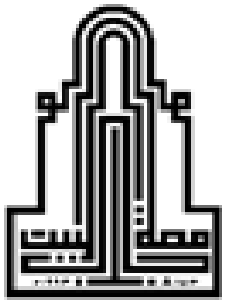


A Spiral Non-Contiguous Processor Allocation Algorithm for 2D Mesh-Connected Multicomputers

خوارزمية التخصيص غير المتجاور الحلقي في متعددات الحواسيب الشبكية ثنائية الأبعاد

By

Ibrahim Jum'a Alrawahna
(1320901012)

Supervisor
Prof. Ismail Ababneh

Co-supervisor
Dr. Saad Bani-Mohammad

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree of Science in Computer Science

Prince Hussein Bin Abdullah College for Information Technology
Al al-Bayt University

May, 2016

المنارة للاستشارات

www.manaraa.com

Dedication

I dedicate this work to my mother for her love, encouragement and support, she were the light in my path. Without her, nothing of this would have been possible.

Thank you for everything, I love you!

www.manaraa.com

Acknowledgments

In the name of Allah the Most Merciful

I offer my sincerest gratitude to my advisors, Prof. Ismail Ababaneh and Dr.Saad Bani-Mohammad, for their valuable contributions, knowledge, encouragement and helpful advices. As well as their vision that brought this work forward, for being there any time I knocked at their door. I wish them both more and more success and giving.

I am highly indebted to my mother; she raised me well, encouraged and gave me hope and unconditional love. I wish her happiness and good health. Thanks are due to my brothers, sisters, wife and children for their supporting.

Also very special thanks are due to my brother Prof.ali alrwahna, without his support and encouragement, this thesis couldn’t have been done.

Finally, I would like to take a moment to thank my university " Al al-Bayt University", lecturers and employees, for the moral support and encouragement during my entire graduate studies.

www.manaraa.com

Table of Contents

www.manaraa.com

List of Figures

Page number

www.manaraa.com

www.manaraa.com

www.manaraa.com

List of Abbreviations

(FCFS) First Come, First Serve

(LCFS) Last Come first Server

(SSD) Shortest Service Demand First

(NAS) Numerical Aerodynamic Simulation

(SAS) Spiral Allocation Strategy

(FFT) Fast Fourier Transform

(FF) First-Fit

(BF) Best-Fit

(MBS) Multiple Buddy System

(DQBT) Divide and Conquer Binomial Tree

(FS) Frame Sliding

(2D) Two-Dimensional

(2DBS) The Two Dimensional Buddy System

www.manaraa.com

A Spiral Non-Contiguous Processor Allocation Algorithm for 2D Mesh-Connected Multicomputers

By
Ibrahim Jum'a Alrawahna

Supervisor
Prof. Ismail Ababneh

Co-supervisor
Dr. Saad Bani-Mohammad

Abstract

Because the processors allocated to a parallel job in non-contiguous processor allocation in two dimensional can be dispersed over the mesh, communication overhead can be high because of long distances between these processors. Increasing communication distances can increase the transfer time, and also increase contention with the messages of other jobs. In this research, we have proposed a new non-contiguous allocation strategy, referred to as the Spiral Allocation Strategy (SAS) that starts allocation with the center of the mesh and scans for free processors using a spiral search around the center of the mesh. The aim is to reduce distances among processors allocated to a job. Using simulation, we compared the performance of SAS with that of existing non-contiguous strategies. The results show that SAS performs relatively better than several previous policies considered in this thesis in some cases, where the improvement is expressed in terms of reduced average job turnaround time.

www.manaraa.com

Chapter 1

Introduction

www.manaraa.com

1.1 Overview

A parallel computer is a set of processors that are able to work cooperatively to solve a large computing problem fast, while a parallel program is the program that can be executed on a number of processors at the same time [5, 8, 11].

Parallel computers are classified into two types: Shared-Memory computers and Distributed-Memory computers. In shared-memory computers (also known as multiprocessors) (see Figure 1.1.A), all processors share access to a common memory, typically via a bus. In distributed-memory computers (also known as multicomputers) (see Figure 1.1.B), every processor in the system has its own local memory and can communicate with the other processors by sending messages through an interconnection network [5, 6, 8, 12].

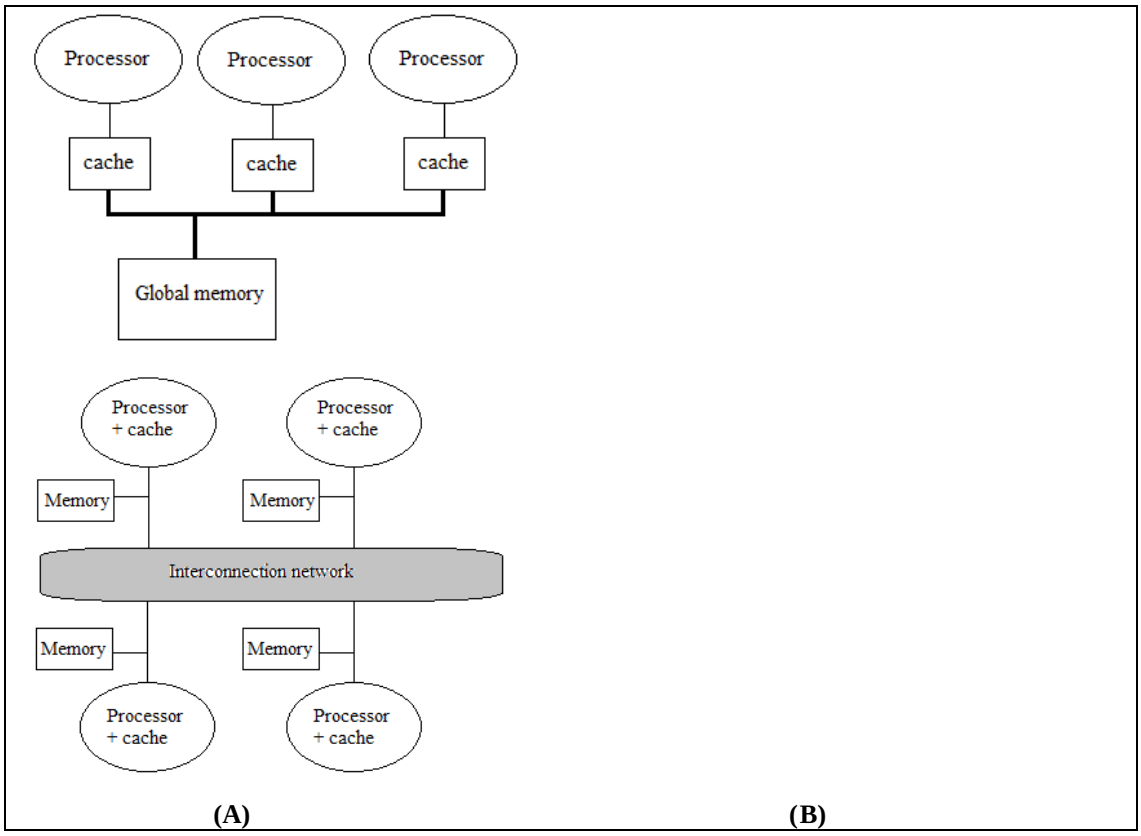


Figure 1.1 A) The basic structure of a shared-memory multiprocessor, B) The basic structure of a distributed-memory multicomputer [6].

www.manaraa.com

Interconnection networks carry data between processors and from processors to memory, and they can be divided into two categories: direct and indirect networks. Direct networks consists of point-to-point communication links among processing nodes. Examples of direct networks include the Star-Connected Network, Linear Arrays, Meshes, and k-d Meshes. Indirect networks are built using switches and communication links. Examples of indirect networks include the crossbar network, bus-based network, and multi-stage networks [9,19,20].

The mesh network is simple, scalable and easy to implement as compared to other networks that are used in multicomputers. These properties make the mesh network popular in multicomputers [1, 2, 4, 10, 11, 13, 17, 18]. The mesh network has been used in practical and experimental parallel machines. The Touchstone Delta system [1, 2, 4, 18], the Intel Paragon [3, 18], and the iWARP [1, 2, 4, 17, 18] are examples of two-dimensional (2D) mesh connected multicomputers, while the CrayT3D and the IBM BlueGene/L [3] are examples of three-dimensional (3D) mesh-connected multicomputers. Figure 1.2 illustrates an (8 × 8) 2D mesh network.

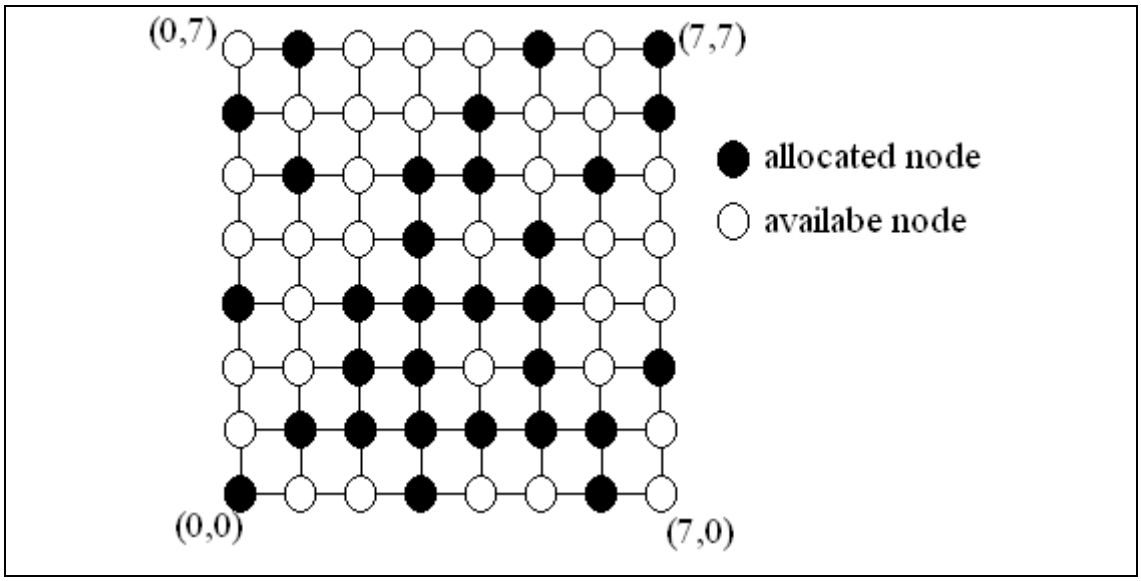


Figure 1.2 an example of a 8 x 8 2D mesh

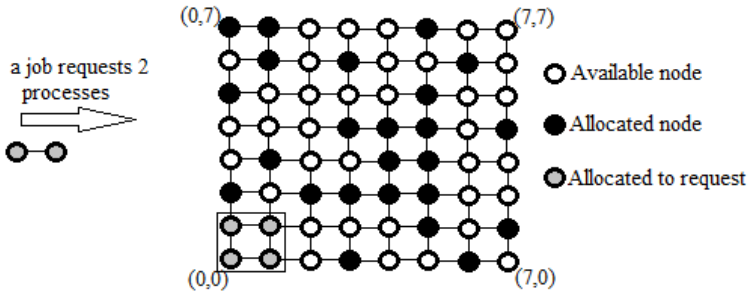
www.manaraa.com

Processor allocation is responsible for selecting a set of processors that will execute an incoming parallel job, whereas job scheduling is responsible for the selection of the next job to execute [1, 2, 4, 7, 10, 11, 15, 17, 18, 21]. Many processor allocation strategies have been suggested for 2D mesh-connected multicomputers. These strategies are categorized into two types: contiguous and non-contiguous strategies [1, 7].

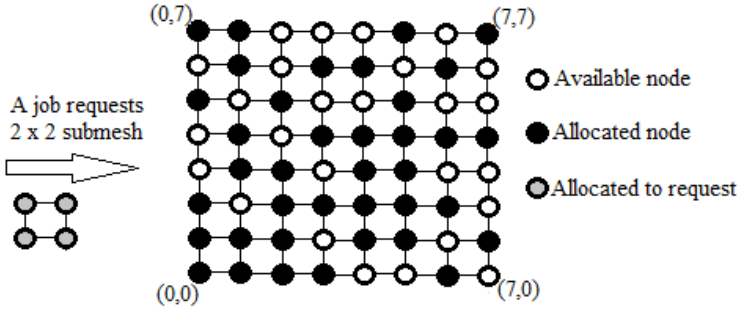
In contiguous allocation strategies, the set of processors allocated to a job are physically contiguous and have the same topology as the mesh multicomputer [1, 2, 3, 4, 7, 15, 17, 18, 21]. The Two Dimensional Buddy System (2DBS) [15], the Frame Sliding (FS) [15], First Fit (FF) and Best Fit (BF) [16] are examples of contiguous allocation strategies that have been proposed for 2D mesh multicomputers. The contiguous allocation strategies suffer from the fragmentation problem, which can degrade the utilization of the parallel system significantly [1, 2, 3, 10, 11, 13, 14, 15, 18, 21].

The fragmentation problem is classified into two categories: External and Internal fragmentation. External fragmentation occurs when there is a sufficient number of processors available for the incoming job, but allocation fails because the available processors are not contiguous or they do not form a sub-mesh of the requested shape [2, 3, 4, 10, 11, 14, 17, 18, 21], as shown in figure (1.3.A) . Internal fragmentation occurs when the number of allocated processors is larger than that required by the incoming job [11, 14, 16], as shown in figure (1.3.B). The fragmentation problem reduces system utilization and increases job turnaround time (the time that a job spends in the system from arrival until leaving) [1, 2, 13].

www.manaraa.com



A) An internal fragmentation of 2 processors.



B) An external fragmentation of 4 processors contiguous allocation strategy.

Figure 1.3 (A) Internal fragmentation (B) External fragmentation [15].

To solve the fragmentation problems, non-contiguous allocation has been presented. In non-contiguous allocation, a job can execute on multiple disjoint smaller sub-meshes rather than always waiting until a single contiguous sub-mesh of the requested size and shape is available. However, non-contiguous allocation suffers from increased message contention inside the network that results from different messages competing for network resources (e.g. channels and buffers). [1, 2, 3, 4, 7, 14, 17, 18]. Random [2, 18], Paging [1, 18] and the Multiple Buddy Strategy (MBS) [1, 18] are examples of non-contiguous allocation strategies that have been proposed for 2D mesh multicomputers

www.manaraa.com

1.2 Motivation

Most existing non-contiguous allocation strategies suffer from message contention inside the network [1, 2, 4, 10, 14, 18, 21]. The main goal of any non-contiguous allocation strategy suggested for 2D mesh-connected multicomputers is to improve system performance by reducing the job response time and maximizing the system utilization. This is achieved by eliminating processor fragmentation and increasing the degree of contiguity among allocated processors so as to alleviate the communication overhead.

Motivated by the above and according to previous research findings, there is a need for a new non-contiguous allocation strategy for 2D mesh-connected multicomputers that preserves some degree of contiguity between allocated processors in order to reduce the distance between the allocated processors and decrease message contention inside the network .

This study proposes a new non-contiguous allocation strategy for 2D mesh-connected multicomputers, which is referred to as Spiral Allocation Strategy (SAS). In this strategy, there is no internal or external fragmentation, and can alleviate the communication overhead and improve system performance as compared to that of previous non-contiguous allocation (Random, Paging(0) and MBS).

The proposed allocation strategy was implemented in the Procsimity simulation tool that is widely used in processor allocation and job scheduling in parallel systems [7]. The simulation results show that the proposed strategy improves system performance in terms of job response time as compared to the previous non-contiguous allocation strategies Random, Paging(0) and MBS.

www.manaraa.com

1.3 Outline of the Thesis

The rest of the thesis is organized as follows:

Chapter 2 describes well-known non-contiguous allocation strategies that have been proposed for 2D mesh-connected multicomputers. It also describes the method of study used in this research.

Chapter 3 introduces the Spiral allocation strategy (SAS) as a new noncontiguous allocation algorithm for 2D mesh-connected multicomputers.

Chapter 4 presents the results of extensive simulation experiments carried out in order to evaluate the performance of the SAS strategy and compare it against existing well-known non-contiguous allocation strategies.

Chapter 5 draws the conclusions of this research and outlines possible directions to continue this work in the future.

www.manaraa.com

Chapter 2

Background and Preliminaries

www.manaraa.com

2.1 Introduction

Among numerous important interconnection topologies developed for distributed-memory computers, two dimensional (2D) mesh topology has become popular because of its regularity, simplicity, easy to implement, and scalability [10,11,13]. Processor allocation is responsible for selecting a set of processors to allocate to incoming parallel job requests [1, 2, 4, 7, 10, 11, 15, 17, 18, 21]. Many processor allocation strategies have been suggested for 2D mesh-connected multicomputers. These strategies are categorized into two types: contiguous and non-contiguous strategies [1, 7, 10, 13, 15, 18, 21].

