Al Al-Bayt University

Prince Hussein bin Abdullah Faculty of Information Technology

A Horizontal Partitioning Non-Contiguous Processor Allocation Strategy for 2D Mesh-Connected Multicomputers

استراتيجية التخصيص غير المتجاور باستخدام التقسيم الافقي في متعددات الحواسيب الشبكية ثنائية الابعاد

Supervised by

Dr. Saad Bani-Mohammad

Presented by

Nabeel Khalid AbuOlaim

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

2017

Committee Decision

This Thesis (A Horizontal Partitioning Non-Contiguous Processor Allocation Strategy for 2D Mesh-Connected Multicomputers) was successfully defended and approved on 31/12/2007.

Examination Committee	Signature
Dr. Saad Bani-Mohammad (Supervisor)	…….
Prof. Ismail Ababneh	…….
Dr. Akram Hamarsheh	…….
Dr. Wail Mardini	…….

Dedication

To the memory of my mother, who always believed in my ability to be successful.

To the memory of my father, my idol.

Acknowledgments

First of all, my great thanks to my supervisor, Dr. Saad Bani-Mohammad for his support and encouragement during the completion of this work. In many stages of this research I benefited from his advice. His positive outlook and confidence in my research inspired me and gave me confidence. His careful editing contributed enormously to the production of this thesis.

I would like to thank my wife, my children, my brothers and sisters, and I ask my God to give them what they wish.

I would also like to thank my friend and colleague, Mr. Daifallah Al-Sardi, for supporting me throughout the writing of this thesis.

I thank all of them.

Table of Contents

Contents

Table of Figures

List of Tables

A Horizontal Partitioning Non-Contiguous Processor Allocation Strategy for 2D Mesh-Connected Multicomputers

استراتيجية التخصيص غير المتجاور باستخدام التقسيم الافقي في متعددات الحواسيب الشبكية ثنائية الابعاد

Supervised by

Dr. Saad Bani-Mohammad

Presented by

Nabeel Khalid AbuOlaim

Abstract

Processor allocation strategies that had been devised for mesh-connected multicomputers are classified into two types, contiguous and non-contiguous. In contiguous strategies, the allocated processors must be physically adjacent, and form a contiguous shape similar to the original topology. Contiguous allocation suffers from both external and internal fragmentation problems. In non-contiguous allocation strategies, the job request can execute on multiple separated smaller sub-meshes rather than waiting until a single sub-mesh of the requested size and shape is available.

There are many proposed strategies for non-contiguous allocation that have different levels in working to balance between distributing the requested job among the processors in the mesh, and keeping a good level of contiguity between the allocated processors.

In this research, we have proposed a new non-contiguous processor allocation strategy for 2D mesh-connected multicomputers, referred to as Horizontal Partitioning Strategy (HPS) that partitions the job request based on the sub-meshes available for allocation in the system so as to maintain some degree of contiguity. These sub-meshes are called Free-rows, and each of them represents a row of free processors that is equal to the width of the mesh. HPS strategy rebuilds the job request to accommodate in Free-rows and it always tries to allocate a job request contiguously in Free-rows in order to decrease the distance traversed by a message, and hence reduce the message contention inside the network.

Using simulation, we compared the performance of HPS with the existing well-known non-contiguous allocation strategies Paging(0), MBS, and GABL. The results show that the performance of HPS allocation strategy is much better than that of other non-contiguous allocation strategies for both job size distributions considered when the all-to-all communication pattern is used, and it is close to that of the non-contiguous allocation strategies considered when the one-to-all and random communication patterns are used. Moreover, HPS exhibits high system utilization as it manages to eliminate both internal and external fragmentation.

Chapter 1
1. Introduction

1.1. Overview

Progress in technology has made the technique of constructing parallel computers with a large number of processors viable, and the availability of reasonably priced processors and high-speed networks have made it viable to develop programs that use distributed resources (Foster, 1995).

A parallel computer is primarily based at the concept of employing multiple resources to solve a specific problem. To implement this concept, it is necessary to divide the problem into small problems, after which the available resources are allocated to these problems in order to resolve those small problems within the form of general solution, where these partial problems may be separated or can also be overlapped (Foster, 1995; Kumar, Grama, Gupta, & Karypis, 2003).

The vast computational power of parallel computers greatly reduces the time of executing the program, which enables solutions to huge and complicated problems that had been lengthy or intractable without parallel processing, such as weather forecasting management in the atmosphere (Foster, 1995; Kumar, Grama, Gupta, & Karypis, 2003).

A Parallel computer is a set of processors that can cooperate with each other to achieve parallel processing for diverse computational needs, and a parallel program is the program that can be executed on multiprocessors (Foster, 1995; Bani Mohammad, 2008). In parallel computing, we are able to save time, resolve larger problems, and reduce cost by means of using multiple “cheap” computing resources in place of paying for time on a supercomputer (Foster, 1995; Kumar, Grama, Gupta, & Karypis, 2003).

Parallel computers are divided into two classes: Shared memory computer systems and distributed memory computer systems. In Shared memory computers, additionally known as multiprocessors, all processors share access to a common memory, typically via a bus interconnection network. In distributed memory computers, additionally known as multicomputers, each processor in the system has its own local memory and can communicate with the other processors by sending messages through an interconnected network (Bani Mohammad, 2008).

Parallel computers are constructed by connecting processors and memory to hold data using various interconnection networks. These networks can be categorized into two types: direct connection networks and indirect connection networks. Direct networks have point-to-point communication links among processing nodes, and these networks are static, which means that the nodes are directly connected to each other (i.e., point-to-point connection). Some examples of direct networks are star-connected network, linear arrays, and meshes (Kumar, Grama, Gupta, & Karypis, 2003). In indirect networks, the nodes are connected to each other via switches (Kumar, Grama, Gupta, & Karypis, 2003; Mohapatra, 1998), and it may be subdivided into three parts: bus networks, multistage networks and crossbar switches (Kumar, Grama, Gupta, & Karypis, 2003).

A 2D mesh interconnection network is an example of direct networks, where each node in the system has a direct connection to its neighbors. The mesh network is straightforward, scalable to implement and popular in multicomputers (Adve & Vernon, 1994).

Figure 1.1 shows an example of a 6×6 2D mesh, where allocated processors are denoted by black circles and free processors are denoted by white circles.

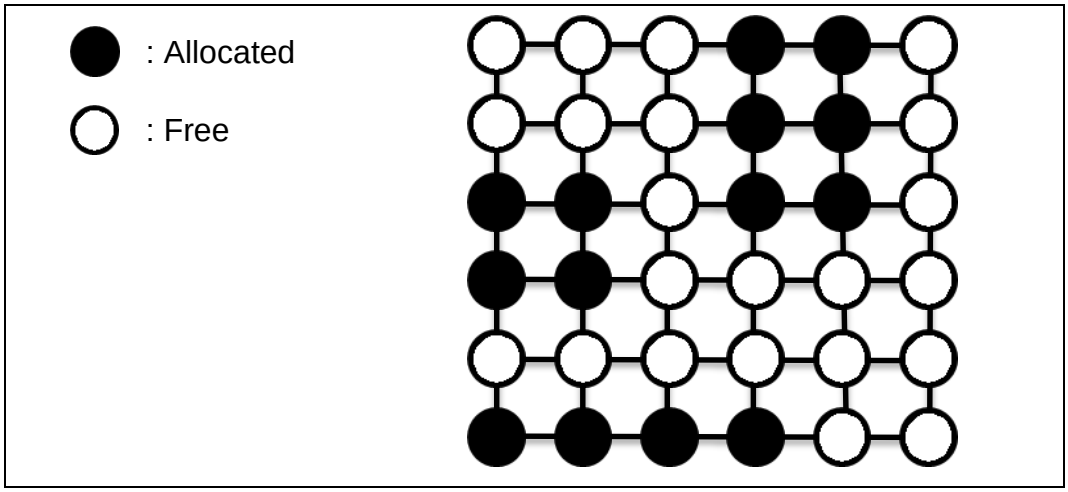


Figure 1-1: An example of a 6×6 2D mesh

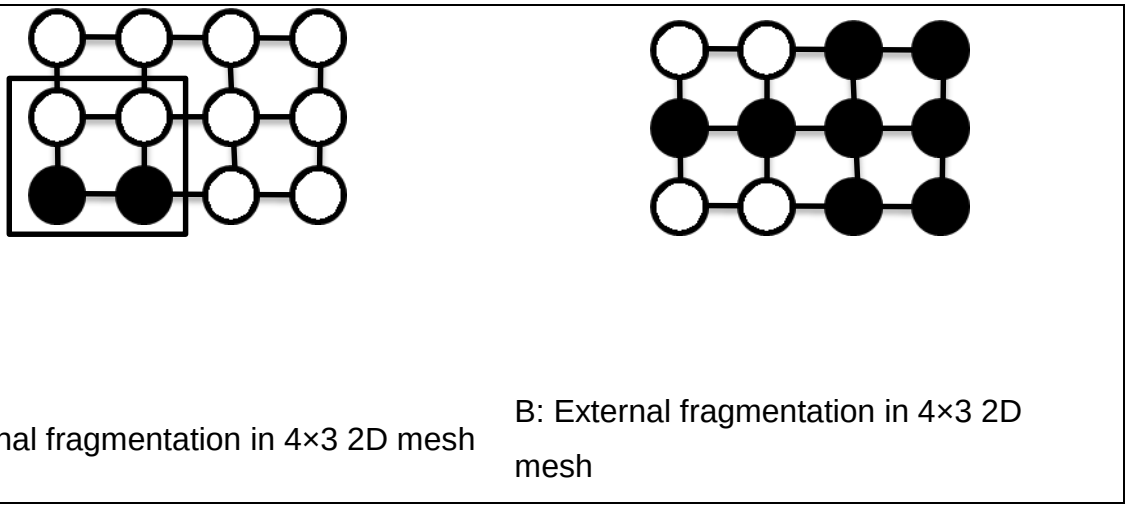
The performance of a multicomputer system depends on the processor allocation and job scheduling strategies used. The processor or sub-mesh allocator is responsible for allocating a certain sub-mesh of free processors to incoming jobs, where the sub-mesh is held by the job for a particular computation and released after computation, while the job scheduler is responsible for determining the order of the jobs to be scheduled (Babbar & Krueger, 1994; Bani Mohammad, 2008).

1.2. Processor Allocation

Many processor allocation strategies had been devised for mesh-connected multicomputers, and these can be classified into two main categories: contiguous and non-contiguous allocation strategies (Bani Mohammad, 2008).

Contiguous strategies depend mainly on the processors that will be allocated to a job in mesh network and that they must be physically adjacent, and form a contiguous shape according to the original topology (Lo, Windisch, Liu, & Nitzberg, 1997). This causes external and internal fragmentation problems. The processor allocation strategy produces external fragmentation when there are free processors enough in number to satisfy the job request, but they are not allocated to it because they are not contiguous, while internal fragmentation occurs when the allocation strategy allocates more processors than its requested (Lo, Windisch, Liu, & Nitzberg, 1997; Bani Mohammad, 2008).

Figure 1.2-A shows an example of internal fragmentation when four processors are assigned to a new job that only requests two processors, the black frame shows the set of allocated processors for the new job request, while the number of requested processors is shown in black circles. Figure 1.2-B shows an example of external fragmentation, assuming a contiguous allocation strategy is used, where the incoming job requests four processors that are available in the system, but these processors are not allocated to the job because they are not contiguous.



A: Internal fragmentation in 4×3 2D mesh

B: External fragmentation in 4×3 2D mesh

Figure 1-2: Internal and External Fragmentation

The restriction that jobs have to be allocated to contiguous processors reduces the chance of successfully allocating a job, and it is viable for the allocation strategy to fail to allocate a job even as there are enough number of processors available in the mesh system (Bani Mohammad, 2008; Bani-mohammad, Ould-KHaoua, Ababneh, & Mackenzie, 2006).

In non-contiguous allocation strategies, a job can execute on multiple separated smaller sub-meshes rather than waiting until a single sub-mesh of the requested size and shape is available. There are many proposed strategies for non-contiguous allocation that differ in distributing the requested job among the processors in the mesh. Although these strategies keep a level of contiguity between the allocated processors, they may cause high communication overhead by increasing message contention inside the network (Ababneh & Almomani, 2012; Bani Mohammad, 2008). However, dropping the contiguity condition can reduce processor fragmentation and increase system utilization, but it is able to cause high communication overhead, which may be alleviated by maintaining a good degree of contiguity between the allocated processors (Ababneh & Almomani, 2012; Lo, Windisch, Liu, & Nitzberg, 1997; Bani-mohammad, Ould-KHaoua, & Ababneh, 2007). The system utilization is the percentage of processors that are utilized over time (ProcSimity v4.3, 1996).

1.3. Motivation and Contribution

Many processor allocation strategies devised for 2D mesh-connected multicomputers suffer from several problems that include external fragmentation, internal fragmentation, and message contention in the network. The primary purpose of any non-contagious allocation strategy is to enhance the system performance with the aid of reducing the job turnaround time and maximizing the system utilization (Chang & Mohapatra, 1998; Lo, Windisch, Liu, & Nitzberg, 1997; Bani Mohammad, 2008), where job turnaround time is the time that the job spends in the system from arrival to departure (Bani-mohammad, Ould-KHaoua, Ababneh, & Mackenzie, 2006).

The performance of any non-contiguous allocation strategy is affected by the degree of contiguity between the allocated processors and also by the topology of these processors. This is because in a heavy system loads, the degree of contiguity between allocated processors affects on the communication overhead. When the degree of contiguity is increased, the communication overhead is decreased and thus the system performance is improved in terms of job turnaround time (Lo, Windisch, Liu, & Nitzberg, 1997; Bani-mohammad, Ould-KHaoua, & Ababneh, 2007).

Motivated by the above observations and the preceding research findings (Bani-Mohammad, Ababaneh, & Hamdan, 2010; Babbar & Krueger, 1994; Lo, Windisch, Liu, & Nitzberg, 1997; Bani Mohammad, 2008; Bani-mohammad, Ould-KHaoua, Ababneh, & Mackenzie, 2006), there is a need to propose a new non-contiguous allocation strategy to investigate a good degree of contiguity in order to improve the system performance. To achieve this, we propose a new non-contiguous allocation strategy suggested for 2D mesh-connected multicomputers that maintains a high degree of contiguity among allocated processors via partitioning the job request based totally on the sub-meshes available for allocation inside the system. These sub-meshes are called Free-rows, and each of them represents a row of free processors that equal the width of the mesh system. The proposed strategy continually tries to allocate a job request contiguously in Free-rows in order to decrease the distance traversed by jobs' messages, and therefore reduce message contention inside the network.

