ISSN (ONLINE): 2349-7084
GLOBAL IMPACT FACTOR 0.238
ISRA JIF 0.351

INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING IN RESEARCH TRENDS
VOLUME 2, ISSUE 4, APRIL 2015, PP 275-279

Effects of Processes Forcing on CPU and Total Execution-Time Using Multiprocessor Shared Memory System

Dr. Subhi R. M. Zeebaree, Karwan Jacksi

Abstract— In this paper the applications of Shared Memory systems towards the implementation of the Parallel Processing approach is provided. Multiple tasks can be dealt with the applications of such systems by using the principles of Shared Memory Parallel Processing programming called Application-Program. The influences of forcing processes amongst processes of Shared Memory system relying on Parallel Processing approach principals are given. These influences are related with computing total and CPU execution times. The CPU usage is also determined with its changing manner depending on the load size and the number of participated CPUs.

Index Terms— Parallel processing, shared memory, multi-core, multi-processors, multiprocessor shared memory system, processing units, computing, program instructions.

1 Introduction

In early days, centralized machines called mainframe computers were used for computing. These machines were large and powerful systems were they could perform all of the computations for a set of clients. Executions of the computations were scheduled in time. Afterwards, these mainframes were developed gradually to systems able to execute multiple concurrent computations simultaneously.

Later on, Personal Computers (PCs) were introduced that gave users the ability to perform their programs on demand without having to wait for the mainframe scheduler to execute their programs. Accordingly, PCs were slowly reducing the prominence of the mainframes while they are dramatically getting more powerful [1].

Even though the processing speeds of the recent PCs have improved increasingly, they cannot keep pace with the size increases in remote sensing datasets due to higher resolution sensors and larger study regions. A single workstation may not be adequate to process these large datasets, but there are progressively more powerful parallel computers available for use within the scientific community [2].

The evolutions in the computing industry made a shift in the design philosophy of microprocessors concerning the use of multiple processing cores. This change lets a single device to execute multiple instructions at the same time [3].

These evolutions illustrate an upper limit in core frequency and expose a trend in parallel approach of computing rather than of serial to overcome this limit. Instead of having one powerful processing unit as in earlier processors, modern processors are composed of multiple processing elements (multicores) [1]. The architecture of multiprocessor with multicores for each, has clearly delivered new opportunities for improving performance of Computer simulations [4]. Since 2001, the commercial general-purpose multicore processors are available. They indicate wide gaps in time among the outlines of the first general-purpose multicore processor by IBM in 2001, the subsequent releases by other general-purpose processor manufacturers, and the eventual mainstream acceptance of general-purpose multi-core processors in 2005. All of which are signaled by the release of multicore processors by Intel and AMD [5].

In the parallel computing era, parallel programming prototypes are quite challenging and emerging topic. The recognition of the computing applications that require enormous computing requirements lead to the confrontation of the obstacles regarding the development of the efficient programming models. These programming models bridge the gap between the hardware ability to perform the computations and the software ability to support the performance for those applications. Therefore, a better programming model is needed to facilitate easy development and on the other hand to port a high performance [6].

For a better utilization, the programs should be optimized to exploit all of the available processing elements in a processor. Therefore, processing intensive parts of older programs should be optimized for multi-core processors. Replacing hardware with a faster one to speed up the time seems to be over, for that reason the parallelization of the code is necessary to utilize the power of the future computing. Generally this task is insignificant and needs a convinced amount of knowledge of the processor architecture and parallelization techniques [1].

A parallel system is a combination of a parallel algorithm and a machine running it, and both of them influence with several variables. Parallel algorithms can be specified using a wide range of models and paradigms [7]. With the arrival of multicore processors, parallel programming for shared memory architecture is entering the mainstream. However, parallel pro-

- *Assist. Prof. Dr. Subhi R. M. Zeebaree, head of IT department at Akre Technical College, Duhok Polytechnic Univeristy, Kurdistan, Iraq, Tel: +964-750-433-2160. E-mail: subhi.rafeeq@dpu.edu.krd*
- *Karwan Jacksi is currently pursuing PhD degree program in Computer Science at University of Zakho, Kurdistan, Iraq and Eastern Mediterranean Univerisy, Cyprus. Tel: +90-533-852-8257. Email: Karwan.Jacksi@uoz.ac*

gramming at large is considered to be tough. A key solution to this problem sets in the use of libraries which hide parallelism from the programmer. This has been experienced in the field of numeric for a long time, where best quality libraries that encapsulate parallelism are available [8].

