بسم الله الرحمن الرحيم

Al al-Bayt University

Prince Hussein Bin Abdullah College for Information Technology

Computer Science Department

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

التخصيص غير المتجاور باستخدام التحشير في متعددات الحواسيب ثنائية الأبعاد

by

Mohammad H. Yassen

Supervisor: Dr. Saad Bani-Mohammad
Co-supervisor: Prof. Ismail Ababneh

A Thesis submitted to the
Deanship of Higher Educations in Partial Fulfillment of the Requirements
for the Degree of Master in Computer Science

Mafraq, Jordan

December, 2012

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

التخصيص غير المتجاور باستخدام التحشير في متعددات الحواسيب ثنائية الأبعاد

اعداد الطالب
محمد حسين محمد ياسين
0920901011

اشراف الدكتور
سعد بني محمد

اعضاء لجنة المناقشة	التوقيع
1. الدكتور سعد بني محمد (مشرفاً ورئيساً)	.....
2. الاستاذ الدكتور اسماعيل عبابنه (مشرفاً مشاركاً)	.....
3. الدكتور خالد بطيحة (عضواً)	.....
4. الدكتور عمر شطناوي (عضواً)	.....
5. الاستاذ الدكتور محمد القطاونه (عضواً)	.....

قدمت هذه الرسالة استكمالاً لمتطلبات الحصول على درجة الماجستير في علم الحاسوب في كلية الامير الحسين بن عبدالله لتكنولوجيا المعلومات في جامعة آل البيت.

نوقشت وأوصي باجازتها بتاريخ : 19/12/ 2012.

Abstract

In non-contiguous allocation, a job request can be split into smaller parts that are allocated possibly non-adjacent free sub-meshes rather than always waiting until a single sub-mesh of the requested size and shape is available. Lifting the contiguity condition is expected to reduce processor fragmentation and increase system utilization. However, the distances traversed by messages can be long, and as a result the communication overhead, especially contention, is increased. The extra communication overhead depends on how the allocation request is partitioned and assigned to free sub-meshes. In this research, a new non-contiguous processor allocation strategy, referred to as A Compacting Non-Contiguous Processor Allocation Strategy (CNCPA), is suggested for the 2D mesh networks. In the proposed strategy, a single job is compacting into more than one free location within the allocated processors, where the remaining available processors (free processors) form a large sub-mesh in the system. To evaluate the performance improvement achieved by the proposed strategy and compare it against well-known existing non-contiguous strategies, we conducted extensive simulation experiments under the assumption of wormhole routing and the communication patterns, one-to-all, random and near neighbor. The results show that the proposed strategy eliminates both the internal and external fragmentation and reduce the communication overhead and hence improves performance in terms of job turnaround time and system utilization.

Inscription

I present this thesis To …….

the best whom I have ever met

the one who lighten my path

who lead me to pass with no fear

To my mother

To the one who taught me the proud and bravery

Who make me decisive

Who told me there is not anything out of reach

To my father

To My inspiration

My light, My peace

To my sisters and brothers

To My path mates

The loyalty and true love symbols

To my friends

Thankfulness

I am thankful for all those who helped us achieving this thesis, which we worked hardly to do it very well. This thesis considered as an effective result for our huge lonely efforts lasted the study years. Those efforts are required in the computer world, that needs a specific research and deep study.

I would like to thank my supervisors (Dr. Saad Bani-Mohammd) and (Prof. Ismail Ababneh). They were my reliable source that gave me the accurate information and exact require i need, so widely.

Thanks to all of the teaching staff in Prince Hussein Bin Abdullah collage for Information Technology.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, parallel computers have become very popular for solving large-scale computationally intensive problems [20, 32]. Parallel computing is a form of computation in which many calculations are carried out simultaneously. In parallel computing, we can save time, solve larger problems, and reduce cost by using multiple "cheap" computing resources instead of paying for time on a supercomputer [20].

A parallel computer is defined as a set of processors that cooperate together in order to find a solution for the computation problem. Parallel computers solve problems relatively quickly when compared to conventional computers [32, 38, 39].

Parallel computers are divided into two classes: (1) parallel computers with shared memory and (2) parallel computers with distributed memory [38]. In shared memory computers, also known as multiprocessors, all processors communicate together via a shared memory. On the other hand, in distributed memory computers, also known as multicomputers, processors communicate by means of interchanging messages through an interconnection network [39].

An interconnection network is a network that transports data between individual components of a parallel computer to accomplish tasks collectively. The network consists of many elements including buffers, channels, switches, and controllers that work together to provide and deliver data. Interconnection networks can be categorized into two main categories: direct interconnection networks and indirect interconnection networks [45, 39].
In direct interconnection networks, also called point-to-point networks, each node has a point-to-point connection to one or more nodes that are called neighbors, allowing for direct communication between these nodes. Examples of direct networks are the mesh [1, 34], k-ary n-cube [28, 34, 39], and hypercube [34, 39]. These examples are

common direct interconnection networks that have been implemented in commercial and experimental machines [34, 39]. In indirect networks, multiple intermediate stages of switches are used to interconnect the nodes of a multiprocessor. Examples of indirect networks include the crossbar, bus, and multistage interconnection networks [39].

Direct interconnection networks have been extensively employed in large-scale multicomputers because of their scalability. They can be scaled up by adding nodes and channels based on the predefined network structure [1, 39]. Moreover, direct interconnection networks are able to exploit communication locality (near neighbor communication) that is exhibited by many real-world applications.

A 2D mesh interconnection network is a mesh-connected multicomputer, with processors having the order of nodes in the network, and point-to-point that connects each node to its neighbors. Mesh interconnection network is a special case of k-ary n-cube networks as the number of dimensions, n is two [51].

Figure 1.1 shows an example of a 6×5 2D mesh, where allocated processors are denoted by black circles and free processors are denoted by white circles.

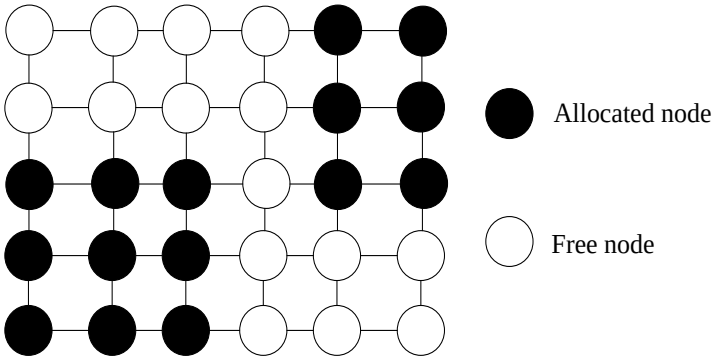

Figure 1.1: An example of a 6X5 2D mesh

Mesh multicomputers are suitable for different applications such as matrix computations, image processing and problems whose task graphs can be embedded naturally into the mesh. Examples of parallel computers that use mesh as an interconnection network include Tera Computer, Cray T3D, MIT J-Machine and the IBM BlueGene/L [5, 38, 39].

1.1. Processor Allocation

Processor allocation and job scheduling are critical for exploiting the full computing power of a multicomputer. Job scheduling involves the scheduling discipline used at the job level [37, 48]. It controls the selection of the next job for which processors are to be allocated. Processor allocation involves the assignment of a collection of processors to a parallel job with the goal of maximizing utilization and minimizing processor fragmentation over a stream of jobs [28, 37].

Processor allocation strategies are divided into two categories: *contiguous* and *non-contiguous*. In contiguous allocation, jobs are allocated distinct contiguous processor sub-meshes for the duration of their execution. Contiguous allocation has the problem of processor fragmentation [28, 30, 40, 42].

Processor fragmentation can be classified into *internal* and *external* fragmentation. Internal fragmentation occurs when more processors are allocated to a job more than it requires [10, 28, 29, 46, 49]. When a job is assigned more processors than it requires, the extra allocated processors are not used for actual computation, instead they are wasted. External fragmentation occurs when a sufficient number of processors are available to satisfy a request, but they cannot be allocated because they are not contiguous for example [29]. Figure 1.2 shows the internal fragmentation of 2 processors, and figure 1.3 shows the external fragmentation of 4 processors assuming that the contiguous allocation algorithm is applied.

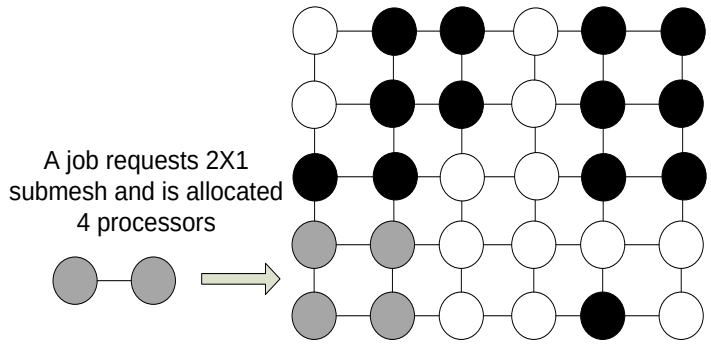

Figure 1.2: An internal fragmentation of 2 processors

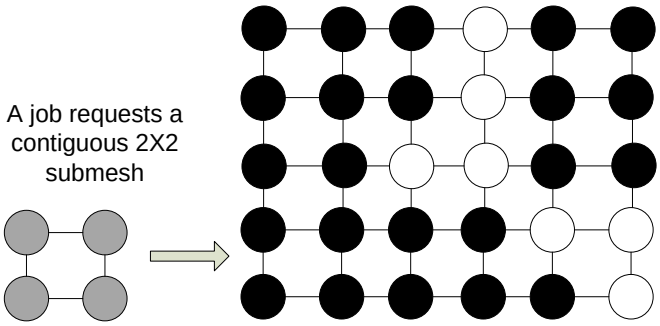

Figure 1.3: An external fragmentation of 4 processors assuming that the contiguous allocation algorithm is applied

A lot of research has been carried out to solve the problem of external fragmentation. For example, non-contiguous allocation has been considered [10, 12, 29, 39, 46, 53]. In non-contiguous allocation a job can be executed on multiple disjoint smaller sub-meshes rather than waiting until a single sub-mesh of the requested size becomes available [10, 12, 29, 39, 42, 46, 53]. Non-contiguous allocation outperforms contiguous allocation because it can minimize the external fragmentation [39]. Although non-contiguous allocation increases message contention inside the network, dropping the contiguity condition can reduce processor fragmentation and increase system utilization [39]. In general, the main goal of any processor allocation strategy is to reduce the job turnaround time and to maximize system utilization [29, 29, 40], where the turnaround time is the time that the job spends in the mesh system from arrival to departure, whereas the system utilization is the percentage of processor that are utilized over time [39, 45].

1.2. Motivation

Previous researchers recommended that a new non-contiguous allocation strategy for mesh-connected multicomputers is needed [10, 29, 39, 40, 46]. The existing non-contiguous allocation algorithms [10, 12, 30, 54] suffer from several problems such as external fragmentation, internal fragmentation, and message contention inside the network [10, 29, 39, 53]. Furthermore, allocation of processors to job requests is not based on free contiguous sub-meshes as in [10, 29]; but rather on artificial predefined geometric or arithmetic patterns. For example, in [10] ANCA subdivides job requests into two equal parts, and the subparts are successively subdivided in a similar fashion if allocation fails for any of them. In [29], MBS bases partitioning on a base-4 representation of the number of processors requested, and partitioning in Paging [29] is based on the characteristics of the page, which is globally predefined independently from the request. Hence these strategies may fail to allocate an available large sub-mesh, which in turn can cause degradation in system performance, such as the turnaround times of jobs [10, 29, 39]. In [41], GABL is based on available processors, regardless of their position in the mesh system which may allocate a job to sub-meshes that are far apart from each other in the mesh system which increases the communication overhead and thus affects the performance in terms of turnaround time [29]. Figures 1.4 and 1.5 show an example of a 4×4 2D mesh. In figure 1.4, the job requests a 5×1 sub-mesh, and this job request is not allocated contiguously because there is no available sub-mesh of size 5×1 in the mesh system. According to the GABL strategy, the job request is divided into two sub-requests, the first one is 4×1, which is allocated to available sub-mesh as shown in figure 1.5, and then the second sub-request 1×1 is allocated in another sub-mesh [29].

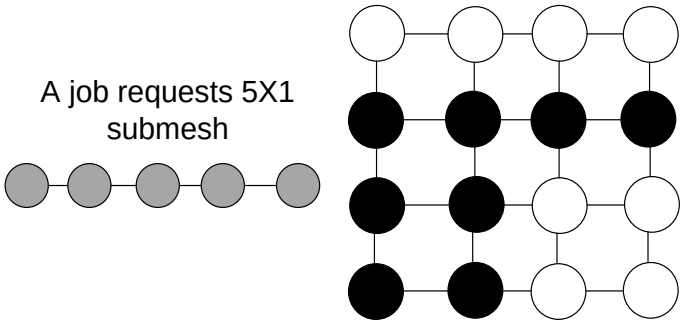

Figure 1.4: A job requests 5X1 2D sub-mesh in 4X4 mesh

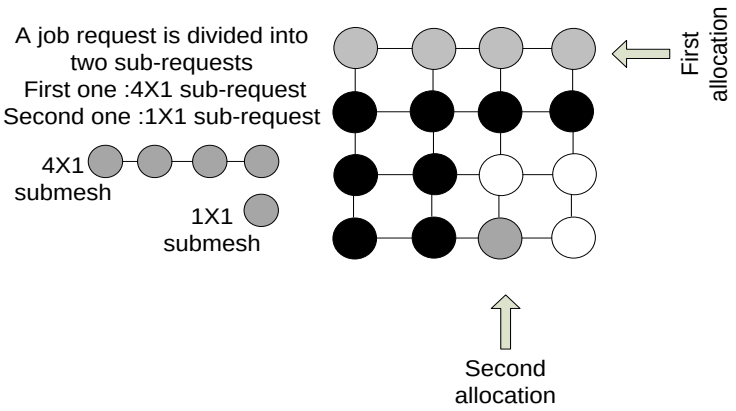

Figure 1.5: a job request 5X1 is divided into two sub-requests in GABL allocation algorithm

1.3. Thesis Statement

All current allocation strategies used in mesh-connected multicomputers can be classified into two categories: contiguous and non-contiguous. The existing contiguous allocation strategies manage to achieve complete sub-mesh recognition capability but at the expense of high processor fragmentation. On the other hand, most existing non-contiguous allocation strategies suffer from several problems that include internal fragmentation, external fragmentation, and message contention inside the network. Also, most existing non-contiguous allocation strategies do not exploit knowledge of the current state of the system (e.g., currently available sub-meshes).

In this thesis, we propose a new non-contiguous processor allocation strategy for 2D mesh connected multicomputers, referred to as A Compacting Non-Contiguous Processor Allocation Strategy (CNCPA). The proposed strategy compacts a single job into more than one free location within the allocated processors and it is expected to improve the system performance in terms of average turnaround time and mean system utilization.

1.4. Main Contribution

To address the above research interests listed in section 1.2 (motivation section), this thesis presents a new non-contiguous allocation strategy that overcome the limitations of the existing strategies suggested previously for the 2D mesh networks.

