UNIVERSITY *of* GLASGOW

Efficient Processor Allocation Strategies for Mesh-Connected Multicomputers

A Thesis Submitted

by

Saad O. Bani Mohammad

for

The Degree of Doctor of Philosophy

to

The Faculty of Information and Mathematical Sciences

University of Glasgow

Abstract

Efficient processor allocation and job scheduling algorithms are critical if the full computational power of large-scale multicomputers is to be harnessed effectively. Processor allocation is responsible for selecting the set of processors on which parallel jobs are executed, whereas job scheduling is responsible for determining the order in which the jobs are executed. Many processor allocation strategies have been devised for mesh-connected multicomputers and these can be divided into two main categories: *contiguous* and *non-contiguous*. In contiguous allocation, jobs are allocated distinct contiguous processor sub-meshes for the duration of their execution. Such a strategy could lead to high processor fragmentation which degrades system performance in terms of, for example, the turnaround time and system utilisation. In non-contiguous allocation, a job can execute on multiple disjoint smaller sub-meshes rather than waiting until a single sub-mesh of the requested size and shape is available. Although non-contiguous allocation increases message contention inside the network, lifting the contiguity condition can reduce processor fragmentation and increase system utilisation.

Processor fragmentation can be of two types: *internal* and *external*. The former occurs when more processors are allocated to a job than it requires while the latter occurs when there are free processors enough in number to satisfy another job request, but they are not allocated to it because they are not contiguous. A lot of efforts have been devoted to reducing fragmentation, and a number of contiguous allocation strategies have been devised to recognize complete sub-meshes during allocation. Most of these strategies have been suggested for 2D mesh-connected multicomputers. However, although the 3D mesh has been the underlying network topology for a number of important multicomputers, there has been relatively little activity with regard to designing similar strategies for such a network. The very few contiguous allocation strategies suggested for the 3D mesh achieve complete sub-mesh recognition ability only at the expense of a high *allocation overhead* (i.e., allocation and de-allocation time). Furthermore, the allocation overhead in the existing contiguous strategies often grows with system size. The main challenge is therefore to devise an efficient contiguous allocation strategy that can exhibit good performance (e.g., a low job turnaround time and high system utilisation) with a low allocation overhead.

The first part of the research presents a new contiguous allocation strategy, referred to as Turning Busy List (TBL), for 3D mesh-connected multicomputers. The TBL strategy considers only those available free sub-meshes which border from the left of those already allocated sub-meshes or which have their left boundaries aligned with that of the whole mesh network. Moreover TBL uses an efficient scheme to facilitate the detection of such available sub-meshes while maintaining a low allocation overhead. This is achieved through maintaining a list of allocated sub-meshes in order to efficiently determine the processors that can form an allocation sub-mesh for a new allocation request. The new strategy is able to identify a free sub-mesh of the requested size as long as it exists in the mesh. Results from extensive simulations under various operating loads reveal that TBL manages to deliver competitive performance (i.e., low turnaround times and high system utilisation) with a much lower allocation overhead compared to other well-known existing strategies.

Most existing non-contiguous allocation strategies that have been suggested for the mesh suffer from several problems that include internal fragmentation, external fragmentation,

and message contention inside the network. Furthermore, the allocation of processors to job requests is not based on free contiguous sub-meshes in these existing strategies. The second part of this research proposes a new non-contiguous allocation strategy, referred to as Greedy Available Busy List (GABL) strategy that eliminates both internal and external fragmentation and alleviates the contention in the network. GABL combines the desirable features of both contiguous and non-contiguous allocation strategies as it adopts the contiguous allocation used in our TBL strategy. Moreover, GABL is flexible enough in that it could be applied to either the 2D or 3D mesh. However, for the sake of the present study, the new non-contiguous allocation strategy is discussed for the 2D mesh and compares its performance against that of well-known non-contiguous allocation strategies suggested for this network. One of the desirable features of GABL is that it can maintain a high degree of contiguity between processors compared to the previous allocation strategies. This, in turn, decreases the number of sub-meshes allocated to a job, and thus decreases message distances, resulting in a low inter-processor communication overhead. The performance analysis here indicates that the new proposed strategy has lower turnaround time than the previous non-contiguous allocation strategies for most considered cases. Moreover, in the presence of high message contention due to heavy network traffic, GABL exhibits superior performance in terms of the turnaround time over the previous contiguous and non-contiguous allocation strategies. Furthermore, GABL exhibits a high system utilisation as it manages to eliminate both internal and external fragmentation.

The performance of many allocation strategies including the ones suggested above, has been evaluated under the assumption that job execution times follow an exponential distribution. However, many measurement studies have convincingly demonstrated that the execution times of certain computational applications are best characterized by heavy-tailed job execution times; that is, many jobs have short execution times and comparatively few have very long execution times. Motivated by this observation, the final part of this thesis reviews the performance of several contiguous allocation strategies, including TBL, in the context of heavy-tailed distributions. This research is the first to analyze the performance impact of heavy-tailed job execution times on the allocation strategies suggested for mesh-connected multicomputers. The results show that the performance of the contiguous allocation strategies degrades sharply when the distribution of job execution times is heavy-tailed. Further, adopting an appropriate scheduling strategy, such as Shortest-Service-Demand (SSD) as opposed to First-Come-First-Served (FCFS), can significantly reduce the detrimental effects of heavy-tailed distributions. Finally, while the new contiguous allocation strategy (TBL) is as good as the best competitor of the previous contiguous allocation strategies in terms of job turnaround time and system utilisation, it is substantially more efficient in terms of allocation overhead.

To my parents,

To my wife and children,

To my brothers and my sister

for their endless love, support and encouragement

Acknowledgments

I would like to express my deep gratitude to my supervisors, Dr. Mohamed Ould-Khaoua and Dr. Lewis M. Mackenzie for their inspiring guidance, valuable advice and constant encouragement throughout the progress of this work. Their suggestions, criticism and their frequent questions motivated this research and they never failed to provide their help at all stages of this research.

I would also like to thank Dr. Ismail Ababneh for his help and advice at the early stages of my Ph.D. program and for the time reviewing my papers and giving me constructive and insightful comments and reviews. My gratitude also goes to Prof. Joe Sventek for his helpful comments and the time for reading my first year report through my first year VIVA.

I am highly indebted to the Al al-Bayt University, Jordan, for the financial support and for granting me a scholarship to pursue my higher studies and give my thanks to my colleagues there. My thanks are also to all the staff of the Department of Computing Science, University of Glasgow, for their kind and friendly support. I am also grateful to my caring colleagues and friends here in the UK and back home for their friendship and encouragement during my time at Glasgow University.

I would like to dedicate this thesis to my family: my parents, my brothers and my sister, whose love and encouragement from a distant land were the motivating factors for completion of this work. Finally, I would like to express my dearest gratitude to my wife, whose unconditional love, support, patience, and caring were and always will be a source of inspiration, my son Yamen and my daughter Salma, who are the blessings of my life. This thesis would not have been possible without the help of my family. I could never be as happy as I am without each of you. You are all very precious to me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Parallel computers are generally considered to be one of the most feasible ways of achieving the ever-growing computational power required by many real-life parallel applications, especially in the fields of science and engineering [43, 70, 90]. A Parallel Computer consists of a set of processors that cooperate with each other to find a solution to a given problem [36]. The inter-processor communication may be based on either the *shared-memory* or *distributed-memory* model. In shared-memory architectures, also known as *multiprocessors*, processors communicate via shared memory. However, in distributed-memory parallel computers, also known as *multicomputers*, processors communicate by means of interchanging messages through an interconnection network [4, 29, 64, 83].

Generally, interconnection networks can be divided into two categories: *indirect* and *direct* networks [4, 5, 14, 29, 32, 64, 83]. In indirect networks, multiple intermediate stages of switches are used to interconnect the nodes (i.e., processors) of a multiprocessor; examples of indirect networks include the crossbar [32, 83], bus [5, 83], and multistage interconnection networks [14, 83]. In direct networks, each node has a point-to-point

connection to one or more nodes (known as its neighbours), allowing for direct communication between these nodes; examples of direct networks include the mesh [4, 82], *k*-ary *n*-cube [29], and hypercube [43]. Direct 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 [4, 29, 64, 90]. Moreover, direct networks are able to exploit communication locality (nearest neighbour communication) that is exhibited by many real-world applications.

Among the various multicomputer architectures, those based on the mesh network have received much attention due to the simplicity, structural regularity, partition-ability, and ease of implementation of this network [9, 18, 20, 21, 27, 31, 33, 35, 51, 52, 77, 78, 85, 99]. Meshes are suited to a variety of applications, including matrix computations, image processing and problems whose task graphs can be embedded naturally into the mesh [27, 89, 95]. Moreover, the mesh has been used as the underlying network in a number of practical and experimental parallel machines, such as the Intel Paragon [39], the Cray XT3 [19, 60], the MIT J-machine [61], the Cray T3D [67], the Cray T3E [25], the iWARP [15], the IBM BlueGene/L [10, 55, 97, 98], and the Delta Touchstone [40].

Definition 1.1*: An n-dimensional mesh has* $k_0 \times k_1 \times ..... \times k_{n-2} \times k_{n-1}$ *nodes, where* k_i *is the number of nodes along the* i^{th} *dimension and* $k_i \geq 2$*. Each node* a *is identified by* n *coordinates,* $\rho_0(a), \rho_1(a), ..., \rho_{n-2}(a), \rho_{n-1}(a)$*, where* $0 \leq \rho_i(a) < k_i$ *for* $0 \leq i < n$*. Two nodes* a *and* b *are neighbours if and only if* $\rho_i(a) = \rho_i(b)$ *for all dimensions, except for one dimension* j*, where* $\rho_j(b) = \rho_j(a) \pm 1$*. Each node in a mesh refers to a processor, and any two neighbours are interconnected by a direct communication link.*

Definition 1.2*: A 2D mesh, referred to as* $M(W,L)$*, consists of* $W \times L$ *processors, where* W *is the width of the mesh and* L *is its length. Every processor is denoted by a pair of coordinates* (x,y)*, where* $0 \le x < W$ *and* $0 \le y < L$*. A processor is connected by a bidirectional communication link to each of its neighbours.*

Definition 1.3*: In a 2D mesh,* $M(W,L)$*, a sub-mesh* $S(w,l)$ *is a two-dimensional sub-mesh of nodes belonging to* $M(W,L)$ *with width* w *and length* l*, where* $0 < w \le W$ *and* $0 < l \le L$*.* $S(w,l)$ *is represented by the coordinates* (x,y,x',y')*, where* (x,y) *is the lower left corner of the sub-mesh, and* (x',y') *is its upper right corner. The lower left corner node is called the base node of the sub-mesh, whereas the upper right corner node is the end node. Here,* $w = x' - x + 1$ *and* $l = y' - y + 1$*. The size of* $S(w,l)$ *is* $w \times l$ *processors.*

Definition 1.4*: In a 2D mesh,* $M(W,L)$*, a suitable sub-mesh* $S(w,l)$ *is a free sub-mesh that satisfies the conditions:* $w \ge \alpha$ *and* $l \ge \beta$ *assuming that the allocation of* $S(\alpha,\beta)$ *is requested, where the allocation refers to selecting a set of processors to an incoming job.*

Figure 1.1 shows an example of a 4×4 2D mesh, where allocated processors are denoted by shaded circles and free processors are denoted by clear circles. The mesh network has the desirable property of being partitionable into smaller sub-meshes [18, 49, 73, 77, 79, 85]. For example, (0, 0, 2, 1) represents the 3×2 sub-mesh S in Figure 1.1, where (0, 0) are the coordinates of the base of the sub-mesh and (2, 1) are the coordinates of its end. A partitionable system has the advantage of enabling the allocation of multiple simultaneous jobs, which can result in good processor utilisation [18, 77, 85]. The execution time of a job can often be reduced by allocating as many processors to the job as possible. In the presence of multiple jobs, the mesh can be partitioned into sub-meshes so that each job can be

allocated its own sub-mesh [85]. When a job departs the mesh system, its allocated processors need to be combined with other idle processors in the mesh system. Otherwise, severe processor fragmentation may arise, causing degradation in the overall system performance [18, 49, 73, 77, 79, 85].

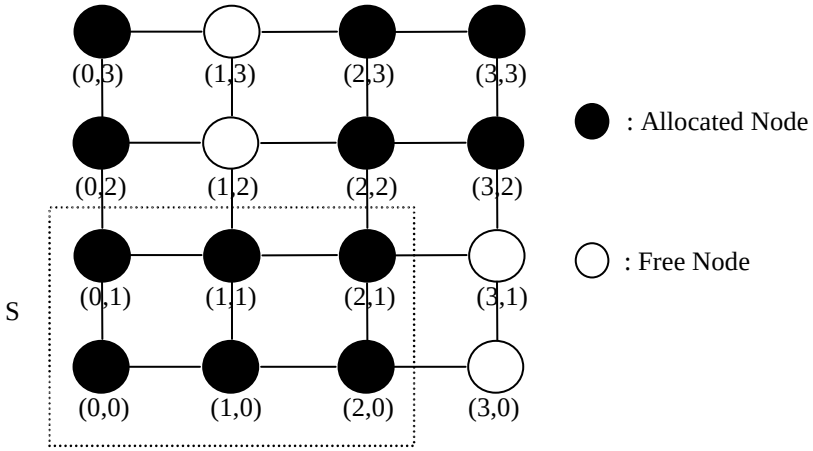


Figure 1.1: An Example of a 4×4 2D mesh

In this research, we assume that jobs executing on mesh-connected multicomputers are parallel programs consisting of tasks that communicate with each other via message passing. Upon arrival, a job requests the allocation of a sub-mesh of a given size. As previously reported in definition 1.4, the selection of the processors to be allocated to the job is referred to as *processor allocation*.

