www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 6 Issue 6 June 2017, Page No. 21756-21762
Index Copernicus value (2015): 58.10 DOI: 10.18535/ijecs/v6i6.35

Deterministic Routing Algorithm for Shared Memory Processing (SMP)

Soha S. Zaghloul, PhD[1], Ashwag Homod Alotebi[2], Noura Saleh AL Maghrabi[3]
Asma Abdurahman Almulifi[4], Hend Ibrahim Alshaya[5]
[1]King Saud University, Computer Science Dept.,
Saudi Arabia, Riyadh
smekki@ksu.edu.sa
[2]King Saud University, Computer Science Dept.,
Saudi Arabia, Riyadh
435203998@student.ksu.edu.sa
[3] King Saud University, Computer Science Dept.,
Saudi Arabia, Riyadh
435203944@student.ksu.edu.sa
[4]King Saud University, Computer Science Dept.,
Saudi Arabia, Riyadh
436203795@student.ksu.edu.sa
[5]King Saud University, Computer Science Dept.,
Saudi Arabia, Riyadh
43520400@student.ksu.edu.sa

Abstract: ***Parallel computing involves the simultaneous deployment of various resources and computers to solve computational problems by using multiple processors. The most common parallel computing model is the shared memory processing (SMP) model. In this model, a number of identical processors communicate with each other by using one large logical shared memory with the same amount of access time for the entire memory area. The parallel programming performance is affected by the run time, which is in turn affected by the number of processors and the size of the problem. Therefore, this paper presents an application of XY deterministic routing in an SMP system based on a 4x4 2D mesh topology network with two cores and two threads per core. The sequential run time and the related parallel run time for each thread were measured.***

The sequential time that must be achieved to compensate for the warm-up overhead time consumed by the Java Virtual Machine (JVM) was at least 60 seconds. The results revealed that the achieved sequential time was equal to 69.057 seconds, whereas the achieved parallel times for threads 1 and 2 of the first core and threads 1 and 2 of the second core were 70.066, 68.112, 44.869, and 42.412 seconds, respectively. On the basis of the degree of parallelism, the parallel programming performance was evaluated in terms of speedup and efficiency. The performance evaluation results demonstrated that an increase in the degree of parallelism results in faster speed up and decreased efficiency.

Keywords: Deterministic routing algorithm, Sequential computing, Parallel computing, SMP.

1. Introduction

The continuous development of computer architectures [1] and the growing use of modern systems are increasing the need for fast computers that can implement numerous tasks in a short period of time. In traditional serial com- putting, tasks are performed sequentially over a long duration. In practice, this time consumption problem can be solved by performing various operations simultaneously, which has led to the development of another type of computation called parallel computing in which several tasks are executed simultaneously [2, 3]. Figure 1 shows a comparison of serial and parallel computing.

Parallel computing refers to the simultaneous deployment of various re- sources and computers to solve single computational problems by using multi- plea processing entities. Such entities may be multiprocessor systems composed of multiple processors that are linked via bus or switch networks in a sin- gel machine. Alternatively, they may be multicomputer systems composed of various independent computers that are connected by computer or telecommunication networks. A control/coordination mechanism is also deployed in parallel computing [2, 4].

The concept of parallel computing is based on dividing the problem to be solved into various discrete parts through a divide-and-conquer method, such that the resultant parts can be executed concurrently and independently [3]. Each part is separated into a set of instructions that are operated simultaneously on multiple processes, thus enhancing speedup and decreasing

DOI: 10.18535/ijecs/v6i6.35

operation time and cost. Parallel computing also allows for the use of non-local resources and can be applied to solving larger and more complicated and time- critical problems, which typically impose high requirements in terms of both processing power and memory. Other benefits of parallel computing include the efficient use of underlying parallel hardware, load distribution, fault tolerance, synchronization, and communication [2, 5].

These benefits have increased the deployment of this computing paradigm in various industrial, commercial, scientific, and engineering fields, such as image processing [6, 7], differential evolution [8-10], big data, data mining, databases, and computer science mathematics. In addition, various scientific and industrial studies have taken ad- vantage of large, complicated computation methods that would take years to reach completion without parallel computers [3].