In contiguous allocation strategies, the set of processors allocated to a specific job are physically contiguous and have the same topology as the mesh multicomputer [1, 2, 3, 4, 7, 15, 18, 21]. The contiguous allocation strategies suffer from the fragmentation problem, so the utilization of the parallel system can be degraded significantly [1, 2, 3, 10, 11, 13, 14, 15, 21]. As previously reported in Chapter 1, the fragmentation problem is classified into two categories: External and Internal fragmentation. External fragmentation occurs when there is a sufficient number of processors available for the incoming job, but allocation fails because the available processors are not contiguous or they do not form a sub-mesh of the requested shape [2, 3, 4, 10, 11, 14, 17, 18, 21](please see Figure 1.3.A in Chapter 1) . Internal fragmentation occurs when the number of allocated processors is larger than that required by the incoming job [11, 14, 16] (please see Figure 1.3.B in Chapter 1).

The fragmentation problem reduces system utilization and increases job turnaround time (the time that a job spends in the system from arrival until leaving) [1, 2, 13]. Therefore, non-contiguous allocation has been proposed to solve the fragmentation

www.manaraa.com

problems. In non-contiguous allocation, a job can execute on multiple disjoint smaller sub-meshes rather than always waiting until a single contiguous sub-mesh of the requested size and shape is available. Lifting the allocation contiguity condition can increase system utilization [1, 2, 3, 4, 10, 13, 14, 16, 17, 18, 21]. However, non-contiguous allocation suffers from increased message contention that results from different job messages competing for network resources (e.g. channels and buffers). [1, 2, 3, 4, 7, 14, 17, 18, 21].

2.2 Preliminaries

The target system is a 2D mesh-connected multicomputer, referred to as M(W, H) containing W×H processors, where W and H are the width and height of the mesh system. Each processor is denoted by a pair of coordinate (x, y), where 0<x <W-1 and 0 < y <H-1. Each processor is connected by bidirectional communication links to its neighbor processors.

2.3 Related Allocation Strategies

In this part, a brief review of some previous non-contiguous allocation strategies in 2D mesh multicomputers is presented.

2.3.1 Random allocation strategy:

Random [2, 3, 15, 18] is a simple strategy in which a request for a given number of processors is satisfied with the same number of processors that are selected randomly, This strategy causes no internal or external fragmentation since all jobs are assigned exactly the requested number of processors, if available. However, high communication interference amongst jobs would be expected because no type of contiguity is enforced in this strategy [2, 3, 15, 18, 21].

www.manaraa.com

2.3.2 Paging allocation strategy:

In paging [1, 2, 14, 15, 18, 21], the whole 2D mesh is divided into square pages with side lengths of 2^{size_index} ,where *size_index* is a positive integer number. A Paging algorithm is denoted as Paging *indexing_scheme* (*size_index*), where indexing_scheme is the order of scanning available pages, such as row-major, shuffled row-major, snake-like, and shuffled snake-like indexing), as shown in figure 2.1.

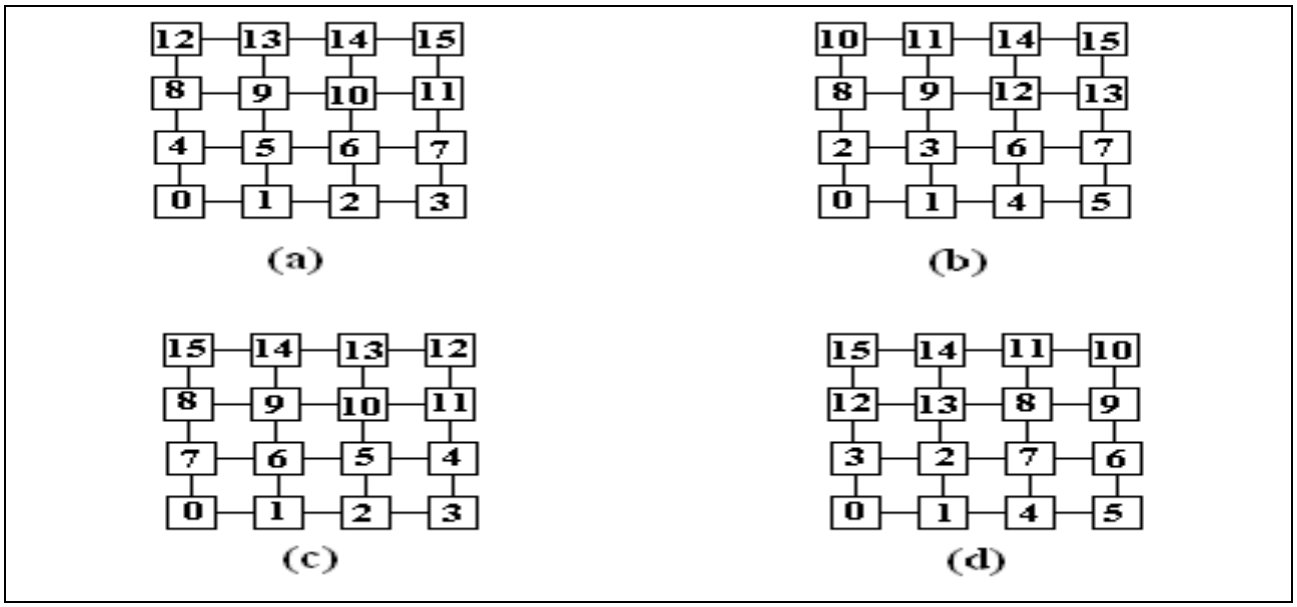


Figure 2.1: (a) Row-major indexing, (b) Shuffled row-major indexing, (c) Snake-like indexing, and (d) Shuffled snake-like indexing [15].

For example, in Paging row-major (1), the page is a (2x2) sub-mesh. A request for k processors is accomplished by allocating $\left\lceil k \Big/ \left(2^{size\_index} \times 2^{size\_index} \right) \right\rceil$ free pages. Figure 2.2 shows an example of Paging allocation that uses the row-major indexing scheme and a page size of one (2*2 blocks). Assume a request for nine processors arrives, the first three available pages will be allocated. They are the third page, fifth page and

www.manaraa.com

sixth page. The job is allocated three pages, resulting in 25 percent internal fragmentation.

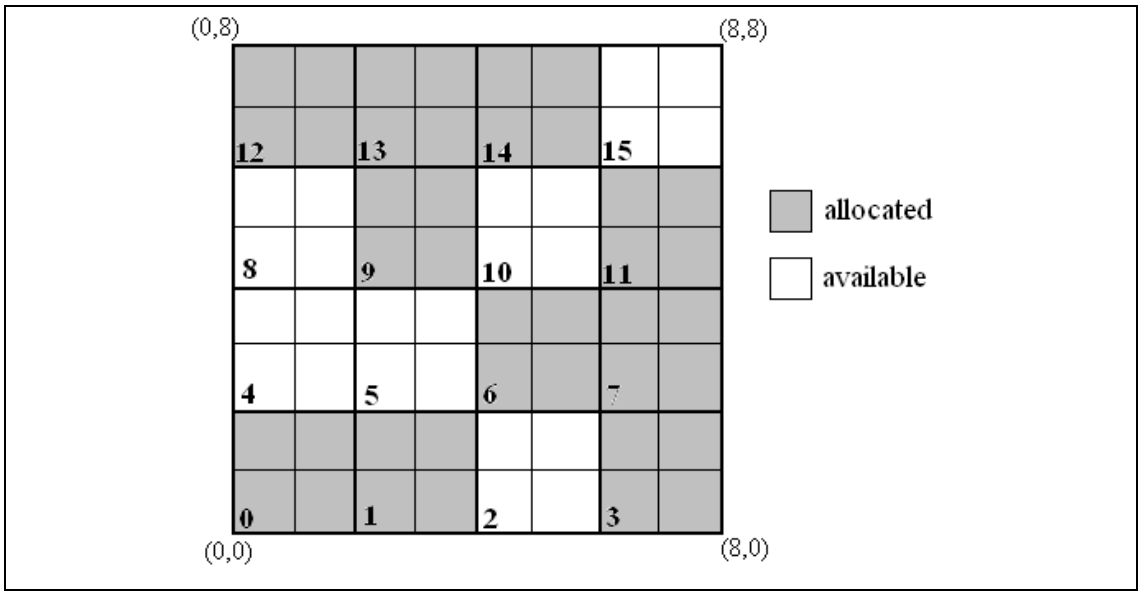


Figure 2.2: Example 2D mesh for the Paging $_{\text{row-major}}$ (1) allocation strategy. Page blocks are labeled with the index given by the row-major indexing scheme.

When page_size=0 there is no internal and external fragmentation, but when page_size ≥ 1 internal fragmentation may be introduced [1, 2, 14, 15, 18]. In this research, we considered only the row-major indexing scheme, because the other indexing schemes have only little impact on the performance of paging [1].

2.3.3 Multiple Buddy Strategy (MBS)

MBS eliminates fragmentation by applying the noncontiguous allocation model to the mesh system, while still maintaining contiguity within individual blocks [15].

www.manaraa.com

In MBS [1, 2, 3, 4, 15, 18], the 2D mesh network is divided into non-overlapped square blocks with side lengths that are power of 2. The number of requested processors (k) is represented as a base-4 number of the form:

$\sum_{i=0}^{\lfloor \log_4 k \rfloor} d_i \times (2^i \times 2^i)$, where $0 \le d_i \le 3$.

Then the requests are allocated according to the number of *di* blocks of size $2^i \times 2^i$. If a required block is unavailable, MBS searches for bigger block, which it repeatedly breaks down into buddies until it produces blocks of the required size. If that fails, the requested block is broken into four requests for smaller blocks and then the searching process is repeated again [1, 2, 3, 4].

The MBS strategy is composed of five parts:

System initialization

The mesh system is divided into blocks, which are non-overlapped square blocks with side lengths that are power of 2. The initialization process allows the strategy to be acceptable to any size mesh system [15]. Figure 2.3 shows the initial blocks of a (12*11) 2D mesh system.

www.manaraa.com

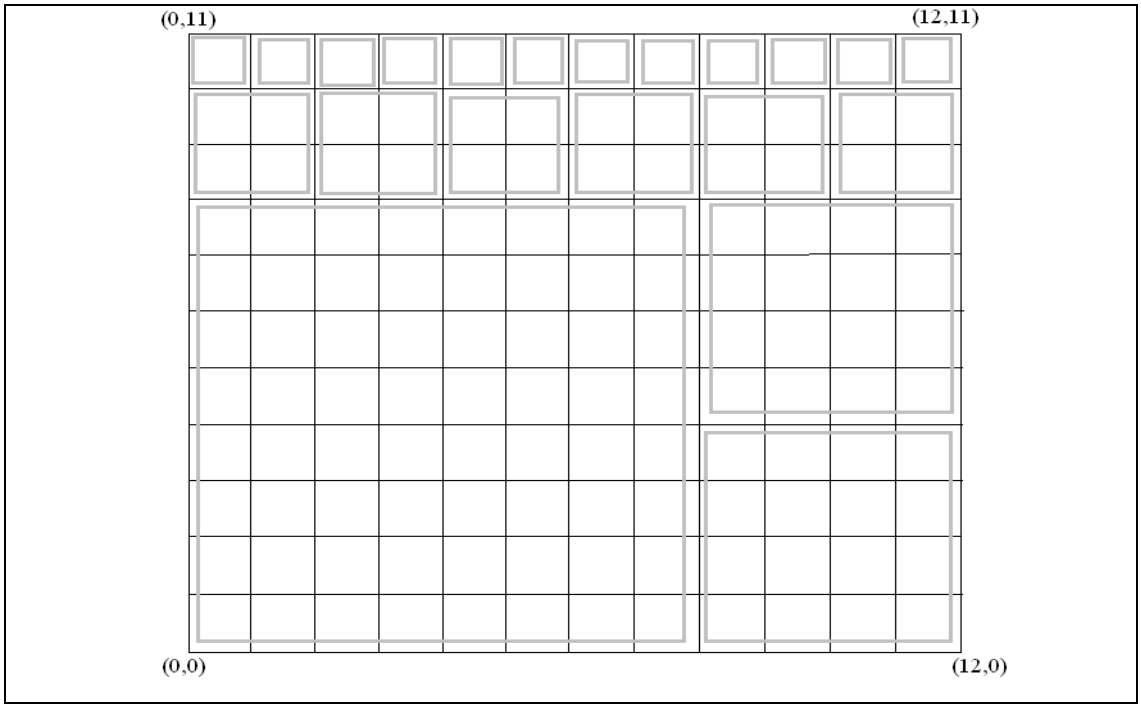


Figure 2.3: Initial blocks of a (12 * 11) 2D mesh system.

Request factoring algorithm

The number of requested processors (k) is represented as a base-4 number of the form:

$\sum_{i=0}^{\lfloor \log_4 k \rfloor} d_i \times (2^i \times 2^i)$, where $0 \le d_i \le 3$. Then the requests are allocated according to the number of di blocks of size $2^i \times 2^i$ [15].

Buddy generating algorithm

The buddy generation algorithm divides a large block into several smaller blocks to satisfy the $2^i \times 2^i$ requests [15].

Allocation and deallocation algorithms

First, the request is factored and stored in the appropriate element of the request array, Array[i]. If possible, each request for a block of size i is allocated immediately from free blocks. Otherwise, MBS searches for a larger block by breaking it into

www.manaraa.com

smaller buddies. If that fails, the request will be broken down into 4 smaller requests, which are stored in Array[i - 1] [15].

The deallocation procedure of the MBS strategy needs to return all blocks owned by the job to the mesh system, and merge the buddies up to restore the larger blocks[15].

2.4 Switching Method

In most multicomputer systems, a message enters the network and is switched or routed from a source node to its destination node through a series of intermediate nodes. Three types of switching techniques are common: store-and-forward, virtual cut-through and wormhole switching.

Store-and-forward switching: In store-and-forward switching, the message is divided into fixed-length packets that are routed from source to destination. Each packet contains a header that contains the data needed for routing the packet. The entire packet is forwarded to the next node in its path after stored at every intermediate node [19, 20].

The major disadvantage of store-and-forward switching is that, since the packet is stored entirely at each intermediate node, the time to transmit a packet from source to destination is directly proportional to the number of hops in the path. Furthermore, at each intermediate node, we need a buffer space to hold at least one packet [19, 20].

Virtual cut-through switching: In virtual cut-through, a message is stored at an intermediate node only if the next channel required is occupied by another packet. The distance between the source and destination has little effect on communication latency. In an extreme case, when a message encounters is blocked at all the

www.manaraa.com

intermediate nodes, the virtual cut-through technique reduces to store-and-forward[19, 20].

Wormhole switching: Wormhole switching is a variant of the virtual cut-through technique that avoids the need for large buffer spaces. In wormhole switching, a packet is transmitted between nodes in small units called flits. A flit is the smallest unit of a message on which flow control can be performed. The header flit(s) of a message contains all the necessary routing information and all the other flits contain the data. The flits of the message are transmitted through the network in a pipelined fashion. Since only the header flit(s) has the routing information, all the trailing flits follow the header flit(s) contiguously. Flits of two different messages cannot be interleaved at any intermediate node. Successive flits in a packet are pipelined asynchronously in hardware using a handshaking protocol. When the header flit is blocked, then all the trailing flits occupy the buffers at the intermediate nodes [19, 20]. Wormhole switching is used in this research when examining the performance of the non-contiguous allocation algorithms. We have limited ourselves to wormhole switching because it has been used in the existing non-contiguous allocation strategies [1, 2, 4, 10, 14, 15, 16, 18]

2.5 Routing Algorithm

Routing algorithms are crucial to the efficient operation of interconnection networks as they specify the paths packets will take when messages are being sent among the processors of the network. A good routing algorithm reduces the latency of the message by minimizing the number of hops that are required for packets to reach

www.manaraa.com

their destination, moreover routing should be able to handle deadlock situations [19,20].

Dimension-ordered routing in $2D$ mesh is called XY routing [15,19, 20]. The two dimensions of a mesh are labeled as X and Y. A message is first routed in the X direction completely and then in the Y direction. Figure 2.4 shows an example of XY routing between a source node and a destination node in 8×8 mesh network. Dimension-ordered routing is used in this research. We have limited ourselves to dimension-ordered routing because it has been used in the existing noncontiguous allocation strategies [1,4,10,15,21].