The remainder of this chapter is organized as follows. Section 1.1 describes the different types of processor allocation algorithms and provides an overview of the processor allocation strategies proposed previously for 2D and 3D mesh-connected multicomputers. We limit our attention to these low-dimensional meshes because they have received much consideration by researchers recently [9, 11, 16, 24, 26, 28, 31, 33, 34, 35, 45, 51, 52, 71, 72, 73, 74, 75, 76, 77, 78, 79, 81, 97]. Furthermore, many parallel machines in the real world, such as the iWARP [15], the MIT J-machine [61], the Intel Paragon [39], the Cray T3D [67],

the IBM BlueGene/L [10, 55, 97, 98], and the Cray T3E [25] have used these low-dimensional meshes as their underlying topology. Section 1.2 presents the motivations for the present research. Section 1.3 presents the thesis statement. Section 1.4 presents the main contributions of this research. Finally, Section 1.5 provides an outline of the rest of the thesis.

1.1 Processor Allocation

Efficient processor allocation and *job scheduling* are critical if the full computational power of large-scale multicomputers is to be harnessed effectively [9, 27, 31, 78, 94]. Processor allocation is responsible for selecting the set of processors on which a parallel job is executed, whereas job scheduling is responsible for determining the order in which jobs are selected for execution [9, 11, 20]. The job scheduler selects the next job to execute using the scheduling policy, and then the processor allocator finds free processors for the selected job [50, 66]. If an arriving job cannot be run immediately, due to a lack of free processors or the existence of other waiting jobs, for example, it is diverted to the waiting queue. Once processors are allocated to a job, the job holds these processors exclusively until it finishes running. At this time, it departs from the system and the processors are freed for use by other jobs.

A processor allocation strategy may have a partial or full sub-mesh recognition capability [85, 99]. Full sub-mesh recognition capability means that the allocation strategy can identify a free sub-mesh of the requested size as long as it exists in the mesh system, while partial recognition capability means that the allocation strategy may fail to identify a free sub-mesh of the requested size although one exists. Having full sub-mesh recognition capability improves system performance, but increases the time needed to allocate a sub-mesh to a new job, as has been shown in [26, 31, 34, 94, 97]. With increased system size, the time to search

for free processors that satisfy an incoming request might be comparable to the job's execution time [46]. Hence it is important to develop techniques for minimizing the search time (also referred to as the allocation time). Minimization of the allocation time in mesh-connected multicomputers is fundamental. This is because a major goal of parallel execution is to minimize the turnaround time of jobs (i.e., the time that a job is expected to spend in the mesh system from arrival to departure). However, the allocation time of many existing allocation strategies [26, 31, 34, 94, 97] increases when the number of processors in the mesh increases.

Processor allocation strategies can be divided into two main categories: *contiguous* and *non-contiguous* [18, 49, 71, 72, 73, 77, 79, 85]. In the contiguous allocation strategy, jobs are allocated distinct contiguous processor sub-meshes for the duration of their execution [9, 11, 21, 27, 31, 33, 35, 38, 48, 65, 74, 78, 80, 94, 99]. Such a strategy can lead to high processor fragmentation, as has been shown in [99]. High processor fragmentation degrades system performance parameters, such as the average turnaround time of jobs and the mean system utilisation (i.e., the percentage of processors that are utilized over a given period of time).

Processor fragmentation is of two types: *internal* and *external* [11, 85]. Internal fragmentation occurs when more processors are allocated to a job than it requires, whereas external fragmentation occurs when there are free processors sufficient in number to satisfy a pending allocation request, but they are not allocated because they are not contiguous.

Examples of contiguous allocation strategies[1] that have been developed for 2D mesh-connected multicomputers include the Two Dimensional Buddy System (2DBS) [48], Frame Sliding (FS) [65], Adaptive Scan (AS) [41], and First Fit (FF) and Best Fit (BF) [99]. The

2DBS [48] is simple, but it applies to square mesh systems only and suffers from internal and external processor fragmentation. The FS strategy [65] is applicable to a mesh of any size and any sub-mesh shape, but it suffers from external fragmentation as it cannot recognize all free sub-meshes. The frame sliding operation is such that it may skip over a large-enough free sub-mesh because the frame sliding operation is by the job's width and length. The AS strategy [41] has been shown to improve system performance by switching the orientation (i.e., rotation) of any allocation request that cannot be accommodated in the requested orientation. A job that requests an $\alpha \times \beta$ sub-mesh may be allocated a $\beta \times \alpha$ sub-mesh. However, the allocation time of AS is high compared to FS because the AS strategy scans processors in the mesh system with a vertical stride distance of 1 processor (i.e., Jumps to successive processors are by 1 processor). The FF and BF strategies [99] can detect all large-enough free sub-meshes, but they lack complete sub-mesh recognition ability in that they do not consider switching the orientation of requests.

Examples of contiguous allocation strategies that have been suggested for 3D mesh-connected multicomputers include First Fit (FF) and Best Fit (BF) [34], Turning First Fit (TFF) and Turning Best Fit (TBF) [34], and the Allocation Algorithm for the IBM BlueGene/L [97]. The FF and BF strategies [34] are simple, but they do not permit changing the orientation of requests, hence they suffer from high external processor fragmentation. The TFF and TBF [34] improve performance by considering all orientations of the request when needed, however their allocation overhead (i.e., allocation and de-allocation time) is high. The Allocation Algorithm for the IBM BlueGene/L [97] assumes that a job can utilize an integer number of midplanes (a midplane is a page of $8 \times 8 \times 8$ processors). Otherwise, there is internal processor fragmentation, which can be severe because this allocation unit is

[1] The details of the existing contiguous allocation strategies will be provided in Chapter 2.

rather large.

Although contiguous allocation suffers from low overall system utilisation [31, 33, 85], it has been proposed for use in the IBM BlueGene/L for security reasons; because of the sensitive nature of some of its applications, a BlueGene/L job is allocated a sub-mesh of processors that is isolated from sub-meshes allocated to other jobs [97].

So as to reduce the processor fragmentation that contiguous allocation suffers from, non-contiguous allocation has been proposed [18, 44, 49, 71, 72, 77, 85]. In non-contiguous allocation, a job can execute on multiple disjoint smaller sub-meshes rather than always waiting until a single sub-mesh of the requested size and shape is available [18, 44, 49, 71, 72, 77, 85]. In Figure 1.1 above, if a job requests the allocation of a sub-mesh of size 2×2, contiguous allocation fails because no 2×2 sub-mesh of free processors is available. However, the four free processors (depicted in the figure by white circles) can be allocated to the job when the non-contiguous allocation is adopted. Although non-contiguous allocation can increase message contention in the network, lifting the contiguity condition is expected to reduce processor fragmentation and increase processor utilisation, as has been shown in [85].

The wide adoption of *wormhole routing*[2] [11, 18, 83] in practical systems has encouraged researchers to consider non-contiguous allocation for multicomputers that use networks characterised by long communication distances (e.g., the mesh) [18, 49, 71, 72, 77, 85]. A major advantage of wormhole routing over earlier switching techniques, especially store-and-forward, is that message latency is less sensitive to message distance, especially under

[2] Wormhole routing is a switching technique which has been used in multicomputers. The detailed operation of wormhole routing will be provided in Chapter 2.

light to moderate traffic conditions [2, 43]. Recognising that wormhole routing can mitigate the additional communication overhead, non-contiguous allocation has received increased interest from the research community due to its ability to allocate small sub-meshes of free processors scattered throughout the mesh-connected multicomputer instead of waiting until a single large free sub-mesh is available, which significantly decreases external processor fragmentation [11, 18, 71, 72, 77, 85]. Experiments on a 208-processor Paragon, a multicomputer based on a 2D mesh with wormhole routing, have indicated that the communication overhead in non-contiguous allocation may not be so severe as to offset the benefits of reduced fragmentation [85].

The method used for partitioning allocation requests has considerable impact on the performance of non-contiguous allocation [71, 72]. In particular, the partitioning process should aim to maintain a high degree of contiguity between the processors allocated to a parallel job. This is so that the communication overhead is reduced without adversely affecting the overall system performance [71, 72, 73, 79].

Existing non-contiguous allocation strategies[3] include Random [85], Paging [85], Multiple Buddy Strategy (MBS) [85], Adaptive Non-Contiguous Allocation (ANCA) [18], Adaptive Scan and Multiple Buddy (AS&MB) [49], and several recent Paging variants [24]. In Random [85], both internal and external fragmentations are eliminated, but high communication interference amongst jobs is to be expected. In Paging [85], there is some degree of contiguity among processors allocated to a parallel job, and contiguity can be increased by using larger pages. However, there can be internal processor fragmentation for page sizes larger than one. MBS [85] has been shown to improve performance compared to the earlier strategies, but it may fail to allocate a contiguous sub-mesh of free processors

although one exists. Hence, it can increase the communication overhead. ANCA [18] subdivides the request into 2^i equal parts during the i^{th} iteration. Also, it requires that allocation to all parts occur in the same partitioning and allocation iteration, which can result in skipping over the possibility of allocating larger sub-meshes for a large part of the request in a previous iteration. This can increase the communication overhead. Moreover, allocation fails if a side length of the sub-parts reaches one, which can cause external fragmentation. The performance of AS&MB [49] in terms of response times and service times can be almost identical to that of MBS [85] as has been shown in [44]. However, AS&MB suffers from high allocation overhead for large meshes. In the Paging variants [24], the unit of allocation is a single processor, whereas it can be larger in MBS [85] and ANCA [18]. As a consequence, the Paging variants can require a long time to reach an allocation decision in large machines [97].

1.2 Motivations

The results of previous research suggest that new contiguous as well as non-contiguous allocation strategies for mesh-connected multicomputers are needed. The motivation for the development of a new *contiguous* allocation strategy for the 3D mesh network has been driven by the observation that the existing contiguous allocation strategies suggested for the 3D mesh achieve complete sub-mesh recognition capability only at the expense of a high allocation overhead [31, 34, 94, 97] that accounts for the time required to allocate and de-allocate processors to an incoming job. The allocation overhead of the previously proposed algorithms for contiguous allocation in 3D meshes and tori grow with the system size [26, 31, 34, 94].

[3] The details of the existing non-contiguous allocation strategies will be provided in Chapter 2.

The motivation for the development of a new *non-contiguous* allocation strategy for the 2D mesh has been driven by the observation that the existing non-contiguous allocation strategies suggested for the 2D mesh network suffer from several problems that include internal fragmentation, external fragmentation, and message contention inside the network [18, 24, 84, 85]. Furthermore, the allocation of processors to job requests is not based on free contiguous sub-meshes in all of the existing strategies [18, 85] but rather on artificial predefined geometric or arithmetic patterns [18, 85]. For example, in [18], ANCA subdivides the job request into two equal parts, and the subparts are successively subdivided in a similar fashion if allocation fails for any of them. In [85], MBS bases partitioning on a base-4 representation of the number of processors requested, and partitioning in Paging [85] 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.

Many previous studies [6, 11, 18, 27, 31, 33, 34, 35, 38, 48, 49, 51, 52, 74, 78, 85, 94, 99] have used the exponential distribution for job execution times when evaluating the performance of a new allocation strategy. Therefore, an exponential distribution has been assumed for our suggested allocation strategies in order to evaluate their performance properties against those of the existing strategies. However, many measurement studies [22, 47, 56, 57, 58, 59, 88, 96] have convincingly demonstrated that the execution times of certain computational jobs can be characterised by heavy-tailed distributions; that is, many jobs are short and fewer are long. Heavy-tailed distributions can capture this variability and have been shown to behave quite differently from the exponential distribution [22, 57, 58, 75]. In particular, when sampling random variables that follow a heavy-tailed distribution, the probability of large generated values is non-negligible [22, 47, 56, 57, 58, 59, 88, 96].

1.3 Thesis Statement

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 allocation overhead. On the other hand, existing non-contiguous allocation strategies suffer from several problems that include internal fragmentation, external fragmentation, and message contention inside the network. Also, they do not exploit knowledge of the current state of the system (e.g., currently available sub-meshes).

A number of measurement studies have convincingly demonstrated that the execution times of many computational jobs can be characterised by heavy-tailed distributions (e.g., Bounded Pareto). However, the effectiveness of most suggested allocation strategies have been evaluated under the assumption of exponentially distributed execution times, which may not reflect all possible practical scenarios.

This thesis will justify the following key claims:

T1: A contiguous allocation strategy can be developed that exhibits competitive system performance (e.g., a low job turnaround time and high system utilisation) with a lower allocation overhead compared to existing strategies for 3D mesh-connected multicomputers. This is achieved by maintaining a list of allocated sub-meshes in order to efficiently determine the processors that can form an allocation sub-mesh for a new allocation request.

T2: A non-contiguous allocation strategy for 2D mesh-connected multicomputers can be developed where requests are partitioned by tracking free sub-meshes so as to

maintain a high degree of contiguity. This strategy is free from both internal and external fragmentation, and reduces message contention. It also improves system performance in terms of job turnaround times compared to the existing strategies and exhibits a high system utilisation as it manages to eliminate both internal and external fragmentation.

