A Row Based Non-Contiguous Processor Allocation Strategy for 2D Mesh-Connected Multicomputers

By

Dhaifallah S. Alsardia

Supervisor

Dr. Saad Bani-Mohammad

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

Deanship of Graduate Studies

Al al-Bayt University

May, 2017

Committee Decision

This Thesis (A Row Based Non-Contiguous Processor Allocation Strategy for 2D Mesh-Connected Multicomputers) was successfully defended and approved on 24/05/2017.

<u>Examination Committee</u>	**<u>Signature</u>**
Dr. Saad Bani-Mohammad (Supervisor)	…………………………
Prof. Ismael Ababneh	…………………………
Dr. Omar Shatnawi	…………………………
Dr. Shadi Aljawarneh	…………………………

Dedication

To my parents,
To my brothers and sisters
for their endless love, support and encouragement.

Acknowledgments

First of all, I would like to express my deep gratitude to my supervisor, Dr. Saad Bani-Mohammad for his inspiring guidance, valuable advice, and constant encouragement throughout the progress of this work. His suggestion and his frequent questions motivated this thesis and he never failed to provide his help at all stages of this thesis.

My great thanks are for my parents, my brother, and my sisters, without their encouragements and support I could not do anything.

I appreciate all of my friends who encouraged me during my master study; they truly helped me a lot.

List of Content

List of Figures

List of Tables

A Row Based Non-Contiguous Processor Allocation Strategy for 2D Mesh-Connected Multicomputers

By

Dhaifallah S. Alsardia

Supervisor

Dr. Saad Bani-Mohammad

Abstract

Multicomputer systems typically support diverse types of applications with various sizes and characteristics in a multiuser environment. Therefore, it is critical to use efficient processor allocation strategies to exploit the computation power of such systems. An efficient processor allocation strategy is that which maximizes system utilization and minimizes the jobs' turnaround time. In mesh-connected multicomputers, the processor allocation strategies can be classified into two main categories: contiguous and non-contiguous. In contiguous allocation, a job is allocated a submesh only if its processors are contiguous and form a shape the same as the connecting network. This allocation condition could lead to high processor fragmentation which could decrease the system performance in terms of system utilization and turnaround times of jobs. Non-contiguous processor allocation has been adopted as a feasible solution to the processor fragmentation problem. This adoption has encouraged by the emergence of the wormhole routing and advances in switching techniques which have made the communication latency less sensitive to the distance between the communicating nodes. Moreover, the experimental evidence has shown that only a slight improvement can be gained from further improving the existing contiguous allocation strategies. In non-contiguous allocation, a job request can be partitioned and allocated multiple disjoint submeshes instead of being queued waiting for a one to be available. This is expected to improve the system utilization and hence the average turnaround times of jobs. However, an extra communication overhead is expected due to the contention among messages of different jobs. The existing non-contiguous allocation strategies use various techniques to capture and allocate the available submeshes, however, in general, they focus on maintaining a high degree of contiguity among the processors of the allocated submeshes by compacting submeshes allocated to different jobs next to each other. In this thesis, a new non-contiguous allocation strategy, referred to as Row Based Strategy (RBS), has been suggested for 2D mesh-connected multicomputers, which alleviates the message contention inside the network. RBS classifies the incoming job requests according to their sizes into large and small in order to allocate them in a way that minimizes the contention among different jobs' messages. The simulation results have revealed that the proposed strategy is superior to that of the existing non-contiguous and contiguous allocation strategies in terms of job turnaround time when the all-to-all communication pattern is used, and this is due to its ability to alleviate message contention inside the network. Also, in most cases, it is relatively better than other allocation strategies for the one-to-all and random communication patterns.

Chapter One

Introduction

Parallel computers have been considered as one of the most powerful computing platforms that support large and complex applications in various areas. A parallel computer consists of multiple processing units that cooperate to solve a computational problem (Foster, 1995; Kumar, et al., 2003).

Parallel computers can be generally classified according to the memory architecture into two types: *shared memory* and *distributed memory* model. In shared memory model, also known as multiprocessors, processors communicate by modifying data in a shared memory, while in distributed memory model, also known as multicomputers, since each processor has its own memory, the processors communicate by exchanging messages via an interconnection network (Foster, 1995; Kumar, et al., 2003).

Interconnection networks provide a mechanism for data transfer among processing nodes. Typical, interconnection networks consist of links and switches. Generally, interconnection networks can be classified into static (also referred to as direct) and dynamic (referred to as indirect) networks. In dynamic networks, links are connected to each other dynamically by means of switches to form communication paths among processing nodes; examples of dynamic networks include bus-based

(Ferreira, et al., 1994), multistage interconnection (Kruskal and Snir, 1983) and crossbar (Fujii, et al., 1997). In static networks, there are point-to-point or direct communication links among nodes; examples of static networks include mesh (Adve and Vernon, 1994), *k*-ary *n*-cube (Min, 2003), and hypercube (Duato, et al., 1997).

Direct networks have been implemented in many large-scale multicomputer systems because they are scalable; it can be simply scaled up by adding nodes and channels based on the predefined network structure. Moreover, direct networks can exploit communication locality exhibited by many real-world applications (Bani-Mohammad, 2008).

Many networks architecture have been proposed for multicomputers, yet the mesh topology has gained much popularity because of its simplicity, scalability, regularity and ease of implementation (Babbar and Krueger, 1994; Das Sharma and Pradhan, 1996; Chang and Mohapatra, 1998; Yoo and Das, 2002). Two-dimensional mesh is an extension of a linear array to two-dimensions. Each node in 2D mesh is denoted by an ordered pair (x, y) to represent its row and column position respectively. Each node (except those at the edges) is connected to four neighbors by direct communication links.

Various regular structure applications such as matrix computations and image processing map very naturally into a 2D mesh. Three-dimensional mesh is a generalization of 2D mesh, where weather modeling and structural modeling are examples of computations that can be mapped naturally into this topology (Foster, 1995; Kumar, et al., 2003). Because of these features, mesh topology has been adopted in many commercial and experimental multicomputers.

The Intel Paragon (Intel Corporation, 1991), the Delta Touchstone (Intel Corporation, 1991), and the iWARP (Peterson, et al., 1991) are examples of 2D mesh-connected multicomputers. Examples of 3D mesh-connected multicomputers include the MIT J-machine (Noakes, et al.), the IBM blueGene/L (Blumrich, et al., 2003), and the Cray XT3 (Cray, 2005). 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.

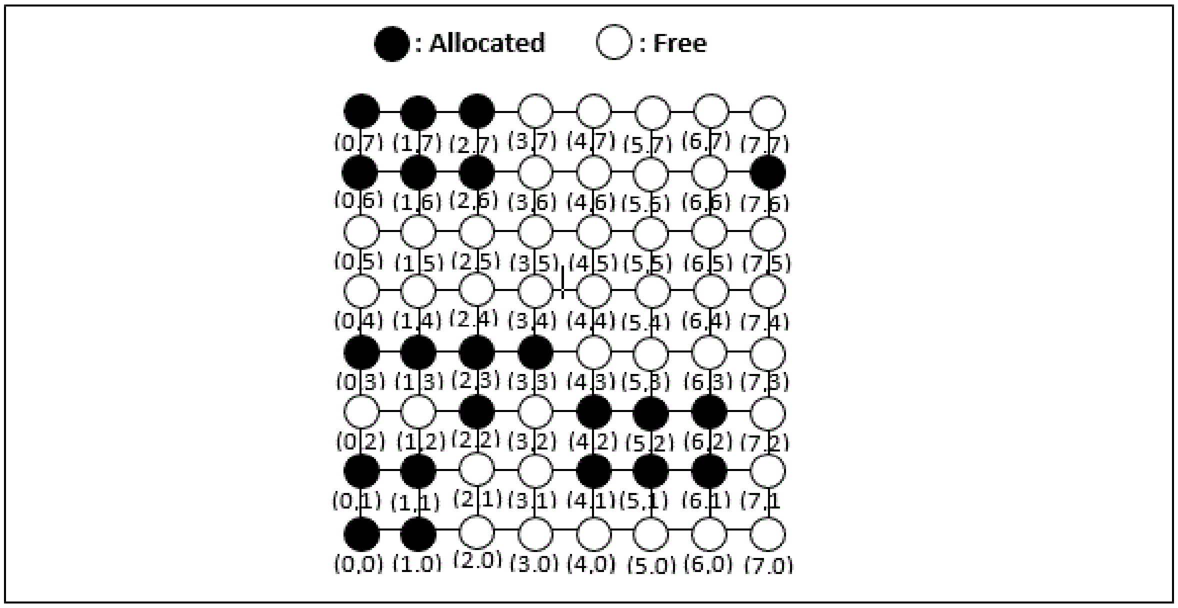


Figure 1.1 An example of an 8 × 8 2D mesh

1.1 Processor Allocation

Multicomputer systems typically support diverse types of applications with diverse sizes and characteristics in a multiuser environment. Therefore, processor management system is considered as a critical factor in exploiting the computational power of multicomputers (Windisch, et al., 1995; Chang and Mohapatra, 1998; Yoo and Das, 2002). Processor management system mainly comprised of *processor allocation* and *job scheduling*. Processor allocation is the assignment of a requested number of free processors to a requested job, while job scheduling is the policy that specifies the order of selecting a waiting job for execution (Babbar and Krueger, 1994; Ababneh and Bani-Mohammad, 2011). If the processor allocator failed to find a requested submesh for a selected job because of size and/or shape conditions, or if there are already awaiting jobs in the system, then it joins the waiting jobs queue. Once the allocator finds a suitable submesh for a selected job, then the job exclusively holds the processors in this submesh for the whole time of its execution. Upon completion of execution, the allocated processors are freed and become available for executing another job (Lo, et al., 1997; Windisch, et al., 1995; Chang and Mohapatra, 1998).

It is the allocation algorithm responsibility to find available submeshes for incoming job requests. This process is called *submesh recognition ability*. If the allocation algorithm can always find a submesh for an incoming job if at least one is available, then it is considered to have a *complete* recognition ability. Although,

the performance of the system improves as the submesh recognition of the allocation algorithm improves. Adopting a complete recognition ability algorithm could increase the complexity and the allocation overhead (i.e., allocation and deallocation time). The aim of any allocation algorithm is to minimize the *job turnaround time* (i.e. the time that the job spends in the system from arrival to departure (ProcSimity User's Manual, 1997)). Therefore, a good allocation algorithm is the algorithm that realize recognition-completeness with little allocation overhead (Yoo and Das, 2002).

Processor allocation strategies can be classified into two main categories: *contiguous* and *non-contiguous*. In contiguous allocation strategies (Li and Cheng, 1991; Zhu, 1992; Das Sharma and Pradhan, 1996; Chuang and Tzeng, 1994; Ababneh, 2001; Ababneh, et al., 2010), jobs are allocated to distinct submeshes of physically adjacent processors, with the same topology as the underlying interconnection network. Although, these strategies aim to eliminate the inter-process interference since only the communication of the same process are expected within a mesh, and hence the communication overhead is alleviated by decreasing the distances among the allocated processors. These strategies can cause high processor fragmentation because of the contiguity condition (Lo, et al., 1997; Chang and Mohapatra, 1998). This fragmentation is expected to degrade the system performance in terms of job turnaround time, due to the degradation of the *mean system utilization* (i.e. the percentage of processors that are utilized over a given time (ProcSimity User's Manual, 1997)).

Processor fragmentation comes out into two forms: *internal* and *external* (Das Sharma and Pradhan, 1996; Lo, et al., 1997; Chang and Mohapatra, 1998; Seo, 2005). Internal fragmentation occurs when a job is allocated more processors than it requests; typically, because of a restricted shape of submeshes allocation.

For example, powers of two squares as in (Li and Cheng, 1991), results in extra processors to be allocated to a requested job, while these processors are wasted and not used in the actual computation. External fragmentation occurs when a waiting job cannot be allocated even if the requested number of processors is available; this is because of the contiguity and shape conditions. Assuming that the system state shown in Figure 1.1 and the allocation algorithm is contiguous, if a job requests a 4×3 submesh of processors, then the algorithm fails to allocate the requested sub-mesh in spite of the sufficient number of free processors that are exist in the mesh.

Experimental evidence has shown that little performance improvement can be obtained from refinements of contiguous allocation algorithm (Lo, et al., 1997; Chang and Mohapatra, 1998). The evolution in networking technology such as the wormhole routing (Ni and McKinley, 1993; Mohapatra, 1998) and faster switching technique have reduced the impact of the distance between the communicating nodes on the communication latency (Lo, et al., 1997; Chang and Mohapatra, 1998), which has made the non-contiguous allocation feasible. The communication latency is the time that the message takes to be received by the destination node.

Several non-contiguous allocation strategies have been proposed (Lo, et al., 1997; Mache, et al., 1997; Chang and Mohapatra, 1998; Wu, et al., 2003; Moghaddam and Naghibzadeh, 2006; Bani-Mohammad, et al., 2007; Ababneh, 2008; Bani-Mohammad, et al., 2015; Bani-Mohammad, 2017), which can eliminate both internal and external fragmentation. In non-contiguous allocation, a job can be allocated to multiple disjoint smaller submeshes instead of being queued waiting for a contiguous one with a fit shape to be available. Although this leads to a better system utilization, the dispersal of submeshes, that can execute the same job, may increase the communication overhead due to the inter-process contention produced by messages from different jobs and long distances between the communicating nodes (Chang and Mohapatra, 1998; Lo, et al., 1997).

Therefore, it is desirable for the processor allocation strategy to be hybrid between contiguous and non-contiguous allocation strategies; meaning that, the allocation strategy should have the ability to partition the job while maintaining a high degree of contiguity among the allocated processors (Lo, et al., 1997; Bani-Mohammad, et al., 2007). Yet, it stills the allocation strategy responsibility to recognize and allocate the available sub-meshes in a way that minimizes the communication overhead in order to improve the overall system performance.

1.2 Motivation and Contribution

The non-contiguous processor allocation model has solved the problem of fragmentation that has been considered as the performance bottleneck of the contiguous processor allocation strategies and degrades the system performance in terms of job turnaround time and system utilization because of the physical contiguity and shape allocation conditions (Li and Cheng, 1991; Zhu, 1992; Lo, et al., 1997).

Non-contiguous allocation improves the system performance in terms of system utilization up to 78% for common workloads (Wan, et al., 1996; Lo, et al., 1997); this improvement is due to the ability of allocating several scattered submeshes to a requested job (Mache and Lo, 1997).

The main performance bottleneck of the non-contiguous processor allocation strategies is the message contention inside the network. The study proposed in (Min and Mutka, 1994), classifies the contention into two types: *internal contention* and *external contention*. Internal contention occurs when two or more routing paths within the same job try to use a physical channel at the same time. This type of contention is an inherent property of each job and it can occur in both contiguous and non-contiguous allocation strategies, while external contention occurs when two or more routing paths of different jobs try to use the same physical channel simultaneously. This type of contention occurs only in the non-contiguous allocation model. When non-contiguous allocation is adopted in a system with wormhole routing technique, the external contention increases the delay of the communication time (Min and Mutka, 1994).

Obviously, there is a tradeoff between the processor utilization due to the fragmentation problem and the jobs turnaround time due to the network contention (Min and Mutka, 1994; Moore and Lionel, 1996).

The contention depends on the switching technology in the underlying network and the communication pattern among the allocated processors (Min and Mutka, 1994). Although, contention can be negligible, when the *software latency* (i.e., the latency at sender and receiver for processing the message) is high or when the message size is small (Moore and Lionel, 1996), the communication overhead increases significantly due to the message contention among the messages of different jobs. This would increase the delay, and defect the gain of improved system utilization; and consequently, degrades the system performance in terms of jobs turnaround time (Min and Mutka, 1994; Mache and Lo, 1997). To improve the performance of the non-contiguous allocation strategies, it is important to choose the allocation strategy that causes minimal message contention (Mache and Lo, 1997), where the *spatial layout* (i.e., the geometric location) of the allocated submeshes in the mesh system plays a significant role in the interference among jobs' messages (Mache and Lo, 1997).

The existing non-contiguous allocation strategies (Lo, et al., 1997; Mache, et al., 1997; Chang and Mohapatra, 1998; Wu, et al., 2003; Moghaddam and Naghibzadeh, 2006; Bani-Mohammad, et al., 2007; Ababneh, 2008; Bani-Mohammad, et al., 2015; Bani-Mohammad, 2017) use various techniques to capture and allocate free sub-meshes in the mesh system.

However, in general, they focus on maintaining a high degree of contiguity among the processors in the allocated sub-meshes rather than reducing message contention in the submeshes that are allocated to different jobs.

Moreover, it is observed that the existing non-contiguous allocation strategies compact different sub-meshes to preserve larger contiguous sub-meshes for incoming job requests, expecting that this would reduce the communication overhead. Although this seems to be a good technique, but many experimental results in existing non-contiguous allocation strategies (Lo, et al., 1997; Bani-Mohammad, et al., 2007; Bani-Mohammad, et al., 2010; Bani-Mohammad, et al., 2015; Bani-Mohammad, 2017), under common system conditions, reveal that the average system utilization increases as the load of the job requests increases, until it eventually stabilized to a value about 80%, meaning that during the overall execution time, there would be, in average, about 20% of unutilized processors in the system, and a full system utilization is unachievable. Therefore, compacting different allocated submeshes seems to be a less significant factor in reducing the overall communication overhead. In contrast, considering the spatial layout when allocating submeshes for different job requests can reduce the message contention between the messages of these jobs, which results in reducing the overall communication overhead and hence improves the system performance.

Motivated by the above observations, a new row based non-contiguous processor allocation strategy for 2D mesh-connected multicomputer, referred to as Row Based Strategy (RBS) is proposed. The proposed strategy considers the spatial layout of the allocated submeshes in the mesh system. RBS classifies the incoming job requests according to their sizes, (large and small); in order to allocate them in submeshes that have minimal shared physical communication channel for the routing paths of their messages. Therefore, to alleviate message contention especially for large jobs, RBS tries to maintain a high degree of contiguity among the processors allocated to the same job with a little allocation overhead.

The simulation experiments results reveal that RBS performs much better than the previous non-contiguous and contiguous allocation strategies considered in this thesis in terms of jobs turnaround time when the all-to-all communication pattern is used. This is because all to all communication pattern produces much message collision and it is considered as the weak point of the non-contiguous allocation strategies (Suzaki, et al., 1996). The results have also shown that the performance of RBS is relatively better than that of the previous non-contiguous allocation strategies for one-to-all and random communication patterns in most cases. However, it is not better than that of the other contiguous and non-contiguous allocation strategies when the near neighbor communication pattern is used, because the privilege in this communication pattern is for the strategies that maintain a high degree of contiguity and maintain a rectangular shape of the allocated submeshes.

1.3 Outline of the Thesis

The rest of the thesis is organized as follows. Chapter 2 describes well-known contiguous and non-contiguous allocation strategies that have been proposed for mesh-connected multicomputer. Also, it presents some preliminaries required for understanding the subsequent chapters and provides a list of assumptions used in this research. Finally, the chapter describes the method of study used in this research and justifies the selection of simulation as a study tool.

Chapter 3 introduces the Row Based Strategy (RBS) as a new non-contiguous allocation algorithm for 2D mesh-connected multicomputers and describes the main features of the proposed strategy.

Chapter 4 analyzes and discusses the results of the simulation experiments and compares the performance of the proposed strategies against that of the well-known non-contiguous and contiguous ones.

Chapter 5 summarizes the main results presented in this research and outline possible directions to continue this work in the future.

Chapter Two

Background and Preliminaries

The main objective of this chapter is to describe some of the existing contiguous and non-contiguous allocation strategies that have been proposed in the literature (Li and Cheng, 1991; Zhu, 1992; Chuang and Tzeng, 1994; Das Sharma and Pradhan, 1996; Lo, et al, 1997; Chang and Mohapatra, 1998; Ababneh, 2001; Wu, et al., 2003; Moghaddam and Naghibzadeh, 2006; Bani-Mohammad, et al., 2007; Ababneh, 2008; Ababneh, et al., 2010; Bani-Mohammad, et al., 2015; Bani-Mohammad, 2017) for 2D mesh-connected multicomputers. This chapter also describes the system model assumed in this study. Such background is necessary for understanding the subsequent chapters.

2.1 Related Allocation strategies

This section overviews some of the existing contiguous and non-contiguous allocation strategies that have been proposed for 2D mesh-connected multicomputers.

2.1.1 Contiguous allocation strategies

Many non-contiguous allocation strategies (Li and Cheng, 1991; Zhu, 1992; Chuang and Tzeng, 1994; Das Sharma and Pradhan, 1996; Ababneh, 2001; Ababneh, et al., 2010) have been proposed for 2D mesh-connected multicomputers. Generally, most of them aim to reduce fragmentation caused by contiguity constraints in the mesh system, since the problem of high processor fragmentation can significantly affect the system performance. Below we describe some of the well-known strategies.