This paper sought to presents an application of XY deterministic routing in shared memory processing based on a 4x4 2D mesh topology network with two cores and two threads per core. Section II of this paper introduces gen- earl background information about parallel programming models, with a focus on shared memory models, and introduces the main concepts of deterministic and adaptive routing algorithms. The primary metrics that are used to evaluate parallel processing models are described in Section III, and Section IV explains the experiments in detail. Section V then analyzes and discusses the results, and the final section concludes the paper and presents future research opportunities.

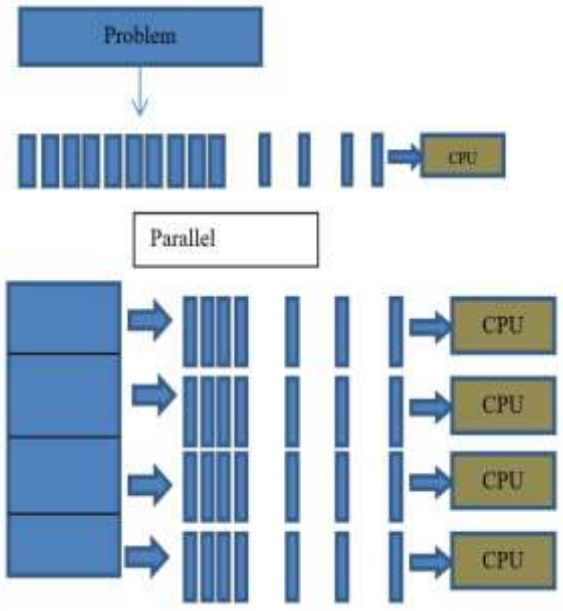

Figure 1: Comparison of serial and parallel computing [3]

2. Background

2.1. Parallel Programming Models

The multiprocessors that are used in parallel architectures can be categorized into three types on the basis of their communication models: shared memory, distributed memory, and hybrid memory [4].

A. Shared Memory Paradigm

In the shared memory paradigm, all processors communicate with each other by using one large, common logical shared memory over a high-speed network and treat this memory as a global address space. A system based on this paradigm is also referred to as a symmetric multiprocessor system, because its processors are identical and the access time is the same for the entire memory area [3, 4, 11]. The shared memory paradigm offers specific advantages regarding programmability, because it has the same memory organization as that of sequential programming models.

Therefore, there is no need to consider the details of data partitioning, communication, migration, and distribution. This paradigm also prevents the multiplicity of data items, and programmers have a low level of individual responsibility in this model. However, no high-performing, practical shared-memory machine exists, because there is no scalable shared memory that permits numerous processors to access various locations at the same time. Moreover, the use of a single shared memory prevents processors from high-speed access. There are also many hardware requirements, high costs, and high complexity because of the presence of deadlocks during application development [4, 5, 12].

B. Distributed Memory Paradigm

The distributed memory paradigm involves several processing elements called nodes that aggregate in clusters. It also involves an interconnection network, in which the nodes are connected to one another and data transmission among nodes is supported. Each node is an independent module that is composed of a small local memory unit and processor and simultaneously communicates private data to other processors. In this paradigm, for programming purposes, messages are transmitted among nodes by using a message-passing model [13, 14].

Such programming is based on the exchange of send/receive communication primitives among several processors that communicate over the network. A system based on the distributed memory paradigm is also called an asymmetric multiprocessor system, because it includes different types of processors, such as a front-end processor that acts as the access point to all other back-end processors and controls the distribution of data [4, 12, 15].

Programmers can apply the distributed memory paradigm to achieve eve- efficient performance, because it optimizes programs so that they can benefit from locality by saving commonly deployed data in local memory and de- crease remote memory access. Furthermore, the relevant message-passing mod- ells have few hardware requirements, low complexity, and low costs. However, this paradigm requires significant effort from individual programmers, who must take responsibility for managing all of the details concerning communication, task scheduling, and data distribution. In addition, message-passing models can easily give rise to deadlocks during the communication process and high communication overheads [4, 5, and 12].