The proposed strategy maintains some degree of contiguity between the allocated processors, which in turn reduces the communication overhead between these processors, and hence improves system performance in terms of job turnaround time (Bani-Mohammad, Ababaneh, & Hamdan, 2010; Bani Mohammad, 2008).The performance of the proposed strategy is compared against that of the existing non-contiguous allocation strategies Random (Lo, Windisch, Liu, & Nitzberg, 1997), Paging(0) (Lo, Windisch, Liu, & Nitzberg, 1997), MBS (Lo, Windisch, Liu, & Nitzberg, 1997) and GABL (Bani Mohammad, 2008).

The simulation results display that our strategy performs much better than the previous non-contiguous allocation strategies considered in this thesis in terms of job turnaround time when the all-to-all communication pattern is used. This is because all-to-all communication is considered to be the weak point of the non-contiguous allocation strategies because it produces much message collision (Suzaki, et al., 1996; Alsardia, 2017), and our strategy has been proposed to alleviate this collision by maintaining some degree of contiguity between allocated processors.

The results also showed that the performance of our strategy is close to that of the non-contiguous allocation strategies considered when the one-to-all and random communication patterns are used. This is because in these communication patterns, the number of messages generated by jobs is small as compared with the all-to-all communication pattern and thus all the non-contiguous allocation strategies considered in this thesis including our proposed strategy have the same ability to alleviate the contention inside the network. Also, the performance of the proposed strategy is not better than that of the other non-contiguous allocation strategies considered when the near-neighbor communication pattern is used, and this is because the near-neighbor communication pattern is suitable for the strategies that keep a high degree of contiguity and maintain a rectangular form of the allocated sub-meshes (Alsardia, 2017).

1.4. Outline of the Thesis

The rest of the thesis is organized as follows:

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

Chapter 3 introduces the proposed allocation strategy, A Horizontal Partitioning Non-Contiguous Processor Allocation Strategy for 2D Mesh-Connected Multicomputers (HPS), and presents the definitions and features for the proposed strategy. The allocation and de-allocation process are explained in this chapter.

In chapter 4, the simulation experiments are analyzed and the performance of the proposed strategy has been compared against that of the existing well-known non-contiguous allocation.

Chapter 5 concludes this thesis and introduces some directions for future work.

Chapter 2
2. Background and Preliminaries

2.1. Related Work

There are many proposed non-contiguous allocation strategies for 2D mesh connected multicomputers and this is a brief review of them.

2.1.1. Random allocation strategy:

Random strategy has been proposed to allocate any size of job request that the mesh can consist, via allocating them randomly one by one inside the 2D mesh. This strategy gives relatively great system utilization, because it allocates the same size of the job request within the mesh and it does not cause any internal or external fragmentation, however, the random allocation strategy has excessive communication overhead because the contiguity among the allocated processors is not taken into consideration in this strategy (Lo, Windisch, Liu, & Nitzberg, 1997).

2.1.2. Paging allocation strategy:

In this strategy, initially the mesh is divided into pages, the page is the basic unit of allocation and it is a square sub-mesh of processors. The page size is equal to 2^{Page_size} , where the page_size is a positive integer. The order of the processors in the pages and the order of the pages within the mesh are determining by using four indexing scheme, row-major, shuffled row-major, snake-like and shuffled snake-like. The paging strategy is indicated as Paging indexing_scheme(page_size) (Lo, Windisch, Liu, & Nitzberg, 1997).

Figure 2-1 shows the four indexing schemes, where figure 2-1.A shows a row-major indexing, figure 2-1.B shows a shuffled row-major indexing, figure 2-1.C shows a snake-like indexing and figure 2-1.D shows a shuffled snake-like indexing.

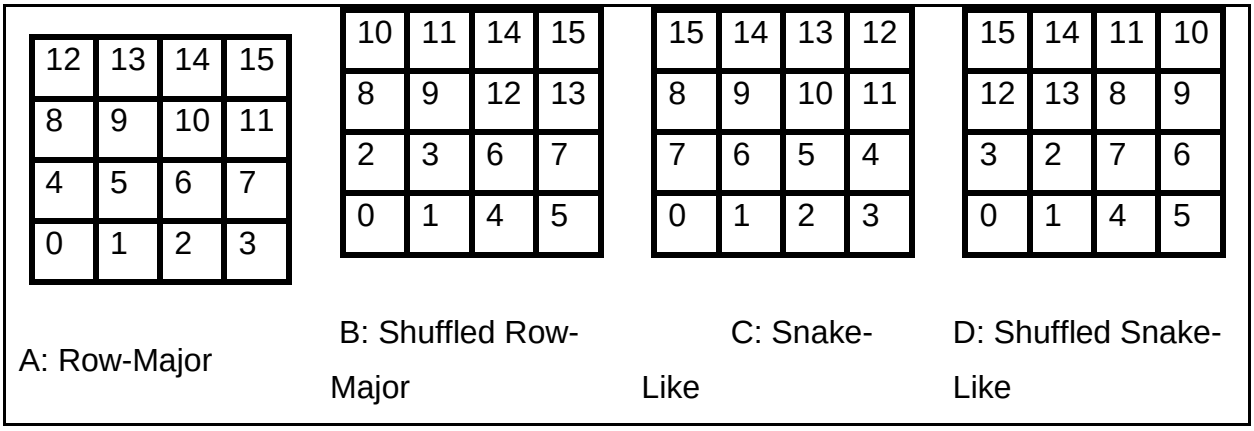


Figure 2- 1: Four different indexing schemes used by the Paging strategy

A request for n processors is accomplished by allocating:

$\left\lceil \frac{n}{2^{Page\_size} * 2^{Page\_size}} \right\rceil$ free pages.

Figure 2-2 shows an example of the Paging allocation, which uses the snake-like indexing scheme and a page size of one (2×2 blocks) indicating as Pagingsnake-like(1). Assume a job requests for ten processors. By using the Free Page List (FPL), which is an ordered list that keeps track of all the unallocated pages (Lo, Windisch, Liu, & Nitzberg, 1997), the first three items in FPL (3rd, 4th and 6th) as shown in figure 2-2 are removed and the twelve processors are allocated, and this causes an internal fragmentation of 16%.

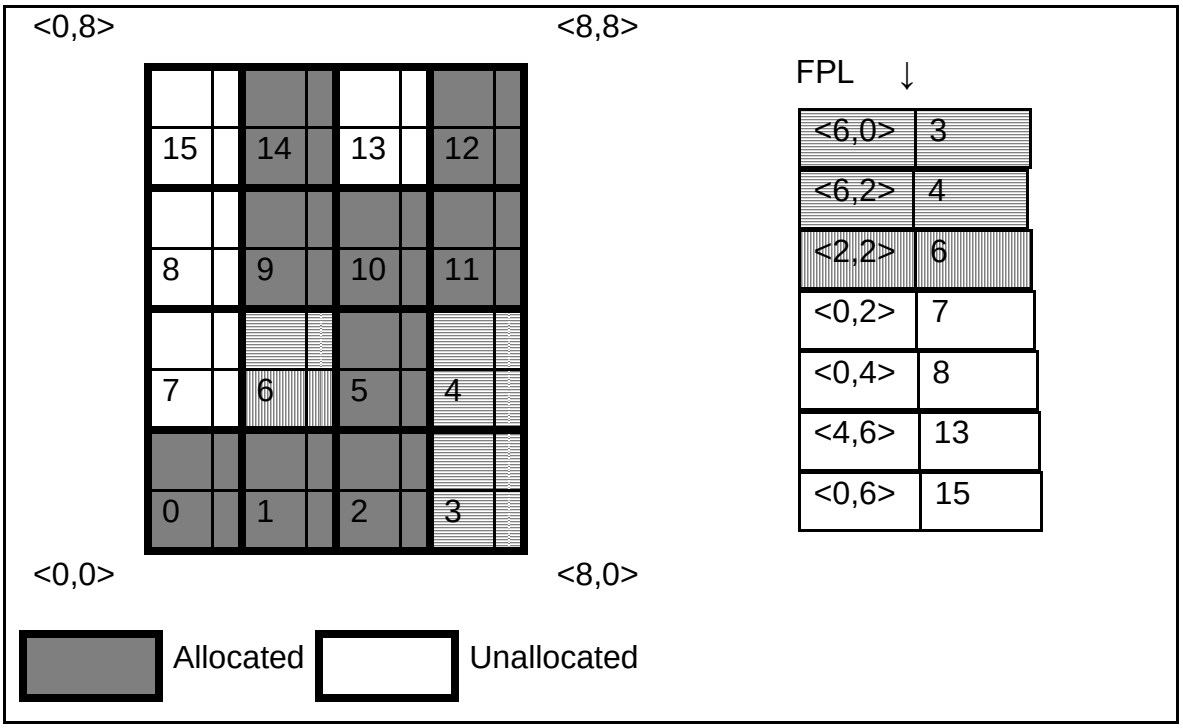


Figure 2- 2: Example of mesh and free page list for the Paging snake-like (1) allocation

This strategy gives some contiguity by allocating processors into pages, but this causes an internal fragmentation when the page_size > 0, and there is no internal or external fragmentation when page_size is 0 (Lo, Windisch, Liu, & Nitzberg, 1997; Bani-Mohammad, Ababaneh, & Hamdan, 2010).

2.1.3. Multiple Buddy Strategy (MBS):

This strategy divides initially the 2D mesh into distinct square sub-meshes that are allocated as non-contiguous blocks, where the side length of these blocks is power of 2 upon initialization. MBS keeps up free block records (FBR) for all free processor squares of a similar size. The entry FBR$[i]$ includes the number of available squares of size $2^i \mathrm{x} 2^i$, and the job size of n processors is represented as a base-4 number of the form:

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

The allocation of the job request is determined according to the number of d_i blocks of size $2^i \mathrm{x} 2^i$ processors using FBR. If a required block is unavailable then the strategy searches for a larger block in FBR and breaks it down into four adjacent buddies until it reaches the proper size. Allocation in the MBS strategy succeeds when the number of free processors in the mesh is sufficient. MBS does not produce any internal or external fragmentation problems (Lo, Windisch, Liu, & Nitzberg, 1997; Bani-mohammad, Ould-KHaoua, Ababneh, & Mackenzie, 2006).

2.1.4. Greedy-Available Busy List Allocation Strategy (GABL):

In this strategy, the job request is divided based on the available sub-meshes for allocation in the mesh, and it tries contiguously to allocate the entire job request in a single available sub-mesh. If the allocation fails, GABL allocates the largest free sub-mesh that is available in the mesh and can fit inside the job request, then tries to allocate the next large sub-mesh whose side lengths do not exceed the side lengths of the previously allocated sub-mesh, and this step is repeated until the job request is allocated (Bani Mohammad, 2008; Bani-mohammad, Ould-KHaoua, & Ababneh, 2007).

Figure 2.3 shows an example of the GABL allocation strategy of a 6×6 2D mesh, where allocated processors are denoted by black circles and free processors are denoted by white circles. Assume a job requests for ten processors (5×2). GABL scans the mesh and tries to allocate this job request contiguously. It fails, and then it allocates the largest free sub-mesh that is available inside the mesh and can fit inside the job request. In this case a 2×3 available sub-mesh of processors has the coordinates (1,1,3,2) is allocated, where the first two coordinates identify the lower left corner of the sub-mesh and the last two coordinates identify the upper right corner of the sub-mesh, then it continues to allocate another sub-mesh (4,0,5,1).

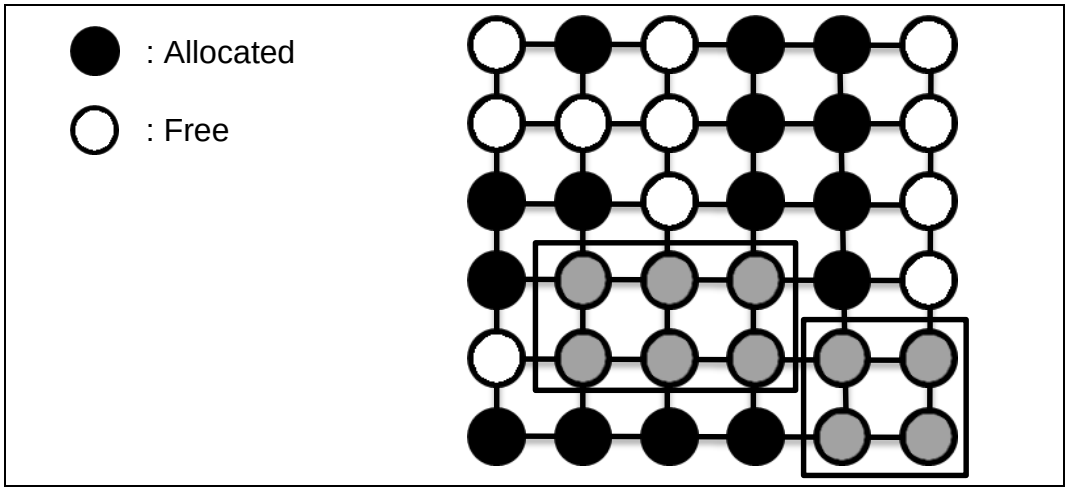


Figure 2- 3: Allocation in GABL Allocation Strategy

GABL decreases the message communication overhead by maintaining a high level of contiguity through dividing the job request into several large free sub-meshes that are available in the mesh system (Bani Mohammad, 2008; Bani-mohammad, Ould-KHaoua, & Ababneh, 2007).

2.2. System Model

The switching method is used in most multicomputer system to transfer a message from a source to destination through a series of intermediate nodes. The switching techniques has a large effect on the communication latency inside the direct network, and the most important switching techniques that have been used in multicomputer system are Store-and-forward (Kumar, Grama, Gupta, & Karypis, 2003), Virtual cut-through and Wormhole switching (Mohapatra, 1998; Kumar, Grama, Gupta, & Karypis, 2003; Ni & McKinley, 1993).

The switching method used in this thesis is the wormhole switching (moreover known as wormhole routing) to decide the way messages are handled as they travel through intermediate nodes, and it is used to offer low communication latency and reduce buffer requirements (Kumar, Grama, Gupta, & Karypis, 2003; Mohapatra, 1998; Ni & McKinley, 1993).

Wormhole routing has been utilized in almost all new generation of parallel computer systems which includes the iWARP (Peterson, Sutton, & Wiley, 1991), the MIT j-machine (Noakes, Wallach, & Dally, 1993), the intel Paragon (Intel Corporation, 1991) and the IBM BlueGene/L (Blue Gene Project, 2010). It has been a powerful switching technique where its communication latency is distance insensitive (Kumar, Grama, Gupta, & Karypis, 2003; Mohapatra, 1998; Ni & McKinley, 1993).