Two Dimensional Buddy System (2DBS): The 2DBS allocation (Li and Cheng, 1991) is proposed to square meshes with a side length of the power two. A requested job is also allocated to a square sub-mesh with a side length that is rounded up to the nearest power of two of the maximum side length of the requested job. If a job requests a sub-mesh of size $w \times h$, such that $w \leq h$, then the 2DBS allocates a sub-mesh of size $s \times s$, where $s = 2^{\lceil log_2(\max(w, h)) \rceil}$. For example, if a job requests a sub-mesh of size 2×4 it is allocated a square sub-mesh of size 4×4, which is more than its request, causing a 50% of internal fragmentation, as shown in Figure 2.1. This strategy suffers from high internal and external fragmentation because of the rigid side length condition, and it lacks complete sub-mesh recognition ability. Also, it is applicable only to square meshes (Zhu, 1992; Lo, et al., 1997;Chang and Mohapatra, 1998).

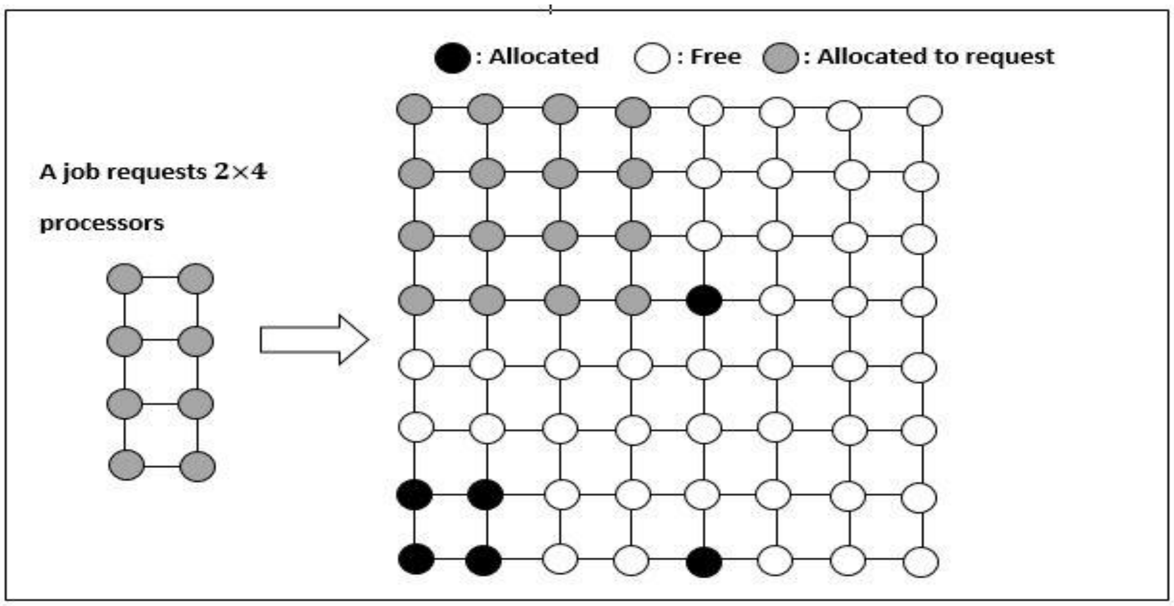


Figure 2. 1 An allocation using the 2D Buddy allocation strategy.

Frame Sliding (FS): The frame sliding strategy (Chuang and Tzeng, 1994) is proposed to reduce the fragmentation problem caused by 2DBS, it is applicable to any shape of a sub-mesh request in any mesh system. The FS algorithm slides a frame of a requested sub-mesh size across a bit array that represents allocated and free processors, to find an available sub-mesh. It starts at the lower leftmost free processor as a base of a candidate frame and examines for suitable frame by horizontal and vertical strides equivalent to width and length of the frame, respectively. The searching process ends when a suitable frame is found or when all candidate frames were checked. Although FS eliminates internal fragmentation, but it cannot recognize all available sub-meshes and it suffers from high external fragmentation. FS may fail to allocate a sub-mesh even a one exist because the jumps are by width and height of the requested sub-mesh. (Lo, et al., 1997; Chang and Mohapatra, 1998). An example of such case is shown if Figure 2.2.

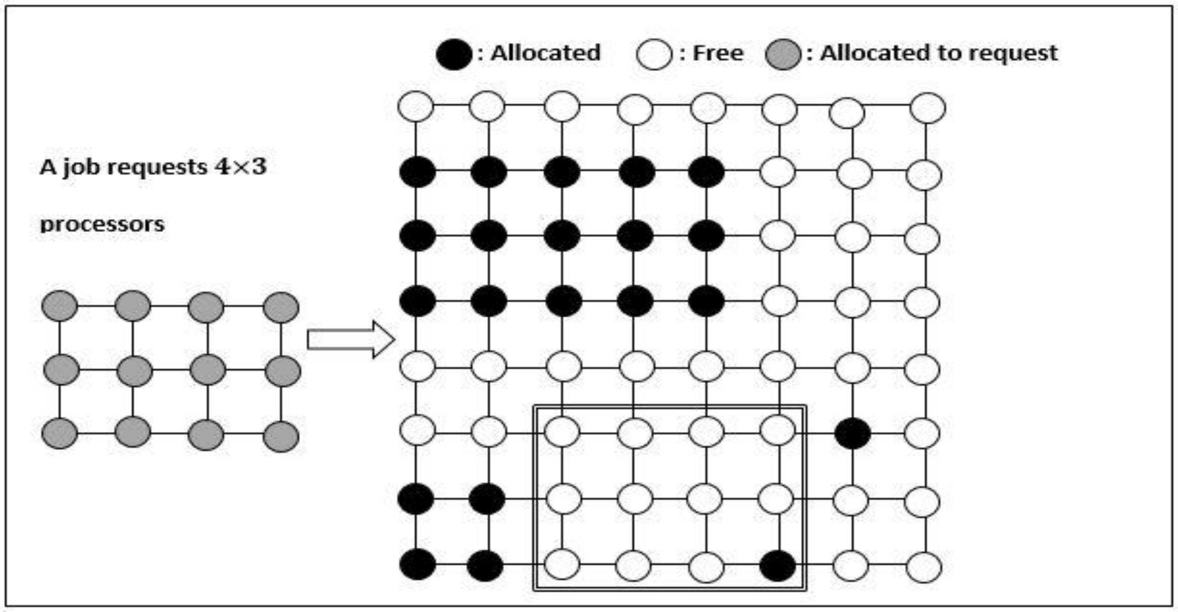


Figure 2.1: An allocation using the frame sliding strategy.

First Fit (FF) and Best Fit (BF): These strategies (Zhu, 1992) scan free sub-meshes represented in a bit array. FF allocates the first found sub-mesh with a sufficient number of processors, whereas BF allocates a sub-mesh with the least number of allocated neighbors to conserve a large contiguous mesh. Figure 2.3 shows the allocation of a job request for a 3×3 sub-mesh using FF and BF. These strategies have better sub-mesh recognition ability than 2DBS, nevertheless, they could fail to allocate large enough sub-meshes since they do not consider switching the requested shape orientation. Although the BF attempts to reduce the probability of fragmentation, both strategies suffer from significant external fragmentation (Lo, et al., 1997).

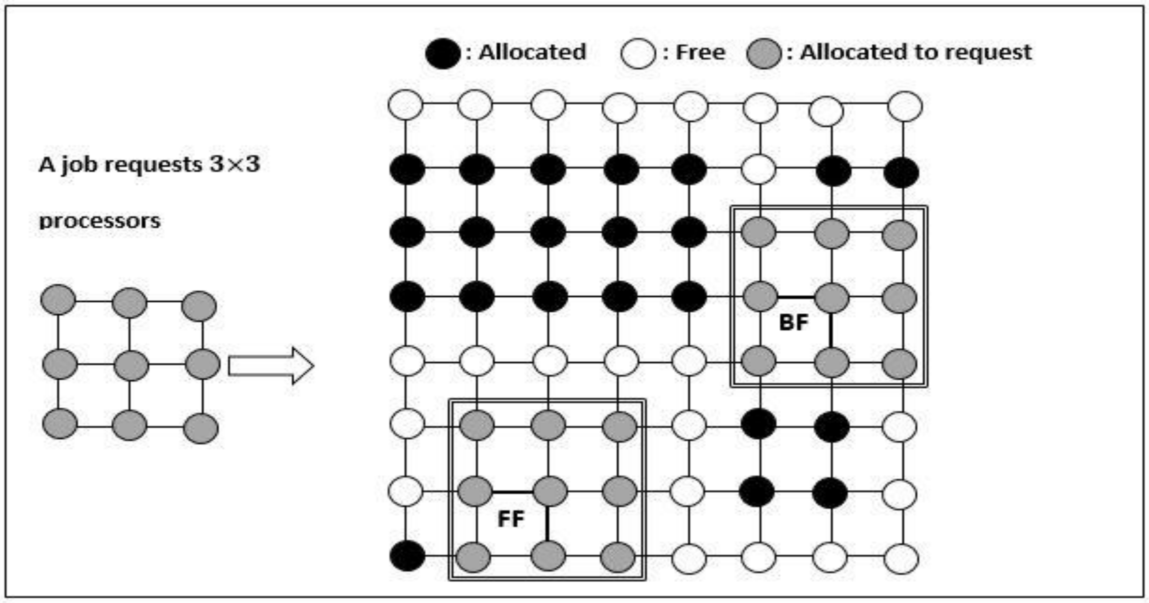


Figure 2.2: An allocation using First Fit and Best Fit strategies.

2.1.2 Non-Contiguous Allocation Strategies

Experimental evidence has shown that little performance improvement can be gained by refinements of contiguous allocation strategies (Lo, et al., 1997; Chang and Mohapatra, 1998). The wormhole routing (Ni and McKinley, 1993) and faster switching technique have made the communication latency less sensitive to the distance between the communication nodes, which has made the non-contiguous allocation feasible (Lo, et al., 1997; Chang and Mohapatra, 1998). Non-contiguous allocation allows a job to be executed when there are enough free processors in the mesh. Several non-contiguous allocation strategies have been proposed for 2D mesh multicomputers (Lo, et al., 1997; Chang and Mohapatra, 1998; Wu, et al., 2003; Moghaddam and Naghibzadeh, 2006; Bani-Mohammad, et al., 2007; Ababneh, 2008; Bani-Mohammad, et al., 2015; Bani-Mohammad, 2017). Some of the well-known non-contiguous allocation strategies that have been proposed in the literature are described below.

Random allocation strategy: This strategy (Lo, et al., 1997) simply allocates k randomly selected available processors to a request for k processors. Despite its simplicity and fragmentation elimination, it does not enforce contiguity. Therefore, it is expected to cause much communication interference between jobs (Lo, et al, 1997).

Paging: In paging strategy (Lo, et al., 1997), the whole mesh is partitioned into equal sized sub-meshes called pages. The page size is 2^{page_size}; where $page\_size$ is non-negative integer. The page is the basic unit of allocation. Four different indexing schemes are proposed for indexing the pages (row-major, shuffled row-major, snake-like, and shuffled snake-like) as shown in Figure 2.4. A paging algorithm is represented by indexing scheme and page size as $paging_{indexing\_scheme}(page\_size)$. A job that requests k processors is allocated $\ulcorner (\frac{k}{2^{page\_size}.\ 2^{page\_size}}) \urcorner$ page of processors, by traversing the free page list according to the given indexing scheme (Lo, et al., 1997). Paging (0) eliminates both internal and external fragmentation. Much contiguity can be enforced by increasing the page size, but as the page size increases, the paging would probably incur much internal fragmentation. Partitioning in Paging is based on the characteristics of page, which is globally predefined and independently from the request (Bani-Mohammad, et al., 2010). Consequently, it may fail to allocate a job contiguously even a one sufficient mesh is available. Figure 2.5 illustrates an allocation example of Paging $_{\text{row-major}}$ (0).

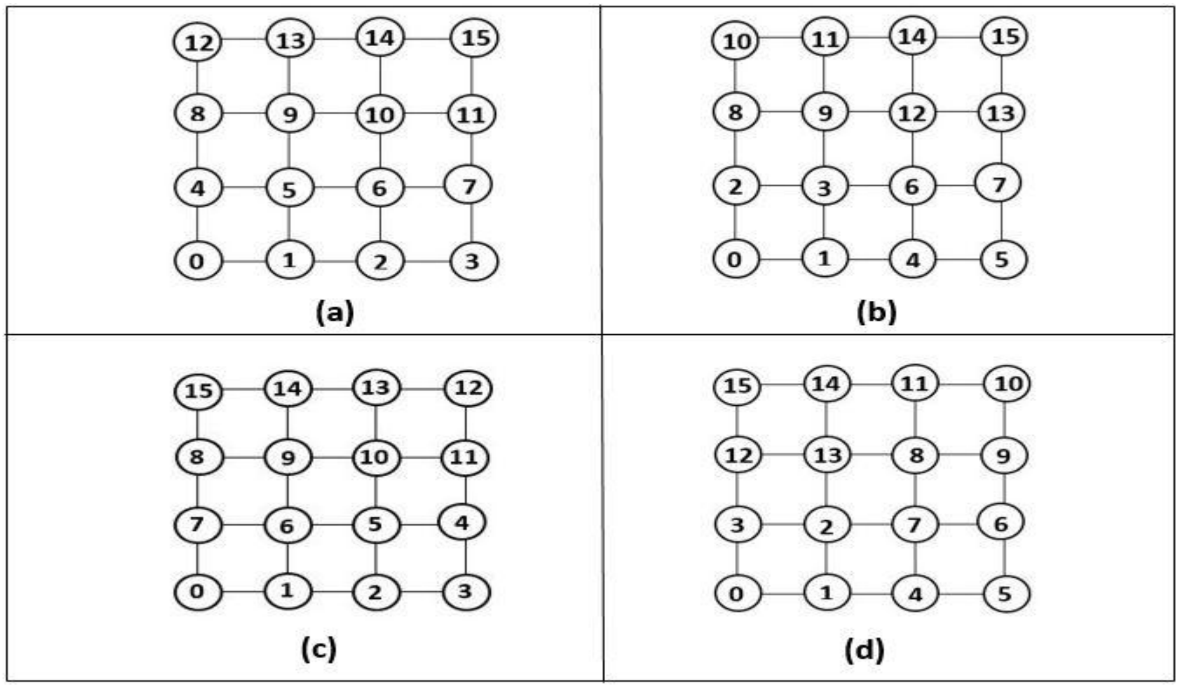


Figure 2.3: Paging(0) using different indexing schemes: (a) Row-major indexing, (b) Shuffled row-major, (c) snake-like indexing, and (d) shuffled snake-like indexing.

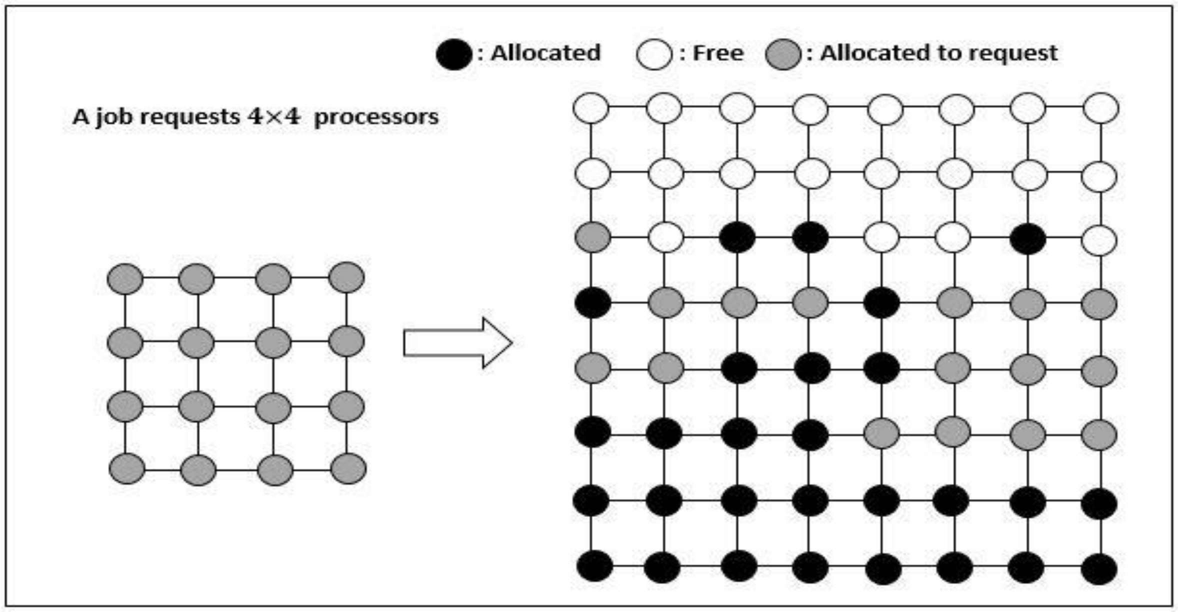


Figure 2.4: An allocation using Paging $_{\text{Row\_major}}$ (0) strategy.

Multiple Buddy Strategy (MBS): The MBS (Lo, et al., 1997) is an extension of the 2D buddy strategy. The mesh is divided into distinct square sub-meshes with side lengths equal to the powers of two upon initialization. The number of processors requested by an incoming job is factorized into a base of four representation of $\sum_{i=0}^{\log_4 p} d_i \times (2^i \times 2^i)$, where $0 \leq d_i \leq 3$. The request is then allocated to the mesh according to the factorized number in which d_i number of $2^i \times 2^i$ blocks is required. If a required block is unavailable, MBS recursively searches for a larger block and repeatedly breaks it down into buddies until it produces blocks of the desired size. If that fails, the requested block is then broken into four requests for smaller blocks and the searching process repeats. MBS eliminates fragmentation, while still maintaining contiguity within individual blocks (Lo, et al., 1997).

A main drawback of the MBS is that it may fail to allocate an available sub-mesh contiguously to a requested job because it is restricted to base 4 blocks of allocation. An example of MBS allocation is shown in Figure 2.6.

Expanding Square Strategy (ESS): This strategy (Moghaddam and Naghibzadeh, 2006) introduced with the aim of minimizing internal and external message-passing contention. ESS works as follows: when the system receives a job request, each idle node in the mesh builds a square around itself and starts to expand it and in each expansion, all the idle nodes are added to the cluster. If in the last expansion, the number of idle nodes exceeded the number of requested processors, then the nodes with minimum sum distance from all other allocated nodes in their corresponding clusters are added and the job is assigned to these nodes. Figure 2.7 illustrates an example of the ESS allocation.

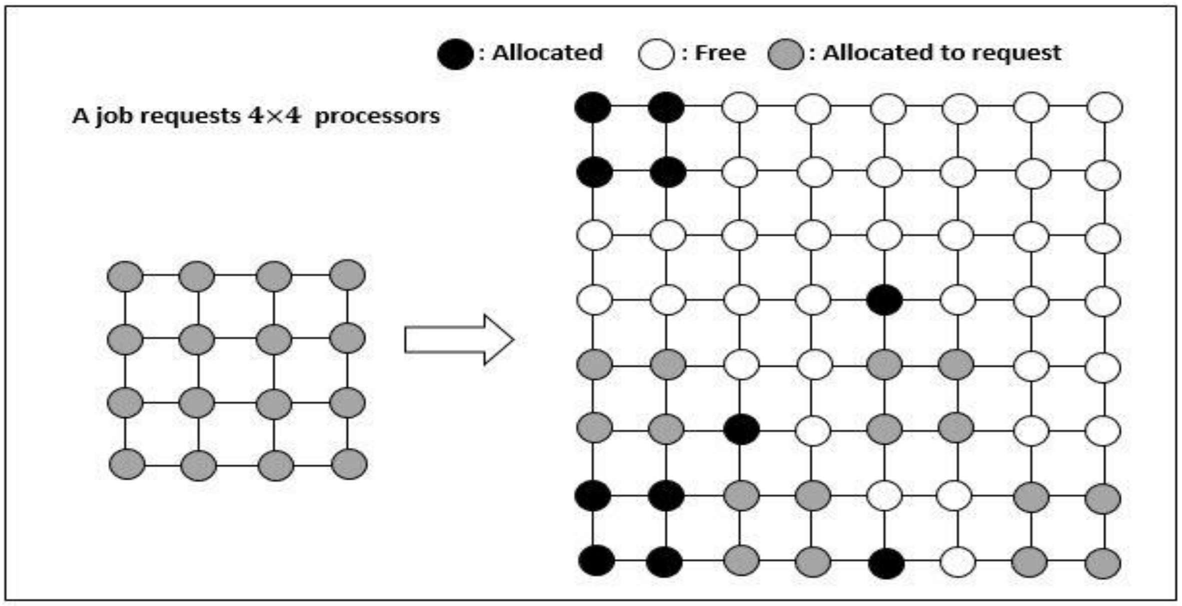


Figure 2.5: An allocation using MBS allocation strategy.