C. Hybrid Memory Paradigm

To combine the programming simplicity of shared memory architectures with the performance of distributed memory architectures, a hybrid type, also referred to as a distributed shared memory architecture, has been developed. Such a system is constructed on top of a message-passing distributed architecture and presents an exposed interface to enable a shared architecture. In addition, it allows for the programming of multiple computers to be simplified by simulating a shared address space. Within such a system, there exists a communication library that is responsible for the mapping from remote memory accesses to message passing, thus avoiding the need for programmers to participate in the message communication process. However, hiding the memory access locality from programmers may result in reduced performance and inefficient memory access [4, 12, and 16].

This paper focuses on SMP programming paradigms based on the concept of parallelism. Such paradigms can be represented with several programming models, as presented in the following subsection.

2.2 Shared Memory Programming Models

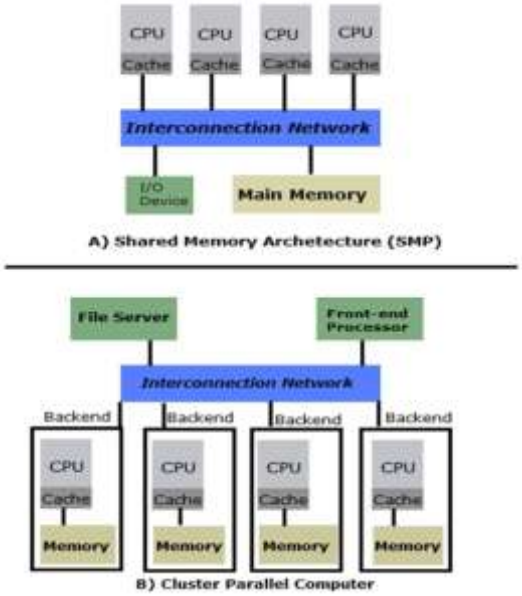

Figure 2: The difference between shared memory (SMP) and distributed memory architectures

The two main types of models used to represent shared memory computer paradigms are Java threads and Open Message passing (OpenMP). These models differ in their syntax, semantics, levels of abstraction, and principles of parallelism. Java is a programming language that supports parallelism in the form of threads. Parallel SMP programs written in Java are primarily implemented through thread execution, wherein each thread is an independent control flow that uses a global address space to share data with other threads [15].

OpenMP is a programming model and application program interface de- signed for shared memory paradigms. It supports parallelism by means of a group of parallel directives, environment variables, and run-time library rout- tines. It is usually presented in the C, C++, and FORTRAN languages. This model has a high level of abstraction, thus simplifying the design of parallel applications from the developer perspective. However, OpenMP is not deployed as a thread model because of its low flexibility, and it is not treated as a standard [5, 17].

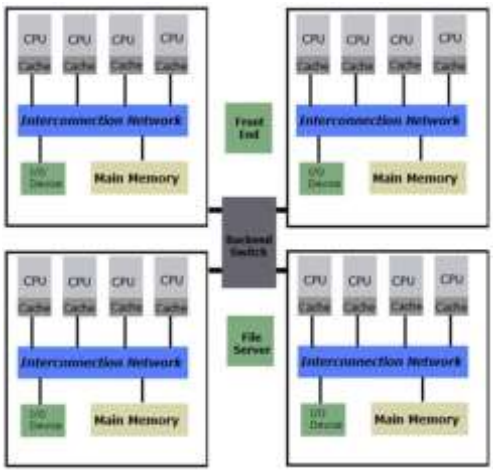

Figure 3: Hybrid architecture

3. Routing Algorithms

In practice, routing is the process of transferring data from a specific source node to a target destination node by using a well- defined strategy and routing path [19]. In other words, a routing algorithm specifies the path that must be followed by a data packet traveling between these two nodes. There are various types of routing algorithms available that differ in their main characteristics. On the basis of the method used to select a path, routing algorithms can be classified into two main types: deterministic and adaptive routing algorithms. These two types of algorithms differ in their dependence on the network conditions and the number of possible paths that are determined between a pair of source and destination nodes.