2 Related Works

In 2010, authors of [9] worked on a distributed memory system that depends on client/servers principles. They rely on a network with any number of nodes; one of them was a client and the others were servers. Their algorithms were capable of calculating the Started, Consumed, and Terminated CPU times, total execution time and CPU usage of servers and CPU, and total execution times for the Client.

In 2011, writers of [10] presented a simple analytical model for studying the speed up of shared-memory programs on multi-core systems. The proposed model derived the speed-up and speed-up loss from data dependency and memory overhead for various configurations of threads, cores and memory access policies in uniform memory access systems. Their analysis presented that speed-up loss conquered by memory contention, especially for problems with larger sizes.

In 2011, the authors of [11] proposed an efficient implementation of parallel programming OpenMP on embedded multi-core platform. For embedded multi-core platform, there are limited memory resources. Incorporating limited memory hierarchy into OpenMP was challenging. Their work offered the solution to extend OpenMP custom directive to expand performance.

TABLE 1
Average Values of CPU and Total Execution times for eight Sorting-Algorithms with (110000, 275000 and 540000) Elements.

Array Size (Elements)	Number of CPU's	Real Consumed CPU Time (second)	Thread Total Execution Time (second)
110000	1	104.728	122.728
	2	111.519	113.511
	3	100.785	101.784
	4	84.047	84.831
	5	83.878	84.102
	6	82.928	83.897
	7	82.053	82.95
	8	69.176	70
275000	1	671.829	801.815
	2	733.178	735.056
	3	668.052	668.714
	4	564.13	565.116
	5	563.568	564.555
	6	562.758	563.729
	7	562.05	563.1
	8	476.935	478
540000	1	2623.231	3080.177
	2	2815.755	2818.722
	3	2549.507	2550.438
	4	2124.85	2125.766
	5	2123.24	2124.172
	6	2122.903	2123.789
	7	2121.15	2122.022
	8	1788.163	1789.996

In 2011, essayists in [12] studied implementation for matrix multiplication on a cluster of computers and developed a performance model for their implementation. Their experiment is based on the master – slave model in homogenous computers to compute the performance of experiment. They compute the execution time for many examples to compute the speed up. The developed performance model has been checked and it has been shown that the parallel model is faster than the serial model and the computation time was reduced.

3 The Proposed Shared Memory System and Parallel Processing Approach

The hardware of the system consists of a computer with a multi-core processor, containing a general program that control the overall tasks. Since all of the processors are in a single computer, hence building a network to connect them is useless. The operation has been tested on a system with the following parameters: Intel core i7 with eight Logical-Processors (Logical CPUs) designed in a single chip, frequency of 2.0 GHz for each processor, 4 GB of RAM. The secondary storage of the computer for the application case study contains the original data of related case-study that should be sent to the Shared Memory CPUs, and the receiving results that calculated by

TABLE 2
MAXIMUM VALUES OF CPU AND TOTAL EXECUTION TIMES FOR EIGHT SORTING-ALGORITHMS WITH (110000, 275000 AND 540000) ELEMENTS.

Array Size (Elements)	Number of CPU's	Real Consumed CPU Time (second)	Thread Total Execution Time (second)
110000	1	104.143	100.219
	2	78.205	80.008
	3	42.697	43.342
	4	33.821	34.037
	5	24.053	24.602
	6	18.647	19.168
	7	14.968	15.564
	8	13.001	13.603
275000	1	641.367	771.935
	2	519.339	520.939
	3	294.466	295.045
	4	242.347	242.808
	5	188.064	188.685
	6	151.715	152.092
	7	126.375	126.934
	8	117.276	118.005
540000	1	2563.996	3020.163
	2	2078.321	2080.197
	3	1194.69	1195.294
	4	947.027	948.544
	5	785.282	786.433
	6	644.955	645.917
	7	547.071	547.887
	8	533.988	534.92

them. Each Logical Processor will function as a separate PC which will has a fragment of the cache-memory address space and the rights of reading and storing data on the Shared Memory. Depending on the environment of the under test case study and situations of each state during that case study, the selection of Logical Processes which will participate with the task will differ from one state to another. These states can use the Logical Processes according to the load division.

