جامعة آل البيت

Al al-Bayt University

An Efficient Processor Allocation Algorithm for 2D Mesh Connected Multicomputers

خوارزمية فعالة لتخصيص المعالجات في متعددات الحواسيب ثنائية الابعاد

By

Abeer Bassam Al Shdefat

Supervisor

Prof. Saad Bani-Mohammed

Co-supervisor

Prof. Ismail Ababneh

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

Deanship of Graduate Studies

Al al-Bayt University

جامعة آل البيت

عمادة الدراسات العليا

نموذج رقم (1)

نموذج تفويض

أنا عبير بسام عبدالكريم الشديفات

افوض جامعة آل البيت بتزويد نُسخ من رسالتي، للمكتبات أو المؤسسات أو الهيئات أو الأشخاص عند طلبهم حسب التعليمات النافذة في الجامعة.

التوقيع: **التاريخ:** **2019/1/7**

جامعة آل البيت

عمادة الدراسات العليا

نموذج رقم (2)

نموذج اقرار والتزام بقوانين جامعة آل البيت وانظمتها وتعليماتها لطلبة الماجستير والدكتوراه.

انا عبير بسام عبدالكريم الشديفات. الرقم الجامعي: 1220901009

تخصص: علم الحاسوب. كلية: الامير حسين بن عبدالله الثاني لتكنولوجيا المعلومات.

أُعلنُ بأني قد التزمت بقوانين جامعة آل البيت وانظمتها وتعليماتها وقراراتها السارية المفعول المتعلقة بإعداد رسائل الماجستير والدكتوراه عندما قمت شخصياً بإعداد رسالتي / اطروحتي بعنوان:

An Efficient Processor Allocation Algorithm for 2D Mesh Connected Multicomputers

خوارزمية فعالة لتخصيص المعالجات في متعددات الحواسيب ثنائية الابعاد

وذلك بما ينسجم مع الأمانة العلمية المتعارف عليها في كتابة الرسائل والأطاريح العلمية. كما أنني أُعلن بأن رسالتي هذه غير منقولة أو مستلة من رسائل أو أطاريح أو كتب أو أبحاث أو أي منشورات علمية تم نشرها أو تخزينها في أي وسيلة اعلامية، وتأسيساً على ما تقدم فأنني اتحمل المسؤولية بأنواعها كافة فيما لو تبين غير ذلك بما فيه حق مجلس العمداء في جامعة آل البيت بإلغاء قرار منحي الدرجة العلمية التي حصلت عليها وسحب شهادة التخرّج مني بعد صدورها دون أن يكون لي الحق في التظلم أو الأعتراض أو الطعن بأي صورة كانت في القرار الصادر عن مجلس العمداء بهذا الصدد.

التوقيع التاريخ: 2019/1/7

Committee Decision

This Thesis An Efficient Processor Allocation Algorithm for 2D Mesh Connected Multicomputers was Successfully Defended and Approved on 7th Jan. 2019.

Examination Committee	Signature
Prof. Saad Bani-Mohammed, (Supervisor) Dep. of Computer Science, Al al-Bayt University bani@aabu.edu.jo	.....
Prof. Ismail Ababneh, (Co-Supervisor) Dep. of Computer Science, Al al-Bayt University bani@aabu.edu.jo	.....
Prof. Omar Shatnawi, (Member) Dep. of Computer Science, Al al-Bayt University dromali@aabu.edu.jo	.....
Dr. Akram Hamarsheh, (Member) Dep. of Computer Science, Al al-Bayt University hamarshi@aabu.edu.jo	.....
Prof. SHADI A. ALJAWARNEH, (External Member) Dep. of Software Engineering, Jordan University of Science and Technology saaljawarneh@just.edu.jo	.....

iv

Dedication

This thesis is dedicated to the soul of my mother (GOD has mercy on her), my father, my husband, my daughters, my brothers, and my sisters for their support, encouragement, and ultimate love

.

Acknowledgments

AL Humdillah who gave me the ability to achieve this work. It toke me awhile, but finaly I have compleat it. I am grateful to my main supervisor, Prof. Saad Bani Mohammed, whose patience, kindness, and encouragement helped me in all the times to complete this thesis. Also i am extremely grateful to my co-supervisor, Prof. Ismail Ababneh, whose help, suggestions as well as his experience, have invaluable to me.

Thank you my family, from the bottom of my heart for your love, incrgment, and support. I would never be glad as I am without all of you. You are priceless to me.

Abeer AL Shdefat.

Table of contents

List of Figures

List of Figures

ix

List of Tables

List of Abbreviations

Abbreviation	Meaning
AS	Adaptive Scan
BF	Best Fit
FCFS	First-Come-First-Served
FF	First-Fit
FS	Frame Sliding
LSSA	L-Shaped Sub-mesh Allocation
NAS	Neighbor Allocation Strategy
2DBS	Two Dimensional Buddy System

An Efficient Processor Allocation Algorithm for 2D Mesh Connected Multicomputers

A Master Thesis By

Abeer Bassam Al Shdefat

Supervisor:

Prof. Saad Bani-Mohammed

Co-supervisor:

Prof. Ismail Ababneh

Department of Computer Science, Al al-Bayt University, 2019

Abstract

In the 2D mesh connected multicomputers, two types of strategies for processor allocation were proposed: the contiguous and Non-Contiguous. In contiguous strategies, the processors allocated have to be physically adjacent, and in some strategies must have the same shape as the mesh topology. Contiguous allocation schemes suffer from fragmentation problem, which has direct influence on the performance of the system when, average response time and system utilization are considered. In Non-Contiguous allocation schemes the request of the job can be executed on some separated smaller sub-meshes rather than waiting for a submesh of the requested sub-mesh, shape and size is available on the system. There are many devised Non-Contiguous allocation strategies that vary on the rate of contiguity kept among the allocated small sub-meshes to the job request in the mesh-system.

In this research, we have proposed a new Non-Contiguous processor allocation strategy for 2D-mesh-connected multicomputers, referred to as Neighbor Allocation Strategy (for short NAS). The NAS allocates a number of smaller sub-meshes, which have a rate of contiguity among them, the requests from jobs. NAS rebuilds the job request to be accommodated in to the possible free sub-meshes in the system and always allocates the job request contiguously to remove internal and external fragmentation, and hence maximize system utilization and minimize average response time.

Using simulation, performance of NAS, the contiguous allocation scheme FF, and the Non-Contiguous allocation scheme LSSA were compared. The outcomes show that the performance of NAS in term of average response time is encouraging than that of all other allocation schemes when the one_to_all communication pattern is used for the 2 distributions considered in this work considering job size. On the other hand, when All-to-All and near neighbor communication patterns are used the FF has the superior performance over all other schemes because it allocates rectangular sub-mesh to the job request, which minimize interference between massages so it minimize communication overhead, while NAS results are improved than LSSA. Furthermore, results show that NAS maximize system utilization more than the other two strategies FF and LSSA, this is because of its ability to remove both of internal as well as external fragmentation.

خوارزمية فعالة لتخصيص المعالجات في متعددات الحواسيب ثنائية الابعاد

رسالة ماجستير قُدمت من قبل:

عبير بسام الشديفات

المشرف:

أ.د سعد بني محمد

المشرف المشارك

أ.د اسماعيل العبابنه

قسم علم الحاسوب، جامعة آل البيت، 2019م

الملخص

تقسم استراتيجيات التخصيص للمعالجات في الحواسيب المتوازية الى نوعين: استراتيجيات التخصيص المتجاور واستراتيجيات التخصيص غير المتجاور. في التخصيص المتجاور يكون التجاور بين المعالجات المخصصة لمهمة معينة شرطا اساسيا، كما تشترط بعض الاستراتيجيات ان يكون الشكل المخصص نفس شكل الشبكة التي تربط بين المعالجات في النظام، وهذا بدوره يؤدي الى ظهور ما يعرف باسم مشكلة الكسيرات، والتي تؤثر سلبيا على اداء النظام من حيث تقليل نسبة استغلال المعالجات في النظام وزيادة الوقت الذي تقضيه المهام في النظام.

جاءت استراتيجيات التخصيص غير المتجاور لحل مشكلة الكسيرات، حيث انها لا تشترط التجاور ما بين المعالجات المخصصة لمهمة معينة مما يؤدي الى تحسين اداء النظام بما يتعلق بوقت المكوث للمهام في النظام وكذلك معدل استغلال المعالجات في النظام، بالرغم من ان هذا النوع من التخصيص قد يؤدي الى زيادة التزاحم ما بين الرسائل بين المعالجات المخصصة للمهمة، الا انه يساعد على التخلص من مشكلة الكسيرات وبالتالي يزيد من نسبة استغلال معالجات النظام. وتعاني معظم استراتيجيات التخصيص غير المتجاور من مشكلة الكسيرات بالإضافة الى حاجتها الى الشكل المنتظم المشابه لشكل شبكة النظام، لذلك فقد اقترحنا في هذه الرسالة استراتيجية تخصيص غير متجاور جديدة تسمى استراتيجية الجار للتخصيص (Neighbor Allocation Strategy) والتي تقلل مشكلة الكسيرات في النظام، حيث تعمل الاستراتيجية المقترحة على تخصيص مجموعة فرعية من الاشكال المخصصة بحيث تحافظ على درجة من التجاور بين المعالجات المخصصة للمهمة وهذا بدوره يؤدي الى تحسين في اداء النظام من حيث معدل استغلال المعالجات في النظام ومعدل مكوث المهام في النظام.

تمت مقارنة اداء الخوارزمية الجديدة (NAS) مع اداء استراتيجية التخصيص المتجاور ((First Fit والغير متجاور (L-Shape Submesh Allocation Strategy) باستخدام المحاكاة، وقد اظهرت النتائج تفوق الخوارزمية المقترحة (NAS) على باقي الاستراتيجيات بما يتعلق بمعدل استغلال المعالجات في النظام بسبب قدرتها على تقليل مشكلة الكسيرات في النظام. اما فيما يخص معدل وقت المكوث للمهام في النظام فقد تفوقت الاستراتيجية المقترحة (NAS)عند استخدام نمط التراسل (One to All) ، في حين تفوقت استراتيجية التخصيص المتجاور عند استخدام (All to All) و (Near Neighbor) على استراتيجيات التخصيص الغير متجاور، بينما تفوقت الاستراتيجية المقترحة (NAS) على استراتيجية (L-Shape Submesh Allocation Strategy) عند استخدام انماط التراسل (All to All) و (Near Neighbor).

Chapter 1
Introduction

Overview

The evolution on technology has made the concept of parallel computers viable, as an efficient solution, where a high performance is needed with a low cost relatively. This encourages the development of programs that use distributed resources, to correspond with the high performance demands from new applications such as engineering, science, and intensive data applications like data bases and real time applications (Foster, 1995).

A parallel computer is a group of processors that is enabled of working together on a cooperative manner, to solve a problem. Using Decomposition, which is the process of dividing the problem into smaller parts that may be potentially executed in parallel; the available resources are allocated to the sub problems in order to solve the whole problem eventually (Foster, 1995; Kumar, Grama, Gupta, & Karypis, 2003).

Using parallel computing, we are able to decrease the execution time to solve huge problems, as well as the cost; processors are available with reasonable price. This needs parallel programs, which are programs that can be executed in multiprocessors (Foster, 1995; Bani Mohmmad, 2008).

Parallel computers can be classified according to the way used for communication among processors in the system as the following: machines communicate through shared-memory model or distributed-memory model. In shared-memory, known as multiprocessors, the shared-memory address space is used for communication among processors, where all the processors have access to the memory. In the other hand, the distributed-memory systems communicate using an interior communication network, where massages are exchanged among processors using the interconnected network, this type is called multi-computers (Bani Mohmmad, 2008).

The main responsibilities of interior communication networks are to deliver a fast as well as reliable communications among different processors. Generally, two categories of interconnection networks: indirect connection networks and direct connection networks. In indirect connection networks, also known as dynamic, multiple stages of intermediate switches and communication links are used to form paths, which connect processors in the system. The bus networks, the crossbar switches and the multistage networks are examples of such type of networks. On the other hand, in direct connection networks, the nodes are connected to each other directly by wires (i.e., point-to-point connection), also called static networks. The star network, the mesh networks, the k-ary n-cube networks, and the hypercube networks are examples of such type of networks (Kumar, Grama, Gupta, & Karypis, 2003; Bani Mohmmad, 2008).

A 2D mesh interconnection network is an example of direct networks, in which each processer is connected directly to its neighbors. Many large- scale multiprocessing systems have been used the 2D mesh topology. This is due to its simplicity, regularity, as well as its ease of implementation, scalability (i.e., it can be scaled up as the system need), and for the benefit from locality to other neighbor processors in the communication manner (Ding, Bhuyan, 1993; Yoo, & Das, 2002; Kumar, Grama, Gupta, & Karypis, 2003; Bani Mohmmad, 2008; Ababneha, Bani Mohmmad, & Ould Khaoua, 2010).

Meshes are appropriate implementation for a number of applications like image processing, matrix multiplications and any other application that can get benefit from mesh structure. Also, it was used in the implementation of some multiprocessors as in the IBM-BlueGene(L), Intel Touchstone Delta, and Intel Paragon (Bani Mohmmad, 2008; Bani Mohammad & Ababneh, 2013).

An example of a 5×5 2D-mesh is presented in Figure 1.1, where free processors are denoted with white circles and allocated processors are denoted by black circles.

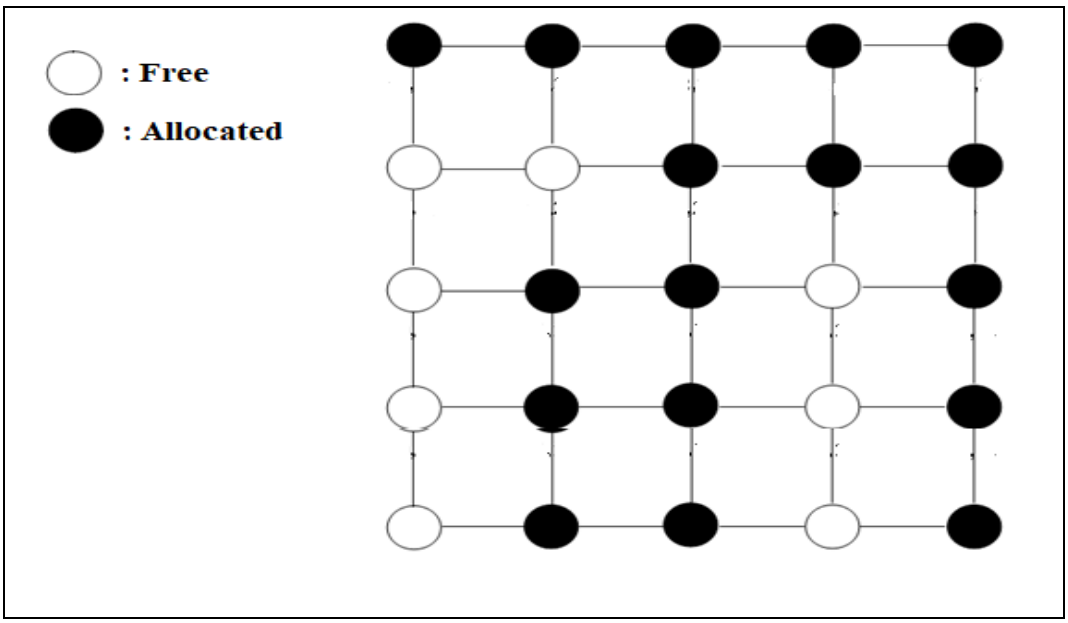


Figure 1-1: An example of a 5x5 2D mesh.

Definition 1: A 2D Mesh $M(a, b)$, is composed of $a \times b$ processors, where a the width and b is its height. Each processor can be identified by two coordinates (x, y), where $0 \leq x < a$ and $0 \leq y < b$. A processor is connected to its neighbors by bidirectional communication channels (Seo, & Kim, 2003; Bani Mohmmad, 2008).

Definition 2: In a 2D mesh $M(a,b)$, a sub-mesh $S(\alpha,\beta)$, is a 2D sub-mesh which consists of processors that are a sub set of the $M(a,b)$ processors, where $0 < \alpha \leq a$ and $0 < \beta \leq b$, the size of this sub-mesh is $\alpha \times \beta$. This sub-mesh can be denoted by the (x,y,x',z'), where (x,y) is the bottom-left corner of the submesh, which is defined as the base-node of the submesh, and (x',z') is the top-right corner of the sub-mesh, which is called the end node of the sub-mesh (Seo, & Kim, 2003).

Definition 3: In 2D mesh $M(a,b)$, a fit sub-mesh $S\ (\alpha',\beta')$ is an empty submesh, where $\alpha' \leq \alpha$ and $\beta' \leq \beta$ provided that the allocated processors for a job request that needs a sub-mesh of $S(\alpha,\beta)$ (Bani Mohmmad, 2008).

