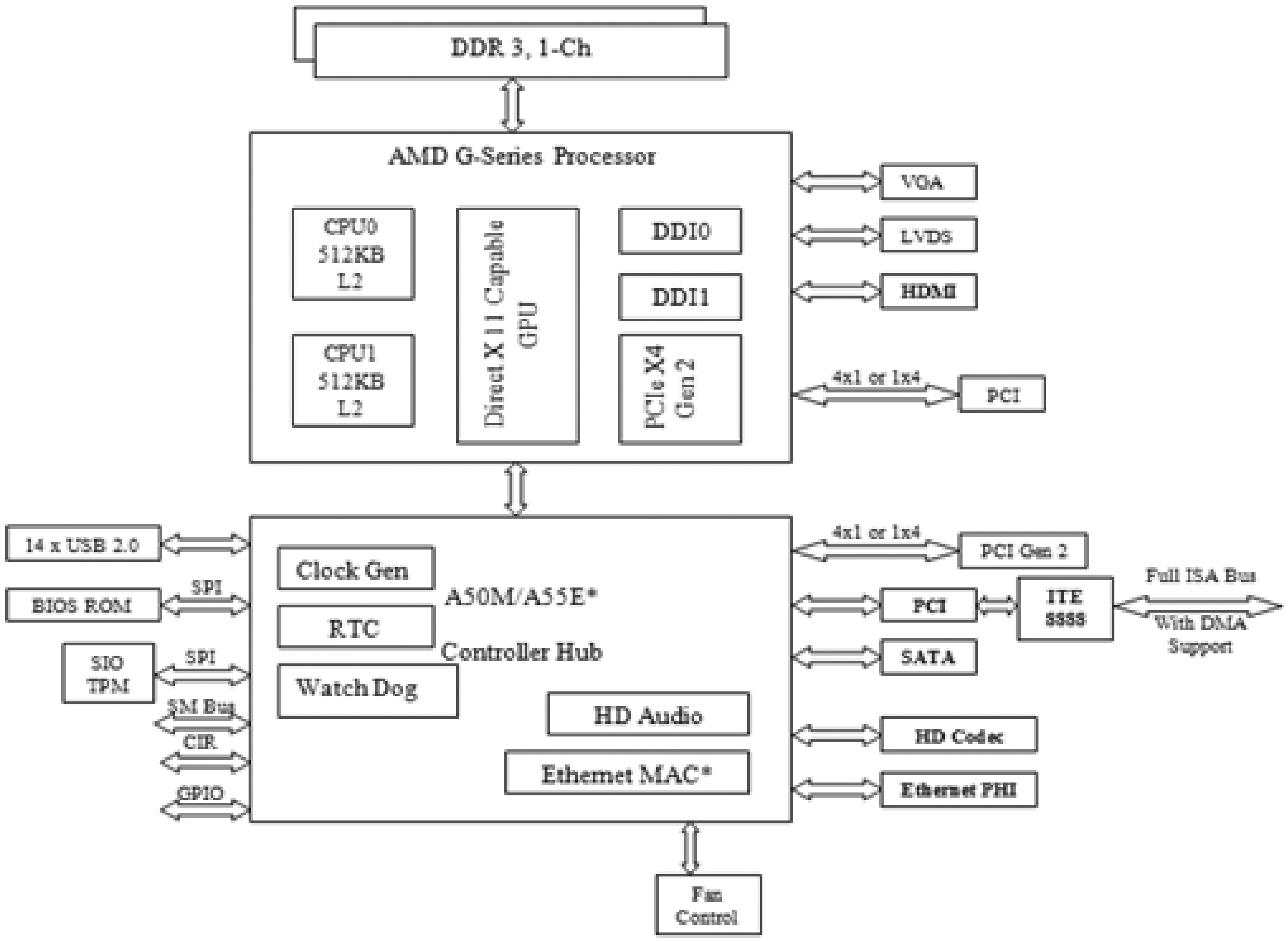

Figure 8: AMD G Series platform Block diagram

being able to run Android, Linux or Windows, there's also a host of connectivity choice packed in. There's a VGA connector and 2 USB 2.0 ports, along with an Ethernet jack and a pair of I/O connectors one of which can be connected to Gizmo Sphere's Explorer board, which has an area for prototyping, keypad, and LCD [13].

