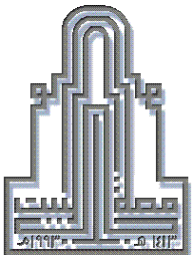

Al al-Bayt University

Prince Hussein Bin Abdullah College for Information Technology

Irregular Strategy for Sub-mesh Allocation strategy in 2D Mesh-Connected Multicomputers

الإستراتيجية الغير منتظمة للتخصيص في متعددات الحواسيب الشبكية ثنائية الأبعاد

By

Ra'ed Thiab Awwad Al Harafsheh

1320901009

Supervisor

Dr. Saad Bani-Mohammed

Co-supervisor

Prof. Ismail Ababneh

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

May, 2016

Dedication

This thesis is dedicated to my parents, my wife, my children, and Dr.Saad Bani-Mohammad for their courtliness, love,appreciating to support and encouragement.

To every one who ask Allah to help me in my thesis. For all their, I ask my God to give them what they wish, and I thank all of them.

Acknowledgments

This thesis was monitored by my main supervisor, Dr. Saad Bani-Mohammed and my co-supervisor, Prof. Ismail Ababneh, who spent a lot of time helping me to write the thesis. I am grateful for them to help me in choosing the main idea of the thesis, obtaining the results, writing the thesis and moreover the encouragement to complete this thesis perfectly.

On another hand, I would like to thank the university with its crew for facilitating the research procedures, and my family for encouraging me. Also, special thanks to my wife who stood by me, supported and encouraged me all the time during my study. I could not finish this without her, I will remember her efforts forever.

Ra'ed TH. Al Harafsheh

Table of contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Meaning
MBS	Multiple Buddy Strategy
FCFS	First-Come-First-Served
FF	First Fit
BF	Best Fit
ISA	Irregular Shape Allocation

Abstract

Contiguous and non-contiguous processor allocation strategies are two categories of the processor allocation strategies used to allocate an incoming job request in the mesh-connected multicomputer. Contiguous allocation suffers from high processor fragmentations as the processors allocated to a job are physically contiguous and have the same topology as that of the interconnection network of the multicomputer. This leads to a degradation in system performance in terms of average turnaround time of jobs and mean system utilization. In non-contiguous allocation, a job can execute on separate smaller sub-meshes rather than waiting until a single sub-mesh of the requested size and rectangular shape is available. Although non-contiguous allocation increases message contention inside the network, lifting the contiguity condition can reduce processor fragmentation and increase system utilization. Most existing non-contiguous allocation strategies that have been suggested for the mesh suffer from processor fragmentation and message contention inside the network. In addition, the allocated sub-meshes should be in regular shapes. In this thesis, we present a new non-contiguous allocation strategy, referred to as irregular shape allocation (ISA) strategy that eliminates processor fragmentation and alleviates the contention inside the network. The main idea of ISA is that it does not depend on rectangular shape as other previous allocation strategies, where the allocated sub-meshes in ISA can be in any shape (regular or irregular), in order to improve the system performance in terms of job turnaround time and system utilization. We compare the performance of ISA with that of the well-known contiguous and non-contiguous allocation strategies. The simulation results have shown that the ISA allocation strategy has the same performance in terms of both job turnaround time and system utilization as Paging(0) and MBS when the communication pattern used is one to all, and better performance in terms of both job

turnaround time and system utilization than that of the previous allocation strategies considered in this thesis when the communication patterns used are all to all and random. While the result show that the contiguous allocation strategy perform better than non contiguous allocation strategy in terms of job turnaround time when communication pattern used is near neighbor.

Chapter 1

Introduction

1.1 introduction

Many problems such as Atmosphere, chemistry, physics, defense and web search are so large and/or complex that it is impractical or impossible to solve them on a single computer, especially given limited computer memory. So, there is a need to build a parallel computer from cheap commodity components (Gottlieb, Allan. Almasi, George S, 1989). Parallel computing is the use of multiple computer resources to solve large problems concurrently by dividing the problem into smaller ones (Fan Wu, Ching-Chi Hsu, and Chou, Li-Ping, 2003). Parallel computing has become the dominant paradigm in computer architecture, mainly in the form of multi-core processors (Asanovic, et al., 2006).

Parallel computers can be roughly classified according to the level at which the hardware supports parallelism, with multi-core and multi-processor computers having multiple processing elements within a single machine, while clusters and grids use multiple computers to work on the same task. Specialized parallel computer architectures are sometimes used alongside traditional processors, for accelerating specific tasks (Asanovic, et al., 2006), considering the interconnection network between the processors and the message interaction in a mesh-connected multicomputer. Both two dimensional (2D) and three-dimensional (3D) meshes and tori have been used in recent commercial and experimental parallel computers, such as the Intel Paragon (Intel Crop, Paragon XP/S Product Overview, Internet 2015), the IBM BlueGene/L (Gara et al.,2005), and the Cray XT3 (Cray, Cray XT3 Datasheet, Internet 2015), and they have

gained increasing importance in the design of distributed memory multi processor systems. This is due to their regularity, scalability, simplicity, partition ability and ease of construction (Seo, K.-H, 2005). To achieve high performance in mesh connected multicomputer, processor allocation strategies have been proposed in order to increase system utilization and decrease job turnaround time, where system utilization is the percentage of processors that are utilized over time (ProcSimity V4.3, 1997), and job turnaround time is the time that the job spends in the mesh system from arrival to departure (ProcSimity V4.3, 1997).

In computing, processor allocation is necessary for any application to be executed on the system, it is responsible to select and allocate free processors to newly arrived jobs to be able to execute. While the scheduler controls the order in which waiting jobs in the waiting queue are executed (ProcSimity V4.3, 1997). Processor allocation strategies are divided into two types: contiguous (Chiu, G.-M. Chen, S.-K, 1999, Chuang, P.-J. Tzeng, N.-F, 1994), and non-contiguous (Bani-Mohammad, et al., 2007, Lo, et al. 1997), that are employed in a multicomputer. In contiguous allocation, the processors that are allocated to a parallel job are physically contiguous and have the same shape of the mesh system. Contiguous allocation suffers from both internal and external fragmentation (Bani-Mohammad, et al.,2009, Chang, C.-Y. and Mohapatra, P, 1998, Lo, et al., 1997). Internal processor fragmentation occurs when the allocation strategy allocates processors for an incoming job request more than it requires, while external processor fragmentations occurs when there are enough number of free processors in the mesh system but they cannot be allocated because they do not have a rectangular shape (Attari S and Isazadeh, 2006, Moghaddam S. and Naghibzadeh, M, 2006, Chang, C.-Y. and Mohapatra, P, 1998). The problem of internal and external fragmentation has been eliminated by using the non-contiguous allocation. In non-contiguous allocation,

the job can execute on multiple disjoint smaller sub-networks rather than always waiting until a single sub-network of the requested size and shape is available (Chang, C.-Y. and Mohapatra, P, 1998, Lo, et al., 1997), and this results in improving the system performance in terms of system utilization and job turnaround time.

The main goal of any allocation strategy is to improve system performance. To achieve this goal, the allocation strategies should concentrate on three pivots: minimizing processor fragmentation, increasing the system utilization and decreasing the job turnaround time. So, a new non-contiguous allocation strategy has been proposed in this thesis to achieve these three pivots. The proposed strategy differs from the previous strategies by allocation the sub-mesh of the processors regardless of its shape (regular or irregular).

1.2 Motivation and Contribution

Contiguous and non-contiguous allocations are used to allocate an incoming job request to the appropriate sub-mesh size in the mesh-connected multicomputer. As previously reported in Section 1.1, contiguous allocation suffers from high processor fragmentation as the processors allocated to a job are physically contiguous and have the same shape of the interconnection network. This causes degradation in system performance in terms of average turnaround time of job and mean system utilization. Non-contiguous allocation achieves high performance over contiguous allocation strategies, because the job request can be split into smaller parts rather than always waiting until a sub-mesh of the requested size and rectangle shape is available. The main factors that affect the performance of the processor allocation strategies include processor fragmentations (internal and external) and message contention. Most non-contiguous allocation

strategies that have been suggested for the mesh suffer from processor fragmentation and message contention inside the network (Bani-Mohammad, et al., 2006, Lo, et al., 1997). Moreover, the contiguous and non-contiguous allocation strategies proposed for 2D meshes always allocate an incoming job request in a regular shape (Lo, et al., 1997).

Motivated by the above observations, this thesis presents a new non-contiguous allocation strategy that always allocate an incoming job request while the requested number of free processors is available in the mesh-connected multicomputer. This proposed strategy is referred to as Irregular Shape Allocation (ISA) strategy that eliminates the processor fragmentation and alleviates the contention inside the network. The main idea of ISA is that it does not depend on rectangular shape as other previous allocation strategies, where the allocated sub-meshes in ISA can be in any shape (regular or irregular), and hence improves the system performance in term of job turnaround time and system utilization. The performance of ISA is compared against that of the existing non-contiguous allocation strategies Paging(0) (Lo, et al., 1997), Random (Lo, et al., 1997) and MBS (Lo, et al., 1997). These strategies have been selected because they have been shown to perform well in (Fan Wu, Ching-Chi Hsu, and Chou, Li-Ping, 2003). Furthermore, ISA is also compared against the contiguous First Fit strategy (FF) (Zhu, Y. H, 1992), as this has been used in several previous related studies (Yoo, B.-S and Das, C.-R, 2002). The proposed strategy improves the system performance in terms of job turnaround time and system utilization as compared to the existing contiguous and non-contiguous allocation strategy considered in this thesis.

1.3 Outline of thesis

The rest of the thesis is organized as follows. Chapter 2 describes the well-known allocation strategies that have been proposed for 2D mesh-connected multicomputer and considered in this thesis; it also presents the system model assumed in this thesis. Finally, the chapter justifies the selection of simulation as a study tool.

Chapter 3 introduces the proposed allocation algorithm as a new non-contiguous allocation algorithm for 2D Mesh-Connected multicomputer, and presents the definitions for the proposed algorithm and also the required notations related to mesh network. The ISA allocation and de-allocation process are explained in this chapter, and extensive simulation experiments are carried out to evaluate the performance of the proposed algorithm and compare it with the previous contiguous and non-contiguous allocation strategies.

Chapter 4 summarizes the results presented in this work and introduces some directions for future work.

Chapter 2
Background and Preliminaries

2.1 Related Work:

In this section a brief description of some allocation strategies that have been suggested for 2D network is presented.

First-Fit (FF) and Best-Fit (BF) strategies: In these strategies (Zhu, Y. H, 1992), the free sub-meshes are scanned and the job is allocated to the appropriate free sub-mesh. The FF strategy allocates an incoming job to the first available sub-mesh it finds, while the BF strategy tries to allocate the job to a sub-mesh that has the largest number of busy neighbors and smallest surrounding free area.

Random: In this strategy (Lo, et al., 1997), an incoming job request, which is a request for *n* processors, is allocated randomly if there are enough processors for an incoming job request. The random strategy eliminates both the internal and external fragmentation, but there is no contiguity enforced under this strategy, so we expect much communication interference between jobs' messages.

Paging Allocation Strategy: In this strategy (Lo, et al., 1997), the entire 2D mesh is initially divided into pages, which are square blocks having the side lengths of $2^{page-size}$, where the page is the basic unit of allocation. In this strategy, the pages are scanned according to the indexing schemes (row-major indexing, shuffled row-major indexing, snake-like indexing, and shuffled snake-like indexing). If a job requests for *k* processors, then the requested number of processors will be removed from the list of free processors and the corresponding pages are allocated.