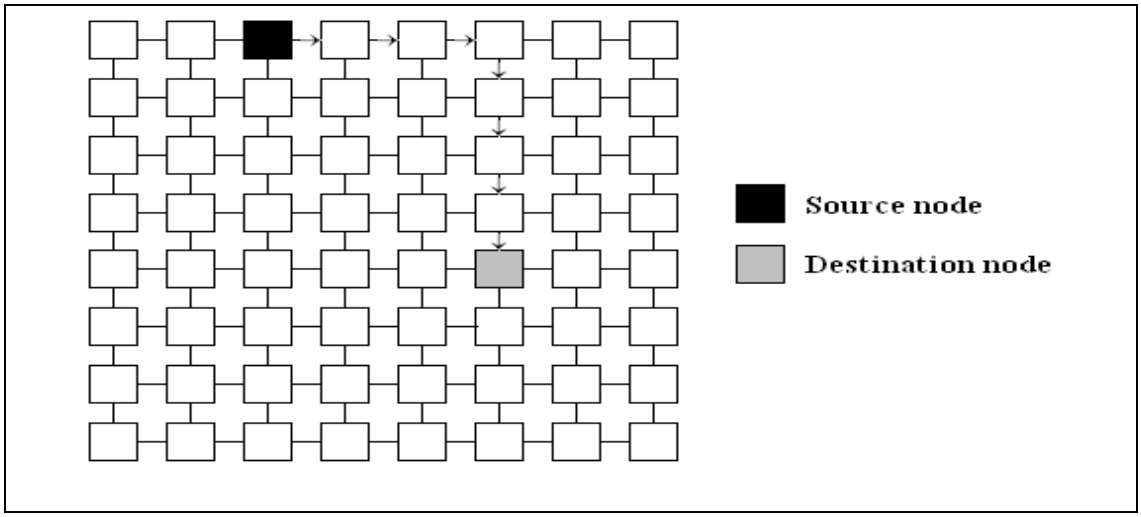


Figure 2.4: Dimension-ordered (XY) routing in 8×8 mesh network

2.6 Communication Patterns

The experiments measure the effects of message-passing overhead on overall performance. The ProcSimity Simulator models the exchanging of messages between the processors allocated to an incoming job to the system. Processors allocated to an incoming job communicate with each other according to a given communication

www.manaraa.com

pattern selected from common parallel applications. [15]. To evaluate the performance of the SAS strategy and compare it against existing well-known non-contiguous allocation strategies, eight communication patterns have been considered. They are One-to-All, random, Near neighbor , Parallel Fast Fourier Transform (FFT), Divide and Conquer, ALL_TO_ONE, ring and NAS Multigrid Benchmark.

2.7 The Simulation Tool (ProcSimity Simulator)

ProcSimity is a software tool for research in the area of processor allocation and job scheduling in distributed memory multicomputers. ProcSimity was developed at the University of Oregon; it was written in the C programming language. ProcSimity was designed to investigate some of the key performance bottlenecks in the fields of job scheduling and processor allocation, such as the fragmentation problem and the message contention overhead [7].

The architecture assumed in ProcSimity consists of a network of processors interconnected through message routers at each node. Adjacent nodes are connected by two uni-directional channels and messages may be routed by either store-and-forward, virtual cut-through or wormhole flow control. ProcSimity currently supports both the mesh interconnection topology and the general k-ary n-cube topology, and uses dimension order routing [7].

When ProcSimity simulates a mesh-connected multicomputer, independent user jobs that arrive at the system request sub-meshes of free processors. If an arriving job can not be run immediately, due to a lack of free processors or because there are other waiting jobs, the job is diverted to the waiting queue. The job is selected to be

www.manaraa.com

executed from the waiting queue based on the scheduling strategy used, and then the processor allocation algorithm determines and allocates the set of processors on which the job will execute. The allocated processors may be contiguous or noncontiguous. The new job holds these processors exclusively until it finishes running. At this time, it departs the system and its processors are freed for use by other jobs [7].

2.8 Justification of the Method of Study

There are three main approaches to evaluating the performance of any system: measurement, analysis, and simulation [8]. In the measurement approach, the system is implemented in full and its performance is measured directly. The analysis approach uses mathematical analysis from first principles to evaluate the system. In the simulation approach, a model for the real system is designed and experiments are conducted either for understanding the behavior of the system or for evaluating various strategies.

In this research, simulation has been selected as the method of study. Simulation is used to better understand the expected performance of the real system and to test the effectiveness of the system design. Simulation is generally more adequate because it involves fewer approximations than conventional approaches; it is often used because it is the only viable alternative. Analysis may be too difficult, or may require too many simplifying assumptions that restrict their applicability to a limited number of scenarios. In addition, difficulties arise from trying to make realistic assumptions, and from size. The measurement approach is sometimes impossible because the system

www.manaraa.com

does not exist or it would take too much time to build. In other cases, it is irrelevant because we cannot change the configuration as desired [8].

In this research, we used ProcSimity, which has already been validated extensively[7]. We have easily incorporated our suggested algorithms into the simulator. This has helped to considerably cut down the development time and debugging of the code. Extensive simulation experiments were conducted so as to compare the performance of the noncontiguous allocation strategies considered in this work.

www.manaraa.com

Chapter 3

Spiral allocation strategy (SAS): A New Non-contiguous Allocation Algorithm for Mesh-Connected Multicomputers

www.manaraa.com

3.1 Introduction

Many processor allocation strategies have been proposed for 2D mesh-connected multicomputers. They are based on contiguous allocation, where the set of processors allocated to a specific job are physically contiguous and have the same topology as the mesh multicomputer [1, 2, 3, 4, 7, 15, 18, 21]. The contiguous allocation strategies suffer from the high processor fragmentation problem, so the mean response time and the utilization of the parallel system can be degraded significantly [1, 2, 3, 10, 11, 13, 14, 15, 18, 21].

To solve the fragmentation problems, non-contiguous allocation has been proposed, where a job can execute on multiple disjoint smaller sub-meshes rather than having to wait until a single contiguous sub-mesh of the requested size and shape is available. However, non-contiguous allocation suffers from increased message contention inside the network that result from different jobs messages competing for network resources (e.g., channels and buffers) [1, 2, 3, 4, 7, 14]. Lifting the contiguity condition can be expected to reduce processor fragmentation and increase processor utilization [1, 2, 4, 13, 14]

Most existing non-contiguous allocation strategies suffer from message contention inside the network [1, 2, 4, 10, 14, 18, 21]. The main goal of any non-contiguous allocation strategy suggested for 2D mesh-connected multicomputers is to improve system performance by reducing the job turnaround time and maximizing the system utilization. This is achieved by eliminating processor fragmentation and increasing the degree of contiguity among allocated processors so as to alleviate the communication overhead.

www.manaraa.com

Motivated by the above and according to previous research findings, there is a need for a new non-contiguous allocation strategy for 2D mesh-connected multicomputers that preserves some degree of contiguity between allocated processors in order to reduce the distance between the allocated processors and hence decrease message contention inside the network.

This study proposes a new non-contiguous allocation strategy for 2D mesh-connected multicomputer, which is referred to as Spiral Allocation Strategy (SAS), and compares its performance properties using detailed simulations against the performance of the previous non-contiguous allocation strategies: Random, Paging(0) and Multiple Buddy Strategy (MBS).

3.2 The proposed Spiral Allocation Strategy (SAS)

The target system is a 2D mesh-connected multicomputer, referred to as M(W, H), as shown in Figure 3.1. This figure shows an example of a (4 × 4) 2D mesh, where allocated processors are denoted by shaded squares and free processors are denoted by white squares.

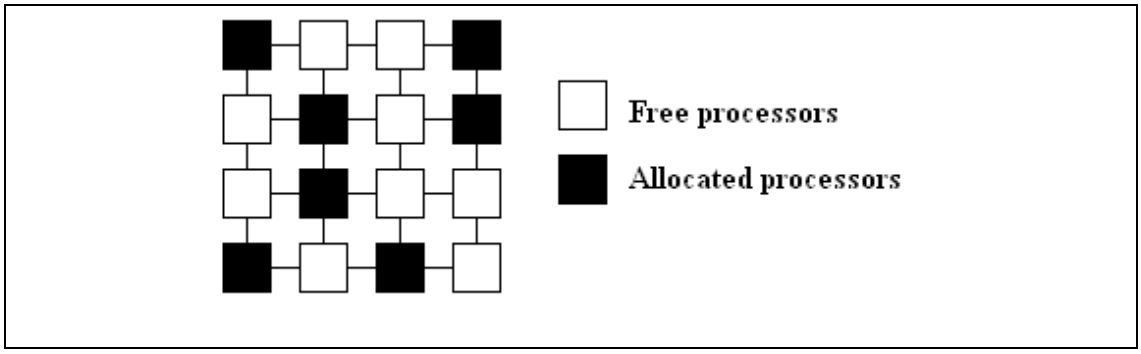


Figure 3.1: An example of a (4 × 4) 2D mesh

In SAS, an allocation request for k processors is satisfied by the first k available processors selected as follows, starting at the center of the mesh. The coordinates of the central node is (x, y), where $x = \lfloor (W-1)/2 \rfloor$ and $y = \lfloor (H-1)/2 \rfloor$, and

www.manaraa.com

scanning for the remaining needed processors proceeds from this node using a squared spiral anticlockwise movement around the central node, allocating the free nodes encountered (see Figure 3.2). This process continues until the job is allocated the number of processors it has requested or the upper-left corner of the mesh is reached. If the number of free processors is less than that requested, allocation is not attempted and it fails.

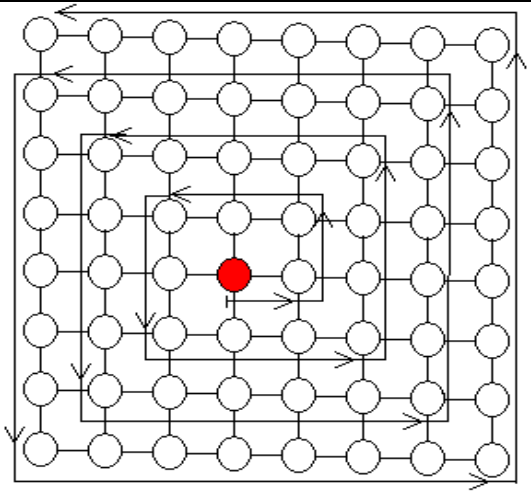


Figure 3.2: Spiral Allocation Strategy of a 8 x 8 2D mesh

Allocation in SAS is implemented by the algorithm outlined in Figure 3.3, while the deallocation algorithm is outlined in Figure 3.4. Note that allocation always succeeds if the number of free processors is $\geq a \times b$.

Procedure SAS_Allocate (a,b**):**
Begin {
//Mesh is M(W, H); incoming job J requests the allocation of an a ×b sub-mesh; this code is for W=H
Job_Size =a∗b; Total_Allocated = 0;
m=(W-1)/2; n=(H-1)/2
Central node has coordinates (m, n); jid is the ID of the current job
move=1 // number of current horizontal/vertical movements
Step1. if (number of free processors < Job_Size) return failure.
Step2. else{
if (Central node is free) {Mesh(m,n) = jid; Total_Allocated++}
Step3. j =0

www.manaraa.com

```
        while (Total_Allocated <= Job_Size and ++j<=move ) do{
          if (m+ j , n) is free{      //move right
            Mesh(m+j, n) = jid; Total_Allocated++;
          }
        }
        j = 0;
Step4.  while (Total_Allocated <= Job_Size and ++j<=move) do{
          if (m , n+ j) is a free{       //move up
            Mesh(m,n+j)=jid; Total_Allocated++;
          }
        }
        move=move+1; j = 0;
Step5.  while (Total_Allocated <= Job_Size and ++j<=move) do{
          if (m- j , n) is a free{        //move left
            Mesh(m-j,n)=jid; Total_Allocated++;
          }
        }
        j = 0;
Step6.  while (Total_Allocated <= Job_Size and ++j<=move) do{
          if (m , n- j) is a free{       //move down
            Mesh(m,n-j)=jid; Total_Allocated++;
          }
        }
        move=move+1; j =0;
Step7. if (Total_allocated == Job_Size return success.
       else go to Step 3.
}
End
```

Figure 3.3 Outline of the SAS allocation algorithm

```
Procedure SAS_De-allocate ():
Begin {
   jid is id of the departing job;
   for all Mesh elements Mesh(i, j) do
    if (Mesh(i,j) == jid) Mesh(i,j) = FREE;
}
End.
```

Figure 3.4 Outline of the SAS deallocation algorithm

To explain how the allocation process works, consider the example of Figure 3.5, which is a 8×8 mesh is illustrated. Suppose there are 32 allocated processors in this example. Assume that an incoming job requests a 4 × 4 sub-mesh. Note that allocation always succeeds if the number of available processors is equal or larger than the number requested. In this example, there are 32 available processors, and we can allocate 16 processors. Allocation starts at the central node with coordinates

www.manaraa.com

(3,3). The next movement is to the right of the central node, but the node is allocated to another job, so the algorithm will move to the next node that has the coordinates (4,4). We can see that it is available, so it will be allocated. The next movement is to the left but the node is allocated, so we go to the node with the coordinates (2,4). The next movement is downwards but the node is allocated, so the algorithm will move to the next node that has the coordinates (2,2) and we can see that it is available, so it will be allocated. for the sake of conciseness by using the same procedure the next processors that will be allocated to the job request are with coordinates ((5,2) ,(5,3) ,(5,5), (4,5), (3,5), (1,3), (1,2), (1,1), (3,1), (4,1), (5,1) and (6,1)) respectively.

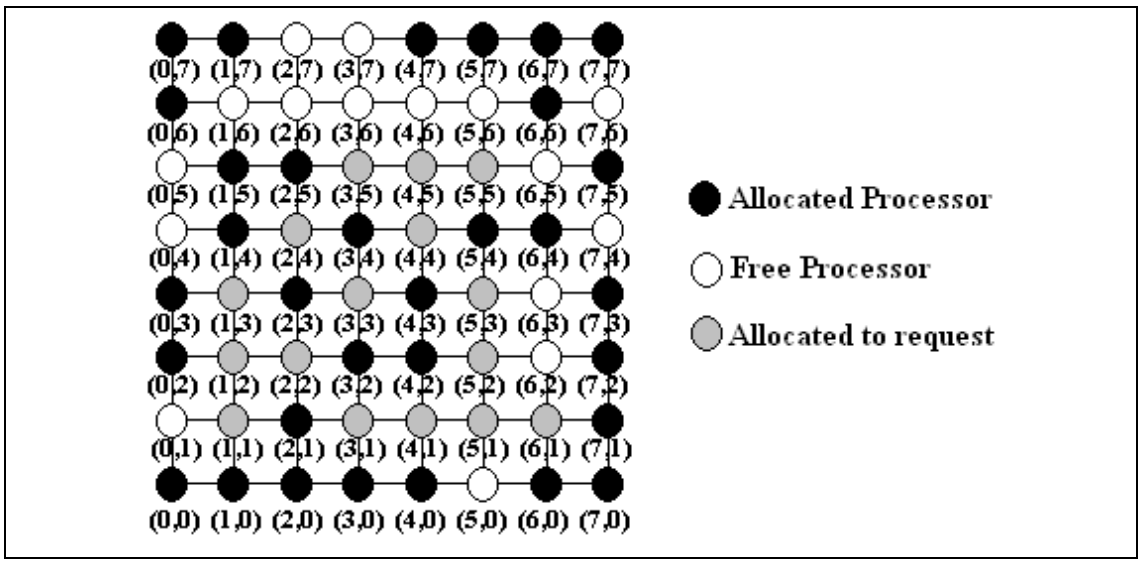


Figure 3.5: A 8 × 8 mesh with 32 free processors

www.manaraa.com

Chapter 4
Simulation Results

www.manaraa.com

We have conducted extensive simulation experiments so as to compare the performance of the noncontiguous allocation strategies considered in this thesis. The proposed processor allocation strategy (SAS) was implemented in the C language and later integrated into the ProcSimity simulator [7]. The target mesh assumed is square with side lengths L. Jobs have exponential distribution average inter-arrival times. The job scheduling scheme is First Come First Serve (FCFS), where only the job at the head of the queue is considered for allocation. This policy was used in many other related studies [2, 3, 4, 10, 14, 15, 18, 21].

Processors allocated to a job communicate with each other using some common communication patterns. We have used eight communication patterns. They are One-to-All, Random, Near neighbor, Parallel Fast Fourier Transform (FFT), Divide and Conquer, All-to-One, Ring and NAS Multigrid Benchmark. Simulation parameters are illustrated in Table 4.1. These values were used in many other related studies [1, 2, 4, 10, 11, 14, 15, 16, 18, 21].