Definition 4: In a 2D mesh $M(a,b)$, any two sub-meshes $S(a,b,c,d)$ and $S'(e,f,g,h)$ are said to be neighbor sub-meshes, if any of the following conditions is true, for any border node; which is the node that has at least one neighbor node that does not belong to the same sub-mesh. Assume that the coordinates of the border node in the $S(a,b,c,d)$ sub-mesh is (z,w), and the coordinates of the border node in the $S'(e,f,g,h)$ sub-mesh is (z',w') (Seo, & Kim, 2003).:

$$z = z' and\ w' = w + 1.$$

$$z = z' and\ w' = w - 1.$$

$w = w' and\ z' = z + 1.$

$w = w' and\ z' = z - 1$

Mesh systems have the property of being divided among different jobs composing set of sub-meshes (Lo, Windisch, Liu, & Nitzberg, 1997; Bani Mohmmad, 2008). Figure 1-2 shows a sub-mesh system, where four jobs are allocated,

for example, Job 2 (3, 2), with the size of 6, and coordinates (2, 2, 4, 3); where the coordinates (2, 2) represents the base node of Job 2 and (4, 3) represents the end node this Job, as stated previously in definition 2. Job 1 is neighbor to Job 2, as definition 4 states; where Job 1 border node is (1, 3) and Job 2 border node is (2, 3) which satisfies the third condition in definition 4, where $3 = 3\ and\ 2 = 1 + 1$. In the same way, Job 2 is a neighbor to Job 3 and Job 4, as shown in figure 1-2. The process of allocation is as important as the de-allocation; when a Job is finished, the state of the processors should be changed from allocated to free, so the same sub-set of processors can be assigned to another job request (Seo, & Kim, 2003; Bani Mohmmad, 2008).

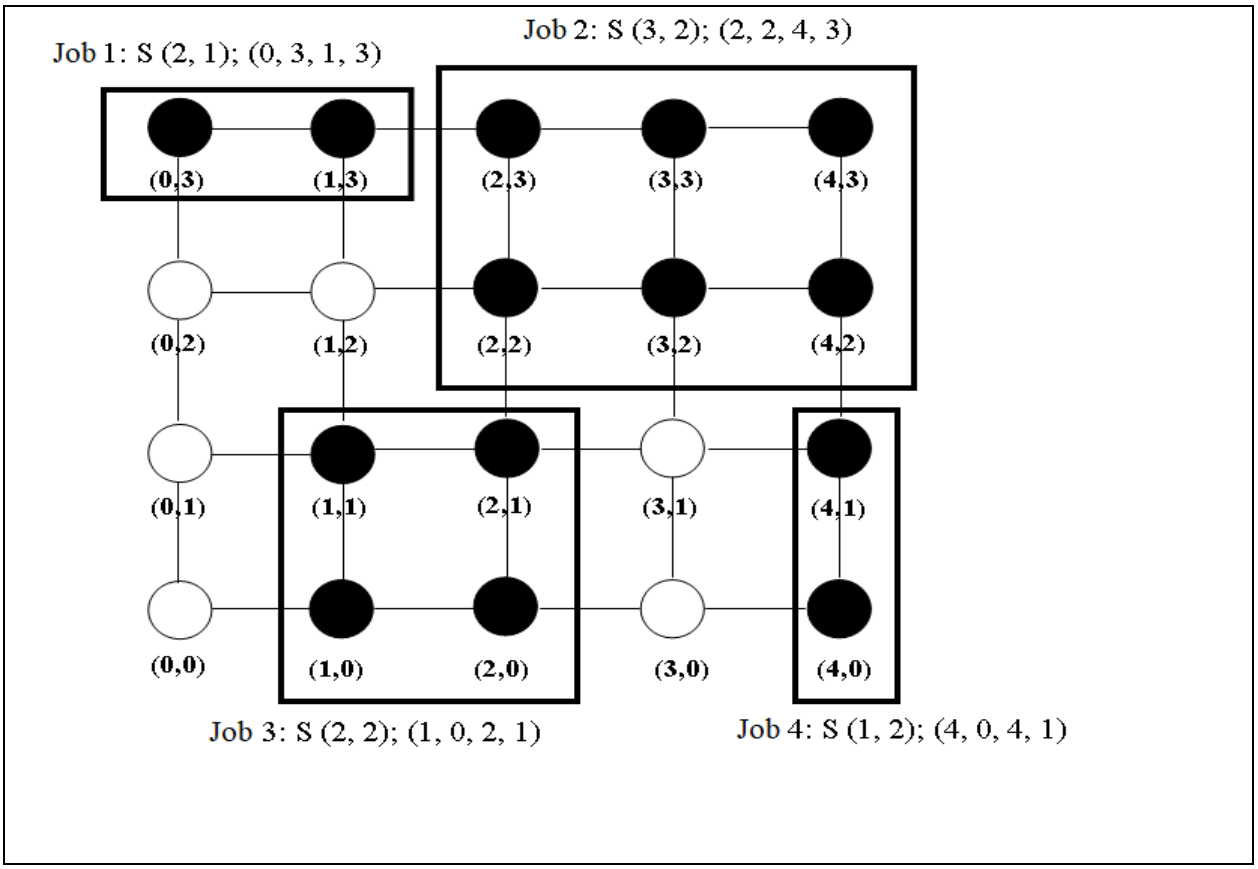


Figure 1-2: An example of a 5x4 2D mesh, having four different jobs.

Processor Allocation

Parallel computer systems made up of a lot of processors connected together using a high-speed interconnection network to provide some powerful computation capabilities to different applications on engineering and scientific researches (Yoo, & Das, 2002; Bani Mohmmad, 2008).

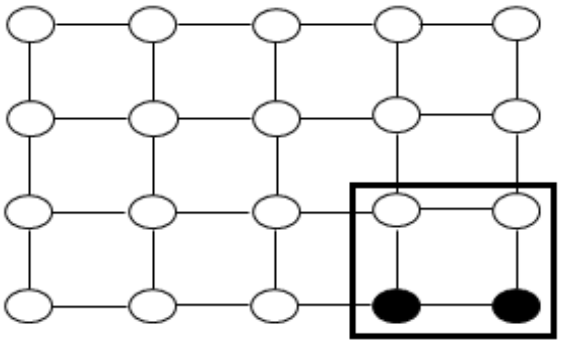
To gain the best from parallel computers, the utilization of such computers is an important performance metric in processor allocation. The job scheduler determines the order of which jobs will be executed, based on its scheduling policy (Lo, Windisch, Liu, & Nitzberg, 1997; Yoo, & Das, 2002; Bani Mohmmad, 2008). When a job arrives to the system, the processor allocator starts trying to find the requested sub-mesh; if a fit free sub-mesh is found, the job is allocated, else the job is sent to the waiting queue. The moment that a job is allocated to a sub-mesh of processors, the processors of the allocated sub-mesh are held by that job for its entire lifetime and then released upon termination of the computations (Bani Mohmmad, 2008).

Two types of allocation strategies have been under investigation by researchers, contiguous and Non-Contiguous allocation strategies (Lo, Windisch, Liu, & Nitzberg, 1997; Seo, & Kim, 2003; Bani Mohmmad, 2008; Bani Mohmmad, Ababneha, 2018).

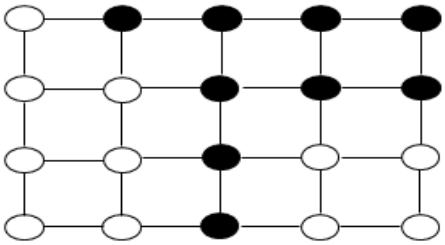
In contiguous allocation, the physical contiguity among allocated processors is necessary and in these strategies (Chuang, & Tzeng, 1994) the shape of the allocated sub-mesh should be the same of the mesh itself (Lo, Windisch, Liu, & Nitzberg, 1997; Seo, & Kim, 2003; Bani Mohmmad, Ababneha, 2018). This restriction imposes the fragmentation problem, which has direct influence on the system utilization

and response time. Fragmentation has 2 types: internal fragmentation and external fragmentation. Internal fragmentation happend if the allocation strategy assigns the job processors further than the actual required number of processors. While external fragmentation exists when an enough number of processors is available, but the allocation strategy is unable to allocate the processors, because of the contiguity condition for example (Lo, Windisch, Liu, & Nitzberg, 1997; Seo, & Kim, 2003; Bani Mohmmad, 2008).

Figure 1-3 A shows an example of internal fragmentation for a job that requests 2 processors but the assumed allocation algorithm allocates 4 processors for this request causing an internal fragmentation of 50%. The processors allocated by the allocation strategy are marked with the black frame and the required number of processors are represented by black circles. Figure 1-3 B shows an example of external fragmentation, assuming that contiguous allocation strategy is used, where a job requests 8 processors, which they are obtainable in mesh multicomputer, while they will not be allocated to the job as they are not contiguous.



A: Internal fragmentation in 5×4 2D mesh



B: External fragmentation in 5×4 2D mesh

Figure 1-3: Internal fragmentation and external fragmentation.

In Non-Contiguous allocation, a sub-mesh shape of the allocated sub-meshes and the physical contiguity among the allocated processors are not necessary; wherever smaller sub-meshes in the system are free, they are allocated to the requested job regardless the contiguity among the allocated processors and also the shape of the allocated sub-meshes (Seo, & Kim, 2003; Bani Mohammad, Ould Khaoua, Ababneh, & Mackhenzie, 2007; Bani Mohmmad, 2008; Bani Mohammad, Ababneh, 2015).

Many previous researchers present many Non-Contiguous allocation schemes that improve the performance of the system in terms of average turnaround time; which is a total time that the job spend in the multicomputer from the arrival until departure (Seo, & Kim, 2003; Bani Mohmmad, 2008), in addition to system utilization. Despite of this improvement, Non-Contiguous allocation suffers from communication overhead, which can be alleviated by preserving a good degree of contiguity among the allocated processors (Seo, & Kim, 2003; Bani Mohammad, Ould Khaoua, Ababneh, & Mackhenzie, 2007; Bani Mohmmad, 2008; Bani Mohammad, Ababneh, 2015; Bani Mohmmad, Ababneha, 2018).

An important characteristic of an allocation strategy is to be recognition complete; where the allocation algorithm has the ability to find a fit sub-mesh requested by a job whenever it is existing in the system. This maximize system utilization and reduces response time (Seo, & Kim, 2003; Bani Mohmmad, 2008).

Some of the well-known allocation strategies were not completely recognition such as Two-Dimensional Buddy System (Lo, Windisch, Liu, & Nitzberg, 1997; Yoo, & Das, 2002; Seo, & Kim, 2003; Bani Mohmmad, 2008), Frame Sliding (Ding, & Bhuyan, 1993), First Fit and Best Fit (Zhu, 1992), while some other allocation strategies are completely recognition but with high allocation overhead such as Adaptive Scan (Chuang, & Tzeng, 1994; Seo, & Kim, 2003; Yoo, & Das, 2002; Bani Mohmmad, 2008).

Motivation and Contributions

Starting from the point that when the processor fragmentation is decreased, the performance of the system is improved in terms of both job turnaround time and utilization, where this is the goal of most previous allocation algorithms (Zhu, 1992; Chuang, & Tzeng, 1994; Lo, & Windisch, Liu, & Nitzberg, 1997; Yoo, & Das, 2002; Seo, & Kim, 2003; Bani Mohmmad, 2008). Also, it was found that if the shape condition is dropped as in the L-Shape Sub-mesh allocation algorithm (Seo, & Kim, 2003), the system performance is improved, as it tries to fit the requested job into the dispersed processors in the system, in other words, the external-fragmentation is decreased, which is considered to be a bottle-neck problem in the processor contiguous allocation schemes (Seo, & Kim, 2003), and hence the system utilization is improved (Lo, Windisch, Liu, & Nitzberg, 1997; Yoo, & Das, 2002; Seo, & Kim, 2003; Bani Mohmmad, 2008).

Although the L-Shape Sub-mesh allocation algorithm was considered as a processor contiguous scheme, it does not satisfy the shape condition of these strategies (Lo, Windisch, Liu, & Nitzberg, 1997; Seo, & Kim, 2003).

To sum-up:

Minimizing the fragmentation will improve system performance in terms of both job turnaround time and utilization.

Dropping the shape condition increase system performance in terms of both job turnaround time and utilization.

Motivated by the above observations, an efficient processor allocation algorithm for 2D mesh connected multicomputer is proposed. The proposed algorithm alleviates external fragmentation by trying to allocate any sub-meshes that have any contiguity among them, and this is accomplished by allocating the first rectangular sub-mesh, and then allocate the remaining processors as neighbors to the allocated submesh.

The proposed strategy preserves a contiguity among the allocated processors, which decreases the communication overhead in the network as well as alleviating the external fragmentation and hence improves performance in term of utilization (Seo, & Kim, 2003; Bani Mohammad, Ould Khaoua, Ababneh, & Mackhenzie, 2007; Bani Mohmmad, 2008). The performance of the proposed scheme has been compared with the well-known First-Fit (FF) allocation scheme (Zhu, 1992), and the L-shaped submesh allocation scheme (LSSA) (Seo, & Kim, 2003). In this thesis, we examine the influence of different communication patterns on the performance of our proposed allocation scheme. The communication patterns considered are One-to-All, All-to-All (Lo, Windisch, Liu, & Nitzberg, 1997; Bani Mohammad, Ould Khaoua, Ababneh, & Mackhenzie, 2007), and Near Neighbor (Bani Mohmmad & Ababneh, 2013) communication patterns.

All of the three strategies the FF, LSSA, and NAS, were implemented and tested at the same simulation environment. The simulation results show that the performance of our proposed allocation scheme is better than that of the previous contiguous and Non-Contiguous allocation strategies that have been considered in our research work in terms of system utilization. This is because of the ability of our proposed allocation strategy in alleviating external fragmentation by allocating any requested available sub-mesh in the system while preserving a degree of contiguity among the allocated processors; and this enhance system performance. In terms of average turnaround time the simulation results show that the proposed NAS strategy has much better performance over all other schemes when one-to-all pattern is used. However, when All-to-All and near neighbor patterns used, experiments show that the performance of our proposed scheme is close to that of FF and better than LSSA.

Thesis Outline

Thesis main topics are as follows:

Chapter 2 describes some of the contiguous and the Non-Contiguous L-Shape processors allocation schemes that have been proposed for 2D-mesh-connected multicomputers, and explains the way of investigation used in this work.

Chapter 3 describes the proposed allocation scheme, Neighbor Allocation Strategy as a Non-Contiguous Processor Allocation scheme for 2D-Mesh-Connected Multicomputers (NAS), and shows the details of this strategy.

In chapter 4, outcomes from the simulation are analyzed and a comparison had been conducted with other contiguous and Non-Contiguous processor allocation schemes for 2D-mesh-connected multicomputers.

In Chapter 5, conclusions and some directions for future work are presented.

Chapter 2
Background

Related Works

In this chapter, we give a brief description of some of the well-known 2D mesh contiguous (Zhu, 1992; Lo, Windisch, Liu, & Nitzberg, 1997; Yoo, & Das, 2002; Bani Mohmmad, 2008) and Non-Contiguous (Seo, & Kim, 2003) allocation strategies that have been studied extensively (Zhu, 1992; Lo, Windisch, Liu, & Nitzberg, 1997; Yoo, & Das, 2002; Seo, & Kim, 2003; Bani Mohmmad, 2008), where most of the presiding studies (Zhu, 1992; Lo, Windisch, Liu, & Nitzberg, 1997; Yoo, & Das, 2002; Bani Mohmmad, 2008) have tried to reduce the influence of external fragmentation problem, which is inherited by contiguous allocation.

2-Dimensional-Buddy-System:

Two-Dimensional-Buddy-System, known as 2-DBS, allocates a square sub-mesh to any job request, where its length is 2^k, where $k = \lceil \log(a, b) \rceil$. For instance, assume a job requests 9 processors, then k = 2 and a 4×4 sub-mesh is allocated to that request. That will cause the fragmentation problem, internal fragmentation of 44%, this is shown in figure 2-1, where the allocated sub-mesh is surrounded by a rectangular with black border, and the black processors are the processors requested by the job. This scheme has internal & external fragmentation, also it is applicable only to square mesh-systems (Seo, & Kim, 2003; Bani Mohmmad, 2008) with side lengths of 2^k (Chuang, & Tzeng, 1994).