A. Deterministic and Adaptive Routing

In a deterministic routing algorithm, the routing path (typically the shortest path) is specified by the receiver and sender. Because this path is fixed for the same network correspondents, there is a specific, unique route for each message regardless of the network con- dictions. However, when a packet is transmitted over a congested network, the entire network fails because such an algorithm does not consider the network conditions [18, 19].

The most common type of deterministic routing algorithm is an XY algorithm, which is a dimension-order routing algorithm. In such an algorithm, packets are initially routed in the horizontal direction (x axis) until the tar- get column is reached, after which they are routed in the vertical direction (y axis) to their destination. Thus, the address of each router represents coordinates. In practice, such an algorithm is appropriate for a network with a torus or mesh topology. The operating principle of an XY routing

DOI: 10.18535/ijecs/v6i6.35

algorithm is illustrated in the figure 4 [19].

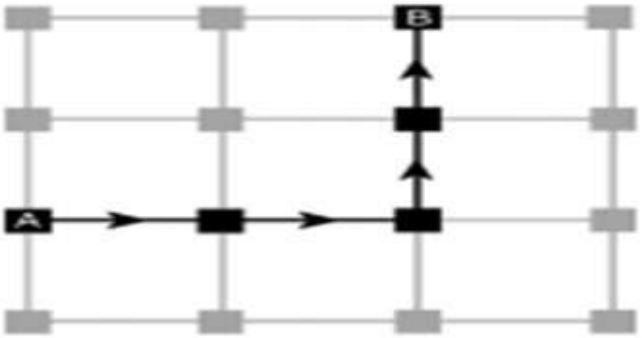

Figure 4: XY routing algorithm [19].

In an adaptive routing algorithm, data may be transmitted between two nodes over several different paths, depending on the network conditions and load, information about available output channels, and traffic conditions. In this type of routing algorithm, each router is aware of the traffic situation in the network and adjusts its routing accordingly. In practice, adaptive routing algorithms integrate network congestion information into their decisions about routing and route messages around detected regions of congestion. Moreover, this type of routing offers enhanced data flow in a network; however, it results in more complex nodes [18-21].

The most common type of adaptive routing algorithm is an even-odd turn model, which restricts the priority of data in specific locations within the network topology, on the basis of the evenness or oddness of the column in which a given packet is located. This in turn provides many livelock and deadlock-free paths for packets [20].

B. Pseudo code for XY routing in a two-dimensional mesh

Figure 5 presents the pseudo code for an XY routing algorithm.

```
1) BEGIN
2) Set Destination Router Co-ordinate (Dx, Dy), Source
     Router Co-ordinate (Sx, Sy) and Current Router Co-
     ordinate (Cx, Cy)
3) IF Dx greater than Cx
        a. RETURN East Port
4) ELSEIF Dx less than Cx
        a. RETURN West Port
5) ELSEIF Dx equals to Cx
        a. IF Dy less than Cy
                i. RETURN South Port
        b. ELSEIF Dy greater than Cy
                i. RETURN North Port
        c. ELSEIF Dy equals to Cy
                i. RETURN Current Router
        d. END
6) END
7) IF current router is Destination router
        a. Reached Destination
8) ELSE
        a. GOTO STEP 3
9) END
10) END
```

Figure 5: The pseudo code for an XY routing algorithm

3 Performance Metrics

The efficiency of parallel processing for a specific problem can be practically evaluated by using the following metrics: speedup, efficiency, and Amdahl's

3.1 Speedup

The speedup factor is the ratio of the execution time required for one processor performing sequential computations to solve a problem of size N Tseq to the execution time required for K processors performing parallel computations T-par to solve the same problem. Thus, it can be expressed as follows [2, 22]:

Speedup (N, K) = (T seq (N, 1))/ (T par (N, K)). (1) Speedup is directly dependent on the number of processors until a saturation point is reached. After this point, the addition of more processors does not
Result in more efficient performance. Speedup is also based on the best sequential program for a one-processor system, whereas the underlying programs for different parallel implementations might be different

3.2 Efficiency

Efficiency is a measure of how much speedup is provided by the addition of another processor and is expressed as follows [4, 22]:

E(N, K) = speedup(N, K)/K (2)
According to this relation, the efficiency is inversely correlated with the
Number of processors [4].