Table 4.1: The ProcSimity Simulation Parameters used in the Simulation Experiments

Parameters	Description	Value
Architecture	Dimensions of the architecture to be simulated for mesh architecture	8*8 12*12 16*16 20*20
Packet Size	Value representing the number of 1-byte flits in each packet that is sent through the mesh.	8
Flow Control Mechanism	Method of flow control employed in the network switching elements.	Wormhole routing.
Buffer Size	Number of flits that can be held in each network switching element buffer.	1

www.manaraa.com

Routing Delay	Number of network cycles needed for a flit to be routed through a network switch to the output channel.	3
Router Type	The type of routing hardware implemented in the network switches .	XY Routing
Allocation Strategy	The processor allocation strategies to be used.	Paging(0), MBS, Random and SAS
Scheduling Strategy	The job scheduling strategy to be used.	First Come First Serve (FCFS)
Job Size Distribution	The distribution of the number of processors requested by each job.	Uniform: Job widths and lengths are uniformly distributed over the range from 1 to the mesh side lengths.
Mean Interarrival Time	The mean for random generation of Poisson interarrival times.	The values are determined through experimentation with the simulator.
Mean Time Between Sends	The average time between message sends for each process in a job executing the Random communication pattern.	0.0
Communication Pattern	The specific communication pattern that each job executes when message passing.	One-to-All, random, Near neighbor, Parallel Fast Fourier Transform (FFT), Divide and Conquer, All_to_One, ring and NAS Multigrid Benchmark.
Message Size	The number of bytes in each message.	8
Mean Massages per Job	Specifies the number of messages to be sent by each job.	5.0
Number of Runs	The number of times each simulation is duplicated for accuracy and establishing confidence intervals.	The values are determined through experimentation with the simulator with relative errors are below 5% of the means.
Number of Jobs	The number of jobs simulated in each run	1000

Each simulation run consists of 1000 completed jobs. Simulation results are averaged over enough independent runs so that the confidence level is 95% that the relative errors don't exceed 5%. The main performance parameter observed is the *average response time*. The response time is the time that a job spends in the system from arrival to departure. The independent variable in these experiments was the system load, defined as the inverse of the mean interarrival time of jobs. Its range of values

www.manaraa.com

from low to heavy loads has been determined through experimentation with the simulator allowing each allocation strategy to reach its upper limits of utilization. In the figures that are presented below, the *x*-axis represents the system load while the *y*-axis represents results of average response time.

Average response Time:

In Figures 4.1, 4.2, 4.3 and 4.4 the average response times of jobs are plotted against the system load for the one-to-all communication pattern. The results show that SAS performs better than all other noncontiguous allocation strategies considered across the mesh sizes assumed in this thesis. In Figure 4.1, for example, the average response times of the SAS strategy are about 11%, 26%, and 17% of those of MBS, Random, and Paging (0), respectively under the job arrival rate of 0.025 jobs/time units.

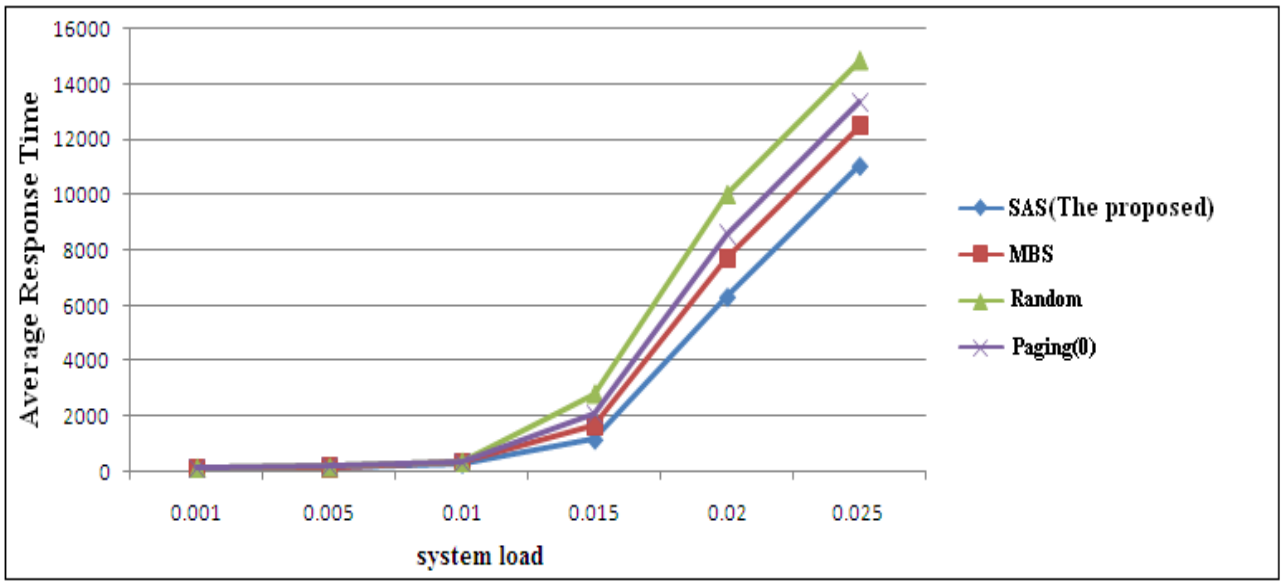


Figure 4.1: Average response time vs. system load for the one-to-all communication pattern and uniform side lengths distribution in an 8 × 8 mesh.

www.manaraa.com

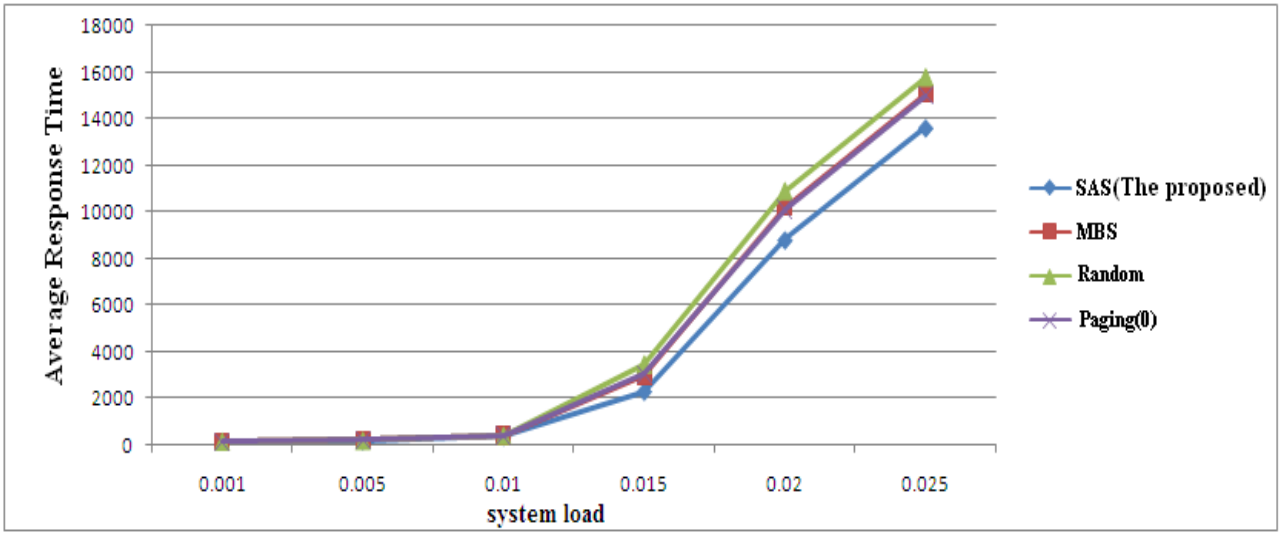


Figure 4.2: Average response time vs. system load for the one-to-all communication pattern and uniform side lengths distribution in a 12 × 12 mesh.

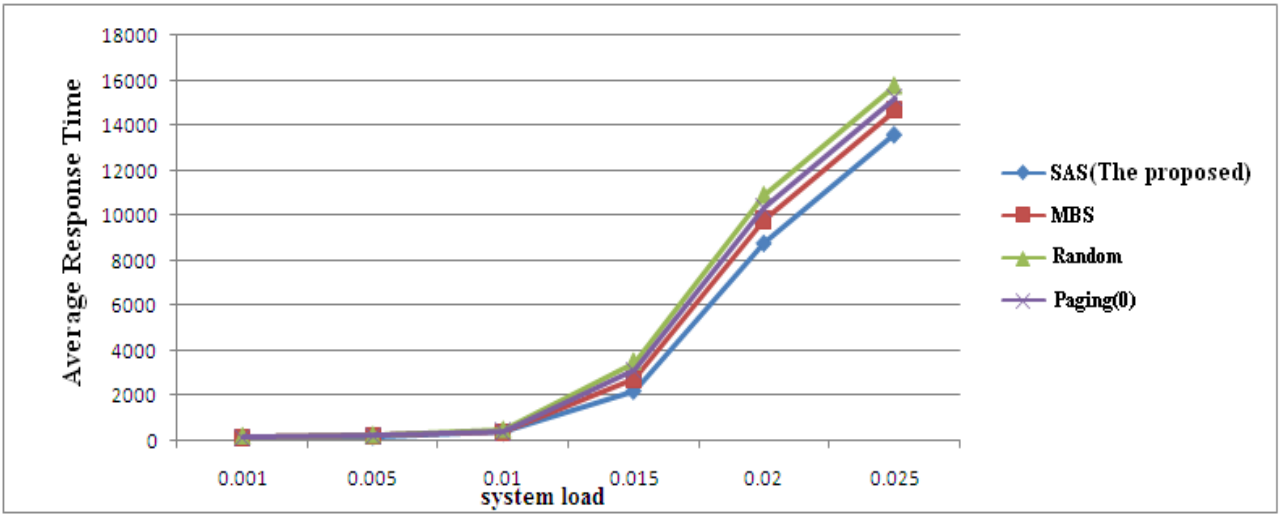


Figure 4.3: Average response time vs. system load for the one-to-all communication pattern and uniform side lengths distribution in a 16 × 16 mesh.

www.manaraa.com

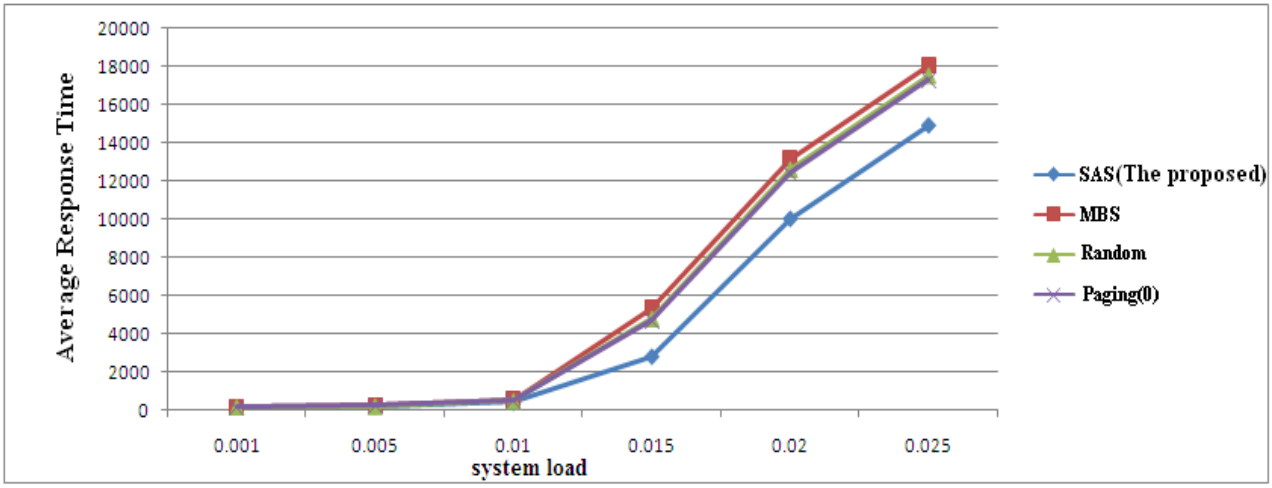


Figure 4.4: Average response time vs. system load for the one-to-all communication pattern and uniform side lengths distribution in a 20 × 20 mesh.

In Figures 4.5, 4.6, 4.7 and 4.8, the average response times of jobs are plotted against the system load for the Random communication pattern. The results show that the SAS performs better than Random strategy, but it is worse than both MBS and Paging(0) strategies. In Figure 4.5, for example, the average response times of the SAS strategy are about 12% compared to MBS and 3% to Paging (0) under the job arrival rate of 0.05 jobs/time units.

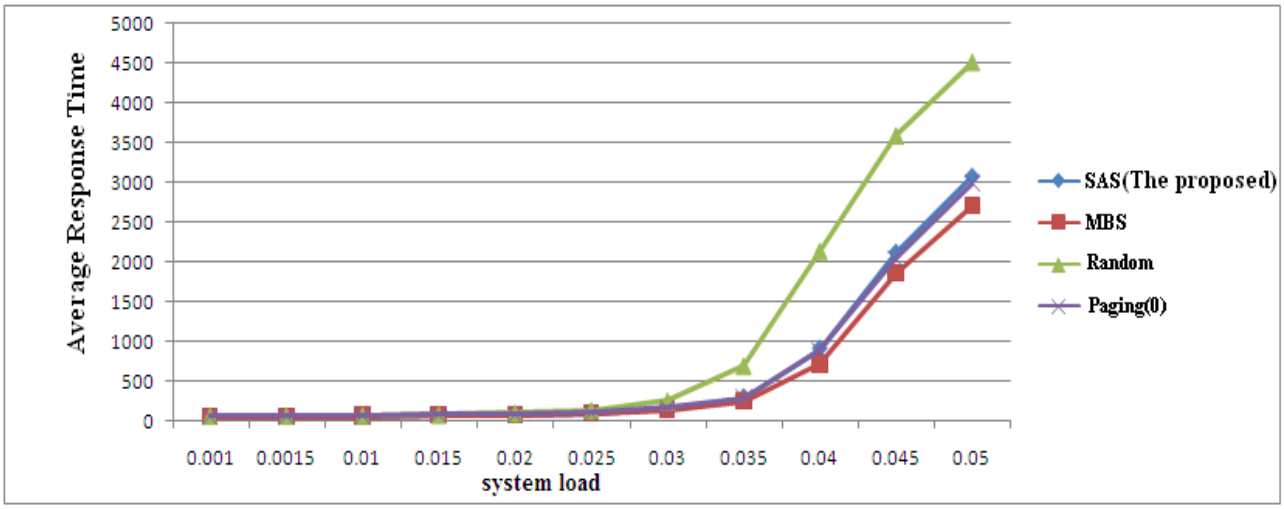


Figure 4.5: Average response time vs. system load for the Random communication pattern and uniform side lengths distribution in a 8 × 8 mesh.

www.manaraa.com

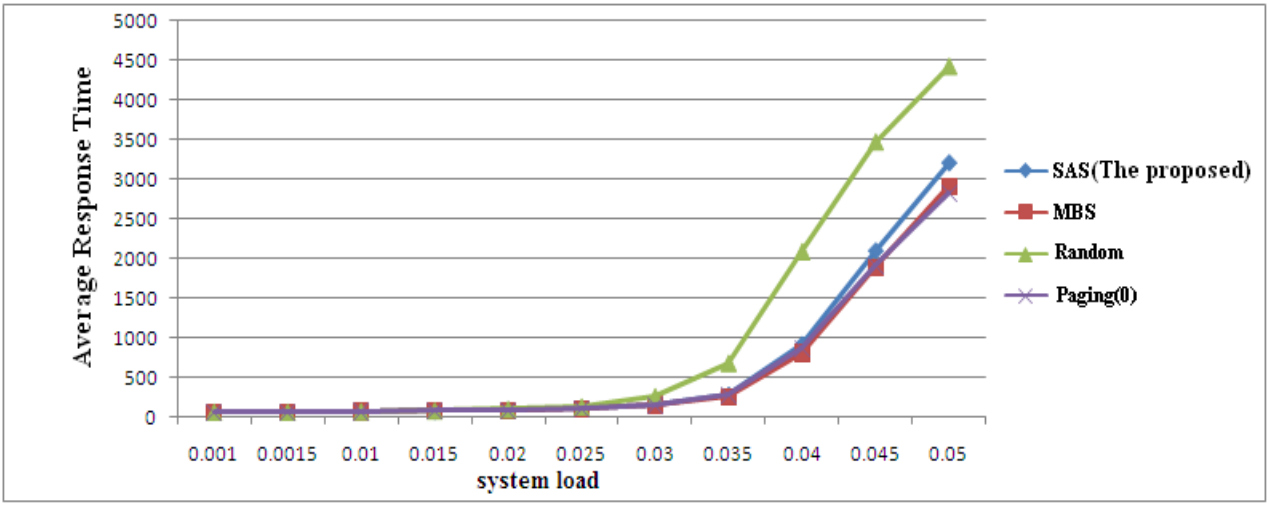


Figure 4.6: Average response time vs. system load for the Random communication pattern and uniform side lengths distribution in a 12 × 12 mesh.