Figure 2.1 shows an example of the paging allocation, which uses the row-major indexing scheme and a page size of (2×2 blocks). Assume a job requests for seven processors. The first two items in the list of free processor (1st, 4th) as shown in figure 2.1 are removed and the eight processors are allocated, which causes an internal fragmentation of (0.125). When the page-size is 0, then there is neither internal nor external fragmentation.

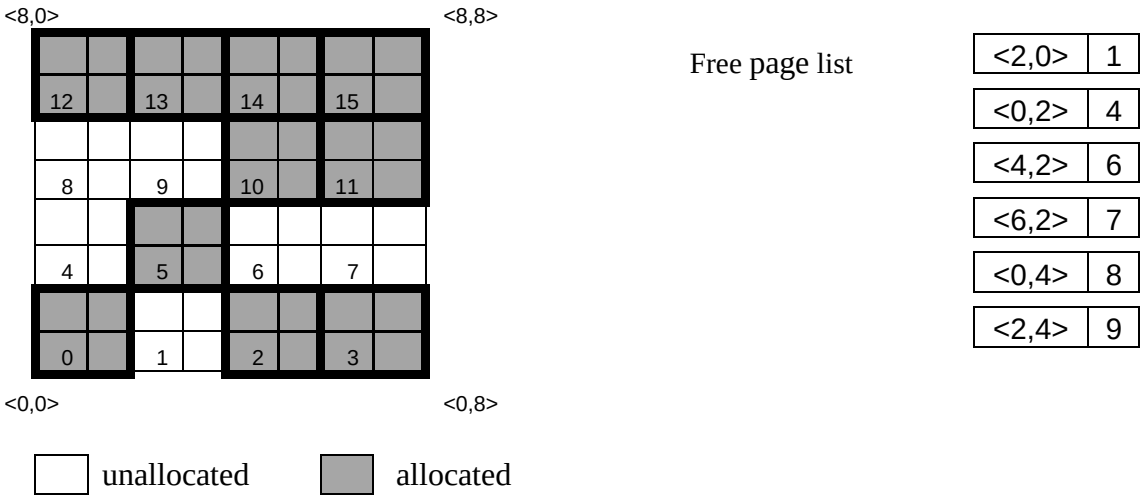

Figure 2.1: Mesh and free page list for the paging $_{\text{row-major}}(1)$ allocation strategy.

Multiple Buddy Strategy (MBS): This strategy (Lo, et al., 1997) is an extension of the 2D buddy strategy (Li, K. Cheng, K.-H, 1991). In MBS, the mesh system is initially divided into non-overlapped square sub meshes with side lengths that are power of 2. In general, a request for *n* processors is represented as a base-4 number of the form $_{2\times 2,\ 2\ \times\ 2\ ,.....,2\times 2,2\times 2}^{m\ \ m\ m-1\ \ m-1\qquad\quad 1\ \ 1\ 0\ \ 0}$, where $m=\log_4 p$. If a block of a desired size is unavailable, MBS searches for a larger block and continuously breaks it down until it reaches the appropriate size.

Leapfrog: In this strategy (Fan Wu, Ching-Chi Hsu, and Chou, Li-Ping, 2003), a new data structure, the *R*-array, is proposed to represent the mesh at first. The allocated processors are represented by a negative value, while the non-allocated processors are represented by a positive value. Both negative and positive values represent the number of allocated and non-allocated processors followed that value, respectively. The leapfrog can jump to the processors that can serve as a base of a free sub-mesh, which causes the search space faster than any other strategies considered in this thesis.

Assume a job $j(w, h)$ arrives to the mesh $M(W, H)$. The allocation process in leapfrog will start scanning from the lower leftmost coordinate $R(0, 0)$ to check whether the scan reaches the right or top reject sets. If the process reaches the right reject set, the process directs its scan to the first value of the next higher row, if the process reaches the top reject set, the process aborts and the job is queued. However, if the currently checked processor (i, j) is not in the right or top reject sets, the process will check h vertically processors from processor (i, j) to processor $(i, j+ h-1)$. With the statistical data in the *R*-array, the process can determine whether these h processors are free and long enough to form the required free sub-mesh.

Example 1: Consider the mesh system shown in Figure 2.2-(a) and assuming that a job request of $J(3\times3)$ arrives to the mesh system. The first-fit process first checks R(0, 0) and finds its value being -3. The process then leapfrogs this allocated processors and directly reaches processor (3, 0) which is not in the right or top reject sets. Then the process finds that the value of R(3, 0) is 5 which is positive and larger than the width of the job (3). Then, the process intends to check whether the other two vertically values at R(3, 1) and R(3, 2) are positive and long enough to satisfy the required job. However, the value of R(3, 1) is -2 which is less than the width of incoming job (3), the process then leapfrogs this allocated processors to processor (5, 0) which is also not in the right

or top reject sets and it has a positive value 3. After checking the three values in R(5, 0), R(5, 1), and R(5, 2), the process finds that the three values at R(5, 0), R(5, 1), and R(5, 2), are positive and long enough to allocate the incoming job. Thus, the process assigns the free sub-mesh at processor (5, 0) to the incoming job $J(3\times3)$ as it shown in figure 2.2-(b).

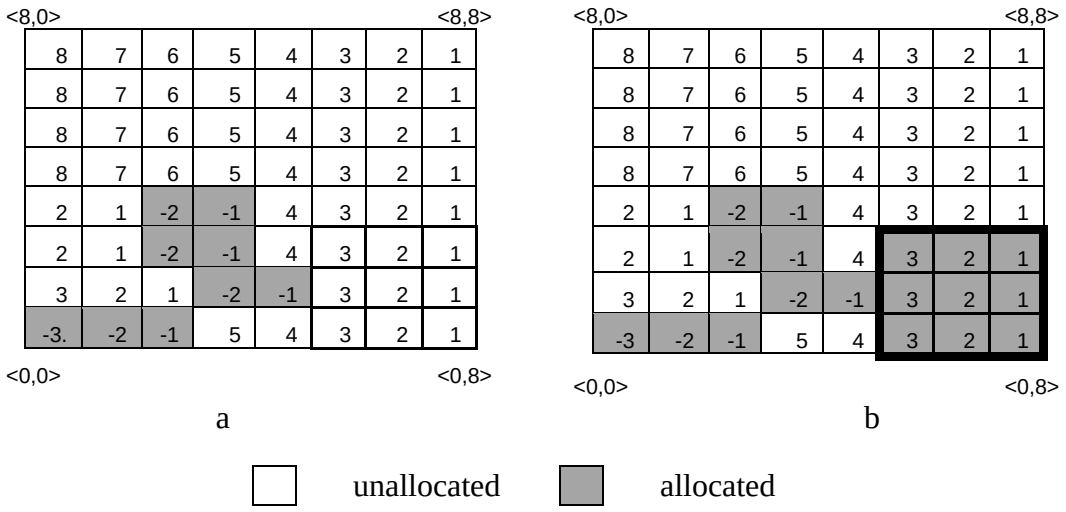

Figure 2.2: (a) a mesh (8×8) before allocating a job of (3×3), (b) the mesh after allocating a job request (3×3).

2.2 System Model

We use the simulation to evaluate and compare the performance of the non-contiguous allocation strategies (ISA, Paging(0), Random, MBS), and the performance of the contiguous allocation strategy (FF) with that of the proposed ISA strategy. In this thesis, we use the communication patterns, one-to-all, all-to-all, random, and Near Neighbour communication patterns. In one-to-all, a randomly selected processor allocated to a job sends a message to all other processors allocated to the same job. In all-to-all communication pattern, each processor allocated to a job sends a message to each other

processors allocated to the same job. In Random communication pattern, the message source and destination is a random pair of processors allocated to the same job. In Near Neighbour communication pattern, each processor allocated to a job sends a message to its neighbours (up, down, left and right). The main parameters measured in the simulation experiments are the job turnaround time and the system utilization. The interconnection network is the way in which the nodes connected to each other via two unidirectional channels with no wrap-around edges, and the routing is performed in the network through XY routing, where is the node sends a message firstly through X axis right or left then through Y axis up or down depending on the location of destination node. The switching method used in this thesis is the wormhole switching (also called wormhole routing) to determine the way messages are handled as they travel through intermediate nodes. This type of switching has been used in this thesis as it has been widely used in practical multicomputer due to its low buffering requirement and good performance (ProcSimity V4.3, 1997). In this thesis, our focus is on the performance of allocation strategies. Thus we fixed the choice of scheduling algorithm using the straight forward First-Come-First-Served (FCFS) scheduling.

2.3 The Simulation Tool (ProcSimity Simulator)

ProcSimity is a software tool written in C programming language for research in the area of processor allocation and job scheduling for distributed memory multicomputer (ProcSimity V4.3, 1997). ProcSimity was developed at the University of Oregon (ProcSimity V4.3, 1997), it is widely used for processor allocation and job scheduling in mesh-connected multicomputer (Ababneh, I and Fraij, 2001, Ababneh, I, 2006, Bani-Mohammad, et al.,2007, Bani-Mohammad, et al., 2006, Lo, Bunde, D. P. Leung, V. J. and Mache, J, 2004, Mache, J. Lo, V. and Garg, S, 2000, Lo, V. and Mache, J, 2002,

Mache, J. Lo, V and Windisch, K, 1997, Lo, et al., 1997), because it is an open source and also it has detailed operations of multicomputer networks (ProcSimity V4.3, 1997, Windisch, K. Miller, J. V. and Lo, V, 1995).

The tool supports experimentation for highly parallel systems based on the mesh and k-ary n-cube topologies, and for a range of control and routing technologies. ProcSimity has some main issues, it models a stream of independent user incoming jobs in the system and processor allocation, where the selection of a set of processors for a newly incoming job depends on the job request for a specific number of processors or a specific sized block. The processors are allocated for their entire lifetime, and then released when the execution is completed. The processor allocation algorithms included in our tool are divided into two categories: contiguous allocation algorithms, in which the set of processors allocated to a job request are physically contiguous and non-contiguous allocation algorithms, in which the processors allocated to a job are dispersed over all the mesh.

Processor scheduling that refers to the scheduling discipline used at the job level controls the selections of the next job for which processors are to be allocated. If the mesh has enough processors for an incoming job, then the free processors are allocated to that job, otherwise the incoming job is sent to the system waiting queue. Before a waiting job can leave the waiting queue, the scheduler must place the job at the head of the queue, and when a job is ready to be executed, the allocator assigns it to the available sub-mesh of processors in the mesh, which may be contiguous or non-contiguous, depending on the allocation strategy used. The execution job still holds the processors in the mesh until it terminates its running, at this time it releases the allocated processors to be used by another incoming job request. The architecture modelled by ProcSimity consists of a network of processors interconnected through

message routers at each node. Neighbours nodes are connected by two unidirectional channels, and messages may be routed by either store-and-forward, virtual cut-through, or wormhole flow control.

Each simulation run contains the values of the measured metrics (utilization, turnaround time, service time and finish time), and the final simulation results are averaged over enough independency runs so that the confidence level is 95% and relative errors do not exceed 5%.

Chapter 3

Irregular Shape Allocation (ISA)

3.1 Preliminaries:

The mesh is represented by *M(W, H)*, where *W* is the width of mesh and *H* is its height. The number of processors in the mesh system is equal to $W \times H$. The incoming job request is represented by *j(w, h)*, where the number of processors requested by an incoming job is $W \times H$. The processor located in row *j* and column *i* is identified by coordinates *(j, i)*. The searching process starts from the lower-leftmost coordinate (0, 0) of the mesh, where 0 <= *i* < *W* and 0 <= *j* < *H*.

Definition 1. The coverage set of regions to an incoming job is the free processors in each row that can serve as the base of the sub-mesh to host that job, considering the contiguity between rows. Otherwise, randomly free processors without contiguity will be selected.