The main objective of this research is to obtain an optimal performance in a Multi-core Processor. And the optimization technique used is Affinity switching. Assigning a process scheduling to one processor is known as processor affinity. Affinity switching means assigning a process to a specific core dynamically in a Multicore processor, based on the loads of the cores the next process will be assigned to the same core or to the other core. That is consider a Multicore processor with two cores say Core 1 & Core 2 and each of this core is assigned with some tasks, say task 1, task 2 and task 3 etc. Initially the first two tasks will be assigned to Core 1 and Core2 respectively, now the load in the two cores are measured if the Load 1 is greater than Load 2 then the next task will be assigned to Core 2. To demonstrate the effectiveness of this process testing is done in two different applications, Searching and Sorting Numbers and Words

3. RELATED WORKS

Sheela Kathavate, N.K. Srinath [2014], analyses the performance improvement of a parallel algorithm on multi core systems. The results shows major speed up achieved on multi core systems with the parallel algorithm i.e. they tested their experiment with Matrix multiplication algorithm with OpenMP, and concludes that it performs better than the sequential algorithm. The maximum speedup achieved with two cores is 1.96 which is almost twice the speed of the execution and with four logical processors is almost three times [2]. Ishwari Singh Rajput, Bhawnesh Kumar, Tinku Singh [2012] the compares the performance of three different types of sorting algorithms viz. sequential quick sort, parallel quicksort and hyper quicksort based on comparing average sorting times and speedup and concluded that parallel sorting algorithms i.e. parallel quick sort and hyper quicksort performs well in all respects in comparison to sequential quick sort [3].

Vijayalakshmi Saravanan, Mohan Radhakrishnan, A.S. Basavesh [2012], checks how the performance potential advantage the parallel programming model over sequential programming model and concluded that optimizing the code using OpenMP raises the performance than sequential execution and outperforming well with parallel algorithms[4]. Pranav Kulkarni1, Sumit Pathare [2014] presented a low cost Multicore processors that are need to be paralleled to exploit the Multicore processor throughput gain. Unfortunately, writing parallel code is more difficult than writing serial code, where the OpenMP programming model enters the parallel computing picture and creates multi-threaded applications more easily while retaining the benefits of serial programming. Its performance is a systematic and quantitative approach for constructing software systems to convene the performance intention such as response time, throughput, scalability and resource utilization.[1]

Michael Sub and Claudia Leopold [2004], suggest an easy sorting algorithm to demonstrate problems with recursion and the avoidance of busy waiting. They also compare several solution approaches with respect to programming expense and performance: stacks, nesting and a work queue (for recursion), as well as condition variables and the sched_ yield–function (for busy waiting), and showed that both the approach may afford ample savings in processor time [7].

4. OVERVIEW OF PROPOSED WORK

Performance of any system will be based on the execution time. So considering speed as important criteria we implement a simple searching and sorting application in a sequential single core environment and Parallel Multicore Environment. We use a ordinary sequential programming for implementing the applications in a single core Processor, and Open-mp Parallel Programming for implementing in Multicore Processor. Then we analyze the execution time of each system, the fastest solution for these applications is through Parallel Multicore environment.