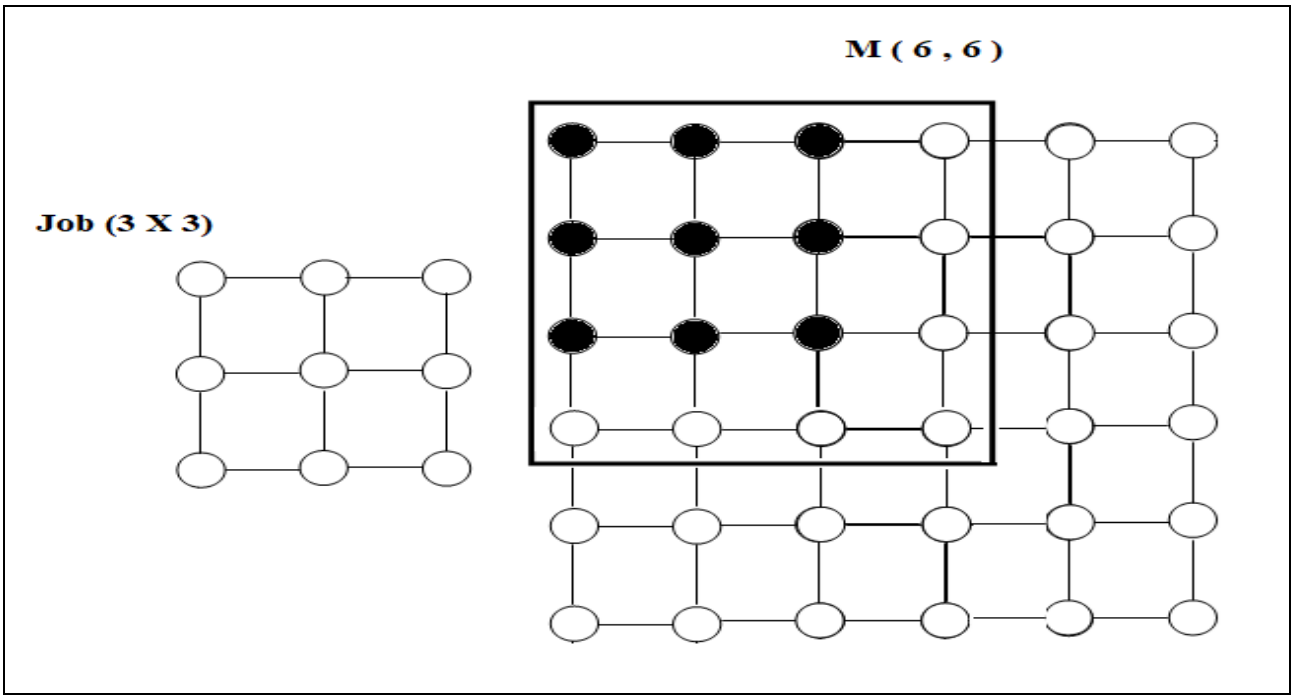


Figure 2-1: 2DBS allocates a 4x4 sub-mesh for 3x3 job with internal fragmentation of 44%.

Frame Sliding (FS):

Frame sliding scheme; known as FS, can be used with any mesh-system regardless of its size or shape (Chuang, & Tzeng, 1994). FS removes internal fragmentation by allocating exactly the requested size. This strategy works with the assumption of having a frame of the side lengths of the requested job, sliding over the system from lower left to the right and bottom to top until it finds a fit sub-mesh, otherwise, FS fails to allocate the job and it is moved to the waiting queue. Despite the advantages of FS as compared to the 2DBS, it is still having some limitations such as external fragmentation as well as it is not recognition complete. In some cases, it fails to allocate a free sub-mesh in the system although it is exist; this problem is due to the fixed sliding frame (Zhu, 1992; Chuang, & Tzeng, 1994; Lo, Windisch, Liu, & Nitzberg, 1997; Seo, & Kim, 2003; Bani Mohmmad, 2008). This has a direct influence on the system utilization (Ding, Bhuyan, 1993). Figure 2-2 demonstrates how frame sliding works, and the case where it fails to allocate a sub-mesh although it is available in the system.

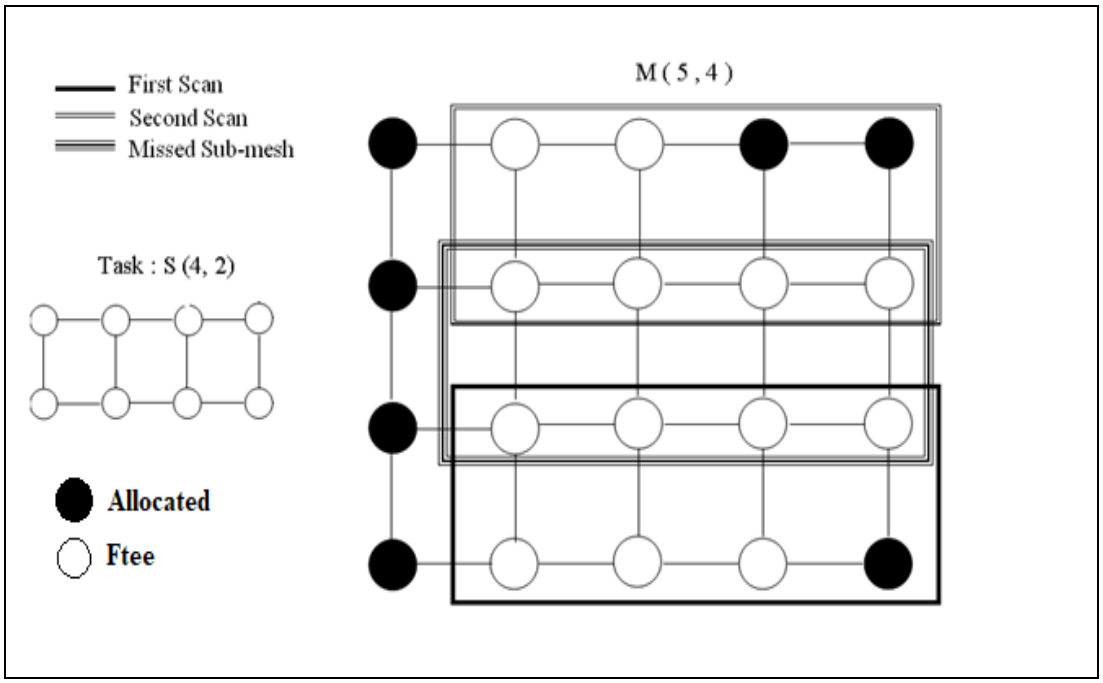


Figure 2-2: Outline FS strategy.

Adaptive Scan (AS):

Adaptive scan, known as AS, (Seo, & Kim, 2003; Bani Mohmmad, 2008) was built on the idea of FS discussed previously, but with the ability of adaptation in two main aspects; the orientation of the request in the sense that if a job arrives in the form S (a, b) and the processor allocator fails to allocate a job request, then it tries the other orientation of the request to be in the form S (b, a) and tries to allocate it again. Also in the contrary of FS, it scans the system by moving a window vertically with a side length of 1 processor and horizontally the same as the allocated sub-mesh. Yet, the relation between the side length of the window and the allocation time is the inverse relationship; shorter side length, means more allocation time, which makes AS not applicable for huge mesh systems (Ding, Bhuyan, 1993; Lo, Windisch, Liu, & Nitzberg, 1997; Seo, & Kim, 2003; Bani Mohmmad, 2008).

First-Fit (FF) and Best-Fit (BF) Strategies:

FF and BF strategies (Zhu, 1992) are applicable to any mesh system regardless of its size. In these strategies, internal fragmentations are removeed by allocating precisely the requested amount of procesors. The nods that can used like base-nodes for sub-meshes which can hold the job are stored in an array of size M , where M is mesh size. When using FF, the first free base is selected as the base for the job request. On the other hand the BF, choses the base that has the biggest number of busy neighbors and smallest number of free areas to be the base of the allocated submesh. Figure 2-3 demonstrates how the FF and BF schemes work with a certain job request. The simulation results (Zhu, 1992; Bani Mohmmad, 2008) show that the performance of FF and BF is close in both of average turnaround time as well as system utilization. FF and BF suffer from external fragmentation and do not support the orientation of the job request, this reduce the performance of the system in both of average turnaround time as well as system utilization (Zhu, 1992; Seo, & Kim, 2003; Bani Mohmmad, 2008).

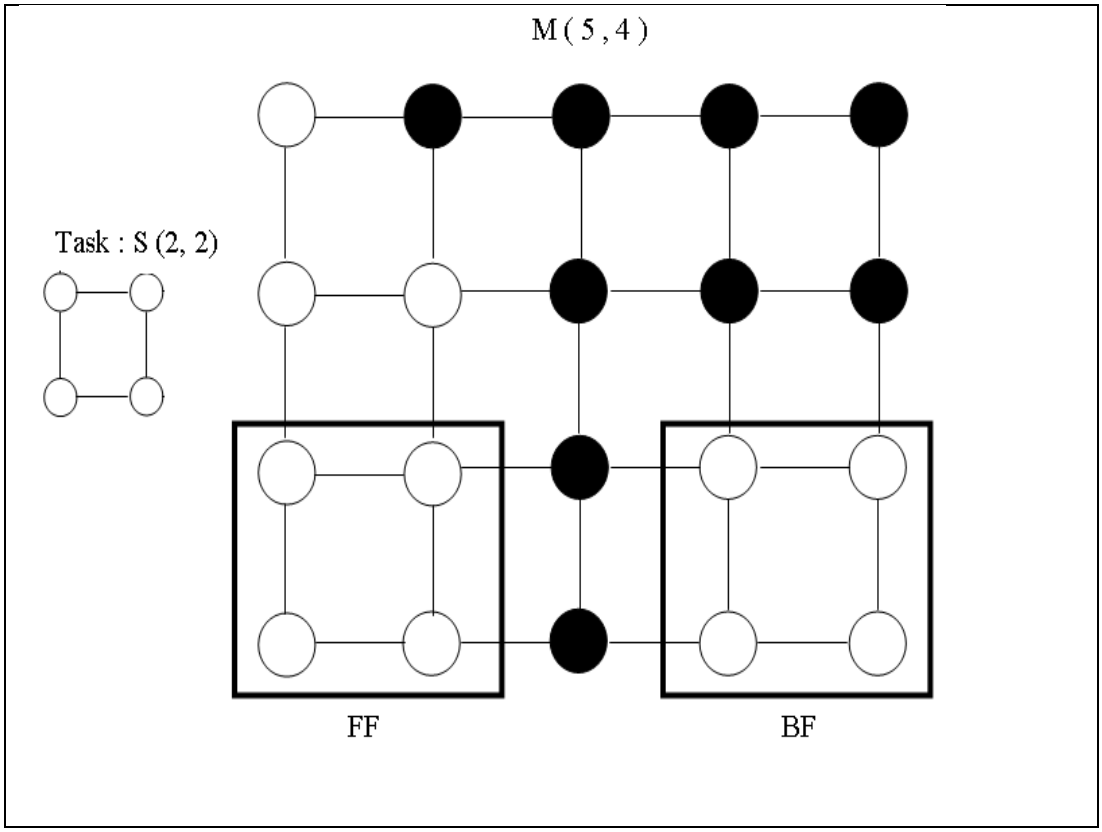


Figure 2-3: Outline FF and BF strategies.

Flex Fold Strategy:

In this strategy (Seo, & Kim, 2003; Bani Mohmmad, 2008), the orientation of the job request presented in the AS scheme (Seo, & Kim, 2003; Bani Mohmmad, 2008) is improved leading to higher system utilization. When a job $S(a,b)$ arrives to the system, the Flex Fold Strategy tries to allocate the job in the following order: $S(a,b),(b,a),(a/2,2b)\ and\ (2a,b/2)$ searching the system to find the fit sub-mesh for the job request. Despite of its improvement in the system utilization, this strategy suffers from external fragmentation and has a constraint on the job sides length, where both of these sides must be even (Seo, & Kim, 2003; Bani Mohmmad, 2008).

ALL Shapes FF Sub-Mesh Allocation Strategy (ASFF):

ALL Shapes FF contiguous sub-mesh allocation strategy (ASFF), attempts to allocate the incoming job request by permitting all shapes. Most contiguous allocation strategies (Seo, & Kim, 2003; Bani Mohmmad, 2008; Ababneh, Bani-Mohammad, & Ould Khaoua, 2010) switch the orientation of incoming job request when allocation fails. The results in (Seo, & Kim, 2003; Bani Mohmmad, 2008; Ababneh, Bani-Mohammad, & Ould Khaoua, 2010) show that this switching enhance the performance of these schemes.

Given a job request for n processors, ASFF constructs all the valid request shapes, then these shapes are considered for allocation in a specific order. The allocation is stopped upon the first successful allocation. For example, if 12 processors are requested and mesh system size is 6×6, then ASFF generates $(4,3), (3,4), (6,2), and\ (2,6)$ shapes respectively, while if a job requests 25 processors, then only one shape $(5,5)$ is generated and ASFF uses FF proposed in (Zhu, 1992) for allocation.

L-Shape Sub-Mesh Allocation Strategy (LSSA):

The L-Shape Sub-Mesh Allocation Strategy (LSSA) (Seo, & Kim, 2003) takes another approach that differs from the previous strategies proposed in (Zhu, 1992; Chuang, & Tzeng, 1994; Lo, Windisch, Liu, & Nitzberg, 1997; Bani Mohmmad, 2008), in which the requested sub-mesh is divided into two sub-meshes, one is larger than the other. In figure 2-4, four types of the LSSA allocated sub-meshes are presented.

The restrictions on the even values for the side lengths of the job request in the Flex Fold strategy (Seo, & Kim, 2003; Bani Mohmmad, 2008), is removeed in the LSSA strategy.

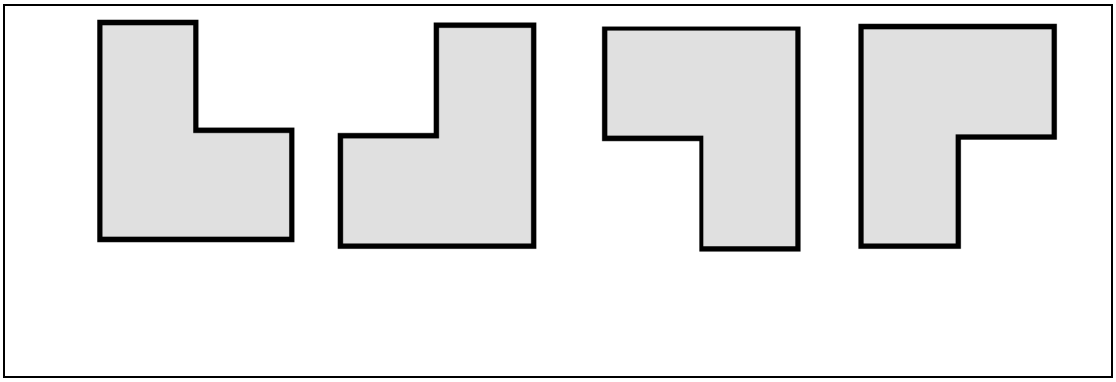


Figure 2-4: The LSSA allocation types.

Notation 1: For the two adjacent sub-meshes, the left or up sub-mesh side $S(c,d)$ is denoted by $SL(c,d)$ and right or bottom sub-mesh side $S(e,f)$ is denoted by $SR(e,f)$ (Seo, & Kim, 2003).

Notation 2: The L-shaped sub-mesh $LS(c,d,e,f)$ is represented by$[(x_{11},y_{11},x_{12},y_{12})$ & $(x_{21},y_{21},x_{22},y_{22})]$, where (x_{11},y_{11}) is the bottom-left corner of the base node, and (x_{12},y_{12}) is the top-right corner of the sub-mesh $SL(c,d)$, while (x_{21},y_{21}) and (x_{22},y_{22}) are the bottom-left and top-right corners of the sub-mesh $SR(e,f)$, respectively (Seo, & Kim, 2003).

Notation 3: In the L-shaped sub-mesh allocation strategy, the sub-meshes constructed from $S(a,b)$, for even a , where $a \geq b \geq 2$ are as follows $LS(a/2, b+k, a/2, b-k)$ where,$1 \leq k \leq b$. For odd a, where $a \geq b \geq 2$ the sub-meshes constructed from $S(a,b)$ are $LS(\lceil a/2 \rceil + k, b + \lfloor a/2 \rfloor - k, \lfloor a/2 \rfloor - k, b - \lceil a/2 \rceil - k)$ where $1 \leq k \leq (b-1-\lceil a/2 \rceil)$. LSSA changes k until it finds available free space. If $b-1-\lceil a/2 \rceil$ is -1, LSSA uses -1 as k. k is incremented by 1while it is less than 4 ,otherwise k is incremented by $\lfloor b/4 \rfloor$.

LSSA works as the following: when a job requests a submesh $S(a,b)$, LSSA check if the requested number of processors $a*b$ requested is in the system. The second test to know if the incoming job request is square, if $a\ and\ b$ are equal, then folding and L-shaping are applied only to one side a. The LSSA searches for free sub-meshes in the following sequence the original sub-mesh $S(a,b)$, its rotated sub-mesh, $S(b,a)$, folded sub-mesh $S(a/2,2b)$ for even a, its rotated sub-mesh $S(2b,a/2)$, folded sub-mesh $S(b/2,2a)$ for even b, its rotated sub-mesh $S(2a,b/2)$. If any one of these sub-meshes is found, then the allocation is done. Otherwise, L-shape sub-meshes will be searched for the requested job and its rotated sub-mesh.