Definition 2. A free run in a mesh is a sequence of free processors along the horizontal axis. A free run counted from processor p is defined as the sequence of free processors whose leftmost processor is p. The length of this free run is defined as $(1 + k)$ if processor p is free and is followed by a sequence of k free processors, where $k > 0$ (Fan Wu, Ching-Chi Hsu, and Chou, Li-Ping, 2003).

Definition 3. An allocated run in a mesh is a sequence of allocated processors along a horizontal axis. An allocated run counted from processor p is defined as the sequence of the allocated processors whose leftmost processor is p. The length of the allocated

run is $(1 + k)$ if processor p is allocated and is followed by a sequence of k allocated processors, where $k < 0$ (Fan Wu, Ching-Chi Hsu, and Chou, Li-Ping, 2003).

Definition 4. An (*r*-array) is a two-dimensional array for a mesh. For each processor *p*, there is an element in the r-array storing the length of the free or allocated run counted from p . For example, if there is a free run of length *v* counted from processor *(j, i)*, the value of element *(j, i)* in the r-array, denoted as *r(j, i)*, is *v*. If there is an allocated run of length *v* counted from processor *(j, i)*, then *r(j, i)* is *–v* (Fan Wu, Ching-Chi Hsu, and Chou, Li-Ping, 2003). The positive value means that the number of free processors is equal to that value. Negative number denotes to the allocated processors that are equal to the negative value.

Definition 5. An (*R*-array), is a one-dimensional array that contains the first coordinates *(j, i)* for all free contiguous processors in each row that should be allocated for an incoming job request.

Definition 6. A request-array is a one-dimensional array that contains the number of free processors in each row that can be allocated for an incoming job request.

Definition 7. A sub-mesh in mesh *M(W, H)*, is unshaped frame, so the sub-mesh for an incoming job request may or may not like a rectangular shape, depending on the number of free processors in the mesh.

Figure 3.1 shows a mesh *M(8, 8)*, that represents an allocated sub-meshes *S1*, *S2* and *S3* using ISA strategy. The gray nodes denote to the allocated processors and the white denote to the free ones. The allocated sub-mesh can be identified by a pair of coordinates *(j, i)* as a rectangular shape that is shown in allocated sub-mesh *S1* in figure 3.1, and this is used in the previous allocation strategies. Moreover, the allocated sub-

mesh may not identified by a pair of coordinates *(j, i)* as shown in the allocated sub-meshes *S2* and *S3* in figure 3.1 which is considered in our proposed ISA strategy.

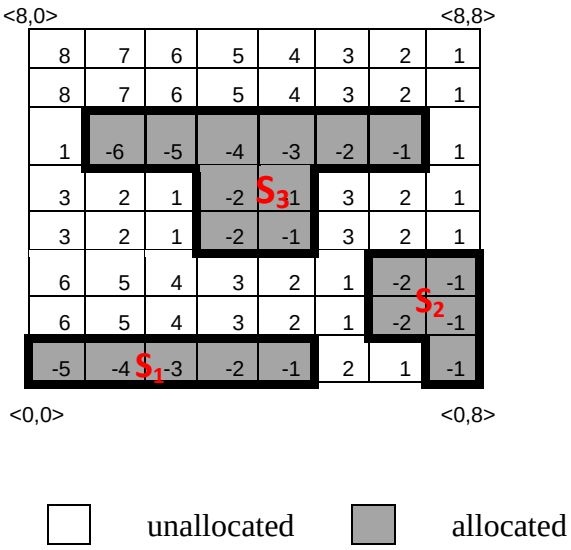

Figure 3.1: An Example of 8×8 2D mesh.

3.2 Irregular Shape Allocation Strategy (ISA)

In the next tow sub-sections we show how the ISA allocate the free processors to the incoming job request, and how can it release the allocated processors to be used again by another job request.

3.2.1 ISA allocation

When a job *j(w, h)*, arrives to the mesh system, where *w* is the width of the job request and *h* is its height, the allocation process starts scanning the processors in the mesh system according to the row-major indexing scheme in an attempting to allocate the requested sub-mesh. The idea of the proposed scheme in this thesis is to allocate the sequence of free contiguous processors for an incoming job request as much as possible.

To allocate the first positive value in the mesh (free base node), the algorithm starts scanning from the lower-leftmost corner of the mesh system at node (0, 0). If it has a negative value, the strategy skips the number of processors according to this negative value. Otherwise the strategy starts allocating the processors from the first positive value (first base node), considering the contiguity between processors in one row and if $w \times h$ processors are not greater than the positive value, the processors are allocated and the allocation is done by storing the node *(j, i)* in *R* array and the positive value in *R(j, i)* in request array then increment the size of *R* array and request array with parameters *k* and *re* respectively. Where *k* is the number of processors in *R* array (size of *R* array) and *re* is the number of free processors in each row that will be allocated for a job (size of request array). If $w \times h$ processors are greater than the positive value, the strategy initially allocates the free processors starting from the first positive value (first base node), then it stores the node *(j, i)* in *R* array and the positive value *R(j, i)* in request array then increment the size of *R* array and request array with parameters *k* and *re* respectively. Then, subtracts the positive value in the first base node from an original job request to get a new job request, and then it scans for a new free processor (second base node) in the next higher row, considering the contiguity with the previous row by one processor at least. The strategy repeats the previous step until all processors for an incoming job request are allocated, then the allocation is done. If the strategy reaches a higher row in the mesh system without completely allocating the incoming job request and there are still enough non-contiguous processors in the mesh system, then it allocates them according to the row major scheme starting from the first node in the mesh system. Otherwise the allocation fails, and the job is queued.

Example 1. Consider the mesh system shown in Figure 3.2-(a), and assuming that a job request *J(2×7)*, the proposed ISA strategy looks for 14 free processors, it starts scanning

from the first row of the mesh system at node (0, 0) and it finds a negative value -5 at node (0, 0), this means that the ISA strategy skips the next 5 contiguous processors until it reaches to the node (0, 5), and then it allocates the processors from the node (0, 5), with a positive value of 2. Then the ISA strategy stores the node (0, 5) in *R* array and the positive value 2 in request array then increment the size of *R* array and request array with parameters *k* and *re* respectively. Then, subtracts the positive value 2 at node (0, 5), from an incoming job request, results in a new job request of 12 processors, then the strategy goes to the next row, and starts scanning the row at node (1, 0), which it finds a positive value 6, first it checks the contiguity with the previous allocation at node (0, 5), and the contiguity is exist between the nodes (0, 5) and (1, 5). Then the ISA strategy allocates the next 6 contiguous processors by storing the node (1, 5) in *R* array and the positive value 6 in request array then increment the size of *R* array and request array with parameters *k* and *re* respectively again, and then subtracts the positive value 6 at node (1, 0), from the job request of 12 processors to get a job request of 6 processors, then it goes to the next upper row and starts scanning it at node (2, 0), which it finds a positive value 6 at node (2, 0), which is contiguous with the previous row, then it stores the node (2, 0) in *R* array and the positive value 6 in request array then increment the size of *R* array and request array with parameters *k* and *re* respectively. Then, it subtracts the positive value 6 at node (2, 0) from the job request of 6 processors to get a job request of 0 processors, which means that the incoming job request of 14 processors is allocated and the allocation is done and the final value of parameters (R array, request array, k and re), become R[0, 5, 1, 0, 2, 0], request[2, 6, 6], k = 6 and re = 3. The incoming job request is allocated as shown in Figure 3.2-(b).

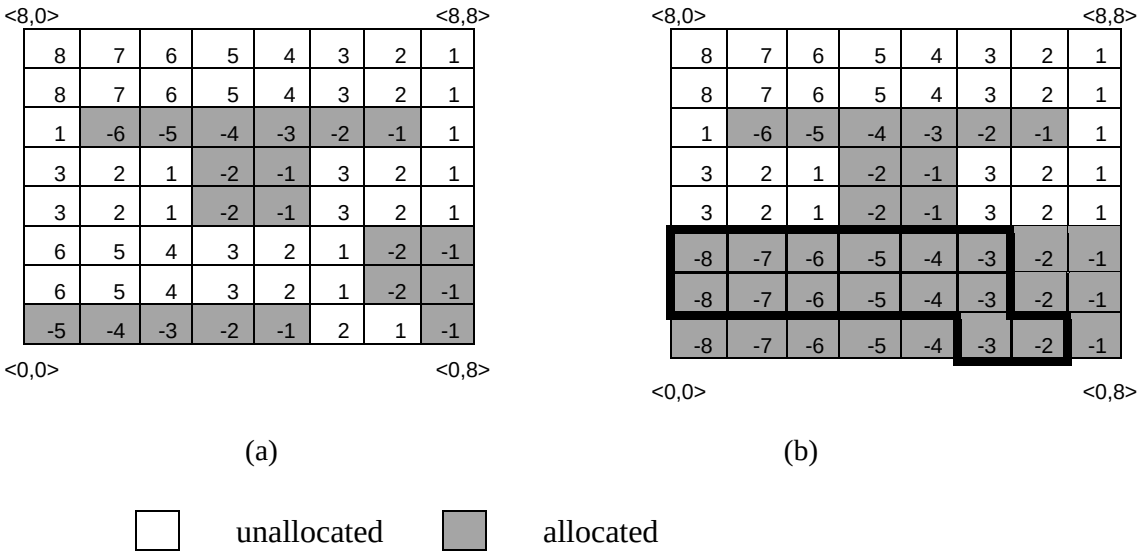

Figure 3.2: (a) a mesh (8×8) before allocating a job of (2×7), (b) the mesh after allocating a job request of 14 processors.

Example 2: Consider the mesh system shown in Figure 3.3-(a), and assuming that a job request of *J(2×3)* arrives to the mesh system, then the proposed ISA strategy starts looking for the 6 free processors. Initially, it starts scanning the mesh system from the node (0, 0), it finds the negative value of -8, then it skips the next eight contiguous processors until it reaches the node (0, 8), which represents the last column in the mesh system which is not less than the width of the mesh system (*W*), so the proposed ISA strategy jumps to the next upper row (row 1), and it starts scanning it from the node (1, 0), which finds a negative value of (-8), then it skips the next eight contiguous processors until it reaches the node (1, 8), which represents the last column in the mesh system which is also not less than the width of the mesh system (*W*), so it jumps again to the next upper row (row 2), and starts the scanning process until it finds a negative value (-8), then it skips the next eight contiguous processors until it reaches the node (2, 8), which represents the last column in the mesh system which is again not less than the width of the mesh (*W*), then it jumps to the next upper row (row 3), and starts scanning process from the node (3, 0), until it finds a positive value of (3), then the

allocation process is starting by storing the node (3, 0) in *R* array and the positive value 3 in request array and then increment the size of *R* array and request array with parameters *k* and *re* respectively. Then, it subtracts the positive value of 3 at node (3, 0) from an incoming job request of 6 processors to get a job request of 3 processors, then it jumps again to the next upper row (row 4), skipping the free processors in row 3 without allocation because there is no contiguity with the previous allocated processors. In row 4, the first 3 free processors are allocated and then it stores the node (4, 0) in R array and the positive value 3 in request array and then increment the size of *R* array and request array with parameters *k* and *re* respectively. Then, the subtraction process is repeated by subtracted the positive value 3 at node (4, 0) from the job request of 3 processors to get 0 processors, which means that the job request is allocated and allocation is done and the final value of parameters (R array, request array, k and re), become R[3, 0, 4, 0], request[3, 3], k = 4 and re = 2. The incoming job request is allocated as shown in figure 3.3-(b).