T3: The performance of contiguous allocation strategies can be significantly affected by both the type of the distribution adopted for job execution times and the scheduling strategy adopted for determining the order in which jobs are selected for execution. To date, no study has been reported that analyses the impact of heavy-tailed job execution on the performance of the allocation strategies. When the performance of the new contiguous allocation strategy described in T1, as well as the traditional allocation strategies, is re-visited in the context of jobs with execution times following both heavy-tailed and exponential distributions, using First-Come-First-Served (FCFS) scheduling strategy, the performance of the allocation strategies degrades when the distribution of job execution times is heavy-tailed, an appropriate scheduling strategy should be adopted to deal with heavy-tailed distributions and, in this regard, our analysis will demonstrate that the Shortest-Service-Demand (SSD) scheduling strategy exhibits superior performance over the FCFS scheduling strategy.

1.4 Main Contributions

To address the above research concerns listed in the motivations section, this thesis presents efficient contiguous and non-contiguous allocation strategies that overcome the limitations of the existing strategies suggested previously for the 2D and 3D mesh networks.

In the first part of this research, an efficient contiguous allocation algorithm, referred to as Turning Busy List (or TBL for short), for 3D mesh-connected multicomputers is proposed. The TBL strategy considers only those available free sub-meshes which border from the left of those already allocated sub-meshes or which have their left boundaries aligned with that of the whole mesh network.The TBL strategy can identify a free sub-mesh of the requested size as long as it exists in the mesh system. It can do so because it relies on a new approach that maintains a list of allocated sub-meshes to determine the processors that can form an allocation sub-mesh for a new allocation request. The TBL strategy is shown to exhibit a lower allocation overhead than that in the previous strategies [34]. Moreover, simulation results show that system performance, in terms of parameters such as turnaround times and system utilisation, is as good as that of the previously promising proposed strategies [34].

In the second part of this research, a new non-contiguous allocation algorithm, referred to as Greedy Available Busy List (or GABL for short), for the 2D mesh-connected multicomputer is suggested. The GABL 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 with low allocation overhead. The desirable features of non-contiguous allocation are reducing processor fragmentation and alleviating the communication overhead between processors allocated to a job by maintaining a high degree of contiguity between them. Moreover, GABL 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 GABL strategy relies on a new approach that maintains a higher degree of contiguity among processors than that of the previous non-contiguous allocation strategies. 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 GABL has better performance in terms of the turnaround time than the previous non-contiguous allocation strategies proposed in [85]. Moreover, when message contention is increased inside the network due to using all-to-all communication patterns, for example, GABL exhibits superior performance over previous contiguous and non-contiguous allocation strategies. Furthermore, GABL is able to eliminate internal as well as external fragmentation from which several previous allocation strategies suffer.

In the Final part of this research, the performance of the existing contiguous allocation strategies for 3D mesh-connected multicomputers, including the ones proposed in this research, is revisited in the context of heavy-tailed job execution times. To the best of our knowledge, this research is the first to consider heavy-tailed distributions in the context of processor allocation in mesh-connected multicomputers. In this part, the performance of allocation strategies is measured in terms of the usual performance parameters [6, 9, 18, 21, 27, 31, 33, 34, 35, 38, 71, 72, 73, 74, 75, 76, 77, 78, 79, 85, 94, 99], including the average turnaround time and mean system utilisation, as well as the measured allocation overhead, that is, the time that the allocation and de-allocation operations take per job. Our results show that the system performance of the allocation strategies degrades considerably when the distribution of job execution times is heavy-tailed. Our analysis also shows that when job execution times follow a heavy-tailed distribution, the SSD scheduling strategy improves the performance of the allocation strategies compared to the FCFS scheduling strategy. In addition, the results show that our suggested contiguous allocation strategy has a low allocation overhead and its system performance in terms of average turnaround time and

mean system utilisation is as good as the best competitor of the previous contiguous allocation strategies.

1.5 Outline of the Thesis

The rest of the thesis is organised as follows. Chapter 2 describes well-known contiguous and 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 describes the method of study used in this research and justifies the selection of simulation as a study tool.

Chapter 3 introduces the Turning Busy List (TBL) as a new contiguous allocation algorithm for 3D mesh-connected multicomputers, and discusses the main features of this algorithm. Also, extensive simulation experiments are carried out in order to compare the performance of the proposed allocation strategy against well-known contiguous allocation strategies.

Chapter 4 introduces the Greedy Available Busy List (GABL) strategy as a new non-contiguous allocation algorithm for 2D mesh-connected multicomputers. The main features of the GABL strategy are also discussed, and extensive simulation experiments are carried out in order to evaluate the performance of the this strategy and compare it against existing well-known contiguous and non-contiguous allocation strategies.

Chapter 5 conducts an extensive performance study of the existing contiguous allocation strategies, including the one proposed in Chapter 3 for 3D mesh-connected multicomputers when the job execution times follow a heavy-tailed distribution. The strategies are evaluated using simulation experiments for both FCFS and SSD scheduling strategies under a variety of system loads and system sizes.

Chapter 6 summarises the main results presented in this research and outlines possible directions to continue this work in the future.

Chapter 2

Background and Preliminaries

2.1 Introduction

Space sharing can be used in addition to *time sharing* in parallel computers due to the presence of multiple processors in such computers [11, 17, 37]. In space sharing, a job is allocated a distinct subset of processors; that is, no processor is concurrently assigned to more than one job [6, 11, 17, 37]. In time sharing, a processor spends an interval of time executing a job, then it switches to the execution of another one [6, 11, 17, 37]. The overhead that results from the context switches[1] in time sharing degrades system performance, and as a result it has become less popular in practical systems [11, 17].

Most existing allocation strategies employ space sharing [9, 11, 18, 20, 21, 24, 26, 27, 31, 33, 34, 35, 38, 48, 49, 51, 52, 65, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 85, 94, 99] and can be categorised as *contiguous* and/or *non-contiguous*. In contiguous allocation [9, 20, 21, 26,

[1] A context switch is the process of storing and restoring the state (context) of processors such that multiple jobs can share these processors.

27, 31, 33, 34, 35, 38, 48, 52, 65, 74, 75, 78, 94, 99], the allocated processors are physically contiguous and have the same topology as the underlying multicomputer network (i.e., mesh) in order to keep minimal the communication overhead between allocated processors. A direct consequence of contiguous allocation is that good system utilization is not achievable due to the *fragmentation problem* that contiguous allocation suffers from [18, 85]. As previously reported in Chapter 1, the fragmentation problem is of two types: *internal* and *external* processor fragmentation. Internal fragmentation occurs when some of the processors allocated to a job are not used, whereas external fragmentation occurs when a sufficient number of free processors are available to satisfy a job request but they are not allocated to it because they are not contiguous.

Figure 2.1 shows a job that requested 2 processors and was allocated 4 processors; hence there is an internal fragmentation of 50%. Figure 2.2 shows the existence of an external fragmentation of 4 processors due to processor non-contiguity, assuming that the allocation strategy is contiguous. The 4 free processors are not allocated to the request because they are not contiguous. To solve this problem, some researchers [18, 24, 49, 71, 72, 84, 85] have opted for non-contiguous allocation where a job can be executed on multiple disjoint sub-meshes rather than waiting until a single sub-mesh of the requested size is available. Initially, non-contiguous allocation did not receive much attention from researchers. This is because the communication latency was very sensitive to the distance between communicating nodes when store-and-forward switching was dominant in the first generation of multicomputer networks [11]. However, advances in switching technique, such as *wormhole switching (also widely known as wormhole routing)* [2, 4, 11, 13, 29, 71, 72, 83], have made non-contiguous allocation plausible in mesh-connected multicomputers. This is because one of the advantages of wormhole switching over earlier switching schemes, mainly store-and-forward, is that message latency depends less on the message distance [2, 43].

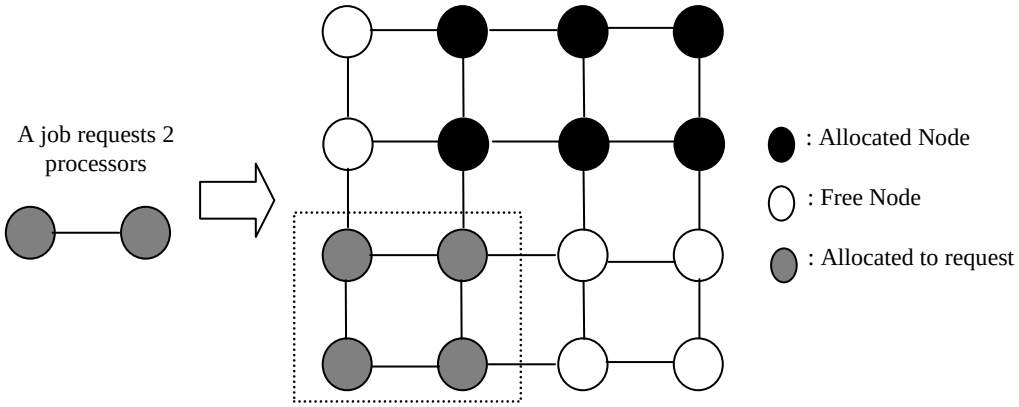


Figure 2.1: An internal fragmentation of 2 processors

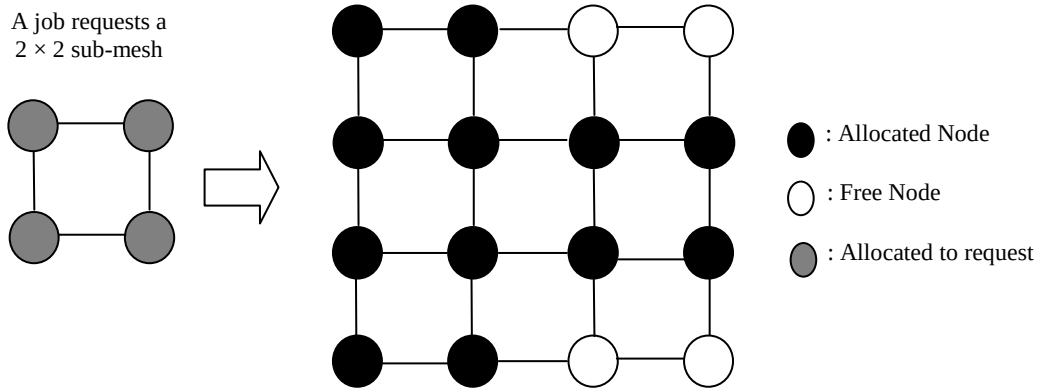


Figure 2.2: An external fragmentation of 4 processors assuming that the allocation strategy is contiguous.

The main objective of this chapter is to describe some of the existing contiguous and non-contiguous allocation strategies that have been proposed in the literature [18, 24, 34, 41, 48, 49, 65, 84, 85, 97, 99] for mesh-connected multicomputers. This chapter also describes the system model assumed in this study. Such background is necessary for understanding the subsequent chapters. The remainder of this chapter is organized as follows. Section 2.2 describes the existing allocation strategies. Section 2.3 provides the system model assumed in this research. Section 2.4 outlines the list of assumptions used in this research. Section 2.5 describes the simulation tool (ProcSimity Simulator) while Section 2.6 justifies the selection of simulation as a tool of study. Finally, Section 2.7 summarises this chapter.

2.2 Related Allocation Strategies

This section provides a brief overview of some existing contiguous and non-contiguous allocation strategies that have been suggested for both the 2D and the 3D mesh-connected multicomputers.

2.2.1 Contiguous Allocation Strategies for 2D and 3D Mesh

Contiguous allocation has been extensively investigated for mesh-connected multicomputers [9, 20, 21, 27, 31, 33, 34, 38, 48, 51, 52, 65, 78, 80, 94, 99]. Most of the previous studies have focused on reducing the degrading effects of high processor fragmentation caused by contiguous allocation. Below we describe some of the well-known strategies.

***Two Dimensional Buddy System (2DBS)*:** The 2DBS allocation [48] applies to square mesh systems with power of two side lengths. Processors allocated to jobs also form square sub-meshes with power of two side lengths. If a job requests a sub-mesh of size $\alpha \times \beta$ such that $\alpha \leq \beta$, the 2DBS allocates a sub-mesh of size $s \times s$, where $s = 2^{\lceil \log_2(\max(\alpha,\beta)) \rceil}$. For example, if a job requests 2 processors it is allocated a square sub-mesh of processors with a side length of 2, resulting in 2 idle processors and an internal fragmentation of 50% as shown in Figure 2.1 above. This strategy suffers from internal and external processor fragmentation [18, 20, 77, 85, 99]. Furthermore, it cannot be used for non-square meshes [18, 77, 85].

***Frame Sliding (FS)*:** The frame sliding strategy [65] is applicable to a mesh of any size and shape. FS searches for an appropriate allocation using a set of sequenced non-overlapping processor frames (i.e., processor sub-meshes). It is assumed that an arriving job requests a

processor sub-mesh of rectangular shape. Processor frames of the same side lengths as the requested sub-mesh are searched from left to right and from bottom to top. Jumps to successive frames are by the job's width and length. The goal of searching is to find a suitable frame for allocation; i.e., all its processors are free and it is large enough to accommodate the allocation request. This process ends with either finding a suitable allocation or when all frames are scanned and no appropriate frame is found. Figure 2.3 gives the states of a 6×5 mesh and the allocation algorithm is invoked for a 3×2 request. An allocation process starts with the first free processor found starting from the lowest-leftmost corner of the sub-mesh. It can be seen from this figure that the first frame considered is not allocated because there is an allocated processor inside that frame. The request then slides horizontally by the width of the job request, which goes outside of the mesh. After that, the requested frame slides vertically to the top of the mesh by the length of the job request, but again the new frame of processors is not allocated because it contains allocated processors. This process continues, and we notice that it ends without finding a suitable frame for allocation. The allocation strategy fails to allocate a sub-mesh to the job request although one exists. A problem with this strategy is that it may not recognise free sub-meshes because the jumps are by the job's width and length [85].