3.3 Amdahl's Law

The reduction in efficiency that occurs with an increasing number of processors is related to the limit of parallel performance. After a specific threshold number of processors, the addition of more processors cannot offer further improvement in performance and consequently results in performance degradation because of the reduction in the time saved by additional task division and the increase in communication overheads. Therefore, Amdahl's law is an efficient method of assessing the effectiveness of a parallel program for a given problem. The speedup of a parallel program is expressed as follows [4, 22]:

Speedup (N, K) = 1/(s + p/n) < 1/s (3)

Where p represents the fraction of parallel code and s is equal to 1-p and represents the fraction of serial code. Thus, the maximum possible speedup cannot be more than one second [4]

4 Experiments

4.1 Hardware and Software Requirements

The design stage of this study was conducted on a laptop

(Intel(R) Core(TM) i7-5500U CPU @2.40 GHz, hyper-threaded; 2 threads per core, with a total of two cores and four threads), running the Windows 10 operating system. The Java programming language was used for implementation, in combination with the Net Beans IDE 8.2 software.

4.2 Experimental Stages

In the current study, a network with a 4x4 2D mesh topology was deployed; in which each node had unique xy coordinates described by the following four arrays:

X1[] = 1, 0, 2, 2, 0, 3, 2, 1;
y1 [] = 1, 2, 1, 0, 1, 3, 2, 0;
X2[] = 2, 1, 3, 3, 1, 2, 1, 2;
y2[] = 2,3,1,2,2,3,1,1; .

A random function was used to generate data packets. An XY routing algorithm was used to determine the route that each packet should follow. The Java language has no global concept dictating that each variable should belong to a specific class or method. Thus, access to shared data should be explicitly managed among threads because they share several variables in the SMP architecture. This problem was solved by declaring a specific public class including variables that must be shared. This declared class must, in turn, be accessed by other program classes, which may require the exchange of data values among threads.

Shared data can have a position class (x, y), whereas private data files are generated randomly. Thus, four threads were created in this study, and each acted as a sender. Each sender took a text file as input and read the data, which contained the header source, destination position, and data packet. The following points must be considered when comparing the performance of different versions of a program, such as sequential and parallel versions:

The programs must be executed on the same computer All parallel computer processors must possess the same hardware characteristics, such as clock speed The same key algorithm must be incorporated into both programs

The main factor that affects a programs performance is its run time, regardless of other algorithm and hardware factors. The run time (T) of any program is the time required by that program to compute an answer to a problem. This value depends on the number of processors (K) and the problem size (N), which is defined as the number of computations required to determine the result. Thus, T is a function of both K and N: T (N, K). There- fore, all programs were created by preparing various input datasets covering a range of problem sizes N. The smallest problem size such that Tseq (N, 1)60 seconds was then selected. This amount of time was considered sufficient to compensate for the warm-up overhead time consumed by the JVM.

The sequential programming implementation was executed ten times for each input dataset. The shortest run time was subsequently determined on the basis of the measured value of Tseq (N,1), following a Gaussian distribution. This procedure resulted in a number of packets equal to 350000 for each dataset. The parallel programming implementation was also executed ten times for each value of K, from 2 to the maximum number of available process-sores. Finally, both the speedup and efficiency were plotted against the number of processors (K), and the run time (T) was plotted against the problem size (N) for each value of K. Amdahl's law was then computed.

5 Results and Analysis

For the sequential programming implementation, the achieved Tseq (N, 1) was equal to 69.057 seconds. All threads started at the same time in the parallel programming implementation. However, the results revealed a dramatic decrease in the execution time for three threads.