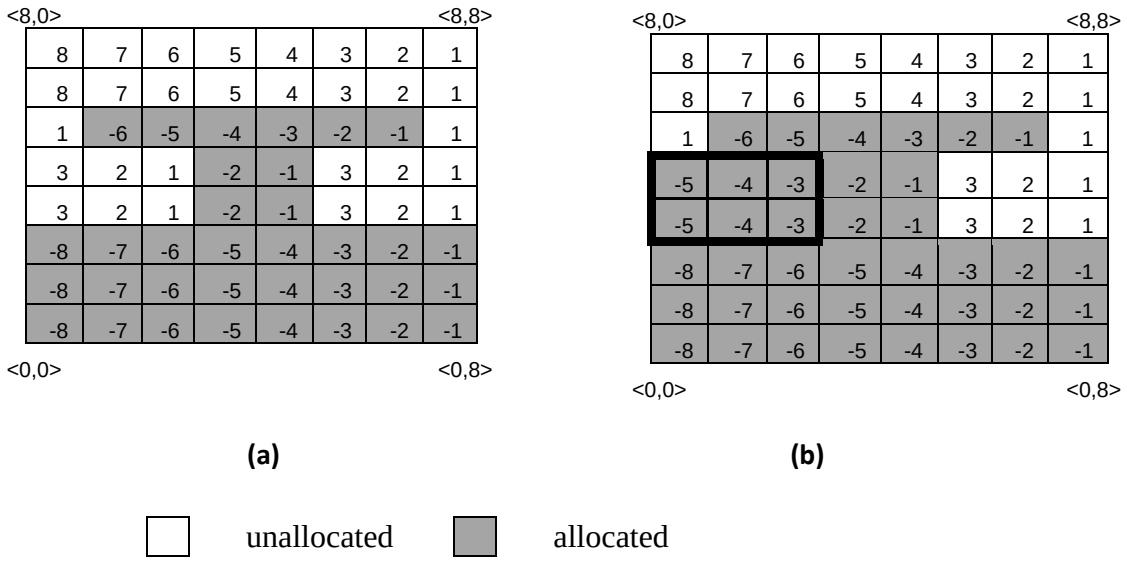

Figure 3.3: (a) a mesh (8×8) before allocating a job of (2×3), (b) the mesh after allocating the job request of 6 processors.

Example 3: Consider the mesh system shown in Figure 3.4-(a), assuming that a job request of *J(3×3)*, arrives to the mesh system, then the proposed ISA strategy starts looking for the 9 free processors. Firstly it starts scanning the mesh system from the node (0, 0), it finds the positive value of 3, then the allocation process is starting by storing the node (0, 0) in *R* array and the positive value 3 in request array and then increment the size of *R* array and request array with parameters *k* and *re* respectively. Then it subtracts the positive value of 3 at node (0, 0) from an incoming job request of 9 processors to get a job request of 6 processors, then it jumps again to the next upper row (row 1), and it starts scanning it from the node (1, 0), which finds a negative value of (-8), which means that there is no contiguity with previous free processors in previous row (row 0), so, the ISA strategy reset all parameters (*k*, *re*, *R* array, request array, JobSize), to their initial values, then restart scanning from previous row (row 0) at node (0, 3), and it finds a negative value -5 at node (0, 3), this means that the ISA strategy skips the next 5 contiguous processors until it reaches to the node (0, 8), which represents the last column in the mesh system, which is not less than the width of the mesh (*W*), so the proposed ISA strategy jumps to the next upper row (row 1), and it starts scanning it from the node (1, 0), which finds a negative value of (-8), then it skips the next eight contiguous processors until it reaches the node (1, 8), which represents the last column in the mesh system which is not less than the width of the mesh (*W*), then it jumps to the next upper row (row 2), and starts scanning process from the node (2, 0), which finds a negative value of (-5), then it skips the next five contiguous processors until it reaches the positive value 3 at the node (2, 5), then the allocation process is starting by storing the node (2, 5) in *R* array and the positive value 3 in request array then increment the size of *R* array and request array with parameters *k* and *re* respectively. Then it subtracts the positive value of 3 at node (2, 5) from a job request

of 9 processors to get a job request of 6 processors, then it jumps again to the next upper row (row 3), and it starts scanning it from the node (3, 0), which finds a negative value of (-8), which means that there is no contiguity with previous free processors in row (2), so the ISA strategy reset again the parameters (*k*, *re*, *R*, request, JobSize), to their initial values, then restart scanning from previous row (row 2) at node (2, 8), which represents the last column in the mesh system which is not less than the width of the mesh (*W*), so the proposed ISA strategy jumps to the next upper row (row 3), and it starts scanning it from the node (3, 0), which finds a negative value of (-8), then it skips the next eight contiguous processors until it reaches the node (3, 8), which represents the last column in the mesh system which is not less than the width of the mesh (*W*), then it jumps to the next upper row (row 4), and starts scanning process from the node (4, 0), which finds a negative value of (-5), then it skips the next five contiguous processors until it reaches the node (4, 5), which contains the positive value of 3, then the allocation process is starting by storing the node (4, 5) in *R* array and the positive value 3 in request array and then increment the size of *R* array and request array with parameters *k* and *re* respectively. Then, it subtracts the positive value of 3 at node (4, 5) from a job request of 9 processors to get a job request of 6 processors, then jumps to the next upper row (row 5), and starts scanning process from the node (5, 0), which finds a positive value of 3, but still there is no contiguity with previous free processors in previous row (row 4), so the ISA again reset the parameters (*k*, *re*, *R*, request, JobSize), to their initial values, then restart scanning from previous row (row 4) at node (4, 8), which represents the last column in the mesh system which is not less than the width of the mesh (*W*), then it jumps to the next upper row (row 5), and starts scanning process from the node (5, 0), it finds the positive value of 3, then the allocation process is starting by storing the node (5, 0) in *R* array and the positive value 3 in request array and then increment

the size of *R* array and request array with parameters *k* and *re* respectively. Then it subtracts the positive value of 3 at node (5, 0) from a job request of 9 processors to get a job request of 6 processors, then jumps to the next upper row (row 6), and starts scanning process from the node (6, 0), which finds a negative value of (-8), which means there is no contiguity with previous free processors in row (5), so, the ISA strategy reset again the parameters (*k*, *re*, *R*, request, JobSize), to their initial values, then restart scanning from previous row (row 5) at node (5, 3), and it finds a negative value of (-5), then it skips the next five contiguous processors until it reaches the node (5, 8), which represents the last column in the mesh system which is not less than the width of the mesh (*W*), then it jumps to the next upper row (row 6), and starts scanning process from the node (6, 0), which finds a negative value of (-8), then it skips the next eight contiguous processors until it reaches the node (6, 8), which represents the last column in the mesh system which is not less than the width of the mesh (*W*), then it jumps to the next upper row (row 7), and starts scanning process from the node (7, 0), which finds a negative value of (-8), then it skips the next eight contiguous processors until it reaches the node (7, 8), which represents the last column in the mesh system which is not less than the width of the mesh (*W*), then it jumps to the next upper row (row 8), which represents the last row (8) in the mesh system, which means the allocation is failed and the job request is queued. As shown in the figure 3.4-(a), the allocation algorithm (ISA) failed in allocation the free processors to the job request, while there is enough free processors in a mesh for an incoming job *J(3×3)*, this is because there is no contiguity between free processors. So, the ISA strategy in this situation will allocate the free processors according to the row major indexing scheme starting from the node (0, 0). In this case the algorithm finds the positive value of 3, then the allocation process is started by storing the node (0, 0) in *R* array and the positive value 3 in request array

then it increments the size of *R* array and request array with parameters *k* and *re* respectively. Then it subtracts the positive value of 3 at node (0, 0) from an incoming job request of 9 processors to get a job request of 6 processors, then it skips the next three contiguous processors until it reaches the node (0, 3), and it finds a negative value -5 at node (0, 3), which means that the ISA strategy skips the next 5 contiguous processors until it reaches to the node (0, 8), which represents the last column in the mesh system which is not less than the width of the mesh system (*W*), so the proposed ISA strategy jumps to the next upper row (row 1), and it starts scanning it from the node (1, 0), which finds a negative value of (-8), then it skips the next eight contiguous processors until it reaches the node (1, 8), which represents the last column in the mesh system which is not less than the width of the mesh system (*W*), then it jumps to the next upper row (row 2), and starts scanning process from the node (2, 0), which finds a negative value of (-5), then it skips the next five contiguous processors until it reaches the node (2, 5) that contains the positive value of 3, then the allocation process is started by storing the node (2, 5) in *R* array and the positive value 3 in request array and then increment the size of *R* array and request array with parameters *k* and *re* respectively. Then it subtracts the positive value of 3 at node (2, 5) from a job request of 6 processors to get a job request of 3 processors, then it skips the next three contiguous processors until it reaches the node (2, 8), which represents the last column in the mesh system which is not less than the width of the mesh system (*W*), then it jumps to the next upper row (row 3), and starts scanning process from the node (3, 0), which finds a negative value of (-8), then it skips the next eight contiguous processors until it reaches the node (3, 8), which represents the last column in the mesh system which is not less than the width of the mesh system (*W*), then it jumps to the next upper row (row 4), and starts scanning process from the node (4, 0), which finds a negative value of (-5),

then it skips the next five contiguous processors until it reaches the node (4, 5) that has the positive value of 3, then the allocation process is started by storing the node (4, 5) in *R* array and the positive value 3 in request array and then incrementing the size of *R* array and request array with parameters *k* and *re* respectively. Then it subtracts the positive value of 3 at node (4, 5) from a job request of 3 processors to get a job request of 0 processors, which means that the job request is allocated and allocation is done. and the final value of parameters (R array, request array, k and re), become R[0, 0, 2, 5, 4, 0], request[3, 3, 3], k = 6 and re = 3. The incoming job request is allocated as shown in figure 3.4-(b).

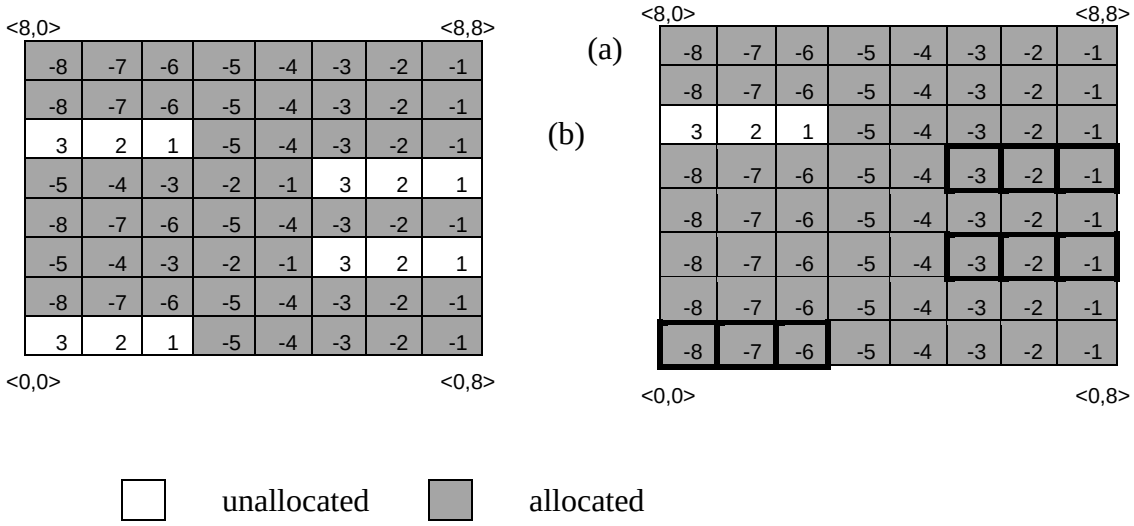

Figure3.4: (a) a mesh (8×8) before allocating job (3×3), (b) the mesh after allocating the job request of 9 processors.