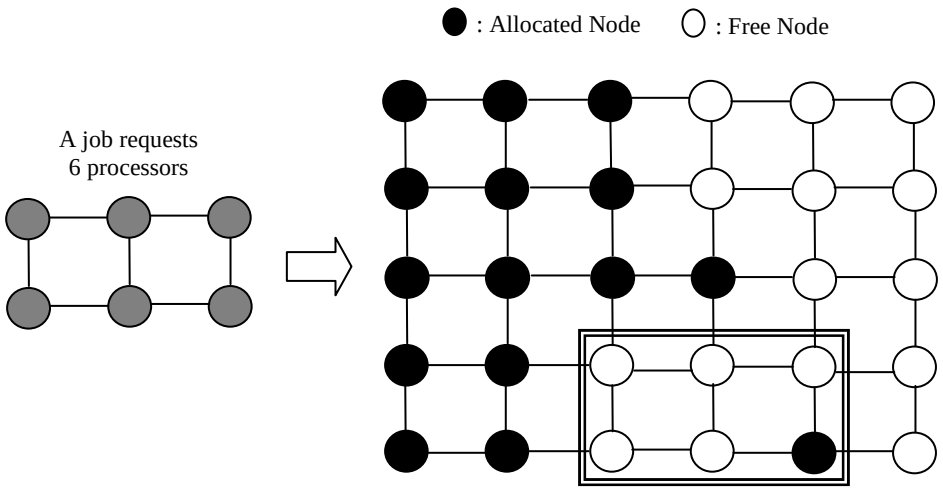


Figure 2.3: An allocation using the frame sliding strategy

***Adaptive Scan (AS)*:** This strategy [41] is an improvement of the FS strategy [65] and uses scanning instead of a sliding operation. That is, it moves a frame vertically with a stride distance of 1 processor and horizontally based on the allocated sub-meshes. Moreover, this strategy supports the re-orientation (i.e., rotation) of the allocation request when allocation fails for the requested orientation. A job that requests an $\alpha \times \beta$ sub-mesh may be allocated a $\beta \times \alpha$ sub-mesh. However, the shorter stride distance increases the allocation time and hence AS is not suitable for large meshes. For the remainder of this dissertation, the terms *rotation* and *re-orientation* will be used interchangeably.

***First Fit (FF) and Best Fit (BF) for 2D Meshes*:** The problem of missing an existing possible allocation encountered in previous strategies is solved in the FF and BF strategies [99]. The processors that can serve as base nodes for the free sub-meshes that can accommodate the current job request are represented by an array of size N, where N is the number of processors in the mesh system. In FF, the first such base is chosen as the allocation base. In BF, a base that has the largest number of busy neighbours and smallest surrounding free area is selected as the allocation base. Given a request for a 2×2 sub-mesh and the mesh shown in Figure 2.4, FF and BF allocate the sub-meshes S_1 and S_2, respectively. The FF and BF strategies [99] can detect all large-enough free sub-meshes, but they lack complete sub-mesh recognition ability in that they do not consider switching the orientation of requests. An in-depth discussion of FF and BF allocation and de-allocation algorithms can be found in [99].

First Fit (FF) and Best Fit (BF) for 3D Meshes: In these two strategies [34], the free sub-meshes are scanned and FF allocates the first sub-mesh that is large enough to hold the job, whereas BF allocates the smallest suitable sub-mesh. Simulation results have shown that these two strategies have comparable performance in terms of average turnaround time and

mean scheduling effectiveness[2]; the performance of FF is close to that of BF, therefore we only consider the FF strategy for the purpose of this study. The strategies FF and BF are not recognition-complete; an allocation request is accommodated only if there exists a large enough sub-mesh with the same orientation as the allocation request, hence they suffer from high external processor fragmentation. Bit arrays are used for the scanning of available processors. The allocation and de-allocation algorithms for the FF strategy are presented in Figures 2.5 and 2.6, respectively.

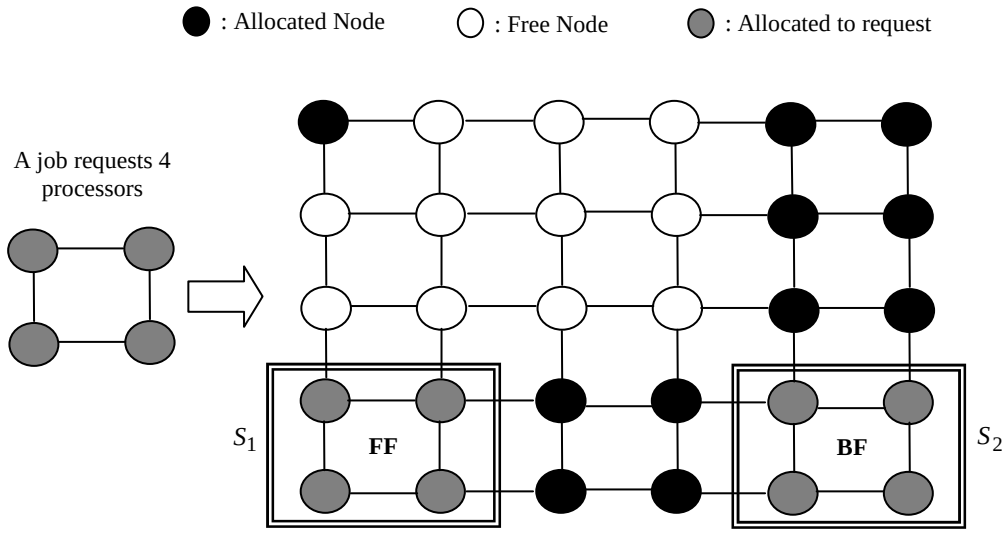


Figure 2.4: An allocation using First Fit and Best Fit strategies

Procedure FF_Allocate (α, β, γ):

{

W = Mesh Width; D = Mesh Depth; H = Mesh Height

Mesh Size = W × D ×H

Job Size = α × β × γ

int w_i, d_j, h_k, w_x, d_y, h_z

int Avail;// To determine the number of processors for an incoming job.

if (Job Size > free processors) return failure

for each w_i from 0 to W - 1

[2] The scheduling effectiveness measures the ability of an allocation algorithm to avoid processor fragmentation [38].

```
        for each d_j from 0 to D - 1
            for each h_k from 0 to H - 1
                if the node (w_i, d_j, h_k) is free then {
                        Avail = 0
                        for each w_x from w_i to w_i + α – 1 provided that w_x < W
                            for each d_y from d_j to d_j + β – 1 provided that d_y < D
                                for each h_z from h_k to h_k + γ – 1 provided that h_z < H
                                    if the node (w_x, d_y, h_z) is free then Avail++;

                        if (Avail==Job Size){
                            for each w_x from w_i to w_i + α – 1
                                for each d_y from d_j to d_j + β – 1
                                    for each h_z from h_k to h_k + γ – 1
                                        allocate the node(w_x, d_y, h_z) to the current job by
                                        setting node's ID to job ID.
                            return success.
                        }
                }
                return failure
}
```

Figure 2.5: Outline of the FF Contiguous Allocation Strategy.

```
Procedure FF_De-allocation ():
{
        jid = id of the departing job;
        For all nodes in the mesh system
        if (nodes' id == jid)
                de-allocate it.
}
```

Figure 2.6: Outline of FF de-allocation algorithm

Turning First Fit (TFF) and Turning Best Fit (TBF) for 3D Meshes: The problem of missing an existing possible allocation mentioned in FF and BF above is solved using TFF and TBF [34]. The TFF and TBF strategies [34] support the rotation of the job request. They consider all orientations of the request when needed. Let (α, β, γ) be the width, depth and

height of a sub-mesh allocation request. The six permutations (α, β, γ), (α, γ, β), (β, α, γ), (β, γ, α), (γ, α, β) and (γ, β, α) are, in turn, considered for allocation. If allocation succeeds for any of these permutations the process stops. For example, assume a free mesh (3, 3, 2) and the job requests (2, 3, 2) and (3, 2, 1) arrive in this order. The second job request cannot be accommodated until it is rotated to (1, 3, 2), as shown in Figure 2.7. Simulation results have shown that the TFF strategy can greatly improve performance in terms of average turnaround time and mean scheduling effectiveness. Changing the orientation of allocation requests can alleviate external fragmentation. Moreover, the performance of TFF is almost identical to that of TBF; therefore only the TFF strategy is considered in this research. In [34], different scheduling strategies, such as First-Come-First-Served (FCFS) and Out-of-Order[3] (OO) have been studied. The goal of OO scheduling is to avoid performance loss due to blocking associated with the head of the FCFS queue.

Allocation Algorithm for the IBM BlueGene/L: In this algorithm [97], the allocation unit is the midplane, which consists of $8 \times 8 \times 8$ processors. The goal of using this large allocation unit is to decrease the allocation overhead. The algorithm supports the rotation of the allocation request. The system is scanned for all 3D rectangular and spatially contiguous sets of free midplanes that match the shape and size of the request. This algorithm assumes that a job can utilize an integer number of midplanes. Otherwise, there is internal processor fragmentation, which can be severe as this allocation unit is rather large, hence the degradation of system utilization can be severe. Furthermore, the allocation overhead depends on the number of midplanes in the mesh system, and it increases when the number of midplanes increases.

[3] In the OO scheduling strategy, the requests in the FIFO waiting queue are considered for allocation in the order of their arrival, this process is stopped when the end of the queue is reached, or when there are no more free processors.

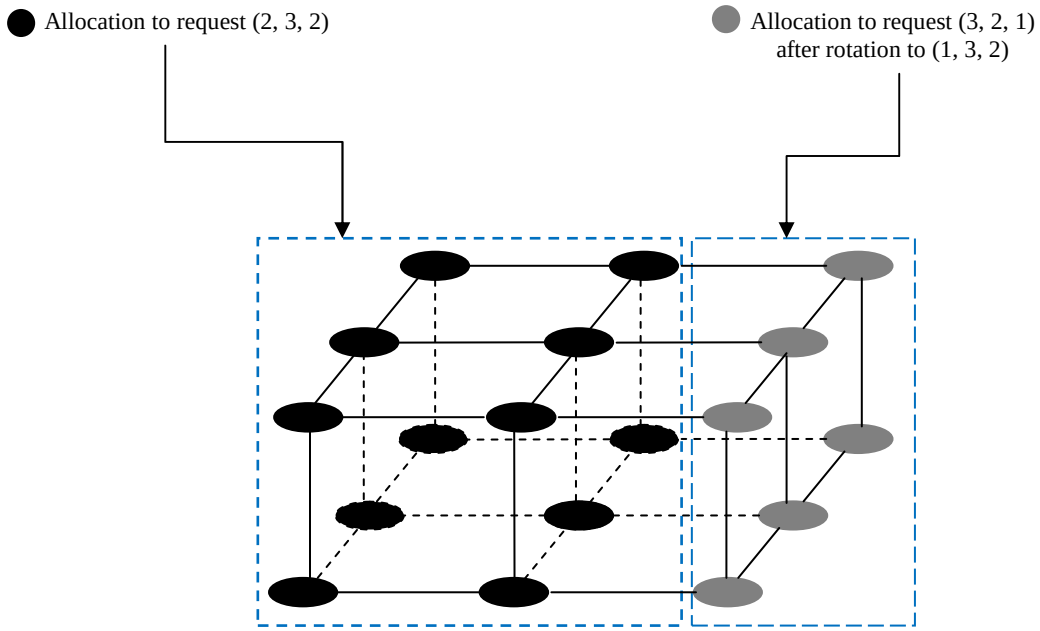


Figure 2.7: Allocation with rotation to request (2, 3, 2) followed by request (3, 2, 1)

The above allocation strategies consider only contiguous regions for the execution of a job. As a consequence, the length of the communication paths is expected to be minimized in contiguous allocation. Only messages generated by the same job are expected within a sub-mesh and therefore there is no inter-job contention in the network. On the other hand, the restriction that jobs have to be allocated to contiguous processors reduces the chance of successful allocation. It is possible that allocation fails in the contiguous allocation strategies while there is a sufficient number of free processors [18, 85], i.e., fragmentation occurs in these strategies.

2.2.2 Non-Contiguous Allocation Strategies for 2D Meshes

Advances in routing techniques such as wormhole routing [4, 29, 83], have made communication latency less sensitive to the distance between communicating nodes [2, 18, 43, 71, 72, 77]. This has made allocating a job to non-contiguous processors plausible in networks characterised by long-diameter, such as the 2D mesh. Non-contiguous allocation

allows jobs to be executed when the number of available processors is sufficient [18, 44, 49, 71, 72, 77, 85]. 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 [85]. 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.

***Paging*:** In the Paging strategy [85], the entire 2D mesh is divided into pages that are sub-meshes with equal side lengths of 2^{size_index}, where $size\_index$ is a positive integer. 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 2.8. 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:

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

where $Psize$ is the size of the page, and α and β are the side lengths of the requested sub-mesh. 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:

$$Internal\_Fragmentation = \frac{\sum_{jobs} Lost\_Processors}{\sum_{jobs} Allocated\_Processors} \quad \text{..... (2.2)}$$

where $Lost\_Processors$ is for a parallel job that requests Job_Size processors, but is allocated $Number\_of\_Allocated\_Pages$. It is calculated using:

$$Lost\_Processors = Number\_of\_Allocated\_Pages \times Psize - Job\_Size \text{ ..... (2.3)}$$

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%.