The schematic diagram of proposed work is shown in the figure 4

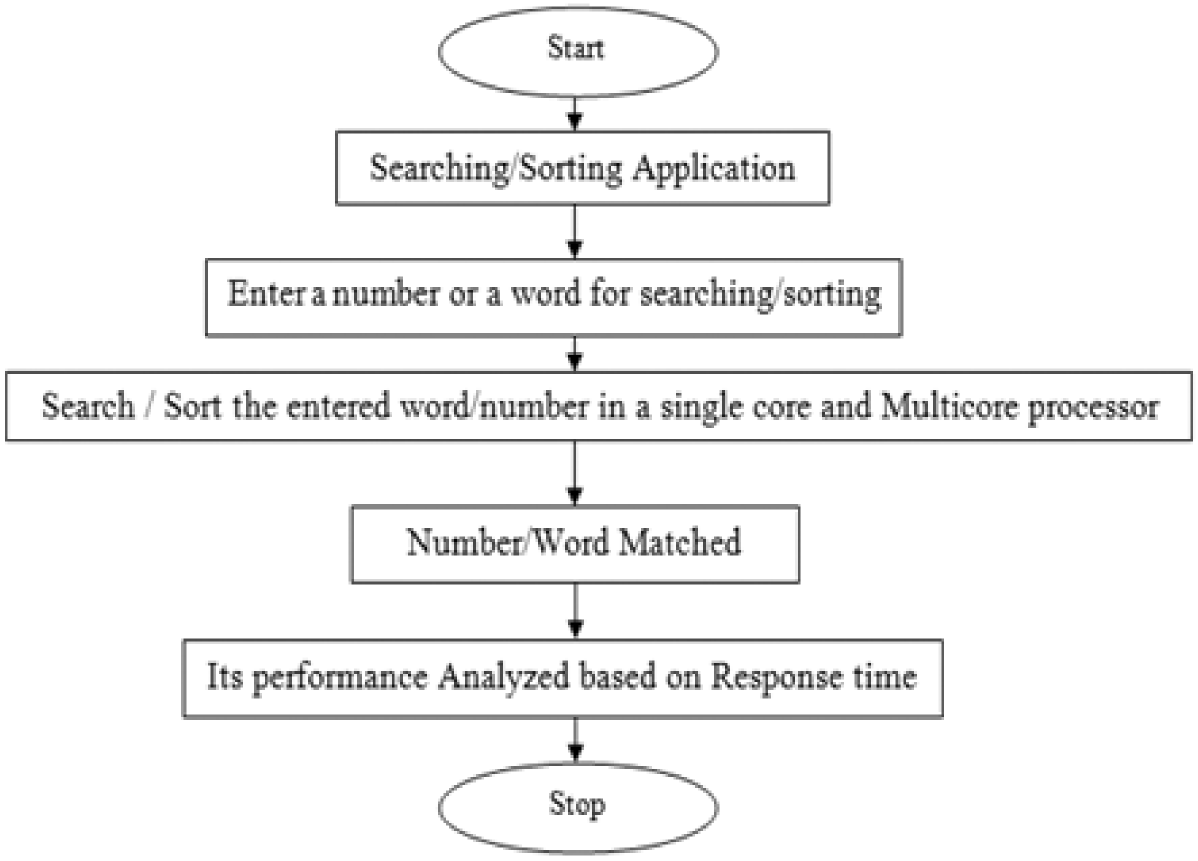
Start
Searching/Sorting Application
Enter a number or a word for searching/sorting
Search / Sort the entered word/number in a single core and Multicore processor
Number/Word Matched
Its performance Analyzed based on Response time
Stop

Figure 9: Modules of Proposed System

4.1. Performance of Parallel Processing.

The total performance advantage of an application can be realized by open Multicore processor (Open Mp) depends solely on the degree to which it can be parallelized. The response time of the CPU can be calculated using CPU performance Equation,

The CPU execution time is calculated by multiplying the CPU clock cycle with clock cycle time.

$$CPU\ \ Execution\ time = CPU\ \ Clock\ \ Cycle \times Clock\ \ Cycle\ time$$

Or,

The CPU execution time is calculated by dividing the CPU Clock Cycle by Clock rate

$$CPU\ \ Execution\ time = \frac{CPU\ \ Clock\ \ Cycle}{Clock\ \ rate}$$

CPU clock cycle can be represented by

$$CPU\ \ Clock\ \ Cycle = Instruction\ Count \times CPI$$

Where Instruction count is the total number of instruction executed and CPI stands for Clock cycle Per Instruction the average number of clock cycles that each instruction consume to execute.

Hence the CPU time is,

$$CPU\ \ Time = \frac{Instruction\ count * CPI}{Clock\ \ rate}$$

4.2. Amdahl’s Law

The response time of the program after the improvement can be analyzed by Amdahl’s law. It is also used to assess practical limits to the number of parallel processors.