Allocation in ISA is implemented by the algorithm outlined in Figure 3.5.

```
Procedure ISA_Allocate():
{
 j              // the number of rows in a mesh.
 i              // the number of columns in a mesh.
```

r // a two dimensional array that represents all processors in the mesh.
R // a one dimensional array that is used to store the coordinates (j, i) for each processor in a mesh that should be allocated for an incoming job request.
Request // a one dimensional array that is used to store the job size in one row.
k // counter used for R array.
re // counter used for request array.
id // identification number for each job
Found // a boolean parameter that takes a false value initially.
array of nodes // an array of nodes where each node contains the parameters (id, k, re, R array, request array).
JobSize // a number of processors that is required for an incoming job

start:

calculate the number of free processors (freecount)
if (freecount < number of requested processors)
{
fail to allocate the requested processors;
}

A1: if (i >= width of the mesh){
Jump to the first processor in the next higher row according to the row major indexing scheme.
}

If (j >= height of the mesh)
{
go to A2;
}

If (the value in r[j][i] < 0){
jump to the next positive value and
go to A1;
}

Store the coordinates (j, i) in R array

if (JobSize <= value in r[j][i]){
Store the jobsize in request array;
increment the counter (re);
found = true
go to A3;
}

Store the value in r[j][i] in request array
increment the counter (re);
updates the JobSize to JobSize = JobSize - r[j][i];
Go to the first positive value in upper row and check the contiguity with the processors that are allocated for the incoming job request in previous row

If there is a contiguity return (j, i), otherwise return (j, -1)

If (i == -1){
Return to the first elements in R-array and start scanning from the next positive value with the main JobSize
go to A1;
}

goto A1;

```
A2: if (not found)
 {
Reset the parameters JobSize, k and re to the initial values.
        allocate the requested processors randomly based on the major indexing
scheme;
        found = true
 }

 A3: if (found)
 {
        update the r-array by giving the allocated processors a negative value
        store the parameters (id, k, re, R, request) in the array of nodes.
        Return (1);
 }
 else
        return (0);
 }
```

Figure 3.5: Outline of the ISA Non-Contiguous Allocation Strategy.

3.2.2 ISA De-Allocation

The de-allocation is the reverse process of allocation. When the job terminates its running, the de-allocation process starts and de-allocate the allocated processors for the terminating job, and then updates the values in r-array from the negative values (allocated processors) to positive values (free processors), and as a result the processors that were allocated for a job become now free to be used again by other job requests (Fan Wu, Ching-Chi Hsu, and Chou, Li-Ping, 2003).

Example 1. Consider the mesh system state shown in Figure 3.6-(a), and assuming that a job *J(2×7)* is finished, then the allocated processors for this job will be freed to be used again. The de-allocation process in the ISA strategy starts working as the following. Firstly, the de-allocation process compares the identification number (id) for the finished job with the id's of the allocated jobs, which were stored in an array of nodes so as to extract the parameters (*k*, *re*, *R* array, request array).

Note, each one of the coordinates of *(j, i)* in R array meets a one value in request array, which means that the first coordinates of *(j, i*) in *R* array meets the first value in request array.

In figure 3.6-(a), we show how the job *J(2×7)* is de-allocated. Firstly, the de-allocation process compares the *id* of the de-allocating job with the id's that were stored in an array of nodes through the allocation process, so as to extract the parameters (*k*, *re*, *R*, request), we find the *R* array that has the coordinates [(0, 5), (1, 0), (2, 0)], and the request array that has the values (2, 6, 6). The de-allocation process starts from the last coordinates in *R* array down to the first coordinates depending on the parameter *k*, and so in request array depending on the parameter *re*. The de-allocation process starts from $k = 4$ after decremented it by 2 to get the coordinates (2, 0), while the last value in request array is 6, which means that the 6 contiguous allocated processors should be de-allocated starting from the coordinates (2, 0), then the parameter *k* is decremented by 2 until it becomes 2 to get the coordinates (1, 0) in R array, and parameter *re* is decremented by 1 until it becomes 1, to get the previous value in request array which is also 6, which means that the 6 contiguous allocated processors should be de-allocated starting from the coordinates (1, 0), then the parameter *k* is decremented by 2 until it becomes 0 to get the coordinates (0, 5) in R array, and parameter *re* is decremented by 1 until it becomes 0, to get the first value in request array which is 2, which means that the 2 contiguous allocated processors should be de-allocated starting from the coordinates (0, 5), then the parameter *k* is decremented by 2 until it becomes -2 which is not greater than 0, which means that the de-allocation process is completed and as shown in figure 3.6-(b), the 14 allocated processors become free to be used again by other job requests.

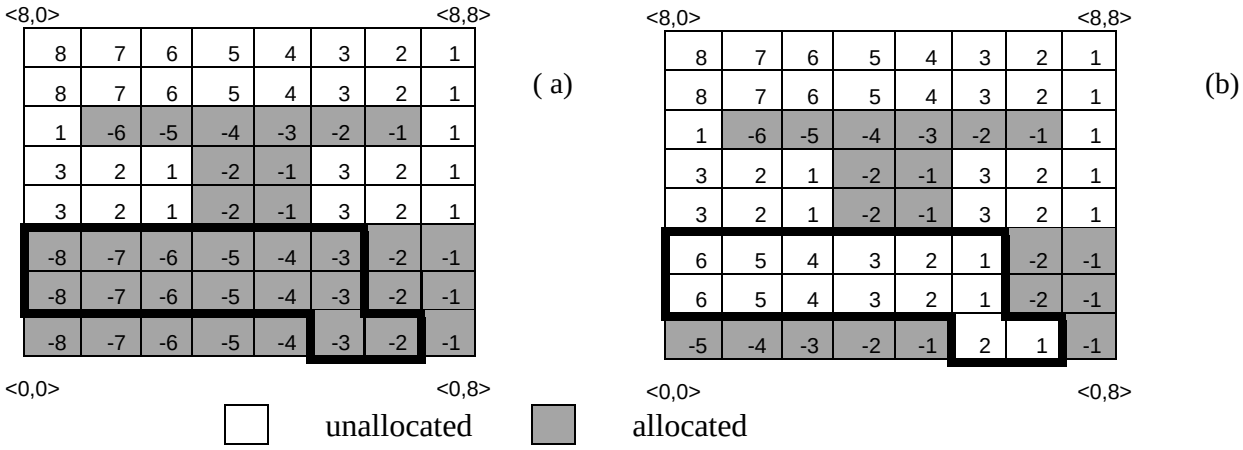

Figure 3.6: (a) a mesh (8×8) before de-allocating a job of (2×7), (b) the mesh after de-allocating a job request of 14 processors.

De-Allocation in ISA is implemented by the algorithm outlined in Figure 3.7.

```
Procedure ISA_DE-Allocate ():
{
count1                    // number of rows that is stored in R array
len1, count2, last        //  loop parameters


start :

if ( id of the de-allocated job is in the array of nods) {
        extract the required parameters (id, k, re, R, request):

        while (k > 0) {

                count1= last row stored in R array; // extract using k-2
                i = last column stored in R array;   // extract using k-1
                next= the next processor after the contiguous allocated processors in each row
                last= the values in request array; // extract using re-1;

                if (i + last < width and next is positive){
                        len1= r[count1][last];
                }

                else
                        len1=0;

                for (count2=1 to last)
                {
                        Change the value of r[count1][last-count2] to positive value starting
                        from 1;
                }
```

```
                    If (previous processor is allocated and i-1>= 0){

                            m=1;

                            while (previous processor is allocated and i-m >= 0){
                                    store the value of –m in r[count1][i-m];
                                    increment m by 1;
                            }
                    }

                    Else if (previous processor is not allocated and i-1>= 0){
                            m=1;
                            len2= value in r[count1][i];

                            while (previous processor is not allocated and i-m >= 0){
                                    increment the value in r[count1][i-m] by m+len2;
                                    increment m by 1;
                            }
                    }

                    Decrement k by 2;
                    If k greater than 0 then
                    i=R[k-1];
                    Decrement re by 1;
}

A4: set the value in node where the id of de allocated job is found to 0;
break;
}

}
```

Figure 3.7: Outline of the ISA Non-contiguous De-Allocation Strategy.

3.3. Performance Evaluation

In this section, the results of the proposed ISA allocation strategy that have been carried out from simulations are presented and compared against those of the Paging(0) (Lo, et al., 1997), MBS (Lo, et al., 1997), FF (Zhu, Y. H, 1992) and Random (Lo, et al., 1997) allocation strategies. We use C++ language to implement the ISA allocation and de-allocation algorithms, and integrated the software into the ProcSimity simulation tool for processor allocation and job scheduling in highly parallel systems (ProcSimity V4.3, 1997, Windisch, K. Miller, J. V. and Lo, V, 1995). The number of processors requested

by each job is generated by uniform distribution, where job widths and heights are uniformly distributed over the range from 1 to the mesh side lengths (ProcSimity V4.3, 1997). Jobs are scheduled using the First Come First Served (FCFS) scheduling strategy. The execution time of a job is the time it takes to finish communicating. The execution times of jobs depend on the time needed for flits to be routed through the nodes, packet size, the number of messages that are sent, message contention and distances messages traverse. The interconnection network for message routing is wormhole flow control with ordered *XY* routing, where the number of bytes in each message (Message size) is 8. Each simulation run consists of 1000 completed jobs per run and the number of runs is varied to get a confidence level of 95% and relative errors do not exceed 5%.

A job remains in the system until an iteration of the communication pattern is completed. We implemented four specific communication patterns when messages are exchanged among the processors that are allocated to a job (Bani-Mohammad, S. Ould-Khaoua, M. and Ababneh, I, 2007, Kumar, et al., 2003). The first one is one-to-all, where a randomly selected processor in each job sends a message to each other processors in the same job. The second communication pattern is all-to-all, where each processor in a job sends a message to all other processors in the same job. This communication pattern causes much message collision and is known as the weak point for non-contiguous allocation algorithms [Bani-Mohammad, S. Ould-Khaoua, M. and Ababneh, I, 2007, Yoo, B.-S. Das, C.-R, 2002]. The third communication pattern is the random communication pattern, where a randomly selected processor sends messages to randomly selected destination within the set of processors allocated to the same job. The fourth communication pattern is near neighbor, where each processor in a job sends a message to its neighbors (up, down, left and right). The table below presents other

parameters that were used in the simulator. The main performance parameters used are the average turnaround time of jobs and the mean system utilization. The independent variable in the simulation is the system load; it is defined as the inverse of the mean inter-arrival time of jobs.

Table 3.1: The System Parameters Used in the Simulation Experiments

Simulator Parameter	**Values**
Dimensions of the Mesh Architecture	16×16
Allocation Strategy	Paging(0), MBS, Random, FF, ISA
Scheduling Strategy	FCFS
Job Size Distribution	Uniform: Job widths and legths are uniformly distributed over the range from 1 to the mesh side lengths.
Inter-arrival Time	Exponential with different values for the mean. The values are determined through experimentation with the simulator, ranged from lower values to higher values.
Number of Runs	The number of runs should be enough so that the confidence level is 95% that relative errors are below 5% of the means. The number of runs ranged from dozens to thousands.
Number of Jobs per Run	1000

Turnaround time

In Figures (3.8-3.11), the average turnaround time of jobs are plotted against the system load for the one to all, all to all, random, and near neighbor communication patterns using the FCFS scheduling strategy. Figure 3.8 shows that the non-contiguous allocation strategies considered in this thesis (Random, Paging(0), MBS and ISA) perform better than the contiguous allocation strategy (FF). This is because the non-contiguous allocation strategies eliminate both internal and external fragmentation and thus improves system performance in terms of average turnaround time of jobs. The simulation results show that the performance of the Paging(0), MBS and ISA allocation strategies is better than that of the Random allocation strategy. This is because the allocated processors in Random are allocated randomly which increases the distances between allocated processors and hence increases the probability of the interference among job's messages which in turn increases the contention and thus degrades the system performance in terms of average turnaround time of jobs.