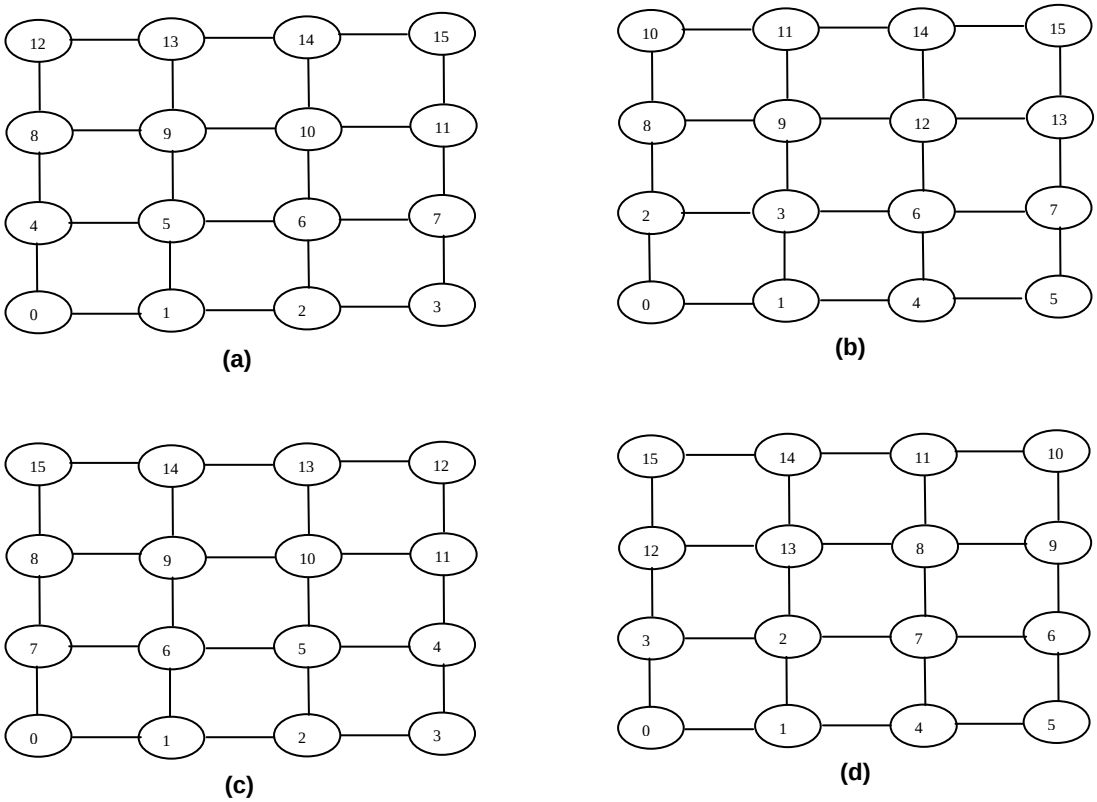


Figure 2.8: Paging(0) using different indexing schemes: (a) Row-major indexing, (b) Shuffled row-major indexing, (c) Snake-like indexing, and (d) Shuffled snake-like indexing

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 [85]. The Paging allocation and de-allocation algorithms are presented in Figures 2.9 and 2.10, respectively.

// $Page\_Side = 2^{size\_index}$; $Psize = Page\_Side \times Page\_Side$

// *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 {

$Job\_Size = \alpha \times \beta$

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

// Allocation:

*Step*1. *if* (*number of free pages* < $P_{request}$) *return failure else go to step* 2

*Step*2. *allocate the first* $P_{request}$ *pages from the list of unallocated pages to the job, setting the IDs of these pages to* ***jid****, and return success.*

} End

Figure 2.9: 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 2.10: Outline of the Paging de-allocation algorithm

***Multiple Buddy Strategy (MBS)*:** In MBS [85], 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 $2^i \times 2^i$, 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:

$$p = d_i \times 2^i \times 2^i + d_{i-1} \times 2^{i-1} \times 2^{i-1} + ... + d_0 \times 2^0 \times 2^0 \quad \text{(2.4)}$$

where the factors $d_0 ..... d_i \in \{0,1,2,3\}$. This strategy attempts to satisfy every term i in the request with d_i free processor blocks of sizes equal to $2^i \times 2^i$ 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 $2^j \times 2^j$ block are $2^{j-1} \times 2^{j-1}$ blocks. If that fails, MBS breaks the request for a $2^i \times 2^i$ block into 4 smaller requests for $2^{i-1} \times 2^{i-1}$ 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. The MBS strategy is composed of five parts: system initialization, request factoring algorithm, buddy generating algorithm, allocation algorithm, and de-allocation algorithm. The detailed operations of these parts are included in Appendix A.

***Adaptive Non-contiguous Allocation (ANCA)*:** In [18], ANCA first attempts to allocate a job contiguously. When contiguous allocation fails, it breaks a job request into two equal-sized sub-frames (i.e., sub-requests). For example, an 8×3 request is partitioned into two 4×3 sub-frames. These sub-frames are then allocated available sub-meshes, if possible. Otherwise, each of these sub-frames is broken into two equal-sized sub-frames, 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-frames, or it has repeated a specified number of times. Moreover, allocation fails if a side length of the sub-frames reaches 1, which can cause external fragmentation.

***Adaptive Scan and Multiple Buddy (AS&MB)*:** AS&MB is a hybrid strategy [49]. Firstly, it attempts to allocate a job contiguously using the adaptive scan strategy [41]. When the adaptive scan strategy fails to allocate a job request, it employs the non-contiguous allocation strategy MBS [85] for allocation. Simulation results in [44] show that the performance of AS&MB is almost identical to that of MBS [85] in terms of average response 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.

***Paging variants*:** In addition to the four indexing schemes considered in [85], the Hilbert and H-indexing space-filling curves have been proposed for ordering processors [24, 84]. 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 [24]. They have also been compared to a strategy that minimises the average pair-wise distance between the processors allocated to a request (see Gen-Algorithm in [24]). 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 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 $size\_index \geq 1$, and it increases with $size\_index$ [85]. 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 2.11. Initially, the request is factorised 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 [85] and ANCA [18]. Any processor allocation strategies like Paging variants that operate at this level of granularity (i.e., a single processor) require a long time to reach the allocation decision [97]. For large machines such as IBM BuleGene/L, allocation strategies that take a reasonable time for allocation and de-allocation operations were proposed [97]. It is to avoid low allocation granularity that the allocation unit in the IBM BlueGene/L, for example, is the midplane, which is an $8 \times 8 \times 8$ three-dimensional page [97]. 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.

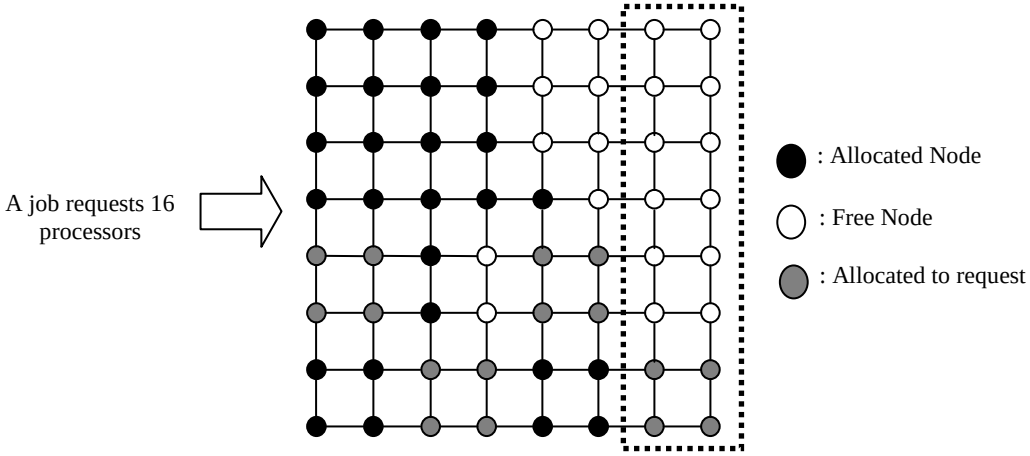


Figure 2.11: An 8 × 8 2D mesh receiving an allocation request for 16 processors in MBS strategy

2.3 System Model

The topology of the interconnection network 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 [23, 83]. The network diameter, node degree, and network degree are often used to characterize a given topology [4, 23, 29]. The diameter is the maximum value of the shortest path lengths between any two nodes. The number of links connecting a node to its neighbours is known as the node degree while the network degree is the maximum node degree in the network.

Many topologies have been proposed for parallel computers, including the hypercube [8, 43] and the mesh [4, 8, 82]. In a hypercube with d dimensions we have $N = 2^d$ nodes each of

degree d. The advantage of the hypercube topology is its small diameter. However, a major drawback of the hypercube network is its lack of scalability, which limits its use in building large-size multicomputers [8]. Among important parameters of an interconnection network of a multicomputer system are its scalability and modularity. 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 [8]. Also, the increase in the system size is expected to result in an increase in performance to the extent of the increase in size [8]. 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) (i.e., as the dimension of the hypercube is increased by one, one more links needs to be added to every node in the network). This property makes the hypercube cost prohibitive for large N [8, 83]. In addition to the changes in the node configuration, a doubling of the size is required for the regular hypercube network to expand and to remain as a hypercube [8].

Moreover, because a computer must be placed in the world we live in (a 3D space), some links in the hypercube, when the number of dimensions > 3, must be longer than others, and longer than link lengths in 2D and 3D meshes. Consequently the longer links in hypercube networks have an adverse effect on the network latency as shown in [62]. Unlike hypercube, links in 2D and 3D meshes can be of the same length, and the length is independent of the size of the mesh system. Furthermore, as the number of nodes increases in the network the average number of hops in the mesh networks, for example, increases more rapidly compared to the hypercube [62]. This allows the mesh networks to exploit the available buffer size to reduce the number of channels that a message occupies, thus reducing the blocking delays. Whereas, in the hypercube, due to the smaller average number of hops, messages occupy almost all the channels between the source and destination nodes increasing the probability of blocking, even with large buffer sizes [62]. Nevertheless, the

mesh network is able to exploit the increase in the buffer size more efficiently compared to the hypercube [62].

Motivated by the above observations, the network topology assumed in this research is the mesh interconnection network. Mesh networks are easily implemented because of the simple regular connection and small number of links per node. Due to the constant node degree, the mesh network is highly scalable. Moreover, the mesh has been widely used in practical multicomputers due to its advantages such as simplicity, scalability, structural regularity, ease of implementation, and partition-ability [8, 9, 18, 21, 27, 31, 33, 35, 51, 52, 77, 78, 85, 99].

The nodes in the mesh are connected to their immediate neighbours by bidirectional links. Each node in the mesh network consists of a processing element (PE) and a router. The PE contains a processor and some local memory. A router in an n-dimensional mesh has $2n$ input and $2n$ output channels that connect the router to its neighbouring routers. There are 2 input and 2 output channels per dimension. A router is connected to its local processor via internal channels, or ports. When each node has one pair of internal channels, it is referred to as *one-port* architecture. In this model, one internal channel is used by the processor to output messages to the network, while the other is used to input messages from the network. A crossbar switch is used to establish a connection between any of the input channels and any of the output channels. In this model, when messages destined for the local node arrive at a router on input channels, they are transmitted to the local node sequentially. The *all-port* architectural model differs from the one-port model in that a node can process (i.e., send/receive) n messages (which equals the number of ports) simultaneously. The discussion can be easily extended to the nodes situated at the corners and edges of the network.

2.3.1 Switching Method

The switching method 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. Various switching methods have been described in the literature for multicomputer networks, of which the three most important ones are *store-and-forward* [83], *virtual cut-through* [13, 29] and *wormhole switching* [13, 16, 18, 54, 83, 85].

***Store-and-forward switching*:** In store-and-forward switching, 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 routing the packet. 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: large buffer spaces are required to store entire packets and the time to transmit a message is directly proportional to the distance between the source and destination nodes [64].

***Virtual cut-through switching*:** Virtual cut-through [13, 29] has been introduced as an enhancement to store-and-forward switching in order to reduce the transmission time. In this switching method, a message header (i.e., the part of the message that contains routing information) is examined upon arrival at an intermediate node, if the next channel requested is busy; the message is entirely stored at the node at location of lead message. Otherwise, it is transmitted to the next node without buffering. 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 utilised 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 size of multicomputer systems [29, 43, 64].

***Wormhole switching*:** The disadvantage of virtual cut-through has motivated the use of its variant wormhole switching. Wormhole switching (also called wormhole routing [29, 43, 54]) has been widely used in practical multicomputers [13, 43] due to its low buffering requirement and good performance. Experimental results in [64] 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 wormhole switching, a message is divided into a sequence of fixed-size units, called *flits*. A flit typically consists of a few bytes. A message starts with a header flit that is used for message transmission and flow control, and each channel buffer needs only to hold one flit. A flit is the smallest unit of data transmission in a wormhole routing network. The *header* flit (containing 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 (i.e., blocked) 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 [4]. 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 [4].

Since wormhole routing uses pipelined transmission [29], it can perform well even in high-

diameter networks, such the mesh [29]. Many experimental machines, such as the iWARP [15] and the MIT J-machine [61]; and commercial ones including the Intel Paragon [39], the Cray T3D [67], the IBM BlueGene/L [10, 55, 97, 98], and the Cray T3E [25] have used wormhole switching. Wormhole switching is used in this research when examining the performance of the non-contiguous allocation algorithms. We have limited ourselves to wormhole switching because it has been used in the existing non-contiguous allocation strategies [44, 49, 71, 72, 77, 85].