Table 1 Measured metrics for different numbers of threads in the parallel programming implementation

Ts	Tp	Degree of	Spe	Effi	Amdah	Amdahl's	Efficien
69	68	2	1.01	1.01	1.62	0.81	
69	44	3	1.53	0.76	2.05	0.68	
69	42	4	1.62	0.81	2.37	0.59	

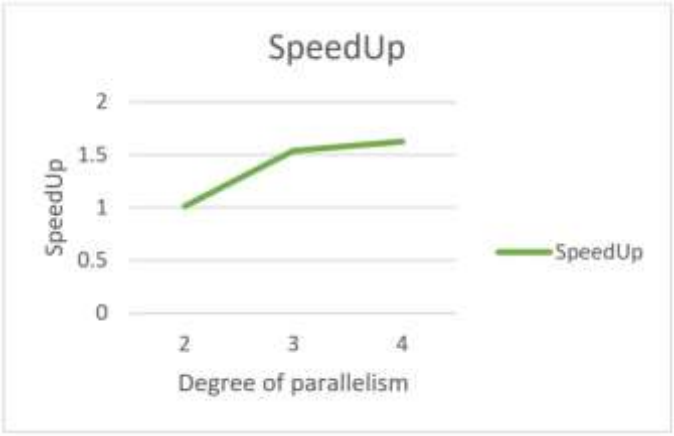

Figure 6: Speedup versus the degree of parallelism

The degree of parallelism represents the number of cores, which was equal to two in this study. The achieved Tpar value for each thread was as follows: Tpar (core1,thread1)=70.066 seconds, Tpar (core1,thread2)=68.112 seconds, Tpar (core2,thread1)=44.869 seconds and Tpar (core2,thread2)=42.412 sec- nods. The highest run time, which was associated with the first thread in the first core, was used to measure the parallel overhead. The computed speedup and efficiency values of the parallel programming implementation with the addition of the other three threads are presented in the table 1.

The following figures illustrate how the speedup and efficiency are related to the degree parallelism. The selected number of packets was 350000 because this value was the smallest number for which a sequential time of Tseq (N, 1)

DOI: 10.18535/ijecs/v6i6.35

=60 seconds. Was achieved.
As seen from the figure 6, there is a direct relationship between the degree of parallelism and the resultant speedup value.

Moreover, the figure 7 illustrates that when the degree of parallelism is equal to two, the resultant efficiency is one, because the parallel time is similar to the sequential time. However, this cannot be considered the optimal case. The figures 8 and 9 illustrate how the Amdahl speedup and efficiency are related to the degree of parallelism.