The load is shared among all the system's Logical Processes without restrictions related with the load in this approach. The data structures and size of the segments will have no effect on the treating with these segments, because the aim is to get as much as unbalanced distribution of the load as possible.

The structures of the case study (e.g. programming styles) might effect on CPU utilization and its context switching related with number dispatching from ready to running state. This might cause ineffective use of Logical Processors capabilities which may yield unpredicted results that should be given using Parallel Processing methods. Therefore, for a system containing eight Logical Processors, it can treats with unbalanced load-division as (1, 2, 3, 4, 5, 6, 7 or 8) Logical Processors participate with the task.

The second phase of the proposed algorithm is load division stage. This stage is responsible of distributing the load among the Logical Processers of the system depending on the used case-study. These responsibilities are as bellow:

- Deciding how many Logical Processers will be participating with the task according to the specified load for it.
- Creating the arrays of related data (as data-files) in order to be accessible by these Logical Processers.
- Monitoring all related Logical Processers in case if they calculated any results.
- Receiving the calculated results from these Logical Processers and store them.

4 SORTING TECHNIQUES CASE STUDY (UNBALANCED LOAD DIVISION)

The unbalanced load-division using eight *sorting algorithms, insertion, selection, quick, heap, shaker, shell, Comb, merge,* are dealt with the case study. For that reason, this test will describe unbalanced load-division, the load will be distributed among all CPUs in unbalanced form.

The results of this case-study consist of two main groups, which are the (average and maximum)-values of (real consumed CPU time and thread total execution time). Number of elements for unsorted arrays are (110000, 275000 and 540000) elements for each array.

In this case-study, the system have (8) CPUs (i.e. Logical Processors), the groups of CPUs that should participate with the task are (1, 2, 3, 4, 5, 6, 7 and 8) CPUs depending on the system. Results of average values are shown in Table (1) and those of maximum values are shown in Table (2).

The results that illustrated in these tables are plotted to be suitable graphs as shown in Figures (1 to 4). Figures (1 and 2) represent the average values of real consumed CPU time and average thread total execution time. Figures (3 and 4) represent the maximum values of real consumed CPU time and thread total execution time.

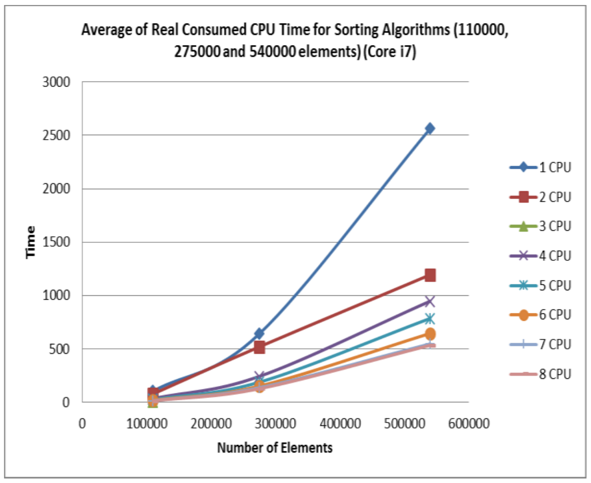

Fig. 1. Average Values of CPU Execution time for eight Sorting-Algorithms with (110000, 275000 and 540000) Elements.

ISSN (ONLINE): 2349-7084

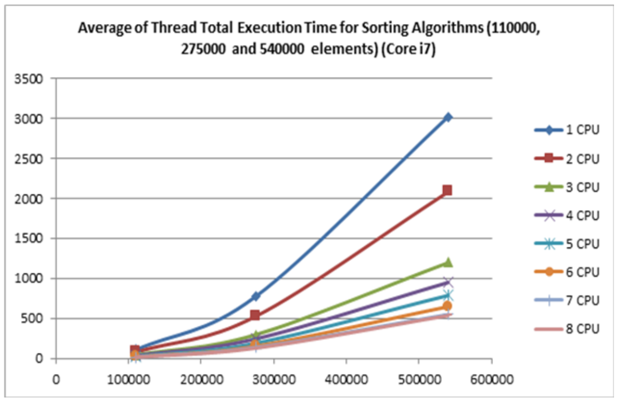

Fig. 2. Average Values of Total Execution time for eight Sorting-Algorithms with (110000, 275000 and 540000) Elements.