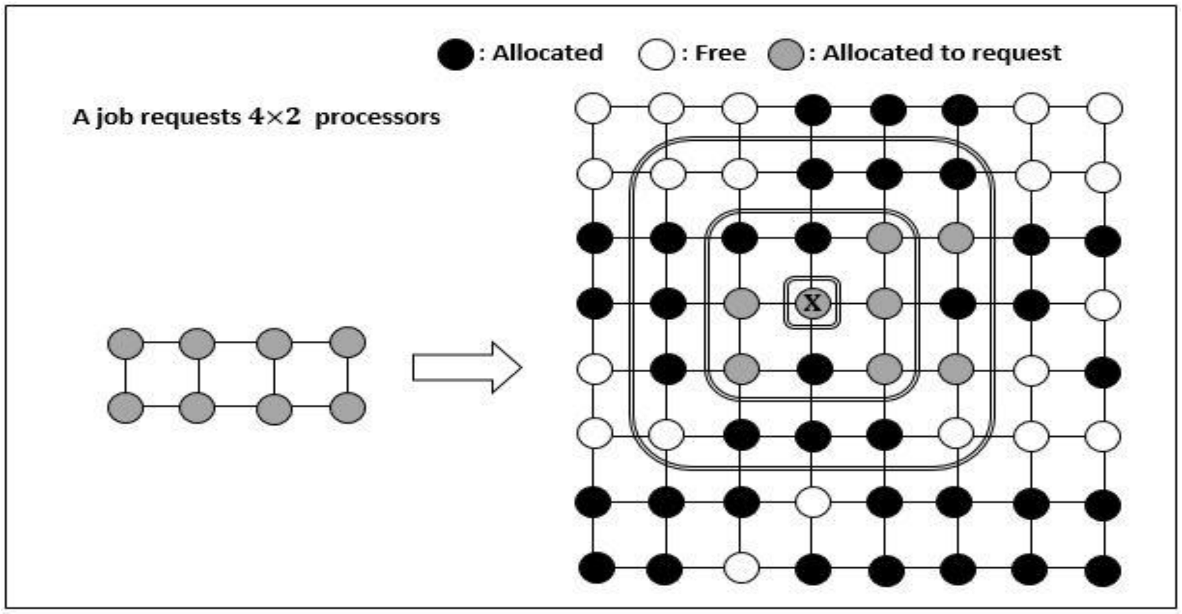


Figure 2.6: An allocation using ESS allocation strategy.

***Greedy Available Busy List (GABL)**:* In GABL strategy (Bani-Mohammad, et al., 2007), when a parallel job is selected for allocation, a sub-mesh suitable for the entire job is searched for. If such a sub-mesh is found, it is allocated to the job and allocation is done. Otherwise, the largest free sub-mesh that can fit inside the request job size is allocated. Then, the largest free sub-mesh whose side lengths do not exceed the corresponding side lengths of the previously allocated sub-mesh is searched for and allocated provided that this does not result in allocating more processors than the requested size. This last step is repeated until the requested number of processors is allocated. Allocated sub-meshes are kept in a busy list. Each element in this list includes the *id* of the job the sub-mesh is allocated to. GABL uses an efficient approach proposed in (Chiu and Chen, 1999), to facilitate the detection of such available sub-meshes with low allocation overhead. GABL aims to maintain a high degree of contiguity to decrease the number of allocated sub-meshes to a job and hence decreases the distance traversed by a message, which can reduce message contention inside the network (Bani-Mohammad, et al., 2007). Even though, GABL may allocate submeshes that are far apart from each other. To illustrates how GABL allocates a job request, consider the system state shown in Figure 2.8, and assume a job request of size 4×4 arrives at the system, GABL always tries to allocate any job request contiguously. It scans the mesh, searching for a free submesh of the requested size, in this case, 4×4. GABL failed to find such a contiguous submesh, then it starts by subtracting one from the maximum side length of the requested submesh and this step is repeated until it finds a suitable available submesh, in this case a 2×3 available submesh of processors with the coordinates $(6,0,7,2)$ is found, where the first two coordinates specify the lower left corner of the submesh and the last two coordinates specify the upper right corner of the submesh Then it continues to allocate another two submeshes: $(3,0,5,1)$ and $(6,6,7,7,)$ as shown in Figure 2.8 by applying the steps described above.

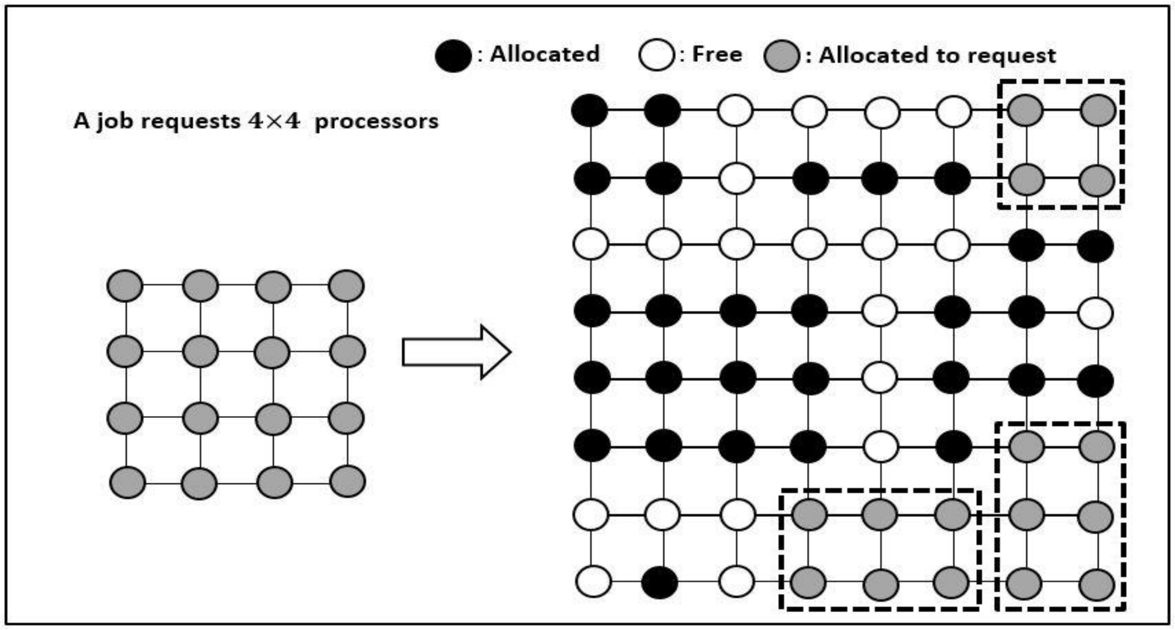


Figure 2.7: an example of allocation using GABL allocation strategy.

2.2 Switching Method

The switching method refers to the method used to transfer a message from a source to a destination usually through a series of intermediate nodes by removing the data from the input channel and placing it on the output channel at each intermediate node. The switching technique has a significant impact on the communication latency in the direct network multicomputer systems. Among several switching techniques that have been used in multicomputer systems, this section briefly describes three most important ones: *Store-and-forward* (Kumar, et al., 2003), *Virtual cut-through* (Drewes, 1996), and *Wormhole switching* (Ni and McKinley, 1993; Mohapatra, 1998).

Store-and-forward switching: In store-and-forward switching, also called packet switching, the message is divided into fixed-length packets that are independently routed to their destination, since each node holds its destination address in its header. Each intermediate node stores the entire packet before forwarding it to the next node in its path. The major drawback of store-and-forward switching is that the time required to transmit a packet from source to destination is proportional to the number of traversed intermediate nodes. Furthermore, we need a buffer space to hold packets at each intermediate node (Ni and McKinley, 1993; Mohapatra, 1998).

Virtual cut-through switching: Virtual cut-through (Drewes, 1996) has been introduced as an enhancement of store-and-forward switching. Virtual cut-through reduces the time and space overhead of storing the entire packet at each intermediate node. In virtual cut-through, an intermediate node stores a packet only if the next required channel is busy. This reduces the impact of the distance between the communicating nodes on communication latency. However, a very large buffer space is required at each node to store all blocked transient packets due to the probability of blocking multiple messages at any node, and this leads to increase in the implementation cost (Ni and McKinley, 1993; Mohapatra, 1998).

Wormhole switching: Wormhole switching (also called wormhole routing (Duato, Yalamanchili, and Ni, 1997)) is a variant of virtual cut-through technique that eliminates the need for large buffer spaces and minimizes the sensitivity of the communication latency to the distance between the communication nodes. In wormhole switching, a packet is divided into fixed-size units called flits

(flow control unit), which is the smallest units of data transmission in wormhole routing network. The header flit(s), which contains the routing information, headway along the routing path and the remaining data flits follow it contiguously in a pipelined fashion. When the header flit blocked due to resource contention (link or buffer), then all trailing flits blocked and occupy the buffers at the intermediate nodes, typically, one flit at each intermediate node. This can block other messages and further, it can lead to a deadlock, where messages wait for each other in a cycle without being able to move forward anymore. Deadlock prevention is a critical issue in wormhole switching and it is usually achieved by suitable choice for routing function (Ni and McKinley, 1993; Mohapatra, 1998).

Since wormhole routing pipelines packets during transmission, it can perform well even in high-diameter networks, such the mesh (Min, 2003). Many experimental machines such as the iWARP (Peterson, Sutton, and Wiley, 1991) and the MIT J-machine (Noakes, et al., 1993), and commercial machines such as the Intel Paragon (Intel Corporation, 1991), the IBM blueGene/L (Blumrich, et al., 2003), and the Cray XT3 (Cray, 2005) have used wormhole switching. Wormhole switching has been used in this research when examining the performance of the allocation strategies. The wormhole switching has been used in this research because it has been used in the previous non-contiguous allocation strategies (Lo, et al., 1997; Mache, et al., 1997; Bani-Mohammad, et al., 2007; Ababneh, 2008;Bani-Mohammad, et al., 2010; Bani-Mohammad, et al., 2015).

2.3 Routing Algorithm

An efficient routing algorithm is critical to the performance of the parallel multicomputer. A routing algorithm determines the path that a message follows from its source to its destination. Routing algorithms can be classified as deterministic and adaptive. Deterministic routing determines a unique path of the message according to the source and destination address. Adaptive routing determines the path of the message according to the current state of the network such as the presences of failure or congestion and routes along alternative paths. When designing a routing algorithm, deadlock handling should be considered (Ni and McKinley, 1993; Mohapatra, 1998; Kumar, et al., 2003).

Dimension-order routing (Ni and McKinley, 1993; Mohapatra, 1998; Kumar, et al., 2003) is a deterministic routing technique and it provides deadlock-free routing for wormhole-routed networks; since messages' path cannot form a deadlock cycle. In Dimension-ordered routing, a sent packet traverses along one dimension at a time until it reaches the appropriate coordinate then it traverses along the next dimension towards the destination.

Dimension-order routing in 2D mesh networks is referred to as XY routing, where the packet first traverses along the X dimension (the mesh width) until it reaches the column of the destination node then it traverses in the Y dimension (the mesh height) until it reaches the destination as depicted in Figure 2.9. XY routing is used in this research when examining the performance of

the allocation strategies. XY routing has been used in this research because it has been used in the previous non-contiguous allocation strategies (Lo, et al., 1997; Bani-Mohammad, et al., 2007; Ababneh, 2008; Bani-Mohammad, et al., 2010; Bani-Mohammad, et al., 2015).

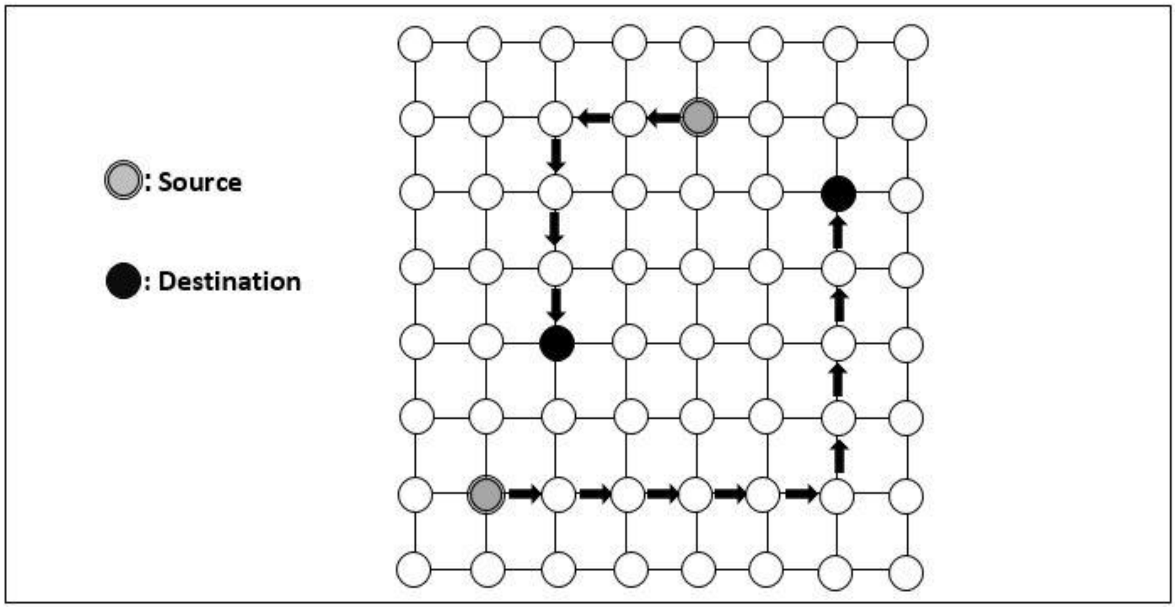


Figure 2.8: Dimension-ordered (XY) routing in an 8×8 2D mesh-connected network.

2.4 Communication Patterns

Processors allocated to a parallel job often communicate with each other according to a given communication pattern (Lo, et al., 1997). When evaluating the non-contiguous allocation, the important parameter to measure is the message contention that caused by the exchanged messages and its impact on the overall system performance. Four communication patterns have been considered in this research work to evaluate the performance of

the proposed non-contiguous allocation algorithm and compared it with that of the existing algorithms. The first communication pattern is *one-to-all* (ProcSimity Manual, 1997), where a randomly selected process sends a message to each other processors allocated to the same job. The second communication pattern is *all-to-all* (ProcSimity Manual, 1997), where each processor in a job sends a message to all other processors allocated to the same job. This communication pattern causes much message contention and is considered as the weak point of the non-contiguous allocation algorithms (Suzaki, et al., 1996). The third communication pattern is *random* (ProcSimity Manual, 1997), where a message is sent between a randomly selected pair of processors (source and destination) within the same job. In the fourth communication pattern *near-neighbor* (Bani-Mohammad and Ababneh, 2013), each processor communicates with its neighbors.

2.5 Assumptions

In the subsequent chapters, extensive simulation experiments will be presented to evaluate the proposed allocation strategy (RBS). In this study, we make the following assumptions which have been mostly used in the literature (Zhu, 1992; Babbar and Krueger, 1994; Suzaki, et al., 1996; Mache, et al., 1997; Chang and Mohapatra, 1998; Ababneh, 2001; Yoo and Das, 2002; Seo, 2005; Bani-Mohammad, et al., 2007; Ababneh, 2008; Bani-Mohammad, 2008; Bani-Mohammad, et al., 2010; Bani-Mohammad, et al., 2015)

- The inter-arrival times of jobs are independent and follow an exponential distribution.
- Jobs are scheduled on a First-Come-First-Served (FCFS) basis.
- The execution times of jobs depend on the time needed for flits to be routed through the node, packet sizes, the number of message sent, message contention and distances messages traverse.
- The side lengths of the sub-meshes requested by jobs are generated independently and follow a given probability distribution. Two distributions have been considered in this research. The first is the uniform distribution over the range from 1 to the mesh side length (L).

- The second is the uniform-decreasing distribution. It is determined by four probability $p1$, $p2$, $p3$, and $p4$, and four integers $l1$, $l2$, $l3$ and $l4$, where the probability that the width (length) of a request falls in the ranges $[1, l1]$, $[l1 + 1, l2]$, $[l2 + 1, l3]$ and $[l3 + 1, l4]$ is $p1$, $p2$, $p3$, and $p4$, respectively. The side lengths within a range are equally likely to occur. For the simulation experiments in this research work, $p1 = 0.4$, $p2 = 0.2$, $p3 = 0.2$, $p4 = 0.2$, $l1 = L/8$, $l2 = L/4$, $l3 = L/2$, and $l4 = L$. These distributions have often been used in the literature (Zhu, 1992; Lo, et al, 1997; Chang and Mohapatra, 1998; Chiu and Chen, 1999; Ababneh and Bani-Mohammad, 2003; Bani-Mohammad, et al., 2006)
- Messages are transmitted inside the network using wormhole switching along with XY routing.
- Messages are of a fixed length (i.e., a fixed number of flits). Moreover, the number of messages that are generated by a given job are correlated to the job size in the one-to-all, all-to-all and near-neighbor communication patterns, since each job does exactly one iteration of the given communication pattern, and it is only one message per job in the random communication pattern.

2.6 The Simulation Tool (ProcSimity Simulator)

Procsimity (Windisch, et al., 1995; ProcSimity Manual, 1997) is a well-known software tool for research in the area of processor allocation and job scheduling for distributed memory multicomputers. It has been developed at the university of Oregon, and the developments efforts have been supported by OACIS and NSF (Windisch, et al., 1995). The tool was written in the C programming language and has been used extensively in evaluating processor allocation and job scheduling strategies in the mesh-connected multicomputers. ProcSimity has been preferred because it is open source and includes a detailed simulation of important operations of multicomputers networks. Moreover, it has been extensively validated in (Windisch, et al., 1995; ProcSimity Manual, 1997).

ProcSimity allows the user to test the performance of scheduling and allocation algorithms on job streams comprising a spectrum of parallel applications.

The tool supports experimentation for highly parallel systems based on the mesh and k-ary n-cube topologies (includes hypercube and torus), and for a range of flow control and routing technologies. The overall purpose for ProcSimity is to provide a convenient environment for performance analysis of processor allocation and job scheduling algorithms. In particular, ProcSimity is designed to investigate some of the key performance bottlenecks in the areas of scheduling and allocation, such as fragmentation and communication overhead problems.

These areas of processor management have been shown to be critical for achieving good price/performance ratios in highly parallel systems in a dynamic multi-user environment. (Windisch, et al., 1995; ProcSimity Manual, 1997).

ProcSimity specifies the target machine environment including the network topology, routing, and flow control mechanisms, and it provides the users with libraries of predefined scheduling and allocation algorithms. In addition, a user can easily develop and integrate its own allocation and scheduling algorithms and even a new communication pattern into ProcSimity tool. Procsimity involves specification of the simulation experiments; it supports both stochastic job streams as well as communication patterns from actual parallel applications. The user can specify detailed simulation of message-passing overhead at the flit level (Windisch, et al., 1995; ProcSimity Manual, 1997).

2.7 Justification of the Method of Study

System performance can be generally, evaluated by using two techniques: analytical modeling and simulation, in addition to conducting measurements on a real practical system, which may be costly or does not permanently available. The level of the desired accuracy is considered as one of the key consideration when adopting a given evaluation technique. In general, analytical models have often low requirements in terms of computation costs, but they often rely on many assumptions and simplifications that restrict their applicability to a limited number of scenarios.

In contrast, simulation models can easily incorporate details to the desired level of accuracy in order to mimic more closely the behavior of the real system. The consequence of this is that simulation often require a longer time to develop and run the code, compare to analytical modeling (Bani-Mohammad, 2008). However, as we have used the ProcSimity simulator that has already been developed and extensively validated (Windisch, et al., 1995; ProcSimity Manual, 1997), we have easily integrated the suggested algorithm into the simulator. This has helped to considerably cut down the development time and debugging of the code.

Chapter Three

Row Based Strategy (RBS): A New Non-contiguous Processor Allocation Algorithm for 2D Mesh-Connected Multicomputer

3.1 Introduction

Conventional allocation strategies (Li and Cheng, 1991; Zhu, 1992; Chuang and Tzeng, 1994; Das Sharma and Pradhan, 1996; Ababneh, 2001, Ababneh, et al., 2010) suggested for mesh-connected multicomputer are based on contiguous allocation, where the processors are allocated to a parallel job only if they are physically contiguous and form a shape that resembles the connecting network topology. These allocation conditions could cause internal and external processor fragmentation and degrade the overall system performance due to the inefficient utilization of the system.

Non-contiguous allocation strategies (Lo, et al., 1997; Chang and Mohapatra, 1998; Wu, et al., 2003; Moghaddam and Naghibzadeh, 2006; Bani-Mohammad, et al., 2007; Ababneh, 2008; Bani-Mohammad, et al., 2015; Bani-Mohammad, 2017) for mesh-connected multicomputers have been proposed with the aim of alleviating the fragmentation problem by ignoring the contiguity conditions. In non-contiguous allocation, a parallel job can be allocated to multiple disjoint available submeshes instead of being queued waiting for a contiguous one to be available (Lo, et al., 1997; Yoo and Das, 2002).

Two main reasons have led to the adoption of non-contiguous allocation; the first one is that the experimental evidence has shown that only slight improvement can be obtained from refining the existing contiguous allocation strategies (Lo, et al., 1997; Chang and Mohapatra, 1998), and the second is the advances in network switching techniques and the emergence of wormhole routing (Ni and McKinley, 1993) which have made the network latency less sensitive to the distance between the communicating nodes (Lo, et al., 1997; Chang and Mohapatra, 1998).

The non-contiguous allocation has improved the system utilization up to 78% (Wan, et al., 1996; Lo, et al., 1997), this improvement can notably improve the overall system performance such as the jobs turnaround times and the jobs finish times. Even though, the non-contiguous allocation suffers from the problem of message contention inside the network, and if the contention increased significantly, then it would increase the communication latency and even would defeat the benefits of the improved system utilization (Min and Mutka, 1994; Mache and Lo, 1997).