The following examples explain the above notations, assuming a job requests a submesh of size $(8,6)$, where a is even, then the possible LSSA sub-meshes are $(4,7,4,5),(4,8,4,4),(4,9,4,3)\ and\ (4,10,4,2)$. Assuming a job requests a submesh of size $(5,5)$ where a is odd, then the possible LSSA allocated sub-meshes are $(3,7,2,2)\ and\ (4,6,1,1)$.

This twist on the shape of the job increases the chance of successful allocation, which enhance system performance in both of job response-time as well as system utilization (Seo, & Kim, 2003).

To sum up, all the previous strategies suffer from external fragmentation and communication overhead in some situations (Seo, & Kim, 2003; Bani Mohmmad, 2008).

Table 2-1: Comparison among allocation strategies.

	Comparison characteristics		
Allocation strategy	Suffer from external fragmentation	Suffer from internal fragmentation	Support rotation
2-DBS	√	√	
FS	√		
AS	√		√
FF, BF	√		
Flex Fold	√		√
ASFF	√		√
LSSA	√		√

System Model

In this thesis, the mesh interconnection network is used, because of its good properties such as structural regularity, simplicity, scalability, ease of implementation, and for the benefit from locality in the communication manner (Ding, Bhuyan, 1993; Yoo, & Das, 2002; Kumar, Grama, Gupta, & Karypis, 2003; Bani Mohmmad, 2008; Ababneha, Bani Mohmmad, & Ould Khaoua, 2010).

Furthermore, the mesh inter-connection networks are simply implemented because of the regular (Bani Mohmmad, 2008; Bani Mohmmad & Ababneh, 2013).

In our experiments, we use 3 patterns of communication to assess the overall performance of our proposed strategy with the other well-known allocation strategies. These communication patterns are: near-neighbor, one_to_all and All-to-All (ProcSimity v4.3, 1996; Bani Mohmmad & Ababneh, 2013).

In near neighbor pattern of communication, every processor allocated to the job sends messages to its neighbor processors, right, left, up and down (Bani Mohmmad & Ababneh, 2013).

In one_to_all pattern of communication, a processor, which is randomly selected within the same job, sends messages to all processors at the same job (ProcSimity v4.3, 1996; Lo, Windisch, Liu, & Nitzberg, 1997; Bani Mohammad, Ould Khaoua, Ababneh, & Mackhenzie, 2007).

In All-to-All pattern of communication, every processor assigned to the job request, forwards a message to all other processors within the same job (ProcSimity v4.3, 1996; Lo, Windisch, Liu, & Nitzberg, 1997; Bani Mohammad, Ould Khaoua, Ababneh, & Mackhenzie, 2007).

ProcSimity Simulator

ProcSimity is a software tool, which uses simulation for processor allocation and job scheduling schemes for both meshes and k-ary_n-cube multicomputers. It is an open-source code, which was developed at Oregon University, and written in C programing language (ProcSimity v4.3, 1996).

ProcSimity is a suitable environment for evaluating processor allocation and job scheduling strategies for mesh connected multicomputers, and it is used to evaluate the system performance of different processors allocation and scheduling schemes from various aspects such as fragmentation problem and communication overhead. ProcSimity supports mesh-connected topology throw establishing a network of processors, where each processor is connected to its neighbor through bidirectional channels in which the exchange of massages occurred (ProcSimity v4.3, 1996; Bani Mohmmad & Ababneh, 2013).

In ProcSimity, when a new job arrives to the system, it requests a number of free processors to be allocated, if there are enough free processors, those free processors are assigned to that job until the job departures the system, and then the allocated processors are freed to be available for another job request Each of the previous steps depends on the strategy used, the job scheduling controls the order in which the jobs will be served, it controls whether the job will be served or send to the waiting queue. The allocation strategy will check if the required number of processors is available, if yes, then the job will be allocated a contiguous or Non-Contiguous sub-mesh in the system depending on the allocation strategy used. The allocated processors are held by the job request for its entire lifetime, when the job finishes its execution, the allocated processors are released to be used by another job request (ProcSimity v4.3, 1996).

When ProcSimity is used as simulation tool, many runs are conducted in order to keep the confidence level above 95%, and relative errors rate less than 5%. In each run, the values of different measured metrics that include response time, system utilization, service time, and finish time are calculated. An overall average for these metrics is computed at the end of all runs (ProcSimity v4.3, 1996).

Method Justifications

Because the real system may not be available as in our case, there are two methods used for performance evaluation of any system: Analytical modeling and Simulation. Analytical models have more high requirements in terms of computational costs and they depend on some of the assumptions that restrict their applicability to special cases. Therefore, the simulation was used in our thesis.

ProcSimity simulator has been commonly used in the evaluation process for processor allocation schemes proposed in 2D-mesh-connected multicomputers, and also it has been extensively authenticated (ProcSimity v4.3, 1996; Bani Mohmmad & Ababneh, 2013; Bani Mohammad, Ababneh, 2015).

Chapter 3
Neighbor Allocation Strategy (NAS)

Introduction

Many allocation strategies had been devised for 2D-mesh-connected multicomputers, which are categorized into 2 types: Contiguous and Non-Contiguous strategies for allocation.

In contiguous allocation strategies used in 2D-mesh-connected multicomputers, the allocated processors must be physically adjacent, and in some of these strategies, the allocated sub-mesh should be the same shape of the mesh itself. As a result, the mesh suffers from external and internal fragmentation problem. The restriction on the contiguity as well as the shape reduce the ability of a successful allocation for the job requests arrive to the system. This motivated the researchers to propose the Non-Contiguous allocation schemes, which are alternative when the allocation scheme fails to allocate request, although processors are available in the system (Bani Mohmmad, 2008).

In Non-Contiguous allocation, the job can be allocated on multiple separated small submeshes instead of waiting for a single submesh of the requested size and shape is available. More recent Non-Contiguous allocation strategies adopt the idea of partitioning the allocation of the request based on the available free submeshes in the system. (Bani-Mohammad, Ababneh, & Hamdan, 2011). Dropping the shape condition and partitioning the allocation request can reduce fragmentation problem and increases system utilization, but it is also able to cause communication overhead (Lo, Windisch, Liu, & Nitzberg, 1997). The communication-overhead

may be alleviated in Non-Contiguous allocation strategies by preserving a good degree of contiguity among the allocated processors (Bani-Mohammad, Ababneh, & Hamdan, 2011). Decreasing the external fragmentation which results from dropping contiguity has been shown more significant influence on system performance overall than the extra communication-overhead associated with Non-Contiguous allocation (Bani-Mohammad, Ababneh, & Hamdan, 2011).

Motivated by the previous observations, we proposed a new processor allocation strategy for 2D-mesh-connected multicomputers, called Neighbor Allocation Strategy (NAS for short), the aim of this strategy is to alleviate the fragmentation issue in the mesh-connected multicomputers by trying to allocate any sub-meshes for any incoming job requests that have any degree of contiguity among them when an enough number of processors is available in the system for the job request.

The Neighbor Allocation Strategy (NAS):

A 2D-mesh-connected multicomputers is represented by $M(w,h)$ where w is the width of the mesh and h is its height, each processor can be identified by two coordinates (x,y), where $0 \leq x < w$ and $0 \leq y < h$. Figure 3-1 shows the initial state of a 2D-mesh-connected multicomputers, where all the processors are not allocated (free), which are shown as white circles.

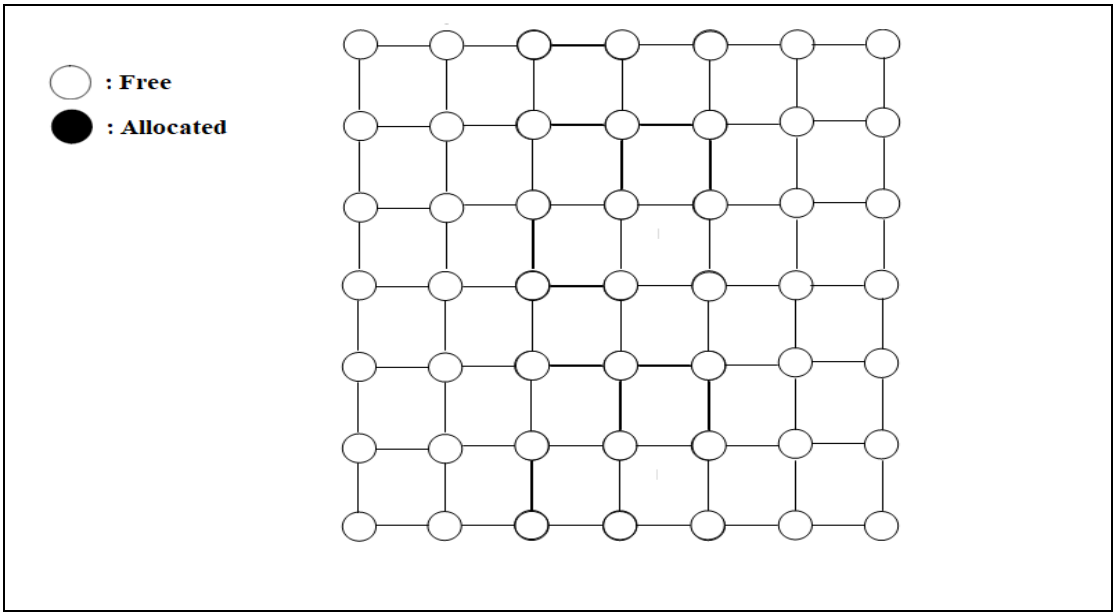


Figure 3-1: An empty 7 × 7 2D mesh.

The incoming job request is represented by $S(a,b)$, where the number of requested processors by a job is $a \times b$. The NAS checks if the amount of the requested processors is available in the system otherwise it signals allocation failure. Then a check if the incoming job request is square is done, if $a\ and\ b$ are equal, then folding and NAS allocation are applied only to one side a. The NAS searches for free sub-meshes in the following sequence until a free sub-meshes are found or all the alternatives failed to be allocated: the original sub-mesh $S(a,b)$, the ASFF for $S(a,b)$, the NAS sub-meshes constructed from $S(a,b)$ and its rotation $S(b,a)$. If the allocation succeeds the FF is used for the allocation.

The NAS largest sub-mesh assigned to the job is called nucleus sub-mesh NS, where it is the largest available contiguous sub-mesh having the same shape as the system itself. For any job that requests a sub-mesh $S(a,b)$, the NAS allocation strategy constructs the nucleus sub-mesh NS as follows, for even a , where $a \geq b \geq 2$, $NS(a/2, b+k)$ where, $1 \leq k \leq b$ for odd a, where $a \geq b \geq 2$, the nucleus sub-mesh NS is $NS(\lceil a/2 \rceil + k, b + \lfloor a/2 \rfloor - k)$ where $1 \leq k \leq (b - 1 - \lceil a/2 \rceil)$. NAS changes k until it finds available free sub-mesh. If $b - 1 - \lceil a/2 \rceil$ is -1, NAS uses -1 as k. k is Incremented by 1 while it is less than 4 , otherwise k is incremented by $\lfloor b/4 \rfloor$ as the LS sub-mesh is constructed in the LSSA (Seo, & Kim, 2003). Next NAS searches the system to allocate the remaining of the requested processors as neighbors to the allocated processors for the job request. This maintains contiguity as well as it decreases the external fragmentation in the system and hence improves performance in terms of system utilization as shown in simulation results in Chapter 4.

To describe the proposed strategy, we give some examples that were selected carefully to clarify how NAS algorithm works. Initially, we assume that the 7×7 mesh system is empty. In the first example the job requests a sub-mesh of size 3×5, as shown in figure 3-2.

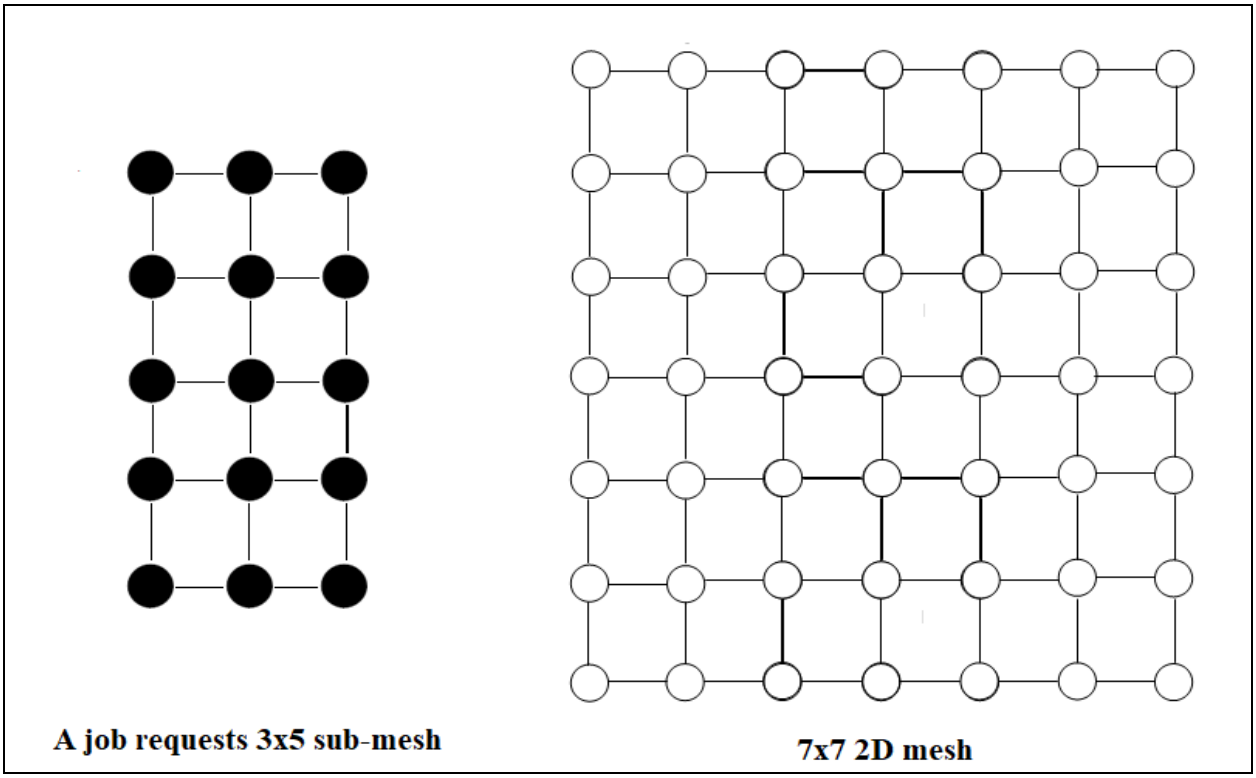


Figure 3-2: Allocation of 3x5 sub-mesh in 7x7 2D mesh by NAS.

The incoming job requests a submesh of size 3×5, where 15 processors should be allocated to this job request. The proposed strategy NAS rebuilds the job request, and constructs its nucleus sub-mesh of size 2×6 and allocate it in the mesh system, as shown in figure 3-3. Next, NAS searches the mesh system for free neighbors processors to be allocated to job request. As shown in figure 3-3, the neighbors processors (1, 2 and 3) are allocated to job request respectively.

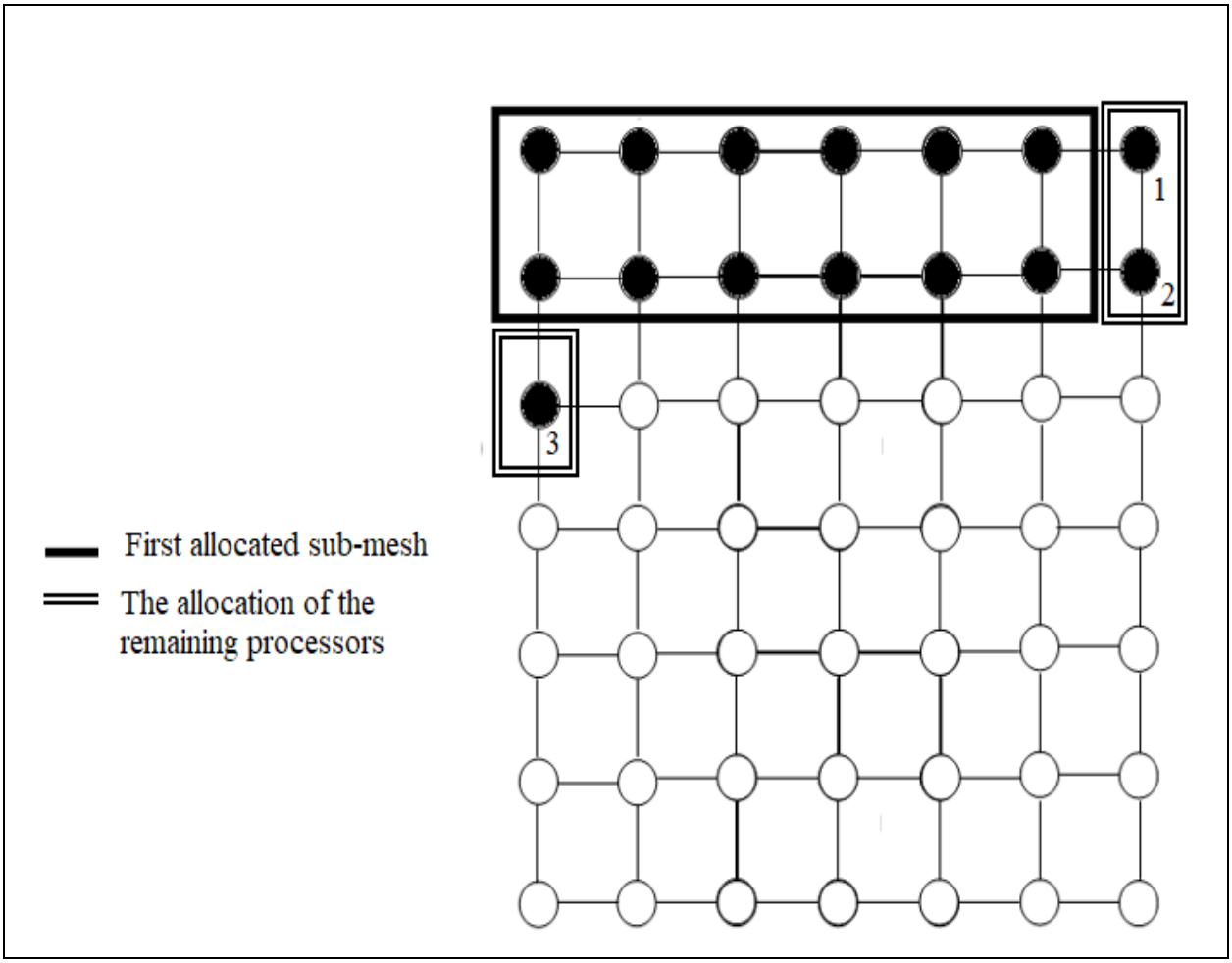


Figure 3-3: Allocation of 3x5 sub-mesh in 7x7 2D mesh.