In this research, a new non-contiguous allocation algorithm, referred to as A Compacting Non-Contiguous Processor Allocation Strategy (CNCPA for short), for the 2D mesh-connected multicomputer is suggested. The CNCPA strategy combines the desirable features of both contiguous and non-contiguous allocation. For example, the desirable features of contiguous allocation include the elimination of the communication overhead between processors allocated to a parallel job, and achieving complete sub-mesh recognition capability. The desirable features of non-contiguous allocation are reducing processor fragmentation. Moreover, CNCPA is general enough in that it could be applied to either the 2D or 3D mesh. However, for the sake of the present discussion, the new non-contiguous allocation strategy is adapted for the 2D mesh in order to compare its performance against that of the existing non-contiguous allocation strategies suggested for the 2D mesh; it is worth pointing out that there has been hardly any non-contiguous allocation strategy which has been suggested for the 3D mesh network.

The proposed CNCPA strategy relies on a new approach that compacts a single job into more than one free locations within the allocated processors, where the remaining available processors (free processors) form a large sub-mesh in the system, and maintains a higher degree of contiguity among sub-meshes than that of the previous non-contiguous allocation strategies [10, 29, 39]. This decreases the number of sub-meshes allocated to a job, hence the distance traversed by messages is decreased, which in turn decreases the communication overhead. Our simulation results indicate that CNCPA has better performance in terms of the turnaround time than the previous non-contiguous allocation strategies proposed in [29]. CNCPA is able to eliminate internal as well as external fragmentation from which several previous allocation strategies suffer.

1.5. Thesis Structure

The remainder of the thesis is organized as follows. Chapter 2 describes some non-contiguous allocation strategies that have been proposed for mesh-connected multicomputers and presents the system model assumed in this research. A list of assumptions used in this research is also provided. Finally, the chapter justifies the selection of simulation as a study tool and describes the method of study used in this research.

Chapter 3 introduces A Compacting Non-Contiguous Processor Allocation Strategy as a new non-contiguous allocation algorithm for 2D mesh-connected multicomputers, and discusses the main features of the proposed strategy.

Chapter 4 discuses and analyzes the simulation experiments were carried out in order to evaluate the performance of the proposed strategy and compare it against existing well-known non-contiguous allocation strategies.

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

Chapter 2

Related work

The main objective of this chapter is to describe some of the existing non-contiguous allocation strategies that have been proposed in the literature [10, 12, 18, 26, 29, 36, 39, 53, 54] for 2D mesh networks.

2.1. Related Non-Contiguous Allocation Strategies

Non-contiguous allocation allows jobs to be executed when the number of available processors is sufficient [10, 29, 40, 46]. Some of the non-contiguous allocation strategies that have been suggested in the literature are described below.

Random: Random allocation is a straightforward strategy in which a request for a given number of processors is satisfied with a number of processors selected randomly [29]. Both internal and external fragmentations are eliminated since all jobs are assigned exactly the requested number of processors, if available. Because no type of contiguity is enforced in this strategy, high communication interference amongst jobs would be expected [29, 39].

Greedy-Available Busy List Allocation Strategy (GABL): In this strategy [29], when a parallel job is selected for allocation, a sub-mesh suitable for the entire job is searched. If such a sub-mesh is found, it is allocated to a parallel job and allocation is done. Otherwise, the largest free sub-mesh that can fit inside S(α, β) is allocated, where α and β are the dimensions of the job request. Then, the largest free sub-mesh whose side lengths do not exceed the corresponding side lengths of the previous allocated sub-mesh is searched under the constraint that the number of processors allocated does not exceed α X β. This last step is repeated until α X β processors are allocated. For example, given the system state shown in Figure 2.1 and a job that requests the allocation of an 8 × 2 sub-mesh, contiguous allocation is not possible and non-contiguous allocation is adopted. The job is allocated the sub-meshes (0, 0, 5, 1) and (2, 2, 3, 3) as follows. Firstly, the algorithm subtracts one from the maximum

length of the side lengths of the job request resulting in 7 × 2 sub-mesh which is not available for allocation in the mesh system. So the subtraction process is repeated again resulting in a 6 × 2 sub-mesh which is available for allocation in the mesh system, so that the sub-mesh (0, 0, 5, 1) is allocated to the job request. Then, the algorithm tries to allocate a sub-mesh whose side lengths do not exceed the corresponding side lengths of the previous allocated sub-mesh (6 × 2) if this does not result in allocating more processors than the original allocation request (8 × 2); in this example, [(6 × 2) + (6 × 2)] > (8 × 2). The algorithm subtracts one from the maximum lengths of 6 × 2 resulting in 5 × 2, but again [(6 × 2) + (5 × 2)] > (8 × 2). So the subtraction process is repeated again until it gets a sub-mesh whose processors, along with the processors of the previous allocated sub-mesh, are less than or equal the number of processors requested by the original request (8 × 2). In this case, a 2 × 2 sub-mesh results from the subtraction process which is available in the mesh system so that the sub-mesh (2, 2, 3, 3) is allocated to the job request.

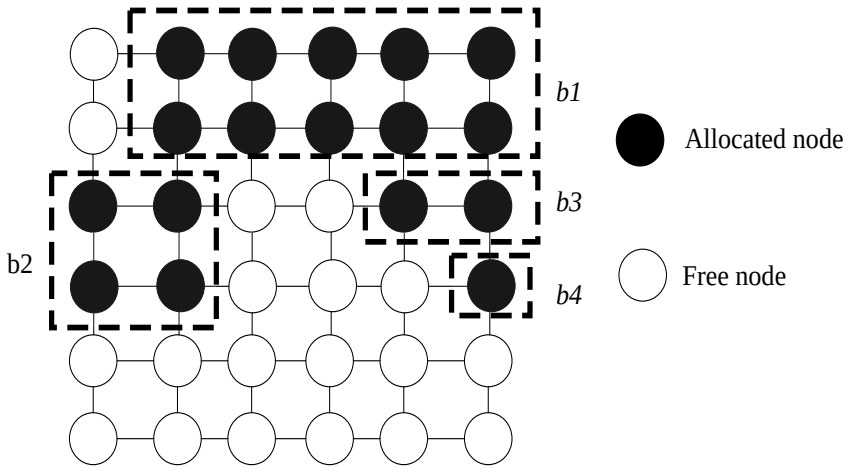

Figure 2.1: A 6 × 6 sub-mesh with 19 free processors forming several free sub-meshes [29]

Allocated sub-meshes are kept in a busy list. Each element in this list includes the id of the job to which the sub-mesh is allocated. When a job departs the system its allocated sub-meshes are removed from the busy list and the number of free processors is updated. Allocation in GABL is implemented by the algorithm outlined in Figure 2.2, while the de-allocation algorithm is outlined in Figure 2.3. Note that allocation always succeeds if the number of free processors is $\geq \alpha \, X \, \beta$. Moreover, it can be noticed that the methodology used for maintaining contiguity is greedy. GABL attempts to allocate large sub-meshes first.

Procedure GABL_Allocate (α, β):

Begin {

Total_Allocated = 0

Job_Size = $\alpha \; X \; \beta$

Step1. If (number of free processors < Job_Size)

return failure.

Step2. If (there is a free S(w, l) suitable for S(α, β))

{

allocate it using the TBL contiguous allocation algorithm.

return success.

}

Step3. αnew = α and βnew = β

Step4. Subtract 1 from max (αnew, βnew) if max > 1

Step5. If (Total _allocated + αnew × βnew > Job_Size) go to step 4

Step6. If there is a free S (w, l) suitable for S(αnew, βnew)

{

Allocate it using TBL contiguous allocation algorithm.

Total_allocated = Total_allocated + αnew × βnew

}

Step7. If (Total_allocated == Job_Size)

return success.

else

go to Step 5.

} End.

Figure 2.2: Outline of the Greedy Available Busy List allocation algorithm

Procedure GABL_De-allocate ():

Begin *{*

jid = id of the departing job;

For all elements in the busy list

if (element's id = jid)

remove the element from the busy list

} End.

Figure 2.3: Outline of the Greedy Available Busy List de-allocation algorithm

Paging Strategy: In this strategy [30], the entire 2D mesh is divided into pages that are square sub-meshes with equal side lengths of (2^{size_index}), where *size_index* is a positive number. A page is the allocation unit. The pages are indexed according to several indexing schemes (row-major, shuffled row-major, snake-like and shuffled snake-like indexing), as shown in Figure 2.4. An ordered list is used to keep track of all unallocated pages. The pages are sorted in the increasing order of their order indices, assigned by the indexing scheme. Each entry in the list contains the corresponding page's row and column indices, and the page's order index. The number of pages a job requests is computed as: [29, 38].

$$P_{request} = \lceil (\alpha \times \beta)/Psize \rceil \quad \text{(2.1)}$$

where *Psize* is the size of the page, and α and β are the side lengths of the requested submesh. If the number of free pages is greater than or equal to $P_{request}$, the first $P_{request}$ unallocated pages are removed from free list and allocated to the requesting job. When a job is de-allocated, pages occupied by it are merged back into the free page list**.** A paging strategy is denoted as Paging (*size_index*). For example, Paging(2) means that the pages are 4×4 sub-meshes.

Paging suffers from internal fragmentation when *size_index* > 0. The internal fragmentation of running jobs is given by:

$$Internal\_Fragmentation = \frac{\Sigma_{jobs} lost\_processors}{\Sigma_{jobs} Allocated\_processors} \quad \text{(2.2)}$$

where *Lost_ Processors* is for a parallel job that requests *Job_Size* processors, but is allocated *Number_of_Allocated_Pages* . It is calculated using:

Lost_ Processors = *Number_of_Allocated_Pages* x *Psize* - *Job_Size* (2.3)

To illustrate this, consider a paging strategy with *size_index* = 1, and suppose a parallel job requests the allocation of a 3×3 sub-mesh. When allocation is carried out for the job it is allocated 3 pages (12 processors). Since only 9 processors are needed there is an internal fragmentation of 25%.

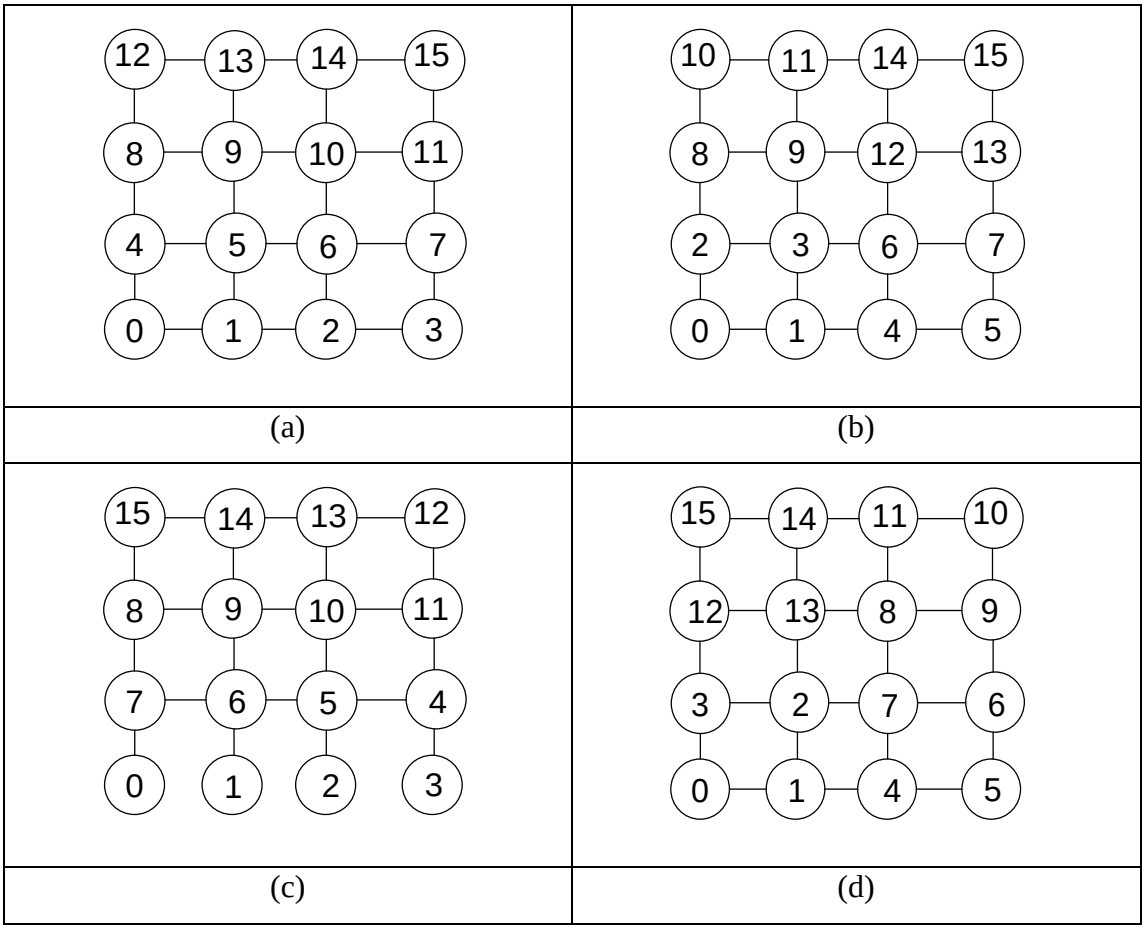

Figure 2.4 : Paging(0) using different indexing schemes: (a) Row-major indexing, (b) Shuffled row-major indexing, (c) Snake-like indexing, and (d) Shuffled snake-like indexing

In this research, only the row-major indexing scheme is considered because the remaining indexing schemes exhibit only a slight impact on the performance of paging, as revealed in [29]. The Paging allocation and de-allocation algorithms are presented in Figures 2.5 and 2.6, respectively [29, 39].

// Page_Side = 2^{size_index}*; Psize = Page _ Side* X *Page _ Side*

// The parameter jid is the id of the job that is being considered for allocation

// α and β are the side lengths of the job's allocation request

Procedure Paging_Allocation (jid, α, β)

Begin {

Job_Size = α x β

$P_{request} = \lceil Job\_Size/Psize \rceil$

// Allocation:

Step1. if (number of free pages < Prequest) return failure else go to step 2

Step2. allocate the first Prequest pages from the list of unallocated pages to the job, setting the IDs of these pages to jid, and return success.

} End

Figure 2.5: Outline of the Paging allocation algorithm

// jid: id of departing job;

Procedure Paging_De-allocation (jid):

Begin {

for all allocated pages

if (page's id == jid)

de-allocate the page and add it to the list of unallocated pages

} End

Figure 2.6: Outline of the Paging de-allocation algorithm

Multiple Buddy Strategy (MBS): In this strategy [29], the mesh is divided into non-overlapping square sub-meshes with side lengths equal to powers of 2 upon initialization. MBS maintains free block records (FBR) for all free processor squares of the same size. The entry FBR[i] contains the number of available squares of size 2^i x 2^i , and an ordered list of the locations of these squares. The number of processors, p , requested by an incoming job is represented as a base 4 number of the following form:

$$\sum_{i=0}^{\lfloor \log_4 p \rfloor} d_i \times \left(2^i \times 2^i\right) \quad \text{..... (2.4)}$$

where $0 \le di \le 3$. This strategy attempts to satisfy every term i in the request with d_i free processor blocks of sizes equal to 2^i x 2^i processors using FBR. If a required block is unavailable, MBS searches for a larger block in FBR and repeatedly breaks it down into 4 adjacent buddies until it produces blocks of the desired size. The 4 buddies of a 2^j x 2^j block are 2^{j-1} x 2^{j-1} blocks. If that fails, MBS breaks the request for a 2^i x 2^i block into 4 smaller requests for 2^{i-1} x 2^{i-1} blocks and repeats the allocation process. In this algorithm, allocation always succeeds when the number of free

processors in the mesh system is sufficient. This is because the request, or parts of it, can be partitioned into requests for 1×1 blocks.