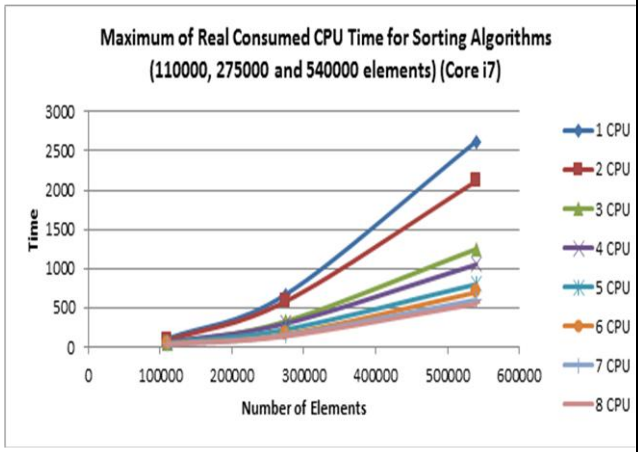

Fig. 3. Maximum Values of CPU time for eight Sorting-Algorithms with (110000, 275000 and 540000) Elements.

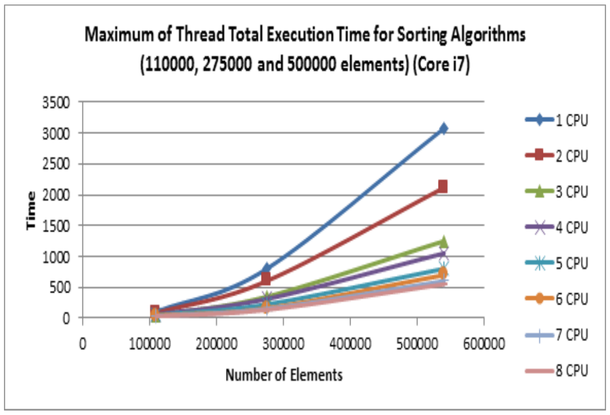

Fig. 4. Maximum Values of Total Execution time for eight Sorting-Algorithms with (110000, 275000 and 540000) Elements.

5 Discussion and Conclusion of the Obtained Results

Figures (1 and 3) illustrate the effects of parallel processing approach on the (average and maximum) values of (real consumed CPU time and thread total execution time) when expending a system of 8-CPUs. According to the principles of the parallel processing, the elapsed execution time should be reduced with increasing number of participated CPUs for solving the same problem. This has been clearly illustrated in all the results.

The timing-values with 1-CPU has been determined depending on parallel execution of the sorting algorithms. Keeping in mind that we can force the CPU to execute its processes in serial manner. Hence, these results take a lesser amount of time less than of those of serial. This is because all of the threads will go into the system in parallel mode which will avoid losing in time, while the serial entering will take more time.

When only one CPU is used with the average value of thread, CPU execution time becomes 2563.996 second for 540000 elements. Whereas CPU execution time decreases to 533.988 second when using eight CPUs. This means that the execution speed became greater 4.8 times. Hence the speed according to total execution time will greater 4.756 times when using eight CPUs instead of one CPU.

In the same way we can conclude that, when using only one CPU with maximum value of CPU execution time of 2623.231 seconds for 540000 elements, while it becomes 550.163 seconds when using eight CPUs, which means that the execution speed is again greater 4.768 times. Hence the speed according to total execution time will be greater 5.58 times when using eight CPUs instead of one CPU.

This is the real activity of parallel processing approaches when using heavy load via shared memory systems by forcing and controlling the direction of processes execution among the available logical processors within the computer system.

The little differences of unexpected results occurred due to the differences among the structures of the depended sorting-algorithms which cause more unbalancing of the load-division, and processing-style of treating with elements of sorting arrays need very high swapping operation with complex-looping system, adding to the environment of the depended system may cause appearing such unexpected results. However, the sorting algorithms may cause unexpected timing-results in some parallel-processing-states with shared-memory systems.