Assuming the system state shown in figure 3-4, the second example shows an incoming job that requests a 4×3, which are 12 processors. In this example, NAS rebuilds the request by constructing a 2×4 nucleus sub-mesh and tries to allocate it in the system, then the remaining 4 processors are allocated as shown in figure 3-5.

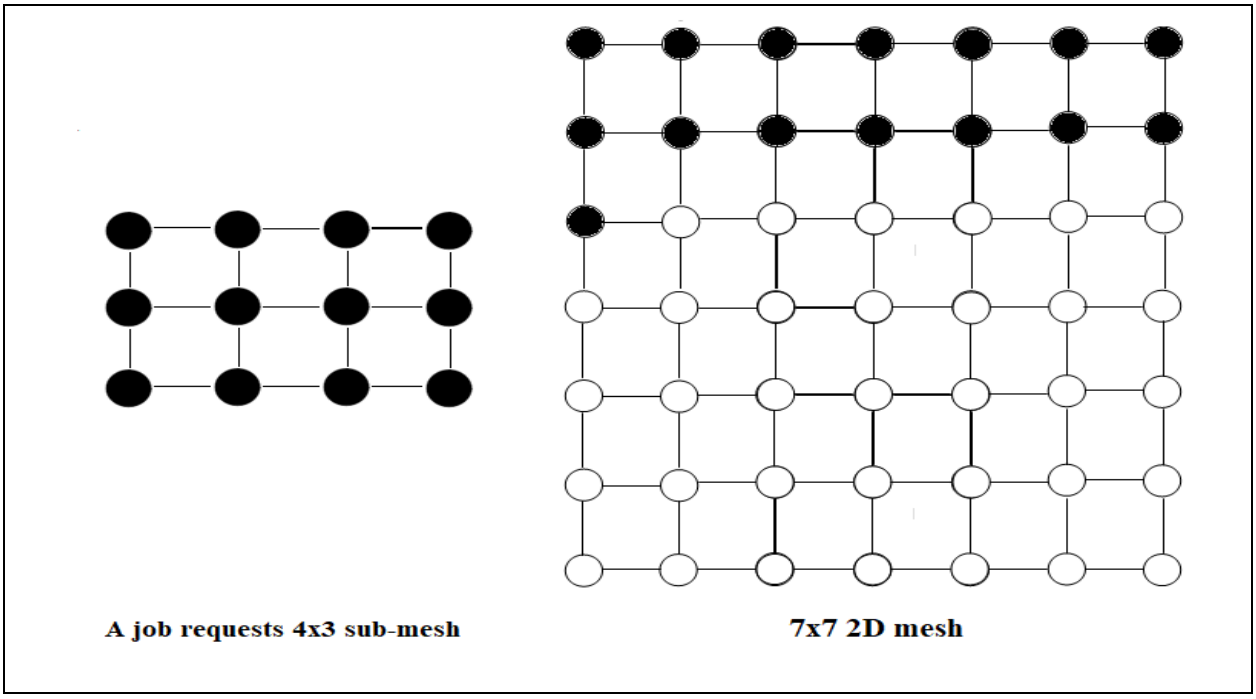


Figure 3-4: Allocation of 4x3 sub-mesh in 7x7 2D mesh by NAS.

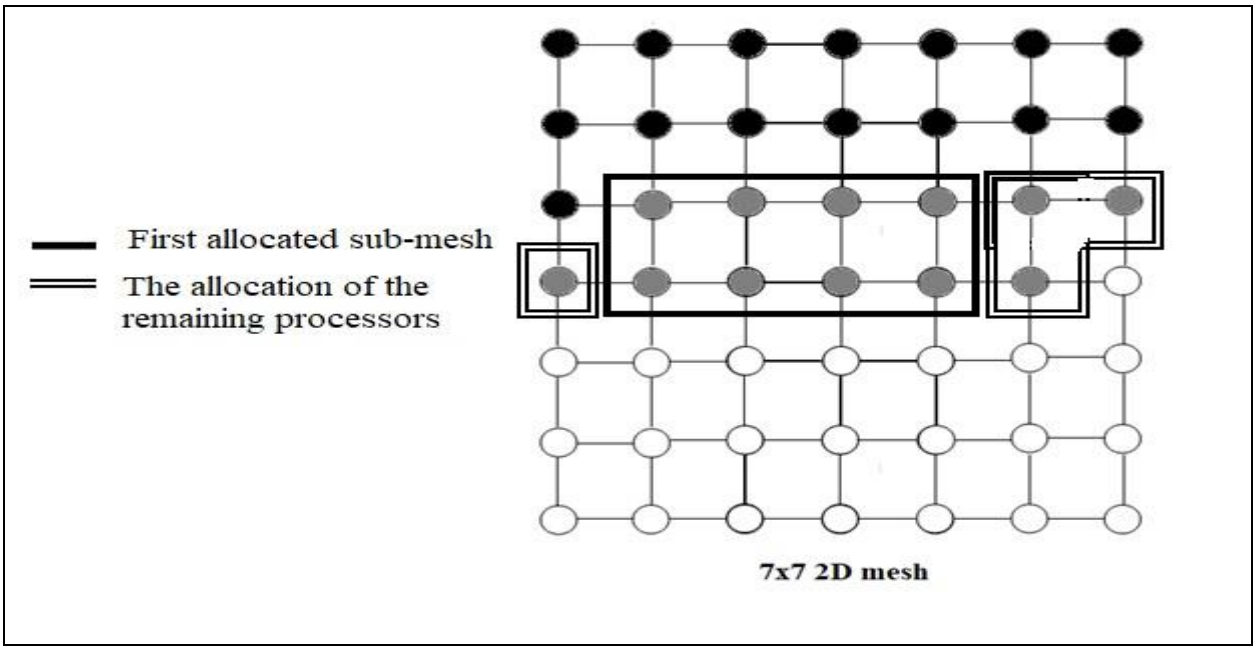


Figure 3-5: The allocation of 4x3 sub-mesh in 7x7 2D mesh.

Figure 3-6 shows a 7×7 2D mesh system after the first job departs the system and another job requests a sub-mesh of size 6×3 arrives to the system

, where 18 processors are needed for this request. Here, the NAS rebuilds the request as a 3×2 sub-mesh as its nucleus sub-mesh and allocates it into the system, then the 6 processors are allocated to the job as shown in figure 3-7.

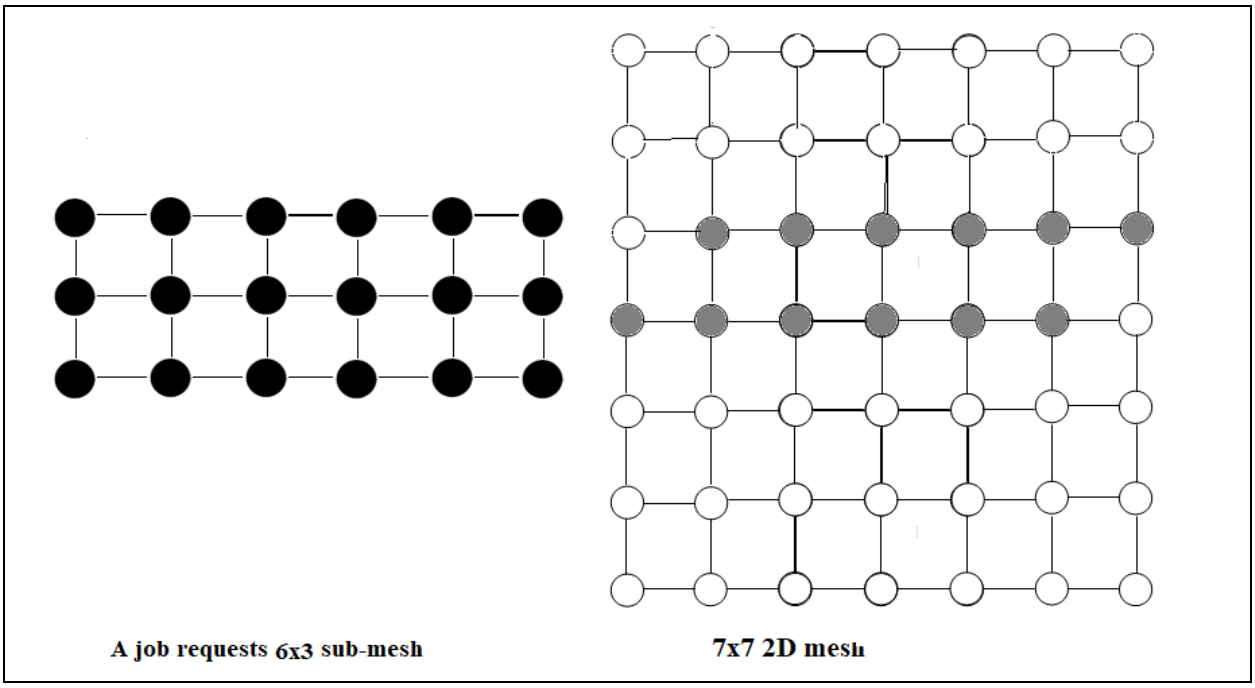


Figure 3-6: A job requests 6x3 sub-mesh and the first job had completed.

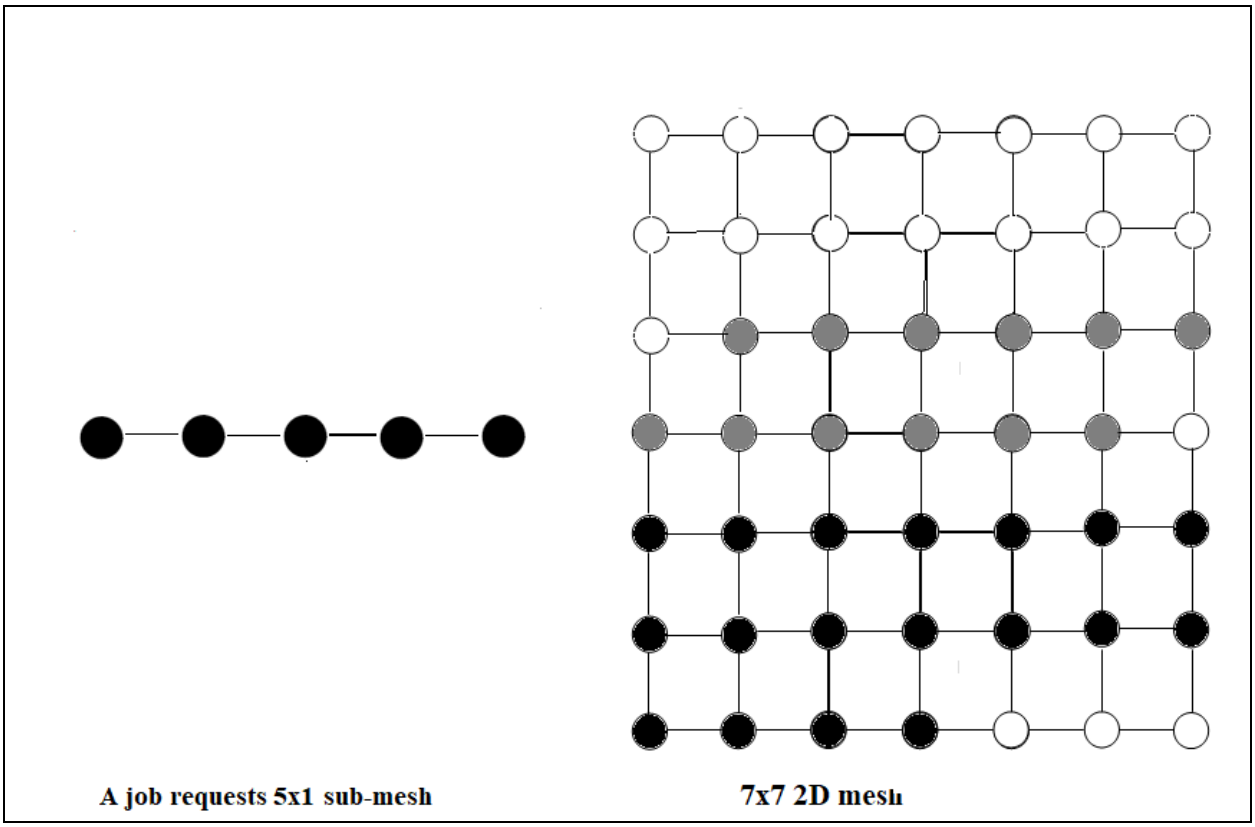


Figure 3-7: A job requests 5x1 sub-mesh.

In the fourth example, a job requests a submesh of size 5×1 as shown in figure 3-7. In this example, NAS tries to rebuild the job request to for a nucleus sub-mesh, but it fails, despite that the required number of processors is available in the mesh system. In this case, NAS considers the nucleus sub-mesh to be 1×1 and scans the mesh system to allocate the remaining of the required number of processors by keeping the contiguity as a primary condition for the allocation as shown in figure 3-8.

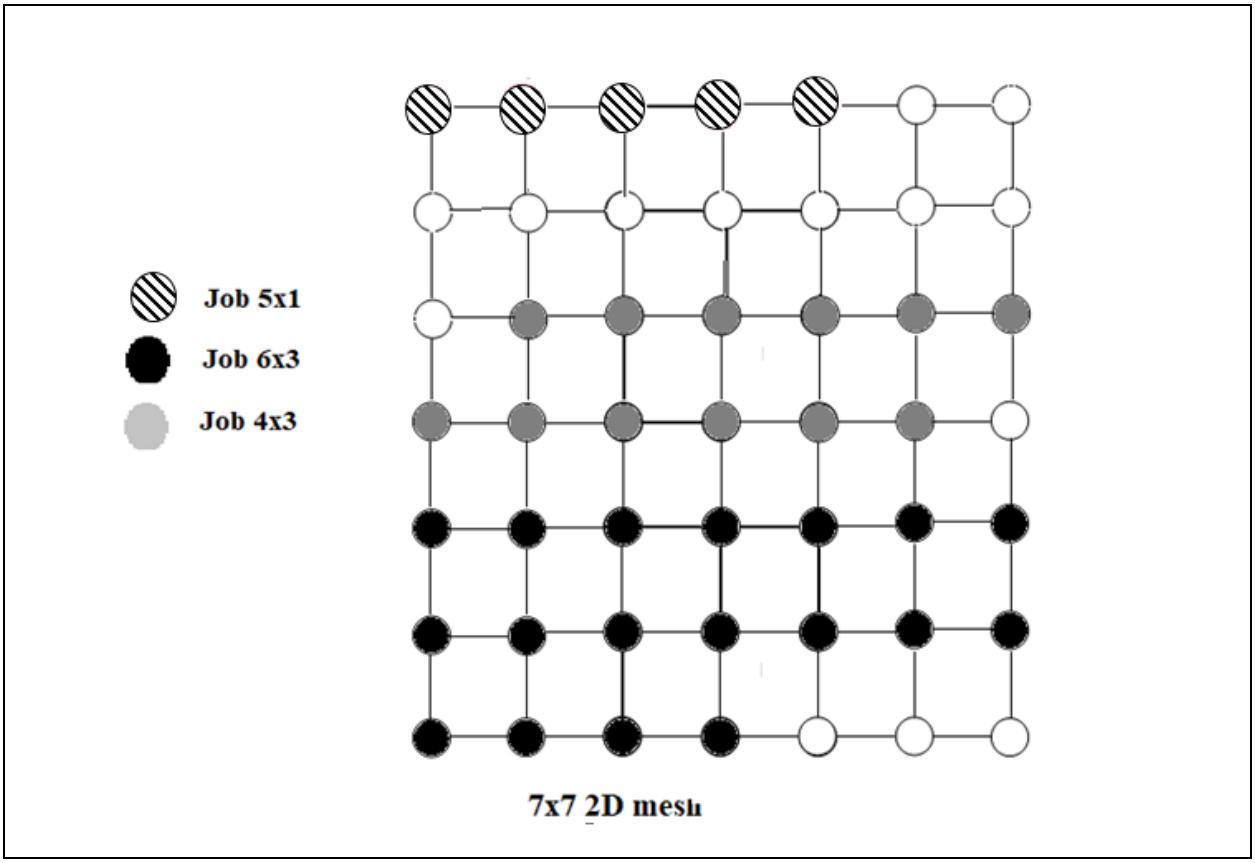


Figure 3-8: The allocation of 5x1 job request by NAS.