2.3.2 Routing Algorithm

Many existing networks, including meshes, provide multiple physical paths for routing a message between any two nodes. The routing algorithm determines the path used by each message in the network. Routing algorithms are divided into two classes, *deterministic* and *adaptive*, according to their ability to modify routing paths based on dynamic network conditions [23, 54, 83]. In deterministic routing, a message always uses the same path between the source and destination; intermediate nodes are unable to redirect messages to any alternative paths. In adaptive routing, intermediate nodes can take the actual network conditions, such as the presence of congestion or failures, into account and determine accordingly to which node a message should be sent [29]. An important issue for any routing algorithm is to ensure freedom from *deadlock*s; deadlock occurs when no message can advance towards its destination because of busy channels and buffers [29, 43]. Many studies [42, 69, 91, 92] have been devoted to addressing this issue in wormhole switched interconnection networks, including meshes [42, 69, 92].

Figure 2.12 illustrates a deadlock situation where each of the 4 messages (M_1, M_2, M_3, and M_4) waits for a communication link that is held by another message, and waiting is circular. It is assumed in the figure that the messages M_1, M_2, M_3, and M_4 are destined respectively to

the nodes C, D, A, and B. The messages are in a waiting cycle, and none of them can progress. Deadlock is a disastrous state because the communication can never be completed.

Deterministic routing has been widely employed in wormhole switched interconnection networks as it offers a simple way to avoid message deadlock. This is achieved by forcing messages to visit the channels in a strict order. *Dimension-ordered* routing [13, 43, 91] is a well-known example of deterministic routing where messages cross network dimensions in a pre-defined order, reducing to zero the offset in one dimension before visiting the next. Consequently, messages always take the same path between a given pair of nodes. For mesh networks, dimension-ordered routing ensures deadlock-freedom. This type of routing is also widely known as *XY* routing when the interconnection topology is the 2D mesh [13, 16, 43, 85]. Dimension-ordered routing is used in this research when examining the performance of the non-contiguous allocation algorithms. We have limited ourselves to dimension-ordered routing because it has been used in the existing non-contiguous allocation strategies [44, 49, 71, 72, 77, 85].

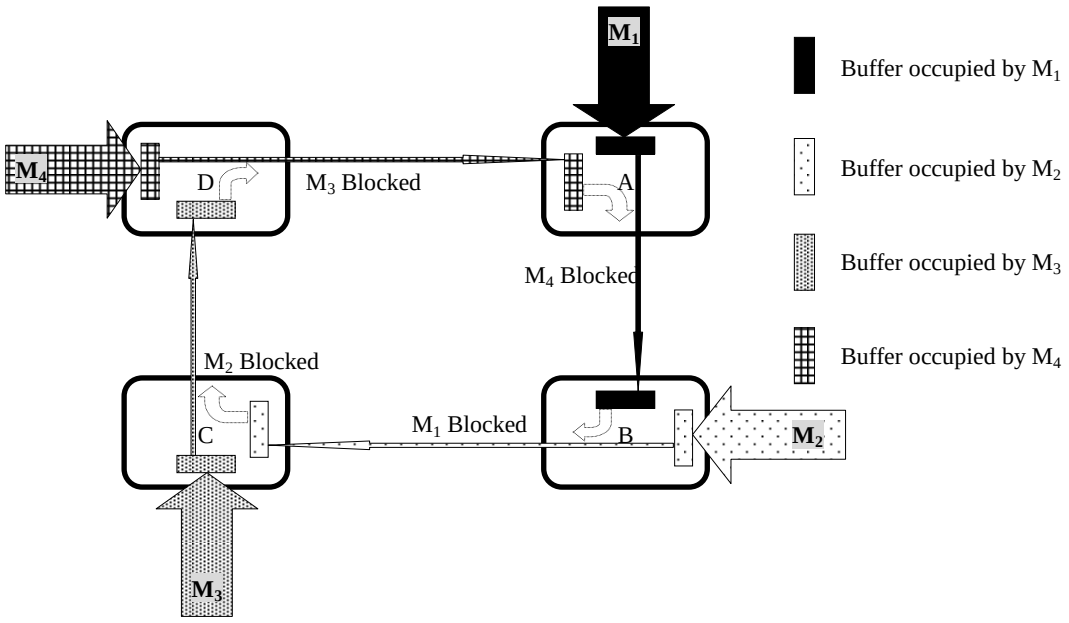


Figure 2.12: A deadlock in wormhole routing caused by 4 messages.

In contrast to deterministic routing, adaptive routing algorithms enable messages to use alternative paths to advance through the network when a communication link is congested or it has failed, for example [42, 92]. The main disadvantage of adaptive routing is the requirement for extra hardware resources, e.g., virtual channels, to deal with the problem of deadlock. A physical channel is divided into two or more virtual channels, where each virtual channel has its own queue, but shares the bandwidth of the physical channel with the other virtual channels. Virtual channels often increase hardware complexity, which can significantly reduce router speed, decreasing overall network performance [12, 30, 43]. This increase in hardware cost has motivated researchers to develop algorithms that can achieve adaptive routing without using virtual channels, leading to more efficient router implementation [1, 12, 30, 54].

2.3.3 Communication Patterns

Processors allocated to a parallel job often exchange messages with each other according to a given communication pattern [85]. When non-contiguous allocation is employed, we are interested 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 algorithms. In the *one-to-all* communication pattern, a randomly selected processor sends a message to all other processors allocated to the same job. In *all-to-all* communication, each processor allocated to a job sends a message to all other processors allocated to the same job. This communication pattern causes much message contention and is considered as the weak point for non-contiguous allocation algorithms [49]. 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 [44, 49, 85].

2.4 Assumptions

In the subsequent chapters, extensive simulation results will be presented to evaluate the performance of our allocation strategies. In this study, we make the following assumptions which have been commonly used in the literature [6, 9, 11, 18, 20, 24, 27, 31, 33, 34, 35, 38, 44, 49, 51, 52, 66, 71, 72, 73, 74, 75, 76, 77, 78, 79, 85, 94, 99]; it is worth mentioning that the last two assumptions are made when examining the performance of the non-contiguous allocation algorithms.

- The inter-arrival times of jobs are independent and follow an exponential distribution.
- Jobs are scheduled on a First-Come-First-Served (FCFS) basis, unless stated otherwise.
- The execution times of jobs are independent and follow an exponential distribution, unless stated otherwise.
- 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 is the exponential distribution, where the side lengths of the requested sub-meshes are exponentially distributed with a mean of half the side length of the entire mesh.
- Messages are transmitted inside the network using wormhole switching along with *XY* routing [2, 4, 11, 13, 29, 71, 72, 83].
- Messages are of a fixed length (i.e., a fixed number of flits). Moreover, the number of messages that are generated by a given job is exponentially distributed.

2.5 The Simulation Tool (ProcSimity Simulator)

This section introduces briefly the well-known ProcSimity simulation tool [50, 66]. ProcSimity is a discrete-event simulation tool [7, 68] that has been developed as a research tool in the area of processor allocation and job scheduling in multicomputers [50, 66]. ProcSimity was developed at the University of Oregon [66], and the development efforts of the simulator have been supported by OACIS and NSF [50]. The tool was written in the C programming language and has been extensively used for processor allocation and job scheduling in mesh-connected multicomputers [24, 33, 35, 44, 45, 50, 51, 71, 72, 73, 74, 75, 76, 77, 78, 79, 85, 86]. This is due to the fact that it is open-source and includes detailed simulation of important operations of multicomputer networks [50, 66]. It is worth noting that the simulator has been extensively validated in [66].

The overall purpose of the ProcSimity is to provide a convenient environment for performance analysis of processor allocation and scheduling algorithms. In particular, ProcSimity has been designed to investigate some of the processor allocation problems, such as fragmentation and communication overhead problems [24, 33, 35, 44, 45, 50, 51, 71, 72, 73, 74, 75, 76, 77, 78, 79, 85, 86]. The architecture modelled by ProcSimity consists of a network of processors interconnected through message routers at each node. Adjacent nodes are connected by bidirectional communication links, and messages may be routed by either store-and-forward or wormhole switching. The ProcSimity supports both the mesh and *k*-ary *n*-cube interconnection topologies with dimension-ordered routing [50, 66].

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 [50, 66]. In addition, third-party scheduling and allocation strategies can be integrated into ProcSimity. ProcSimity also

involves the specification of the simulation experiments; it supports both stochastic job streams as well as communication patterns from actual parallel applications [50, 66]. In ProcSimity, the user can specify the detailed simulation of message-passing overhead at the flit level [50, 66].

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 is selected to be executed 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 [18, 24, 33, 35, 44, 50, 51, 71, 72, 73, 74, 75, 76, 77, 78, 79, 85]. Once a job departs from the system the sub-meshes it is allocated are freed for use by another incoming job.

In ProcSimity, the overhead of allocation and de-allocation (i.e., the time that the allocation and de-allocation operations take per job) is ignored. To compare the allocation strategies in terms of the allocation overhead associated with the allocation and de-allocation operations, we measured the average actual time taken by these operations on a Pentium machine running under Windows XP. The clock cycle of the machine is 3 GHz and the RAM size is 504 MB. The per-job average allocation overhead was computed in milliseconds over enough independent runs so that the confidence level is 95% that relative errors are below 5% of the mean.

2.6 Justification of the Method of Study

In this research, extensive simulation experiments have been conducted to explore performance-related issues of processor allocation in mesh-connected multicomputers. This section discusses briefly the choice of simulation as a tool of study for the purpose of this research, justifies the adoption of ProcSimity as the preferred simulation tool, 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. In general, in addition to conducting measurements on a real practical system or testbed, there exist two techniques for system performance evaluation: analytical modelling and simulation [68]. 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. In contrast, simulation models can easily incorporate details to the desired level of accuracy in order to mimic more closely the behaviour 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 modelling. However, as we have used the ProcSimity simulator that has already been developed and extensively validated [50, 66], we have easily incorporated our suggested algorithms into the simulator. This has helped to considerably cut down the development time and debugging of the code. Most often cost, along with the ease of being able to change configurations, is the prime motivation for developing simulations for expensive systems, such as multicomputers. The processor allocation algorithms designed and analysed in this study are 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. However, the current version of ProcSimity does not support the 3D mesh network. So, we have modified the existing simulator by adding our proposed processor allocation algorithms for both the 2D and the 3D mesh-connected multicomputers. While incorporating the modifications into the simulator, special care has been taken to ensure that the algorithms implemented would function as designed and that the simulator would not exhibit unwanted side-effects; this has been accomplished through implementing one of our algorithms using another simulator [34] and comparing the outputs against those obtained by ProcSimity. Moreover, we have carried out the validation of the simulator for a number of cases and compared the performance results obtained for some-well known strategies (e.g., the FF allocation strategy) against those obtained by other researchers using another simulator [34].

It is worth mentioning that we have evaluated the performance of our processor allocation algorithms based on a real workload trace and compared the results against those obtained from our simulation study based on stochastic workloads. The results of the comparison have revealed that the conclusions reached on the performance merits of the allocation strategies when a real workload trace is used are in general compatible with those obtained when a stochastic workload is used; please see [76] for more details.

2.7 Summary

A number of allocation strategies that use space-sharing strategies have been discussed in this chapter. These strategies can be divided into two types: *contiguous* and *non-contiguous*. In contiguous allocation, processors allocated to jobs are physically contiguous and have the

same topology as the underlying system network. Doing so has the potential of eliminating inter-job communication contention as each job's messages can be routed within the set of processors allocated to that job. However, the restriction that the jobs have to be allocated contiguously reduces the chance of successful allocation, resulting in high processor fragmentation which degrades system performance.

Some researchers have suggested non-contiguous allocation as a way to reduce processor fragmentation that results from contiguous allocation. Wormhole switching techniques have also encouraged the adoption of non-contiguous allocation because it has made communication latency less sensitive to the distance between communication processors. In non-contiguous allocation, a job can execute on multiple disjoint sub-meshes rather than waiting until a single sub-mesh of free processors is available. This increases the number of possible allocations that may be considered, which can reduce processor fragmentation and improve system utilization. However, messages generated from some jobs may pass through the processors allocated to other jobs, which increases message contention inside the network. Nonetheless, lifting the contiguity condition is expected to reduce processor fragmentation and increase processor utilization substantially.

This chapter has provided the system model used in this research. It also includes an outline of assumptions that apply throughout the thesis. Finally, it contains a brief description of the simulation tool (the ProcSimity Simulator) that is used to conduct the performance evaluation of processor allocation strategies. Moreover, a brief discussion of the choice of simulation as a tool of study in this research is included.

In the subsequent chapter, we will describe a new contiguous allocation algorithm for the 3D mesh-connected multicomputers that can overcome the limitations of the existing

contiguous allocation strategies for this class of multicomputers. Our simulation results will reveal that the new algorithm manages to deliver competitive performance (i.e., low turnaround times and high system utilization) with a low allocation overhead compared to previous strategies.