Adaptive Non-contiguous Allocation (ANCA): This strategy [10] aims to reduce the effects of the fragmentation problem. ANCA first attempts to allocate a job contiguously. When contiguous allocation fails, it breaks a job request into two equal size sub-frames (sub-requests). For example, an 8 X 3 request is partitioned into two 4X3 sub-requests. These sub-requests are then allocated available sub-meshes, if possible. Otherwise, each of these sub-requests is broken into two equal size sub-requests, and then ANCA tries to assign all sub-frames to available locations and thus take advantage of non-contiguous allocation, and so on. This process terminates if allocation succeeds for all sub-requests, or it has repeated a specified number of times. Moreover, allocation fails if a side length of the sub-requests reaches 1, which can cause external fragmentation [10, 29, 39]. Simulation results in [50] show that ANCA is inferior to the allocation strategies, GABL, MBS, and Paging, and that these strategies, GABL, MBS, and Paging, have the best performance results, expressed in terms of the average turnaround time and mean system utilization performance parameters; therefore we do not consider ANCA strategy in this research.

Adaptive Scan and Multiple Buddy (AS&MB): AS&MB is a hybrid strategy [26]. Firstly, it attempts to allocate a job contiguously using the adaptive scan strategy [23]. When the adaptive scan strategy fails to allocate a job request, it employs the non-contiguous allocation strategy MBS [29] for allocation. Simulation results in [25] show that the performance of AS&MB is almost identical to that of MBS [29] in terms of average turnaround time and average service time (i.e., the average time it takes for jobs to execute once allocated to processors in the mesh system). However, the shorter stride distance in AS increases the allocation time and hence AS&MB is not suitable for large meshes; therefore we do not consider it in this research [25, 26, 29, 39].

Paging variants: In addition to the four indexing schemes considered in [29], the Hilbert and H-indexing space-filling curves have been proposed for ordering processors [12]. In these studies, different page selection heuristics have been used. Given a request for allocating *p* processors, an attempt is first made to find a set of at

least *p* consecutive free processors. If this fails, the set of *p* processors with the smallest range of processor ranks is allocated to the request. The algorithm that looks for the consecutive free processors is First Fit if it looks for the first large enough set, and it is Best Fit if it looks for the smallest one that is large enough for the request. The snake-like, Hilbert and H-indexing orderings, when used with First Fit and Best Fit consecutive set selection, have been evaluated using simulation [12]. They have also been compared to a strategy that minimizes the average pair wise distance between the processors allocated to a request (see Gen-Algorithm in [12]). The results have shown that the Gen-Algorithm performs relatively poorly, and the relative performance of the strategies depends on the communication pattern used.

In the above non-contiguous allocation strategies, the random strategy ignores the contiguity of processors allocated to a job, leading to increases in communication delays. In GABL [39], the allocation is based on available processors, regardless of their position in the mesh system which may allocate a job to sub-meshes that are far apart from each other in the mesh system which may increases the communication overhead and thus affect the performance in terms of turnaround time. In Paging, there is some degree of contiguity because of the indexing schemes used. Contiguity can also be increased by increasing the parameter *size_index*. However, there is internal processor fragmentation for *size_index* > 1 , and it increases with *size_index* [29]. An issue with MBS is that it may fail to allocate a contiguous sub-mesh, although one exists. For example, if a job requests the allocation of 16 processors in the mesh system shown in Figure 2.7. Initially, the request is factorized as 4×4 number, but because there are no 4×4 or larger free blocks the request is partitioned into 4 requests for 2×2 blocks. The 4 lightly-shaded non-contiguous 2×2 blocks shown in this figure may be assigned to the request although a large enough single contiguous free sub-mesh 2×8, denoted in the figure by a dashed rectangle, is available. We can notice from the figure that communication between processors belonging to blocks assigned to this job can interfere with the communication of other jobs. In fact, contiguous allocation is explicitly sought in MBS only for requests with sizes of the form 2^{2n} , where n is a positive integer. As for ANCA, it can disperse the allocated sub-meshes more than is necessary. It requires that allocation to all sub-frames occur in the same partitioning and allocation iteration, skipping over the

possibility of allocating larger sub-meshes for a large part of the request in a previous iteration. Moreover, ANCA halts the partitioning and search processes when a side length reaches 1, which can cause external fragmentation. In the Paging variant that uses *size_index* = 0 , the unit of allocation is a single processor, whereas it can be larger in MBS [29] and ANCA [10]. Any processor allocation strategies like Paging variants that operate at this level of granularity (i.e., a single processor) requires a long time to reach the allocation decision [54]. For large machines such as IBM BuleGene/L, allocation strategies that take a reasonable time for allocation and de-allocation operations were proposed [54]. It is to avoid low allocation granularity that the allocation unit in the IBM BlueGene/L, for example, is the mid plane, which is an $8 \times 8 \times 8$ three-dimensional page [54]. Therefore, the time that the allocation and de-allocation operations take can be reasonable. The drawback with this approach to solving the granularity problem is that internal processor fragmentation can be high.

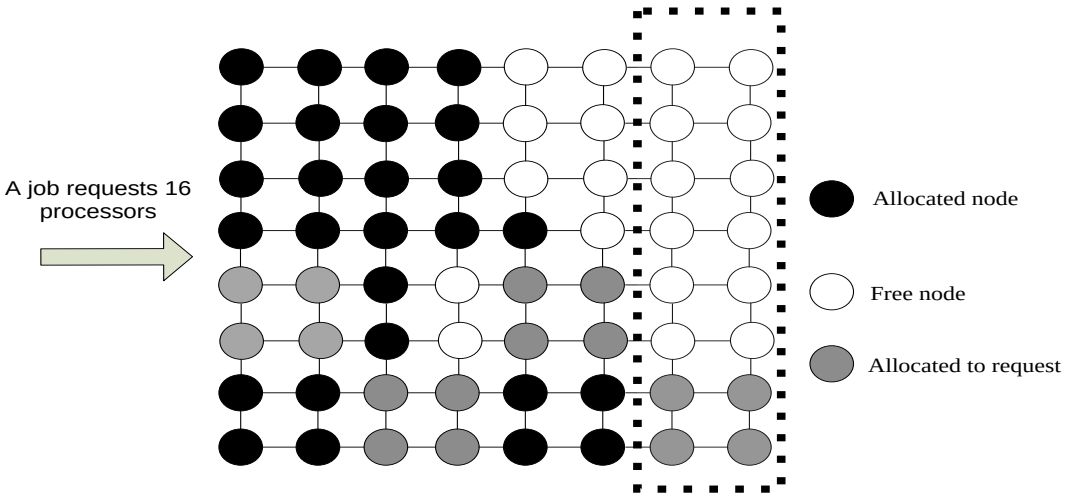

Figure 2.7: An 8 × 8 2D mesh receiving an allocation request for 16 processors in MBS strategy [39]

2.2. System Model

The interconnection network topology describes the way in which the nodes in the network are connected and can be described using an interconnection graph. The vertices of this graph are the nodes while the edges are the physical channels that connect the nodes [1, 14, 39, 52]. The network diameter, node degree, and network degree are often used to characterize a given topology [1, 14, 39]. The diameter is the maximum value of the shortest path lengths between any two nodes. The node degree is the number of links connecting a node to its neighbours while the network degree is

the maximum node degree in the network. Many interconnection network topologies have been suggested for parallel computers, such as the hypercube [3, 26] and the mesh [1, 3, 51]. In a hypercube with d dimensions we have $N = 2d$ nodes each of degree d. The hypercube topologies have many advantages and one of these advantages is its small diameter. However, a main defect of the hypercube network is its lack of scalability, which limits its use in constructing large-size multicomputers [3]. But the scalability and modularity are important parameters of an interconnection network of a multicomputer system. Scalable networks have the property that the size of the system (i.e., the number of communicating nodes) can be increased with minor or no change in the existing configuration [3]. Also, It is expected that the increase in the size of the system that leads to an increase in performance to the extent of the increase in size. [3]. The lack of scalability of the hypercube stems from the fact that the node degree is not bounded and varies by the number of processors in the system (N). This property makes the high cost for a large hypercube N [3, 51].

Motivated by the above observations, a mesh interconnection network is assumed in this research as the network topology. Mesh networks are easily implemented because of the simple regular connection and small number of links per node. Because of the constant node degree, the mesh network is highly scalable. Moreover, the mesh has been widely used in practical multicomputers because it has many advantages such as scalability, structural regularity, simplicity, ease of implementation, and its ability of partitioning [3, 4, 10, 13, 17, 28, 29, 30, 31, 46, 49, 55].

2.2.1. Switching Method

The method of switching determines the way messages are handled as they travel through intermediate nodes. Switching takes place in the router and consists of the receipt of a message, determining the appropriate output channel, and then sending the message through this channel. There are many switching methods, which the three most important ones are *store-and-forward* [53], *virtual cut-through* [8] and *wormhole switching* [6, 8, 10, 29, 52].

- **Store-and-forward switching**: In this method, the message is divided into fixed-length packets that are routed from source to destination. Each packet contains a header that contains the data needed for packet forwarding. A packet is completely stored in each intermediate node before it is forwarded to the next node along its path to the destination. This switching method has two major disadvantages: it requires a large buffer to store entire packets and the time to transmit a message is directly proportional to the distance between the source and destination nodes [35].
- **Virtual cut-through switching**: This method [8, 29] has been introduced as an enhancement to Store-and-forward switching in order to reduce the transmission time. The network latency, especially under low and moderate traffic loads, is noticeably reduced as blocked messages are removed from the network and the channels are simultaneously utilized to transmit unblocked messages. However, the nodes must provide sufficient buffer spaces for all blocked messages passing through it and multiple messages may become blocked simultaneously, so a very large buffer space is required at each node. Therefore, virtual cut-through might be costly to implement due to the high buffer requirement which also has a strong adverse effect on the router speed and on the cost and the size of multicomputer system [24, 31, 39].
- **Wormhole switching**: The drawback of virtual cut-through has encouraged researchers to use of its variant wormhole switching. Wormhole switching has been widely used in practical multicomputers [8, 24] because of its low buffering requirement and good performance. Experimental results in [35] have revealed that network latency in wormhole-switched networks is almost independent from message distance in the absence of message contention for

network resources (buffers and channels). In this method, the message is divided into a fixed length units, called flits (containing typically a few bytes) and the buffer are expected to store only few flits and not the entire message. A flit is the smallest unit of data transmission in a wormhole routing network. The header flit, which contains the routing information, establishes the path through the network while the remaining data flits follow it in a pipelined fashion. If a channel transmits the header of a message, it must transmit all the remaining flits of the same message before transmitting flits of another message. If the header cannot be routed in the network due to contention for resources, the data flits stop moving and remain spread across the channels where they are, keeping all allocated channels and buffers occupied. As a result, they prevent other messages from using these channels, and this in turn leads to chained blocking in the network with the possibility of serious performance degradation under moderate and heavy loads [1]. One common solution to this problem, especially in meshes, is to force the messages to pass through pre-ordered channels so that a blocking chain can be avoided [1]. Since wormhole routing uses pipelined transmission, it can perform well even in high diameter networks, such as the mesh. Many experimental machines, such as the MIT J-machine [33] and the iWARP [9]; and commercial ones such as the Cray T3D, and the IBM BlueGene/L [4, 5, 11] have used wormhole switching. This switching method is used in this research and we have limited ourselves to wormhole switching because it has been used in the existing non-contiguous allocation strategies [25, 26, 29, 39, 42, 44].

2.2.2. Communication Patterns

When the processors allocated to a parallel job, the messages exchanged among the allocated processors according to a specific communication pattern [29, 39]. When non-contiguous allocation is used, we are concerned in measuring message contention that results from exchanging messages and its effects on overall system performance. Three communication patterns have been considered in this research work in order to evaluate the performance of the proposed non-contiguous allocation algorithm. In the *one-to-all* communication pattern, a randomly selected processor sends a message to

all other processors allocated to the same job. In near neighbor communication pattern, the processors allocated to a job are mapped to a virtual two-dimensional array of a size that is equal to the job's allocation request, each of these processors communicates with its virtual neighbors. In the random communication pattern, randomly selected processors send messages to randomly selected destinations within the set of processors allocated to the same job. These three communication patterns were used in previous related studies [25, 26, 29].

2.3. Assumptions

In the following chapters, extensive simulation results will be presented to evaluate the performance of our proposed non-contiguous allocation strategy. In this research, we make the following assumptions which have been commonly used in the literature [10, 17, 18, 19, 23, 25, 26, 29, 30, 31, 37, 40, 41, 42, 43, 44, 45, 46, 49, 50, 55].

- The inter-arrival times of jobs are independent and follow an exponential distribution.
- Jobs are scheduled on a First-Come-First-Served (FCFS) basic.
- The side lengths of the sub-meshes requested by jobs are generated independently and follow a given probability distribution. Two distributions have been considered in this research. The first is the uniform distribution over the range from one to the mesh side length. The second one is the uniform-decreasing distribution, that is based on four probabilities p1 , p2 , p3 , and p4 and three integers l1, l2, and l3, where the probabilities that the width/height of a request falls in the ranges [1, l1], [l1 + 1, l2], [l2 + 1, l3], and [l3 + 1, L] are p1, p2, p3, and p4, respectively. The side lengths within a range are equally likely to occur.
- Messages are transmitted inside the network using wormhole switching along with XY routing [1, 8, 39, 40, 41, 52].