In the last example, a job requests a submesh of size 5×2. In this example, NAS tries to allocate the job by rebuilding the job requests to form its nucleus sub-mesh as 4×3, but this shape is not available in the mesh system as shown in figure 3-9. So, NAS tries again to rebuild the request by forming another nucleus sub-mesh as 5×2, but this shape is also not available in the system as shown in figure 3-9. NAS tries again to rebuild the request by forming another nucleus sub-mesh as 6×1 and allocates it into the system, then the 4 processors are allocated to the job as shown in figure 3-10.

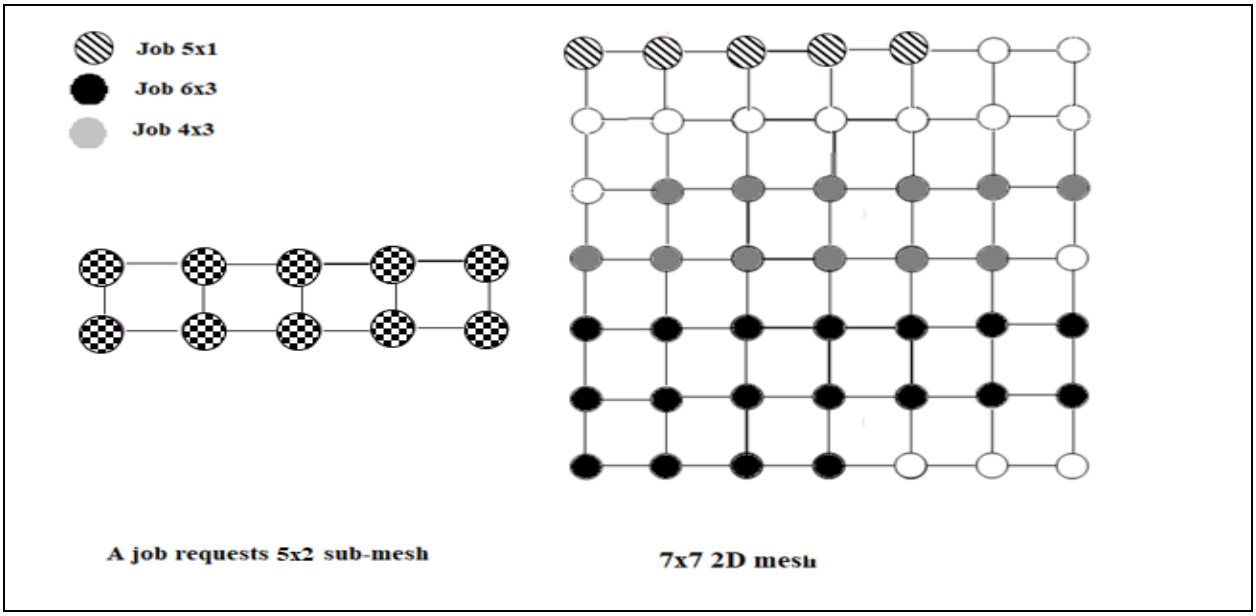


Figure 3-9: A job requests 5x2 sub-mesh.

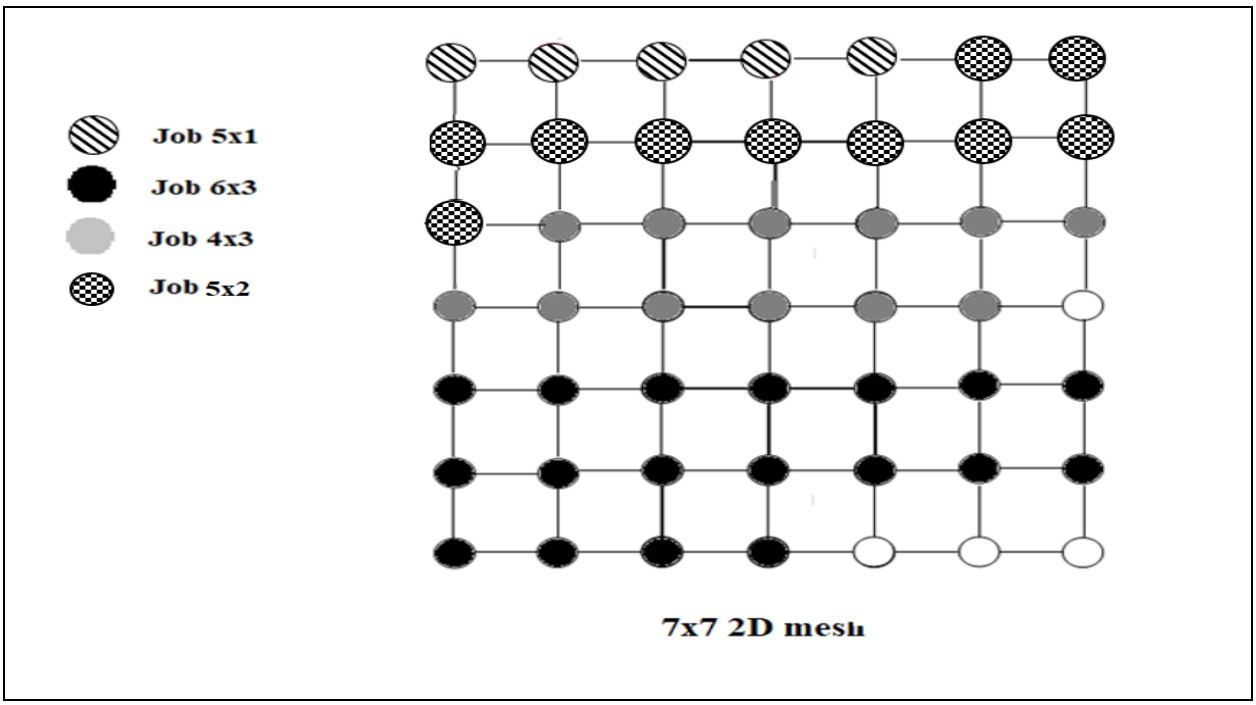


Figure 3-10: The allocation of job 5x2 in the system.

The NAS allocation and deallocation algorithms are shown in figures 3-11, 3-12, respectively.

The NAS allocation and de-allocation algorithms are shown in figures 3-11, 3-12, respectively.

Procedure NAS_Allocate(a, b):{ W: Width of the mesh. H: Height of the mesh. MeshSize=W*H. Job_size = a×b. Rotation_flag = True. Free_alocation_flag = True.		// where a, b are the side length of the job request.
Step 1.	If (a == b) Rotation_flag=False;	
Step 2.	If (MeshSize == 0 or MeshSize < job_size) Return Failure.	

Step 3.	If (ASFF is found) { Assign a×b processors to the job request. MeshSize -= a×b. Add allocated processors info to APA. Return Success.}	//ASFF All Shape First Fit strategy. //APA: Allocated Processors Array, which is an array that contains the job id and the coordinates of the allocated processors.

Step 4.	If (a is even and a >= b and b>=2)
	Go to 5.
	Else If (a is odd and a >= b and b >=2)
	Go to 5.
	Else If (b is even and b >=a and a >=2 and Rotation_flag = True)
	Go to 5.
	Else If (b is odd and b >=a and a>=2 and Rotation_flag = True)
	Go to 5.
	Else If (MeshSize > job_size) {	

Step 4.1.	Consider nucleus sub-mesh (c, d) as (1, 1). Go to 5.1.}	//where c, d the side length of the nucleus sub-mesh as described in step 5.

Step 5.	Compute k value, where k is a variable that is used to rebuild the job sides length (width, height) as follows:
	
	If ((b – 1 – cei l(a/2) == -1 and a >= b) or (a - 1- ceil(b/2) == -1 and b > a)
	
	K=-1;
	
	Else if (a is even and a>b) or (b is even and b>a)
	
	K=-1;
	
	Else
	
	K=0;
	
	Max_k, which is the maximum value for k, where it is computed as follows:
	
	if (a is even and a >= b) or (b is even and b > a)
	
	Max_k = b;
	
	Else if (a is odd and a >= b)
	
	Max_k = b – 1 - ceil(a/2);
	
	Else if (b is odd and b>a)

```
                Max_k=a-1-ceil(b/2);
        While ( k <= Max_k) {
        Construct the nucleus sub-mesh (c, d) using a, b, and k as the following:
                If a is even and a >= b or b is even and b > a then
                        c = a/2, d = b + k.
                        c = b/2, d = a + k, respectively.
                If a is odd and a >= b or b is odd and b > a then
                        c = ceil (a/2) + k, d = b + floor (a/2) - k,
                        c = ceil (b/2) + k, d = a + floor (b/2) - k, respectively.
Step 5.1.       Search the mesh system for the nucleus sub-mesh (c, d) based
                on FF allocation strategy.
                If (the nucleus sub-mesh is found) {
                        Add allocated processors info to APA.
                        Go to 6.}
                Else
Step 5.2                Recalculate k as the following:
                        If k >=4
                                k += floor ( b/4 );
                        else
                                k ++;}
        Free APA from current job;
        if (Meshsize > (a * b) and Free_alocation_flag =True)
                Free_alocation_flag = False.
                Go to 4.1.
        else
        Return failure;
```

```
Step 6.        X= Job_size – ( c * d ).
               Iteration = W * H. // to scan the system at least once or success
               allocation.
               While ( X > 0 and Iteration > 0){
                     Search the mesh for free neighbor processors to the nucleus sub-
                     mesh by scanning the mesh.
                     if (free_processor is found) {
                           X--;
                           Iteration --;
                           Add allocated processors info to APA.}}
               If (X ==0) {
                     Meshsize - = a * b;
                     Assign the processors from APA information to the job.
                     Return Success.}
               Else
                     Free APA from current job;
                     Go to 5.2.
}end procedure
```

Figure 3-11: Outline of the proposed NAS allocation algorithm.

```
Procedure NAS_Deallocate(a, b): {
Job_id = id of the leaving job.
For all nods in APA[W*H]                    //APA: Allocated Processors Array,
      If (node_id = Job_id)                 which is an array that contains the job
            Remove Node.                    id and the coordinates of the allocated
                                            processors.
}end procedure
```

Figure 3-12: Outline of the proposed NAS de-allocation algorithm.

Chapter 4
Results from Simulation

In chapter four, the simulation experiments have been conducted for the proposed NAS allocation scheme, the contiguous allocation scheme FF and the Non-Contiguous allocation scheme LSSA. The performance of NAS has been compared with the existing allocation strategies FF (Zhu, 1992) and LSSA (Seo, & Kim, 2003).

NAS allocation and de-allocation algorithms have been implemented in C language, and integrated into the ProcSimity tool, which is commonly used for testing allocation and job scheduling proposed schemes in parallel systems (ProcSimity v4.3, 1996).

The mesh system used in the simulation experiments is a 2D square mesh with L side lengths . Job inter_arrival time has been exponentially distributed. The job scheduling strategy used is the First Come First Served (FCFS) scheduling. FCFS is used because it is equitable and our role in this work is to examine the performance of our proposed allocation scheme. The job execution time is the period of time required until each job is completed. The execution time of any job is affected by packet size, the amount of messages to be exchanged among processors, the contention on the network, and the distances that massages traverse (Bani Mohmmad, 2008).

In our experiments, two job distributions were used to produce the size of a job. The first distribution is the Uniform over the range from one to the mesh length L, where both side lengths of a job request are generated independently. The second is the Uniform-Decreasing Distribution, where it is based on four probabilities $P1, P2, P3\ and\ P4$, and the four integers $I1, I2, I3\ and\ I4$, where the probabilities that both of the width as well as the height of the job falls in the ranges $[1, I1], [I1 + 1, I2], [I2 + 1, I3], [I3 + 1, I4]\ are\ P1, P2, P3\ and\ P4$, respectively. The Uniform-Decreasing Distribution simulate the situation where the majority of jobs are small comparative to the system size, so in these experiments, the values will be as follows: $P1 = 0.4, P2 = 0.2, P3 = 0.2, P4 = 0.2, I1 = L/8, I2 = L/4, I3 = L/2, I4 = L$ (Zhu, 1992; Lo, Windisch, Liu, & Nitzberg, 1997; Bani-Mohammad, Ould-Khaoua, Ababneh, & Mackhenzie, 2007).

Each run consists of 1000 finished job per run, and the number of runs is varied to get a confidence level of 95% and relative-errors do not exceed 5% (Bani Mohmmad, 2008). A job remains in the system until an iteration of the communication pattern is completed. The 3 patterns of communication where used in these experiments, these patterns are near-neighbor, one-to-all, and All-to-All communication patterns (ProcSimity v4.3, 1996; Bani-Mohammad & Ababneh, 2013). Table 4-1 shows the simulator parameters that have been used.

Table 4-1: The system parameters used in the simulation experiments.

Simulation Parameter	Value
Dimensions of the Mesh	16×16.
Allocation Strategy	FF, LSSA, NAS.
Scheduling Strategy	FCFS.
Job Distribution	Uniform: job side lengths are uniformly distributed over the interval from 1 to the mesh length.
	Uniform Decreasing: represents the situation where the majority of the jobs are small relative to the size of the system.
Inter_arrival Time	Exponential with different mean values. The mean values ranged from lower to higher values. This was done during the experiments.
Number of runs	The number of runs must be sufficient so that the confidence interval is 95% and the relative-errors 5%.
Number of jobs per Run	1000.

In our work, the performance of the mesh system was measured using average-response time, and utilization. The response (turnaround) time of a job is the time that the job spends in the system from the arrival until leavening the system (Bani-Mohammad 2008; Seo, & Kim, 2003). The system utilization defined as the percentage of the processors being utilized over a given interval of time (ProcSimity v4.3, 1996; Seo, & Kim, 2003). The standalone parameter in the simulation is the job arrival rate, which is the inverse of the mean inter_arrival of tasks.

System Utilization

Figures 4-1 to 4-6 show the mean system utilization when one-to-all, All-to-All and near neighbor communication patterns were used, using the FCFS scheduling scheme, and for uniform and uniform decreasing for job size distributions. The outcomes show that the NAS allocation strategy results are encouraging than all other allocation schemes for the two job size distributions considered in these experiments, at high job arrival rates. This is because of the ability of the NAS allocation strategy to remove both of internal as well as external fragmentation. Mostly, when the number of requested processors is available, NAS will allocates the job request. The system utilization ranged from 62% to 76% and 58% to 79% for uniform and uniform-decreasing job size distributions, respectively.

In figure 4-1, for example, the mean system utilization for all considered allocation algorithms is almost the same for job arrival rates that are below the 0.0005 jobs/time units, since the difference is less than 5% which is the percentage of error in the simulation experiments, so this difference can be ignored. However, for the arrival rates above 0.001 jobs/time units, the proposed NAS allocation algorithm performance is leading than the LSSA and FF allocation algorithms. Because NAS has more ability to remove external fragmentation by utilizing the free processors in the mesh system in an efficient way.

In figure 4-2, the mean system utilization is improved for all allocation strategies when the uniform-decreasing distribution is used. This is because of the increases probability of generating small jobs comparative to the size of the mesh system and hence the allocation of most of these jobs is succeeded. NAS also performs much better than the LSSA and FF allocation algorithms for high job arrival rates.