Chapter 3

Turning Busy List (TBL): A New Contiguous Allocation Algorithm for Mesh-Connected Multicomputers

3.1 Introduction

In distributed-memory multicomputers, jobs are often allocated distinct *contiguous* processor sub-meshes for the duration of their execution to reduce inter-processor communication overhead [9, 20, 26, 27, 31, 33, 34, 48, 51, 52, 65, 75, 78, 94, 97, 99]. Most existing studies [9, 20, 27, 33, 48, 51, 52, 65, 99] on contiguous allocation have been carried out mostly in the context of the 2D mesh network. There has been relatively very little work on the 3D version of the mesh. Although the 2D mesh has been used in a number of parallel machines, such as the iWARP [15] and Delta Touchstone [40], most practical multicomputers, like the Cray XT3 [19, 60], MIT J-Machine [61], Cray T3D [67], IBM BlueGene/L [10, 55], and Cray T3E [25], have used the 3D mesh network as the underlying topology due to its lower diameter and average communication distance [90].

The main shortcoming of existing contiguous allocation strategies for 3D mesh-connected

multicomputers [31, 34, 94, 97] is that they achieve complete sub-mesh recognition capability but with high allocation overhead, that accounts for the time required for the allocation and de-allocation of processors to jobs. Furthermore, the time for both the allocation and de-allocation operations in the previous contiguous allocations strategies [31, 34, 94, 97] tends to grow with the system size.

Motivated by the above observations, this chapter makes the following contributions. It presents a new efficient contiguous allocation strategy that supports the rotation of job requests, referred to as Turning Busy List (TBL for short), which can identify a free sub-mesh of the requested size as long as it exists in the mesh system; The term “turning” refers to the fact that the orientation of an allocation request could be changed when no sub-mesh is available in the requested orientation (please see Section 2.2.1 in Chapter 2). The new proposed allocation algorithm without rotation is used in this chapter for comparison purposes and is referred to as Busy List (BL for short). The proposed allocation strategy relies on a new approach that maintains a list of allocated sub-meshes to determine all the regions consisting of the network nodes (i.e., processors) that cannot be used as base nodes for the requested sub-mesh. These nodes are then subtracted from the right border plane (please see Section 3.2 for the definition of right border plane) of the allocated sub-meshes to find the nodes that can be used as base nodes for the required sub-mesh size.

This chapter also conducts a performance evaluation of the contiguous allocation strategies, including our suggested strategy, in terms of the average turnaround time and mean system utilisation, as well as the allocation overhead that the allocation and de-allocation operations take per job. The results reveal that our proposed allocation strategy has a lower allocation and de-allocation time (i.e., allocation overhead) than well-known existing strategies. The simulation results show this reduction is achieved without scanting other important

performance metrics in that system performance is still as good in terms of turnaround time and system utilisation as that of existing competing strategies.

The remainder of the chapter is organised as follows. Section 3.2 provides preliminary background information that is relevant to the present study. Section 3.3 outlines the new proposed contiguous allocation algorithm for the 3D mesh network. Section 3.4 conducts a comparative performance evaluation of the new strategy against well-known existing ones. Finally, Section 3.5 concludes this chapter.

3.2 Preliminaries

The target system is a 3D mesh-connected multicomputer, where the network is referred to as $M(W, D, H)$, where W is the width of the cubic mesh, D its depth and H its height. Each processor is denoted by a coordinate triple (x, y, z), where $0 \le x < W$, $0 \le y < D$ and $0 \le z < H$ [78]. A processor is connected by bidirectional communication links to its neighbour processors, as depicted in Figure 3.1. The figure shows an example of a $4 \times 2 \times 2$ 3D mesh, where the allocated processors are denoted by shaded circles, while the free processors are denoted by white circles. We assume that a parallel job requests the allocation of a 3D sub-mesh $S(w, d, h)$ of width $w \le W$, depth $d \le D$ and height $h \le H$. The following definitions have been adopted from [27, 77, 78].

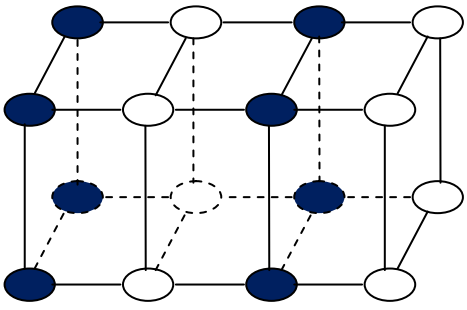


Figure 3.1: An example of a $4 \times 2 \times 2$ 3D mesh

***Definition* 1:** *A sub-mesh* $S(w,d,h)$ *of width* w, *depth* d, *and height* h, *where* $0 < w \le W$, $0 < d \le D$ *and* $0 < h \le H$ *is specified by the coordinates* (x,y,z,x',y',z'), *where* (x,y,z) *are the coordinates of the base of the sub-mesh allocated to a parallel job and* (x',y',z') *are the coordinates of its end, as shown in Figure 3.2.*

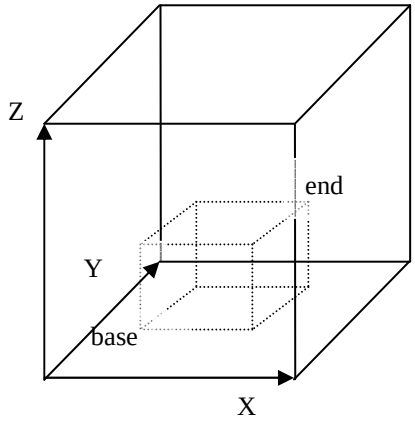


Figure 3.2: A sub-mesh inside the 3D mesh.

***Definition* 2:** *The size of* $S(w,d,h)$ *is* $w \times d \times h$ *processors.*

***Definition* 3:** *An allocated sub-mesh is one whose processors are all allocated to a parallel job.*

***Definition* 4:** *A free sub-mesh is one whose processors are all unallocated.*

***Definition* 5:** *A suitable sub-mesh* $S(w,d,h)$ *is a free sub-mesh that satisfies the conditions:* $w \ge \alpha$, $d \ge \beta$ *and* $h \ge \gamma$ *assuming that the allocation of* $S(\alpha,\beta,\gamma)$ *is requested.*

Definition 6: *A list of all sub-meshes that are currently allocated to jobs and are not available for allocation to other jobs is called busy list.*

Definition 7: *A prohibited region is a region consisting of nodes that cannot be used as base nodes for the requested sub-mesh. The prohibited region of job* $J(\alpha \times \beta \times \gamma)$ *with respect to an allocated sub-mesh* $S(x_1,y_1,z_1,x_2,y_2,z_2)$ *is defined as the sub-mesh represented by the address* (x',y',z',x_2,y_2,z_2), *where* $x' = \max(x_1 - \alpha + 1, 0)$, $y' = \max(y_1 - \beta + 1, 0)$ *and* $z' = \max(z_1 - \gamma + 1, 0)$. *For example, if a job* J *requests the*

allocation of a sub-mesh of size $2 \times 2 \times 2$*, the prohibited region of* $J(2\times2\times2)$ *with respect to the allocated sub-mesh (*1, 1, 0, 2, 2, 1*), is the sub-mesh (*0, 0, 0, 2, 2, 1*).*

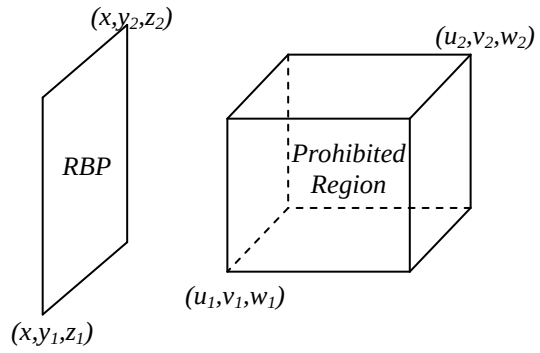
Definition 8: *The three sub-meshes* $(W-\alpha+1,0,0,W-1,D-1,H-1)$, $(0,D-\beta+1,0,W-1,D-1,H-1)$, *and* $(0,0,H-\gamma+1,W-1,D-1,H-1)$ *are automatically not available for accommodating the base node of a free* $\alpha\times\beta\times\gamma$ *sub-mesh for* $J(\alpha\times\beta\times\gamma)$, *whether the nodes in these sub-meshes are free or not; otherwise the sub-mesh would grow out of the corresponding mesh boundary plane (rightmost, deepest and highest planes) of* $M(W,D,H)$. *These three sub-meshes are called automatic prohibited regions of* $J(\alpha\times\beta\times\gamma)$ *and must always be excluded during the sub-mesh allocation process.*

Definition 9: *The Right Border Plane (RBP) of a sub-mesh* $S(x_1,y_1,z_1,x_2,y_2,z_2)$ *with respect to a job* $J(\alpha\times\beta\times\gamma)$ *is defined as the collection of nodes with address* (x_2+1,y',z') *where* $\max(y_1-\beta+1,0)\le y'\le y_2$ *and* $\max(z_1-\gamma+1,0)\le z'\le z_2$. *A RBP of sub-mesh* S *is a plane located just off the right boundary of* S.

3.3 The Proposed Turning Busy List Allocation Strategy (TBL)

The proposed TBL allocation strategy is based on maintaining a list of allocated sub-meshes; referred hereafter as the busy list. The list is scanned to determine all prohibited regions. All prohibited regions that result from the allocated sub-meshes are subtracted from each RBP of the allocated sub-meshes to determine the nodes that can be used as base nodes for the required sub-mesh size. A job $J(\alpha\times\beta\times\gamma)$ is allocatable if there exists at least one node that does not belong to any of the prohibited regions and the three automatic prohibited regions of $J(\alpha\times\beta\times\gamma)$. Figure 3.3 shows all possible cases for subtracting prohibited regions from a RBP; please see Appendix B where the figures are provided for each case.

The allocated sub-meshes in the busy list are sorted in the decreasing order of the third coordinates of their upper right corner node (i.e., end node); so that the number of subtraction operations required can be reduced. The algorithm that is used to detect the base nodes for any allocation request is formally presented in Figure 3.4, and the new proposed allocation algorithm is outlined in Figure 3.5. For the illustration, we assume that there is a hypothetical allocated sub-mesh b_0 with address $(-1,0,0,-1, D-1, H-1)$ at the head of the busy list. The RBP of the hypothetical allocated sub-mesh is the left boundary plane of the mesh. A list, *RBP _ Nodes* contains a plane if the nodes of the plane are available for allocation to the job $J(\alpha \times \beta \times \gamma)$ selected for execution.



3.3.1 ((x< u1)||(x> u2)||(z2< w1)||(z1> w2)||(y2< v1)||(y1> v2))
In this case the result is RBP itself.

3.3.2 (u1≤x≤u2)&&(v1≤y1≤v2)&&(v1≤y2≤v2)&&(w1≤z2≤w2)&&(z1<w1)
RBP (x, y1, z1, x, y2, w1-1)

3.3.3 (u1≤x≤u2)&&(v1≤y1≤v2) && (y2>v2)&&(w1≤z2≤w2)&&(z1<w1)
RBP1 (x, y1, z1, x, y2, w1-1); RBP2 (x, v2+1, w1, x, y2, z2)

3.3.4 (u1≤x≤u2)&&(v1≤y2≤v2)&&(y1<v1)&&(w1≤z2≤w2)&&(z1<w1)
RBP1 (x, y1, z1, x, y2, w1-1); RBP2 (x, y1, w1, x, v1-1, z2)

3.3.5 (u1≤x≤u2)&&(v1≤y1≤v2)&&(v1≤y2≤v2)&&(w1≤z1≤w2)&&(z2>w2)
RBP (x, y1, w2+1, x, y2, z2)

3.3.6 (u1≤x≤u2)&&(v1≤y1≤v2)&&(y2>v2)&&(w1≤z1≤w2)&&(z2>w2)
RBP1 (x, v2+1, z1, x, y2, w2); RBP2 (x, y1, w2+1, x, y2, z2)

3.3.7 (u1≤x≤u2)&&(v1≤y2≤v2)&&(y1<v1)&&(w1≤z1≤w2)&&(z2>w2)
RBP1 (x, y1, z1, x, v1-1, w2); RBP2 (x, y1, w2+1, x, y2, z2)

3.3.8 (u1≤x≤u2)&&(v1≤y1≤v2)&&(v1≤y2≤v2)&&(z1<w1)&&(z2>w2)
RBP1 (x, y1, z1, x, y2, w1-1); RBP2 (x, y1, w2+1, x, y2, z2)

3.3.9 $(u_1 \le x \le u_2) \&\& (v_1 \le y_1 \le v_2) \&\& (y_2 > v_2) \&\& (z_1 < w_1) \&\& (z_2 > w_2)$

RBP1 $(x, y_1, z_1, x, v_2, w_1\text{-}1)$*; RBP2* $(x, v_2\text{+}1, z_1, x, y_2, z_2)$*; RBP3* $(x, y_1, w_2\text{+}1, x, v_2, z_2)$

3.3.10 $(u_1 \le x \le u_2) \&\& (v_1 \le y_2 \le v_2) \&\& (y_1 < v_1) \&\& (z_1 < w_1) \&\& (z_2 > w_2)$

RBP1 $(x, y_1, z_1, x, v_1\text{-}1, z_2)$*; RBP2* $(x, v_1, z_1, x, y_2, w_1\text{-}1)$*; RBP3* $(x, v_1, w_2\text{+}1, x, y_2, z_2)$

3.3.11 $(u_1 \le x \le u_2) \&\& (y_2 > v_2) \&\& (y_1 < v_1) \&\& (z_1 < w_1) \&\& (z_2 > w_2)$

RBP1 $(x, y_1, z_1, x, v_1\text{-}1, z_2)$*; RBP2* $(x, v_2\text{+}1, z_1, x, y_2, z_2)$*; RBP3* $(x, v_1, z_1, x, v_2, w_1\text{-}1)$

RBP4 $(x, v_1, w_2\text{+}1, x, v_2, z_2)$

3.3.12 $(u_1 \le x \le u_2) \&\& (y_2 > v_2) \&\& (y_1 < v_1) \&\& (z_1 \ge w_1) \&\& (z_2 \le w_2)$

RBP1 $(x, y_1, z_1, x, v_1\text{-}1, z_2)$*; RBP2* $(x, v_2\text{+}1, z_1, x, y_2, z_2)$

3.3.13 $(u_1 \le x \le u_2) \&\& (y_2 > v_2) \&\& (y_1 < v_1) \&\& (z_1 < w_1) \&\& (w_1 \le z_2 \le w_2)$

RBP1 $(x, y_1, z_1, x, v_1\text{-}1, z_2)$*; RBP2* $(x, v_2\text{+}1, z_1, x, y_2, z_2)$*; RBP3* $(x, v_1, z_1, x, v_2, w_1\text{-}1)$

3.3.14 $(u_1 \le x \le u_2) \&\& (y_2 > v_2) \&\& (y_1 < v_1) \&\& (z_2 > w_2) \&\& (w_1 \le z_1 \le w_2)$

RBP1 $(x, y_1, z_1, x, v_1\text{-}1, z_2)$*; RBP2* $(x, v_2\text{+}1, z_1, x, y_2, z_2)$*; RBP3* $(x, v_1, w_2\text{+}1, x, v_2, z_2)$

3.3.15 $(u_1 \le x \le u_2) \&\& (v_1 \le y_1 \le v_2)\ \&\& (v_1 \le y_2 \le v_2) \&\& (w_1 \le z_1 \le w_2) \&\& (w_1 \le z_2 \le w_2)$

No RBP in this case.