2.4. The Simulation Tool (ProcSimity Simulator)

This section contains a short description of the ProcSimity simulation tool [37]. This tool is discrete-event simulation software [2] that has been developed as a research tool in the area of processor allocation and job scheduling in multicomputers [37]. ProcSimity was developed at the University of Oregon [37]. The language used for writing this software was the C programming language and the ProcSimity has been extensively used for processor allocation and job scheduling in mesh-connected multicomputers [23, 25, 27, 29, 37, 42]. This is because it is a open-source and includes detailed simulation of important operations of multicomputer networks [37]. The general purpose of the ProcSimity is to provide a suitable environment for performance analysis of processor allocation and job scheduling algorithms. Especially, ProcSimity was designed to study some of the problems of processor allocation, such as fragmentation and communication overhead problems [23, 25, 29, 37, 42]. The architecture modeled by ProcSimity consists of a network of processors connected to each others through message routers. Adjacent nodes are connected by bidirectional communication links, and messages can be sent either by store-and-forward or wormhole switching. The ProcSimity supports both the mesh and k-ary n-cube interconnection network topologies with dimension-ordered routing [37, 29]. The ProcSimity simulator specifies the target machine environment, including the network topology, routing, and flow control mechanism, and it involves the selection of a scheduling and an allocation algorithm from a set of provided algorithms. Also, third-party scheduling and allocation strategies can be integrated into ProcSimity. ProcSimity also involves determining the simulation experiments; it supports both stochastic job streams as well as communication patterns from actual parallel applications. In this tool, the user can specify the detailed simulation of message-passing overhead at the flit level [37, 39]. When ProcSimity simulates a mesh-connected multicomputer, independent user jobs that arrive at the system, request sub-meshes of free processors. If the number of free processors in the mesh system is not enough to satisfy the job request, or there are other waiting jobs in the queue, the job is diverted to the waiting queue. The job which is to be executed is selected from the waiting queue based on the underlying scheduling strategy, and then the processor allocation algorithm determines and allocates the set of processors on which the job will execute. The allocated processors may be contiguous or non-contiguous based on

the allocation strategy used. When a job is allocated a set of processors, it runs there to completion. It may not be moved to other locations during execution [23, 25, 29, 37, 42]. Once a job departs from the system the sub-meshes it is allocated are freed for use by another incoming job.

2.5. Justification of the Method of Study

In this research we choose ProcSimity simulator as a tool of study. So, we will discuss briefly the reasons of choosing this tool in this study, and further provides information on the techniques used to reduce the opportunity of simulation errors. After some consideration, simulation has been selected as the method of study in this research. Generally and in addition to conducting measurements on a real practical system or test bed, there exist two techniques for system performance evaluation: analytical modeling and simulation [39]. One of the key considerations when adopting a given evaluation technique is the level of the desired accuracy. In general, analytical models have often low requirements in terms of computation costs, but they often rely on many assumptions and simplifications that restrict their applicability to a limited number of scenarios. On other hand, simulation models can easily incorporate details to the desired level of accuracy in order to mimic more closely the behavior of the real system. The consequence of this is that simulations often require a longer time to develop and run the code, compared to analytical modeling. However, as we have used the ProcSimity simulator that has already been developed and extensively validated [37, 39], we have easily incorporated our suggested algorithm into the simulator. This has helped to significantly reduce development time and debugging of the code. The cost along with the ease of being able to change configurations is the main motivation for developing simulations for expensive systems, such as multicomputers. The processor allocation algorithm designed and analyzed in this study is for mesh-connected multicomputers, which could consist of a large number of processors. Such a study could not be easily carried out on a practical system, as the experimental setup would require substantial and expensive resources.

ProcSimity has been widely used to evaluate the performance of processor allocation algorithms suggested for 2D mesh-connected multicomputers. Taking into account the modifications to the simulator, special care has been taken to ensure that the algorithm

implemented would function as designed and that the simulator would not exhibit unwanted side-effects. This has been achieved by carrying out the validation of the simulator for a number of cases and compared the performance results obtained for some-well known strategies against those obtained by other researchers using another simulator [19].

Chapter 3

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

3.1. Introduction

Most existing allocation strategies [4, 15, 18, 19, 23, 30, 44, 45, 55] proposed for mesh-connected multicomputers are based on contiguous allocation, where the processors assigned to a parallel job are physically contiguous and have the same topology as that of the interconnection network of the multicomputer. In contiguous allocation, jobs are allocated distinct contiguous processor sub-meshes for the duration of their execution. Contiguous allocation strategies often result in high processor fragmentation, leading to degradation of the system performance in terms of average turnaround time of jobs and mean system utilization [55]. The main goal of a any processor allocation strategy is to reduce the job turnaround time and to maximize the system utilization by reducing the problem of processor fragmentation. Several studies have attempted to reduce processor fragmentation [10, 17, 23, 29, 31, 46]. One of the proposed solutions is to adopt non-contiguous allocation [10, 25, 26, 29, 40]. In non-contiguous allocation, a job can be executed on multiple disjoint smaller sub-meshes rather than waiting until a single sub-mesh of the requested size becomes available. Although non-contiguous allocation increases message contention inside the network, dropping the contiguity condition can reduce processor fragmentation and increase system utilization [10, 29, 40].

Most existing studies have been performed in the context of contiguous allocation [15, 36, 55]. There has been relatively very little work on non-contiguous allocation. Although contiguous allocation eliminates contention among the messages of concurrently executing jobs, non-contiguous allocation can eliminate processor fragmentation that contiguous allocation suffers from. In addition, most existing

research on contiguous and non-contiguous allocation has been carried out in the context of the 2D mesh [10, 17, 18, 26, 29, 31, 36, 46, 55].

The existing non-contiguous allocation strategies suggested for the 2D mesh suffer from several problems that include internal fragmentation, external fragmentation, and message contention inside the network [10, 25, 26, 29]. Furthermore, allocation of processors to job requests is not based on free contiguous sub-meshes as in [10, 30]; but rather on artificial predefined geometric or arithmetic patterns. For example, in [10] ANCA subdivides job requests into two equal parts, and the subparts are successively subdivided in a similar fashion if allocation fails for any of them. In [29], MBS strategy bases partitioning on a base-4 representation of the number of processors requested, and partitioning in Paging [29] is based on the characteristics of the page, which is globally predefined independently from the request. Therefore, these strategies may fail to allocate an available large sub-mesh, which affect the system performance, such as the turnaround times of jobs [10, 29, 39]. In [39], GABL is based on available processors, regardless of their position in the mesh system which may allocate a job to sub-meshes that are far apart from each other in the mesh system and hence increases the communication overhead that affects the performance in terms of jobs turnaround time [39]. Figures 3.1 and 3.2 show an example of a 4×4 2D mesh. In figure 3.1, the job requests a 5×1 sub-mesh, and this job request is not allocated contiguously because there is no available sub-mesh of size 5×1 in mesh system. According to the GABL strategy, the job request is divided into two sub-requests, the first one is 4×1, which is allocated to available sub-mesh as shown in figure 3.2, and then the second sub-request 1×1 is allocated in another sub-mesh [39].

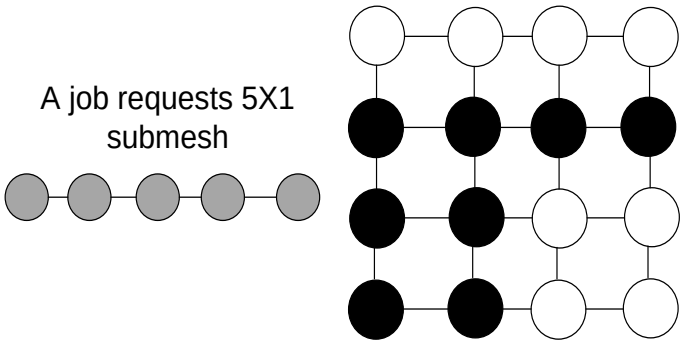

Figure 3.1: A job requests 5X1 2D sub-mesh in 4X4 mesh

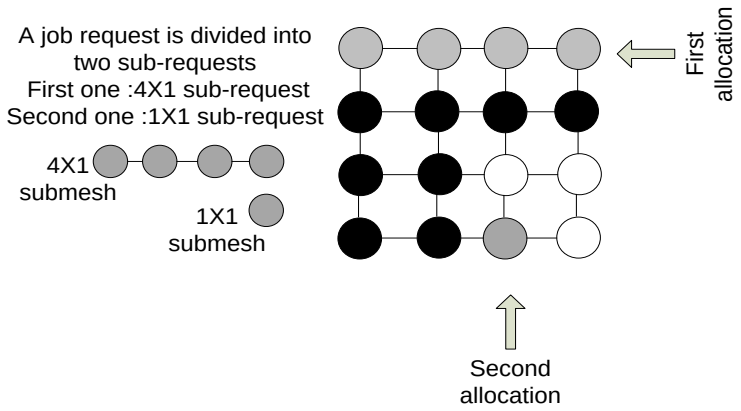

Figure 3.2: A job request 5X1 is divided into two sub-requests in GABL allocation algorithm

Motivated by the above observations, this chapter makes the following contributions. We describe a new non-contiguous allocation strategy, referred to here as A Compacting Non-Contiguous Processor Allocation Strategy for 2D Mesh-Connected Multicomputers (CNCPA for short), for the 2D mesh, and compare its performance properties using detailed simulations against those of the previous non-contiguous allocation strategies GABL, MBS and Paging (0) [39, 54]. These three strategies have been selected because they have been shown to perform well in [29, 39].

The remainder of the chapter is organized as follows. Section 3.2 describes our proposed non-contiguous allocation strategy. Finally, Section 3.3 concludes this chapter.

3.2. The Proposed A Compacting Non-Contiguous Processor Allocation Strategy for 2D Mesh-Connected Multicomputers (CNCPA)

The target system is a 2D mesh-connected multicomputer, referred to as M(W, L) , where W is the width of the mesh, and L is its length.

In CNCPA strategy, a single job is compacted into more than one free locations within the allocated processors, where the remaining available processors (free processors) form a large sub-mesh in the mesh system. As a result, the strategy is expected to improve the system performance in terms of average turnaround time and mean system utilization.

In CNCPA strategy, when a parallel job is selected for allocation, a sub-mesh suitable for the entire job is searched. If such a fit sub-mesh is found it is allocated to the job. A fit sub-mesh is defined as the sub-mesh where its dimensions are greater than or equal to the job dimensions. For example, if dimensions of the job are $a \times b$ and sub-mesh dimensions are $x \times y$ the relation between the job and sub-mesh should be [$x \geq a$ and $y \geq b$]. If such a sub-mesh is found it will be allocated to the job. Otherwise, the job is divided into two sub-requests where the size of the first sub-request fits the first available sub-mesh. After that, the strategy searches for the second available sub-mesh for the remaining sub-requests where the second available sub-mesh should be close to the first one. If the second available sub-mesh is not enough for the size of the second sub-request then the remaining job is divided into another two sub-requests where the size of the first sub-request fits the available sub-mesh. This step is repeated until the original job request is allocated.

In order to describe the proposed strategy, we give an example in which we assume a job requests a sub-mesh of size 3×2. First, check the number of available processors. If the available ones are enough then the contiguous allocation algorithm First-Fit [55] is used to search for a suitable sub-mesh in order to allocate the first available sub-mesh, as shown in figures 3.3 and 3.4.

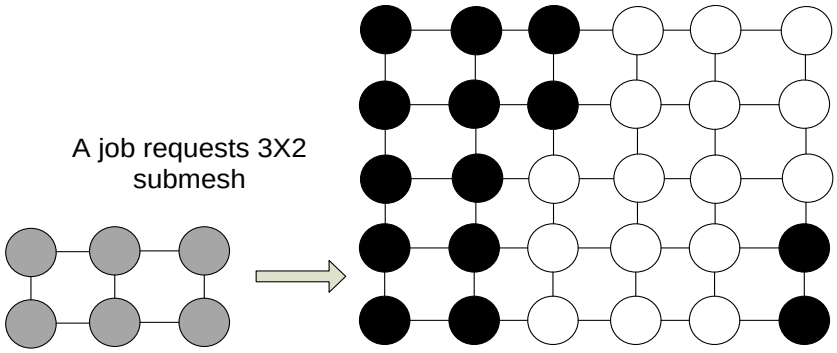

Figure 3.3: A job requests a 3X2 2D sub-mesh in 6X5 mesh

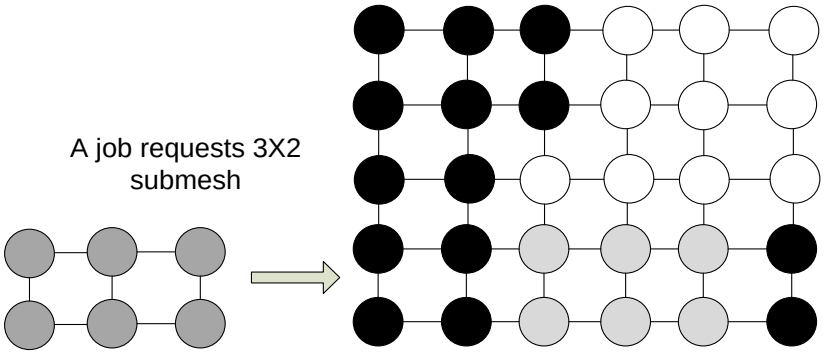

Figure 3.4: The job is allocated the first available 3X2 2D sub-mesh

Assuming the allocation state in figure 3.5 and the job requests a sub-mesh of size 3×2. First, contiguous allocation fails because there is no available contiguous sub-mesh of size 3×2 in the mesh system. In this case, our proposed algorithm works as the following: the job request is divided into two sub-requests (1×2 and 2×2) where the first sub-request is allocated to the first available sub-mesh as shown in figure 3.6, and the second sub-request (2×2) is allocated if possible, but because the second available sub-mesh that is close to the first one is not enough for the second sub-request (2×2), the process is repeated again for the remaining request (2×2) as shown in figures 3.6 and 3.7.

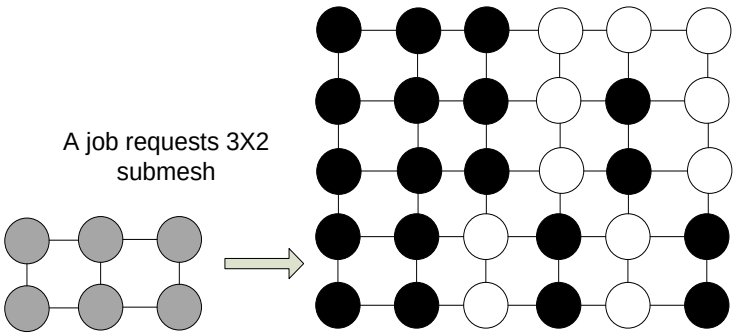

Figure 3.5: A job requests 3X2 2D sub-mesh in 6X5 mesh but not found it As soon as possible

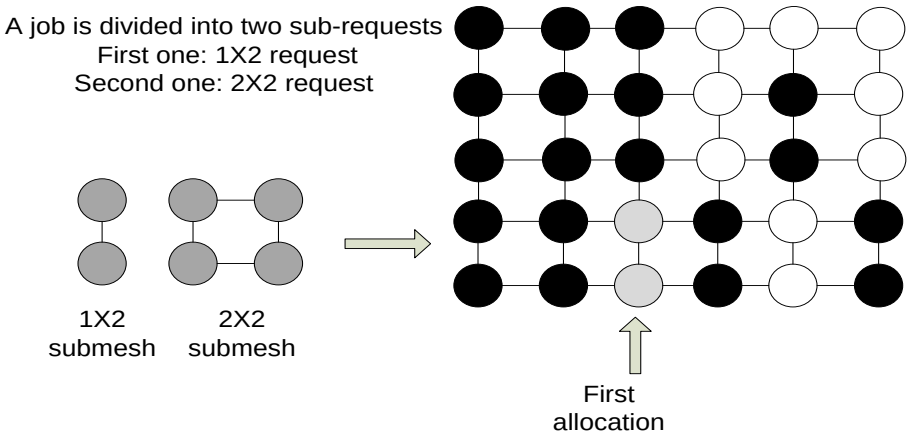

Figure 3.6: Divide a job request (3X2) in 6X5 mesh, allocate the first sub-mesh

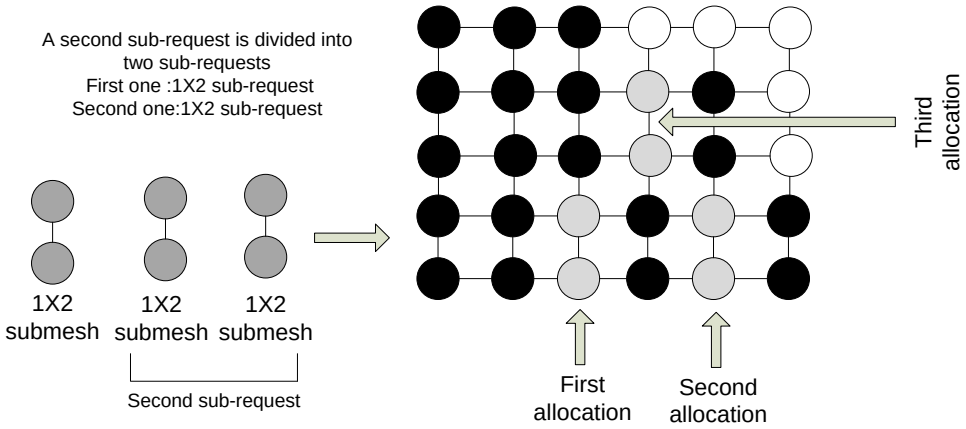

Figure 3.7: Divide a job request (2X2) in 6X5 mesh, allocate the second sub-mesh

Allocation in CNCPA is implemented by the algorithm outlined in Figure 3.8, while the de-allocation algorithm is outlined in Figure 3.9. Note that allocation always succeeds if the number of free processors is $\geq a \times b$

Procedure CNCPA_Allocate (a **,** b***):***
Begin *{*
no_free_processor = 0
Job_Size = a x b

*Step*1. *If* (*no_free_processor* < *Job_Size*)
return failure.