Wormhole routing is used in this thesis because it is widely used in realistic multicomputer and also it has been used when examining the performance of the existing non-contiguous allocation strategies. This is because of its low buffering requirement and appropriate overall performance (Lo, Windisch, Liu, & Nitzberg, 1997; Bani-mohammad, Ould-KHaoua, & Ababneh, 2007).

Mesh interconnection network is assumed in this thesis as the network topology. This is because of its good characteristics such as ease of implementation, simplicity, structural regularity and scalability. Mesh networks are easily implemented because of the simple regular connection and small number of links per node (Bani Mohammad, 2008).

Dimension-ordered routing in 2D mesh is performed via XY routing. The two dimensions of a mesh are labeled as X and Y. The source node sends a message first through X axis right or left then through Y axis up or down depending on the location of destination node. Dimension-ordered routing is used in this thesis because it has been used in existing non-contiguous allocation strategies (Lo, Windisch, Liu, & Nitzberg, 1997; Bani Mohammad, 2008).

In this thesis, we use four communication patterns in the simulation to assess and compare the overall performance of the non-contiguous allocation strategies. These communication patterns are random, near-neighbor, one-to-all and all-to-all (ProcSimity v4.3, 1996; Bani-Mohammad & Ababneh, 2013).

In random communication pattern, the source and destinations are selected randomly (ProcSimity v4.3, 1996).

In near-neighbor communication pattern, each processor allocated to a job sends a message to its neighbor processors, up, down, left and right (ProcSimity v4.3, 1996).

In one-to-all communication pattern, a randomly selected processor sends a message to all other processors allocated within the same job (ProcSimity v4.3, 1996).

In all-to-all communication pattern, every processor allocated to a job sends a message to every other processor allocated within the same job (Bani-Mohammad & Ababneh, 2013).

2.3. The Simulation Tool (ProcSimity Simulator)

ProcSimity is a simulation tool for research in processor allocation and job scheduling algorithms for the mesh-connected multicomputers, and it is an open-source code, which was written in the C programming language at the University of Oregon (ProcSimity v4.3, 1996).

ProcSimity provides a suitable environment for evaluating processor allocation and job scheduling algorithms for the mesh-connected multicomputers, and it is used to investigate various job scheduling and processor allocation strategies performance, such as fragmentation problem and communication overhead. ProcSimity architecture supports the mesh interconnection topology by consisting of a network of processors interconnected through message routers at each node. Neighbors' nodes are connected by two unidirectional channels, and it may be routed messages by either store-and-forward, virtual cut through or wormhole switching (Bani-mohammad, Ould-KHaoua, & Ababneh, 2007; Lo, Windisch, Liu, & Nitzberg, 1997; Mohapatra, 1998; ProcSimity v4.3, 1996; Windisch, Miller, & Lo, 1995).

In ProcSimity, job scheduling controls the selections of the next job for which processors are to be allocated. When the next independent user job arrives, it requests sub-mesh of free processors. If the mesh has sufficient processors for an incoming job, then the free processors are allocated to that job, if the number of free processors is not enough to satisfy the job request in the mesh, or there are other waiting job in the waiting queue, the incoming job is diverted to the waiting queue. The job is selected to be executed from the waiting queue based on the underlying scheduling strategy. When a job is ready to be executed, the job allocation algorithm assigns it to the available sub-meshes of processors in the mesh, which may be contiguous or non-contiguous, depending on the allocation strategy used. The execution job still holds the processors until it terminates its running, when it departs the system, the allocated processors are freed for use by another incoming job request (ProcSimity v4.3, 1996; Windisch, Miller, & Lo, 1995).

Each simulation run carries the values of the measured metrics that include system utilization, turnaround time, service time and finish time, and the very last simulation results are averaged over enough runs in order that the confidence level is 95% and relative errors do not exceed 5% (ProcSimity v4.3, 1996; Windisch, Miller, & Lo, 1995).

2.4. Justification of the method of study

There are two techniques exist for evaluating the system performance: analytical modeling and simulation. These two techniques are used because carrying out the measurements on a real practical system is costly or the real system may not available. In general, analytical models have often high requirements in terms of computation costs. Therefore, simulation has been selected as a tool of study in this research.

ProcSimity simulator has been widely used to evaluate the performance of processor allocation algorithms suggested for 2D mesh-connected multicomputers and also it has already been developed and extensively validated (ProcSimity v4.3, 1996).

Chapter 3
Horizontal Partitioning Strategy (HPS): A New Non-Contiguous Processor Allocation Strategy for 2D Mesh-Connected Multicomputers

3.1. Introduction

There are many processor allocation strategies that had been devised for mesh-connected multicomputers, these are classified into two types, contiguous and non-contiguous allocation strategies.

Contiguous strategies depend mainly on the processors that will be allocated to a job in 2D mesh and they must be physically adjacent, and form a contiguous shape according to the original topology. This causes external and internal fragmentation problems. The restriction that jobs have to be allocated to contiguous processors reduces the chance of successfully allocating a job, and it is viable for the allocation strategy to fail to allocate a job even as there are enough number of processors available in the mesh (Bani Mohammad, 2008).

In non-contiguous allocation strategies, a job can execute on multiple separated smaller sub-meshes rather than waiting until a single sub-mesh of the requested size and shape is available. The communication overhead may be alleviated in non-contiguous strategies by maintaining a good degree of contiguity among the allocated processors, however, dropping the contiguity condition can reduce processor fragmentation and increase system utilization, but it is able to cause high communication overhead (Ababneh & Almomani, 2012; Lo, Windisch, Liu, & Nitzberg, 1997; Bani-mohammad, Ould-KHaoua, & Ababneh, 2007).

Motivated by the above observations, in this chapter, we describe a new non-contiguous processor allocation strategy for 2D mesh-connected multicomputers, referred to as Horizontal Partitioning Strategy (HPS for short), the main aim of this strategy is to alleviate the message contention in the network, which is the primary purpose of any non-contagious allocation strategies.

3.2. The Horizontal Partitioning Allocation Strategy (HPS):

The 2D mesh-connected multicomputers is represented by N(W, H), where W is the width of mesh and H is its height.

The coordinates of any processor (node) in the mesh is denoted by an ordered pair (x, y), where $0 \leq x < W$ and $0 \leq y < H$.

Figure 3.1 shows and illustrates the initial state of the 2D mesh where it is empty when all its processors are unallocated, and the unallocated processors are denoted by white circles.

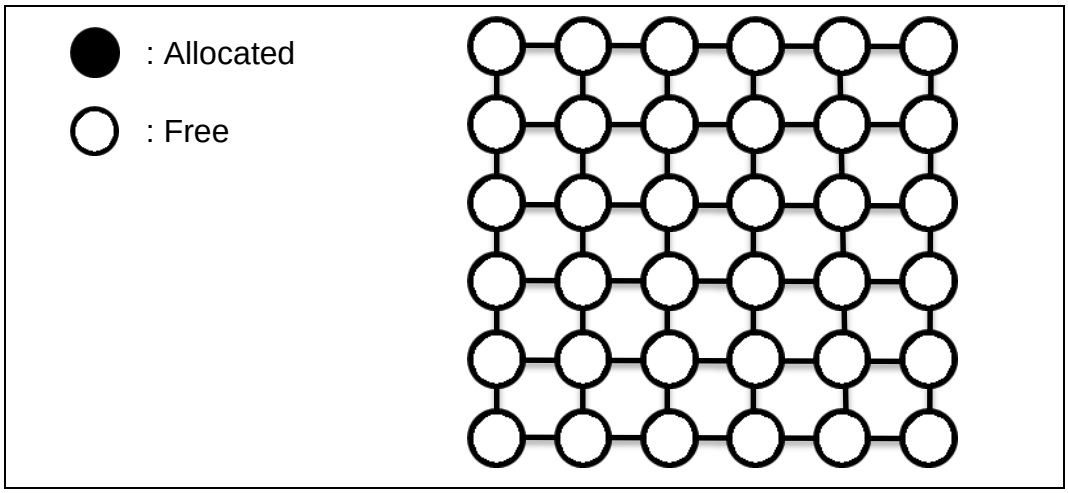


Figure 3-1: An empty 6×6 2D mesh

The incoming job request is represented by j(a, b), where the number of requested processors by an incoming job is a × b.

Each row in the mesh is represented by its y coordinate and it is represented as R(y).

The HPS allocation strategy partitions the job request based on the sub-meshes available for allocation in the system so as to maintain a high degree of contiguity. These sub-meshes are called Free-rows, and each of them represents a row of free processors that is equal to the width of the mesh.

Definition 1: A block represents any row of processors that is equal to the mesh width.

Definition 2: Free-row represents a row of free processors that is equal to the mesh width.

HPS strategy rebuilds the job request to accommodate in Free-rows and it always tries to allocate a job request contiguously in Free-rows in order to decrease the distance traversed by a message, and hence reduce message contention inside the network.

To describe the proposed strategy, we give some examples and figures were carefully selected to illustrate how the algorithm works. Initially, we assume that the mesh is empty as shown in Figure 3.2, where white circles represent the free processors, and we have a job that requests a sub-mesh of size 3×5.

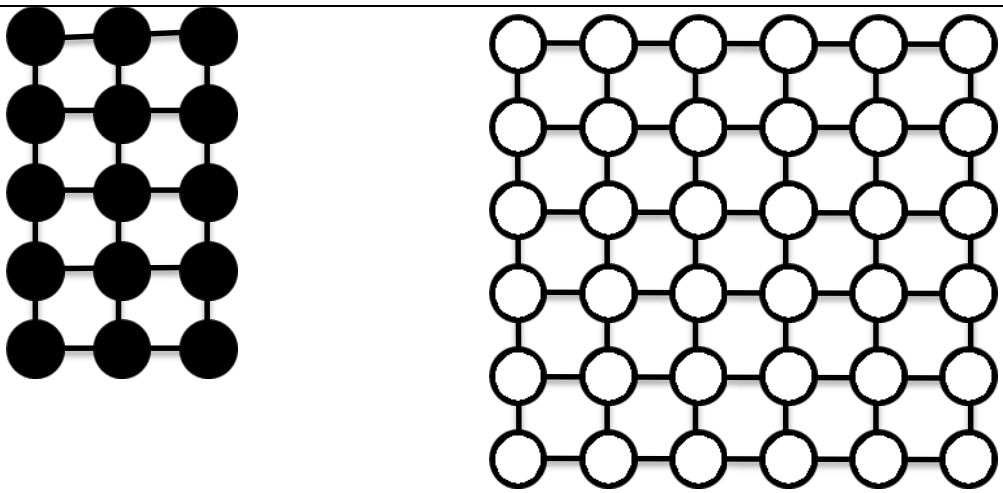


Figure 3- 2: Allocation of 3×5 sub-mesh in 6×6 2D mesh by HPS

The incoming job requests a sub-mesh of size 3×5 as shown in Figure 3-2, the job needs 15 processors. The proposed strategy (HPS) searches initially for the first Free-row that equals the mesh width and this exists in the mesh system as shown in Figure 3-3, so the proposed algorithm allocates the first six processors in the first Free-row, and then finds and allocates the second six processors in the second Free-row, and the last three processors in the request job is less than the mesh width, so the algorithm tries to find a block that have exactly 3 free processors in the mesh, but it fails, then it tries to find 3+1 free processors, and again it fails to find the requested processors. The process continues until it finds 3+3 free processors and allocation is done in the third row, as shown in Figure 3-3.

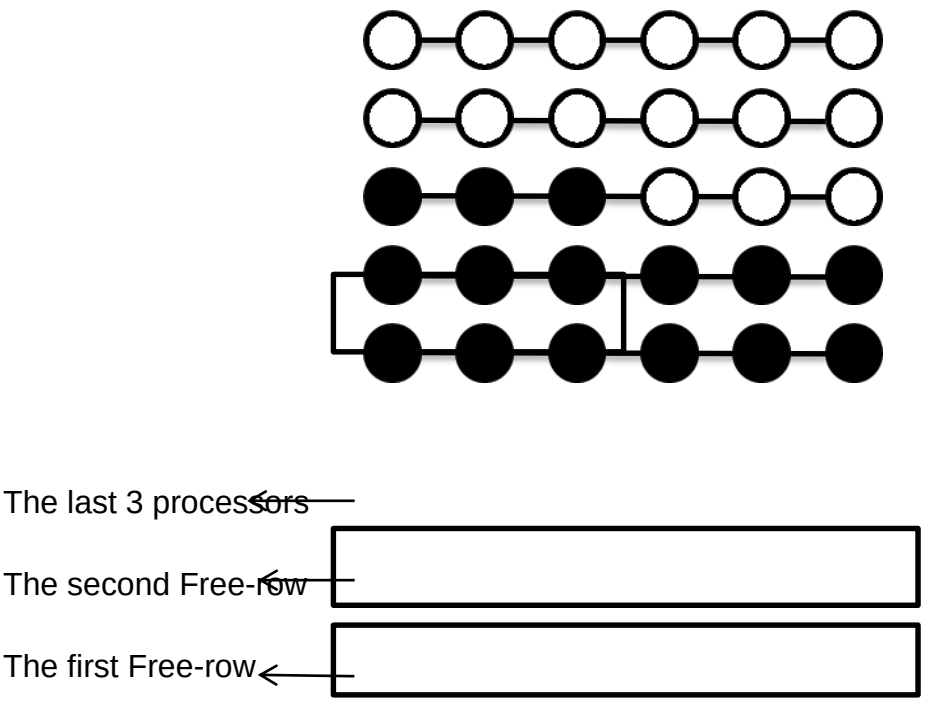


Figure 3- 3: Allocation of 3×5 sub-mesh in 6×6 2D mesh

Figure 3.4 shows the second example of allocation for a job request of size 3×3, where the job requests a 3×3 sub-mesh as shown in figure 3-4. In this example, the HPS strategy searches for the first Free-row that equals the mesh width.

It finds the requested processors and allocates the first six processors in the fourth Free-row as shown in figure 3-5. The last three processors in the job request is less than the mesh width, so the algorithm tries to find a block that has exactly three free processors in the mesh, and it finds these three free processors in the third row as shown in figure 3-5.