As a reason of that, there is no previous work depended on the exact structure as the same of our proposed parallel processing system with the same algorithms and the same hardware properties, it will be difficult to produce fair comparison. Hence, this paper produced an efficient parallel processing system handling any size of unbalanced load division to be processed among the processors of the system with no restrictions of the used OS and which programming language to be depended.

When we used C++ package language for programming the algorithms it is not meaning that it is the unique language that should be used. Also, when we depended on a shared memory system with eight logical processors, it is not does not mean that the programmer or the user cannot depend on other systems with number of processors less or greater than eight. This system can be applied automatically on any system regardless of the number of its processors, and the algorithm is capable of detecting number of the processor directly. There are no restrictions on the type of the OS.

REFERENCES

[1] Venkatesan V. "Exploring Shared Memory and Hybrid Parallelization Strategies for Image Processing", M.Sc. thesis, University of Houston, 2008.

[2] Phillips R.D., Watson, L.T., and Wynne, R.H. "Hybrid Image Classification Using a Shared Memory Parallel Algorithm", Pecora 16th conference on Global Priorities in Land Remote Sensing, Sioux Falls, South Dakota October 23- 27, 2005.

[3] Grant R.E. "Analysis and Improvement of Performance and Power Consumption of Chip Multi-Threading SMP Architectures", M.Sc. thesis, Queen's University, Canada, 2007.

[4] Byrd J. M. R. "Parallel Markov Chain Monte Carlo", Ph.D. thesis, The University of Warwick, 2010.

[5] Tam D. "Operating System Management of Shared Caches on Multicore Processors", Ph.D. dissertation, University of Toronto, 2010.

[6] Ravela S. C. "Comparison of Shared memory based parallel programming models", M.Sc. thesis, Blekinge Institute of Technology, 2010.

[7] Naiouf M. R. " Parallel processing. Dynamic Load Balance in Sorting Algorithms" Journal of Computer Science & Technology Vol. 4, No. 3, 2004.

[8] Suess M., and Leopold C. "Implementing Data-Parallel Patterns for Shared Memory with OpenMP", NIC Series, Vol. 38, pp. 203-210, 2007.

[9] Yaseen N. "Diagnostic Approach for Improving the Implementation of Parallel Processing Operations", M.Sc. thesis, University of Duhok, 2010.

[10] Tudor B. M. and Teo Y. M. "A Practical Approach for Performance Analysis of Shared-Memory Programs", 25th IEEE International Parallel and Distributed Processing Symposium IPDPS, Anchorage (Alaska) USA, 2011.

[11] Qing W., Zhenzhou J. I., and Tao L. "Efficient OpenMP Extension for Embedded Multicore Platform", Journal of Frontiers of Computer Science and Technology, Vol. 5, No. 1, 2011.

[12] Abu ElEnin S. and Abu ElSoud M. "Evaluation of Matrix Multiplication on an MPI Cluster", International Journal of Electric & Computer Sciences IJECS Vol: 11 No: 01, pp. 59-66, 2011.

Dr. Subhi Rafeeq Mohammed Zeebaree received his BSc, MSc and PhD degrees from University of Technology-Baghdad-Iraq at 1990, 1995 and 2006 respectively. He has a PhD. In Computer Engineering, he is an Assistant Professor since 2012. He started teaching and supervising post graduate courses (PhD and MSc) since 2007 and until now. Ten of his MSc students completed their studding and got MSc degree. Now there are number of PhD students and MSc students under his supervising inside and outside of Iraq. He published many papers and now there are more papers under publishing he did them with his PhD and MSc students. This paper is one of them, which done together with his PhD student Karwan Jacksi. He is the Head of IT Dept. at Akre Technical College, Duhok Polytechnic University (DPU). He is the Chairman of Scientific Affairs and Research Advising Committee of DPU. He is a member of Kurdistan Main-Board of computer postgraduate courses.

Karwan Jacksi received his BSc in Computer Science from University of Duhok/Iraq in 2007, and MSc in Computer Science from Uppsala University/Sweden in 2011. He is now a Split-site PhD student in Computer Science at University of Zakho/Iraq and Eastern Mediterranean University/Cyprus under the supervision of both Dr. Subhi Zeebaree (main supervisor) and Nazife Dimililer (co-supervisor). He is one of the staffs of Computer Science department at University of Zakho.