In Figure 3.9, the performance of the ISA allocation strategies is better than that of the non-contiguous allocation strategies (Random, Paging(0), MBS) and contiguous allocation strategy (FF). The simulation results show that the performance of the ISA allocation strategies is better than that of the Paging(0), Random and MBS allocation strategies. This is because the distances between the allocated processors in Paging(0), Random and MBS are more than those in ISA which increases the probability of the interference among job's messages which increases the contention and thus degrades the system performance in terms of average turnaround time of jobs. For example, the average turnaround time of ISA are (84%), (56%), (48%) and (77%) of that of Paging (0), Random, MBS and FF, respectively, when the load is (0.0001) jobs/time unit.

Figure 3.10, shows that the proposed allocation strategy (ISA) performs better than all other strategies in terms of average turnaround time of jobs using the random communication pattern. For example, the average turnaround time of ISA are (96%), (63%), (97%) and (52%) of that of Paging (0), Random, MBS and FF, respectively, when the load is 0.07 jobs/time unit.

Figure 3.11, shows that the contiguous allocation strategy (FF) performs better than the non-contiguous allocation strategies (Random, Paging(0), MBS, and ISA) for heavy system loads. This is because in near neighbor communication pattern, each node in the mesh sends a message to its neighbors (up, down, right, left), and because the allocated sub-mesh in the FF contiguous allocation strategy is in the rectangular shape, so there is no any inter-job interference and thus reduces the communication overhead. This improves the system performance for the FF contiguous allocation strategy as compared to that of the non-contiguous allocation strategies. The performance of Random is the worst over all allocation strategies. This is because the allocated processors in Random are far from each other, which increases the distances the messages traverse, and hence increases the probability of interference among job's messages, which in turn degrades the system performance in terms of average turnaround time of jobs. For example, in Figure 3.11, the average turnaround times of FF are (0.6%), (0.09%), (0.6%) and (0.4%) of that of Paging (0), Random, MBS and ISA, respectively, when the system load is 0.005 jobs/time unit.

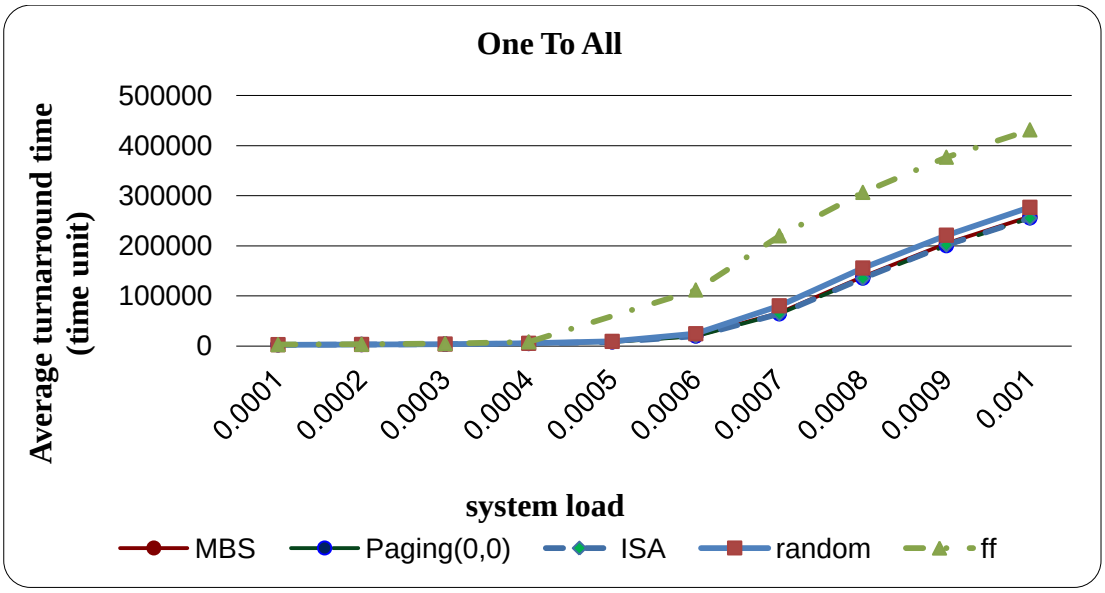

Figure 3.8: Average turnaround time vs. system load using uniform distribution in a 16x16 mesh with communication pattern one to all.

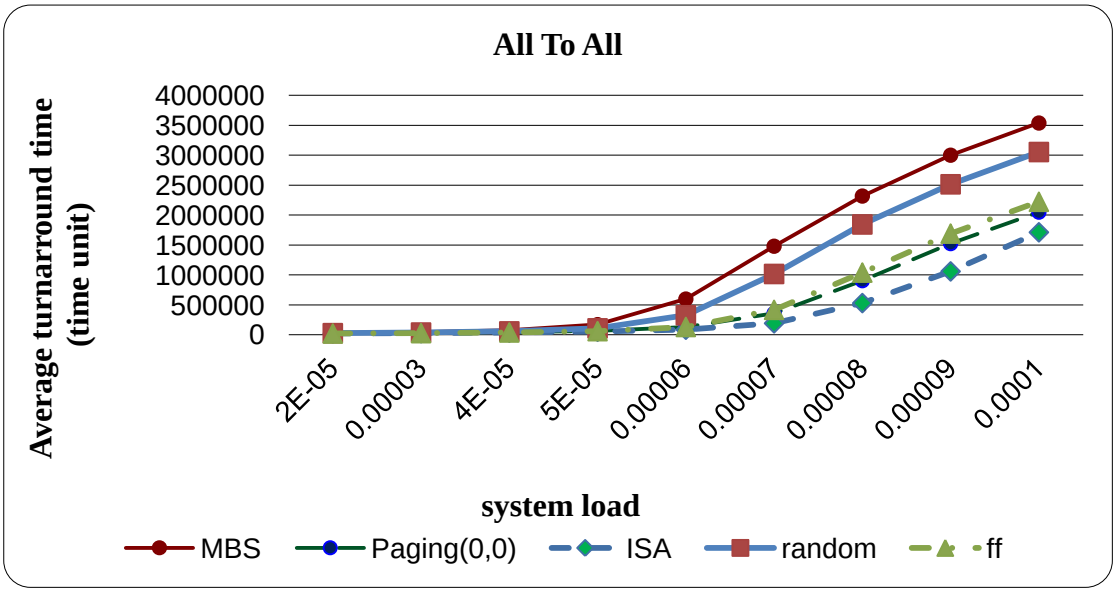

Figure 3.9: Average turnaround time vs. system load using uniform distribution in a 16x16 mesh with communication pattern all to all.

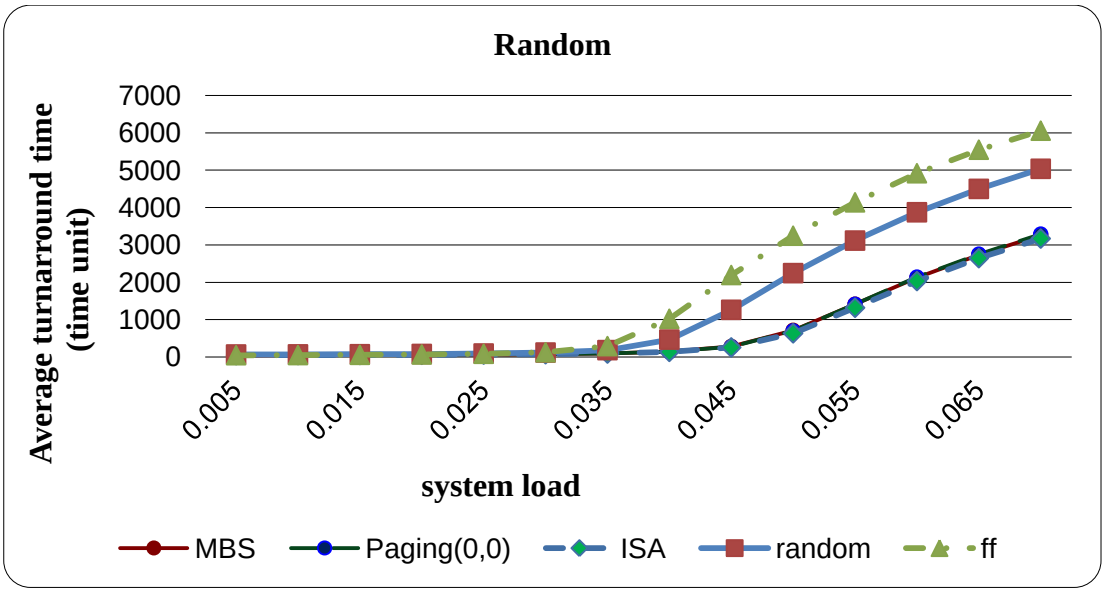

Figure 3.10: Average turnaround time vs. system load using uniform distribution in a 16x16 mesh with communication pattern random.

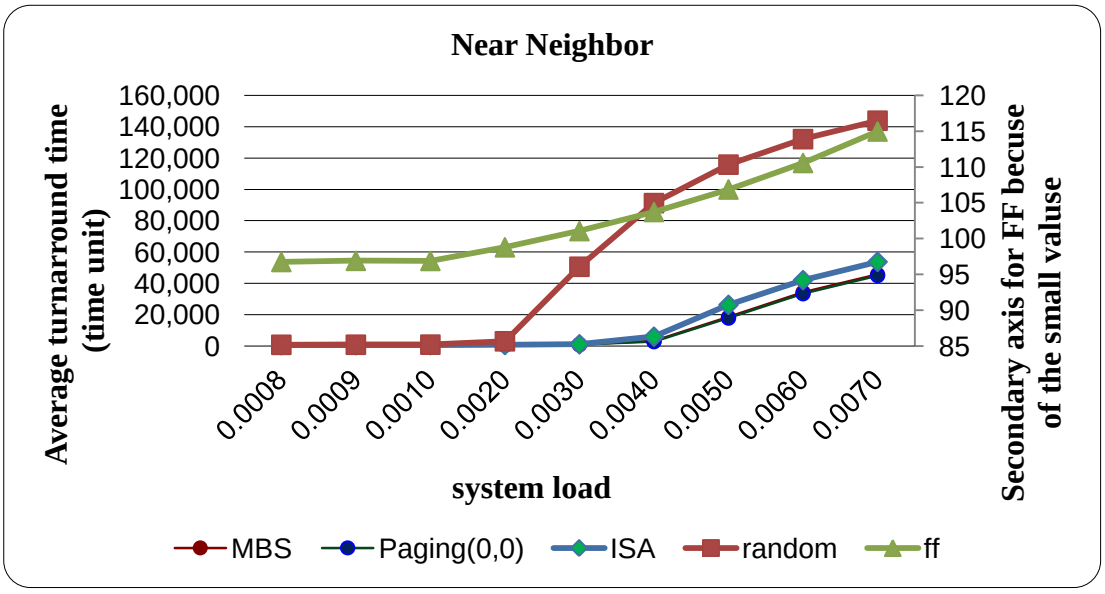

Figure 3.11: Average turnaround time vs. system load using uniform distribution in a 16x16 mesh and communication pattern near neighbor.