Amdahl’s Law for over all speedup

$$Overall\ speed\ up = \frac{1}{(1-F)+\frac{F}{S}}$$

F = Fraction improved

S = Speedup of the improved Fraction.

$$\text{Speedup} = \frac{\text{Execution time of Sequential algorithm}}{\text{Execution time of Parallel algorithm}}$$

$$\text{Efficiency} = \frac{\text{Speedup}}{\text{Number of processor Cores}}$$

Thus Amdahl’s Law along with the CPU Performance equation is a means for estimating the performance improvement.

5. RESULTS AND DISCUSSIONS

The sequential and parallel searching sorting applications were tested on a single core processor without OpenMP and on a Multicore Processor (with two cores). The execution time for number search and sort is

Table 1
Performance comparison of single and Multicore Processor for searching a number.

Number to Search	*Single Core*	*Multi Core*
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

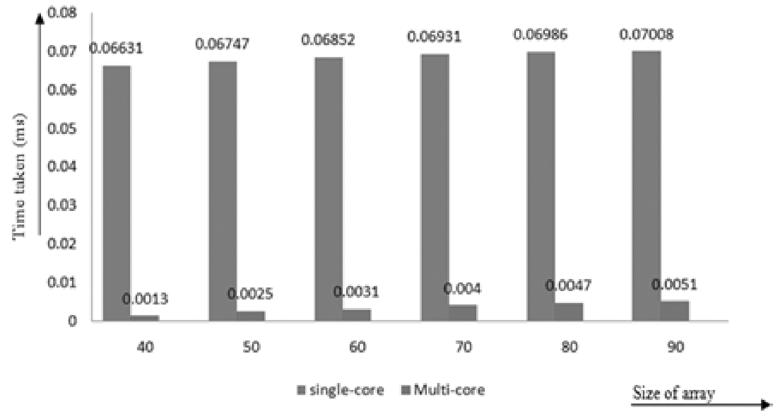

Figure 10: Computation time in Multicore & singlecore processor for number search.

taken. The run time results are shown in Table I and Table II and the corresponding graph is shown in Figure 10 and Figure 11.

The Execution time for word search and word sorting is taken, the run time results are shown in Table 3 and Table 4 and the corresponding graph is shown in Figure 12 and Figure 13.

Table 2
Performance comparison of single and Multicore Processor for sorting numbers

Array Size	*Single Core*	*Multi Core*
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

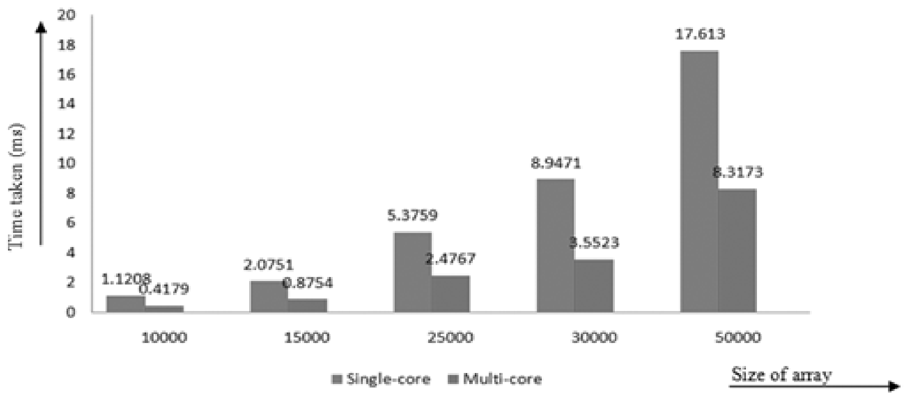

Figure 11: Computation time in Multicore & singlecore processor for Numbers sorting.