Step 2. If (*there is a free sub-mesh S*(*w, l*) *suitable for S*(*a, b*))
{
allocate it contiguously.
update the no_free_processor
return success.
}

```
        Else go to step 3
        {
Step 3. find any free non-contiguous submesh S(w,l)
        {
            count_of_requested_ processor =0

            for each element (processor) in sub-mesh S(w,l)
              while(count_of_requested_ processor != Job_Size )
              {
                  count_of_requested_ processor++
                add element to the  list

          }
        }
Step 4. allocate the elements in the list to the job request
            update the no_free_processor
            return success.
    }

} End.
```

Figure 3.8: The Compaction Non-Contiguous Processor allocation algorithm

```
Procedure CNCPA_Deallocate ():
Begin {
        job_id = id of the departing job;
        For all elements in the mesh
            if (element's id = job_id)
                element states = Idle
} End.
```

Figure 3.9: The Compaction Non-Contiguous Processor de-allocation algorithm

Chapter 4

Performance Evaluation

In this chapter, the results of the simulation experiments that have been carried out to evaluate the performance of the proposed algorithm are presented and compared against those of Paging(0), MBS, and GABL.

4.1. Allocation and De-allocation Time in CNCPA

When a sub-mesh is allocated, CNCPA takes $O(m^2)$ time, where m is the number of allocated sub-meshes. The worst case for CNCPA occurs when the free processors distributed in distant form (i.e., each one of the allocated sub-meshes for any job request equal one processor), and this may increases the distance between communicating processors which increases the communication overhead and thus degrades system performance. In such a case, the worst-case time for CNCPA takes $O(n^2)$, where n is the number of processors in the mesh system. When a job departs the system, the de-allocation algorithm takes $O(m)$ time. The proposed algorithm maintains a linked list, therefore, its space requirement is in $O(m)$.

4.2. Simulation Results

In this section, we will show the results for CNCPA and in addition to the results for the GABL, Paging(0) and MBS allocation algorithms. We have implemented the proposed allocation and de-allocation algorithms, in the C language, and integrated the software into the ProcSimity simulation tool that is widely used for processor allocation and job scheduling in parallel systems [37, 39].

The target mesh modeled in the simulation experiments is square with side lengths L. Jobs are assumed to have exponential inter-arrival times. They are served on a First-Come-First-Served (FCFS) basis. We have limited ourselves to FCFS scheduling because our main purpose here is to compare the allocation strategies. 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 node, packet sizes, the number of messages sent, message contention and distances messages traverse. Two distributions are used to generate the lengths and widths of job requests. The first is the uniform distribution over [1, L], where the width and length of a request are generated independently. The second is the uniform-decreasing distribution, that is based on four probabilities p1 , p2 , p3 , and p4 and three integers l1, l2, and l3, where the probabilities that the width/height of a request falls in the ranges [1, l1], [l1 + 1, l2], [l2 + 1, l3], and [l3 + 1, L] are p1, p2, p3, and p4, respectively. The side lengths within a range are equally likely to occur. The uniform-decreasing distribution represents a case where most jobs are small relative to the size of the system. These distributions have often been used in the literature [29, 39, 46, 55].

The interconnection network uses wormhole routing. Flits are assumed to take one time unit to move between two adjacent nodes, and t_s time units to be routed through a node. Packet sizes are represented by P_{len}. Processors allocated to a parallel job typically exchange messages with each other using three communication patterns. The first communication pattern is one-to-all, where a randomly selected processor sends a packet to all other processors allocated to the same job. The second communication pattern is random, where a randomly selected processor sends packets to randomly selected destinations within the set of processors allocated the same job. The third communication pattern is the Near Neighbour communication pattern, where the processors allocated to a job are mapped to a virtual two-dimensional array of a size that is equal to the job's allocation request. Each of these processors communicates with its virtual neighbours. In the simulation experiments, each communication pattern is completed once, and a job remains in the system until it completes one iteration of the communication pattern being simulated. In all cases, processors allocated to a job are mapped to a linear array of processors using row-major indexing. The simulator selects the sources and destinations from this array, and the mapping is used for determining the x and y coordinates of the sources and destinations of communication operations. Unless specified otherwise, the performance figures shown below are for a 16 × 16 mesh, t_s = 3 time units, P_{len}. = 8 flits. Also, the results for the uniform-decreasing side length distribution are based on $p_1 = 0.4$, $p_2 = 0.2$, $p_3 = 0.2$, $p_4 = 0.2$,

$l_1 = L/8$, $l_2 = L/4$, and $l_3 = L/2$. Simulation parameters are illustrated in Table 4.1. It is worth noting that most of the values of these parameters have been adopted in the literature [26, 29, 39, 46, 55] and have been recommended in [37].

Table 4.1: The System Parameters used in the Simulation Experiments

Simulator Parameter	Values
Dimensions of the Mesh Architecture	16 × 16
Packet Length	8 flits
Flow Control Mechanism	Wormhole Routing
Buffer Size	1 flit
Routing Delay	3 time units
Router Type	Mesh XY Routing
Allocation Strategy	CNCPA, GABL, MBS and Paging(0)
Scheduling Strategy	FCFS
Job Size Distribution	Uniform, Uniform-decreasing
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.
Mean Time between Sends	0.0
Communication Patterns	One-to-All, Random, Near Neighbour
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

Each simulation run consists of 1000 completed jobs. Simulation results are averaged over enough independent runs so that the confidence level is 95% and the relative errors do not exceed 5% [2]. The method used to calculate confidence intervals is called batch means analysis [1, 37, 39]. In batch means method, a long run is divided into a set of fixed size batches, computing a separate sample mean for each batch, and using these batches means to compute the grand mean and the confidence interval. In our simulation experiments, the grand means are obtained along with several values, including confidence interval and relative errors as shown in Table 4.2 which shows the grand means, confidence intervals, and relative errors that outline the results depicted in Figure 4.1 for the load 0.001 jobs/time unit.

Table 4.2: The mean (i.e., mean turnaround time of job), 95% confidence interval, and relative error for the results shown in Figure 4.1 for the load 0.001 jobs/time unit

Algorithm	CNCPA	GABL	MBS	Paging(0)
95% Confidence Interval	[242054.873612155-264866.327669845]	[244554.75412416-264934.31696784]	[248133.947848824-270974.729659176]	[244274.925494352-266760.483489648]
Mean (time unit)	253460.600641	254744.535546	259554.338754	255517.704492
Relative Error	0.045	0.04	0.044	0.044

The main performance parameters used are the *average turnaround time* of jobs and *mean system utilization*. The *turnaround time* of a job is the time that the job spends in the mesh system from arrival to departure. The *system utilization* is the percentage of processors that are utilized over time. The important independent variable in the simulation is the system load. It is defined as the inverse of the mean inte-rarrival time of jobs. Its range of values from low to heavy loads has been determined through experimentation with the simulator allowing each allocation strategy to reach its upper limits of utilization. In the figures that are presented below, the x-axis represents the system load while the y-axis represents results of the performance metric of interest [39].

4.2.1. Turnaround Time:

In Figures 4.1 and 4.2, the average turnaround times of jobs are plotted against the system load for the one-to-all communication pattern. The results reveal that CNCPA performs better than all other non-contiguous allocation strategies for both uniform and uniform-decreasing job size distributions considered in this research. In Figure 4.1, for example, the differences in performance in favor of CNCPA against GABL, MBS, and Paging(0) are respectively as large as 0.5%, 2% and 0.8% when the load is 0.001 jobs/time unit.

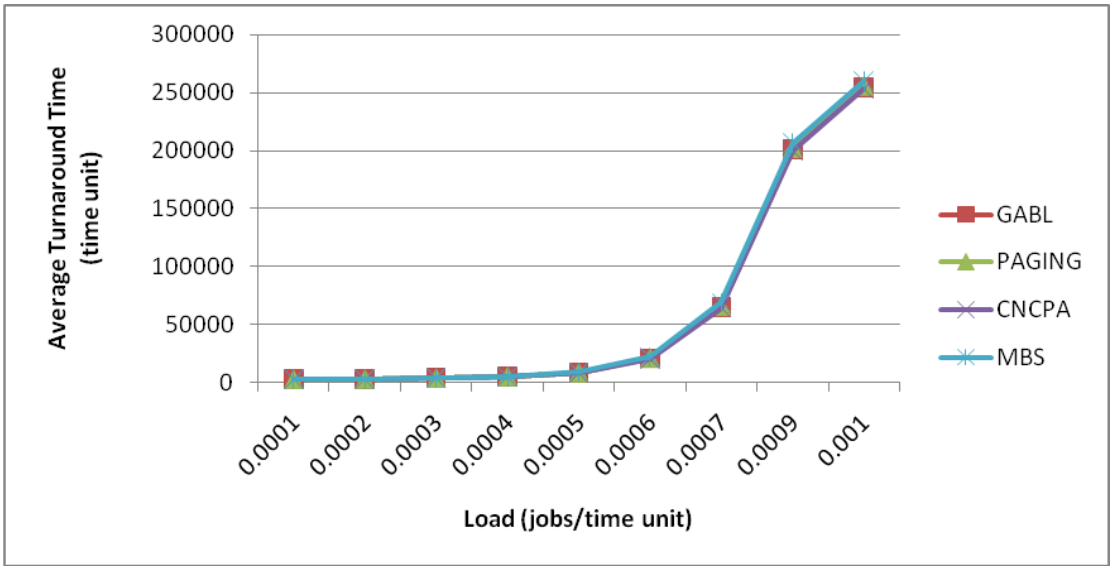

Figure 4.1. Average turnaround time vs. system load for the one-to-all communication pattern and uniform side lengths distribution in a 16 × 16 mesh.

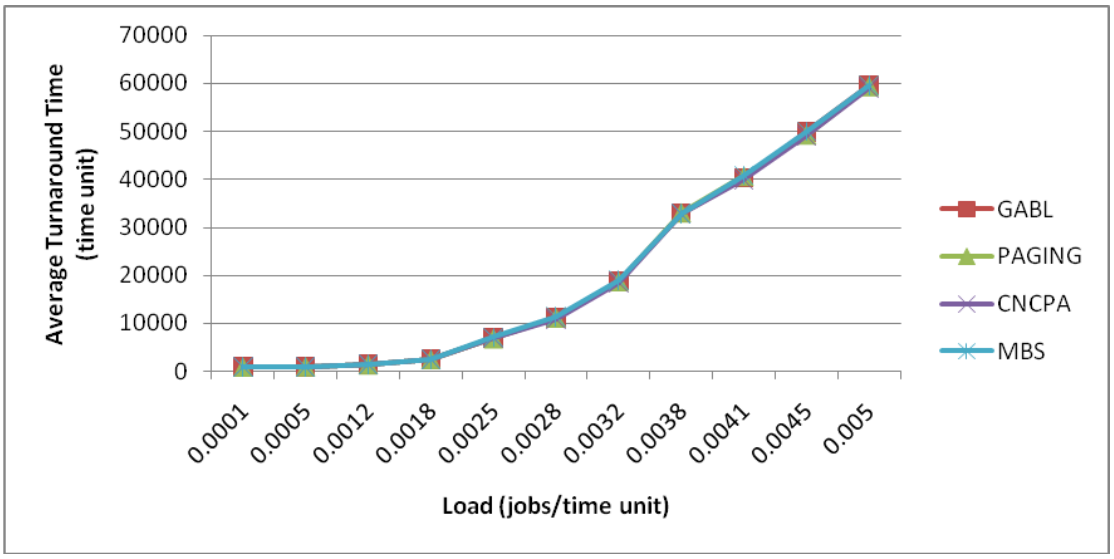

Figure 4.2. Average turnaround time vs. system load for the one-to-all communication pattern and uniform-Decreasing.side lengths distribution in a 16 × 16 mesh.

In Figures 4.3 and 4.4, the average turnaround times of jobs are plotted against the system load for the near neighbor communication pattern. The results reveal that CNCPA performs better than some non-contiguous allocation strategies for both uniform and uniform-decreasing job size distributions considered in this research. In Figure 4.3, for example, the differences in performance in favor of CNCPA against MBS and Paging(0) are respectively as large as 14% and 13% when the load is 0.0089 jobs/time unit.

For GABL, the performance is better than that of other allocation strategies as shown in figures 4.3 and 4.4. This is because the distances between communicating nodes are relatively low when the near neighbor communication pattern is used. Distances between communicating nodes have significant impact on message latency when messages are short. This is the case in the simulation scenarios, where the length of packets is 8 flits. Also, when the distances traversed by messages are short they are less likely to collide with other messages. This in turn decreases the communication overhead. As a consequence, the turnaround time is lower.