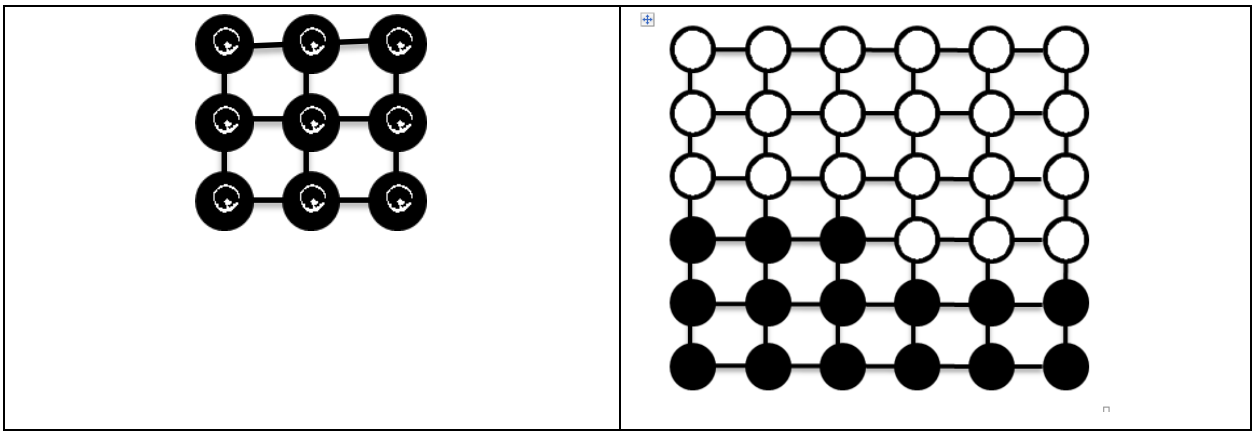


Figure 3- 4: Allocation of 3×3 sub-mesh in 6×6 2D mesh by HPS

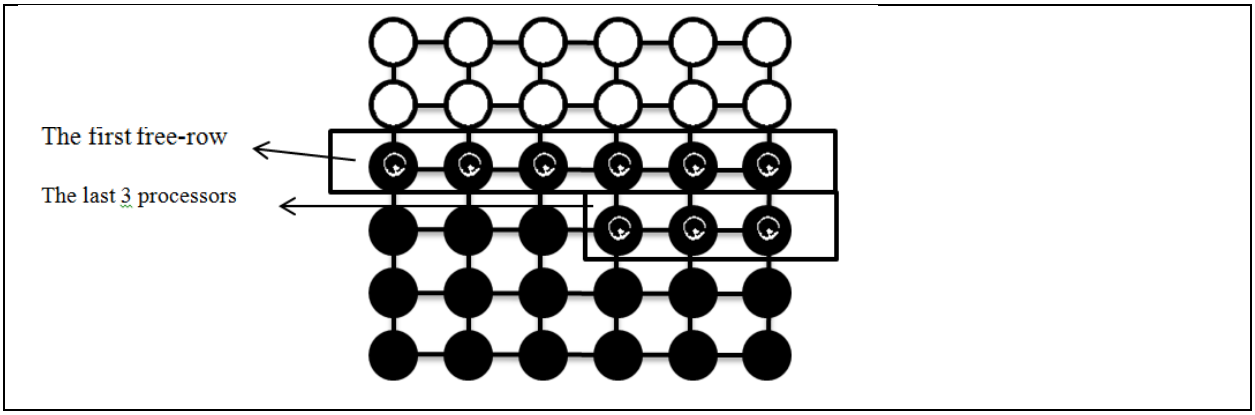


Figure 3- 5: Allocating 3×3 sub-mesh in 6×6 2D mesh

Figure 3.6 shows a 6×6 2D mesh after the first job has been completed and another job requests a sub-mesh of size 4×3.

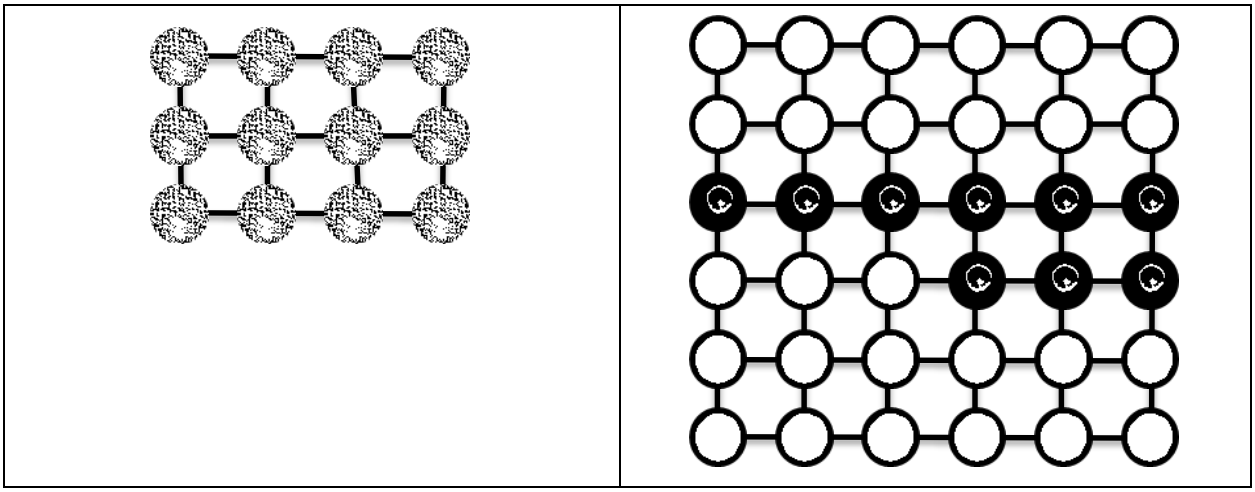


Figure 3- 6: A job requests 4×3 sub-mesh and the first job is completed

The third job requests a 4×3 sub-mesh as shown in Figure 3-6, where 12 processors are needed to be allocated for this request. Here, the algorithm searches for the first Free-row that equals the mesh width, then it finds the requested block and allocates the first 6 processors in the first Free-row, then it finds and allocates the second 6 processors in the next Free-row, as shown in Figure 3-7.

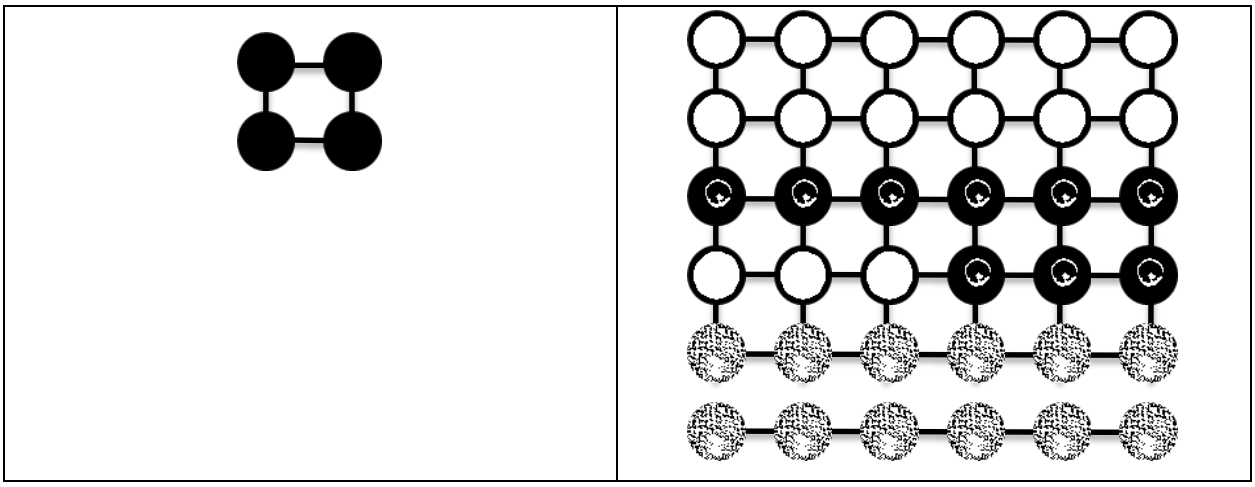


Figure 3- 7: A job request 2×2 sub-mesh and allocating previous 4×3 sub-mesh

In the last example, the fourth job requests a 2×2 sub-mesh as shown in Figure 3-7, so the job needs four processors, and these four processors is less than the mesh width. HPS strategy tries to find a block that has exactly four free processors in the mesh, but it fails to find it. Then it tries to find 4+1 free processors, and again it fails to find the requested processors, the process continues until it finds 4+2 free processors and allocation is done in the fifth row, as shown in Figure 3-8.

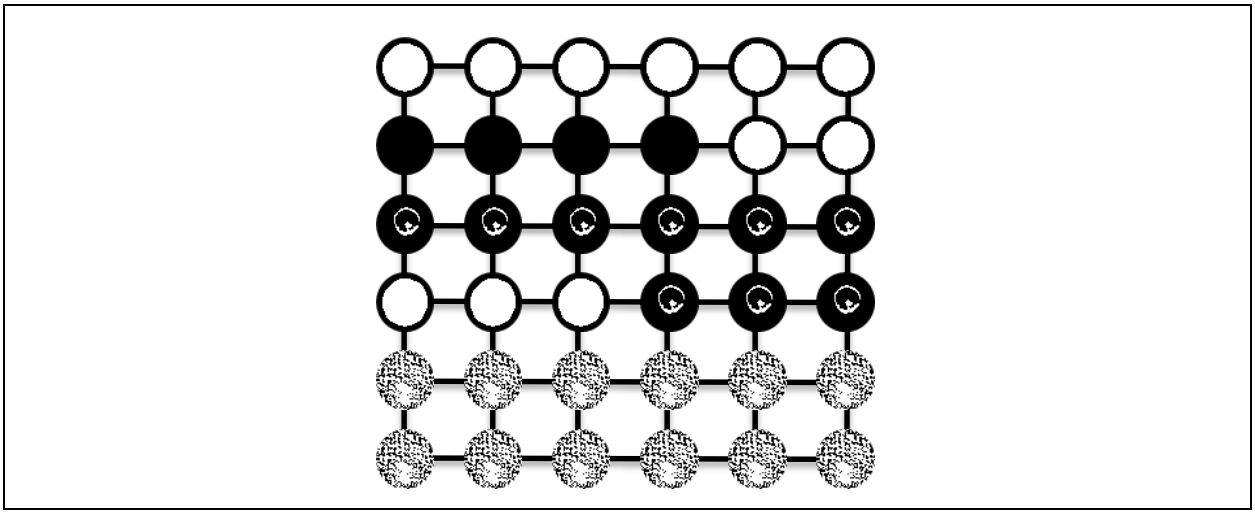


Figure 3- 8: Allocating 2×2 sub-mesh in 6×6 2D mesh

Figure 3-9 outlines the HPS allocation algorithm that has been clarified in the previous examples, where the time complexity of the proposed algorithm in the worst case is O($W^3 \times H^3$).

```
Procedure HPS_Allocate(a,b):
{
W: Width of the mesh.
H: Height of the mesh.
job_size = a×b.
Allocated_Processors = 0.
Step 1.                 if (number of free processor < job_size)
        return failure.
Step 2.         if (job_size = Allocated_Processors)
        return success.
Step 3.         if ((job_size  - Allocated_Processors) >= W){
                 search the mesh rows from first row R(0) to R(H-1) for a Free-row of
processors.
        if (Free-row is found){
        allocate W number of processors from the requested job in these Free-row.
        Allocated_Processors+=W.
        add allocated processors to APL.          //APL: Allocated Processors List: is
        go to step 2.}}                           an array of linked list that contains
the
                                                  coordinates of the allocated
processors
                                                  and its job id.

Step 4 .          X= job_size - Allocated_Processors.
search the mesh rows from first row R(0) to R(H-1) for a row that has exactly X
number of free contiguous processors.
if (these free processors are found){
```

```
        allocate X number of processors from the requested job in these processors.
        Allocated_Processors+ = X.
        add allocated processors to APL.
        go to step 2.}
Step 5 .        Y= 1.
Step 6 .        X= job_size - Allocated_Processors.
search the mesh rows from first row R(0) to R(H-1) for a row that has exactly (X+Y)
number of free contiguous processors.
if (these free processors are found){
        allocate X number of processors from the requested job in these processors.
        Allocated_Processors += X.
        add allocated processors to APL.
        go to step 2.}
        else {
        Y= Y+1.
        If(X+Y=W)
        go to step 7.
        else
        go to 6.}

Step 7.             allocate the requested number of processors in row-major
starting from R(0).
        add allocated processors to APL.
} end procedure
```

Figure 3- 9: Outline of the HPS allocation algorithm

Figure 3-10 outlines the HPS deallocation algorithm.

```
Procedure HPS_Deallocate(a,b):
{
job_id = id of the departing job.
for all nodes in APL[W*W]                                    //APL: Allocated Processors
List: is
        if (node_id =job_id)                                 an array of linked list that contains
the
                remove Node.                                 coordinates of the allocated
processors
}end procedue                          and its job id.
```

Figure 3- 10: Outline of the HPS deallocation algorithm

3.3. HPS Time Complexity:

HPS strategy maintains an array of linked list, Allocated Processors List (APL) that contains the coordinates of the allocated processors in the mesh. Where the size of APL array equals N or (W×H), where W, H and N are the width, the height and the size of the mesh, respectively. So, the time complexity for scanning it is O(N).

The time complexity for searching a contiguous free processors in any row in the mesh is O(W), where W is the width of mesh.

The HPS allocation operation for a job requests k processors is O(k×(W+H×N)), where H represents the number of rows and it equals the height of the mesh.

Chapter 4
Simulation Results

In this chapter, the simulation experiments for the proposed HPS allocation strategy as well as the existing well-known non-contiguous allocation strategies (Paging(0), MBS, and GABL) have been conducted. The performance of HPS has been compared with that of the existing allocation strategies Paging(0) (Lo, Windisch, Liu, & Nitzberg, 1997), MBS (Lo, Windisch, Liu, & Nitzberg, 1997), and GABL (Bani-mohammad, Ould-KHaoua, & Ababneh, 2007).

HPS allocation and de-allocation algorithms were implemented in C programming language, and integrated into the ProcSimity simulation tool that is widely used for processor allocation and job scheduling in parallel systems (ProcSimity v4.3, 1996; Windisch, Miller, & Lo, 1995).

The mesh system assumed in the simulation experiments is a 2D square mesh with side length L. Jobs inter-arrival time has been exponentially distributed. The job scheduling scheme is First Come First Served (FCFS). FCFS scheduling has been used because our main purpose is to compare and evaluate the performance of the allocation strategy and because FCFS is fair. The job execution time is the time needed by each job for completion. The execution time of jobs rely upon the time needed for flits to be routed through the nodes, packet size, the number of messages to be sent, message contention in the network and the distances that the messages traverse (Bani Mohammad, 2008). Two distributions are used to generate the side lengths of the requested sub-meshes. The first is a uniform distribution over the range from one to the mesh side length L, where the width and length of a job request are generated independently.