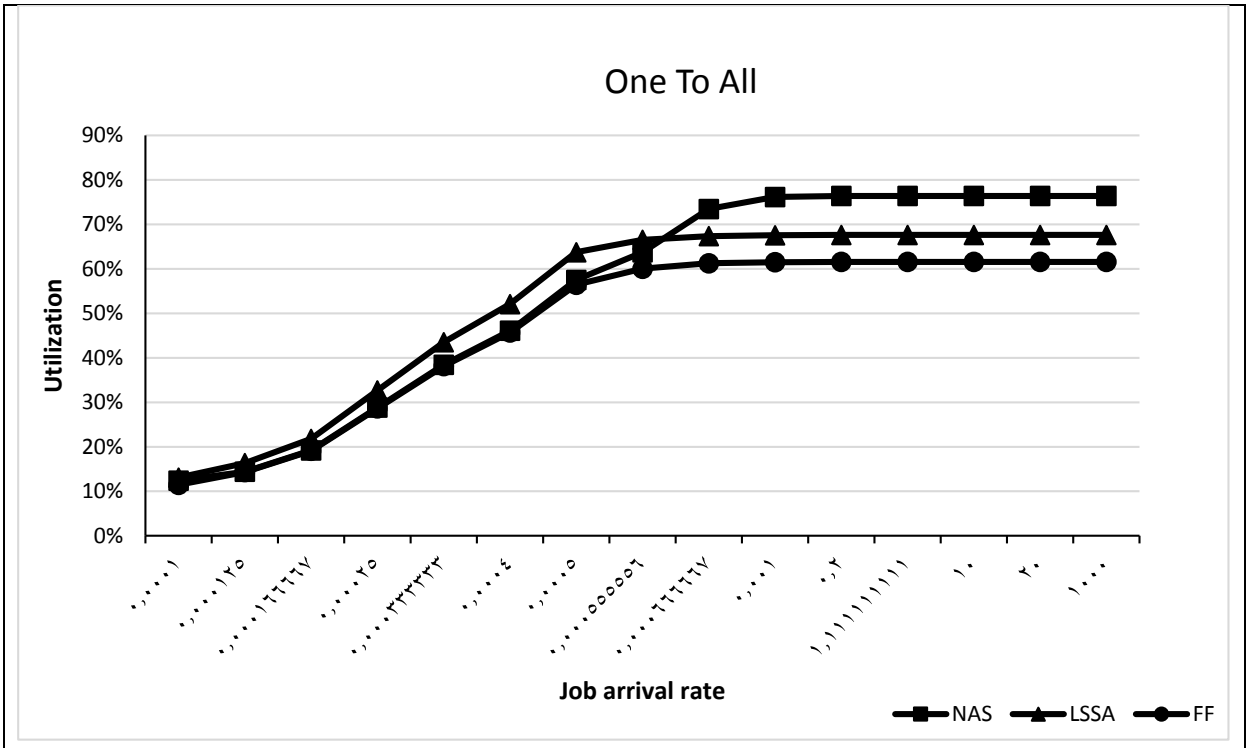


Figure 4-1: Mean system utilization vs. arrival-rate using one_to_all pattern and uniform for job size in a 16x16 mesh.

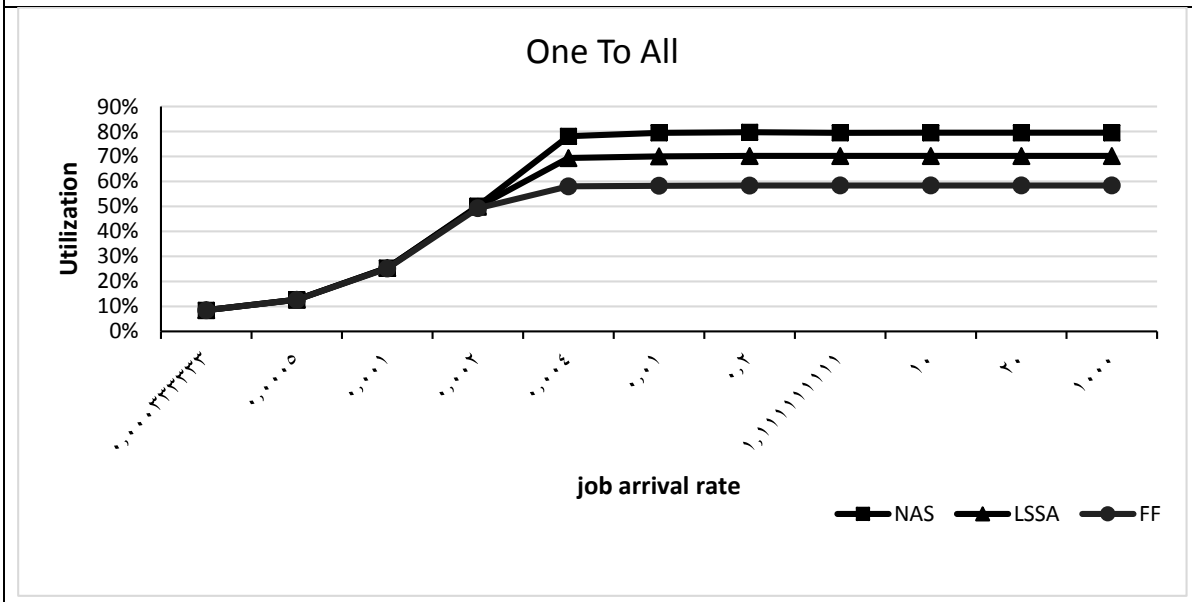


Figure 4-2: Mean system utilization vs. arrival rate using the one_to_all communication pattern and uniform decreasing distribution for job size in a 16x16 mesh.

In figure 4-3, for example, the mean system utilization is almost the same for both of the NAS and LSSA for the arrival rates below 0.00005 jobs/time units and better than that of the FF. This is because of the ability of the NAS and LSSA allocation algorithms to remove both of internal as well as external fragmentation. For high arrival rates (at 0.00006667 jobs/time units and above) of jobs, NAS performs better than the LSSA and FF. This is because of its ability to remove external fragmentation as compared to LSSA, and hence the system utilization is improved.

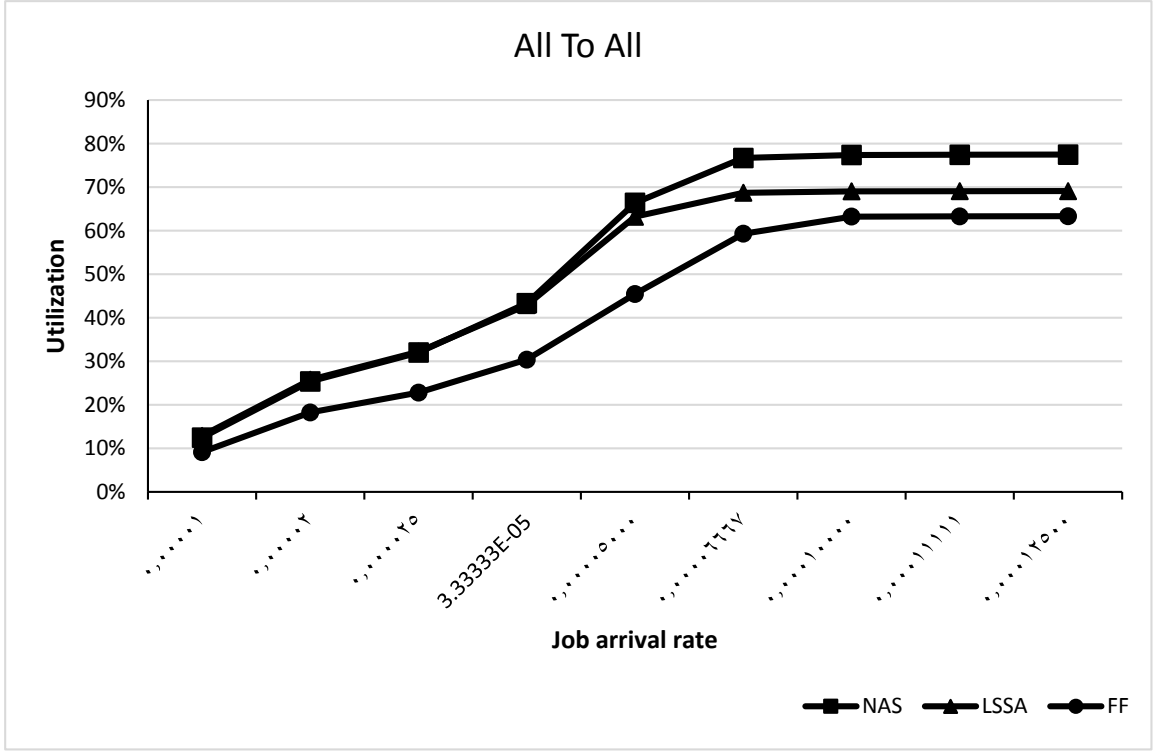


Figure 4-3: Mean system utilization vs. arrival rate using the All-to-All commnication patern and uniform distribution of job size in a 16x16 mesh.

In figure 4-4, the mean system utilization is improved for all allocation strategies when the uniform-decreasing distribution is applied. This is because of the increased probability of generating small jobs comparative to the size of the mesh system, and hence the allocation of most of these jobs is succeeded. NAS performs much better than the LSSA and FF. For example, NAS performs better than the LSSA and FF at job arrival rate equal to 0.000333333 jobs/time units.

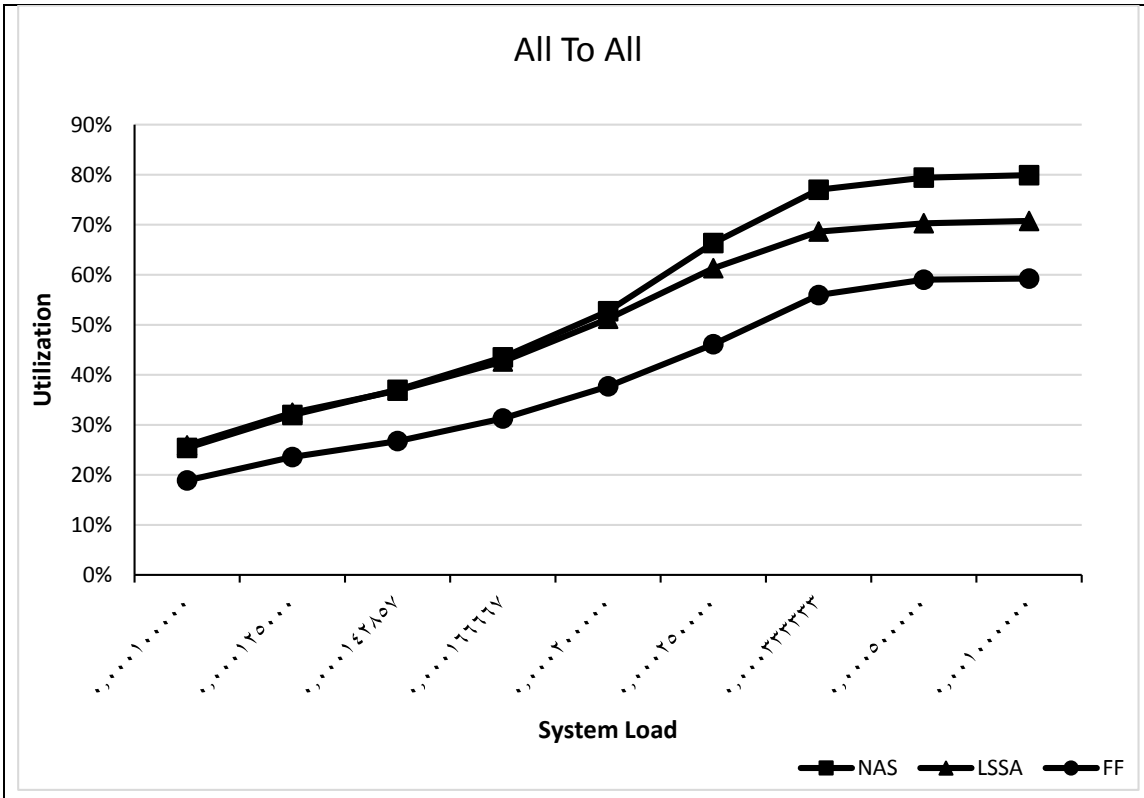


Figure 4-4: Mean system utilization vs. arrival rate using the All-to-All communication pattern and uniform decreasing distribution for job size in a 16x16 mesh.

In figure 4-5, the outcomes show that the mean system utilization for LSSA is better than that of the NAS and FF when the job arrival rates are below the 0.00333333 jobs/time units, but since the difference in performance among these algorithms is below the 5%, which is the percentage of error in the simulation experiments, this difference can be ignored. However, NAS performs better than the LSSA and FF allocation algorithms when the job arrival rates are above 0.005 jobs/time units. This is because of the ability of the NAS allocation algorithm to remove both of internal as well as external fragmentation.

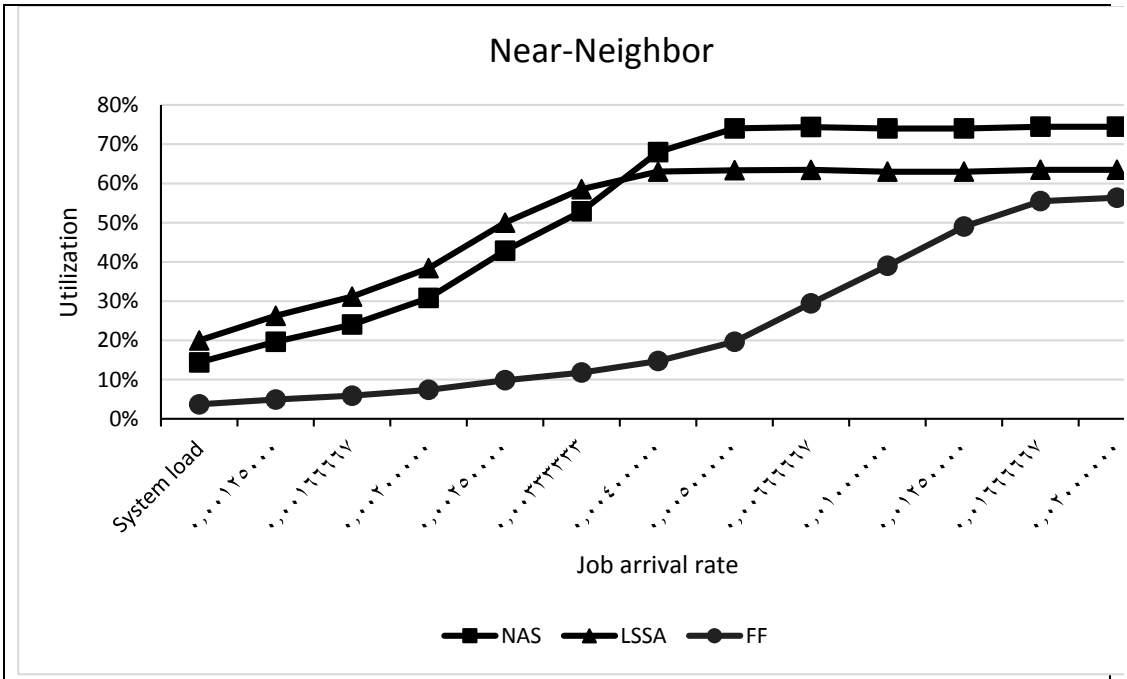


Figure 4-5: Mean system utilization vs. arrival rate using the near neighbor communication pattern and uniform distribution for job size in a 16x16 mesh.

In figure 4-6, the outcomes show that the mean system utilization for the LSSA is better than that of the NAS and FF allocation schemes for the arrival rates below the 0.0125 jobs/time units, but since the difference in performance is below 5%, which is the percentage of error in the experiments, this difference in performance can be ignored. However, NAS performs better than the LSSA and FF for the arrival rates that are above 0.005 jobs/time units. This is because of the ability of the NAS allocation algorithm to remove both of internal as well as external fragmentation.

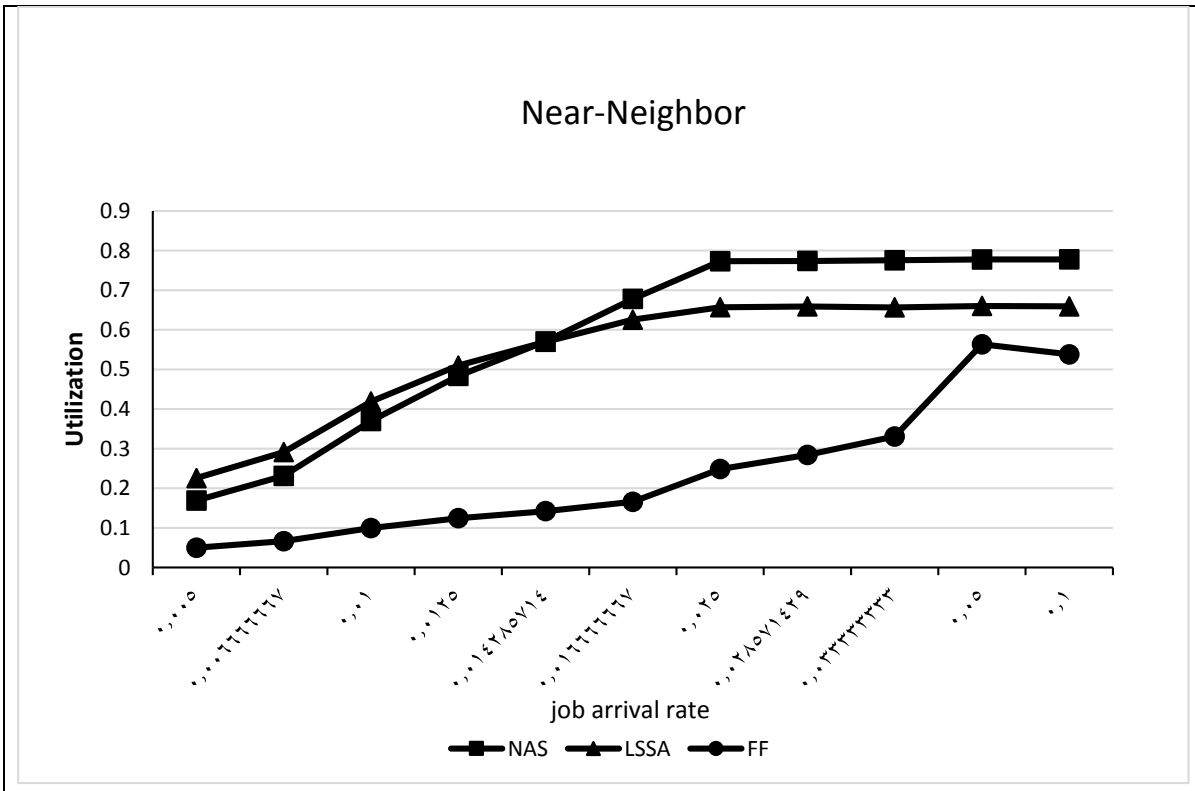


Figure 4-6: Mean system utilization vs. arrival rate using the near neighbor communication pattern and uniform decreasing distribution for job size in a 16x16 mesh.