Utilization

In Figures (3.12-3.15), the system utilization of jobs are plotted against the system load for the one to all, all to all, random, and near neighbor communication patterns using the FCFS scheduling strategy. The results in these figures show that the non-contiguous allocation strategies considered in this thesis (Random, Paging(0), MBS and ISA) perform better than the contiguous allocation strategy (FF). This is because contiguous allocation produces high external fragmentation, which makes allocation is less likely to succeed. As a consequent, the mean system utilization is lower. For heavy system loads, the utilization for all non-contiguous allocation strategies is approximately the same because the non-contiguous allocation strategies considered in this thesis have the same ability to eliminate internal and external processor fragmentation. They always succeed to allocate processors to a job when the number of free processors is greater than or equal to the allocation request. Table 3.2 shows the results of utilization taken for some non-contiguous allocation strategies.

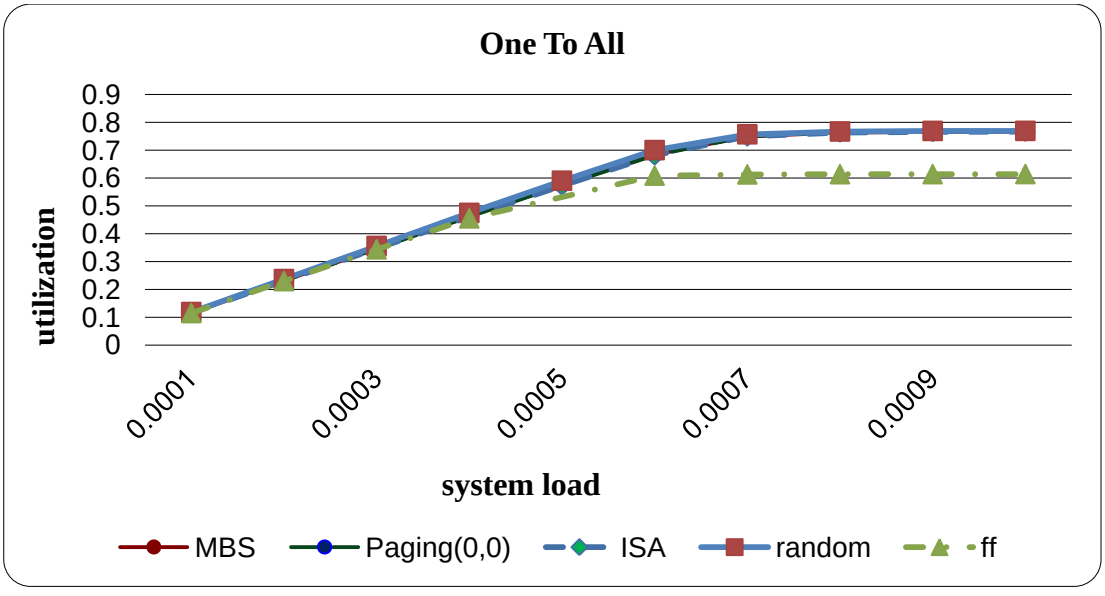

Figure 3.12: Average utilization vs. system load using uniform distribution in a 16x16 mesh with communication pattern one to all.

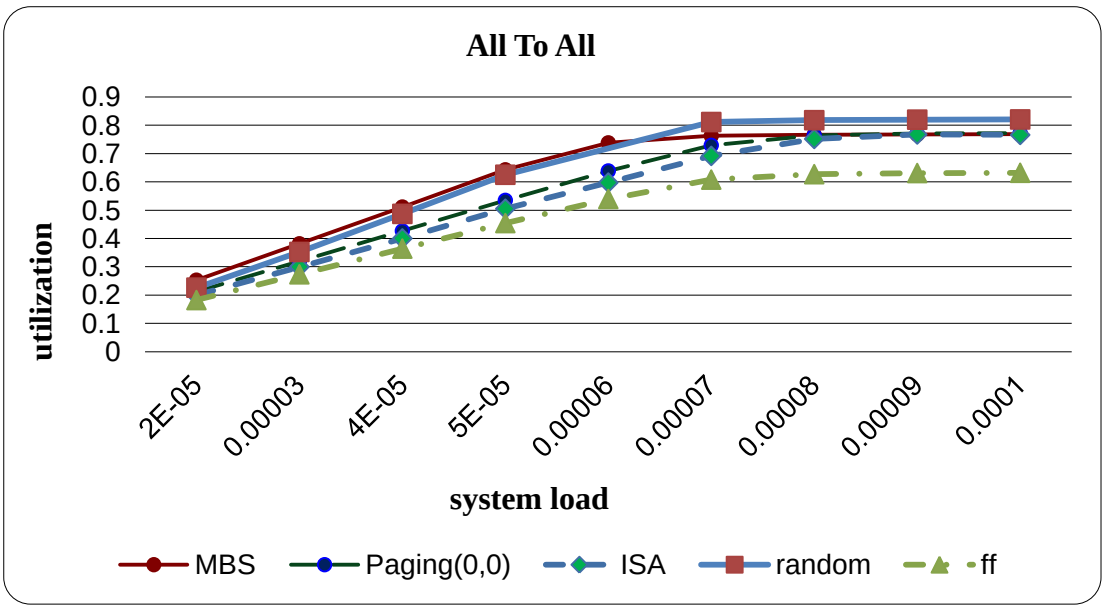

Figure 3.13: Average utilization vs. system load using uniform distribution in a 16x16 mesh with communication pattern all to all.

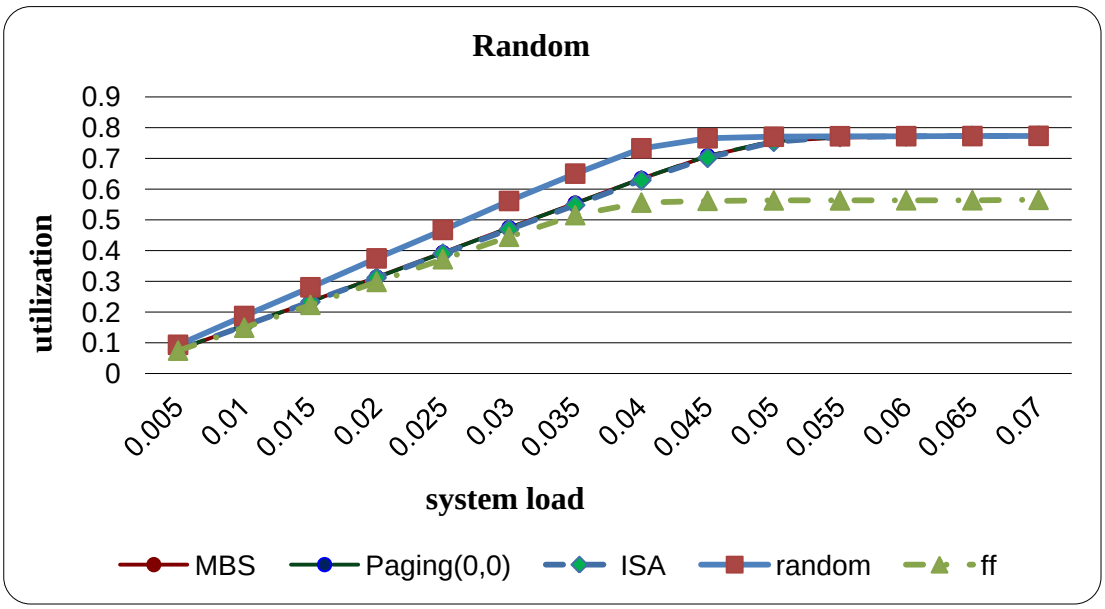

Figure 3.14: Average utilization vs. system load using uniform distribution in a 16x16 mesh with communication pattern random.

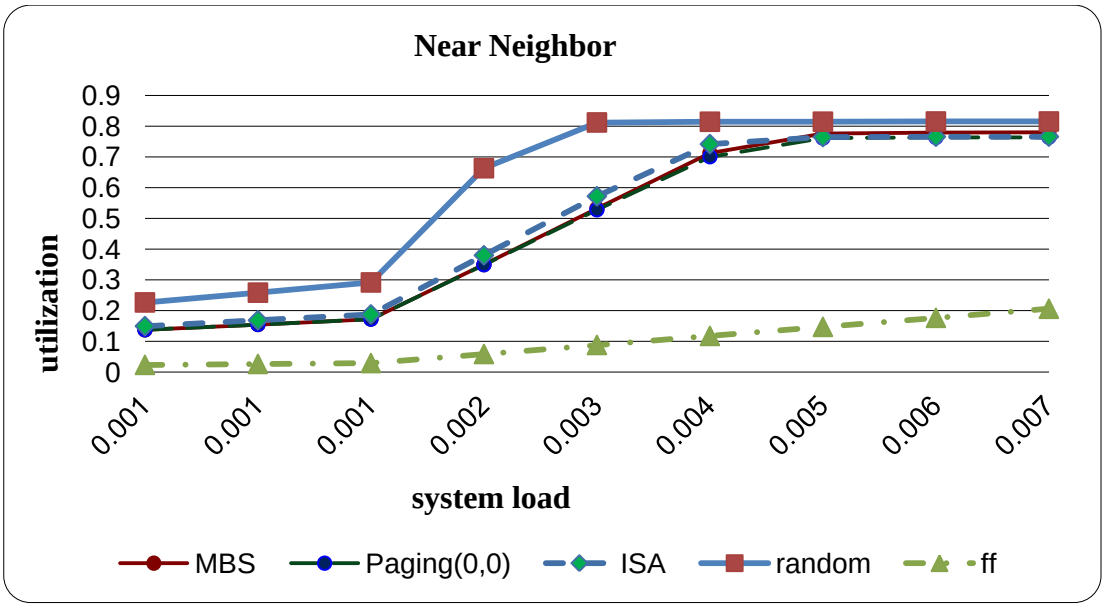

Figure 3.15: Average utilization vs. system load using uniform distribution in a 16x16 mesh with communication pattern near neighbors.

Table. 3.2: The results of utilization taken for some non-contiguous allocation and contiguous allocation strategies for heavy system loads.

Algorithm	Utilization			
	One to all	All to all	Random	Near neighbors
First Fit	61%	63%	56.5%	20.6%
MBS	76.6%	76.8%	77%	78%
Paging(0)	76.6%	77%	77%	76.4%
Random	76.8%	82%	77%	81.5%
ISA	76.6%	76.6%	77%	76.6%

3.4 Conclusions

In this chapter, the performance of the non-contiguous allocation for 2D mesh-connected multicomputer has been investigated. To this time, we have suggested a new non-contiguous allocation strategy, referred to as Irregular Shape Allocation (ISA) strategy, which differs from the earlier non-contiguous allocation strategies in the method used for allocating the requested job. The ISA strategy allocates the job requests based on the free processors available for allocation regardless of the shape of the allocated sub-mesh. The ISA allocation depends on the number of free processors that is sufficient for the requested job. Moreover, it maintains the contiguity as much as possible.

The performance of ISA strategy was compared against that of existing non-contiguous and contiguous allocation strategies. ISA performs well in terms of system utilization and job turnaround time as compared to earlier non-contiguous allocation strategies for the communication patterns, one to all and random and also it is superior for all to all communication pattern. However, the performance of ISA is not better than that of the previous strategies when the Near Neighbor communication pattern is considered.

Chapter 4

Conclusions and Future works

4.1 Summary of the Results

The aim of the present research is the development of a new non-contiguous allocation strategy for 2D mesh-connected multicomputer. We summarize below the major contribution in this study and the simulation results of this study.

The results of the previous research suggested (Fan Wu, Ching-Chi Hsu, and Chou, Li-Ping, 2003, Lo, et al., 1997) that new non-contiguous allocation strategies for mesh-connected multicomputer are needed. The motivation for the development of a new non-contiguous allocation strategy for the 2D mesh network has been driven from the observation that the existing non-contiguous allocation strategies suggested for the 2D mesh achieve complete sub-mesh recognition capability provided that the allocated shapes should be in regular rectangular shapes, which affect the system performance in terms of average turnaround time negatively. Motivated by these observations, a new non-contiguous allocation algorithm, referred to as Irregular Shape Allocation (ISA) has been proposed. The proposed ISA strategy allocates the free sub-mesh in 2D mesh-connected multicomputer regardless of the shape of the allocated sub-mesh.