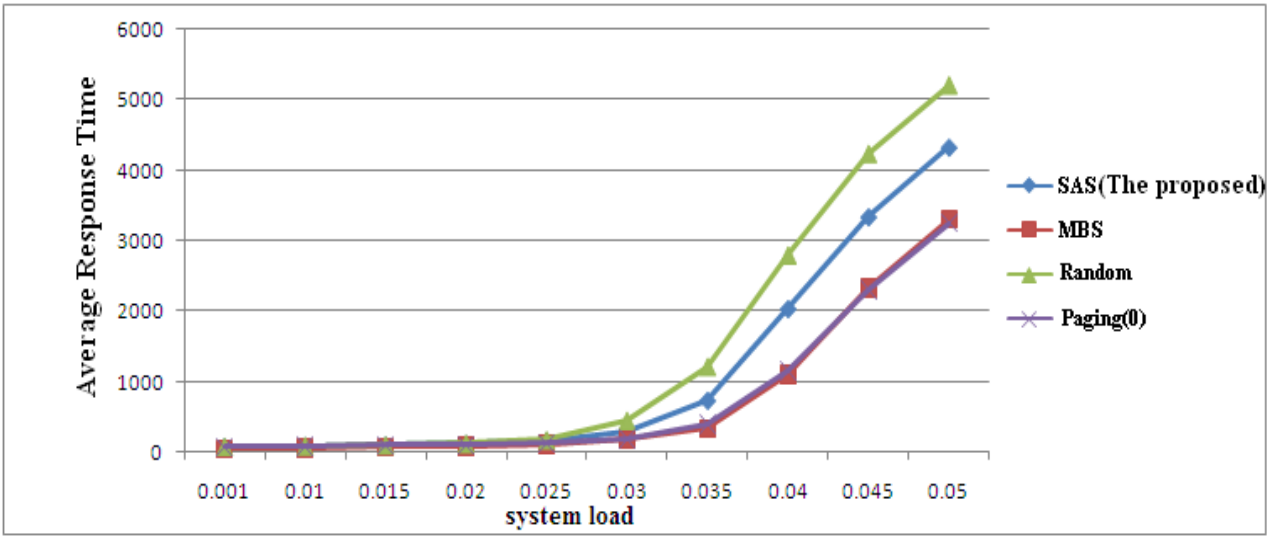


Figure 4.7: Average response time vs. system load for the Random communication pattern and uniform side lengths distribution in a 16 × 16 mesh.

www.manaraa.com

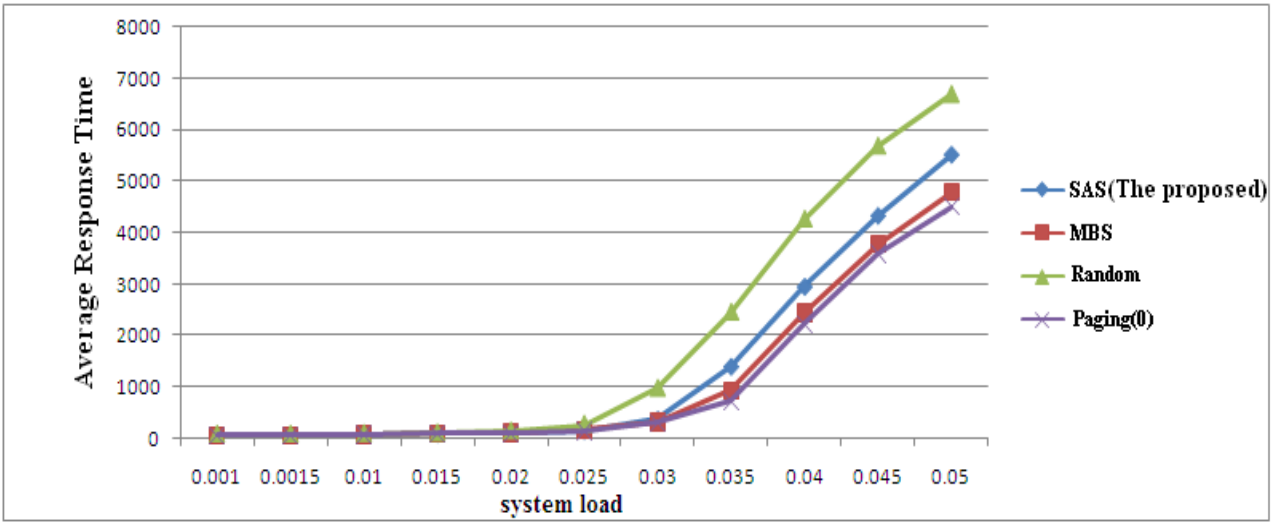


Figure 4.8: Average response time vs. system load for the Random communication pattern and uniform side lengths distribution in a 20 × 20 mesh.

In Figures 4.9, 4.10, 4.11 and 4.12, the average response times of jobs are plotted against the system load for the near neighbor communication pattern. In figure 4.10 ,4.11 and 4.12 The results show that SAS performs better than Random strategy, but it is worse than both MBS and Paging(0) strategies. However in figure 4.9 The SAS performs better than all other noncontiguous allocation strategies considered across the mesh size 8 × 8. For example, the average response times of the SAS strategy are about 22%, 55%, and 27% of those of MBS, Random, and Paging (0), respectively under the job arrival rate of 0.05 jobs/time units.

www.manaraa.com

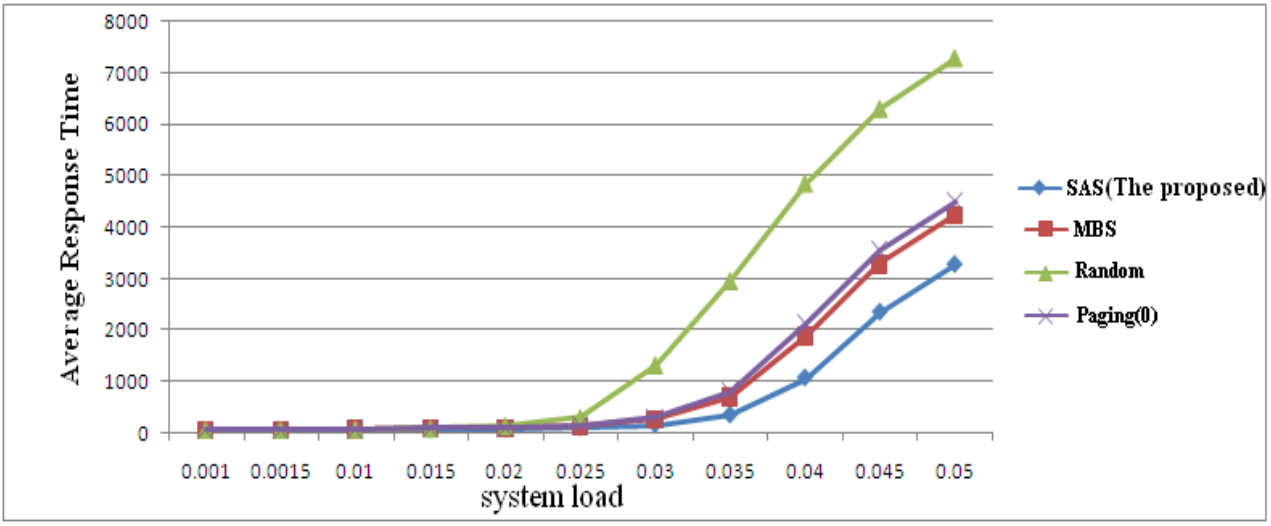


Figure 4.9: Average response time vs. system load for the near neighbor communication pattern and uniform side lengths distribution in a 8 × 8 mesh.

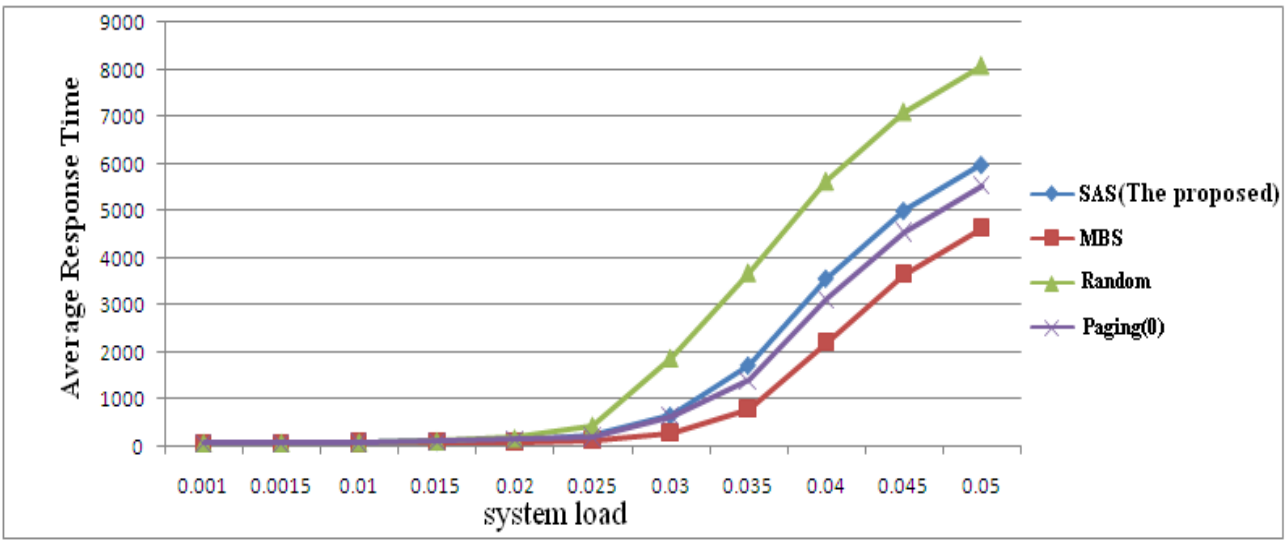


Figure 4.10: Average response time vs. system load for the near neighbor communication pattern and uniform side lengths distribution in a 12 × 12 mesh.

www.manaraa.com

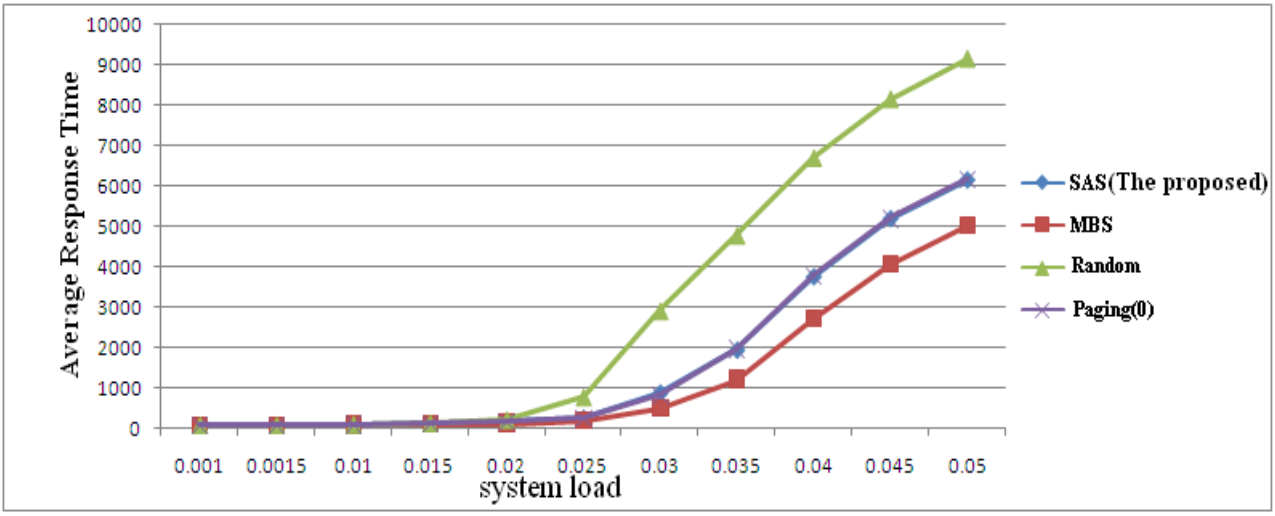


Figure 4.11: Average response time vs. system load for the near neighbor communication pattern and uniform side lengths distribution in a 16 × 16 mesh.

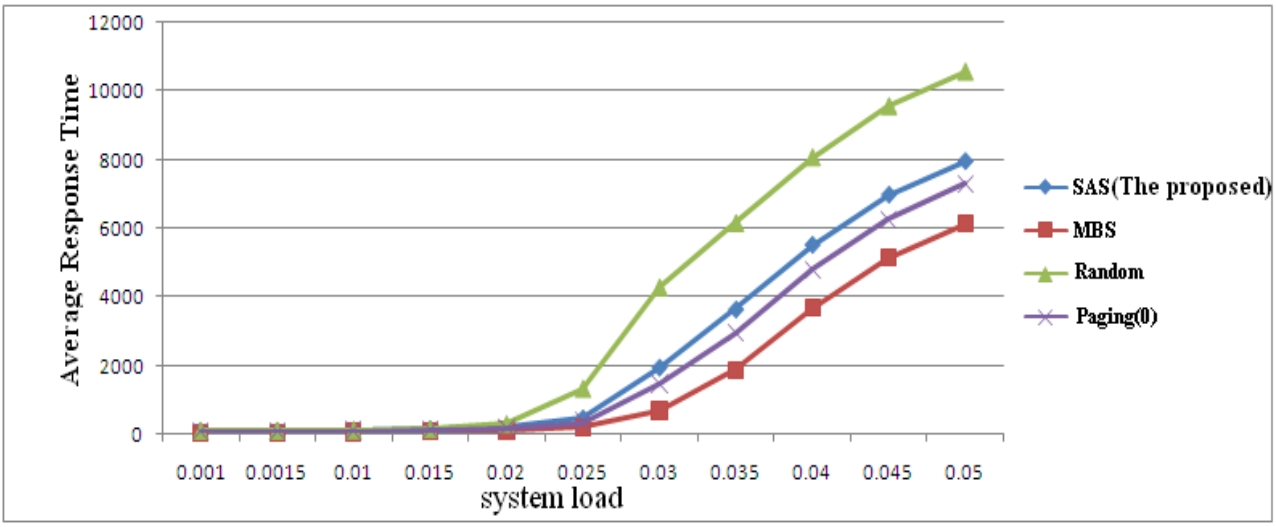


Figure 4.12: Average response time vs. system load for the near neighbor communication pattern and uniform side lengths distribution in a 20 × 20 mesh

In Figures 4.13, 4.14, 4.15 and 4.16, the average response times of jobs are plotted against the system load for the Parallel Fast Fourier Transform(FFT) communication pattern. The results show that SAS performs worse than all other noncontiguous allocation strategies considered across the mesh sizes 8 × 8 and 16 × 16 as shown in figures 4.13 and 4.15. But in figures 4.14 and 4.16 the SAS strategy performs better

www.manaraa.com

than Paging (0), and worse than MBS and Random strategies. For instance, when the job arrival rate is 0.05 jobs/time units, as shown in figure 4.13 the average response time of SAS is increase about 73%, 16%, and 72% of that of MBS, Random, and Paging (0) respectively under the job arrival rate of 0.05 jobs/time units.

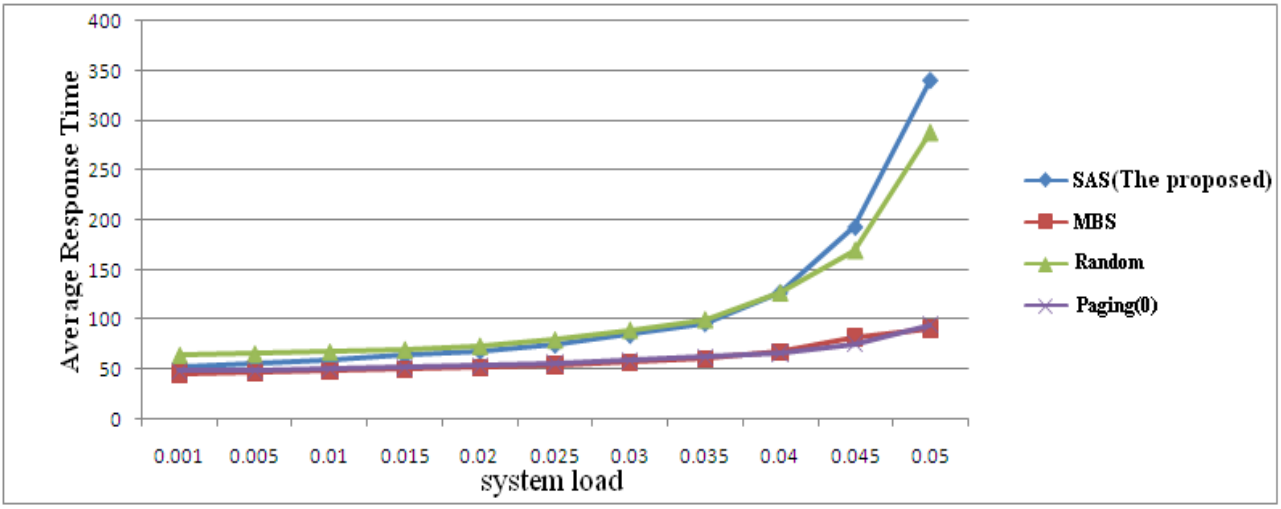


Figure 4.13: Average response time vs. system load for the FFT communication pattern and uniform side lengths distribution in a 8 × 8 mesh