The existing non-contiguous allocation (Lo, et al., 1997; Mache et al., 1997; Chang and Mohapatra, 1998; Wu, et al., 2003; Moghaddam and Naghibzadeh, 2006; Bani-Mohammad, et al., 2007; Ababneh, 2008; Bani-Mohammad, et al., 2015; Bani-Mohammad, 2017) use various techniques, often based on artificial predefined geometric or arithmetic patterns, to recognize and allocate the available submeshes. Generally, most of them focus on maintaining a high degree of contiguity among the processors of the allocated submeshes to a given job. Although, in the wormhole routing the distance between the communicating nodes is not considered as

the major factor in communication latency. In addition, they generally aim to compact different allocated submeshes in the hope of preserving larger available submeshes for incoming job requests, although, the full system utilization is unachievable.

Motivated by the above observations, in this chapter we describe a new non-contiguous allocation strategy for 2D mesh-connected multicomputers, referred to as Row Based Strategy (RBS for short). RBS aims to alleviate the message contention inside the network, which is the main drawback of the non-contiguous allocation strategies.

3.2 Preliminary

The target system is a 2D mesh with size $N = W \times H$, where W is the width of the mesh and H is its height.

Each processor (node) is denoted by an ordered pair (x, y), which are the coordinates of that processor, where $0 \leq x < W$ and $0 \leq y < H$.

Each processor is connected by a bidirectional communication link to its neighbors, and each node except those at the edges is connected by four such links.

Each row in the mesh is denoted by its y coordinate as $R(y)$.

As shown in Figure 3.1, each block of consecutive rows is denoted by its beginning and ending rows $R(b, e)$ respectively, where $0 \leq b < H$ and $b \leq e < H$.

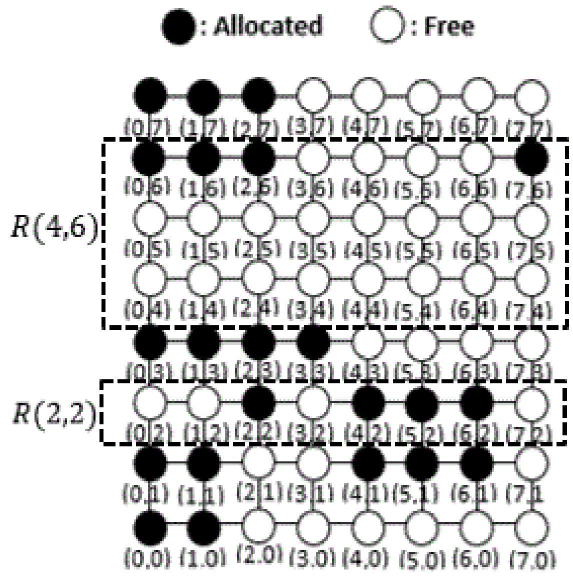


Figure 3.1: An example of an 8×8 mesh system

3.3 The Proposed Row Based Allocation Strategy (RBS)

The RBS allocation strategy classifies the incoming job requests according to the requested submesh size into two categories: *large* and *small*. If the requested submesh size is greater than the mesh width, it is considered large. Otherwise, it is considered small.

The main purpose of this classification is to alleviate message contention among large jobs by reducing the number of allocated processors to a large job in the rows, which already contain processors allocated to other large jobs. Also, it tries initially to allocate small jobs in the upper part of the mesh. Knowing that the messages of two adjacent small jobs allocated next to each other in the same row would not collide.

For a small job request, if the number of free processors in the mesh is sufficient to accommodate the requested submesh size, then it is allocated using one of the two different allocation methods, described below, according to the current allocation state of the mesh.

Method S.1: If the incoming job request is small (i.e. the requested submesh size k is less than or equal to the mesh width), the proposed strategy works as follows: starts at the top row of the mesh downwards, trying to find a row with a sufficient number of free processors to accommodate the requested job size. If found, then the k leftmost free processors at that row are allocated, and the allocation is done.

Method S.2: if a small job request cannot be allocated in a single row, then the strategy allocates it as follows: starts at the top row downwards, and allocates the rightmost free processors in the current row until the requested number of processors is allocated, if there are no more free processors remain in the current row and the required number of processors is not yet allocated, then it steps down to the next lower row in the mesh and continues allocating the same way until the requested number of processors is allocated.

Method S.1 Example: Assume that the mesh shown in Figure 3.2 and a job requests 4 processors, a 2×2 submesh, the strategy begins at top row, $R(7)$, the number of free processors is 2, which is less than the requested number of processors, then it steps down to the next lower row, $R(6)$, now the number of free processors in this row is 5, which is sufficient to accommodate the requested size.

Then the 4 leftmost free processors at $R(6)$ are allocated and the allocation is done.

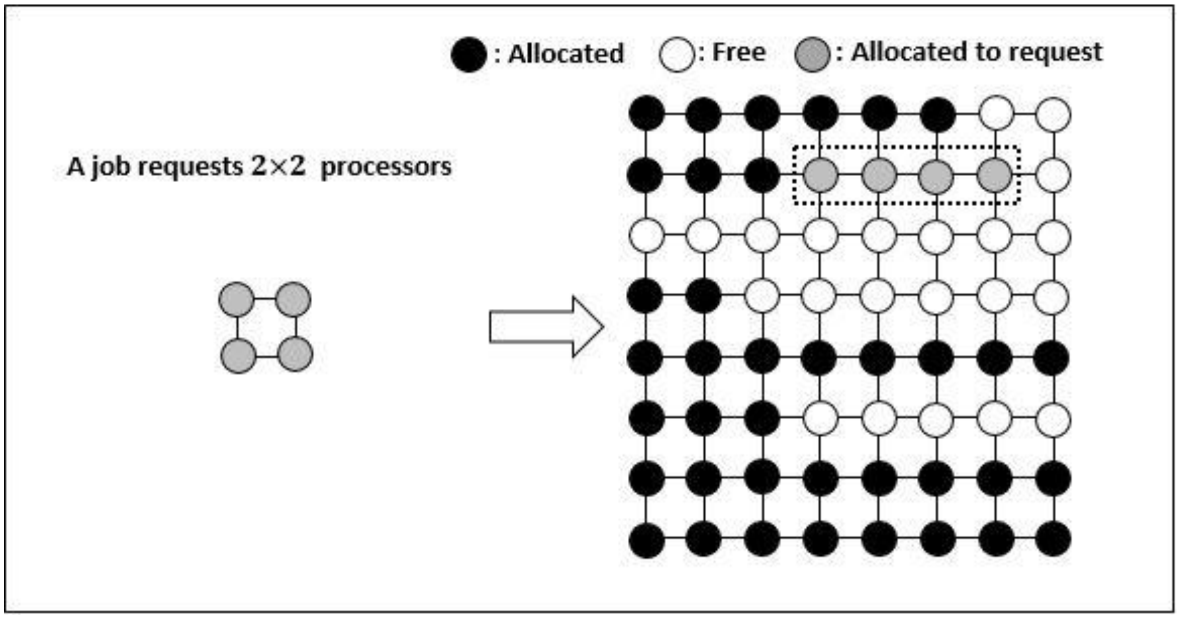


Figure 3.2: Allocating processors to a job requests a 2×2 submesh using the RBS.

Method S.2 Example: Assume that the mesh shown in Figure 3.3 and a job requests 7 processors, a 1×7 submesh. Since there is no a single row which contains a sufficient number of free processors to accommodate the requested number of processors, the strategy begins at top row, $R(7)$, and allocates the rightmost free processors which are 2, then it steps down to the next lower row, $R(6)$, and allocates the only one free processor at that row, and it continues allocating the same way until the requested number of processors are allocated.

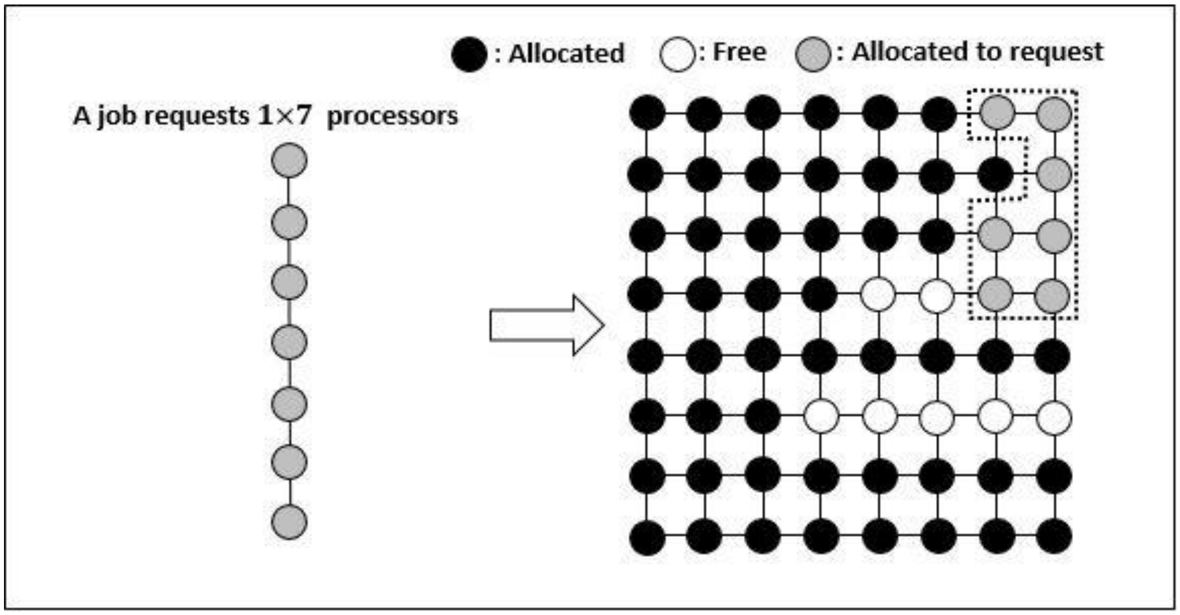


Figure 3.3: Allocating processors to a job requests a 1×7 submesh using the RBS.

For a large job request, if the number of free processors in the mesh is sufficient to accommodate the requested submesh size, it is allocated using one of the three different allocation methods, described below, according to the current allocation state of the mesh.

Initially, the strategy scans the mesh's rows by starting at the bottom row upwards and tries to find a block of free rows, $R(b,e)$, with a sufficient number of processors to accommodate the requested submesh size. Also, for each block of free rows, $R(b,e)$, in the mesh, the strategy records the number of free processors in the rows just above and beneath R(b,e), which are $R(e+1)$ and $R(b-1)$, respectively.

Method L.1: If there is a block of free rows $R(b,e)$ in the mesh with a sufficient number of processors to accommodate the requested size, then they are allocated to the requested job in an upwards row-major fashion, beginning at row $R(b)$, and the allocation is done.

Method L.2: If there is no a block of free rows $R(b,e)$ in the mesh with sufficient number of processors to accommodate the requested job size, then the proposed strategy checks if there is any block of free row(s) $R(b,e)$ with a number of free processors in ($R(b,e) + R(e+1) + R(b-1)$) greater than or equal to the requested submesh size, if such block is found then it starts at row $R(b-1)$ to allocate x rightmost free processors at that row, where x is evaluated as follows:

$x = Max\,(job\ size\ -\ (\,number\ of\ free\ processors\ in(\,R(b,e) + R(e+1))), 0),$

then it continues to the next upper row, $R(b)$, and allocates processors in an upwards row-major fashion and the allocation is done. If there are more than one such block in the mesh, then it chooses the one with the maximum free processors in $R(e+1)$.

Method L.3: If the above two methods failed to allocate the requested submesh size, then it starts at the bottom row upwards allocating the requested number of free processors in a row-major fashion and the allocation is done.

Method L.1 example: Assume that the mesh shown in Figure 3.4 and a job requests 20 processors, a 5×4 submesh. RBS scans the mesh rows, searching for a block of free rows which has a number of free processors greater than or equal to 20, since the free rows block, $R(4,6)$, has a number of free processors equal to 24 which is sufficient to accommodate the requested size. Then it starts at the beginning row,

$R(4)$, allocating the requested number of processors in a row major fashion.

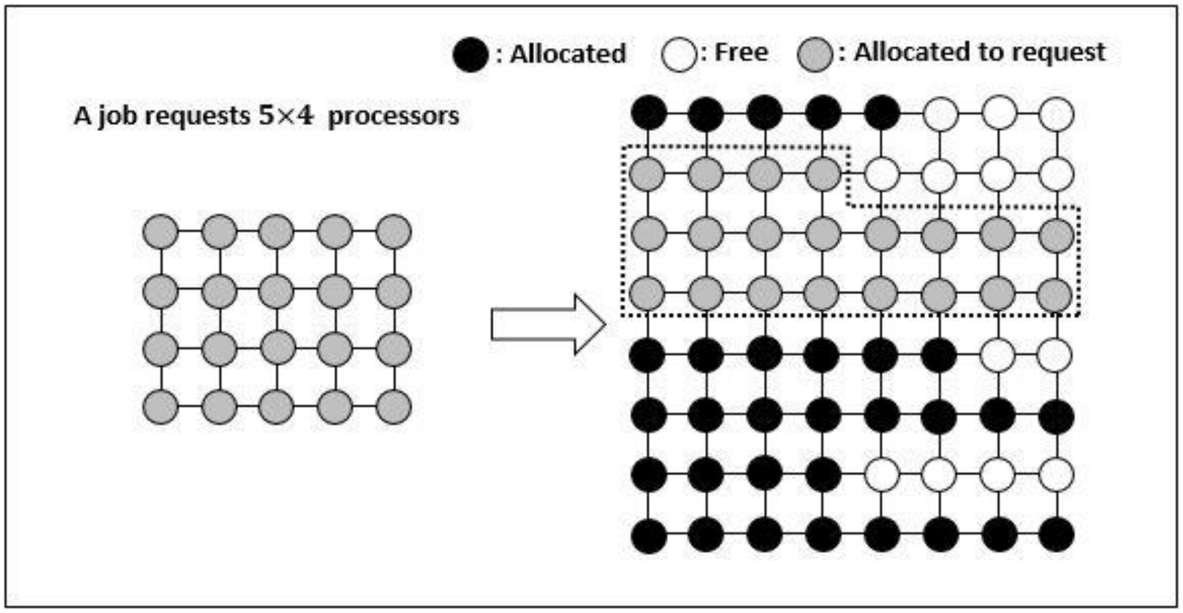


Figure 3.4 :Allocating processors to a job requests a 5×4 submesh using the RBS.

Method L.2 example: Assume that the mesh shown in Figure 3.5 and a job requests 28 processors, a 7×4 submesh. Since the mesh does not include any block of free rows with a sufficient number of processors to accommodate the requested job size, RBS checks if there is any block of free rows, $R(b,e)$, such that the number of free processors in $R(b,e) + R(b-1) + R(e+1)$ is sufficient to accommodate the requested submesh size, in this case it is $R(3,5)$, where the number of free processors in $R(3,5) + R(2) + R(6) = 24 + 4 + 3 = 31$, which is sufficient to accommodate the requested submesh size. therefore, the allocation here is by using method *L.2*, the allocation begins at $R(2)$ by allocating $Max\ ((28 - (24 + 3)), 0) = 1$ rightmost free

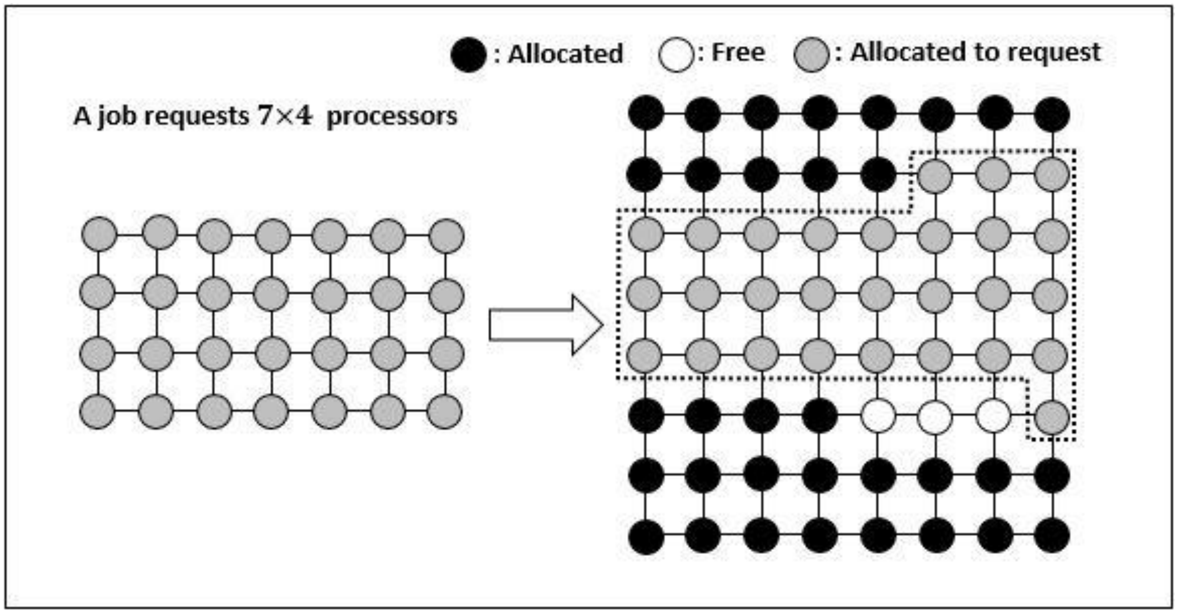


Figure 3.5: Allocating processors to a job requests a 7×4 submesh using the RBS.

Two extra examples that illustrate large job allocation using method *L.2* are depicted in Figures 3.6 and 3.7.

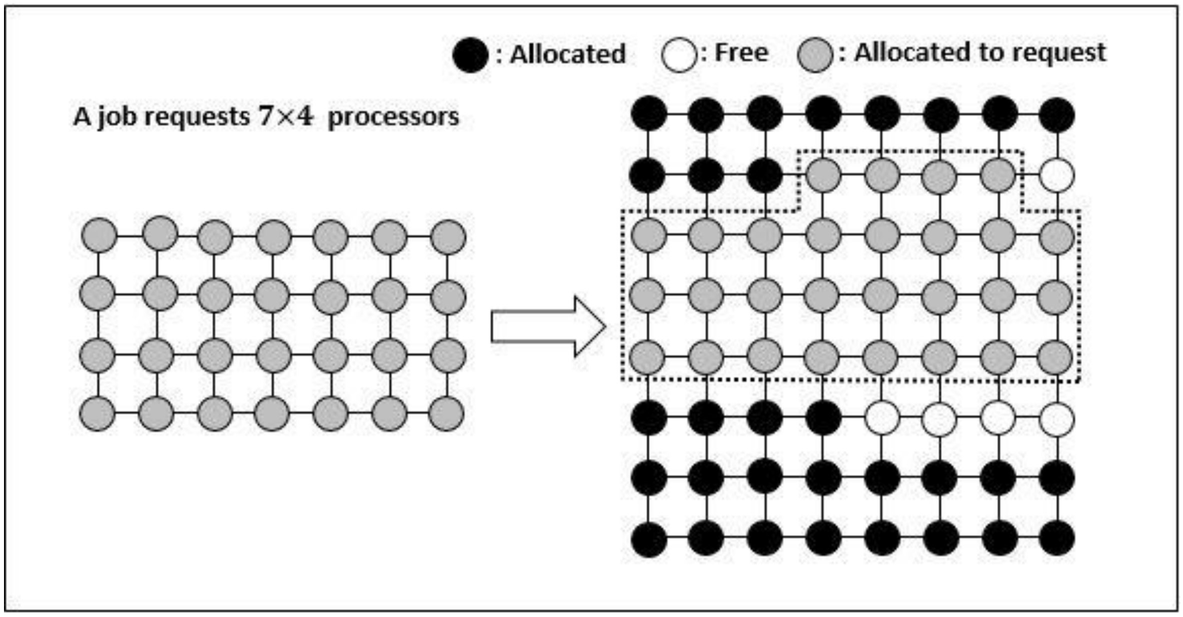


Figure 3.6: Allocating processors to a job requests a 7×4 submesh using the RBS.

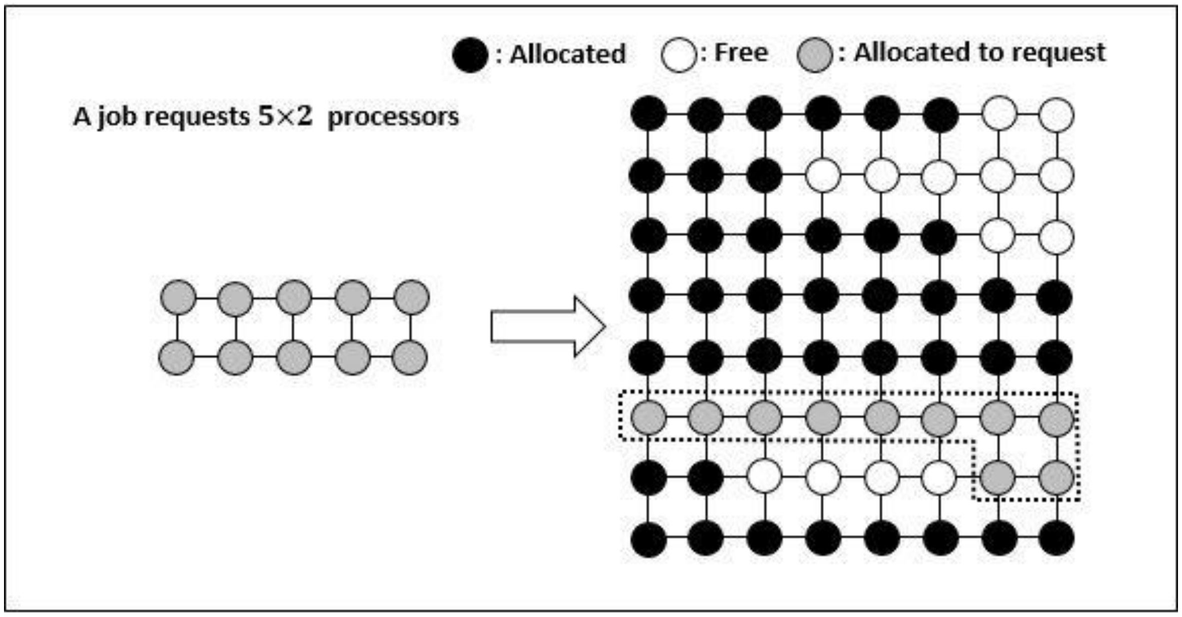


Figure 3.7: Allocating processors to a job requests a 5×2 submesh using the RBS.