The second is a uniform-decreasing distribution that is based on four probabilities p1, p2, p3 and p4, and four integers l1, l2, l3 and l4, where the probabilities that the width/height of a request falls in the ranges [1, l1], [l1 + 1, l2], [l2 + 1, l3] and [l3 + 1, l4] are p1, p2, p3 and p4, respectively. The side lengths within a range are equally likely to occur. The uniform-decreasing distribution represents the case where most jobs are small relative to the size of the system, so the simulation experiments in this research are for p1 = 0.4, p2 = 0.2, p3 = 0.2, p4 = 0.2, l1 = L/8, l2 = L/4, l3 = L/2, l4 = L (Lo, Windisch, Liu, & Nitzberg, 1997; Chang & Mohapatra, 1998; Zhu, 1992; Alsardia, 2017).

The interconnection network for message routing is wormhole routing with ordered XY routing, where the number of bytes in each message (message size) is eight. Each simulation run consists of 1000 completed jobs per run and the number of runs is varied to get a confidence level of 95% and relative errors do not exceed 5% (Mohapatra, 1998; Ni & McKinley, 1993).

A job remains in the system until an iteration of the communication pattern is completed. Processors allocated to a parallel job communicate with each other according to four communication patterns that are considered in this research. These communication patterns are random, near-neighbor, one-to-all and all-to-all communication patterns (ProcSimity v4.3, 1996; Bani-Mohammad & Ababneh, 2013). Table 4.1 below presents the parameters that have been used in the simulator.

Table 4.1: the system parameters used in the simulation experiments.

Simulation Parameters	Value
Dimensions of the Mesh	16×16
Allocation Strategy	Paging(0), MBS, GABL, HPS
Scheduling Strategy	FCFS
Job Size Distribution	Uniform: Job widths and lengths are uniformly distributed over the range from 1 to the mesh side lengths.
	Uniform Decreasing: represents the case where the most jobs are small relative to the size of the system.
Inter-arrival Time	Exponential with different values for the mean. The values are determined through experimentation with the simulator, ranged from lower values to higher values.
Number of Runs	The number of runs should be enough so that the confidence level is 95% that relative errors are below 5% of the means.
Number of Jobs per Run	1000

The main performance parameters used are average turnaround time of jobs, and mean system utilization. The turnaround time of a job is the time that the job spends in the system from arrival to departure (Bani-mohammad, Ould-KHaoua, Ababneh, & Mackenzie, 2006). The system utilization is the percentage of processors that are utilized over a given period of time (ProcSimity v4.3, 1996). The independent variable in the simulation is the system load that is defined as the inverse of the mean inter-arrival time of jobs (Bani Mohammad, 2008).

4.1. Turnaround Time

In Figures 4.1 and 4.2, the average turnaround time of jobs are plotted against the system load for the one-to-all communication pattern. The results show that the performance of HPS allocation strategy is close to that of the other non-contiguous allocation strategies for both job size distributions considered in this research. This is because the number of messages generated by one-to-all is small and hence the communication overhead caused by this communication is not high and the ability of the non-contiguous allocation strategies considered in this thesis including our algorithm to alleviate the contention is approximately the same, which results very close in performance in terms of job turnaround time.

In Figure 4.1, for example, the average turnaround time of HPS is almost the same as Paging(0), GABL and MBS, under the job arrival rate of 0.0009 jobs/time units. Note that the difference in performance between these algorithms is less than 5%, and this is the percentage of error in the simulation experiments, so this difference in performance can be ignored in such a case.

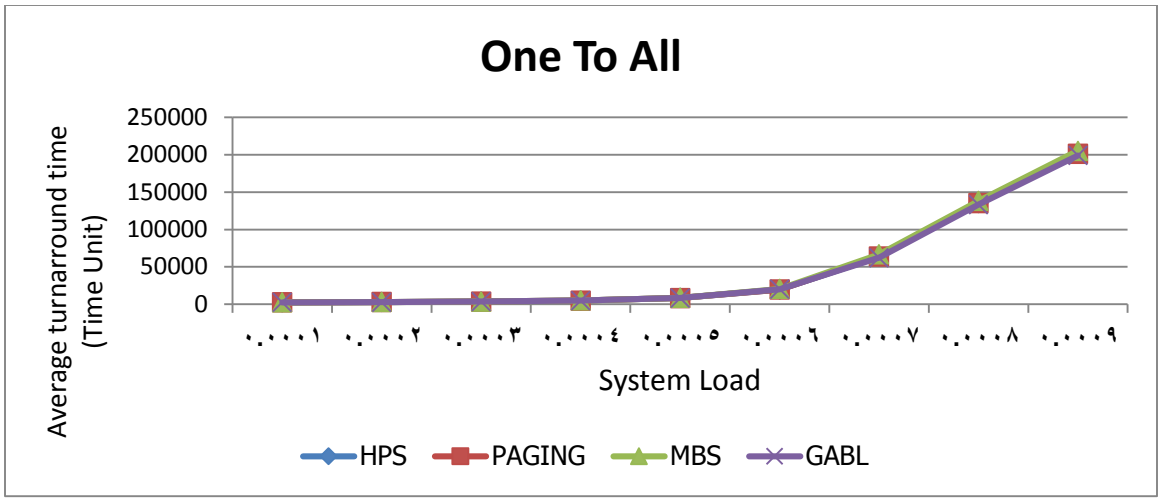


Figure 4.1: Average turnaround time vs. system load for the one-to-all communication pattern and uniform job side lengths distribution in a 16x16 mesh.

In Figure 4.2, the average turnaround time of the non-contiguous allocation strategies is improved when the uniform-decreasing distribution is used, while the relative performance for all the allocation strategies remains the same. This improvement in performance is due to the increasing of the probability of generating small jobs relative to the size of the mesh system and hence the allocation for most of these jobs succeeds.

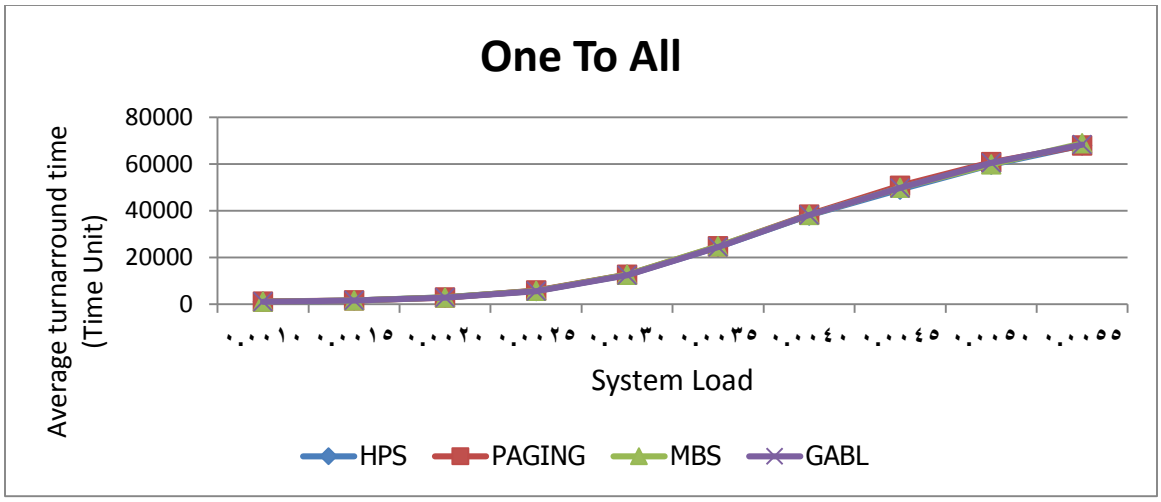


Figure 4.2: Average turnaround time vs. system load for the one-to-all communication pattern and uniform decreasing job side lengths distribution in a 16x16 mesh.

In Figures 4.3 and 4.4, the average turnaround time of jobs is plotted against the system load for the all-to-all communication pattern. The results show that the performance of the HPS allocation strategy is much better than that of the other non-contiguous allocation strategies for both job size distributions considered in this research. This is because HPS is better than the previous non-contiguous allocation strategies in alleviating message contention when the communication overhead is high as in all-to-all communication. This improvement in performance is achieved by maintaining some degree of contiguity among allocated processors by allocating the job request in Free-rows each of them represents a row of free processors that is equal to the mesh width.

In Figure 4.3, for example, the average turnaround time of HPS is 79%, 69% and 40% of that of GABL, Paging(0) and MBS, respectively, under the job arrival rate of 0.0001 jobs/time units.

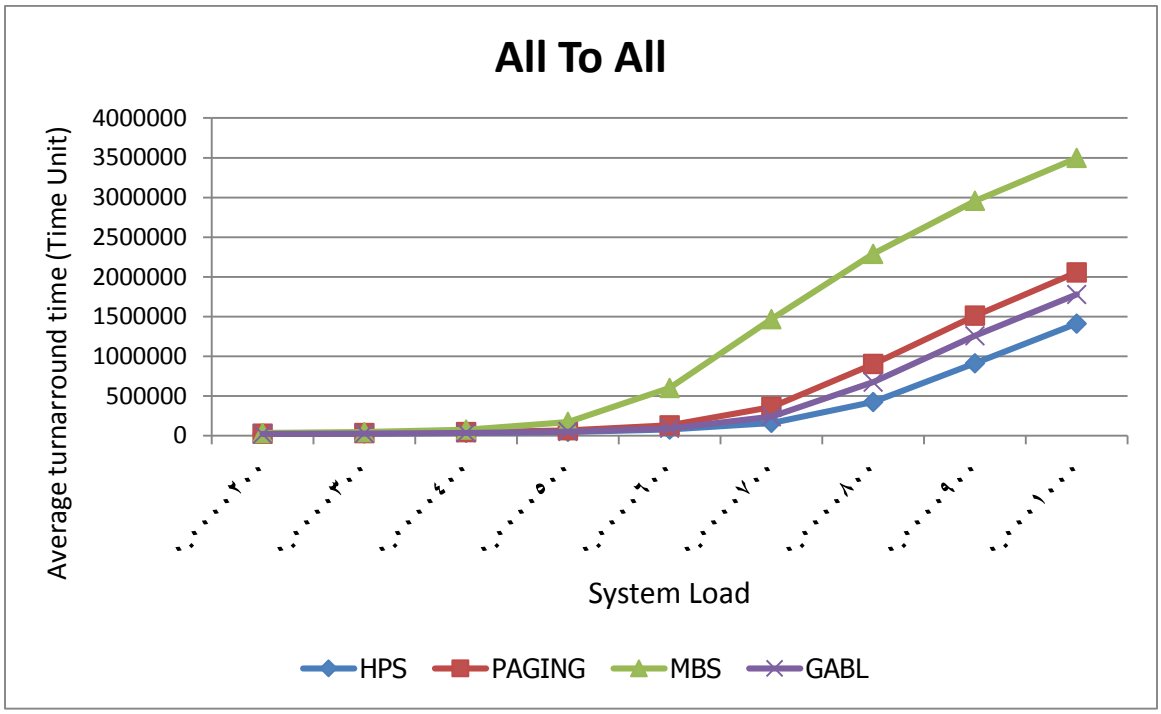


Figure 4.3: Average turnaround time vs. system load for the all-to-all communication pattern and uniform job side lengths distribution in a 16x16 mesh.

In Figure 4.4, the average turnaround times of all the non-contiguous allocation strategies are improved again, this is because the increased probability of small jobs to be allocated (relative to mesh size). When uniform decreasing distribution is used, the average turnaround time of HPS is 72%, 79% and 56% of that of GABL, Paging(0) and MBS, respectively, under the job arrival rate of 0.00055 jobs/time units. In general, HPS allocates the job request along the rows of the mesh, and relatively the small jobs can be allocated in a less number of rows, which decreases the message contention

between different jobs. Furthermore, HPS has the ability to allocate jobs with number of processors that is smaller than or equal to the mesh width in a way that results in less communication overhead and thus reduces the contention among different small jobs.

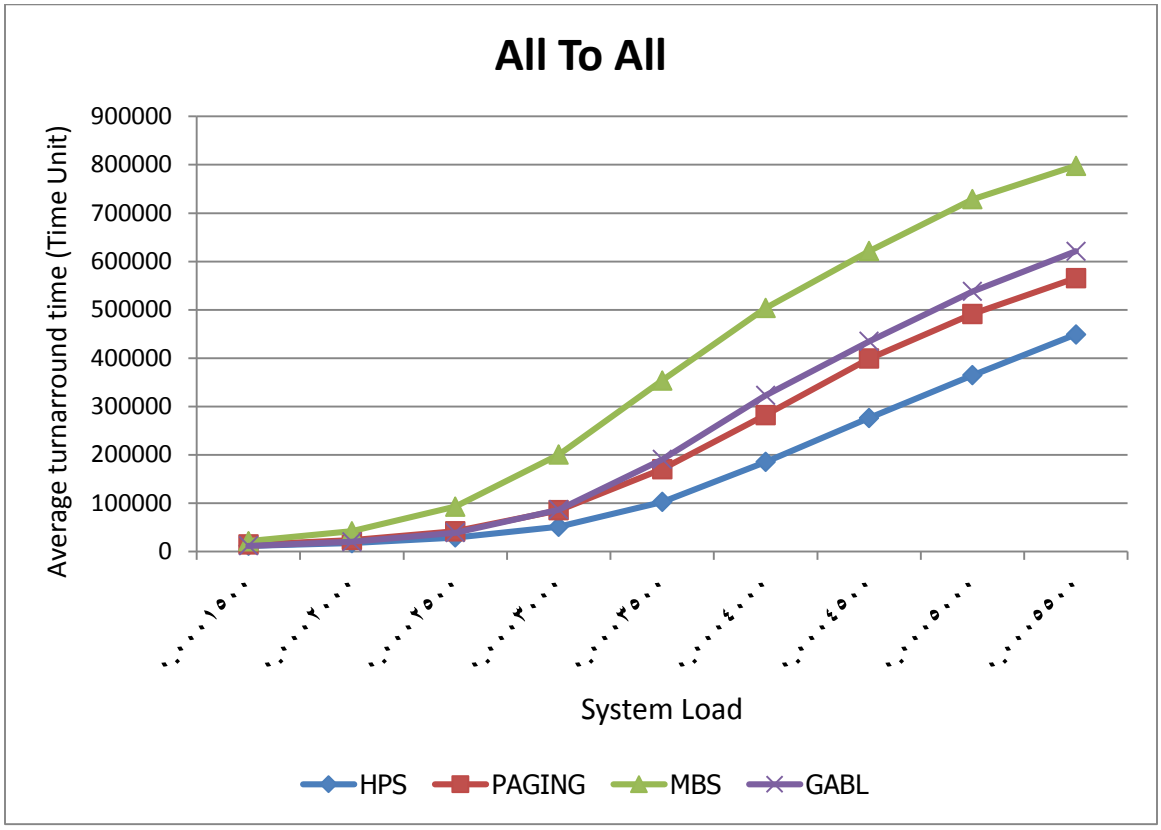


Figure 4.4: Average turnaround time vs. system load for the all-to-all communication pattern and uniform decreasing job side lengths distribution in a 16x16 mesh.