Average Response Time

Figures 4-7 to 4-12 show the average turnaround time of jobs using the one-to-all, All-to-All and near neighbor communication patterns when the FCFS scheduling scheme is used, and for job size distributions considered, uniform and uniform-decreasing. As the figures 4-7 and 4-8 shows that the proposed NAS strategy has superior performance over all other schemes when one_to_all communication pattern is used for both uniform and uniform-decreasing job size distributions. This is because of the ability of NAS to remove external fragmentation as compared to FF and LSSA, which increases the probability of successful allocation, and hence enhance system performance in term of response time of jobs.

but, when All-to-All and near neighbor communication patterns are used, the FF has the superior performance over all other schemes. This is because contiguous allocation strategies allocate a rectangular sub-mesh for the jobs request, and this minimizes inter-job interference, which minimizes the communication overhead in the system (Al-Harafsheh, 2016). However, NAS is better than LSSA because of its ability to remove external fragmentation, and keeping a good degree of contiguity, which reduces communication overhead, and hence improves job response time.

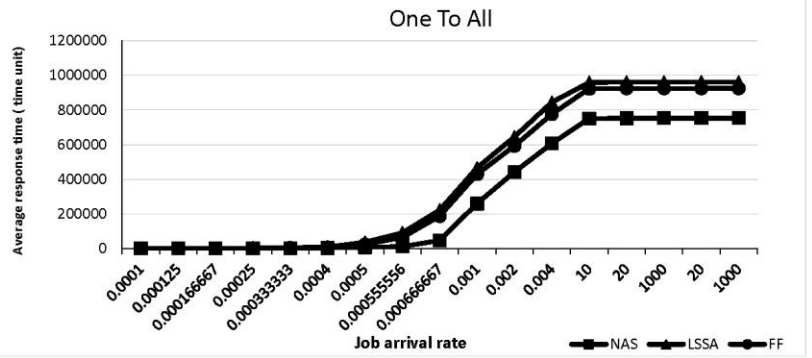


Figure 4-7: Average response time vs. arrival rate using one_to_all communication pattern and uniform distribution for job size in a 16x16 mesh.

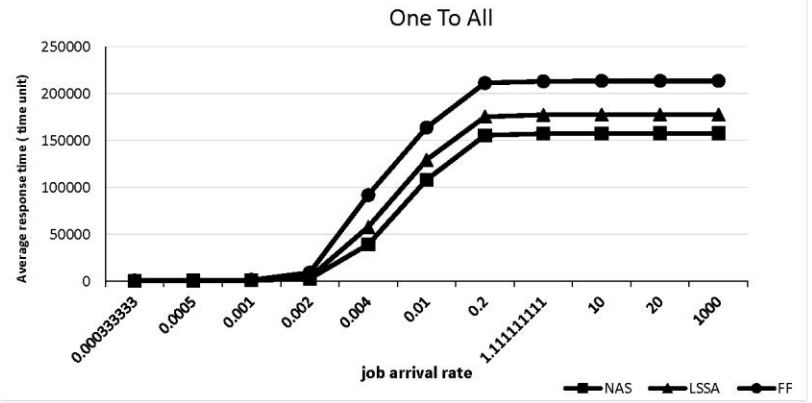


Figure 4-8: Average response time vs. arrival rate using the **one_to_all** communication pattern and **uniform decreasing distribution for job size** in a 16x16 mesh.

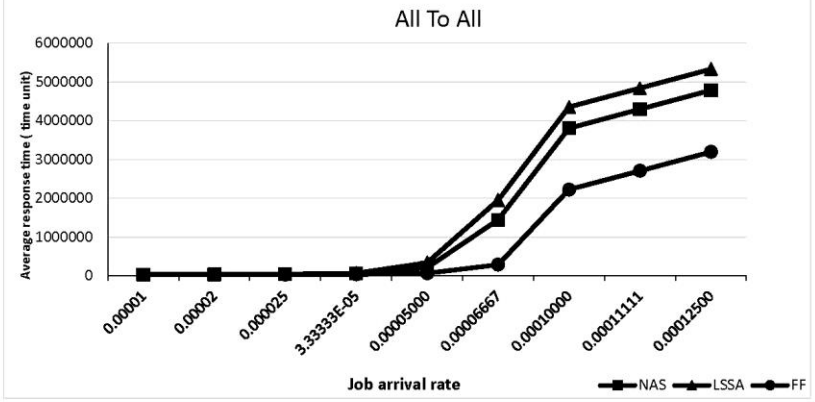


Figure 4-9: Average response time vs. arrival rate using the **All-to-All** communication pattern and **uniform distribution for job size** in a 16x16 mesh.

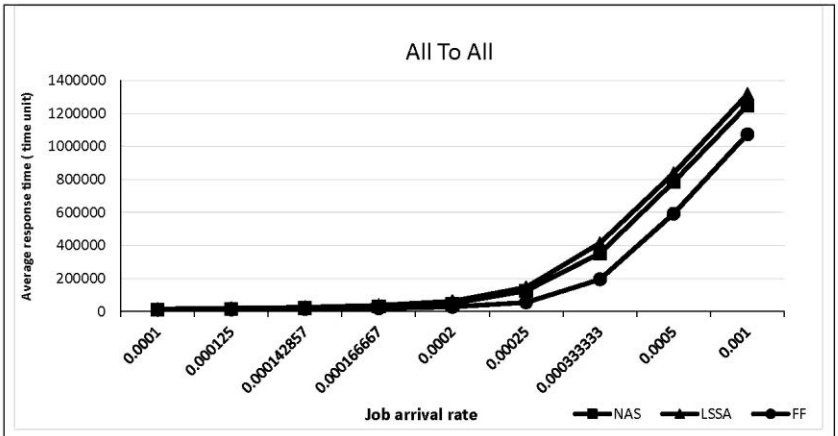


Figure 4-10: Average response time vs. arrival rate using the **All-to-All** communication pattern and **uniform decreasing distribution for job size** in a 16x16 mesh.

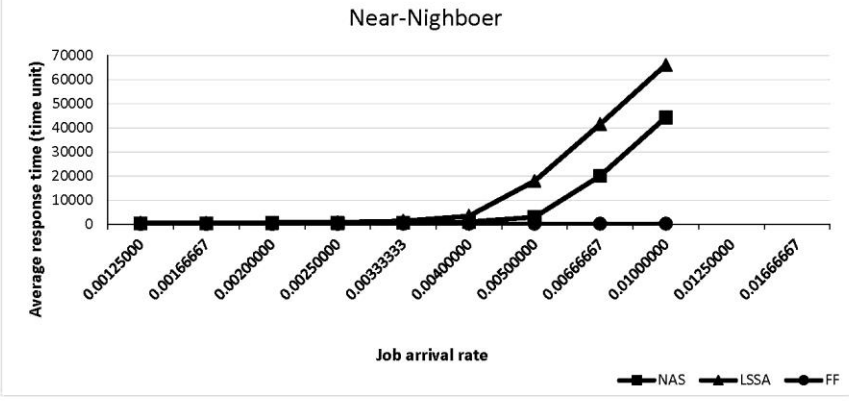


Figure 4-11: Average response time vs.arrival rate using the **near neighbor** communication pattern and **uniform distribution for job size** in a 16x16 mesh.

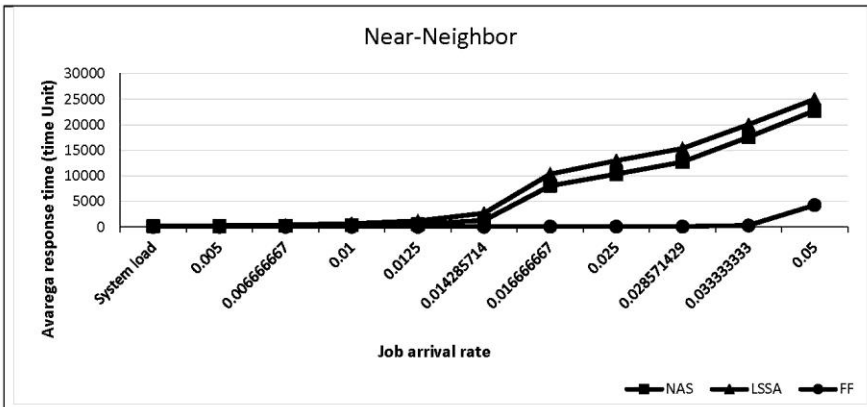


Figure 4-12: Average response time vs. arrival rate using the **near neighbor** communication pattern and **uniform decreasing distribution for job size** in a 16x16 mesh.

Chapter 5
Conclusions and Directions for Future Work

Conclusions

Recently, the topic of processors allocation in mesh-connected multicomputers become the subject for many researches because of the characteristics of mesh topology (Ding, Bhuyan, 1993; Yoo, & Das, 2002; Kumar, Grama, Gupta, & Karypis, 2003; Bani Mohmmad, 2008; Ababneha, Bani Mohmmad, & Ould Khaoua, 2010).

The allocation strategies developed for 2D-mesh connected multicomputers are classified into two types: contiguous and Non-Contiguous allocation schemes.

In contiguous allocation schemes, the processors assigned to the request are physically adjacent, and in some allocation schemes, they should have the same shape of the mesh-system (Chuang, & Tzeng, 1994; Lo, Windisch, Liu, & Nitzberg, 1997; Seo, & Kim, 2003; Bani Mohmmad, Ababneha, 2018). Contiguous allocation schemes suffer from two types of fragmentation problems, external and internal fragmentations. External fragmentation exists when the requested number of processors available in the system however, the allocation scheme fails to allocate the job request because it is not contiguous. Internal fragmentation exists when more processors allocated to the job than requested (Lo, Windisch, Liu, & Nitzberg, 1997; Seo, & Kim, 2003; Bani Mohmmad, 2008).

In Non-Contiguous allocation schemes, the job is executed on a number of small sub-meshes instead of waiting for one sub-mesh of the requested shape to be available. The goal of these strategies is to improve system performance by minimizing communication overhead. This is accomplished by preserving a good degree of contiguity among the allocated processors. This twist, can reduce processors fragmentation and increase system utilization, however it causes high communication overhead (Seo, & Kim, 2003; Bani-Mohammad, Ould-Khaoua, Ababneh, & Mackhenzie, 2007; Bani Mohmmad, 2008; Bani Mohmmad, Ababneha, 2018). Generally, the aim of any allocation strategy is to improve system performance by maximizing system utilization and minimizing average response time of jobs (Lo, & Windisch, Liu, & Nitzberg, 1997; Seo, & Kim, 2003; Bani Mohmmad, 2008).

Motivated by the previous observations, a new Non-Contiguous allocation strategy for 2D-mesh-connected multicomputers, referred to as Neighbor Allocation Strategy (NAS) has been proposed. The goal of this strategy is to remove both of internal as well as external fragmentation, while preserving a good degree of contiguity among the allocated processors in order to enhance system performance in both system utilization as well as average response time of jobs, which is the goal of any allocation scheme.

In NAS allocation strategy, the job request is allocated to a number of small sub-meshes, which have a degree of contiguity among them. In the first step, the nucleus sub-mesh is constructed, and then the remaining of the required number of processors are allocated as neighbors to the nucleus sub-mesh or the other allocated sub-meshes. NAS rebuilds the job request to be accommodated into the available free sub-meshes in the system and always it allocates the job request contiguously in order to alleviate the communication overhead as possible, and hence maximize system utilization and minimize job response time.

The simulation experiments for the proposed NAS allocation strategy have been conducted and compared with the contiguous allocation scheme FF (Zhu, 1992) and the Non-Contiguous allocation scheme LSSA (Seo, & Kim, 2003). The outcomes show that the performance of the NAS allocation scheme when system utilization considered is much better than the allocation schemes considered in this research work for the two job size distributions, when the 3 patterns are used; All-to-All, one-to-all, and near neighbor. This is because NAS has better capabilities than the previous allocation schemes considered in this research to overcome fragmentation in the system.

The outcomes also show that the performance of NAS when considering average-response time is better than that of all other allocation schemes for both job size distributions considered in this research, when the one_to_all is used. This is because of the ability of NAS to remove external fragmentation better than the other allocation algorithms considered, which increases the probability of successful allocation for job requests and decreases the overlapping between the exchanged messages among the allocated processors. On the other hand, when All-to-All and near neighbor are used, the contiguous FF allocation scheme has the superior performance over all other Non-Contiguous schemes considered (Al-Harafsheh, 2016). This is because contiguous allocation schemes allocate a rectangular sub-mesh for the job request, and this minimizes inter-job interference. However, NAS is better than LSSA because of its ability to remove external fragmentation, and maintaining a higher degree of contiguity, which improves the system performance.

Directions for Future Work

The aim of any allocation strategy is to minimize average response time and maximize system utilization. In this research work, the performance of the NAS allocation schemes was investigated in 2D-mesh interconnection networks. The outcomes show that the performance of the proposed NAS allocation scheme is encouraging compared with the allocation schemes considered in this work for the 2 job distributions considered when the All-to-All, one-to-all, and near neighbor were used. As an extension to this, it could be viable to try the NAS Non-Contiguous allocation scheme for use in the 3Dmesh connected multicomputers. More over an enhancement on NAS seems promising in order to increase system performance considering average response time for jobs.

References

Ababneha, I., Bani Mohammadb, S., & Ould Khaoua, M. (2010). All Shapes Contiguous Submesh Allocation for 2D Mesh Multicomputers, International Journal of Parallel, Emergent and Distributed Systems archive, Vol. 25, no. 5, pp. 411-422.

Al Harafsheh, R. (2016). Irregular Shape Strategy for Non-contiguous Sub-mesh Allocation in 2D Mesh-Connected Multicomputers, Master Thesis, Department of Computer Science ,AL Albayet University, Jordan.

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

Bani Mohammad, S., Ababneh, I. (2018). Improving system performance in non-contiguous processor allocation for mesh interconnection networks, Simulation Modelling Practice and Theory, Vol. 80, pp 19-31.

Bani Mohammad, S., Ababneh, I. (2015). A New Compacting Non-Contiguous Processor Allocation Algorithm for 2D Mesh Multicomputers, Journal of Information Technology Research, Vol. 8, pp 19.

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

Bani Mohammad, S., Ababneh, I., Hamdan, M. (2011). Performance evaluation of noncontiguous allocation algorithms for 2D mesh interconnection networks. Journal of Systems and Software vol. 84, no. 12, pp. 2156-2170.

Bani Mohammad, S., Ould Khaoua, M., Ababneh, I., & Mackhenzie, LM. (2007). An Efficient Processor Allocation Strategy that Maintains a High Degree of Contiguity among Processors in 2D Mesh Connected Multicomputers, 2007 ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2007), IEEE Computer Society Press, Phiadelphia University, Amman, Jordan, pp. 934-941.

Chuang, P., & Tzeng, N. (1994). Allocating precise submeshes in mesh connected systems, IEEE Transactions on Parallel and Distributed Systems, vol. 5, no. 2, pp. 211-217.

Ding, J., & Bhuyan, L. (1993). An Adaptive Submesh Allocation Strategy for Two Dimensional Mesh Connected Systems, Proceedings of the 1993 International Conference on Parallel Processing, vol. 2, pp. 193-200.

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

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

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

ProcSimity v4.3. (1996). ProcSimityv4.3 User's Manual. University of Oregon.

Seo, K., & Kim, S. (2003). Improving system performance in contiguous processor allocation for mesh-connected parallel systems, The Journal of Systems and Software vol. 67, no. 1, pp. 45–54.

Yoo, & Das, (2002). A Fast and Efficient Processor Allocation Scheme for Mesh-Connected Multicomputers, IEEE Transactions on Parallel & Distributed Systems, vol. 51, no. 1, pp. 46-60.

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