Method L.3 example: Assume that the mesh shown in Figure 3.8 and a job requests 16 processors, an 8×2 submesh. Since the conditions of methods *L1* and *L2* do not match, method *L.3* is considered by starting at $R(0)$, allocating the requested number of processors in a row-major fashion

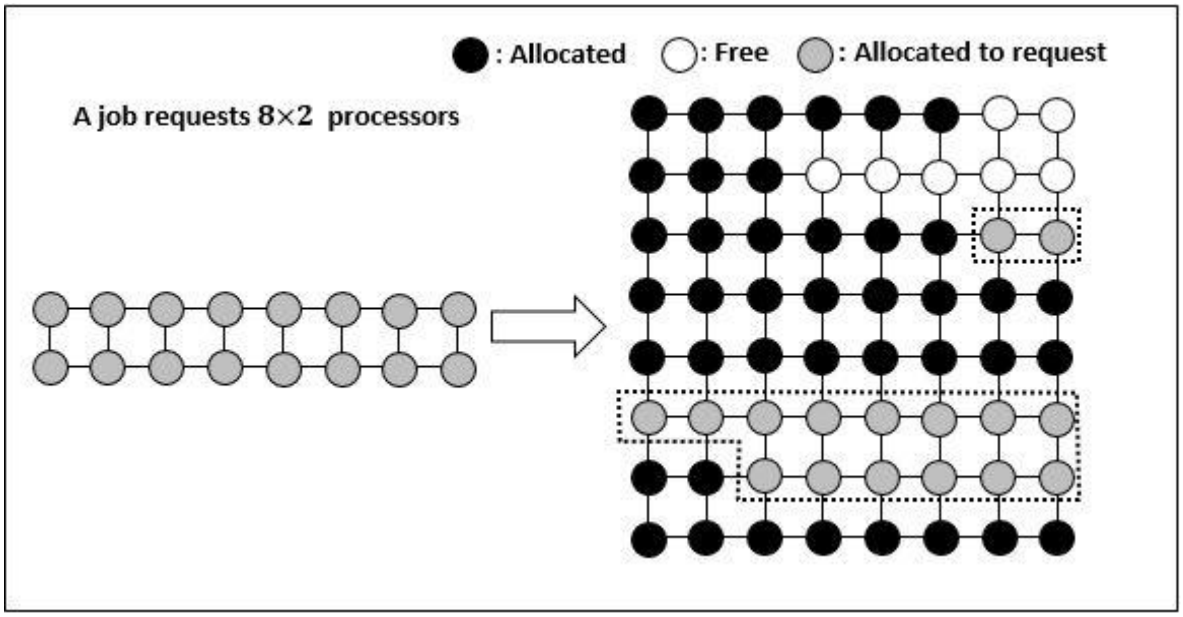


Figure 3.8: Allocating processors to a job requests a 5×2 submesh using the RBS.

```
Procedure RBS_Allocate(a,b):
{
Job_size = a×b.
Total_Allocated = 0.
r = H.
Step 1.   If (number of free processors<Job_Szie)
                   return failure.
Step 2.   If (Job_size > W)
                   go to Step 9.
Step 3.   r = r-1.
Step 4.   If (number of free processors in R(r)□ Job_size)
          {        allocate (Job_size) rightmost free processor in R(r).
                   return success.
          }
Step 5.   If (r>0)
                   go to Step 3.
Step 6.   r = H-1.
Step 7.   if (number of free processors in R(r) > 0)
          {        allocate rightmost free processor in R(r).
                   Total_allocated = Total_allocated+1.
          }
          else     r=r-1.
Step 8.   If (Total_Allocated = Job_Size)
                   return success.
          else
                   go to Step 7.
Step 9.   search the mesh rows from bottom row upwards for a block of free rows,
          R(b,e), such that number of free processors in R(b,e) □  Job_Size.
Step      if (a block of free rows R(b,e) is found, where the number of free processors in
10.       R(b,e) □ Job_size)
          {        allocate the requested number processors in a row-major fashion,
                   starting at R(b).
                   return success.
          }
Step      if (a block of free rows R(b,e) is found, where number of free processors in
11.       (R(b,e)+R(b-1)+R(e+1 )) □ Job_size. )
                   x=max (Job_size - number of free processors in (R(b,e) + R(e) ) , 0
          else     ).

                   go to Step 14.
Step      allocate (x) rightmost free processors in R(b-1).
12.
Step      allocate (Job_size - x) processors in a row-major fashion starting at R(b).
13.       return success.
Step      allocate the requested number of processors in a row-major fashion starting at
14.       R(0).
          return success.
} end procedure
```

Figure 3. 9: Outline of the RBS allocation algorithm.

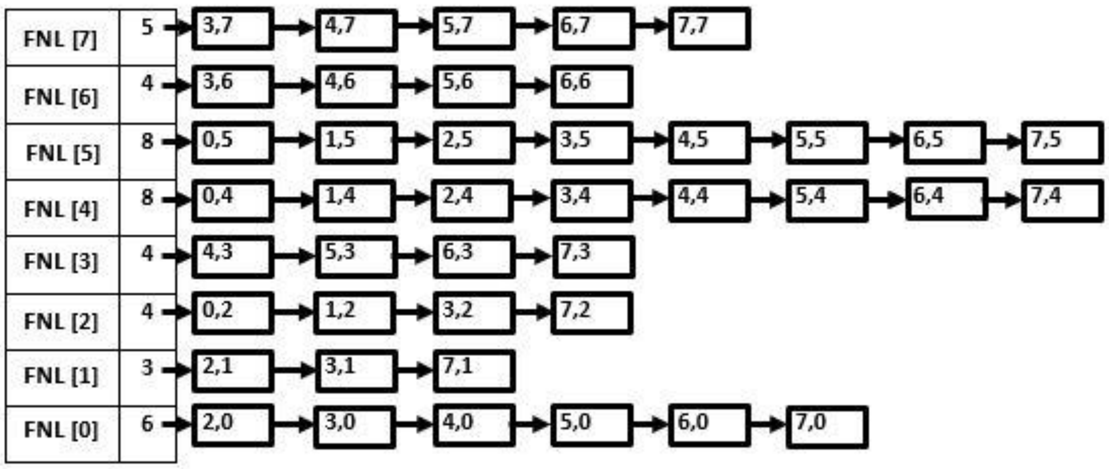
```
Procedure RBS_Deallocate(a,b):
{
  Job_id = id of the departing job.

  for each row, R(r), in mesh
      for each node in R(r)
          if (node's id = Job_id)
          {
              remove node's id.
              add node to freeNodeList[r].      //an array of ordered lists that keep
                                                track of all free nodes in each row.
          }
} end procedure
```

Figure 3.10: Outline of the RBS deallocation algorithm.

3.4 Complexity Analysis for RBS Allocation Strategy

RBS strategy maintains an array of ordered lists, *Free Node Lists (FNL)*, that keep track of all the unallocated nodes and their count for each row in the mesh. As example, assume the allocation state of the mesh shown in Figure 3.1, the corresponding array of the FNL is represented as shown in Figure 3.11.



FNL [7] 5 3,7 4,7 5,7 6,7 7,7
FNL [6] 4 3,6 4,6 5,6 6,6
FNL [5] 8 0,5 1,5 2,5 3,5 4,5 5,5 6,5 7,5
FNL [4] 8 0,4 1,4 2,4 3,4 4,4 5,4 6,4 7,4
FNL [3] 4 4,3 5,3 6,3 7,3
FNL [2] 4 0,2 1,2 3,2 7,2
FNL [1] 3 2,1 3,1 7,1
FNL [0] 6 2,0 3,0 4,0 5,0 6,0 7,0

Figure 3.11: An example of *Free Node Lists*.

3.4.1 The Allocation Time Complexity

The initial scanning operation only requires checking the *count* variable in the elements of the FNL array. In the worst case, it requires traversing all elements of the FNL array, where the size of the FNL array is equal to mesh height (H). Thus, the time complexity for scanning operation is $O(H)$.

The allocation operation for a job requests k processors involves removing k entry from r ordered lists of the FNL array, where r is the number of rows to be traversed to allocate the requested number of processors. Since removing an entry form the front or the back of an ordered list takes $O(1)$ time, thus the allocation time complexity of RBS is $O(r \times k)$. In the worst case (allocating a job of size $W \times H$), it is $O(H.W)$ or $O(N)$, where W, H, and N are the width, the height and the size of the mesh, respectively.

3.4.1 The Deallocation Time Complexity

The deallocation operation traverses all the nodes in every row in the mesh and compare the id of the allocated node with the id of the departing job. If the id is matched, then the *job_id* property of the node is reset and an entry containing the coordinates of the freed node is inserted into the corresponding ordered list in the FNL array (i.e., FNL[y- coordinate of the freed node]). The worst case in RBS occurs when deallocating a job of size $W \times H$. Since inserting an entry into an order list of size W takes $O(W)$ time complexity, then inserting W elements into the same list takes $O(W \times W)$. Repeating this operation in every ordered list in the FNL array would result in $O(H \times W \times W)$ or $O(W \times N)$ time complexity, where W, H, and N are the width, the height and the size of the mesh, respectively.

Chapter Four

Simulation Results

Extensive simulation experiments have been conducted to evaluate the performance of the proposed non-contiguous allocation strategy, Row Based Strategy (RBS), and compare it with the performance of the existing well-known non-contiguous allocation strategies Paging (Lo, et al., 1997), MBS (Lo, et al., 1997) and GABL (Bani-Mohammad, et al., 2007). The Paging and MBS allocation strategies have been chosen since they have been shown to perform well in (Lo, et al., 1997), and the same thing for GABL, as it has been shown to perform well in (Bani-Mohammad, et al., 2007; Bani-Mohammad, et al., 2010; Bani-Mohammad, et al., 2015). The performance of the contiguous First Fit (FF) (Zhu, 1992) allocation strategy has been included in the comparison as a representative of the contiguous allocation strategies since it has been shown an average performance in comparison with other allocation strategies in its class (Lo, et al., 1997). We have implemented the proposed allocation and deallocation algorithms, in the C programming language, and integrated the software into the ProcSimity well-known simulation tool for processor allocation and job scheduling in parallel systems (Windisch, et al., 1995; ProcSimity Manual, 1997).

The mesh system modeled in this research is a 2D square mesh with a side length L. System load is varied according to the frequency of job arrivals which is randomly modeled by an exponential distribution with a mean of average inter-arrival time value. System load is defined as the inverse of mean inter-arrival time. The jobs are served according to First-Come-First- Served (FCFS) scheduling policy. FCFS has been used in this research because it is fair and because we are intended to evaluate and compare the performance of the allocation strategies. The job execution time is the time needed by a job for completion starting from the time of allocation, where job execution time depends on the time needed for flits to be routed through the nodes, packet sizes, the number of messages to be sent, the message contention inside the network and the distances that the messages traverse (Bani-Mohammad, 2008). The side lengths of the sub-meshes requested by jobs are generated independently and follow a given probability distribution. As reported in Chapter 2, Section 2.5, two distributions have been considered in this research. The first is the uniform distribution over the range from 1 to the mesh side length L. The second is the uniform-decreasing distribution. It is determined by four probability $p1$, $p2$, $p3$, and $p4$, and four integers $l1$, $l2$, $l3$ and $l4$, where the probability that the width (length) of a request falls in the ranges [1,$l1$], [$l1+1$,$l2$], [$l2$+1,$l3$] and [$l3$+1,$l4$] is $p1$, $p2$, $p3$, and $p4$, respectively. The side lengths within a range are equally likely to occur. For the simulation experiments in this research work, $p1 = 0.4$, $p2 = 0.2$, $p3 = 0.2$, $p4 = 0.2$, $l1 = L/8$, $l2 = L/4$, $l3 = L/2$, and $l4 = L$. These distributions have often been used in the literature (Zhu, 1992; Lo, et al., 1997; Chang and Mohapatra, 1998; Chiu and Chen, 1999; Ababneh and Bani-Mohammad, 2003; Bani-Mohammad, et al., 2006; Bani-Mohammad, et al., 2010).

The interconnection network uses wormhole routing (Ni and McKinley, 1993; Mohapatra, 1998) along with dimension-order routing (XY routing) (Ni and McKinley, 1993; Mohapatra, 1998). Flits are assumed to take one time unit to move between two adjacent nodes, and t_s time units to be routed through a node. Packet sizes are represented by P_{len}. As previously reported in Chapter 2, Section 2.4, processors allocated to a parallel job communication with each other according to a given communication pattern. Four communication patterns have been considered in this research work. The first communication pattern is *one-to-all* (ProcSimity Manual, 1997), where a randomly selected process sends a message to each other processors allocated to the same job. The second communication pattern is *all-to-all* (ProcSimity Manual, 1997), where each processor in a job sends a message to all other processors allocated to the same job. This communication pattern causes much message contention and is considered as the weak point of the non-contiguous allocation algorithms (Suzaki, et al., 1996). The third communication pattern is *random* (ProcSimity Manual, 1997), where a message is sent between a randomly selected pair of processors (source and destination) within the same job. In the fourth communication pattern, *near-neighbor* (Bani-Mohammad and Ababneh, 2013), each processor communicates with its neighbors. The number of messages that are generated by a job is correlated to the job size in the one-to-all, all-to-all and near-neighbor communication patterns, since each job does exactly one iteration of the given communication pattern, and it is only one message per job in the random communication pattern.

The performance figures presented in the following sections in this chapter adopt the following parameters: the mesh size is a 16×16, t_s **=** 3 time units, P_{len} = 8 flits. Simulation parameters are illustrated in Table 4.1. It is worth noting that most of the values of these parameters have been adopted in the literature (Zhu, 1992; Babbar and Krueger, 1994; Suzaki, et al., 1996; Lo, et al., 1997; Wu, et al., 2003; Bani-Mohammad, et al., 2006; Bani-Mohammad, et al., 2010) and have been recommended in (ProcSimity Manual, 1997).

Table 4. 1: The System Parameters used in the Simulation Experiments.

Simulation Parameter	Value
Dimensions of the Mesh	16×16
Packet Length	8 flits
Flow Control Mechanism	Wormhole Routing
Routing Delay	3 time units
Router Type	Mesh XY Routing
Allocation Strategy	RBS, GABL, MBS, Paging(0), and FF
Scheduling Strategy	FCFS
Job Size Distribution	Uniform: Job widths and lengths are uniformly distributed over the range from 1 to the mesh side lengths L.
	Uniform Decreasing: Represents the case where most jobs are small relative to the size of the system.
Inter-arrival Time	Exponential with different values for mean. The values are determined through experimentation with the simulator, ranged from lower values to higher values.
Mean Time between Sends	0.0

Communication Pattern	One-to-all, all-to-all, Random, and Near Neighbor.
Messages per job	Messages per job are correlated to the job size, since each job does exactly one iteration of the given communication pattern, except for Random communication pattern, where the number of messages per job is only one.
Number of Runs	The number of runs should be enough so that the confidence level is 95% and the relative errors are below 5% of the means. The number of runs ranged from dozens to thousands.
Number of Jobs per Run	1000

Each simulation run consists of 1000 completed jobs. Simulation experiments are repeated for independent runs until the confidence level reaches 95% and the relative errors do not exceed 5%.

The main performance parameters used are the *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. The *system utilization* is the percentage of processors that are utilized over a given period of time. The important independent variable in the simulation is the system load. It is defined as the inverse of the mean inter-arrival time of jobs. Its range of values 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 the results of the performance metric of interest (Bani-Mohammad, 2008).

4.1 Turnaround Time

In Figures 4.1 and 4.2, the average turnaround times of jobs are plotted against the system load for the one-to-all communication pattern. The results reveal that in most cases, the performance of RBS is relatively better than that of the other non-contiguous allocation strategies considered in this research, and they are all substantially superior to the FF contiguous allocation strategy for both job distributions considered in this research. This is because that the non-contiguous allocation strategies considered in this research eliminate both internal and external fragmentation, hence, they achieve better system utilization and that can notably improve the system performance in terms of jobs turnaround times, where this improvement in system utilization outbalanced the impact of the external message contention encountered in non-contiguous allocation. For example, in Figure 4.1, the performance of RBS is almost the same as Paging(0), barely 1% in favor for GABL, and about 2% in favor for RBS compared to MBS, under the job arrival rate of 0.0009 jobs/time unit. However, the performance difference is very clear when comparing with the contiguous FF strategy as it reaches to 54% in favor for the non-contiguous RBS strategy, under the job arrival rate of 0.0009 jobs/time unit.

Although the average turnaround times of all non-contiguous and contiguous allocation strategies are improved when uniform decreasing distribution is used, the relative performance remains almost the same as when the uniform distribution is used.

This improvement in turnaround times is due to the increased probability of small jobs to be allocated. Moreover, for non-contiguous allocation strategies, the message contention decreased since, in the one-to-all communication pattern, the number of messages for a job is correlated to the job size. For example, in Figure 4.2, the performance of RBS is almost the same as MBS, and the relative difference in performance in favor for RBS are 1%, 2%, and 51%, compared to GABL, Paging(0), and FF, respectively, under the job arrival rate of 0.005 jobs/time unit.

Although in one-to-all communication pattern, the number of messages is considerable, but, the contention produced here, due to the relatively small packet size (8 flits) used, does not distinguish the superior contention alleviation feature of RBS, which is more notable when the all-to-all communication pattern is used.

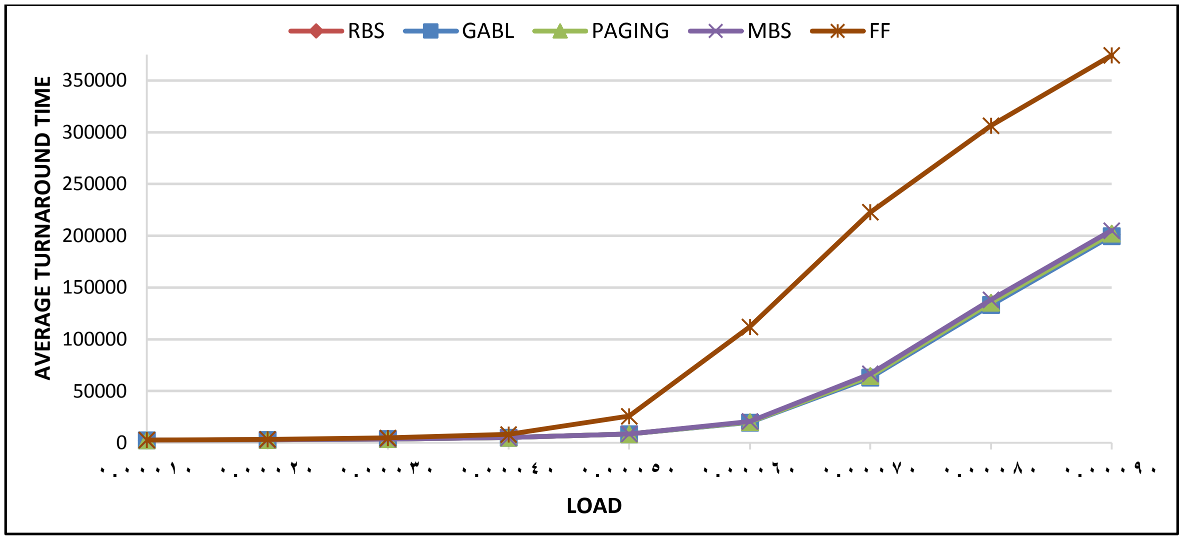


Figure 4.1: Average turnaround time vs. system load for the one-to-all communication pattern and uniform side lengths distribution in a 16×16 mesh.