In Figures 4.5 and 4.6, the average turnaround time of jobs is plotted against the system load for the random communication pattern. The results show that the performance of HPS allocation strategy is close to that of the other non-contiguous allocation strategies for both job size distributions considered in this research.

In Figure 4.5, for example, the average turnaround time of HPS is almost the same as Paging(0) and MBS, under the job arrival rate of 0.06 jobs/time units, and the relative difference in turnaround times between HPS and GABL is 9% in favor for GABL. Note that the difference between the performances of these algorithms is less than 5%, and this is the percentage of error allowed in the simulation experiments and it can be ignored.

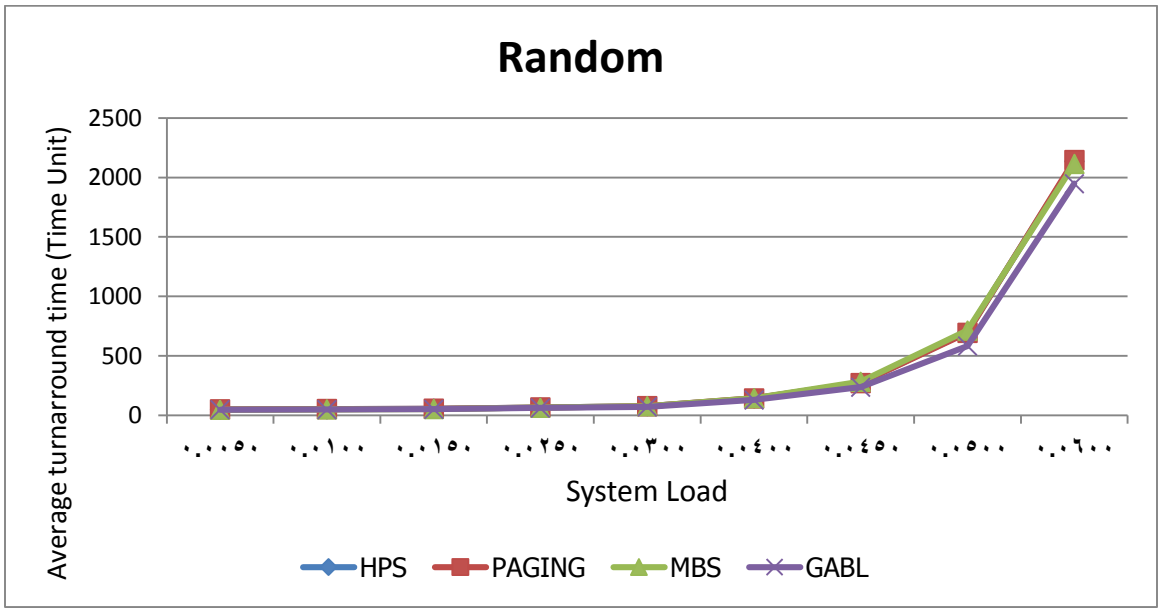


Figure 4.5: Average turnaround time vs. system load for the random communication pattern and uniform job side lengths distribution in a 16x16 mesh.

In Figure 4.6, the average turnaround time of the non-contiguous allocation strategies is improved again, when the uniform decreasing distribution is used. This improvement in performance is due to the increasing of the probability of generating small jobs relative to the size of the mesh system and hence the allocation for most of these jobs succeeds.

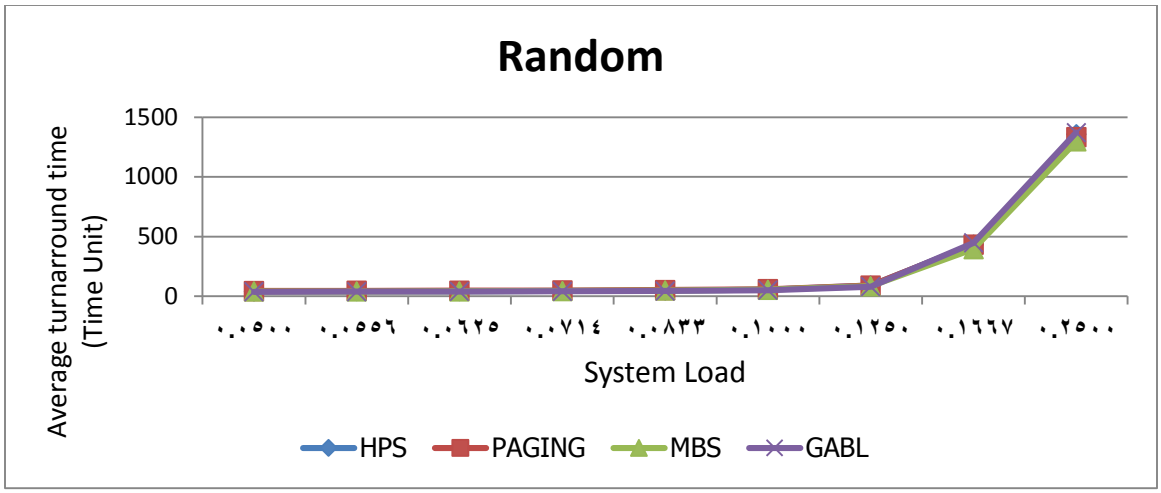


Figure 4.6: Average turnaround time vs. system load for the random communication pattern and uniform decreasing job side lengths distribution in a 16x16 mesh.

In Figures 4.7 and 4.8, the average turnaround time of jobs is plotted against the system load for the near-neighbor communication pattern. The results show that the performance of the HPS allocation strategy is not better than that of the other non-contiguous allocation strategies considered in this thesis. In Figure 4.7, for example, the average turnaround time of HPS is very close to that of the Paging(0) and MBS when the job arrival rate is 0.009 jobs/time units. In Figure 4.8, the performance of HPS is very close to the that of Paging(0). This is because the near-neighbor communication pattern is suitable for the strategies that keep a high degree of contiguity between the allocated processors and at the same time maintain a rectangular form of the allocated sub-meshes, where each node allocated to a job communicates with its left, right, up and down neighbors within the same job. GABL performs better than other non-contiguous allocation strategies under both job size distributions considered, and this is because GABL allocates sub-meshes in a rectangular form and it tries to maintain a high degree of contiguity among the allocated processors.

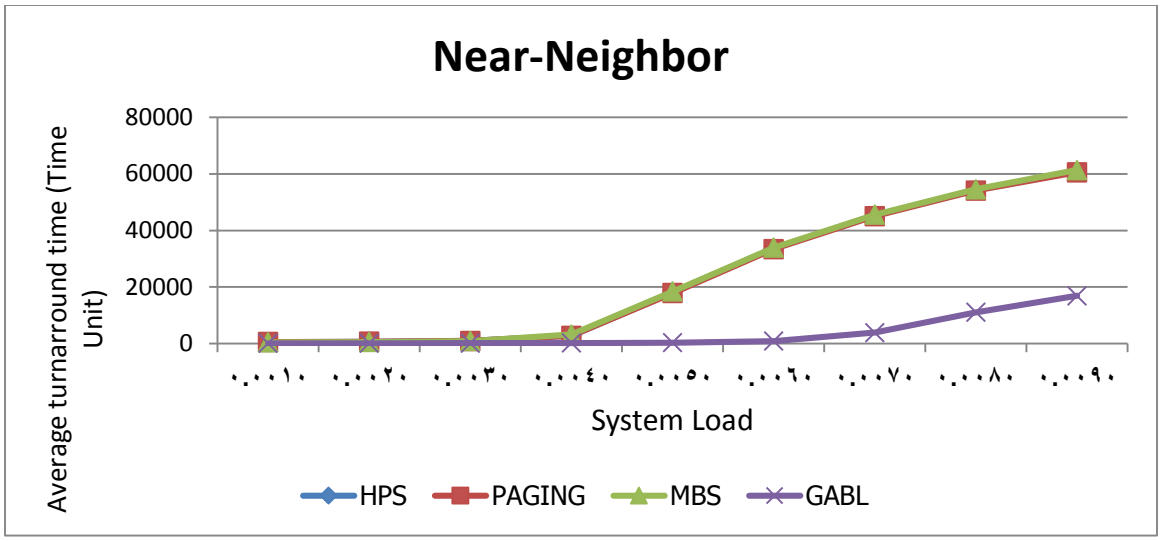


Figure 4.7: Average turnaround time vs. system load for the near-neighbor communication pattern and uniform job side lengths distribution in a 16x16 mesh.

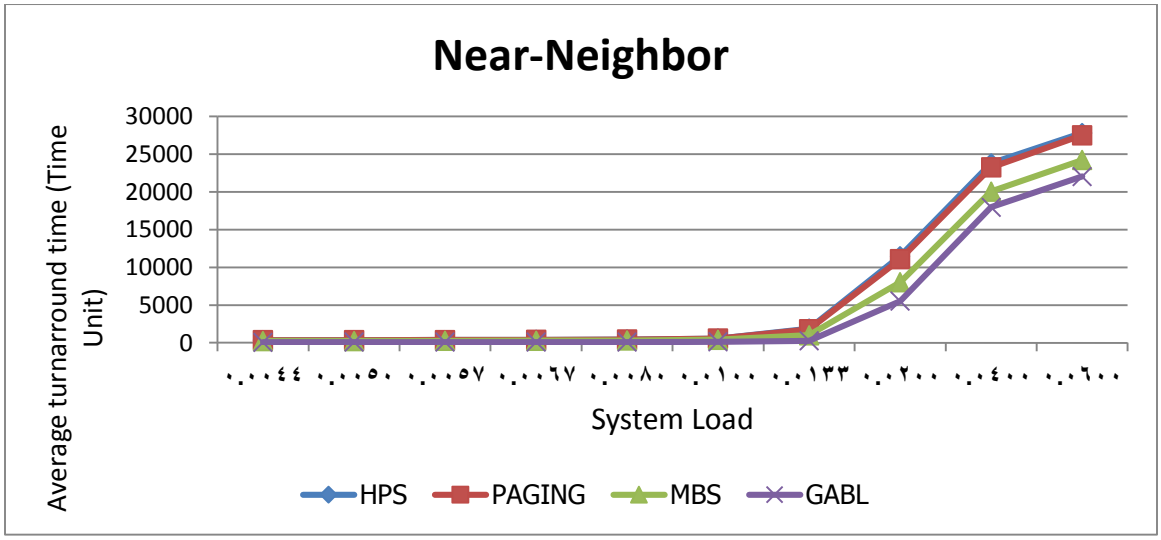


Figure 4.8: Average turnaround time vs. system load for the near-neighbor communication pattern and uniform decreasing job side lengths distribution in a 16x16 mesh.

4.2. System Utilization

In Figures (4.9-4.16), the mean system utilization is plotted against the system load for the four communication patterns, one-to-all, all-to-all, random and near-neighbor using the FCFS scheduling strategy, and the two job size distributions considered in this research. The results show that the mean system utilization for all non-contiguous allocation strategies is approximately the same for both job size distributions, at heavy system load values. The load values ranged from moderate to heavy system loads, where heavy loads cause the waiting queue to be filled very early that allow the allocation strategies to achieve a higher system utilization that ranges from 75% to 78% and from 81% to 85%, for uniform and uniform-decreasing job size distributions, respectively. This is because the non-contiguous allocation strategies considered in this thesis have the same ability to eliminate both internal and external processor fragmentation. They always succeed to allocate processors to a job when the number of free processors is greater than or equal to the allocation request.

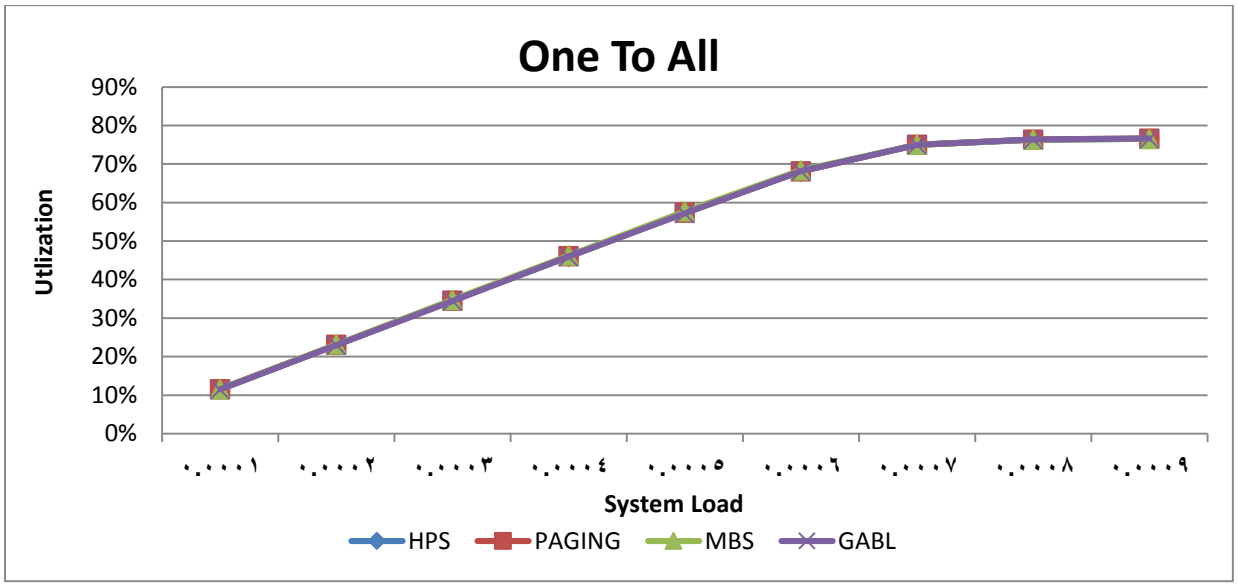


Figure 4.9: Mean system utilization vs. system load for the one-to-all communication pattern and uniform job side lengths distribution in a 16x16 mesh.