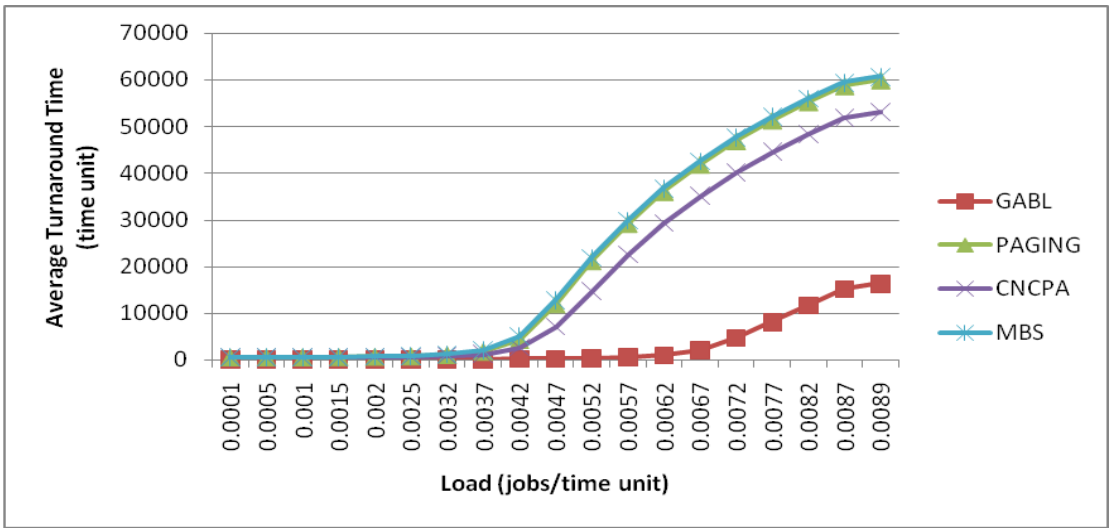

Figure 4.3. Average turnaround time vs. system load for the near neighbor communication pattern and uniform.side lengths distribution in a 16 × 16 mesh

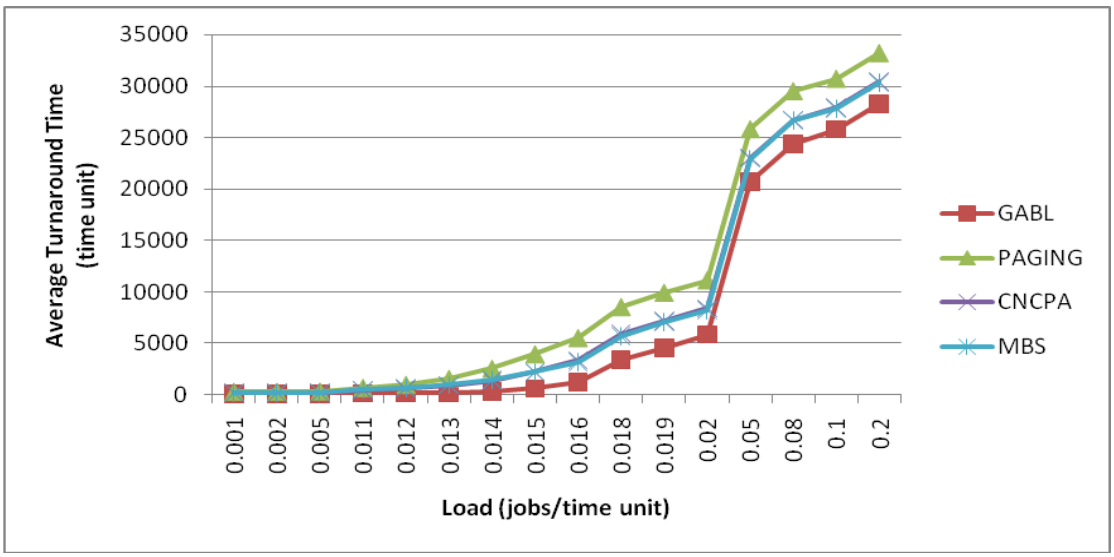

Figure 4.4. Average turnaround time vs. system load for the near neighbor communication pattern and uniform-Decreasing.side lengths distribution in a 16 × 16 mesh.

In Figures 4.5 and 4.6, the average turnaround times of jobs are plotted against the system load for the random communication pattern. Again, The results reveal that CNCPA performs better than the non-contiguous allocation strategies for uniform-decreasing job size distributions considered in this research, but for uniform job size distributions the results reveal that CNCPA performs better than some non-contiguous allocation strategies, MBS and Paging(0), but the performance for GABL is better than that of other allocation strategies including the ones proposed in this research. In Figure 4.6, for example, the differences in performance in favor of CNCPA against GABL, MBS and Paging(0) are respectively as large as 15%, 4% and 14% when the load is 0.18 jobs/time unit.

CNCPA is overall better than some previous non-contiguous allocation strategies at alleviating message contention, but contention in the random communication pattern is lower than that in the one-to-all and near neighbor communication patterns. This is because destinations are chosen randomly and paths are less likely to overlap. Contention that results from the random communication pattern is not sufficient for differentiating among the non-contiguous allocation strategies. For example, the performance of Paging(0) is relatively poor because the distances between nodes are relatively high. Distances between communicating nodes have significant impact on message latency, independently of contention, when messages are short. This is the case in the simulation scenarios, where the length of packets is 8 flits. Also, when messages traverse longer distances they are more likely to collide with other messages. The increase in contention associated with non-contiguous allocation strategies is outweighed by the superior ability of the non-contiguous strategies at allocating free processors.

.

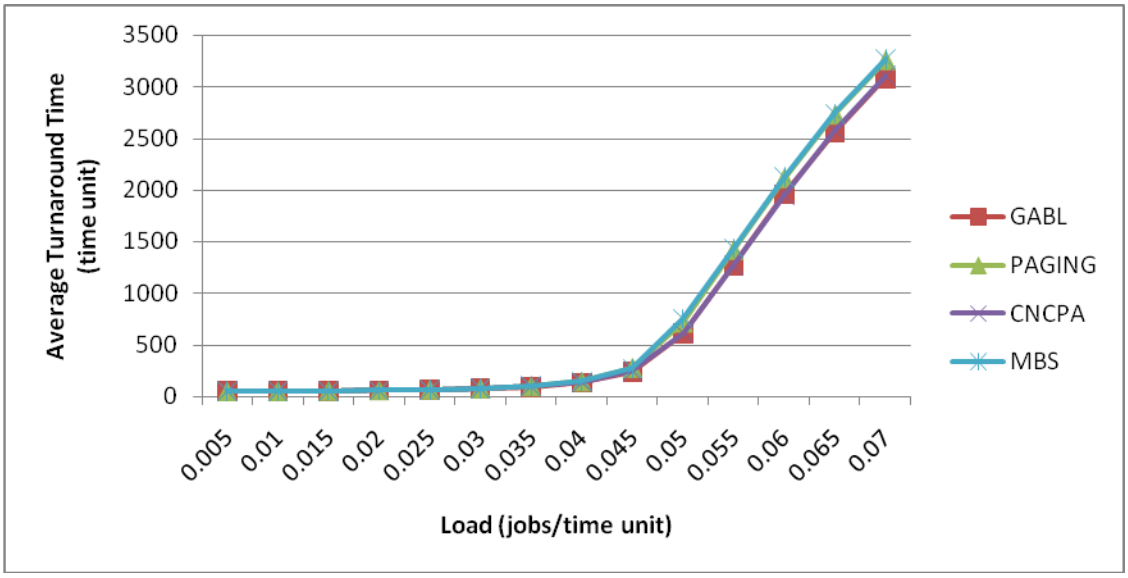

Figure 4.5. Average turnaround time vs. system load for the random communication pattern and uniform.side lengths distribution in a 16 × 16 mesh.

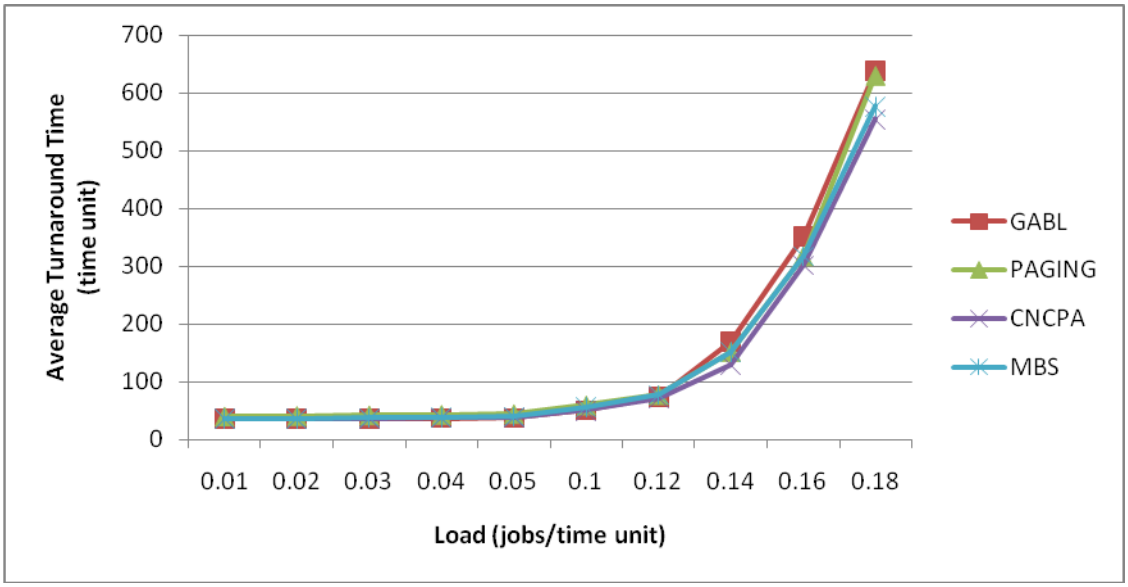

Figure 4.6. Average turnaround time vs. system load for the random communication pattern and uniform-Decreasing.side lengths distribution in a 16 × 16 mesh.

4.2.2. Utilization:

Figures 4.7 and 4.8 depict the mean system utilization of the allocation strategies (CNCPA, GABL, MBS and Paging(0)) for the one-to-all communication pattern. The simulation results in these two figures are presented for a heavy system load. The load is such that the waiting queue is filled very early, allowing each allocation strategy to reach its upper limits of utilization. For *uniform* job size distribution, all non-

contiguous allocation strategies achieve a mean system utilization of 70–77%, but in Figure 4.8, for *uniform-decreasing* job size distribution, all non-contiguous allocation strategies achieve a mean system utilization of 76–82%. This is because the uniform-decreasing job size distribution represents the case where most jobs are small relative to the size of the mesh system and hence the allocation is more likely to succeed. This in turn increases the system utilization. The utilization of the four non-contiguous allocation strategies is approximately the same for both job size distributions. This is because the non-contiguous allocation strategies, considered in this research, 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 the allocation request.

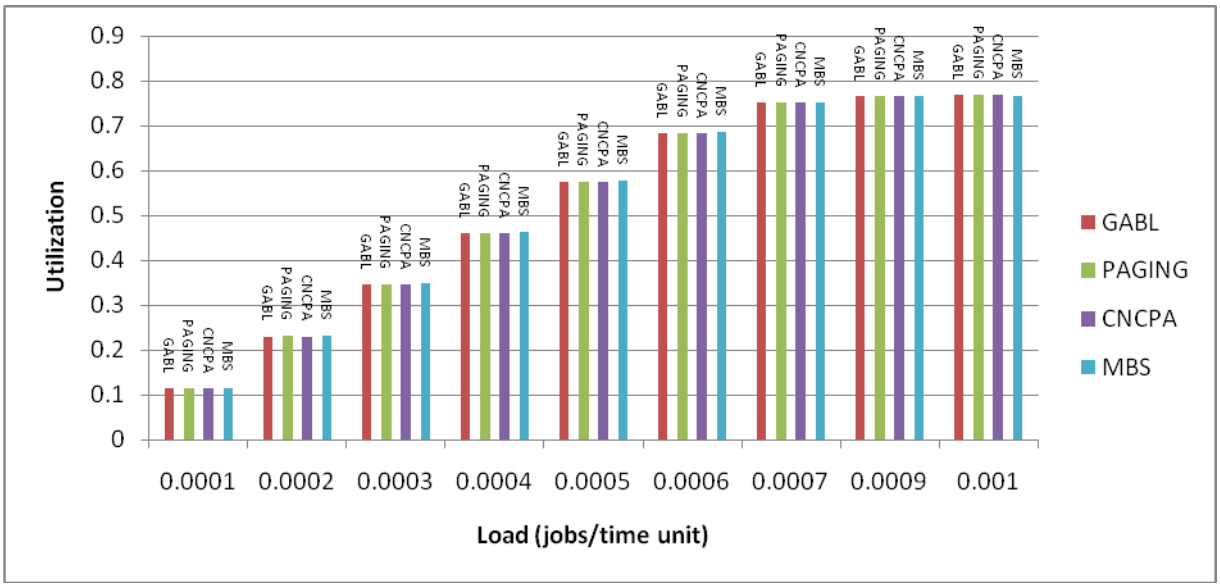

Figure 4.7. System utilization of the non-contiguous allocation strategies (CNCPA, GABL, MBS and Paging(0)), for the one-to-all communication pattern tested, and uniform side lengths distribution in a 16 x 16
mesh

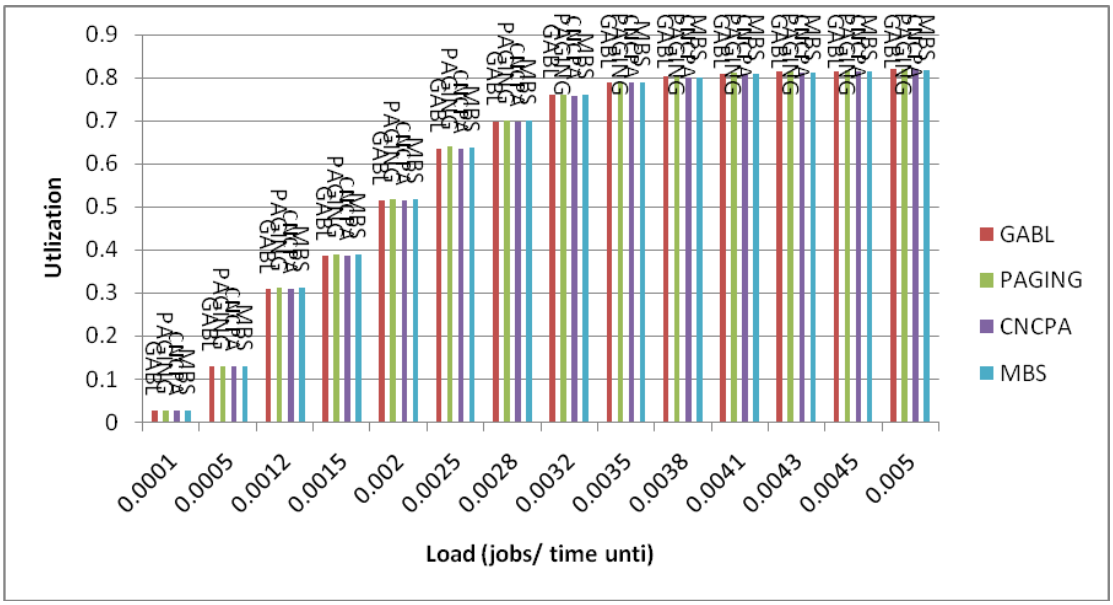

Figure 4.8. System utilization of the non-contiguous allocation strategies (CNCPA, GABL, MBS and Paging(0)), for the one-to-all communication pattern tested, and uniform-Decreasing side lengths distribution in a 16 x 16 mesh.