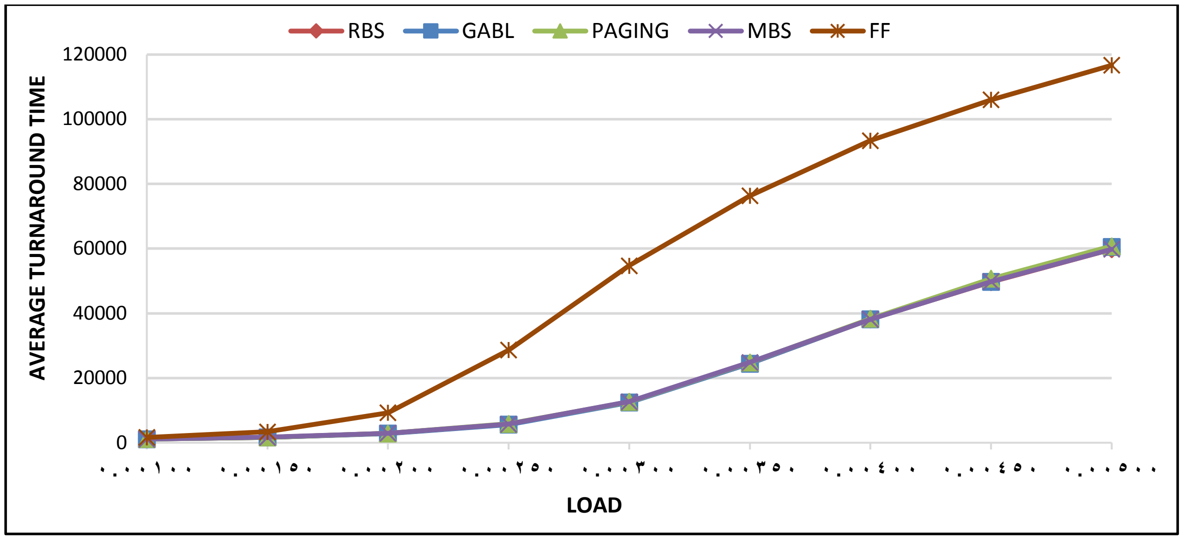


Figure 4.2: Average turnaround time vs. system load for the one-to-all communication pattern and uniform decreasing side lengths distribution in a 16×16 mesh.

In Figures 4.3 and 4.4, the average turnaround times of jobs are plotted against the system load for the all-to-all communication pattern. The results reveal that RBS performs much better than all other contiguous and non-contiguous allocation strategies for both job size distributions considered in this research. This is because RBS is better than the previous non-contiguous allocation strategies at alleviating message contention. In Figure 4.3, for example, the average turnaround times of RBS are 72%, 60%, 31%, and 54% of the average turnaround times of GABL, Paging(0), MBS, and FF, respectively, under the job arrival rate of 0.00009 jobs/time units. Again, as seen in one-to-all, the average turnaround times for all allocation strategies are improved when uniform decreasing distribution is used, however, the relative performance of the allocation strategies remains almost the same for both job size distributions. In Figure 4.4, for example, the average turnaround times of

RBS are 70%, 77%, 52%, and 62% of the average turnaround times of GABL, Paging(0), MBS, and FF, respectively, under the job arrival rate of 0.0005 jobs/time units.

It is worth noting that the FF contiguous allocation strategy substantially outperforms the non-contiguous allocation MBS strategy for uniform side lengths distribution and performs better than it for uniform decreasing distribution. This is because that all-to-all communication pattern produces much message contention and considered as the weak point of the non-contiguous allocation strategies (Suzaki, et al., 1996), where the number of messages per job increases dramatically as the job size increases. If the message contention increases significantly, this would increase the delay, and defeat the gain of the improved system utilization; and consequently, degrades the system performance in terms of jobs turnaround time (Min and Mutka, 1994; Mache and Lo, 1997). This is the case here with MBS, because its main drawback, as previously discussed in Chapter 2, section 2.1.2, is that the submesh allocation is restricted to a base 4 square blocks, therefore, it may fail to allocate a requested submesh contiguously even if a one exist, and may unnecessarily divide a submesh request and allocate the parts far apart of each other, especially for large jobs, and this can seriously increase the message contention.

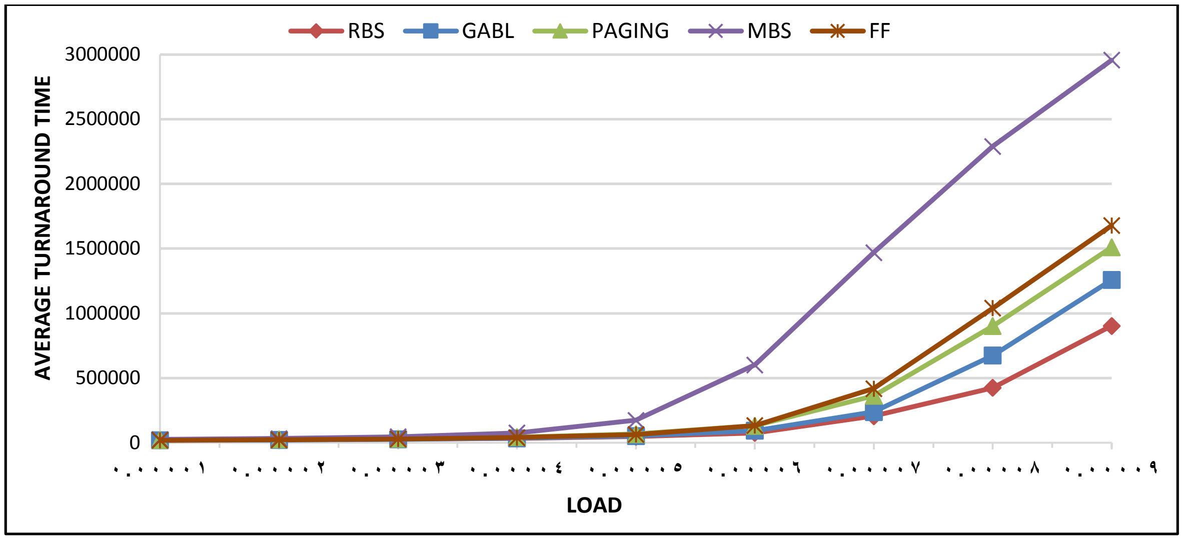


Figure 4.3: Average turnaround time vs. system load for the all-to-all communication pattern and uniform job side lengths distribution in a 16×16 mesh.

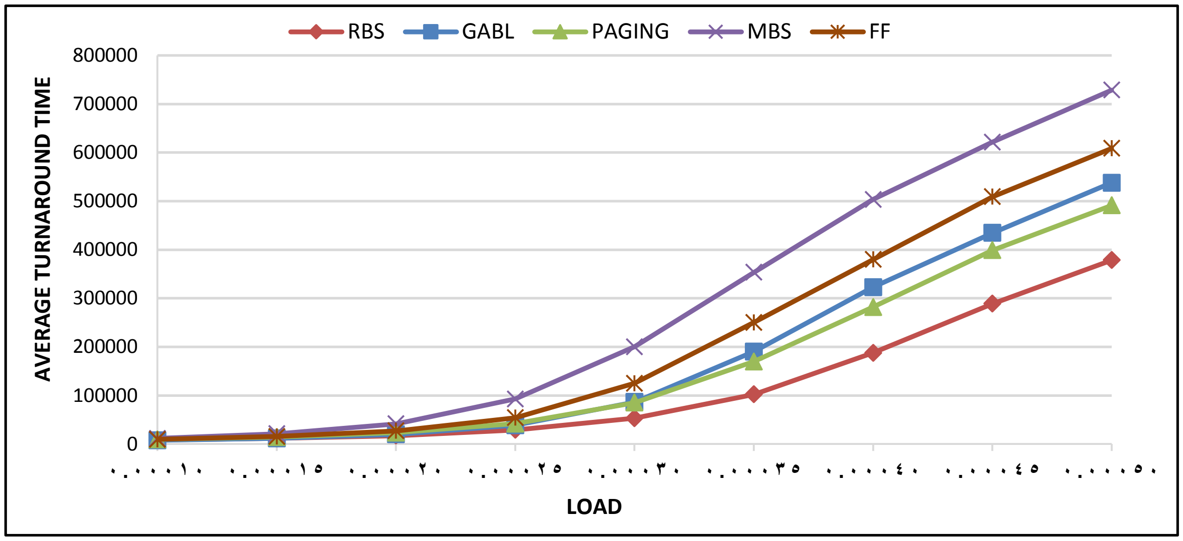


Figure 4.4: Average turnaround time vs. system load for the all-to-all communication pattern and uniform decreasing side lengths distribution in a 16×16 mesh.

In Figures 4.5 and 4.6, the average turnaround times are plotted against the system load for the random communication pattern. The results reveal that in most cases, the performance of RBS is relatively better than that of the other non-contiguous strategies and they are all outperform the FF contiguous allocation strategy. In Figure 4.5, for example, when the job arrival rate is 0.1 jobs/time unit, the relative difference in turnaround times between RBS and GABL is 1% in favor for GABL, and are 3%, 2% and 36% in favor for RBS compared to Paging(0), MBS, and FF, respectively. Figure 4.6 shows a slight relative performance improvement for RBS when the uniform decreasing distribution is used. This is because of the increased probability of small jobs (relative to mesh size) when using this distribution, and since RBS, generally, allocates the jobs along the rows of the mesh, and relatively small jobs can be laid out in a less number of lines, which decreases the contention among different jobs' messages. Moreover, RBS has the ability to allocate jobs that are smaller than or equal to the mesh width in a way that reduces the contention among different small jobs. For example, when the job arrival rate is 0.25, the relative differences in job turnaround times in favor for RBS are 9%, 7%, 4% and 51%, compared to GABL, Paging(0), MBS, and FF, respectively.

Random communication pattern can only give a glance about the ability of the non-contiguous allocation strategies to alleviate the message contention. However, the contention produced when adopting the random communication pattern is not sufficient to distinguish among the non-contiguous allocation strategies. This is because for each job, a one message is sent from a randomly selected source node to another randomly selected destination node within the same job.

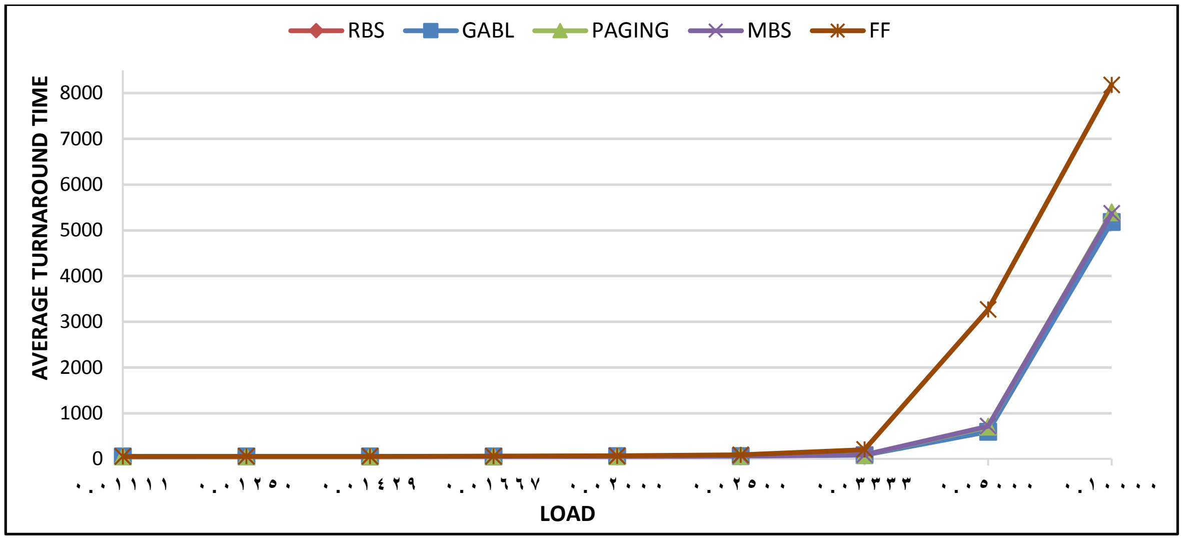


Figure 4.5: Average turnaround time vs. system load for the random communication pattern and uniform job side lengths distribution in a 16×16 mesh.

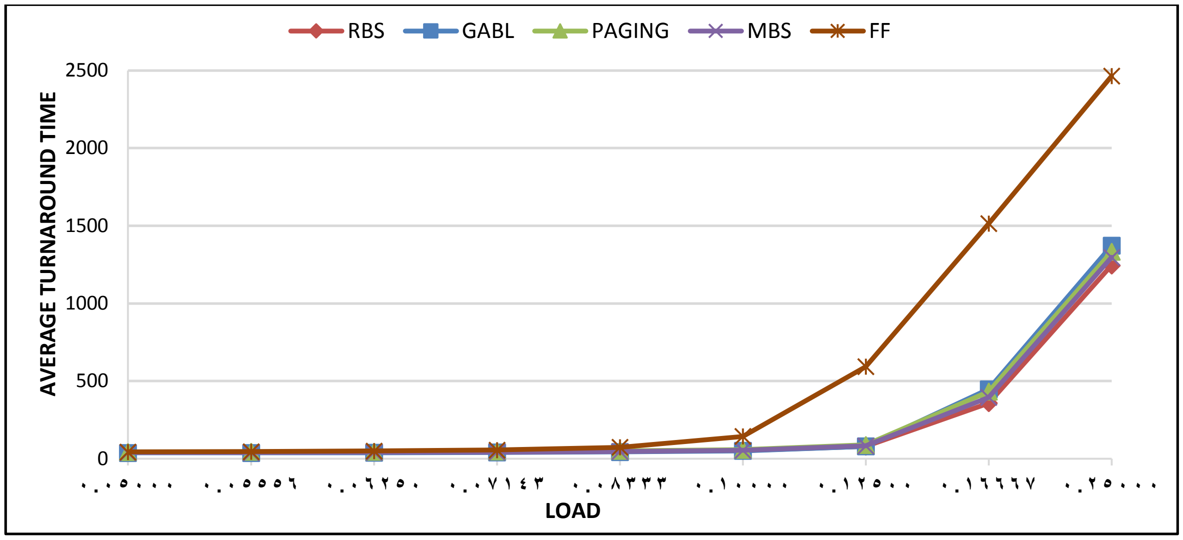


Figure 4.6: Average turnaround time vs. system load for the random communication pattern and uniform decreasing side lengths distribution in a 16×16 mesh.

In Figures 4.7 and 4.8, the average turnaround times are plotted against the system load for the near neighbor communication pattern. The performance of RBS is not better than that of the other contiguous and non-contiguous allocation strategies, however, in Figure 4.7, its performance is very close to the performance of Paging(0) and MBS, in Figure 4.8, it is very close to the performance of Paging(0). This is because in this communication pattern, each node allocated to a job communicates with its neighbors (left, right, up, down) that are allocated to the same job, and this is suitable for the strategies that maintain a high degree of contiguity among the allocated processors for a given job and form rectangular shapes for the allocated submeshes. Figure 4.7 shows that the FF contiguous allocation strategy substantially outperforms all non-contiguous allocation strategies.

This is because, in FF, the allocated submeshes for jobs are contiguous and form rectangular shapes, therefore, no external contention is encountered here. In addition, it is notable that the non-contiguous GABL allocation strategy performs better than other non-contiguous allocation strategies since it combines the desirable features of both contiguous and non-contiguous allocation while it allocates submeshes in a rectangular form and tries to maintain a high degree of contiguity among the processors in the allocated submeshes. The same relative performance can be seen in Figure 4.8 when the uniform decreasing distribution is used, however, the relative performance differences are less severe.

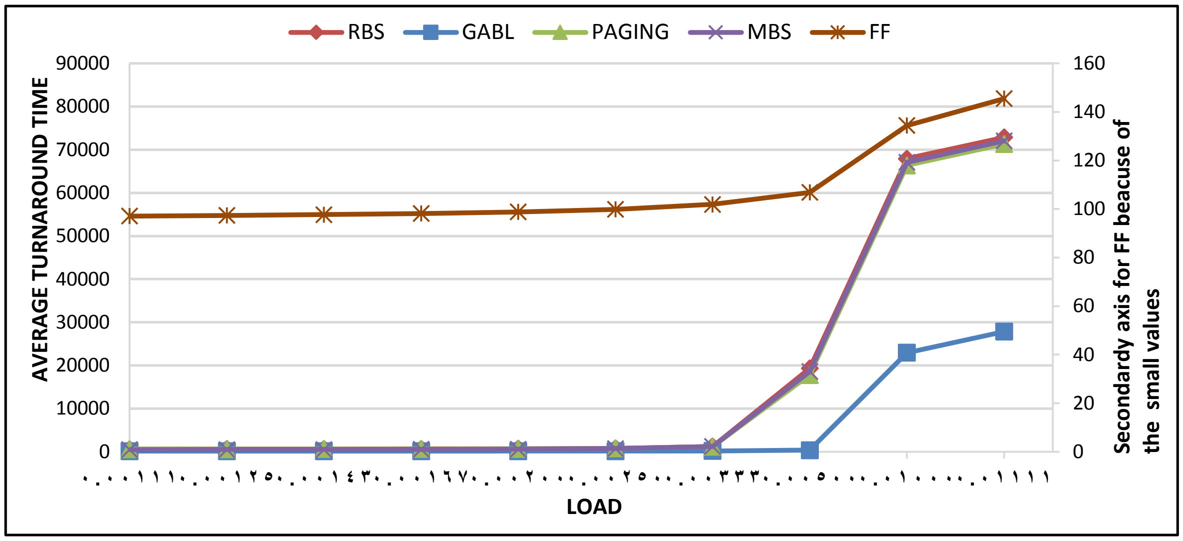


Figure 4.7: Average turnaround time vs. system load for the near neighbor communication pattern and uniform side lengths distribution in a 16×16 mesh.

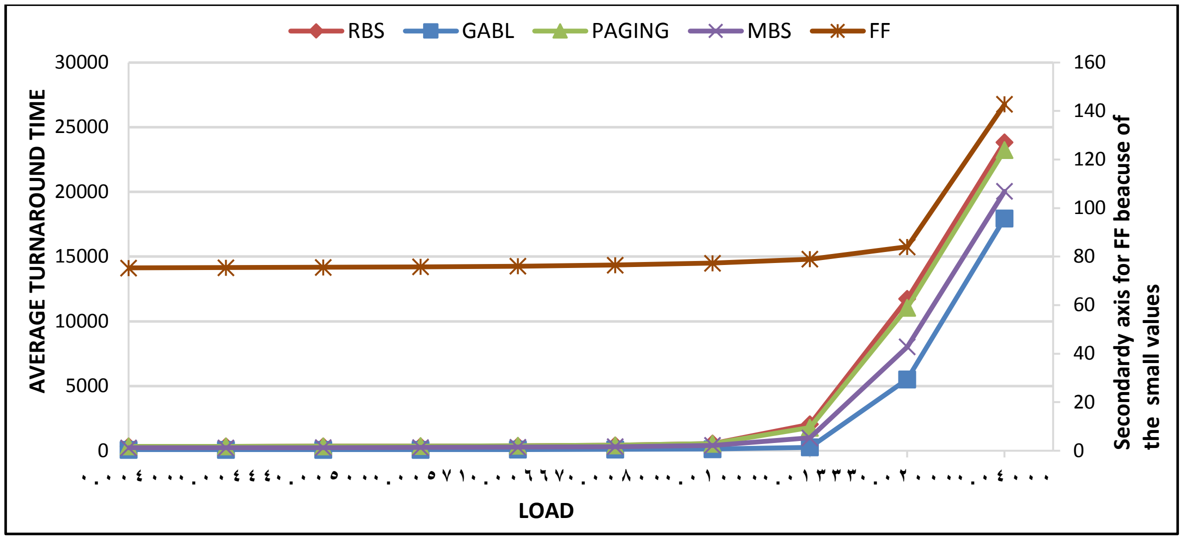


Figure 4.8: Average turnaround time vs. system load for the near neighbor communication pattern and uniform decreasing side lengths distribution in a 16×16 mesh.

4.2 System Utilization

Figures (4.9-4.18) depict the mean system utilization of the investigated allocation strategies (RBS, GABL, Paging(0), MBS, FF) for the four communication patterns and the two job size distributions considered in this research work. The load values ranged from moderate to heavy loads, where heavy loads cause the waiting queue to be filled very early which allows the allocation strategies to reach its upper system utilization limit. The non-contiguous allocation strategies achieve a mean system utilization of 76% to 78%, and 81% to 85%, for uniform and uniform decreasing job size distributions, respectively, while the contiguous FF strategy cannot exceed 63% utilization for both job size distribution.

This is because contiguous allocation causes high external fragmentation since the allocation of a requested submesh requires contiguity among its processors and a shape that resembles the connected network topology; these conditions reduce the chance of successful allocation and consequently reduce the mean system utilization. At heavy load values, the mean system utilization for the non-contiguous allocation strategies are approximately the same for both job size distributions. This is because the non-contiguous allocation strategies have the same ability to eliminate internal and external processor fragmentation. They always succeed to allocate processors to a requested job if there are enough free processors.

It is worth noting that a high mean system utilization rate for an allocation strategy at a given load value may be caused by high message contention which increases the communication delay and makes the jobs to stay a longer time in the system

. As an example, in all-to-all communication pattern, MBS has recorded the highest mean system utilization at moderate system loads, while in the corresponding turnaround time it is the worst.