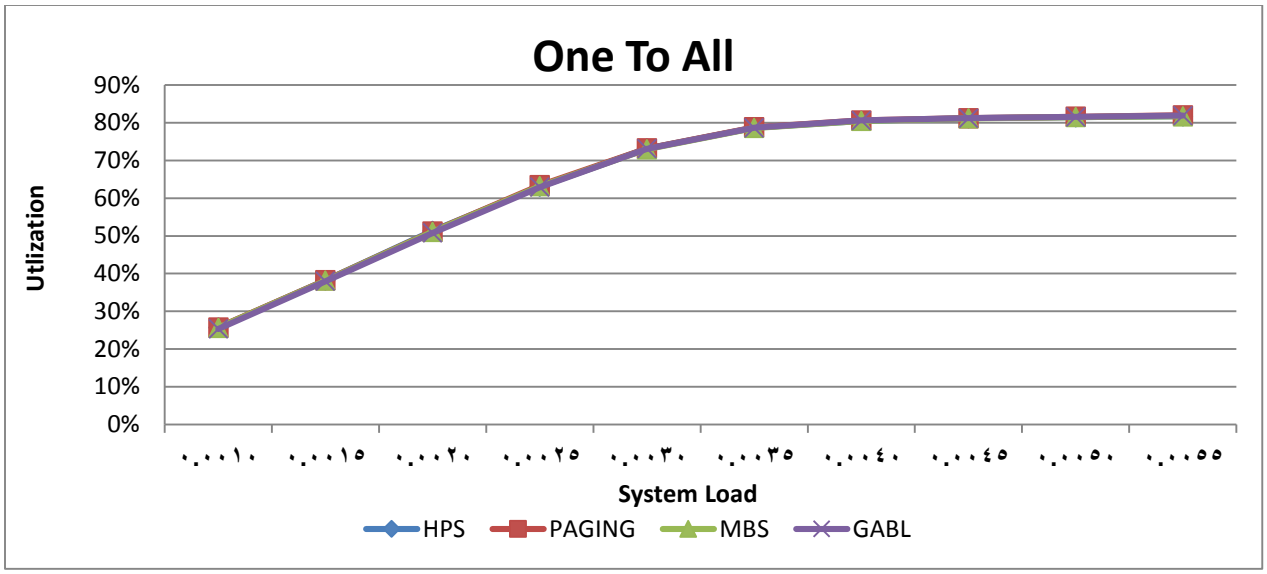


Figure 4.10: Mean system utilization vs. system load for the one-to-all communication pattern and uniform decreasing job side lengths distribution in a 16x16 mesh.

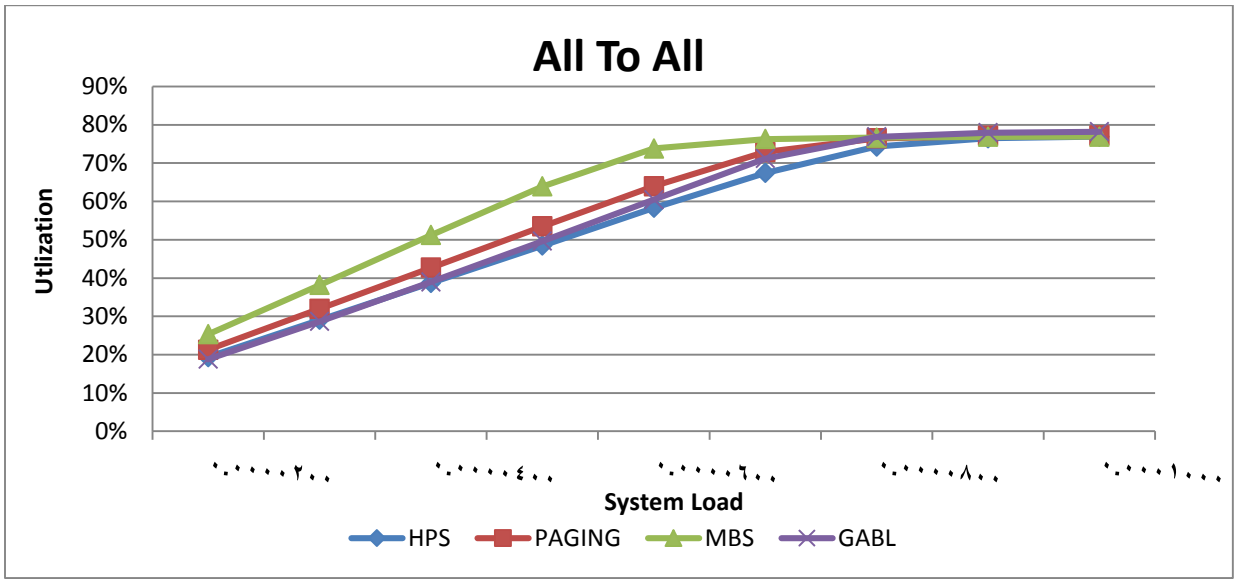


Figure 4.11: Mean system utilization vs. system load for the all-to-all communication pattern and uniform job side lengths distribution in a 16x16 mesh.

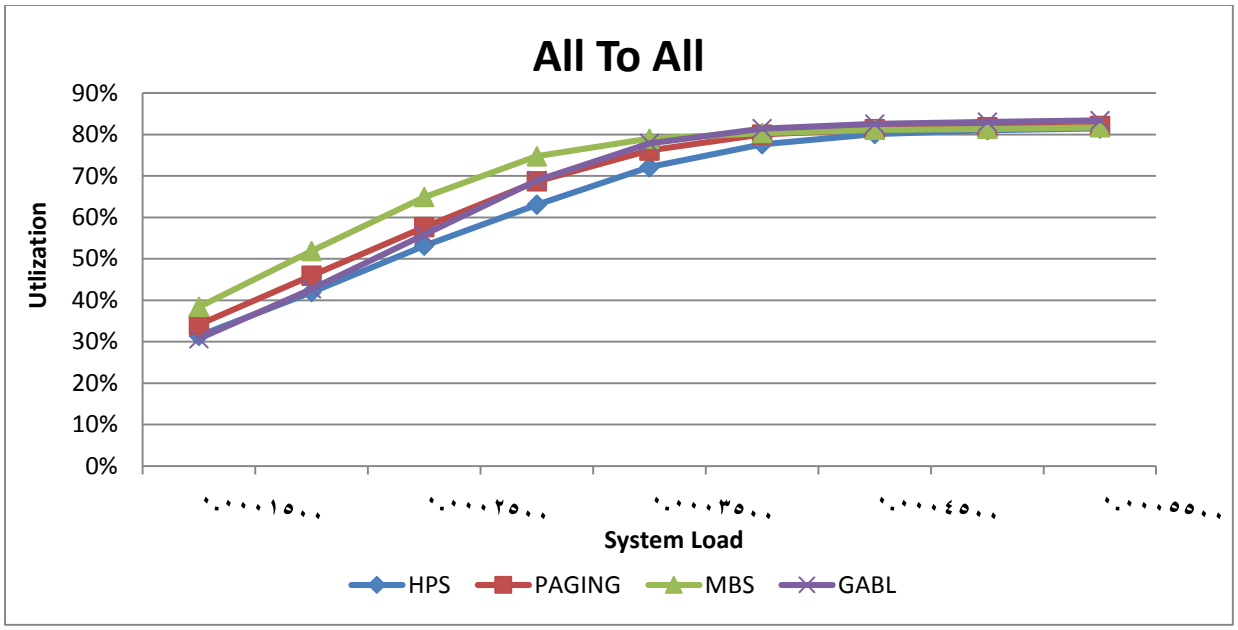


Figure 4.12: Mean system utilization vs. system load for the all-to-all communication pattern and uniform decreasing job side lengths distribution in a 16x16 mesh.

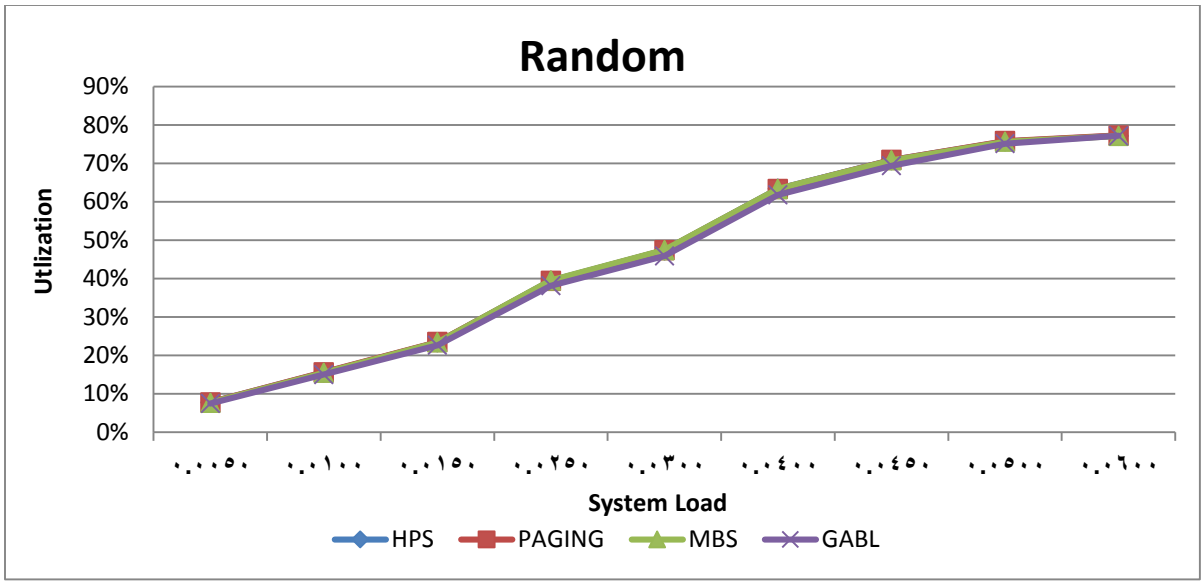


Figure 4.13: Mean system utilization vs. system load for the random communication pattern and uniform job side lengths distribution in a 16x16 mesh.

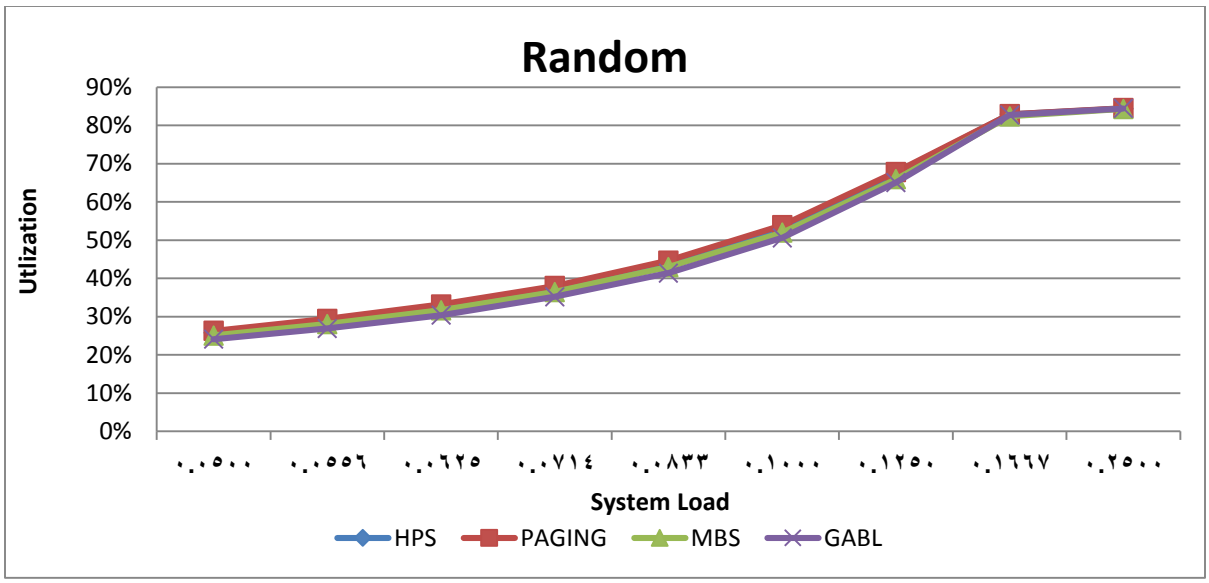


Figure 4.14: Mean system utilization vs. system load for the random communication pattern and uniform decreasing job side lengths distribution in a 16x16 mesh.

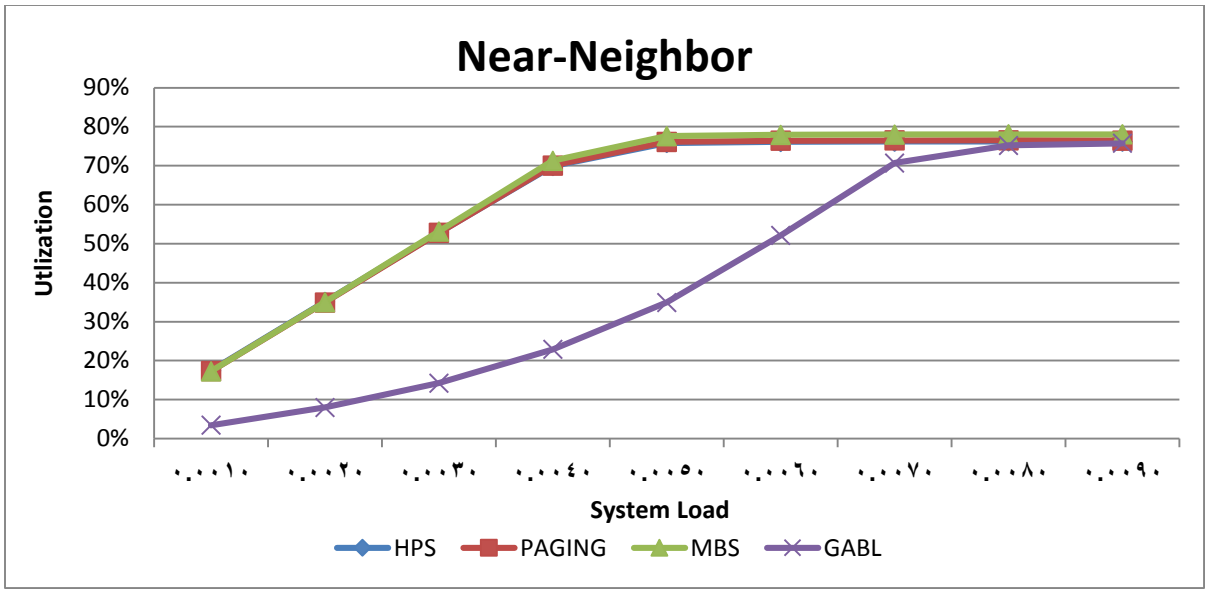


Figure 4.15: Mean system utilization vs. system load for the near-neighbor communication pattern and uniform job side lengths distribution in a 16x16 mesh.