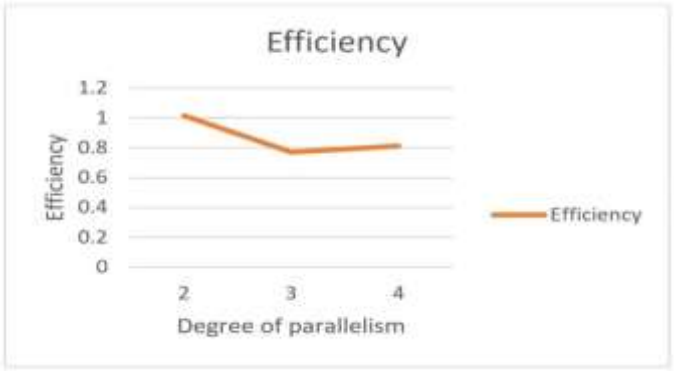

Figure 7: Efficiency versus the degree of parallelism

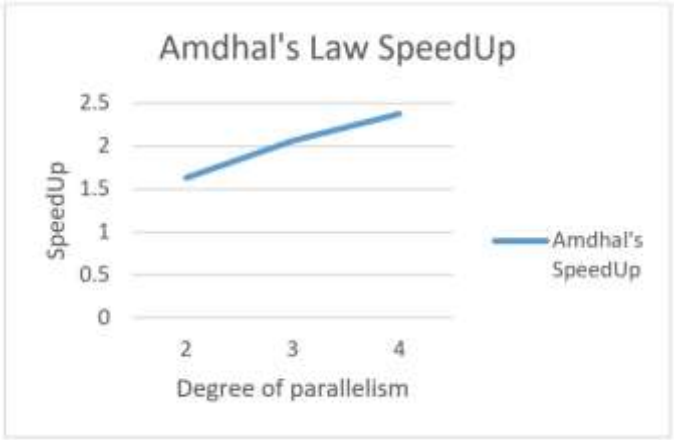

Figure 8: Amdahl Speedup versus the degree of parallelism**.**

As seen from figure 8, there is a direct linear relationship between the Amdahl speedup and the degree of parallelism. Furthermore, figure 9 demonstrates that there is an inverse linear relation- ship between the Amdahl efficiency and the degree of parallelism. The scalability is defined as the number of packets at which the system failed (3500000) divided by the number of packets used in the initial experiments (350000), which is equal to 10. As shown in figure 10, as the number of packets increases, the run time

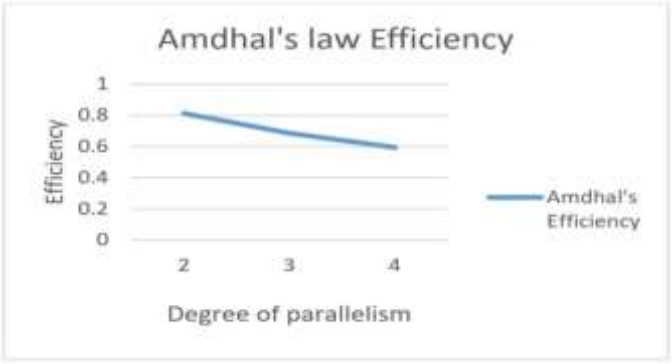

Figure 9: Amdahl efficiency versus the degree of parallelism

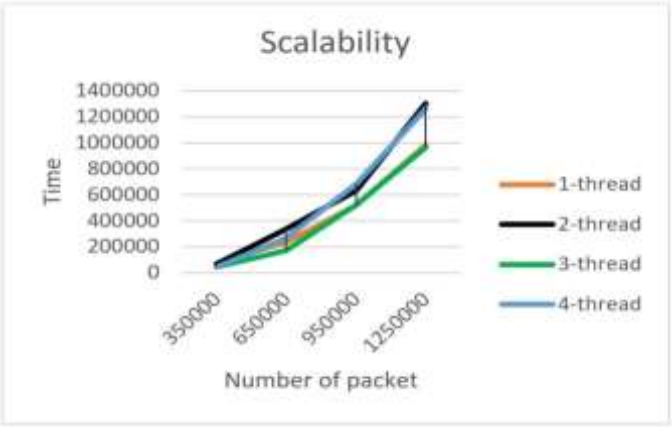

Figure 10: Scalability: time versus number of packets

6 Conclusions and Future Work

This paper presents the application of XY deterministic routing in an SMP system based on a 4x4 2D mesh topology network with two cores and two threads per core. A public class including variables to be shared was declared to explicitly manage access to the shared data among the threads. Both sequential and parallel programming implementations were considered, each was executed ten times for each prepared input dataset. The run time of a program, which is a function of the number of processors and the problem size, has a major effect on performance. According to the findings of this study, the sequential time that must be achieved to compensate

for the warm-up overhead time consumed by the JVM should be equal to or greater than 60 seconds. Here, the achieved Tseq (N,1) was 69.057 seconds, whereas the achieved Tpar values for threads 1 and 2 of the first core and threads 1 and 2 of the second core were 70.066, 68.112, 44.869, and 42.412 seconds, respectively. The highest run time, which was associated with the first thread of the first core, was used to measure the parallel overhead.

This parallel programming performance evaluation revealed that speedup increases with an increasing degree of parallelism, whereas efficiency is inversely correlated with the degree of parallelism. This study may be enhanced

in the future by choosing the source and destination nodes randomly and by scaling the mesh size.

Acknowledgment

The authors would like to extend their sincere appreciation to the Deanship of Scientific Research at King Saud University for its funding this Research group NO (RG-1435-077).