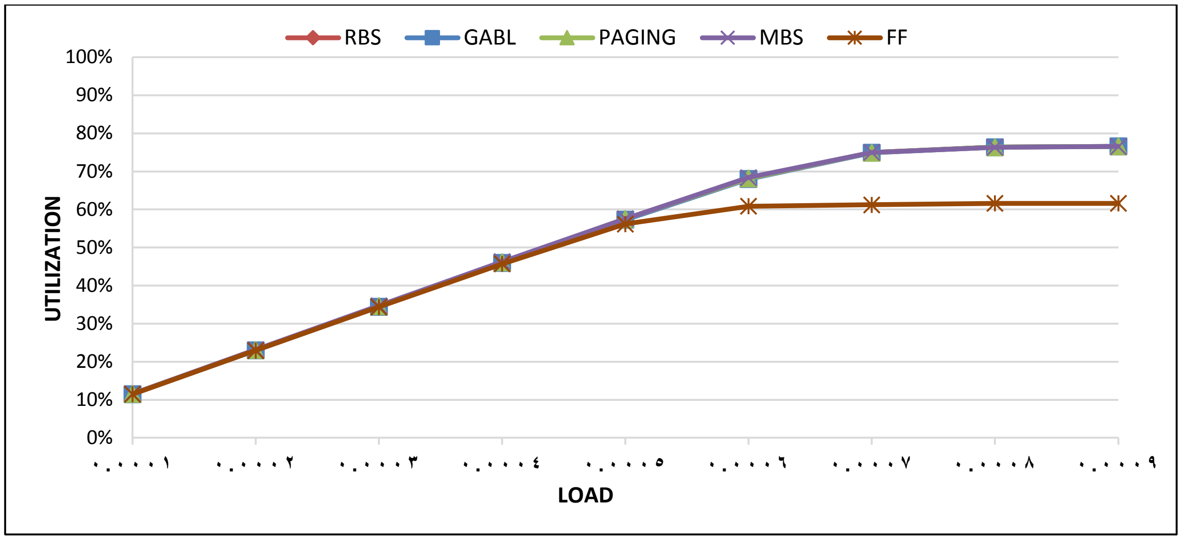


Figure 4.9: Mean system utilization vs. system load for the one-to-all communication pattern and uniform job side lengths distribution in a 16×16 mesh.

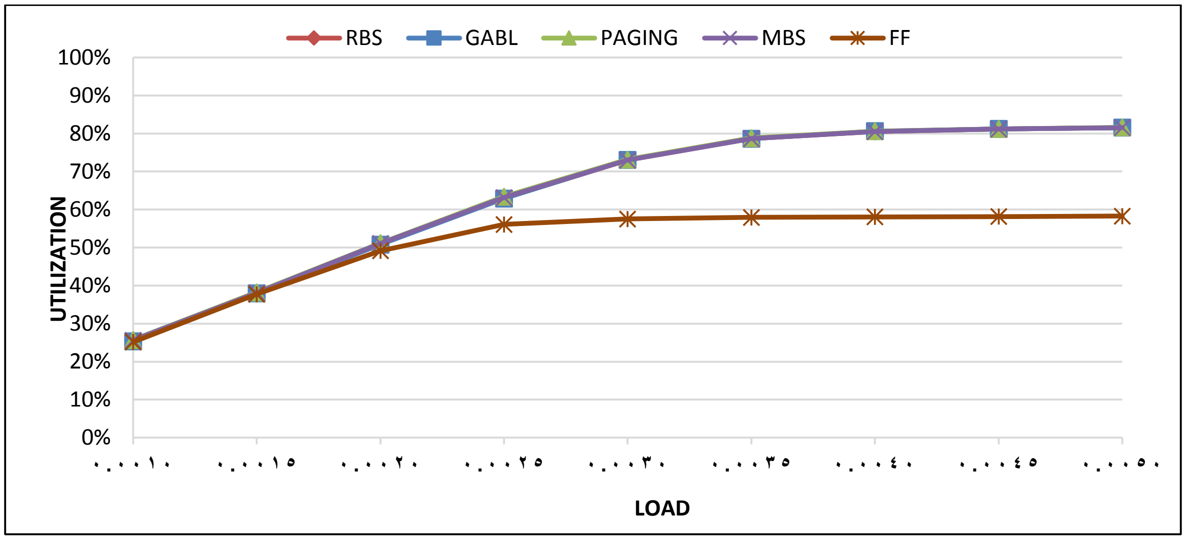


Figure 4.10 Mean system utilization vs. system load for the one-to-all communication pattern and uniform decreasing job side lengths distribution in a 16×16 mesh.

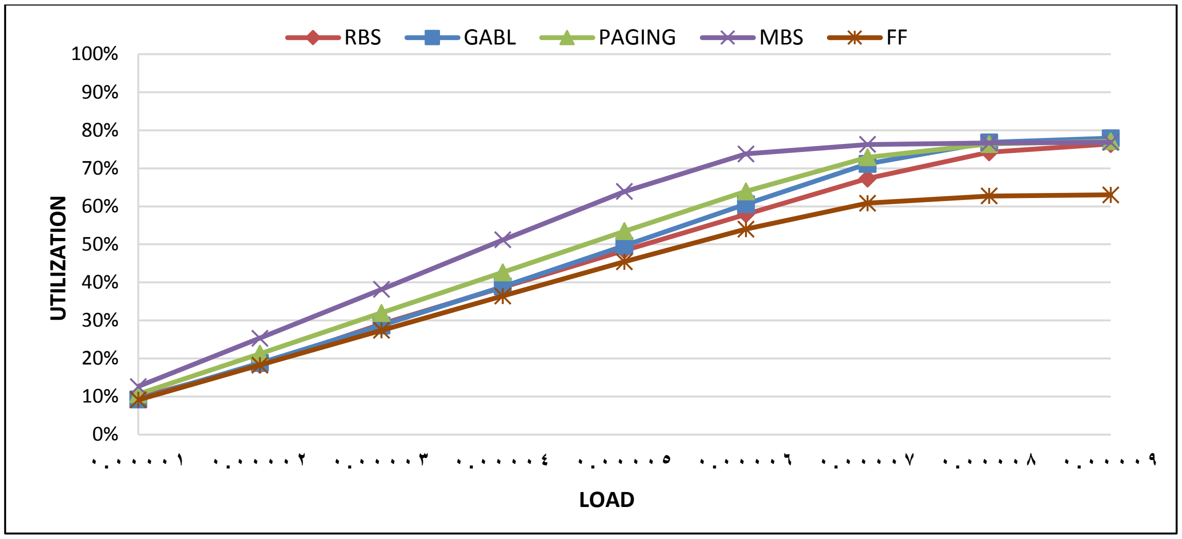


Figure 4.11: Mean system utilization vs. system load for the all-to-all communication pattern and uniform job side lengths distribution in a 16×16 mesh.

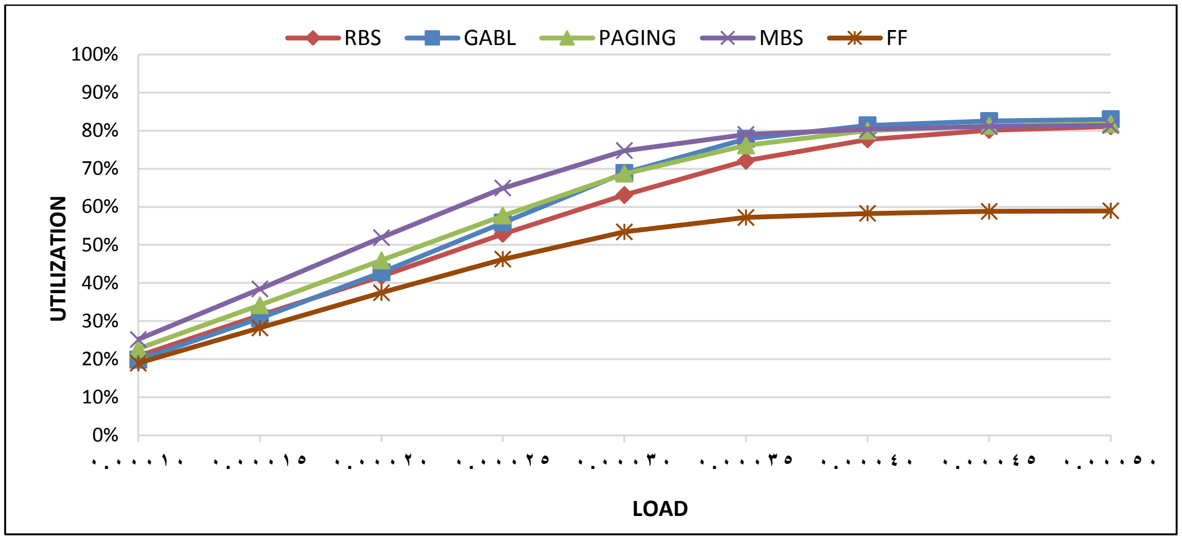


Figure 4.12: Mean system utilization vs. system load for the all-to-all communication pattern and uniform decreasing job side lengths distribution in a 16×16 mesh.

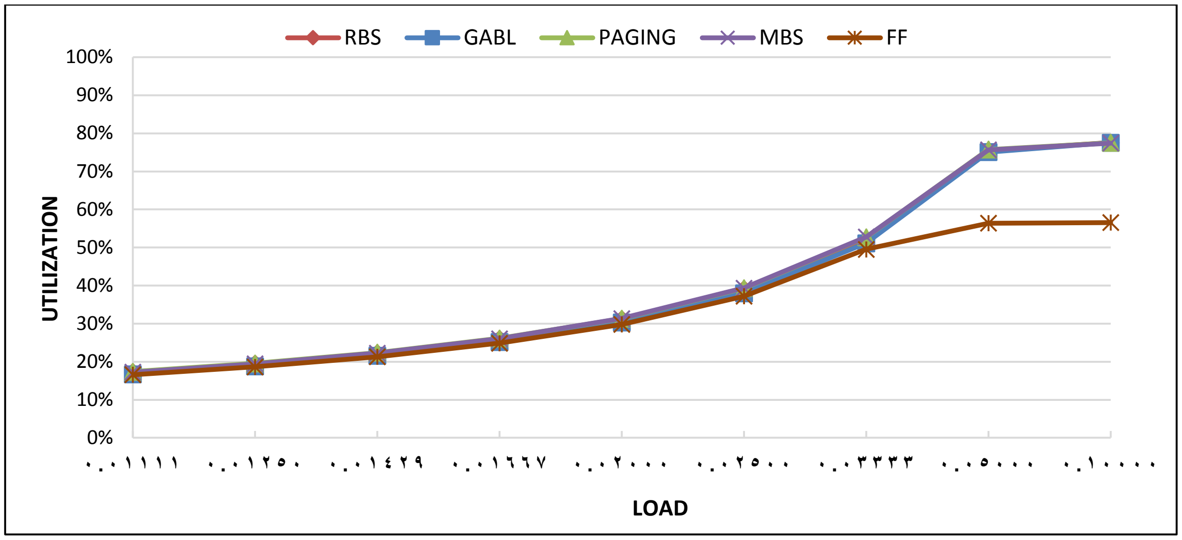


Figure 4.13: Mean system utilization vs. system load for the random communication pattern and uniform decreasing job side lengths distribution in a 16×16 mesh.

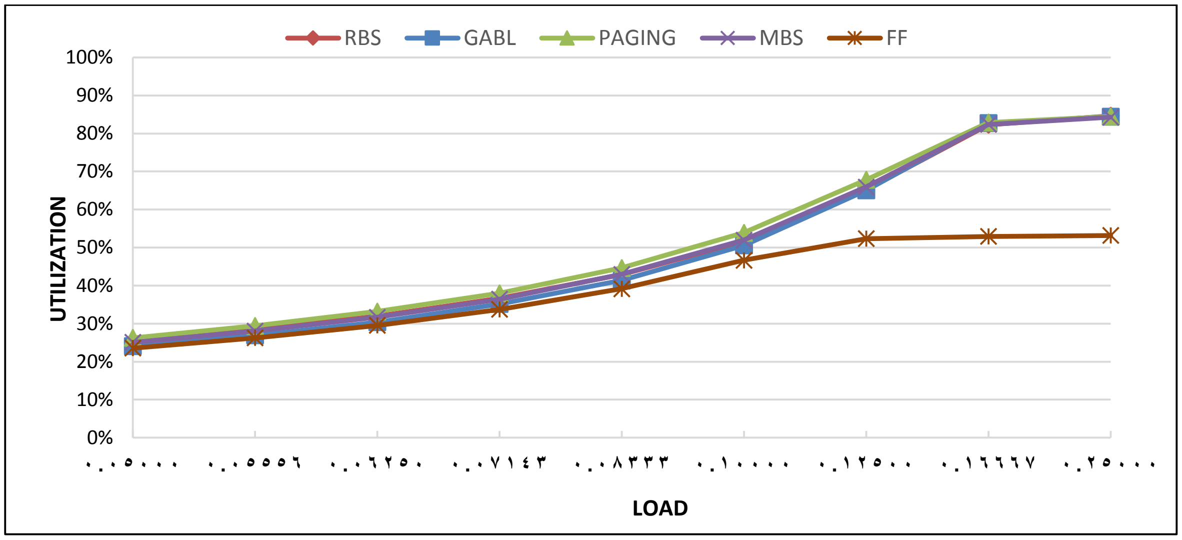


Figure 4.14: Mean system utilization vs. system load for the random communication pattern and uniform decreasing job side lengths distribution in a 16×16 mesh.

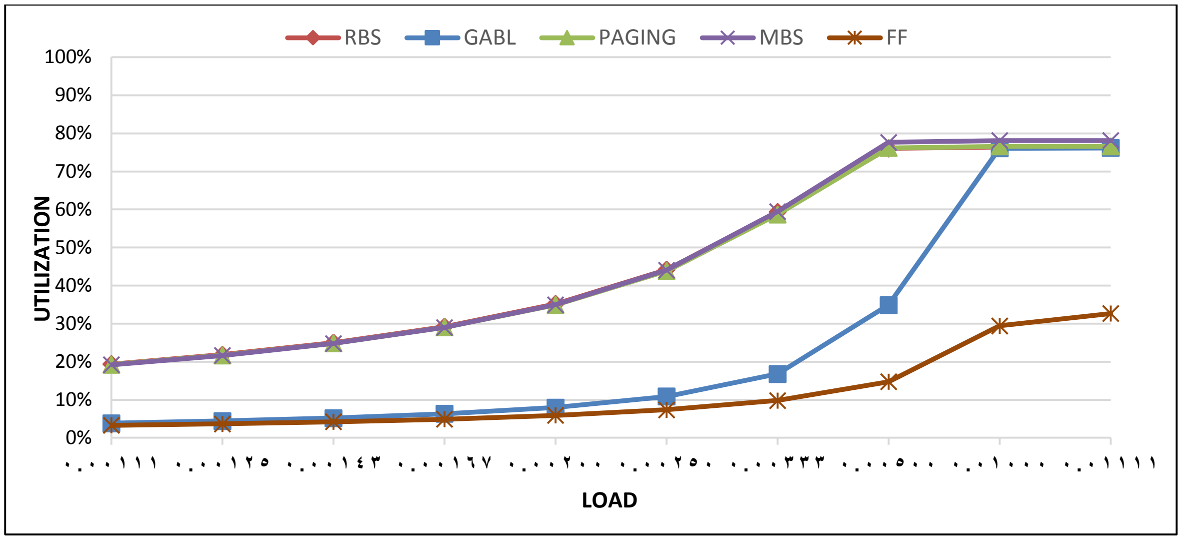


Figure 4.15: Mean system utilization vs. system load for the near neighbor communication pattern and uniform decreasing job side lengths distribution in a 16 × 16 mesh.

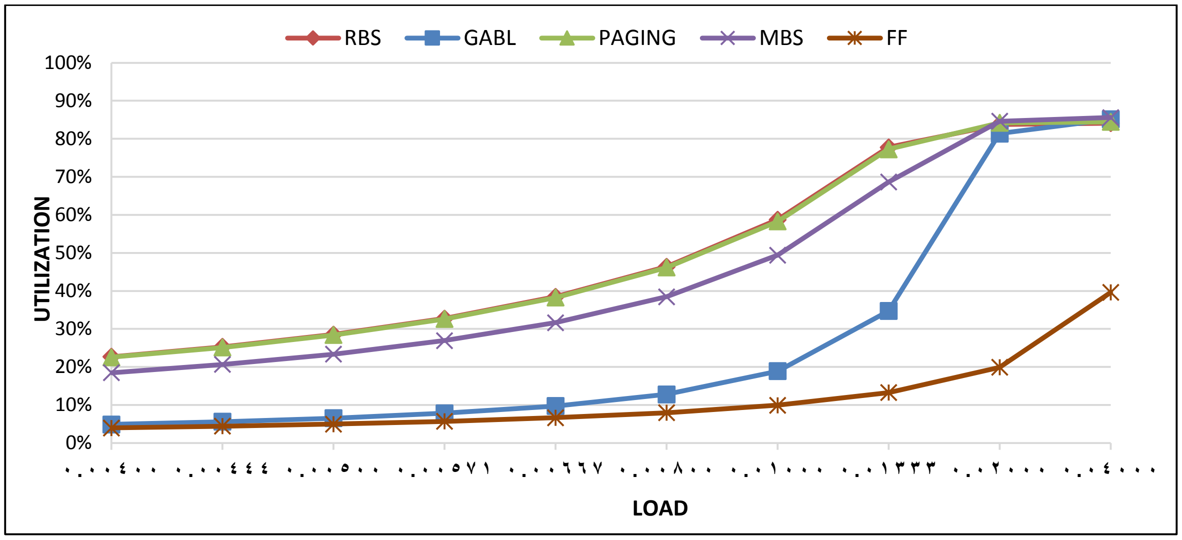


Figure 4.16: Mean system utilization vs. system load for the near neighbor communication pattern and uniform decreasing job side lengths distribution in a 16 × 16 mesh.

4.3 Conclusion

This chapter has investigated the performance merits of the non-contiguous allocation in the 2D mesh network. To this end, we have suggested a new non-contiguous allocation strategy, referred to as Row Based Strategy (RBS for short), which differs from the previous non-contiguous allocation strategies in the method used to allocate different submeshes for the job requests according to their sizes. RBS classifies the incoming job requests according to their sizes into two categories: *large* and *small*, where a job is considered large if the number of requested processors is greater than the mesh width, otherwise, it is considered small. The main goal of this classification is to reduce the contention among messages of different jobs by minimizing the number of processors allocated to a large job in the rows which already contain processors allocated to another large job. Also, it tries to allocate the small jobs in the upper part of the mesh, knowing that the messages of two small jobs allocated next to each other in the same row would not collide.

The performance of RBS has been compared against that of the existing non-contiguous and contiguous allocation strategies. Simulation results have shown that RBS can significantly improve the performance despite the external contention caused by interference among messages of different jobs. RBS also achieves efficient system utilization compared to the contiguous strategies, due to its ability to eliminate internal and external processor fragmentation.

The results have also revealed that, RBS is superior to the previous well known non-contiguous allocation strategies, such as GABL, Paging(0), and MBS in terms of turnaround time for

the all-to-all communication pattern, which is considered as the weak point of the non-contiguous allocation strategies. This is because all-to-all communication pattern produces intensive messages and hence increases the message contention and consequently increases the communication delay. The RBS superiority here is due to its merit at alleviating the message contention inside the network, which can significantly improve the average turnaround times of the jobs.

The results have also shown that the performance of RBS is relatively better than that of the previous non-contiguous allocation strategies for one-to-all and random communication patterns in most cases. However, it is not better than that of the other contiguous and non-contiguous strategies when the near neighbor communication pattern is used, because the privilege in this communication pattern is for the strategies that maintain a high degree of contiguity and maintain a rectangular shape of the allocated submeshes.

Chapter Five

Conclusion and Future Work

5.1 Conclusion

Parallel computers have been considered as one of the most powerful computing platforms that support various types of large and complex applications in fields such as engineering, sciences, and many others. Distributed-memory multicomputers are an important class of parallel computers as they offer a cost-effective alternative of traditional supercomputers (Foster, 1995; Kumar, et al., 2003). Many topologies have been suggested for the multicomputer networks, yet, the mesh topology has gained much popularity, because of its simplicity, regularity, scalability, and partition-ability. Moreover, many applications can be mapped very naturally into the mesh topology, such as matrix computation, image processing, and many other practical applications (Babbar and Krueger, 1994; Foster, 1995; Das Sharma and Pradhan, 1996; Chang and Mohapatra, 1998; Yoo and Das, 2002; Kumar, et al., 2003). Mesh topology has been adopted in many commercial and experimental multicomputers. The Intel Paragon (Intel Corporation, 1991), the Delta Touchstone (Intel Corporation, 1991), and the iWARP (Peterson, et al., 1991) are examples of 2D mesh-connected multicomputers. Examples of 3D mesh-connected multicomputers include the MIT J-machine (Noakes, et al.), the IBM blueGene/L (Blumrich, et al., 2003), and the Cray XT3 (Cray, 2005).