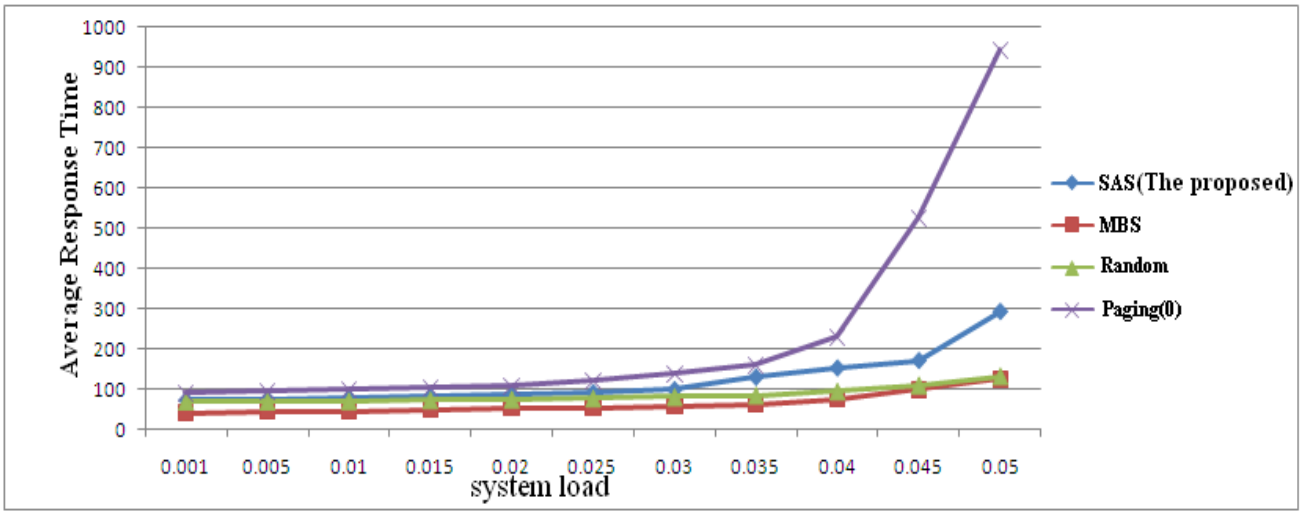


Figure 4.14: Average response time vs. system load for the FFT communication pattern and uniform side lengths distribution in a 12 × 12 mesh

www.manaraa.com

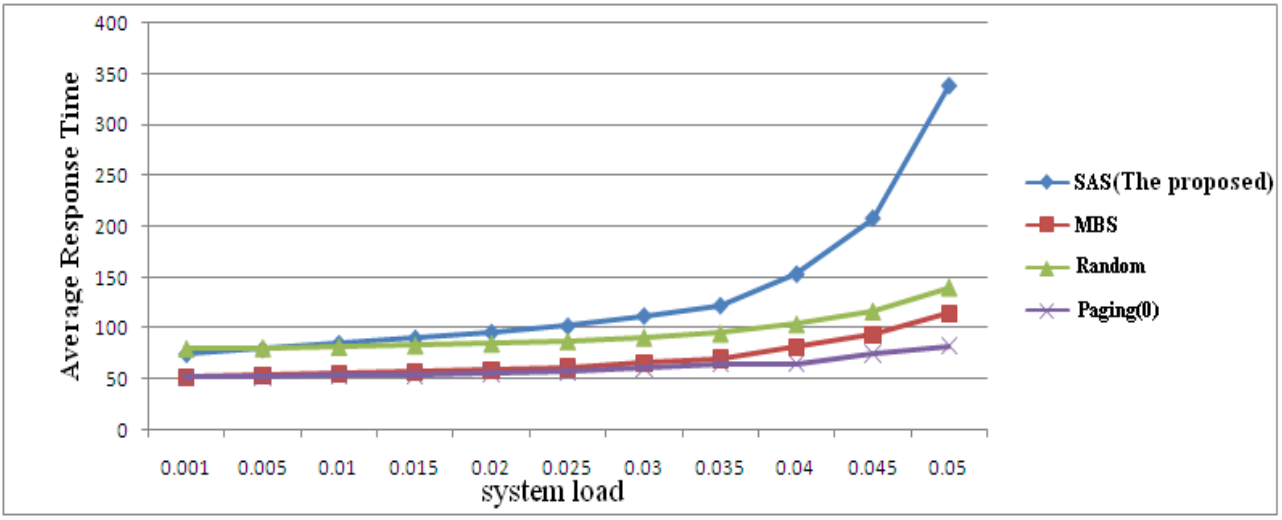


Figure 4.15: Average response time vs. system load for the FFT communication pattern and uniform side lengths distribution in a 16 × 16 mesh

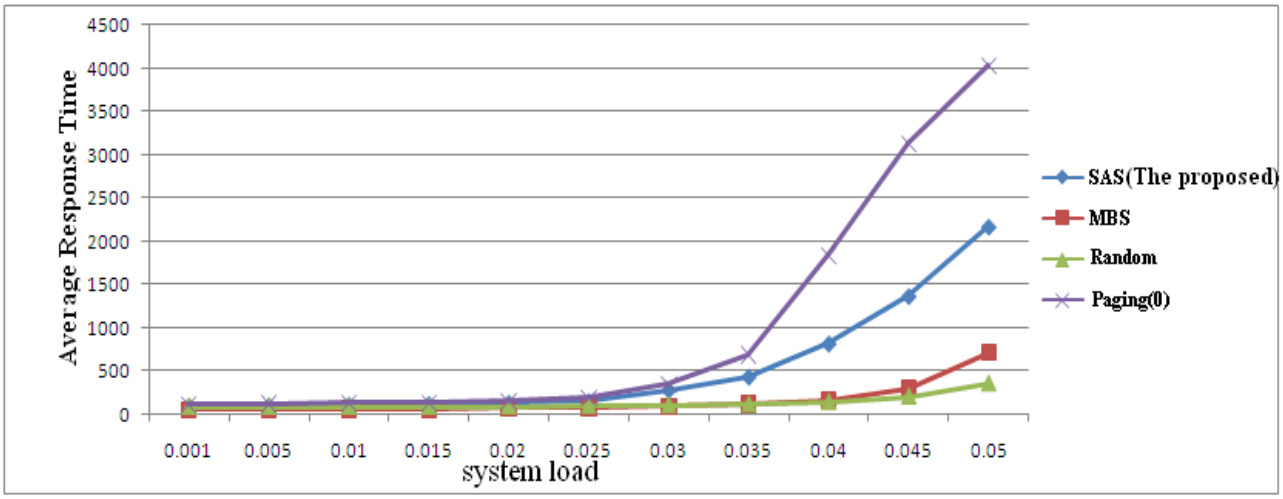


Figure 4.16: Average response time vs. system load for the FFT communication pattern and uniform side lengths distribution in a 20 × 20 mesh

In Figures 4.17, 4.18, 4.19 and 4.20, the average response times of jobs are plotted against the system load for the Divide and Conquer Binomial Tree communication pattern. The results show that the SAS performs better than Random strategy, but it is worse than both MBS and Paging(0) strategies. Furthermore the results show that MBS strategy is substantially superior to the other strategies. For example, in figure 4.17 the average response times of the SAS strategy is increase about 86% compared

www.manaraa.com

to MBS and 29% to Paging (0) under the job arrival rate of 0.05 jobs/time units under the job arrival rate of 0.05 jobs/time units.

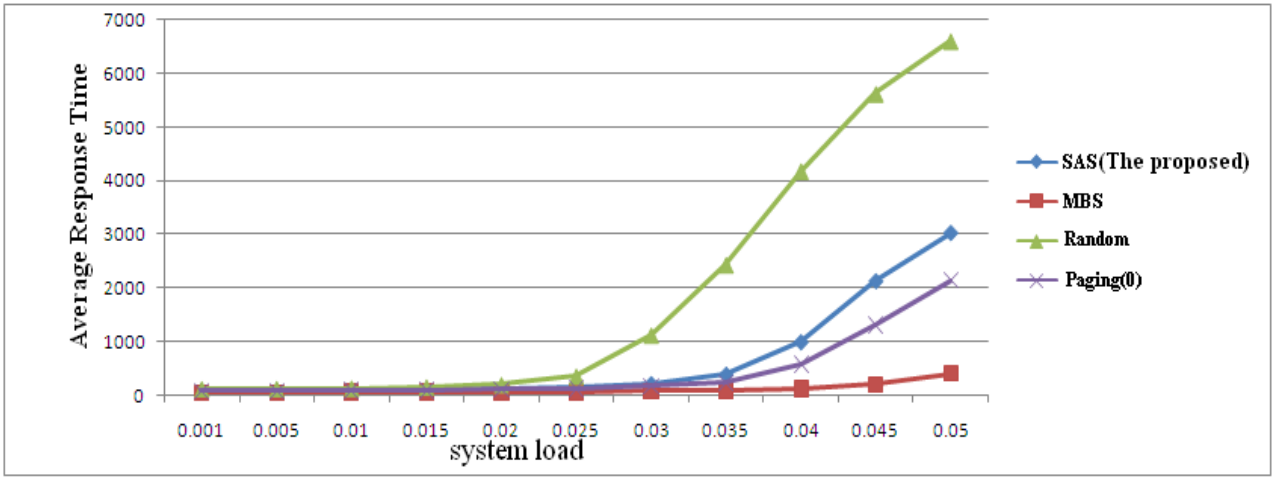


Figure 4.17: Average response time vs. system load for the Divide and Conquer communication pattern and uniform side lengths distribution in a 8 × 8 mesh

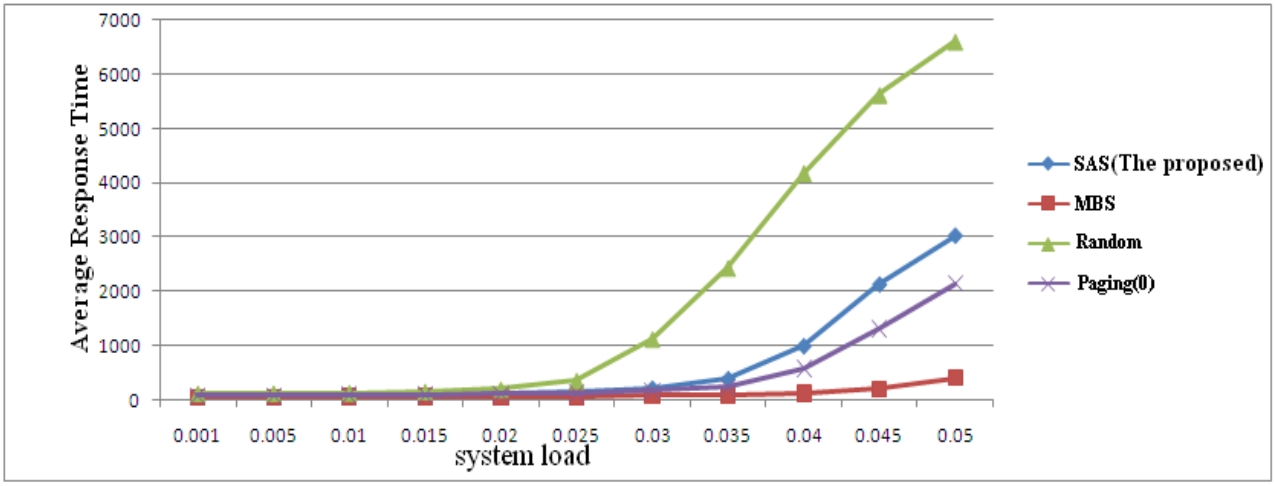


Figure 4.18: Average response time vs. system load for the Divide and Conquer communication pattern and uniform side lengths distribution in a 12 × 12 mesh

www.manaraa.com

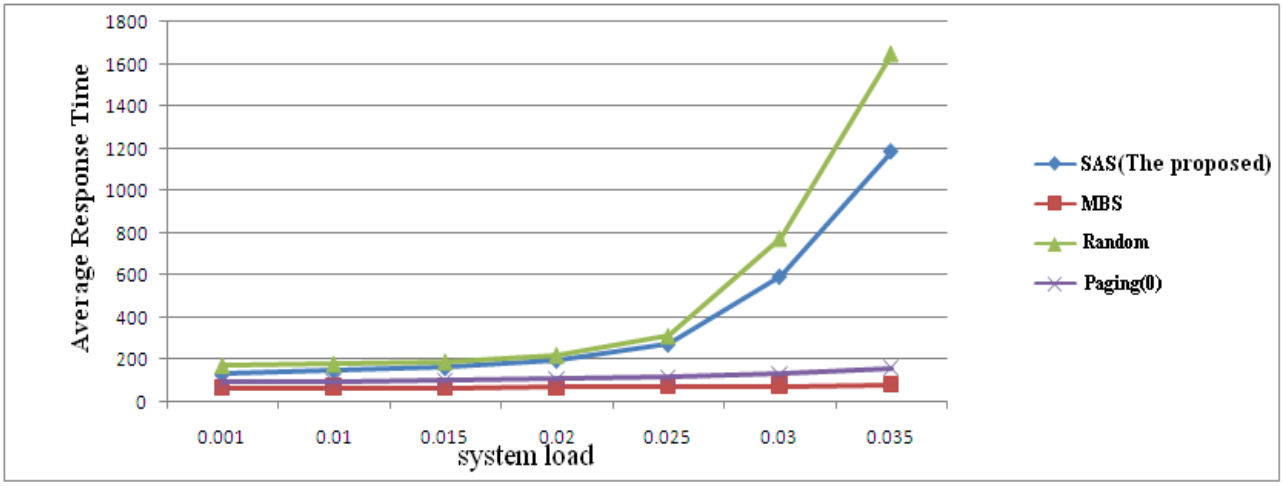


Figure 4.19: Average response time vs. system load for the Divide and Conquer communication pattern and uniform side lengths distribution in a 16 × 16 mesh

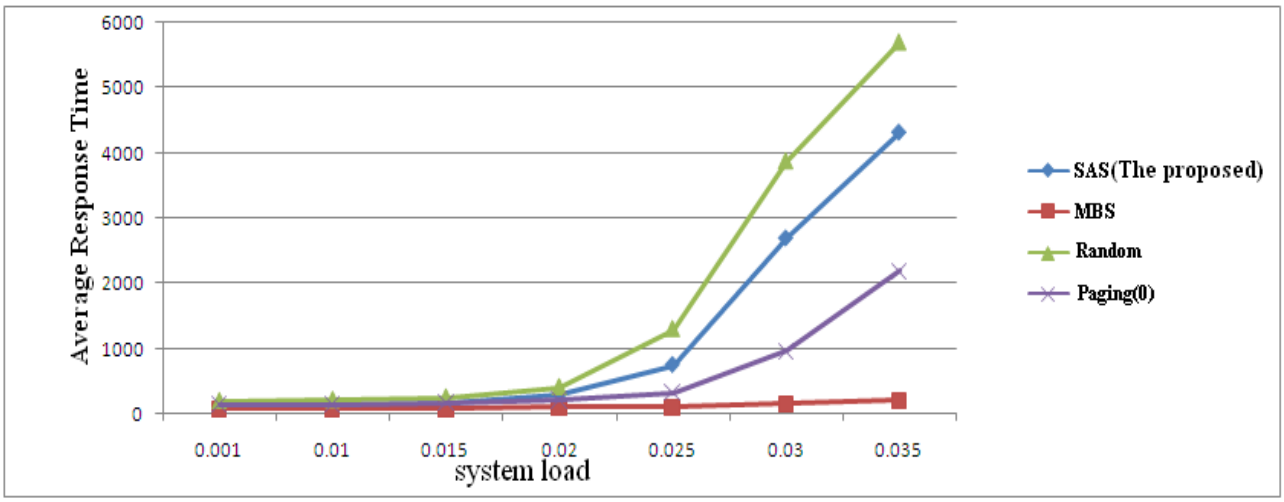


Figure 4.20: Average response time vs. system load for the Divide and Conquer communication pattern and uniform side lengths distribution in a 20 × 20 mesh

In Figures 4.21, 4.22, 4.23 and 4.24, the average response times of jobs are plotted against the system load for the All_To_One communication pattern. The results show that SAS performs better than all other noncontiguous allocation strategies considered across the mesh sizes 8 × 8 ,12×12 and 16 × 16 as shown in figures 4.21,4.22 and 4.23. But in figures 4.24 the SAS strategy performs better than Paging (0) and MBS strategies and worse than the Random strategy. For instance, when the job arrival rate

www.manaraa.com

is 0.025 jobs/time units, as shown in figure 4.22 the average response time of SAS is about 13%, 16%, and 12% of that of MBS, Random, and Paging (0) respectively.