Figures 4.9 and 4.10 depict the mean system utilization of the allocation strategies (CNCPA, GABL, MBS and Paging(0)) for the near neighbor pattern. As previously reported in Figures 4.7 and 4.8, the simulation results in these two figures are presented for a heavy system load. The load is such that the waiting queue is filled very early, allowing each allocation strategy to reach its upper limits of utilization. For uniform job size distribution, all non-contiguous allocation strategies achieve a mean system utilization of 70–78%, but in Figure 4.10, for uniform-decreasing job size distribution, all non-contiguous allocation strategies achieve a mean system utilization of 73–86%.

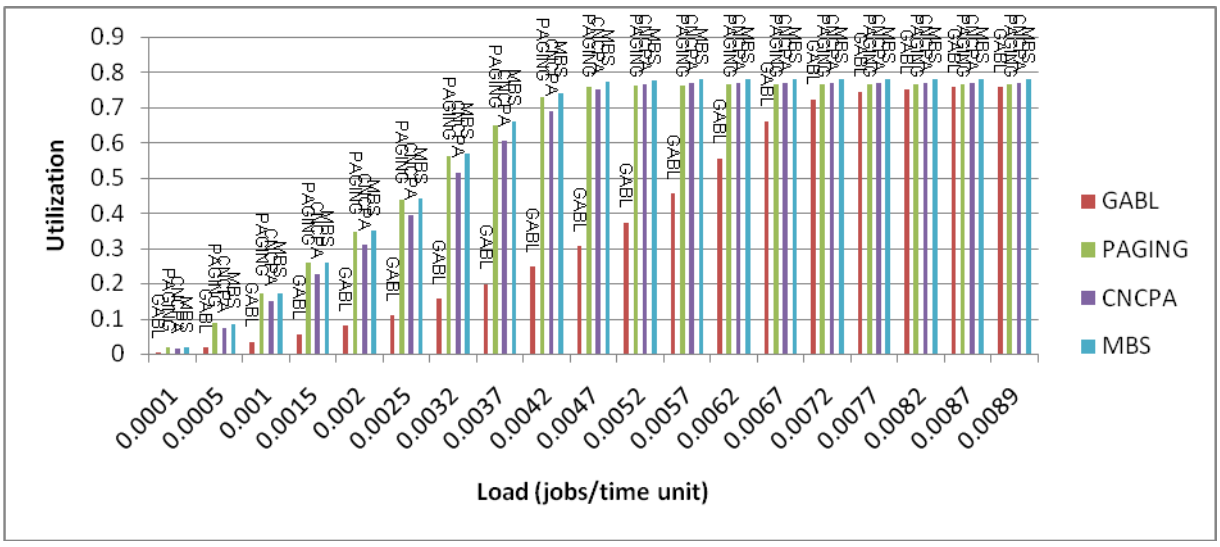

Figure 4.9. System utilization of the non-contiguous allocation strategies (CNCPA, GABL,

MBS and Paging(0)), for the near neighbor communication pattern tested, and uniform side lengths distribution in a 16 x 16 mesh.

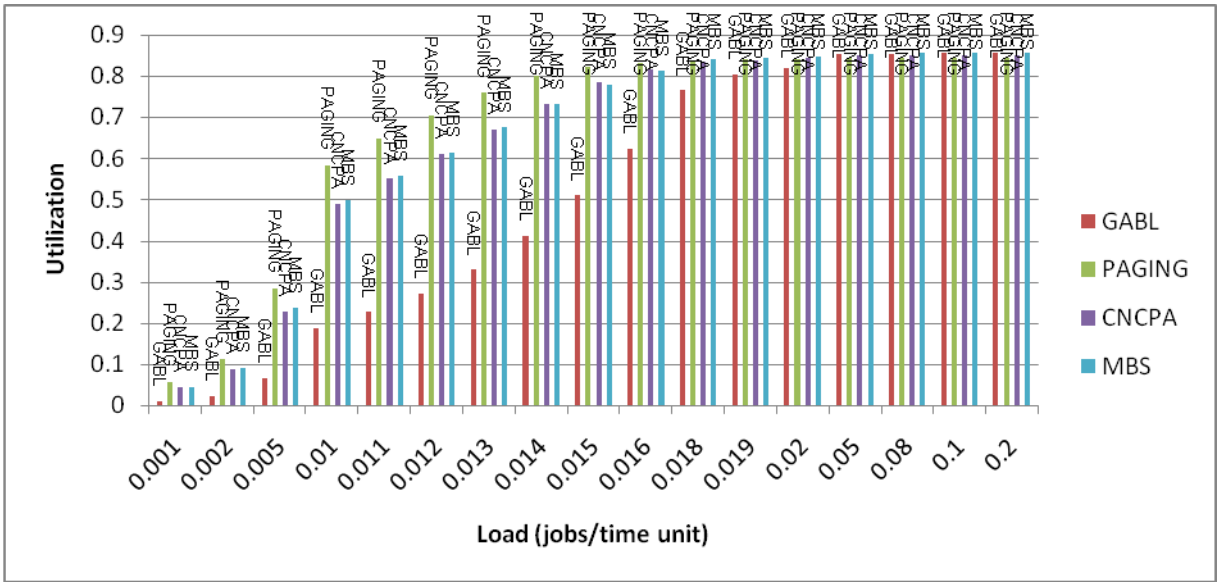

Figure 4.10. System utilization of the non-contiguous allocation strategies (CNCPA, GABL, MBS and Paging(0)), for the near neighbor communication pattern tested, and uniform-Decreasing side lengths distribution in a 16 x 16 mesh.

Figures 4.11 and 4.12 depict the mean system utilization of the allocation strategies (CNCPA, GABL, MBS and Paging(0)) for the random pattern. The simulation results in these two figures are presented for a heavy system load. For *uniform* job size distribution, all non-contiguous allocation strategies achieve a mean system utilization of 70–78%, but in Figure 4.12, for *uniform-decreasing* job size distribution, all non-contiguous allocation strategies achieve a mean system utilization of 73–86%.

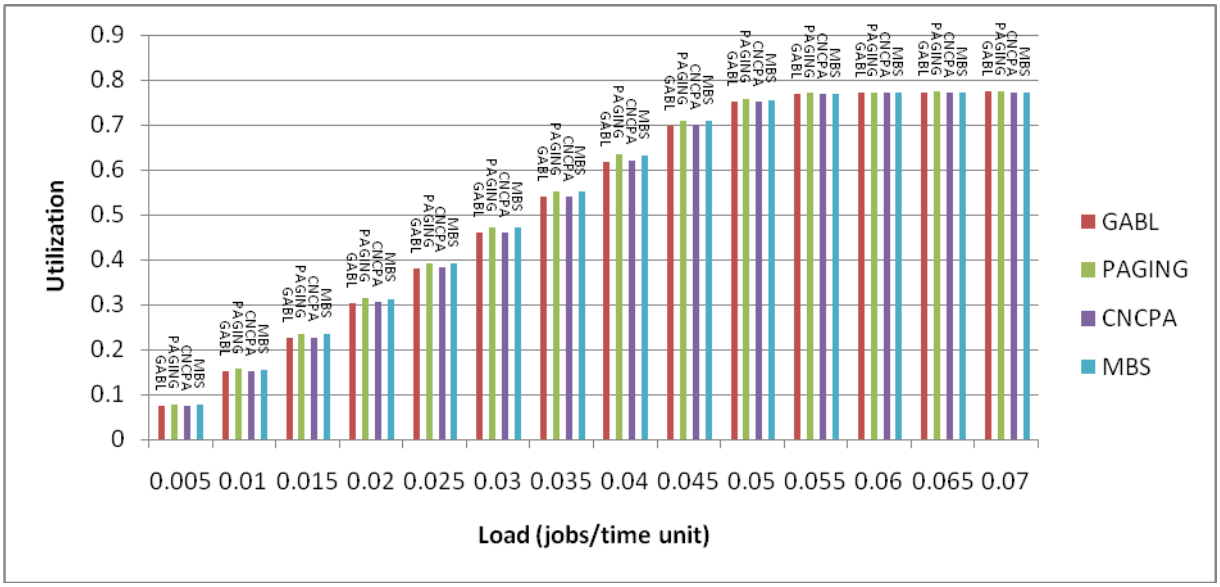

Figure 4.11. System utilization of the non-contiguous allocation strategies (CNCPA, GABL, MBS and Paging(0)), for the random communication pattern tested, and uniform side lengths

distribution in a 16 x 16 mesh.

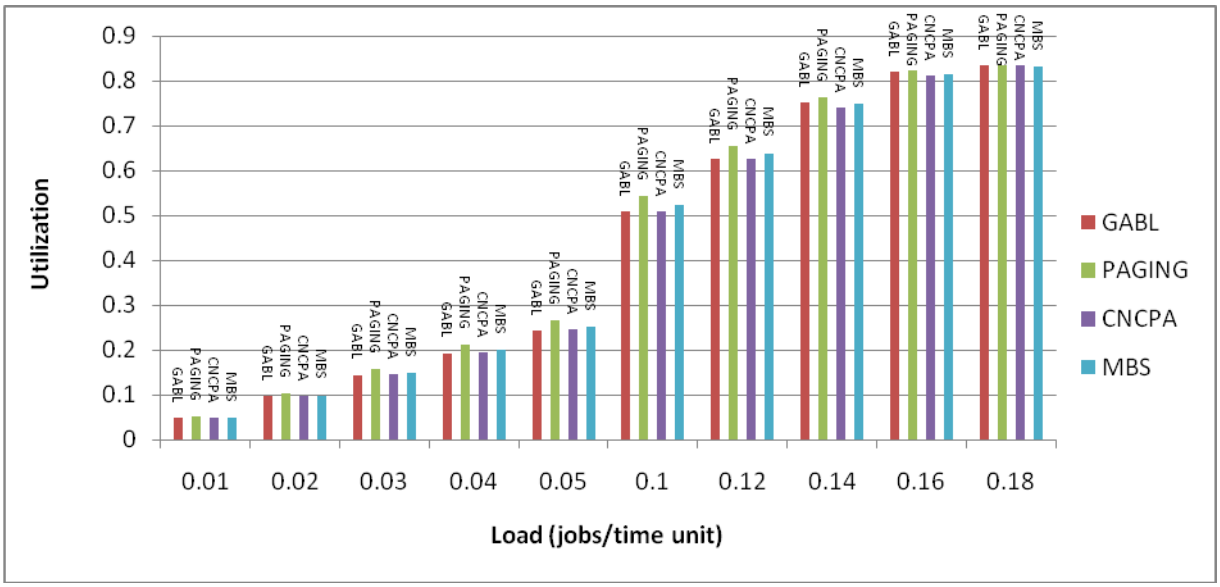

Figure 4.12. System utilization of the non-contiguous allocation strategies (CNCPA, GABL, MBS and Paging(0)), for the random communication pattern tested, and uniform-Decreasing side lengths distribution in a 16 x 16 mesh.

4.3. Conclusions

This chapter has investigated the performance merits of non-contiguous allocation in the 2D mesh network. To this end, we have suggested a new non-contiguous allocation strategy, referred to as A Compacting Non-Contiguous Processor Allocation Strategy, referred to as CNCPA, which differs from the earlier non-contiguous allocation strategies in the method used for decomposing allocation requests. The proposed strategy compacts a single job into more than one free location within the allocated processors. The major goal of this process is to maintain a high degree of contiguity among sub-meshes allocated to a job. This decreases the distance traversed by messages, and which in turn decreases the communication overhead and hence improves the system performance in terms of average turnaround time and mean system utilization.

The performance of CNCPA was compared against that of existing non-contiguous allocation strategies using the FCFS scheduling strategy and the one-to-all, random and near neighbor communication patterns. Simulation results have shown that CNCPA can improve performance for the one-to-all communication pattern despite the additional message contention inside the network that results from the interference

among the messages of different jobs as compared to the well-known non-contiguous allocation strategies such as GABL, MBS and Paging(0).

For near neighbor communication pattern, the results reveal that CNCPA is overall better than that of the previous non-contiguous allocation strategies, MBS and Paging(0), but the performance for GABL is better than that of other allocation strategies including the ones proposed in this research, CNCPA. This is because the distances between communicating nodes are relatively low when the near neighbor communication pattern is used. Distances between communicating nodes have significant impact on message latency when messages are short. This is the case in the simulation scenarios, where the length of packets is 8 flits. Also, when the distances traversed by messages are short they are less likely to collide with other messages. This in turn decreases the communication overhead. As a consequence, the turnaround time is lower.

For random communication pattern, the results reveal that CNCPA performs better than the previous non-contiguous allocation strategies considered in this research for uniform-decreasing job size distributions, but for uniform job size distributions, the results reveal that CNCPA performs better than some non-contiguous allocation strategies, MBS and Paging(0), but the performance for GABL is better than that of the other allocation strategies including the ones proposed in this research.

Chapter 5

Conclusions and Future Directions

In recent years, parallel computers have become very popular for solving large-scale computationally intensive problems [20, 32]. Parallel computers are often considered to be one of the most feasible ways of achieving the enormous computational power required by many real-life parallel applications found in science, engineering, and a number of other fields [26, 39]. Distributed-memory multicomputers are an important class of parallel computers for building large-scale parallel systems [52, 39]. Among the various distributed-memory multicomputers those based on the mesh network have received much attention from the research community due to the simplicity, structural regularity, partition-ability, and ease of implementation of this network topology [10, 13, 17, 18, 29, 31, 46, 55]. Mesh multicomputers are suitable for different applications such as matrix computations, image processing and problems whose task graphs can be embedded naturally into the mesh. It has been used as the underlying network in a number of commercial and experimental multicomputers, including the Tera Computer, Cray T3D, MIT J-Machine and the IBM BlueGene/L [5, 38, 39].

Processor allocation in distributed-memory multicomputers, especially those based on the mesh network, became the subject of much research in recent years [17, 18, 23, 30, 42]. Several commercial and experimental parallel machines have used space sharing for processor allocation [9, 22, 33, 54]. In space sharing, the set of processors in a system, e.g., mesh-connected multicomputer, is partitioned into a set of sub-meshes each of which is exclusively allocated to a single job [21, 39]. Processor allocation strategies are divided into two categories: *contiguous* and *non-contiguous*. In contiguous allocation, jobs are allocated distinct contiguous processor sub-meshes for the duration of their execution. Contiguous allocation has the problem of processor fragmentation [28, 37, 39, 40, 42, 47, 48].

Processors fragmentation can be classified into *internal* and *external* fragmentation. Internal fragmentation occurs when more processors are allocated to a job more than it requires [10, 29, 50]. When a job is assigned more processors than it requires, the extra allocated processors are not used for actual computation, instead they are wasted. External fragmentation occurs when a sufficient number of processors are available to satisfy a request, but they cannot be allocated contiguously because they are not contiguous for example [29].

A number of researchers have adopted non-contiguous allocation to solve the problem of processor fragmentation [10, 23, 29], where a job can be executed on multiple disjoint sub-meshes rather than waiting until a single sub-mesh of requested size and shape is available. In past years, non-contiguous allocation has not attracted considerable research attention because the communication latency was sensitive to the distance in the network used in the first generation of multicomputers [39]. However, the advances in routing technique such as wormhole routing [1, 4, 52] have made non-contiguous allocation plausible in networks characterized by long diameters such as the mesh. Wormhole routing has been widely adopted in the second generation of multicomputers [12, 39]. An advantage of wormhole routing over earlier routing schemes, mainly store-and-forward, is that message latency has become less dependent on message distance [1, 24].