References

[1] B. Bose, "Gaussian and EJ networks Some efficient interconnection topologies for parallel systems", 17th CSI International Symposium in Computer Architecture and Digital Systems (CADS), pp. XV-XV, 30-31 Oct. 2013

[2] S. Rastogi and H. Zaheer," Significance of parallel computation over serial computation", IEEE, 2016

[3] B. Barney, "Introduction to Parallel Computing", [online]:available at: https://computing.llnl.gov/tutorials/parallel_comp/#Whatis

[4] I. Singh, "Review on Parallel and Distributed Computing", Scholars Journal of Engineering and Technology (SJET), vol. 1, no. 4, pp. 218-225, 2013

[5] S. C. Ravela, Comparison of Shared memory based parallel programming models, thesis is submitted to the School of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, 2010

[6] I.A. Ansari, A. Pant and C. W., Ahn, "Robust and false positive free watermarking in IWT domain using SVD and ABC", Engineering Applications of Artificial Intelligence, vol. 49, pp. 114-125, 2016

[7] I.A. Ansari, A. Pant and C. W., Ahn, "SVD based fragile watermarking scheme for tamper localization and self-recovery", International Journal of Machine Learning and Cybernetics, pp.1-15, 2015

[8] H. Zaheer, M. Pant, S. Kumar, O. Monakhov, E. Monakhova and K. Deep, "A new guiding force strategy for differential evolution", International Journal of System Assurance Engineering and Management, pp. 1-14, 2015

[9] H. Zaheer and M. Pant, "A Differential Evolution Approach for Solving Integer Programming Problems", In Proceedings of Fourth International Conference on Soft Computing for Problem Solving, pp. 413-424, 2015

[10] H. Zaheer, M. Pant, S. Kumar and O. Monakhov, "A Novel Mutation Strategy for Differential Evolution", Problem Solving and Uncertainty Modeling through Optimization and Soft Computing Applications, vol. 20, 2016

[11] N. Manchanda and K. Anand, "Non-Uniform Memory Access (NUMA)", New York University, 2010

[12] Zaid Abdi Alkareem Alyasseri, "Survey of Parallel Computing with MATLAB", pp. 1-9, 2010

[13] J. P. Weiss, "Hybrid Computing: Advantages of Shared and Distributed Memory Combined", [online]: available at: https://www.comsol.com/blogs/hybrid-computing-advantages-shared-distributed-memory-combined/, 2014

[14] T. Rauber and G. Rnger, Chapter 2 Parallel Computer Architecture, Parallel Programming, Springer-Verlag Berlin Heidelberg, pp.9-103, 2013

[15] J. Monteiro, "non-uniform memory access (NUMA) architecture and multicomputers, Parallel and Distributed Computing", Department of Computer Science and Engineering (DEI) Instituto Superior Tecnico, 2011

[16] P. Kumar and K. Kumar, "Defining Hybrid Distributed Shared Memory Consistency Models on Unified Framework", international journal of enhanced research in science tech- nology and engineering, vol. 2, no 2, 2013

[17] E. Ajkunic, H. F. kic, E. Omerovic, K. Talic and N. Nosovic, "A Comparison of Five Parallel Programming Models for C++", MIPRO, pp. 2203-2207, 2012

[18] A. Ben Achballah and S. Ben Saoud, "A Survey of Network-On-Chip Tools, (IJACSA) International Journal of Advanced Computer Science and Applications", vol. 4, no. 9, 2013

[19] G. Adamu, P. Chejara and A. Baita Garko, "Review Of Deterministic Routing Algorithm For Network-On-Chip", 2nd international conference on science, technology and manage- ment, 2015

[20] D. Ouellet-Poulin, "Adaptive Routing Algorithms and Implementations", pp.1-4, 2010

[21] L. Shrivastava, G.S. Tomar and S. S. Bhadauria, "A Survey on Congestion Adaptive Routing Protocols for Mobile Ad-Hoc Networks", International Journal of Computer The- ory and Engineering, vol. 3, no. 2, pp. 189-196, April 2011

[22] J. Mathew and R. Vijayakumar, "The Performance of Parallel Algorithms by Amdahl's Law, Gustafson's Trend", Juby Mathew et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, vol. 2, no. 6, pp. 2796-2799, 2011