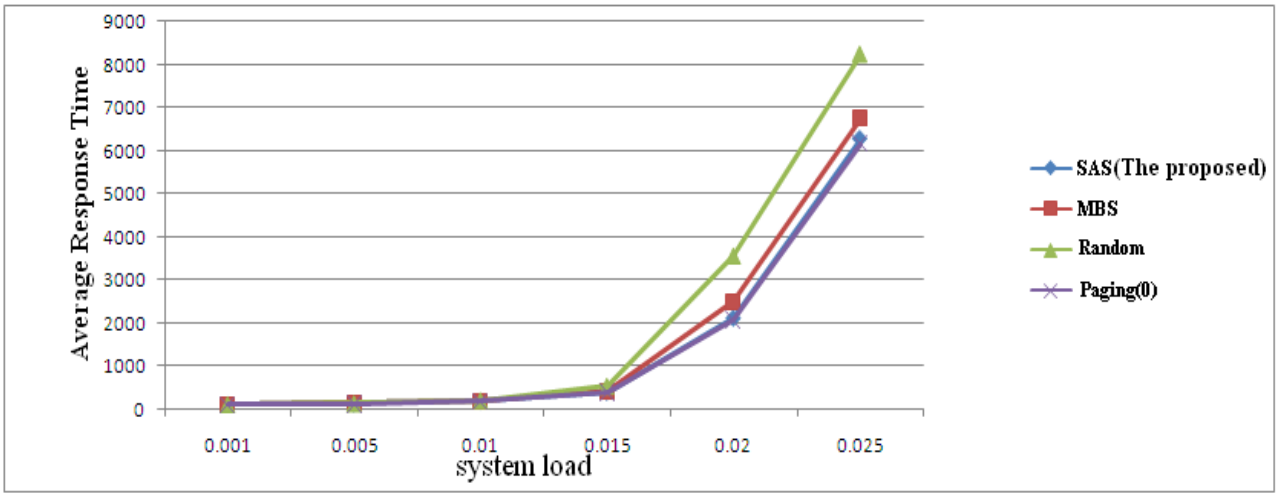


Figure 4.21: Average response time vs. system load for the All _to_One communication pattern and uniform side lengths distribution in a 8 × 8 mesh

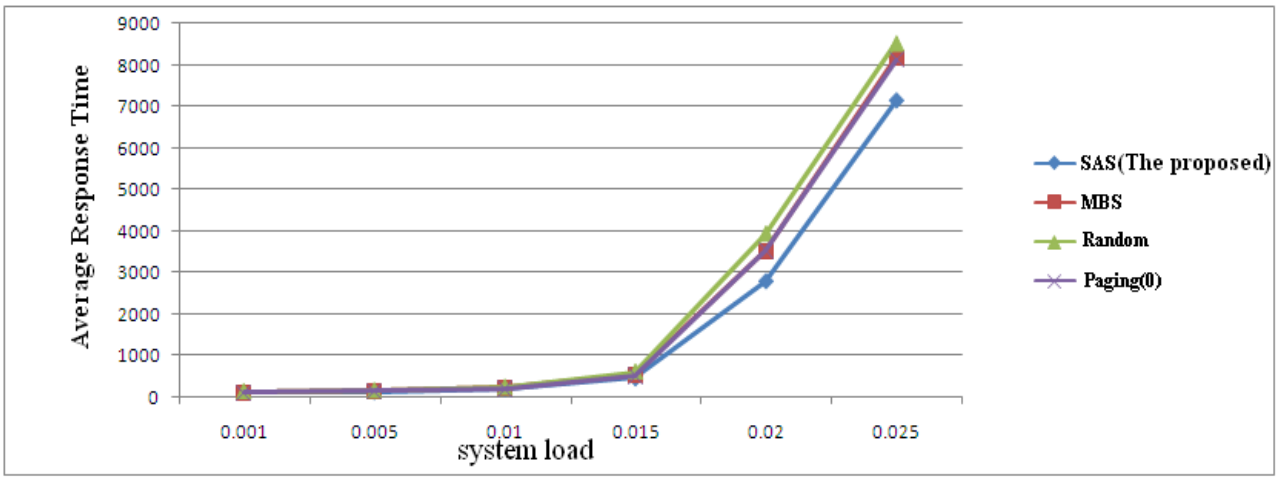


Figure 4.22: Average response time vs. system load for the All _to_One communication pattern and uniform side lengths distribution in a 12 × 12 mesh

www.manaraa.com

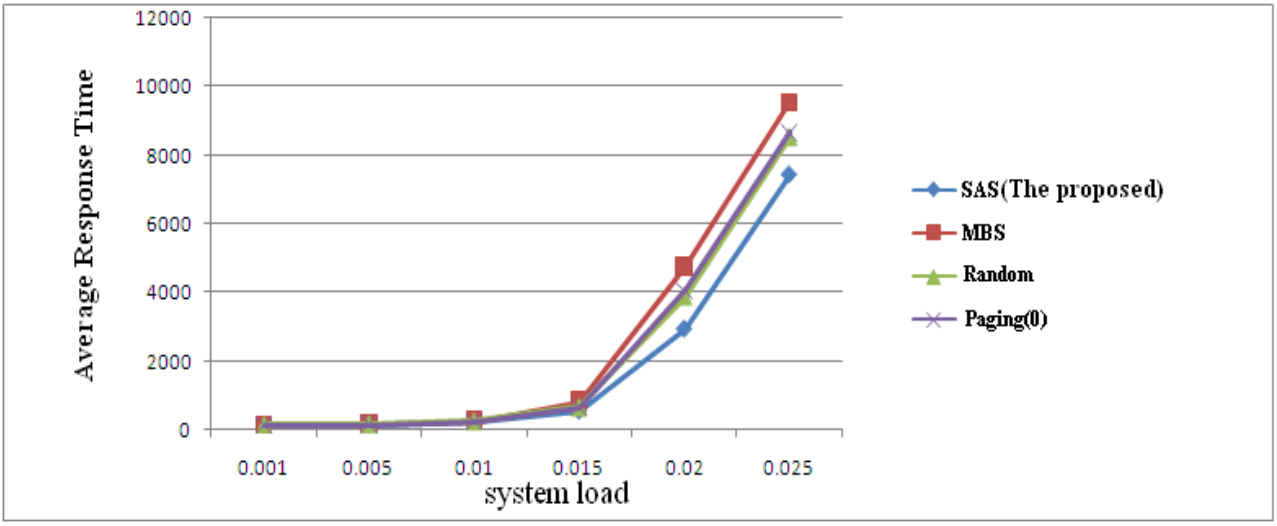


Figure 4.23: Average response time vs. system load for the All _to_One communication pattern and uniform side lengths distribution in a 16 × 16 mesh

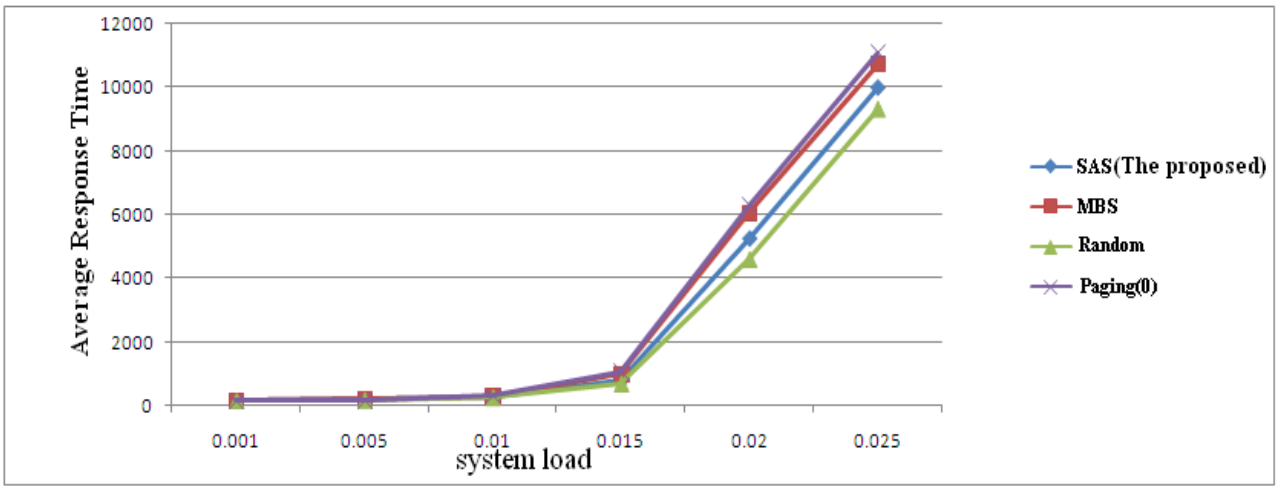


Figure 4.24: Average response time vs. system load for the All _to_One communication pattern and uniform side lengths distribution in a 20 × 20 mesh

In Figures 4.25, 4.26, 4.27 and 4.28, the average response times of jobs are plotted against the system load for the Ring communication pattern. The results show that the SAS performs better than Random strategy, but it is worse than both MBS and Paging(0) strategies as show in figures 4.26 ,4.27 and 4.28. Furthermore the results show that SAS strategy is substantially superior to the other strategies considered across the mesh sizes 8 × 8 as shown in figure 4.25, For example, the average

www.manaraa.com

response time of SAS is about 26%, 66%, and 30% of that of MBS, Random, and Paging (0) respectively under the job arrival rate of 0.025 jobs/time units.

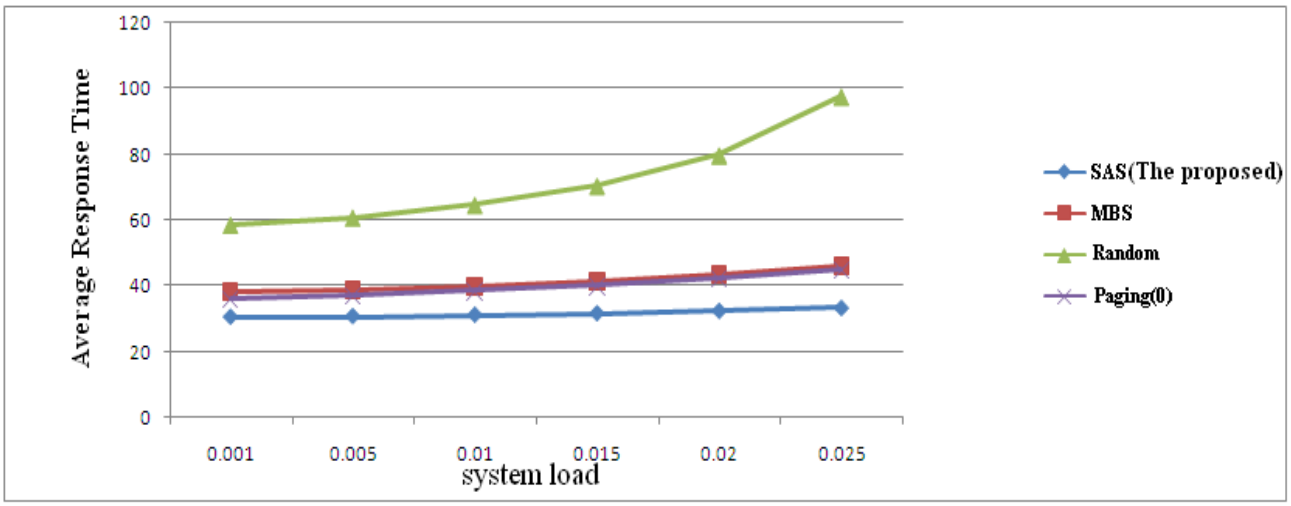


Figure 4.25: Average response time vs. system load for the Ring communication pattern and uniform side lengths distribution in a 8 × 8 mesh

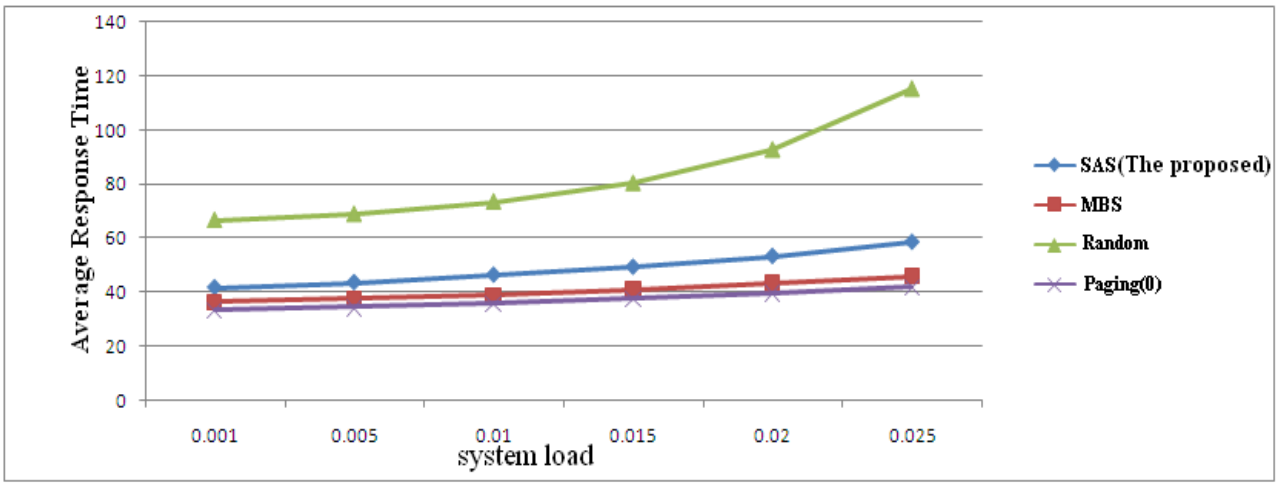


Figure 4.26: Average response time vs. system load for the Ring communication pattern and uniform side lengths distribution in a 12 × 12 mesh

www.manaraa.com

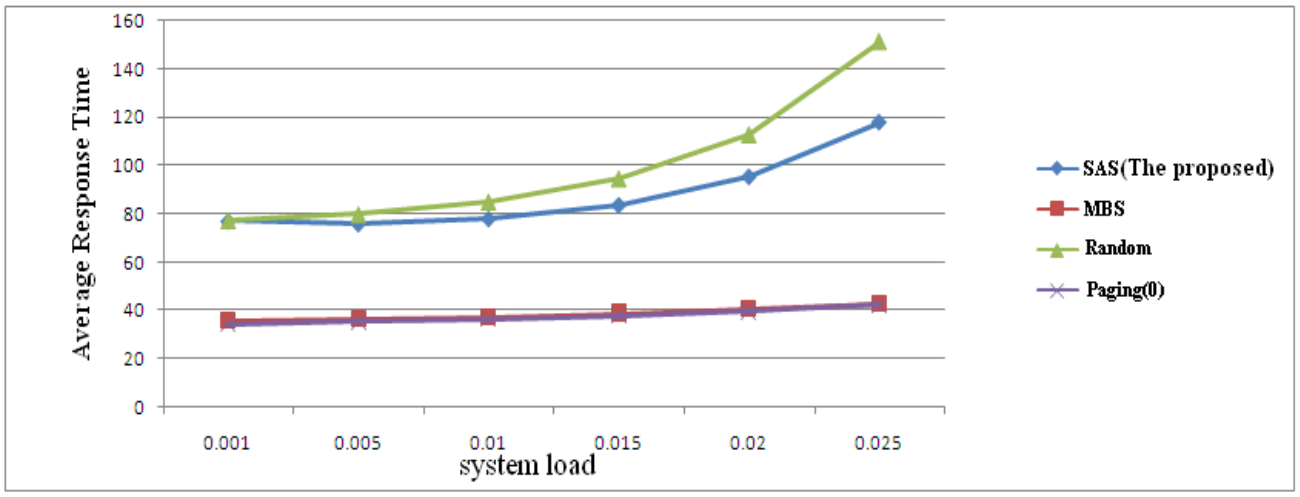


Figure 4.27: Average response time vs. system load for the Ring communication pattern and uniform side lengths distribution in a 16 × 16 mesh

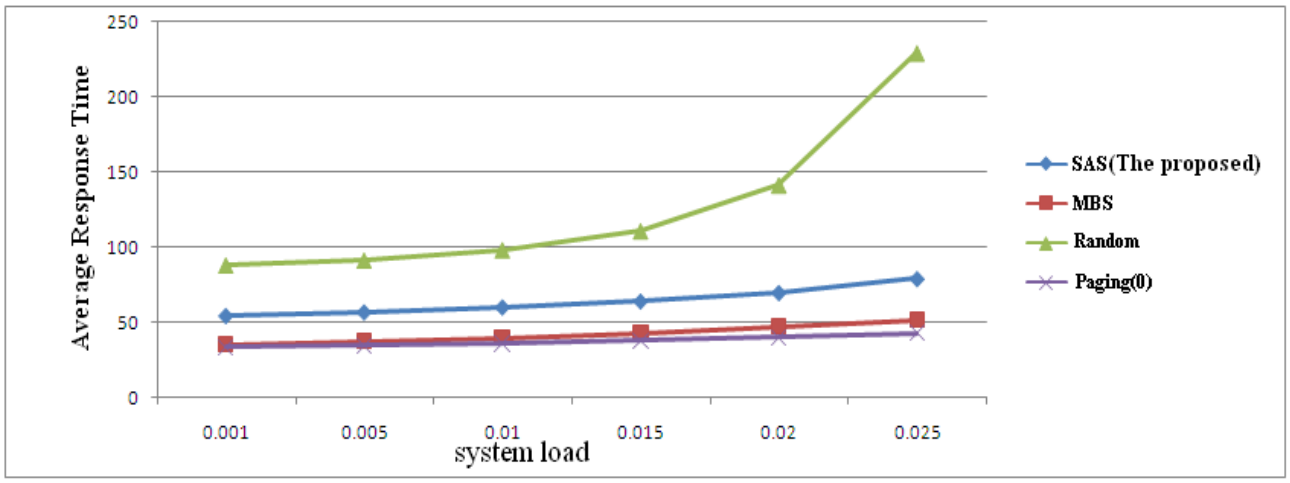


Figure 4.28: Average response time vs. system load for the Ring communication pattern and uniform side lengths distribution in a 20 × 20 mesh