The procedure used for partitioning allocation requests in non-contiguous allocation has a considerable impact on the performance of non-contiguous allocation strategies [10, 29, 40, 41]. Therefore, the process of partitioning in non-contiguous allocation should aim to maintain a high degree of contiguity between the sub-meshes allocated to a given parallel job. This is so that the communication overhead is kept to a minimum without affecting the overall system performance [40, 41].

5.1. Summary of the Results

The main objective of this research has been the development of a new non-contiguous allocation strategy for mesh-connected multicomputers that overcome the limitations of the existing strategies suggested for the 2D mesh networks. Following summarizes the main contributions to this research study.

- There have been many non-contiguous allocation strategies that have been suggested for the 2D mesh network. However most of these suffer from several problems that include internal fragmentation, external fragmentation, as well as message contention inside the network [10, 23, 29, 39]. Moreover, the allocation of processors to job requests is not based on free contiguous sub-meshes in the existing strategies [10, 29, 39]. Instead, it is often based on artificial predefined geometric or arithmetic patterns. In [39], GABL is based on available processors, regardless of their position in the mesh system which may allocate a job to sub-meshes that are far apart from each other in the mesh system which increases the communication overhead and thus affects the performance in terms of turnaround time [39]. Therefore, these strategies may fail to allocate an available large sub-mesh, which in turn can cause degradation in system performance, such as the turnaround times of jobs [10, 29, 39, 40]. Motivated by these observations, this research has suggested a new non-contiguous allocation algorithm, referred to as A Compacting Non-Contiguous Processor Allocation Strategy (CNCPA for short), for mesh-connected multicomputers. The CNCPA strategy combines the main desirable features of both the contiguous and non-contiguous allocation strategies. In this research study the new proposed non-contiguous allocation strategy has been adapted to the 2D mesh in order to compare its performance against that of the existing non-contiguous allocation strategies suggested for the same network.

- The proposed CNCPA strategy relies on a new approach that maintains a higher degree of contiguity among the sub-meshes than that of the previous non-contiguous allocation strategies. This decreases the distance traversed by

messages, which in turn decreases communication overhead and as a result decreases jobs turnaround time. Extensive simulation experiments under a variety of system operating conditions have been carried out to compare the performance of the proposed CNCPA strategy against that of the existing non-contiguous allocation strategies. The results have shown that in most cases the new strategy has better performance in terms of the turnaround time than the previous non-contiguous allocation strategies of [29]. Moreover, when message contention increases inside the network due to using the one-to-all communication pattern, for example, CNCPA exhibits performance over the previous non-contiguous allocation strategies. For instance, under high loads, the differences in performance in favor of CNCPA against GABL [41], MBS, and Paging(0) [29] are respectively as large as 0.5%, 2% and 0.8% for the one-to-all communication pattern and uniform side length distribution considered in this research. Furthermore, the proposed strategy exhibits high system utilization as it manages to eliminate both internal and external fragmentation. For instance, under high loads, CNCPA achieves a mean system utilization of 70% to 77% under the uniform side lengths distributions, but for uniform-decreasing job size distribution, all non-contiguous allocation strategies achieve a mean system utilization of 76–82%.

5.2. Directions for the Future Work

There are many interesting questions and open problems that require further investigation. From my point of view, the most important one is described below.

- The results in [29, 39] have shown that non-contiguous allocation strategies dramatically outperform contiguous allocation strategies for 2D mesh network. A Compacting Non-Contiguous Processor Allocation Strategy (CNCPA), proposed in Chapter 3 can be applied to 3D mesh network. So, it would be interesting to adapt the proposed non-contiguous allocation algorithm (CNCPA) to 3D mesh network and investigate its performance against that of the contiguous allocation in 3D mesh network.

References

[1] A. Al-Dubai, M. Ould-Khaoua, K. El-Zayyat, I. Ababneh, and S. Al-Dobai (2004): *Towards scalable collective communication for multicomputer interconnection networks*, Journal of Information Sciences, vol. 163, no. 4, pp. 293-306.

[2] A. Law and W. Kelton (2000), *Simulation Modelling and Analysis*, Third Edition, McGraw-Hill, Inc., New York.

[3] A. Louri and H. Sung, (1994): *An Optical Multi-Mesh Hypercube: A Scalable Optical Interconnection Network for Massively Parallel Computing*, IEEE/OSA Journal of Light wave Technology, vol. 12, no. 4, pp. 704-716.

[4] B.-S. Yoo and C.-R. 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.

[5] Blue Gene Project (2007):http://www.research.ibm.com/bluegene/index.html.

[6] C.-C. Hsu, (2004): *I/O processor Allocation for Mesh Cluster Computers*, M.Sc. Thesis, Department of Computer Science and Information Engineering, National Taiwan University.

[7] C. G. Glass and L. M. Ni, (1992): *The turn model for adaptive routing*, Proceedings of the19th Annual International Symposium on Computer Architecture, pp. 278-287.

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

[9] C. Peterson, J. Sutton, P. Wiley, (1991): *iWARP: a 100-MPOS, LIW microprocessor for multicomputers*, IEEE Micro, vol. 11, no. 3, pp. 26-29, 81-87.

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

[11] Cray, *Cray XT3 Datasheet*, 2005.

[12] D. Bunde, V. J. Leung and J. Mache, (2004): *Communication Patterns and Allocation Strategies*, Sandia Technical Report SAND2003-4522.

[13] D. Das Sharma and D. K. Pradhan, (1996): *Submesh Allocation in Mesh-Multicomputers Using Busy-List: A Best-Fit Approach with Complete Recognition Capability*, Journal of Parallel and Distributed Computing, vol. 36, no. 2, pp. 106-118.

[14] D. Kulkarni, (2000): *Deterministic and Adaptive Routing in k-ary n-cube Networks*, CS 570 Project Report, Department of Computer Science, Colorado State University, Fort Collins.

[15] F. Wu, C.-C. Hsu and L.-P. Chou, (2003): *Processor Allocation in the Mesh Multiprocessors Using the Leapfrog Method*, IEEE Transactions on Parallel and Distributed Systems, vol. 14, no. 3, pp. 276-289.

[16] G. Min, (2003): *Performance Modelling and Analysis of Multicomputer Interconnection Networks*, PhD Thesis, Department of Computing Science, University of Glasgow.

[17] I. Ababneh and F. Fraij, (2001): *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.

[18] I. Ababneh, (2006): *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.

[19] I. Ababneh, (2001): *Job scheduling and contiguous processor allocation for three dimensional mesh multicomputers*, AMSE Advances in Modelling & Analysis, vol.6, no. 4, pp. 43-58.

[20] I. Foster, (1995): *Designing and Building Parallel Programs*, Concepts and Tools for Parallel Software Engineering, Addison-Wesley, 1st edition.

[21] I. Ismail, (1995): *Space sharing job scheduling policies for parallel computers*, PhD Thesis, Department of Electrical and Computer Engineering, Iowa State University.

[22] Intel Corp., (1991): *Paragon XP/S product overview*, Supercomputer Systems Division, Beaverton, Oregon.

[23] J. Ding and L.-N. Bhuyan, (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.

[24] J. Duato, C. Yalamanchili, and L. Ni,(1997): *Interconnection networks: an engineering approach*, IEEE Computer Society Press

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

[26] K. Suzaki, H. Tanuma, S. Hirano, Y. Ichisugi, C. Connelly, and M. Tsukamoto,(1996): *Multi-tasking Method on Parallel Computers which Combines a Contiguous and Non-contiguous Processor Partitioning Algorithm*, Proceedings of the 3rd International Workshop on Applied Parallel Computing, Industrial Computation and Optimization, Lecture Notes in Computer Science, Springer, London, pp. 641-650.

[27] K. Windisch, J. V. Miller, and V. Lo, (1995): *ProcSimity: an experimental tool for processor allocation and scheduling in highly parallel systems*, Proceedings of the 5th Symposium on the Frontiers of Massively Parallel Computation (Frontiers'95), Washington, DC, USA, IEEE Computer Society Press, pp. 414-421.

[28] K. Windisch, V. Lo, and B. Bose (1995): *Contiguous And Non-contiguous Processor Allocation Algorithms for k-ary n-cubes*, Technical Report, University of Oregon, Oregon, USA.

[29] K. Windisch, V. Lo, and B. Bose (1997*): Non-contiguous processor allocation algorithms for mesh-connected multicomputers*, IEEE Transactions on Parallel and Distributed Systems, vol. 8, no. 7, pp. 712-726.

[30] K.-H. Seo and S.-C. Kim, (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.

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

[32] M. Morris Mano, (1993): *Computer System Architecture*, Person Education Limited, 3rd edition.

[33] M. Noakes, D. A. Wallach, and W. J. Dally, (1993): *The J-machine multicomputer: an architecture evaluation*, Proceedings of the 20th

International Symposium Compute Architecture, pp. 224-235.

[34] N. Alzeidi (2007). *Performance Analysis of Wormhole Switched Interconnection Networks with Virtual Channels and Finite Buffers.* PhD Thesis, The Faculty of Information and Mathematical Sciences, University of Glasgow, Glasgow, U.K.

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

[36] P.-J. Chuang and N.-F. Tzeng, (1994): *Allocating precise submeshes in mesh connected systems*, IEEE Transactions on Parallel and Distributed Systems, vol. 5, no. 2, pp.211-217.

[37] ProcSimity V4.3 (1997): *User's Manual*, University of Oregon.

[38] S. Bani-Mohammad (2002): *Non-Contiguous Processor Allocation for 3D Mesh Multicomputers,* M.Sc. Thesis , the Prince Hussein Bin Abdullah College for Information Technology, Al al-Bayt University, Mafraq, Jordan.

[39] S. Bani-Mohammad (2008): *Efficient Processor Allocation Strategies for Mesh-Connected Multicomputers,* PhD Thesis, The Faculty of Information and Mathematical Sciences University of Glasgow, Glasgow, U.K.

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

[41] S. Bani-Mohammad, M. Ould-Khaoua, and I. Ababneh, (2007): *A New Processor Allocation Strategy with a High Degree of Contiguity in Mesh-Connected Multicomputers*, Journal of Simulation Modelling, Practice & Theory, vol. 15, no.4, pp. 465-480.

[42] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh and L. Mackenzie, (2007): *A Fast and Efficient Processor Allocation Strategy which Combines a Contiguous and Non-contiguous Processor Allocation Algorithms*, Technical Report; TR-2007-229, DCS Technical Report Series, Department of Computing Science, University of Glasgow, Glasgow, U.K.

[43] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, and L. Machenzie, (2007): *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, , Hsinchu, Taiwan.

[44] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, and L. Machenzie, (2006): *A Fast and Efficient Strategy for Sub-mesh Allocation with Minimal Allocation Overhead in 3D Mesh Connected Multicomputers*, Ubiquitous Computing and Communication Journal, vol. 1, no. 1, pp. 26-36, ISSN 1992-8424.

[45] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, and L. Machenzie, (2007): *A Performance Comparison of the Contiguous Allocation Strategies in 3D Mesh Connected Multicomputers*, Proceedings of The 5th International Symposium on Parallel and Distributed Processing and Applications (ISPA'07), LNCS 4742, Springer-Verlag Berlin Heidelberg, pp. 645-656.

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

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

[48] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, and Lewis M. Mackhenzie, (2006): *An Efficient Turning Busy List Sub-mesh Allocation Strategy for 3D Mesh Connected Multicomputers*, Proceedings of the 7th Annual Post Graduate Symposium on the Convergence of Telecommunications, Networking & Broadcasting, (PGNET 2006), Liverpool John Moores University, UK, pp. 37-43.

[49] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, and Lewis M. Mackhenzie, (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.

[50] S. Bani-Mohammad, I. Abaneh, and M. Hamdan, (2011): *Performance Evaluation of Noncontiguous Allocation Algorithms for 2D Mesh Interconnection Networks*, Journal of Systems and Software, Elsevier, Volume 84, No. 12, pp. 2156-2170.

[51] V. Adve and M. Vernon (1994): *Performance Analysis of Mesh Interconnection Networks with Deterministic Routing*, IEEE Transactions on Parallel and Distributed Systems, vol. 5, no. 3, pp. 225-246.

[52] V. Kumar, A. Grama, A. Gupta, and G. Karypis, (2003): *Introduction to Parallel Computing*, The Benjamin/Cummings publishing Company, Inc., Redwood City, California,.

[53] V. Leung, E. Arkin, M. Bender, D. Bunde, J. Johnston, A. Lal, J. Mitchell, C. Phillips, and S. Seiden, (2002): *Processor Allocation on Cplant: Achieving General Processor Locality Using One-Dimensional Allocation Strategies*, Proceedings of the 4th IEEE International Conference on Cluster Computing, IEEE Computer Society Press, pp. 296-304.

[54] Y. Aridor, T. Domany, O. Goldshmidt, J. Moreira, and E. Shmueli, (2005): *Resource allocation and utilization in the BlueGene/L supercomputer*, IBM Journal of Research and Development, vol. 49, no. 2/3, pp. 425-436.

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

ملخص

في التخصيص غير المتجاور، يمكن تجزئة طلب مهمة الى اجزاء اصغر و التي من الممكن تخصيصها بشكل غير متجاور في شبكات جزئية فارغة عوضا عن الإنتضار الطويل حتى يتم توفر شبكة فرعية متجاورة ومطابقة بالحجم والشكل المطلوبين. بغض النضر عن شرط التجاورية يتوقع تقليل عدد الكسيرات وزيادة إستغلال النظام. ومع ذلك، المسافات المقطوعة من قبل الرسائل يمكن ان تكون طويلة. ونتيجة لذلك يتم زيادة مقدارالإتصال على الشبكة، وخاصة التزاحم. مقدار الإتصال الزائد يعتمد على كيفية تقسيم طلب التخصيص وحجزه للشبكات الفرعة الفارغة. في هذه الدراسة، أقترح سياسة تخصيص غير متجاور جديدة، يشار إليها بـ التخصيص غير المتجاور بإستخدام التحشير في متعددات الحواسيب ثنائي الأبعاد. في السياسة المقترحة، يتم تحشير المهمة الواحدة في أكثر من مكان متوفر ضمن المعالجات المخصصة، حيث المعالجات المتاحة والمتبقية تشكّل في النظام شبكة جزئية كبيرة. لتقييم لتحسين الأداء الذي حققته السياسة المقترحة ومقارنتها مع سياسات تخصيص غير متجاور معروفه من قبل، فقد أجرينا تجارب محاكاة واسعة النطاق على إفتراض أن عملية التوجيه (Wormhole routing) وأنماط الإتصال، واحد- لجميع، عشوائي، والجار القريب. بينت النتائج أن السياسة المقترحه ألغت كلا من الكسيرات الداخلية والخارجية و قللت من مقدار الإتصال على الشبكة وبالتالي أدة الى تحسين الأداء من حيث الفترة الزمنية للمهمة وإستغلال النظام.