Table 3
Performance comparison of single and Multicore Processor for searching a word

Query Word	*Single Core*	*Multi Core*
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

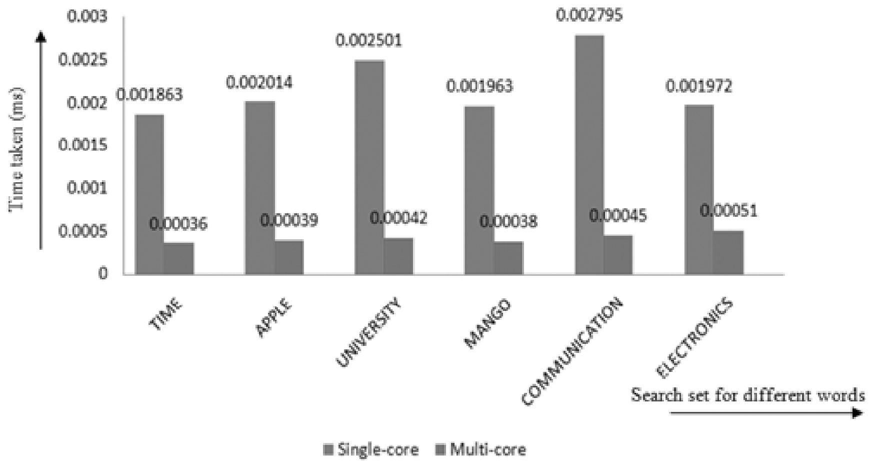

Figure 12: Computation time in Multicore & singlecore processor for word Search.

Table 4
Performance comparison of single and Multicore Processor for Sorting words.

Array Size	*Single Core*	*Multi Core*
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

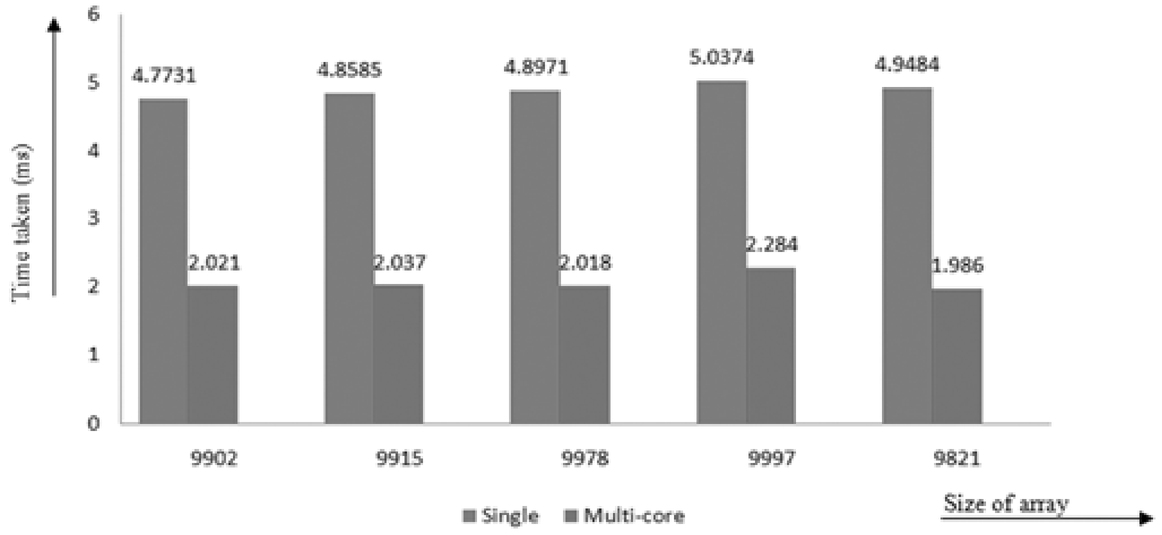

Figure 13: Computation time in Multicore & singlecore processor for word sorting.

6. CONCLUSION & FUTURE WORK

The obtained result from the experimental analysis shows searching and sorting of numbers and words with OpenMP on a Multicore processor with two cores have a speed up of twice the execution with the programming technique without OpenMP on a single core processor. This shows that when the number of core is increased, the time taken for execution of an algorithm is reduced. The results obtained by testing on a finite set of data is also applicable for a large database (DNA pattern search, and sorting of Bioinformatics data etc).