In Figures 4.29, 4.30, 4.31 and 4.32, the average response times of jobs are plotted against the system load for the NAS Multigrid Benchmark communication pattern. The results show that the SAS performs better than Random strategy, but it is worse than both MBS and Paging(0) strategies. However the performance of MBS is very close to that of the non-contiguous Paging(0) strategy. In figure 4.30, for example, the average response times of the SAS strategy increase by about 97% compared to MBS and 96% to Paging (0) under the job arrival rate of 0.1 jobs/time units.

www.manaraa.com

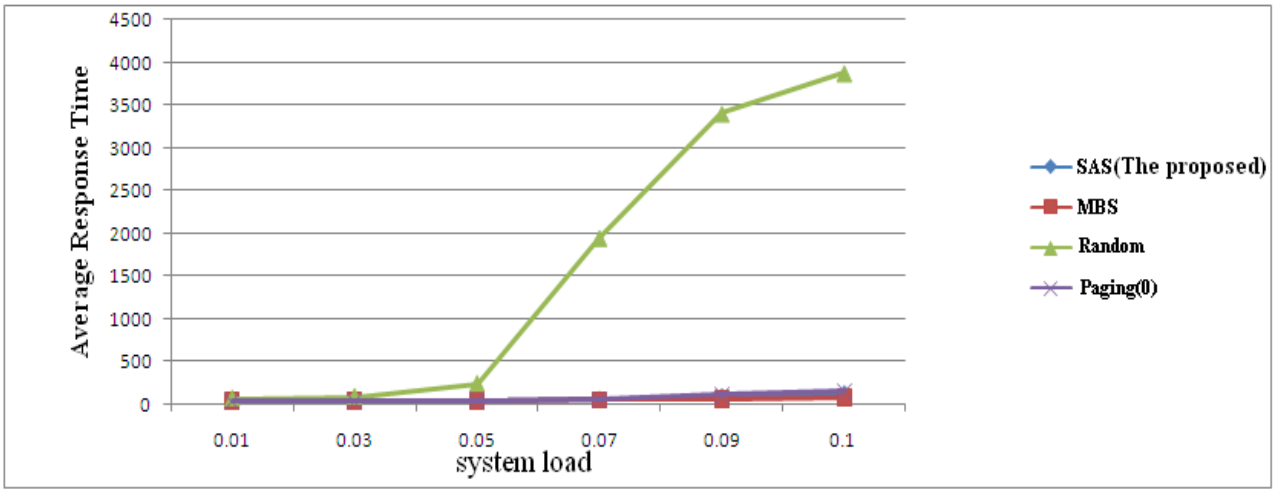


Figure 4.29: Average response time vs. system load for the NAS Multigrid Benchmark communication pattern and uniform side lengths distribution in a 8 × 8 mesh

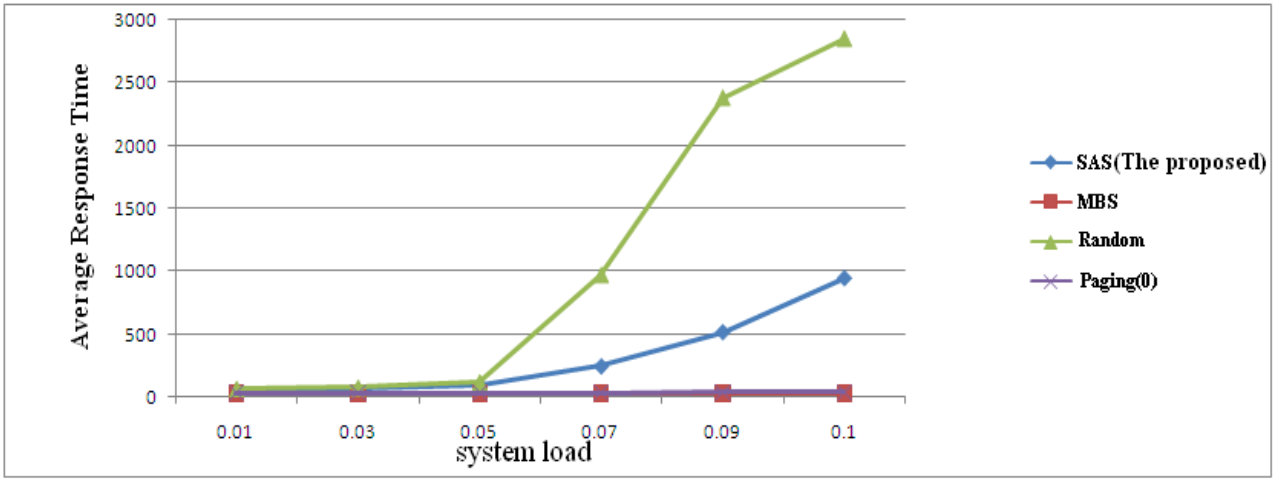


Figure 4.30: Average response time vs. system load for the NAS Multigrid Benchmark communication pattern and uniform side lengths distribution in a 12 × 12 mesh

www.manaraa.com

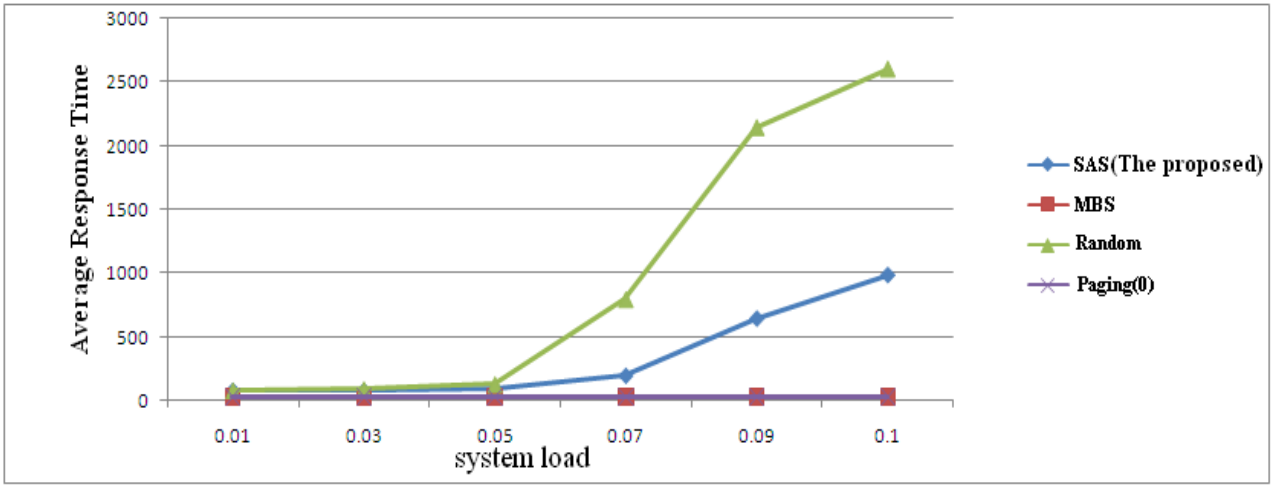


Figure 4.31: Average response time vs. system load for the NAS Multigrid Benchmark communication pattern and uniform side lengths distribution in a 16 × 16 mesh

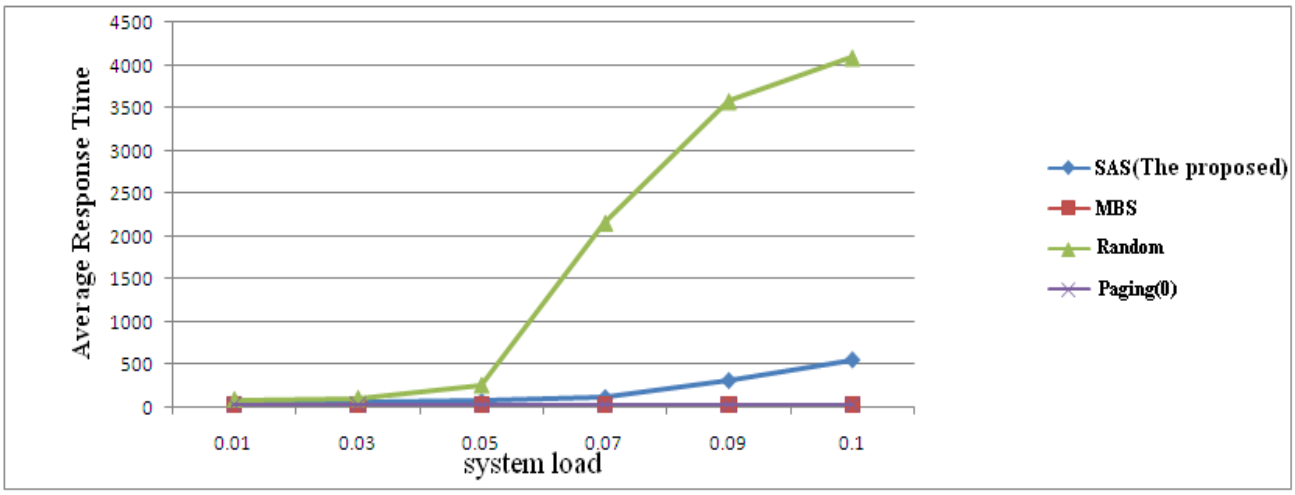


Figure 4.32: Average response time vs. system load for the NAS Multigrid Benchmark communication pattern and uniform side lengths distribution in a 20 × 20 mesh

www.manaraa.com

Chapter 5

Conclusions and Future Research

www.manaraa.com

5.1 Conclusions

We have suggested a new non-contiguous allocation strategy, referred to as the spiral allocation strategy (SAS), which differs from the earlier non-contiguous allocation strategies in the method used for choosing the allocated processors. The SAS strategy starts the allocation process at the central node and allocates free processors that it counters during a squared spiral anticlockwise scan around the central node. The goal is to allocate nodes that are close together, which decreases the distance traversed by messages.

The performance of SAS was compared against that of several existing non-contiguous strategies. The simulation results show that SAS performs better in some cases. For example, when using the one-to-all and all-to-one communication pattern, SAS exhibits superior performance over the previous non-contiguous allocation strategies.

5.2 Future Works

In this research, the performance of the SAS strategy has been evaluated assuming the First Come First Served (FCFS) scheduling strategy. As a continuation of this research in the future, it would be interesting to evaluate the performance with different scheduling approaches, such as smallest job first [7] or window-based job scheduling [13].

Also, as a future work of this research it can be useful to implement a hybrid processor allocation strategy that combines SAS and other contiguous or non-contiguous processor allocation strategies.

www.manaraa.com

References

[1] S. Bani-Mohammad, I. Ababneh and M. Hamdan,"Comparative Performance Evaluation of Non-Contiguous Allocation Algorithms in 2D Mesh-Connected Multicomputers", IEEE International Conference on Scalable Computing and Communications (ScalCom 2010) ،Bradford, UK, pp. 2933- 2939, 2010.

[2] S. Bani-Mohammad, M. Ould-Khaoua, and I. Ababneh, "An Efficient Non-Contiguous Processor Allocation Strategy for 2D Mesh Connected Multicomputers", Journal of Information Sciences, vol. 177, no. 14, pp. 2867-2883, 2007.

[3] R. almomani, I. Ababneh," Communication overhead in non-contiguous processor allocation policies for 3D mesh-connected multicomputers", The International Arab Journal of Information Technology, vol. 9, no. 2, pp. 133-141, 2012.

[4] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, and Lewis M. Mackhenzie, "An Efficient Processor Allocation Strategy that Maintains a High Degree of Contiguity among Processors in 2D Mesh Connected Multicomputers",IEEE Computer Society Press, pp. 934- 941, 13-16 May 2007.

[5] I. Foster, "Designing and Building Parallel Programs, Concepts and Tools for Parallel Software Engineering", Addison-Wesley, 1995.

[6] Hennessy-Patterson – "Computer Architecture. A.Quantitative.Approach".4th.ed.2007

[7] ProcSimity V4.3 User's Manual, University of Oregon, 1997.

[8] J.E. Savage, "Models of Computation: Exploring the Power of Computing", Addison-Wesley, Reading, MA (1998)

[9] V. Kumar, A. Grama, A. Gupta, and G. Karypis, "Introduction to Parallel Computing, Person education Limited",2nd edition, (2003)ISBN 0-201-64865-2.

[10] S. Bani-Ahmad, "Processor Allocation with Reduced Internal and External Fragmentation in 2D Mesh-based Multicomputer". Journal on Applied Sciences, vol. 11, no. 6, pp. 943–952 (2011) ISSN 1812-5654.

[11] S. M. Yoo, H. Choo, H. Y. Youn, C. Yu, and Y. Lee. "On Task Relocation in Two-dimensional Meshes". In Journal of Parallel and Distributed Computing, vol. 60, no. 5, pp. 616 – 638, 2000.

[12] M. Morris Mano, "Computer System Architecture, Person Education Limited", 3rd edition,(1992) ISBN 0-13-175563-3.

[13] I. Ababneh, S. Bani-Mohammad, "A New Window-Based Job Scheduling Scheme for 2D Mesh Multicomputers", Simulation Modelling Practice Theory, vol.19, no.1, pp.482–493, 2011

www.manaraa.com

[14] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, and Lewis M. Mackhenzie, "Non-contiguous Processor Allocation Strategy for 2D Mesh Connected Multicomputers based on Sub-meshes Available for Allocation", Proceedings of the 12th International Conference on Parallel and Distributed Systems (ICPADS'06), IEEE Computer Society Press, USA, vol. 2, pp.41-48,2006.

[15] V. Lo, K. Windisch, W. Liu, and B. Nitzberg, "Non-contiguous processor allocation algorithms for mesh-connected multicomputers", IEEE Transactions on Parallel and Distributed Systems, vol. 8, no. 7, pp. 712-726, 1997.

[16] Y. H. Zhu, "Efficient processor allocation strategies for mesh-connected parallel computers", Journal of Parallel and Distributed Computing, vol. 16, no. 4, pp. 328-337, 1992.

[17] I. Ababneh, An Efficient Free-list Submesh Allocation Scheme for two-dimensional mesh-connected multicomputers, Journal of Systems and Software, vol. 79, no. 8, pp.1168-1179, 2006.

[18] I. Ababneh, Availability-based noncontiguous processor allocation policies for 2D mesh-connected multicomputers, Journal of Systems and Software, vol. 81, no. 7, pp. 1081–1092, 2008.

[19] P. Mohapatra, Wormhole routing techniques in multicomputer systems, *ACM Computing Surveys*, vol. 30, no. 3, pp. 375-411, 1998.

[20] L. M. Ni and P. K. McKinley, A survey of wormhole routing techniques in direct networks. IEEE Computer, vol. 26, no. 2, pp. 62-76, 1993.

[21] Sulieman Bani-Ahmad"Submesh Allocation in 2D-Mesh Multicomputers: Partitioning at the Longest Dimension of Requests" The International Arab Journal of Information Technology, Vol. 10, No. 3, pp. 245-252, 2013

www.manaraa.com

الملخص

اقترحت العديد من استراتيجيات التخصيص المتجاور والغير متجاور في مجال متعددات الحواسيب الشبكية. حيث تعاني استراتيجيات التخصيص المتجاور من مشكلة الكسيرات الخارجية وذلك لأن المعالجات المخصصة للوظيفة يجب أن تكون متجاورة ولها نفس الشكل. والغرض من رفع شرط التجاور في التخصيص غير المتجاور هو تقليل مشكلة الكسيرات. ومع ذلك، ممكن أن يزيد الحمل الزائد في الاتصال وذلك لان المسافات التي تجتازها الرسائل بين المعالجات المخصصة للوظيفة يمكن أن تكون أطول، والرسائل من وظائف مختلفة يمكن أن تتداخل مع بعضها البعض. يعتمد الحمل الزائد في الاتصال على كيفية تقسيم طلب التخصيص وتعيين المعالجات الحرة.

في هذا البحث تم اقتراح إستراتيجية تخصيص غير متجاور جديد واسمها إستراتيجية التخصيص الحلقي حيث تكون البداية من منتصف الشبكة ثم يقوم بمسح على المعالجات الحرة باستخدام البحث الحلقي حول المنتصف. وباستخدام برنامج المحاكاة تم مقارنة إستراتيجية التخصيص الحلقي مع الاستراتيجيات السابقة الغير متجاورة، ظهرت النتائج أن التخصيص الحلقي أفضل نسبيا من الاستراتيجيات السابقة من حيث انخفاض متوسط الوقت الذي تستغرقه الوظيفة في الشبكة.

المنارة للاستشارات
www.manaraa.com