Extensive simulation experiments under a variety of system loads have been carried out in order to compare the performance of the proposed ISA strategy against well-known non-contiguous allocation strategies (MBS, Paging(0), and Random) (Lo, et al., 1997), and the contiguous FF allocation strategy (Zhu, Y. H, 1992). Our results show that the

performance of ISA is very close to that of the best non-contiguous allocation strategies (e.g. Paging(0), MBS, Random) when communication patterns, one to all is considered. Moreover, ISA performs much better than the previous non-contiguous allocation strategies (e.g. Paging(0), MBS, Random) and contiguous FF allocation strategy, when all to all and random communication pattern are considered.

4.2 Directions for the Future Work

The aim of ISA strategy is to increase the system utilization and reduce the turnaround time as much as possible. The results of this thesis have shown that the ISA allocation strategy have superior performance against that of the previous non-contiguous allocation strategies suggested Paging(0) (Lo, et al., 1997), MBS (Lo, et al., 1997), Random (Lo, et al., 1997), for the 2D mesh network. So, it would be interesting to adapt the proposed ISA non-contiguous allocation strategy to be applicable for the 3D mesh-connected multicomputer.

References

[1] Ababneh, I and Fraij, F. Folding contiguous and non-contiguous space sharing policies for parallel computers, Mu'tah Lil-Buhuth wad-Dirasat, Natural and Applied Sciences Series, vol. 16, no. 3, pp. 9-34, 2001.

[2] Ababneh, I. An efficient free-list submesh Allocation Scheme for two-dimensional mesh-connected multicomputers, Journal of Systems and Software, vol. 79, no. 8, pp. 1168-1179, August 2006.

[3] Asanovic, K. RasBodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Keutzer, David A Patterson, William Lester Plishker, JoneShalf, Samuel Webb Williams, Katherine A. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. University of California, Berkeley. Technical Report No.UCB/EECS-2006-183. 18- December-2006.

[4] Attari S and Isazadeh, A. Processor Allocation in Mesh Multiprocessors Using a Hybrid Method, Proceedings of the Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'06), 0-7695-2736-1/06, 2006.

[5] Bani-Mohammad, S. Ould-Khaoua, M. Ababneh, I. and Machenzie, L. Comparative Evaluation of the Non-Contiguous Processor Allocation Strategies based on a Real Workload and a Stochastic Workload on Multicomputers, Proceedings of the 13th International Conference on Parallel and Distributed Systems (ICPADS'07) , vol. 2, pp. 1-7, IEEE, Hsinchu, Taiwan, December 5-7, 2007.

[6] Bani-Mohammad, S. Ould-Khaoua, M. Ababneh, I. and Machenzie, L. 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), Minneapolis, Minnesota, USA, IEEE Computer Society Press, vol. 2, pp. 41-48, 2006.

[7] Bani-Mohammad, S. Ould-Khaoua, M. Ababneh, I. and Mackhenzie, Lewis M. Comparative evaluation of contiguous allocation strategies on 3D Mesh Multicomputers, Journal of System and Software, vol. 82, no. 2, pp. 307-318, 2009.

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

[9] Bunde, D. P. Leung, V. J. and Mache, J. Communication Patterns and Allocation Strategies, Sandia Technical Report SAND2003-4522, Jan. 2004.

[10] Chang, C.-Y. and Mohapatra, P. Performance improvement of allocation schemes for mesh-connected computers, Journal of Parallel and Distributed Computing, vol. 52, no. 1, pp. 40-68, 1998.

[11] Chiu, G.-M. Chen, S.-K. An efficient submesh allocation scheme for two-dimensional meshes with little overhead, IEEE Transactions on Parallel & Distributed Systems, vol. 10, no. 5, pp. 471-486, 1999.

[12] Chuang, P.-J. Tzeng, N.-F. Allocating precise sub-meshes in mesh connected systems, IEEE Transactions on Parallel and Distributed Systems, vol. 5, no. 2, pp. 211-217, 1994.

[13] Cray, Cray XT3 Datasheet, available at: http://www.craysupercomputer.com/downloads/CrayXT3/CrayXT3_Datasheet.pdf, 15-april-2015.

[14] Fan Wu, Ching-Chi Hsu, and Chou, Li-Ping. Processor allocation in mesh multiprocessors using the leapfrog method, IEEE transactions on parallel and distributed system, vol. 14, no. 3, pp.276-289 , March 2003.

[15] Gara A., Blumrich M., Chen D., Chiu G.,Coteus P., Giampapa M., Haring R., Heidelberger P., Hoenicke D., Kopcsay G., Liebsch T., Ohmacht M., Steinmacher B., Takken T., and Vranas P., "Overview of the Blue Gene/L System Architecture," IBM journal of Research and Development, vol. 49, no. 2, pp. 195-212, 2005.

[16] Gottlieb, Allan. Almasi, George S. Highly parallel computing. Redwood City, Calif.: Benjamin/Cummings. ISBN 0-8053-0177-1, 1989.

[17] Intel Crop, Paragon XP/S Product Overview, available at: http://books.google.com/books/about/Paragon_XP_S_product_overview.html?id=qkGNkgAACAAJ, 15-april-2015.

[18] Kumar, V. Grama, A. Gupta, A. and Karypis, G. Introduction To Parallel Computing, The Benjamin/Cummings publishing Company, Inc. Redwood City, California, 2003.

[19] Li, K. Cheng, K.-H. A Two-Dimensional Buddy System for Dynamic Resource Allocation in a Partitionable Mesh Connected System, Journal of Parallel and Distributed Computing, vol. 12, no. 1, pp. 79-83, 1991.

[20] Lo, V. and Mache, J. Job Scheduling for Prime Time vs. Non-prime Time, Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'02), pp. 488-493, 2002.

[21] Lo, V. Windisch, K. Liu, W. and Nitzberg, B. Noncontiguous processor allocation algorithms for mesh connected multicomputers, IEEE Transactions on Parallel and Distributed Systems, vol. 8, no. 7, IEEE Press, Piscataway, NJ, USA, pp. 712-726, July 1997.

[22] Mache, J. Lo, V. and Garg, S. Job Scheduling that Minimizes Network Contention due to both Communication and I/O, Proceedings of the 14th International Parallel and Distributed Processing Symposium(IPDPS'00), pp. 457-463, 2000.

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

[24] Moghaddam S. and Naghibzadeh, M. A New Processor Allocation Strategy Using ESS (Expanding Square Strategy), Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'06), 1066-6192/06, 2006.

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

[26] Seo, K.-H. Fragmentation-Efficient Node Allocation Algorithm in 2D Mesh-Connected Systems, Proceedings of the 8th International Symposium on Parallel Architecture, Algorithms and Networks (ISPAN'05), IEEE Computer Society Press, Washington, DC, USA, 7-9 December, pp. 318-323, 2005.

[27] Windisch, K. Miller, J. V. and Lo, V. “ProcSimity: an experimental tool for processor allocation and scheduling in highly parallel systems”, Proceedings of the Fifth Symposium on the Frontiers of Massively Parallel Computation (Frontiers'95), IEEE Computer Society Press, Washington, USA, pp. 414-421, 6-9 Feb 1995.

[28] Yoo, B.-S. Das, C.-R. A Fast and Efficient Processor Allocation Scheme for Mesh-Connected Multicomputers, IEEE Transactions on Parallel & Distributed Systems, vol. 51, no. 1, pp. 46-60, 2002.

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

الملخص

تصنف استراتيجيات تخصيص المعالجات في الحواسيب المتوازية الى صنفين: استراتيجيات التخصيص المتجاور واستراتيجيات التخصيص غير المتجاور. تشترط إستراتيجيات التخصيص المتجاور التجاور في ما بين المعالجات المخصصة لمهمة معينة وان تكون المعالجات المخصصه بنفس شكل الشبكة التي تربط ما بين المعالجات في النظام، وهذا يؤدي بدوره الى حدوث ما يسمى بمشكلة الكسيرات، والتي تؤثر سلباً على اداء النظام مما يؤدي الى زيادة الوقت الذي تقضيه المهام في النظام وكذلك تقليل نسبة استغلال المعالجات في النظام. جاءت استرتيجيات التخصيص غير المتجاور لحل مشلكة الكسيرات، حيث انها لا تشترط التجاور ما بين المعالجات المخصصة لمهمة معينه مما يؤدي بالتالي الى تحسين اداء النظام فيما يتعلق بمعدل وقت مكوت المهام في النظام وكذلك معدل استغلال المعالجات في النظام، وعلى الرغم من الزيادة في التزاحم ما بين رسائل المعالجات المخصصة في النظام نتيجة استخدام التخصيص غير المتجاور، الا ان هذا النوع من الاستراتيجيات يؤدي الى التخلص من مشكلة الكسيرات وبالتالي زيادة استغلال معالجات النظام.

معظم استراتيجيات التخصيص غير المتجاور في متعددات الحواسيب الشبكية الموجودة حالياً تعاني من مشكلة الكسيرات والتداخل بين الرسائل ضمن المهام المختلفة داخل النظام بالإضافة إلى حاجتها إلى الشكل المنتظم (المستطيل) في التخصيص اذا كان هناك اي تجاور ما بين المعالجات، لذلك فقد اقترحنا في هذه الرسالة إستراتيجية تخصيص غير متجاور جديدة تسمى إستراتيجية الشكل الغير منتظم (Irregular Shape Allocation Strategy) (ISA) والتي تحد من مشكلة الكسيرات وتخفف من التداخل بين الرسائل ضمن الشبكة، حيث كانت الفكرة الرئيسية من الاستراتيجية الجديدة أنه للحصول على درجة من التجاور ما بين المعالجات المخصصة لمهمة

معينه، فإن ذلك لا يشترط ان يكون شكل شبكة المعالجات المخصصة للمهمة منتظماً (على شكل مستطيل) كما هو في الاستراتيجيات السابقة الأخرى, حيث أن الشبكة الفرعية المخصصة يمكن أن تكون على أي شكل (منتظم أو غير منتظم) وهذا يؤدي بدوره إلى تحسين اداء النظام من حيث معدل مكوث المهام في النظام وكذلك معدل استغلال المعالجات في النظام.

تمت مقارنة اداء الخوارزمية الجديدة (ISA) مع اداء استراتيجيات التخصيص المتجاور ممثلة باستراتيجية الـ (First Fit) والغير متجاور ممثلة بالاستراتيجيات التالية (Random, Paging(0), Multiple Buddy Strategy) باستخدام المحاكاة، وقد أظهرت النتائج بإن اداء الاستراتيجية المقترحة (ISA) هو قريب جداً من اداء خوارزميات التخصيص غير المتجاور السابقة (Paging(0) and Multiple Buddy Strategy)، وذلك عند استخدام نمط التراسل (One to All)، في حين انها افضل من استراتيجية التخصيص غير المتجاور (Random) وكذلك افضل من اداء استراتيجية التخصيص المتجاور (First Fit)، كما أظهرت النتائج أن اداء الاستراتيجية المقترحة (ISA) افضل من اداء الاستراتيجيات الاخرى المتجاورة والغير متجاورة عند استخدام نمطي التراسل (Random and All to All)، في حين أظهرت النتائج أن اداء استراتيجيات التخصيص المتجاور ممثلة بالـ (First Fit) افضل من اداء استراتيجيات التخصيص غير المتجاور عند استخدام نمط التراسل (Near Neighbor).