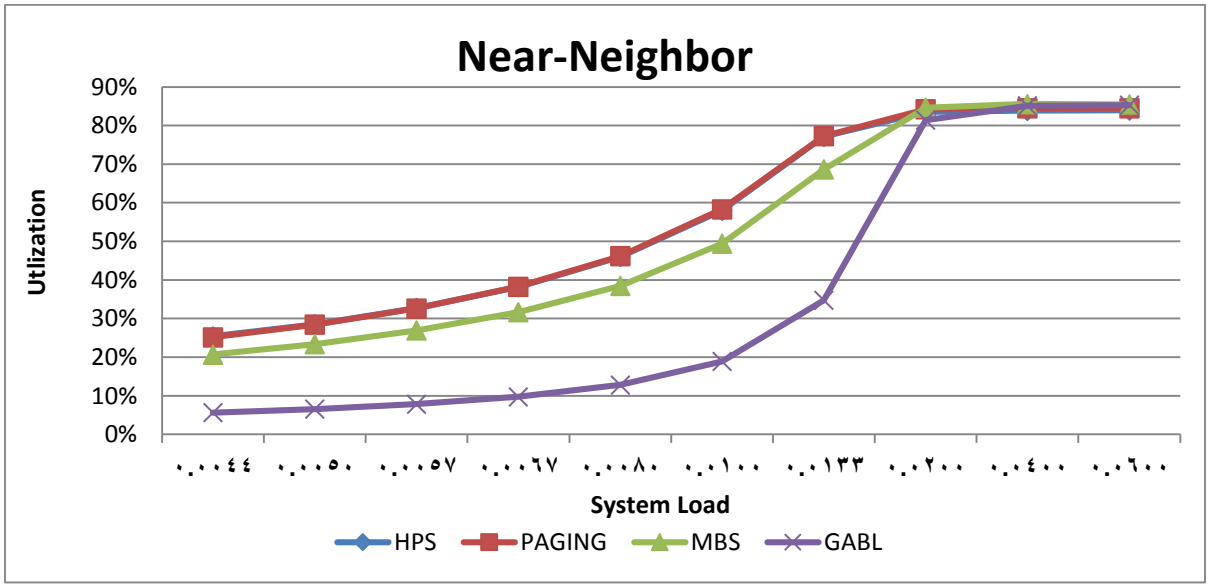


Figure 4.16: Mean system utilization vs. system load for the near-neighbor communication pattern and uniform decreasing job side lengths distribution in a 16x16 mesh.

Chapter 5
Conclusion and Future Work

5.1. Conclusion

In recent years, processor allocation in distributed-memory multicomputers became the subject of much research especially that is based on the mesh network. Processor allocation strategies that had been devised for mesh-connected multicomputers are classified into two types, contiguous and non-contiguous allocation strategies.

In contiguous strategies, the processors that will be allocated to a job must be physically adjacent, and form a contiguous shape according to the original topology. Contiguous allocation suffers from both external and internal fragmentation problems. External fragmentation occurs when there are free processors enough in quantity to satisfy the job request, but they are not allocated to it because they are not contiguous. Internal fragmentation occurs if the strategy allocates more processors than required (Lo, Windisch, Liu, & Nitzberg, 1997; Bani Mohammad, 2008).

In non-contiguous allocation strategies, the job request can execute on multiple separated smaller sub-meshes rather than waiting until a single sub-mesh of the requested size and shape is available. The main objective of these strategies is to maintain minimal communication overhead without affecting the overall system performance and this is achieved by maintaining a good degree of contiguity among the allocated processors, however, dropping the contiguity condition can reduce processor fragmentation and increase system utilization, but it causes high communication overhead (Bani Mohammad, 2008; Bani-mohammad, Ould-KHaoua, & Ababneh, 2007). In general, the aim of any allocation strategy is to minimize the average turnaround time and maximizing the system utilization (Lo, Windisch, Liu, & Nitzberg, 1997; Mohapatra, 1998; Bani Mohammad, 2008).

Motivated by the above observations, a new non-contiguous processor allocation strategy for 2D mesh-connected multicomputers, referred to as Horizontal Partitioning Strategy (HPS for short) has been proposed. The main aim of this strategy is to alleviate the message contention in the network, and thus improves system performance in terms of average turnaround time of jobs, and this is the main purpose of any non-contagious allocation strategy.

The HPS allocation strategy partitions the job request based on the rows available for allocation in the system so as to maintain some degree of contiguity. These sub-meshes are called Free-rows, and each of them represents a row of free processors that is equal to the width of the mesh. HPS strategy rebuilds the job request to be accommodated in the available Free-rows and it always tries to allocate a job request contiguously in Free-rows in order to decrease the distance traversed by a message, and hence reduce message contention inside the network.

The simulation results of the proposed HPS allocation strategy have been carried out, and compared with those of the existing well-known non-contiguous allocation strategies Paging(0) (Lo, Windisch, Liu, & Nitzberg, 1997), MBS (Lo, Windisch, Liu, & Nitzberg, 1997), and GABL (Bani-mohammad, Ould-KHaoua, Ababneh, & Mackenzie, 2006). The results show that the performance of the HPS allocation strategy is much better than that of all other non-contiguous allocation strategies for both job size distributions considered in this research when the all-to-all communication pattern is used. This is because HPS has greater ability than the previous non-contiguous allocation strategies in alleviating message contention in the network through maintaining some degree of contiguity among allocated processors.

The results also show that the performance of HPS is close to that of the non-contiguous allocation strategies considered when one-to-all and random communication patterns are used. This is because in these communication patterns, the number of messages generated by jobs is small as compared with the all-to-all communication pattern and thus all the non-contiguous allocation strategies considered in this thesis including our proposed strategy have the same ability to alleviate the contention inside the network. However, the performance of HPS is not better than that of the other non-contiguous allocation strategies considered when the near-neighbor communication pattern is used, and this is because the near-neighbor communication pattern is suitable for the strategies that keep a high degree of contiguity between the allocated processors and maintain a rectangular form of the allocated sub-meshes (Alsardia, 2017). Moreover, HPS exhibits high system utilization as it manages to eliminate both internal and external fragmentation.

5.2. Directions for the Future Work

The aim of any allocation strategy is to minimize the average turnaround time and maximize the system utilization. For 2D mesh connected multicomputers, the simulation results show that the performance of the proposed HPS allocation strategy is much better than that of the previous non-contiguous allocation strategies considered in this research for both job size distributions considered when the all-to-all communication pattern is used. As a continuation of this research in the future, it would be interesting to adapt the proposed HPS non-contiguous allocation strategy to be applicable for the 3D mesh-connected multicomputer.

References

Ababneh, I., & Almomani, R. (2012). Communication overhead in noncontiguous processor allocation policies for 3D mesh-connected multicomputers. The International Arab Journal of Information Technology(IAJIT), 9(2), pp. 133-141.

Adve, V., & Vernon, M. (1994). Performance Analysis of Mesh interconnection networks with deterministic routing. IEEE Transaction on Parallel and Distributed Systems, 5(3), pp. 225-246.

Alsardia, D. (2017). Arow based non-contiguous processor allocation strategy for 2D mesh-connected multicomputers. Master Thesis. Al Albayt Univesity: Not published.

Babbar, D., & Krueger, P. (1994). A performance comparison of processor allocation and job scheduling algorithms for mesh-connected multiprocessors. Proceeding of the 6^th IEEE symposium on parallel and distributed processing, pp. 46-53.

Bani Mohammad, S. O. (2008). Efficient Processor Allocation Strategies for Mesh-Connected Multicomputers. PhD Thesis, The Faculty of Information and Mathematical Sciences, University of Glasgow, Glasgow, UK.

Bani-Mohammad, S., & Ababneh, I. (2013). On the performance of non-contiguous allocation for common communication patterns in 2D mesh-connected multi- computers. J. Simulation Model. Pract. Theory 32, pp. 155–165.

Bani-Mohammad, S., Ababaneh, I., & Hamdan, M. (2010). Comparative Performance Evaluation of Non-Contiguous Allocation Algorithms in 2D Mesh Connected Multicomputers. Proceedings of the 1oth IEEE International Conference on Computer and Information Technology (CIT 2010), pp. 2933-2939.

Bani-mohammad, S., Ould-KHaoua, M., & Ababneh, I. (2007). An Efficient Non-Contiguous Processor Aallocation Strategy for 2D Mesh Connected Multicomputers. Journal of Information Sciences, 177(14), pp. 2867-2883.

Bani-mohammad, S., Ould-KHaoua, M., Ababneh, I., & Mackenzie, L. (2006). 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, 2, pp. 41-48.

Blue Gene Project. (2010). http://www.research.ibm.com/bluegene/index.html.

Chang, Y., & Mohapatra, P. (1998). Performance improvement of allocation schemes for mesh-connected computers. Journal of Parallel and Distributed Computing, 52(1), pp. 40-68.

Foster, I. (1995). Designing and Building Parallel Programs. Concepts and Tools for Parallel Software Engineering. Addison-Wesley.

Intel Corporation. (1991). Paragon XP/S product overview. Beaverton, Oregon: Supercomputer Systems Division.

Kumar, V., Grama, A., Gupta, A., & Karypis, G. (2003). Introduction to Parallel Computing (2nd ed.). Rewood City, California: The Benjamin/Cummings publishing company, Inc.

Lo, V., Windisch, K., Liu, W., & Nitzberg, B. (1997). Non-contiguous processor allocation algorithms for mesh-connected multicomputers. IEEE Transactions on Parallel and Distributed Systems, 8(7), pp. 712-726.

Mohapatra, P. (1998). Wormhole Routing Techniques for Directly Connected Multicomputer Systems. ACM Computing Surveys, 30(3), pp. 374-410.

Ni, L. M., & McKinley, P. K. (1993). A survey of wormhole routing techniques in direct networks. IEEE Computers, 26(2), pp. 62-76.

Noakes, M., Wallach, D. A., & Dally, W. J. (1993). The J-machine multicomputer: an architecture evaluation. Proceedings of the 20th International Symposium Computer Architecture, pp. 224-235.

Peterson, C., Sutton, J., & Wiley, P. (1991). iWarp: a 100-MOPS, LIW microprocessor for multicomputers. IEEE Micro, 11(3), pp. 26-29.

ProcSimity v4.3. (1996). ProcSiimity v4.3 User's Manual. University of Oregon.

Suzaki, K., Tanuma, H., Hirano, S., Ichisugi, Y., Connelly, C., & Tsukamoto, M. (1996). Multi-tasking Method on Parallel Computers which Combines a Contiguous and Non-contiguous Processor Partitioning Algorithm. Proceedings of the 3rd International Workshop on Applied Parallel Computing, Industrial Computation and Optimization, pp. 641-650.

Windisch, K., Miller, J. V., & Lo, V. (1995). ProcSimity: an experimental tool for processor allocation and scheduling in highly parallel systems. Proceedings of the 5th Symposium on the Frontiers of Massively Parallel Computation (Frontiers'95), pp. 414-421.

Zhu, Y. (1992). Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers. Journal of Parallel and Distributed Computing, 16(4), pp. 328-337.

الملخص

استراتيجية التخصيص غير المتجاور باستخدام التقسيم الافقي في متعددات الحواسيب الشبكية ثنائية الابعاد

تصنف استراتيجيات تخصيص المعالجات في متعددات الحواسيب الشبكية ثنائية الابعاد إلى نوعين رئيسيين: المتجاور والغير متجاور. يشترط في استراتيجيات التخصيص المتجاور أن تكون المعالجات التي يتم تخصيصها للطلب متجاورة فيزيائيا، كما يجب ان يكون شكل شبكة المعالجات المخصصة لطلب ما مشابه لشكل الشبكة الاصلي، ونتيجة لذلك تعاني هذه الاستراتيجيات من مشاكل الكسيرات الخارجية والداخلية.، بينما في استراتيجيات التخصيص الغير متجاور، فيمكن تقسيم الطلب الى اجزاء اصغر وتخصيصه في شبكات منفصله عن بعضها البعض، وذلك بدلا من الانتظار حتى يتوفر شبكة معالجات لها نفس الحجم والشكل للطلب الاصلي.

هناك العديد من الاستراتيجيات المقترحة للتخصيص غير المتجاور، حيث تختلف هذه الطرق عن بعضها البعض في طريقة التخصيص للمعالجات في الشبكة، وبنسب متفاوتة لدرجة التجاور بين المعالجات المخصصة.

تم في هذا البحث اقتراح استراتيجية جديدة للتخصيص غير المتجاور للمعالجات في متعددات الحواسيب الشبكية ثنائية الابعاد، والتي يشار إليها باستراتيجية التقسيم الأفقي (HPS)، حيث يتم في هذه الطريقة تقسيم طلب التخصيص للمعالجات على أساس المعالجات الأفقية المتاحة للتخصيص في النظام، وذلك للحفاظ على درجة عالية من التجاور فيما بين المعالجات المخصصة للمهمة الواحدة، وتسمى هذه المعالجات الأفقية بالكتل الحرة، وكل منها يمثل صف من المعالجات الحرة التي يساوي حجمها عرض الشبكة. تقوم استراتيجية التقسيم الافقي على تغيير شكل طلب التخصيص بحيث يمكن احتوائه في الكتل الحرة، كما تحاول دائما تخصيص الطلب بشكل متجاور في الكتل الحرة من أجل تقليل المسافة التي تحتاجها الرسائل للانتقال بين المعالجات في النظام، وبالتالي تقليل التزاحم داخل الشبكة، مما يسهم في تحسين الأداء من حيث معدل مكوث المهام في النظام.

تم مقارنة أداء استراتيجية التقسيم الافقي مع استراتيجيات التخصيص غير المتجاور المعروفة باستخدام المحاكاة، وتبين النتائج أن أداء استراتيجية التقسيم الافقي أفضل بكثير من استراتيجيات التخصيص غير المتجاور الأخرى التي تم دراستها في هذا البحث، وعلاوة على ذلك، فإن استراتيجية التقسيم الافقي تعطي اشغالا عاليا للنظام لقدرتها على التخلص من الكسيرات الداخلية والخارجية كما هو الحال في استراتيجيات التخصيص غير المتجاور الأخرى التي تم دراستها.