REFERENCES

[1] Pranav Kulkarni, Sumit Pathare (2014), "Performance Analysis of Parallel Algorithm over Sequential Using Open MP" in the proceedings of IOSR Journal of Computer Engineering, vol.16, no.2, pp. 58-62.

[2] Sheela Kathavate, N.K. Srinath (2014), "Efficiency of Parallel Algorithms on Multi CoreSystems Using OpenMP" in the proceedings of International Journal of Advanced Research in Computer and Communication Engineering, ISSN (Online): 2278-1021, Vol. 3, Issue 10, October 2014.

[3] Ishwari Singh Rajput, Bhawnesh Kumar, Tinku Singh (2012), "Performance Comparison of Sequential Quick Sort and Parallel Quick Sort Algorithms" in the proceedings of International Journal of Computer Applications, Volume 57, No. 9, pp. 14-22, November 2012.

[4] Vijayalakshmi Saravanan, Mohan Radhakrishnan, A.S. Basavesh, and D.P. Kothari, (2012), "A Comparative Study on Performance Benefits of Multi-core CPUs using OpenMP" in the proceedings of IJCSI International Journal of Computer Science issues, Vol. 9, Issue 1, No 2, ISSN: 1694-0814, pp 272–278 January 2012.

[5] Barbara Champman, L.H., 2009, "Implementing OpenMP on a high performance embedded Multicore MPSoC", in the proceedings of IEEE International Symposium on Parallel & Distributed Processing, ISBN: 978-1-4244-3751-1, pp 1-8, May 2009.

[6] Dheeraj D., Shruti Ramesh, Nitish B (2012), “Automated Enhanced Parallelization of Sequential C to Parallel OpenMP” in the proceedings of International Journal of Computer Trends and Technology, ISSN: 2231-2803, Vol. 3, Issue. 4, pp. 644–652.

[7] Michael Sub and Claudia Leopold, (2004), “A User’s Experience with Parallel Sorting and OpenMP” in the proceedings of sixth European workshop on OpenMP-EWOMP2004.

[8] www.openmp.org. Openmp application program interface v.3.0.

[9] Rohit Chandra, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, Ramesh Menon, “Parallel Programming in OpenMP”, Morgan Kaufmann Publishers, ISBN 1-55860-671-8, 2001.

[10] Data Mining concepts and Techniques, 2nd Edn. Morgan Kaufmann Publishers, San Francisco.lications (RTCSA), 2012, Pages: 114–123.

[11] Jae Young Park, Kyong Gun Lee, Jong Tae Kim, “ Parellel Merge Sort Implementation Using OpenMP” in the proceedings of the 2011 World Congress in Computer Science Computer Engineering, and Applied Computing.

[12] M. Rajasekhara Babu, M. Khalid, Sachin Soni, Sunil Chowdari Babu, Mahesh “Performance Analysis of Counting Sort Algorithm using various Parallel Programming Models” in the proceedings of International Journal of Computer Science and Information Technologies, Vol. 2 (5), 2011, pp. 2284-2287.

[13] Sanjay Kumar Sharma, Dr. Kusum Gupta “Performance Analysis of Parallel Algorithms on Multi-core System using OpenMP” in the proceedings of international Journal of Computer Science, Engineering and Information Technology (IJCSEIT), Vol. 2, No. 5, October 2012.

[14] www.Gizmosphere.com, Gizmo1 Explorer Kit User Guide, Gizmo1 Quick start guide.

[15] www.amd.com, “AMD Embedded solution guide” July 2013.

[16] Zaid Abdi 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.

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

[18] 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.

[19] C. Liu , J. H. Anderson (2012), Supporting Soft Real-Time Parallel Applications on Multicore Processors in the proceedings of International Conference on Embedded and Real-Time Computing Systems and App Jiawei Han and MichelineKamber. 2006.

[20] Nilesh.S. Korde1, Prof. Shailendra.W. Shende 2 IOSR. (2014), Parallel Implementation of Apriori Algorithm. Journal of Computer Science (IOSR-JCE) e-ISSN: 2278-0661, pp. 01-04.