Many research studies have been investigated the processor allocation in distributed-memory multicomputers, especially those based on mesh network (Li and Cheng, 1991; Zhu, 1992; Chuang and Tzeng, 1994; Das Sharma and Pradhan, 1996; Lo, et al, 1997; Chang and Mohapatra, 1998; Ababneh, 2001; Wu, et al., 2003; Moghaddam and Naghibzadeh, 2006; Bani-Mohammad, et al., 2007; Ababneh, 2008; Ababneh, et al., 2010; Bani-Mohammad, et al., 2010; Bani-Mohammad, et al., 2015; Bani-Mohammad, 2017). Processor allocation strategies can be mainly classified into two groups: contiguous and non-contiguous. In contiguous allocation strategies (Li and Cheng, 1991; Zhu, 1992; Chuang and Tzeng, 1994; Das Sharma and Pradhan, 1996; Ababneh, 2001; Ababneh, et al., 2010), the allocated processors must be physically contiguous and resemble the shape of the underlying network. The main goal of this type of allocation is to alleviate the external message contention, since only messages of the same job are expected within an allocated submesh, and to decrease the distances among the processors allocated to the same job. As a consequence of these allocation limitations, inefficient system utilization is expected due to the high processor fragmentation. Processor fragmentation can be classified into two types: internal and external (Das Sharma and Pradhan, 1996; Lo, et al., 1997; Chang and Mohapatra, 1998; Seo, 2005). Internal fragmentation occurs when a requested job is allocated more processors than it is requested, the extra allocated processors are wasted and not used in the real computation. External fragmentation occurs when a job request cannot be allocated because of the contiguity and shapes allocation conditions, even that the requested number of processors is available.

Two main reasons have led to the adoption the non-contiguous allocation as a plausible solution to the processor fragmentation problem: the first one is that the experimental evidence has shown that only a slight improvement can be gained from further improving the existing contiguous allocation strategies (Lo, et al., 1997; Chang and Mohapatra, 1998). The second is the emergence of the wormhole routing (Ni and McKinley, 1993; Mohapatra, 1998) and advances in the switching techniques that have alleviated the impact of the distance that a message traverse on the communication latency (Lo, et al., 1997; Chang and Mohapatra, 1998). Several non-contiguous allocation strategies have been proposed (Lo, et al., 1997; Mache, et al., 1997; Chang and Mohapatra, 1998; Wu, et al., 2003; Moghaddam and Naghibzadeh, 2006; Bani-Mohammad, et al., 2007; Ababneh, 2008; Bani-Mohammad, et al., 2015; Bani-Mohammad, 2017) which can eliminate both internal and external fragmentation. In non-contiguous allocation, a requested job can be partitioned and allocated multiple disjoint submeshes, instead of being waiting for a contiguous submesh to be available. The non-contiguous allocation strategies can significantly improve the system performance since they can solve the fragmentation problem, however, they suffer from the problem of external message contention, where the messages of different jobs may interfere with each other, and if the contention increased significantly it would increase the communication delay and defeat the gain obtained from the system utilization improvement (Min and Mutka, 1994; Mache and Lo, 1997).

Generally, the aim of any allocation strategy is to reduce the average turnaround time and maximize the system utilization. Moreover, a good allocation strategy must achieve a complete submesh recognition ability while maintaining a little allocation overhead (Yoo and Das, 2002).

The existing non-contiguous allocation strategies (Lo, et al., 1997; Mache, et al., 1997; Chang and Mohapatra, 1998; Wu, et al., 2003; Moghaddam and Naghibzadeh, 2006; Bani-Mohammad, et al., 2007; Ababneh, 2008; Bani-Mohammad, et al., 2015; Bani-Mohammad, 2017) use various techniques to capture and allocate free sub-meshes in the mesh system. However, in general, they focus on maintaining a high degree of contiguity among the processors in the allocated sub-meshes rather than reducing message contention in the submeshes that are allocated to different jobs. In order to maintain a high degree of contiguity among the processors allocated to a given job, these strategies try to compact different allocated submeshes to preserve larger free submeshes for the incoming jobs, although, the full system utilization is unachievable.

To improve the performance of the non-contiguous allocation strategies, it is important to choose the allocation strategy that causes minimal message contention (Mache and Lo, 1997), where the *spatial layout* (i.e., the geometric location) of the allocated submeshes in the mesh system plays a significant role in the interference among jobs' messages (Mache and Lo, 1997).

Motivated by the above observations, a new row based non-contiguous processor allocation strategy for 2D mesh-connected multicomputer, referred to as Row Based Strategy (RBS) is proposed. The proposed strategy considers the spatial layout of the allocated submeshes in the mesh system. RBS classifies the incoming job requests according to their sizes, (large and small); in order to allocate them in submeshes that have minimal shared physical communication channel for the routing paths of their messages. Therefore, and to alleviate message contention especially for large jobs, RBS tries to maintain a high degree of contiguity among the processors allocated to the same job with a little allocation overhead.

Extensive simulation experiments have been carried out in order to compare the performance of the proposed RBS strategy with that of the existing non-contiguous and contiguous allocation strategies. The results have revealed that RBS performs much better than the other non-contiguous and contiguous allocation strategies when the all-to-all communication pattern is used. This is because all-to-all communication pattern produces much message collision and it is considered as the weak point of the non-contiguous allocation strategies (Suzaki, et al., 1996). For instance, for the uniform job size distribution, under a high load, the average turnaround times of RBS are 72%, 60%, 31%, and 54% of the average turnaround times of GABL (Bani-Mohammad, et al., 2007), Paging(0) (Lo, et al., 1997), MBS (Lo, et al., 1997), and FF (Zhu, 1992), respectively. The results have also revealed that the performance of RBS, in most cases, is relatively better than that of the other non-contiguous allocation strategies for the one-to-all and the random communication patterns, and they are all superior to the performance of the FF contiguous allocation strategy. However, the performance of RBS is not better than that of the other contiguous and non-contiguous allocation strategies when the near neighbor communication pattern is used. This is because in this communication pattern, each node allocated to a job communicates with its neighbors (left, right, up, down) that are allocated to the same job, and this is suitable for the strategies that maintain a high degree of contiguity among the allocated processors for a given job and form rectangular shapes for the allocated submeshes. Furthermore, RBS exhibits high system utilization since it eliminates internal and external fragmentation. For instance, under high loads, RBS achieves a mean system utilization up to 78% and up to 85% for uniform and uniform decreasing job size distributions, respectively, but the system utilization for the FF contiguous allocation strategy does not exceed 63%.

5.2 Directions for the Future Works

There are several interesting issues and open problems that worth further investigation. Some of them are briefly described below.

- The performance of the allocation strategies considered in this research has been evaluated based on the First-Come-First-Served (FCFS) scheduling policy. A natural extension of this work would be to evaluate the performance of the proposed allocation strategy with other possible scheduling approaches, such as Out-of-Order (OO) (Ababneh, 2001),

- Shortest-Service-Demand-First (SSD) (Krueger, et al., 1994), and Window-based job scheduling (Ababneh and Bani-Mohammad, 2011).

- In this research, the XY deterministic routing has been used for message routing because it is simple to implement and it contributes in preventing deadlocks, however, it cannot react to changes in networks conditions. In adaptive routing, intermediate nodes take the current network condition, such as failures or congestion into account, when routing the message to the next node in the path. It would be interesting to extend the proposed allocation strategy to this type of routing.

- The performance of the proposed and the existing allocation strategies has been traditionally carried out by means of simulation based on stochastic workload models to generate a stream of incoming jobs. It would be interesting to evaluate the allocation strategies based on real workload traces from different parallel machines, and to compare the results with those obtained in this research.
- The proposed strategy (RBS) has been shown to perform well in 2D mesh network topology. It would be interesting to adapt it to 3D or even a higher dimensional mesh and assess its performance on these network topologies.

References

Ababneh, I. (2001). Job scheduling and contiguous processor allocation for three-dimensional mesh multicomputers. *AMSE Advances in Modelling and Analysis, 6*(4), pp. 43-58.

Ababneh, I. (2008). Availability-based noncontiguous processor allocation policies for 2D mesh-connected multicomputers. *Journal of Systems and Software, 81*(7), pp. 1081-1092.

Ababneh, I., and Bani-Mohammad, S. (2003). Noncontiguous Processor Allocation for Three-Dimensional Mesh Multicomputers. *AMSE Advances in Modelling and Analysis, 8*(2), pp. 51-63.

Ababneh, I., and Bani-Mohammad, S. (2011). A new window-based job scheduling scheme for 2D mesh multicomputers. *Simulation Modelling Practice and Theory, 19*(1), pp. 482-493.

Adve, V., and Vernon, M. (1994). Performance analysis of mesh interconnection networks with deterministic routing. *IEEE Transactions on Parallel and Distributed Systems, 5*(3), pp. 225-246.

Babbar, D., and Krueger, P. (1994). A performance comparison of processor allocation and job scheduling algorithms for mesh-connected multiprocessors. *Proceedings of the 6th IEEE Symposium on Parallel and Distributed Processing*, (pp. 46-53). Dallas, TX.

Bani-Mohammad, S. (2008). *Efficient Processor Allocation Strategies for Mesh-Connected Multicomputers.* Ph.D Thesis, Department of Computing Science, University of Galsgow, Glasgow, U.K.

Bani-Mohammad, S. (2017). All Request Shapes Non-Contgiuous Submesh Allocation Strategy for 2D Mesh Multicomputers. *IEEE International Conference on Engineering & MIS (The IEEE ICEMIS 2017).* Monastir, Tunisia.

Bani-Mohammad, S., and Ababneh, I. (2013). On the performance of non-contiguous allocation for common communication patterns in 2D mesh-connected multicomputers. *Simulation Modelling Practice and Theory, 32*, pp. 155-165.

Bani-Mohammad, S., Ababneh, I., and Hamdan, M. (2010). Comparative Performance Evaluation of Non-Contiguous Allocation Algorithms in 2D Mesh-Connected Multicomputers. *Proceedings of the 10th IEEE International Conference on Computer and Information Technology (CIT 2010)* (pp. 2933–2939). Washington, DC: IEEE Computer Society.

Bani-Mohammad, S., Ababneh, I., and Yassen, M. (2015). Non-contiguous processor allocation in the mesh-connected multicomputers using compaction. *Journal of Information Technology Research*, *18*(4), pp. 57-75.

Bani-Mohammad, S., Ould-Khaoua, M., and Ababneh, I. (2007). An Efficient Non-Contiguous Processor Allocation Strategy for 2D Mesh Connected Multicomputers. *Juornal of Information Sceinces, 177*(14), pp. 2867-2883.

Bani-Mohammad, S., Ould-Khaoua, M., Ababneh, I., and Machenzie, 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 (ICPADS'06). 2*, pp. 41-48. IEEE Computer Society Press.

Blumrich, M., Chen, D., Coteus, P., Gara, A., Giampapa, M., Heidelberger, P., Singh, S., Steinmacher-Burow, B., Takken, Steinmacher-Burowmin, T. and Vranas, P. (2003). *Design and Analysis of the BlueGene/L Torus Interconnection Network.* IBM Research Report RC23025, IBM Research Division. Thomas J. Watson Research Center.

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

Chiu, G.-M., and Chen, S.-K. (1999). An efficient submesh allocation scheme for two-dimensional meshes with little overhead. *IEEE Transactions on Parallel and Distributed Systems, 10*(5), pp. 471-486.

Chuang, P., and Tzeng, N. (1994). Allocating precise submesh in mesh-connected systems. *IEEE Transaction on Parallel and Distributed Systems, 5*(2), pp. 211-217.

Chuang, P.-J., and Tzeng, N.-F. (n.d.). Allocating precise submeshes in mesh connected systems. *IEEE Transactions on Parallel and Distributed Systems, 5*(2), pp. 211-217.

Cray. (2005). Cray XT3 Datasheet.

Das Sharma, D., and Pradhan, D. (1996). Submesh Allocation in Mesh-Multicomputers Using Busy-List: A Best-Fit Approach with Complete Recognition Capability. *Journal of Parallel and Distributed Computing, 36*(2), pp. 106-118.

Drewes, C. (1996). *Simulating Virtual Cut-through and Wormhole Routing in a Clustered Torus.* M.Sc. Thesis, Laboratory of Computer Architecture and Digital Techniques (CARDIT), Faculty of Electrical Engineering, Delft University of Technology.

Duato, J., Yalamanchili, C., and Ni, L. (1997). *Interconnection Networks: An Engineering Approach* (1st ed.). Los Alamitos, CA, USA: IEEE Computer Society Press.

Ferreira, A., vel Lejbman, G., and Song, S. (1994). Bus based parallel computers: A viable way for massive parallelism. *Proceedings of Parallel Architectures Languages Europe (PARLE '94), Lecture Notes in Computer Science 817* (pp. 553-564). Berlin, Heidelberg: Springer Berlin Heidelberg.

Foster, I. (1995). *Designing and building parallel programs: concepts and tools for parallel software engineering.* MA: Addison-Wesley.

Fujii, H., Yasuda, Y., Akashi, H., Inagami, Y., Koga, M., Ishihara, O., Kashiyama, M., Wada, H., and Sumimoto, T. (1997). Architecture and performance of the Hitachi SR2201 massively parallel processor system. *Proceedings of the 11th International Parallel Processing Symposium (IPPS'97)* (pp. 233-241). Washington, DC, USA: IEEE Computer Society Press.

Intel Corporation. (1991). A Touchstone DELTA system description.

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

Krueger, P., Lai, T., and Radiya, V. (1994). Job scheduling is more important than processor allocation for hypercube computers. *IEEE Transactions on Parallel and Distributed Systems, 5*(5), pp. 488-497.

Kruskal, C., and Snir, M. (1983). The performance of multistage interconnection networks for multiprocessors. *IEEE Transactions on Computers, 32*(12), pp. 1091-1098.

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

Li, k., and Cheng, K. -H. (1991). A Two-Dimensional Buddy System for Dynamic Resource Allocation in a Partitionable Mesh Connected System. *Journal of Parallel and Distributed Computing, 12*(1), pp. 79-83.

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

Mache, J., and Lo, V. (1997). The Effects of Dispersal on Message-Passing Contention in Processor Allocation Strategies. *Third Joint Conference on Information Sciences, Sessions on Parallel and Distributed Processing*, (pp. 223-226).

Mache, J., Lo, V., and Windisch, K. (1997). Minimizing Message-Passing Contention in Fragmentation-Free Processor Allocation. *Proceedings of the 10th International Conference on Parallel and Distributed Computing Systems*, (pp. 120-124).

Min, D., and Mutka, M. (1994). A multipath contention model for analyzing job interactions in 2-D mesh multicomputers. *Proceedings of 8th International Parallel Processing Symposium*, (pp. 744-751). Cancun.

Min, G. (2003). *Performance Modelling and Analysis of Multicomputer Interconnection Networks.* Ph.D. Thesis, Department of Computing Science, University of Glasgow, Glasgow, U.K.

Moghaddam, S., and Naghibzadeh, M. (2006). A new processor allocation strategy using ESS (expanding square strategy). *14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'06)*, (pp. 137-140). Los Alamitos, CA, USA: IEEE Computer Society.

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

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

Moore, S., and Lionel, M. (1996). The Effects of Network Contention on Processor Allocation Strategies. *In Proceedings of the 10th International Parallel Processing Symposium*, (pp. 268-274).

Ni, L., and McKinley, P. (1993). A survey of wormhole routing techniques in direct networks. *IEEE Computer, 26*(2), pp. 62-76.

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

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

ProcSimity Manual . (1997). ProcSimity V4.3 User's Manual. University of Oregon.

Seo, K.-H. (2005). Fragmentation-efficient node allocation algorithm in 2D mesh-connected systems. *Proceedings of the 8th International Symposium on Parallel Architecture, Algorithms and Networks (ISPAN'05)* (pp. 318-323). Washington, DC, USA: IEEE Computer Society Press.

Suzaki, K., Tanuma, H., Hirano, S., Ichisugi, Y., Connelly, C., and Tsukamoto, M. (1996). Multi-tasking method on parallel computers which combines a contiguous and a non-contiguous processor partitioning algorithm. *Proceedings of the 3rd International Workshop on Applied Parallel Computing, Industrial Computation and Optimization* (pp. 641-650). London: Springer.

Wan, M., Moore, R., Kremenek, G., and Steube, K. (1996). A batch scheduler for the Intel Paragon with a non-contiguous node allocation algorithm. *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, IPPS '96* (pp. 48-64). Berlin, Heidelberg: Springer Berlin Heidelberg.

Windisch, K., Miller, J., and 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). Washington, DC, USA: IEEE Computer Society Press.

Wu, F., Hsu, C.-C., and Chou, L.-P. (2003). Processor Allocation in the Mesh Multiprocessors Using the Leapfrog Method. *IEEE Transactions on Parallel and Distributed Systems, 14*(3), pp. 276-289.

Yoo, B.-S., and Das, C.-R. (2002). A Fast and Efficient Processor Allocation Scheme for Mesh-Connected Multicomputers. *IEEE Transactions on Parallel and Distributed Systems, 51*(1), 46-60.

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

ملخَّص

تدعم أنظمة متعددات الحواسيب الشبكية أنواع مختلفة من التطبيقات بمختلف الأحجام والخصائص في بيئة متعددة المستخدمين، ولذلك فمن المهم استخدام استراتيجيات فعَّالة لتخصيص المعالجات لاستغلال القدرة الحسابية لهذه الأنظمة، حيث تعتمد فعالية استراتيجيات التخصيص على قدرتها على زيادة استغلال المعالجات وتقليل وقت مكوث المهام في النظام.

تقسم استراتيجيات تخصيص المعالجات في متعددات الحواسيب الشبكية الى فئتين رئيسيتين: متجاورة و غير متجاورة. تعتمد استراتيجيات التخصيص المتجاور في التخصيص على التجاور الفيزيائي بين المعالجات كما وتشترط ان يكون شكل الشبكة الجزئية المخصصة شبيه بشكل شبكة الربط في النظام، وقد يؤدي شرط التجاور هذا الي ظهور مشكلة الكسيرات الخارجية وبشكل كبير؛ وتحدث الكسيرات الخارجية عند وجود مجموعة من المعالِجات المتوفرة في النظام والتي تكفي لطلب معين ولكن لا يمكن تخصيصها لذلك الطلب بسبب عدم تجاورها، وتؤدي هذه المشكلة إلى تدني معدل استغلال المعالجات في النظام، وبالتالي زيادة وقت مكوث المهام في النظام. تم اقتراح خوارزميات التخصيص غير المتجاور كحل عملي لمشكلة الكسيرات، وقد شجع على ذلك التطور في تقنيات توجيه ونقل الرسائل داخل الشبكة مثل (Wormhole Routing) والتي قللت من تأثير المسافة التي تقطعها الرسالة على التأخير الكلي للتراسل، وعلاوة ًعلى ذلك، فقد أظهرت الأدلة التجريبية أن التحسين الإضافي على خوارزميات التخصيص المتجاور الموجودة لا يؤدي إلّا إلى تحسين طفيف على الأداء للنظام بشكل عام، لذلك فقد تم تبني التخصيص غير المتجاور للمعالجات، حيث يمكن من خلاله تقسيم طلب مهمة معينة الى إجزاء وتخصيصها في إجزاء متفرقة من الشبكة بدلا ً من انتظارها لجزء من الشبكة يحتوي على معالجات متجاورة ليصبح متاحاً، ومن المتوقع أن يؤدي ذلك إلى تحسين في معدل استغلال المعالجات في النظام وبالتالي تقليل معدل أوقات مكوث المهام في النظام. ومع ذلك التحسين المتوقع لإداء النظام بسبب استخدام التخصيص غير المتجاور، الا ان استراتيجيات التخصيص غير المتجاور تعاني من مشكلة التزاحم بين رسائل المهام المختلفة، مما قد يؤدي إلى زيادة التأخير في الوقت المستغرق في التراسل.

تستخدم خوارزميات التخصيص غير المتجاور الحالية تقنيات مختلفة لإيجاد وتخصيص الشبكات الجزئية المتاحة، إلّا أنها في الغالب تركز على الحفاظ على درجة عالية من التجاور بين المعالجات المخصصة لمهمة معينة في شبكة جزئية معينة عن طريق تحشير المهام المخصصة بجانب بعضها للحفاظ على أكبر قدر ممكن من المعالجات المتجاورة المتاحة لطلبات المهام القادمة. تم في هذه الرسالة اقتراح استراتيجية تخصيص غير متجاور جديدة، يشار اليها بخوارزمية التخصيص غير المتجاور المُعتَمِدة على الصفوف في متعددات الحواسيب الشبكية ثنائية الأبعاد. تمتلك الخوارزمية المُقتَرَحة القدرة على منع حدوث الكسيرات الداخلية والخارجية، كما وتقلل من تزاحم الرسائل بين المهام المختلفة في الشبكة. تُصنِّف الخوارزمية المُقتَرَحة طلبات المهام الواردة حسب حجمها الى نوعين: كبيرة وصغيرة؛ وذلك بهدف التقليل من تزاحم رسائل المهام المختلفة.

أظهرت نتائج المحاكاة أنَّ أداء الخوارزمية المُقتَرَحة يفوق أداء استراتيجيات التخصيص المتجاور وغير المتجاور الأخرى من حيث معدل أوقات مكوث المهام في النظام، وذلك عند استخدام نمط التراسل (الكل – للكل)؛ وهذا بسبب قدرة الخوارزمية المقترحة على تقليل تزاحم الرسائل في الشبكة، كما أظهرت النتائج أيضاً أنَّ أداء الخوارزمية المقترحة أفضل نسبياً من أداء خوارزميات التخصيص غير المتجاور الأخرى عند استخدام نمطي التراسل (الواحد - للكل) و (العشوائي)، وذلك في معظم الحالات، بينما تفوقت استراتيجيات التخصيص الأخرى على الخوارزمية المقترحة عند استخدام نمط الاتصال (المجاور القريب).