3.3.16 $(u_1 \le x \le u_2) \&\& (v_1 \le y_1 \le v_2) \&\& (y_2 > v_2) \&\& (w_1 \le z_1 \le w_2) \&\& (w_1 \le z_2 \le w_2)$

RBP $(x, v_2\text{+}1, z_1, x, y_2, z_2)$

3.3.17 $(u_1 \le x \le u_2) \&\& (v_1 \le y_2 \le v_2) \&\& (y_1 < v_1) \&\& (w_1 \le z_1 \le w_2) \&\& (w_1 \le z_2 \le w_2)$

RBP $(x, y_1, z_1, x, v_1\text{-}1, z_2)$

Figure 3.3: All possible cases for subtracting a prohibited region from a right border plane.

Procedure Detect (α, β, γ):

Begin *{*

{Mesh M(W, D, H); incoming job J requests for an $\alpha \times \beta \times \gamma$ *free sub-mesh;*

Busy List $B = \{b_0, b_1, b_2, \ldots\ldots, b_m\}$ *where* b_0 *is a hypothetical allocated sub-mesh and* $b_i, 1 \le i \le m$*, are the m already allocated sub-meshes; Both sub-meshes* $(W\text{–}\alpha\text{+}1, 0, 0, W\text{–}1, D\text{–}1, H\text{-}1)$*,* $(0, D\text{-}\beta\text{+}1, 0, W\text{–}1, D\text{–}1, H\text{–}1)$*, and* $(0,0,H\text{-}\gamma\text{+}1,W\text{-}1,D\text{-}1,H\text{-}1)$ *are automatic prohibited regions and automatically not available for accommodating the base node of a free* $\alpha \times \beta \times \gamma$ *sub-mesh for J.}*

Step 1. RBP_Nodes←NULL.

Step 2. for each allocated sub-mesh $b_i\,(x_1, y_1, z_1, x_2, y_2, z_2)$ *from i = 0 to m*

Step 2.1. Construct RBP of b_i*, denoted as* $RBP_i = (x_r, y_{r1}, z_{r1}, x_r, y_{r2}, z_{r2})$*, with respect to* $J(\alpha \times \beta \times \gamma)$*, where* $x_r = x_2\text{+}1$*,* $y_{r1} = max(y_1\text{-}\beta\text{+}1, 0)$*,* $z_{r1} = max(z_1\text{-}\gamma\text{+}1, 0)$*,* $y_{r2} = y_2$ *and* $z_{r2} = z_2$*.*

Step 2.2. if RBP_i *is within any automatic prohibited region then goto Step2.*

Step 2.3. for each allocated sub-mesh $b_j\,(x_1, y_1, z_1, x_2, y_2, z_2)$ *from j =* 1 *to m*

Construct prohibited region of J with respect to b_j*, denoted as* $P_j = (x_{p1}, y_{p1}, z_{p1}, x_{p2}, y_{p2}, z_{p2})$ *where* $x_{p1} = max(x_1\text{-}\alpha\text{+}1, 0)$*,* $y_{p1} = max(y_1\text{-}\beta\text{+}1, 0)$*,* $z_{p1} = max(z_1\text{-}\gamma\text{+}1, 0)$*,* $x_{p2} = x_2$*,* $y_{p2} = y_2$ *and* $z_{p2} = z_2$*.*

Subtract P_j from RBP_i as follows:

Determine the case to which the subtraction belongs by comparing the coordinates of RBP_i and P_j as shown in Figure 3.3.

Switch (subtraction case)

{

case (1): if ($z_{r1} > z_{p2}$) then

begin

add the RBP in Figure 3.3.1 to RBP_Nodes.

goto Step 2.

end

break.

case (2): adjust RBP_i as shown in Figure 3.3.2; break.

case (3): adjust RBP_i as shown in Figure 3.3.3; break.

case (4): adjust RBP_i as shown in Figure 3.3.4; break.

case (5): add the whole RBP in Figure 3.3.5 to RBP_Nodes; goto Step 2.

case (6): add RBP($x, y_1, w_2+1, x, y_2, z_2$) in Figure 3.3.6 to RBP_Nodes
adjust RBP_i as shown in Figure 3.3.6; break.

case (7): add RBP($x, y_1, w_2+1, x, y_2, z_2$) in Figure 3.3.7 to RBP_Nodes
adjust RBP_i as shown in Figure 3.3.7; break.

case (8): add RBP($x, y_1, w_2+1, x, y_2, z_2$) in Figure 3.3.8 to RBP_Nodes
adjust RBP_i as shown in Figure 3.3.8; break.

case (9): add RBP($x, y_1, w_2+1, x, v_2, z_2$) in Figure 3.3.9 to RBP_Nodes
adjust RBP_i as shown in Figure 3.3.9; break.

case (10): add RBP($x, v_1, w_2+1, x, y_2, z_2$) in Figure 3.3.10 to RBP_Nodes
adjust RBP_i as shown in Figure 3.3.10; break.

case (11): add RBP($x, v_1, w_2+1, x, v_2, z_2$) in Figure 3.3.11 to RBP_Nodes
adjust RBP_i as shown in Figure 3.3.11; break.

case (12): adjust RBP_i as shown in Figure 3.3.12; break.

case (13): adjust RBP_i as shown in Figure 3.3.13; break.

case (14): add RBP($x, v_1, w_2+1, x, v_2, z_2$) in Figure 3.3.14 to RBP_Nodes
adjust RBP_i as shown in Figure 3.3.14; break.

case (15): go to Step 2.

case (16): adjust RBP_i as shown in Figure 3.3.16; break.

case (17): adjust RBP_i as shown in Figure 3.3.17; break.

}

goto Step 2.3.

TBL_Allocate(RBP_Nodes, α, β, γ)

} ***End.***

Figure 3.4: Outline of the Detect Procedure in the proposed Contiguous Allocation Strategy.

Procedure TBL_Allocate (RBP_Nodes, α, β, γ):
Begin *{*
int botx, boty, botz;
botx=RBP_Nodes.botx;
boty=RBP_Nodes.boty;
botz=RBP_Nodes.botz;

Add the sub-mesh represented by the address (botx, boty, botz, botx + α -1, boty + β – 1, botz + γ – 1) to the busy list by setting sub-mesh's ID to the job ID.
*}**End.***

Figure 3.5: Outline of the proposed Contiguous Allocation Strategy

Example*:*

To show the operation of the our allocation algorithm let us consider an example where we assume the mesh is free, and three allocation requests for the sub-meshes 2 × 4 × 4, 2 × 1 × 2 and 1 × 2 × 1 arrive in this order. Figure 3.6 illustrates the states of the processors of a 4 × 4 × 4 mesh. The request 2 × 4 × 4 is allocated the sub-mesh (0, 0, 0, 1, 3, 3), then the allocation algorithm is invoked for the 2 × 1 × 2 request. The busy list contains the allocated sub-meshes b_0:(-1, 0, 0, -1, 3, 3) and b_1:(0, 0, 0, 1, 3, 3), and the first RBP (RBP for the hypothetical allocated sub-mesh b_0) is calculated for this request, resulting in (0, 0, 0, 0, 3, 3). The automatic prohibited regions (3, 0, 0, 3, 3, 3), (0, 4, 0, 3, 3, 3), and (0, 0, 3, 3, 3, 3), with respect to the second allocation request, are subtracted from the first RBP, resulting in the plane (0, 0, 0, 0, 3, 2). Then the prohibited region of the allocated sub-mesh b_1:(0, 0, 0, 1, 3, 3) with respect to the second allocation request is calculated, resulting in the (0, 0, 0, 1, 3, 3) prohibited region, which when subtracted from the plane (0, 0, 0, 0, 3, 2) results in the NILL value, implying that no node is available for the job request up to this point. Then, the RBP of the allocated sub-mesh b_1:(0, 0, 0, 1, 3, 3) is calculated, resulting in (2, 0, 0, 2, 3, 3). Again the automatic prohibited regions with respect to the second allocation request are

subtracted from this new RBP, resulting in (2, 0, 0, 2, 3, 2), and the subtraction of the prohibited region of the allocated sub-mesh b_1 from (2, 0, 0, 2, 3, 2) results in (2, 0, 0, 2, 3, 2). Now, any node on the plane (2, 0, 0, 2, 3, 2) can be used as base node for the second allocation request. In this example, (2, 0, 0, 2, 0, 0) is used as base node for the second request and the sub-mesh (2, 0, 0, 3, 0, 1) is allocated to this request, resulting in the following busy list: $\{b_0$:(-1, 0, 0, -1, 3, 3), b_1:(0, 0, 0, 1, 3, 3), b_2:(2, 0, 0, 3, 0, 1)$\}$. The same procedure is repeated for the third request, and the sub-meshes allocated to the three requests are denoted by the black circles, shaded circles and dotted circles, respectively.

In the de-allocation operation, an allocated sub-mesh is de-allocated by removing its corresponding entry from the busy list. The operation of the de-allocation algorithm is presented in Figure 3.7.

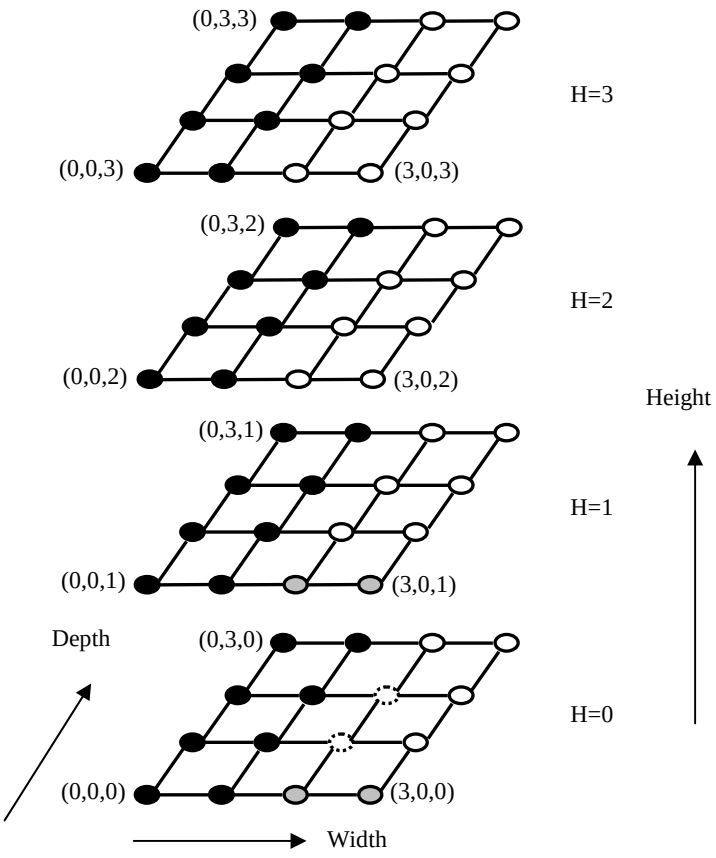


Figure 3.6: Allocation Example

```
Procedure TBL_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 3.7: Outline of the proposed de-allocation algorithm

3.4 Performance Evaluation

In this section, the results from simulations that have been carried out to evaluate the performance of the proposed allocation algorithm are presented and compared against those of the existing strategies First Fit (FF) and Turning First Fit (TFF) [34]. According to [31, 34, 94, 99], the FF strategy allocates an incoming job to the first available sub-mesh that is found but it does not permit the orientation of the allocation request. It has been revealed in [34] that the TFF strategy improves the performance by considering all orientations of the request when needed. It is worth noting that switching request orientation has been used in [31, 34, 94]. FF and TFF strategies have been selected because they have been shown in [34] to perform well compared to other existing strategies. The FF and TFF strategies have been discussed in detail in Chapter 2 (please see Section 2.2.1).

3.4.1 Simulation Results

Extensive simulation experiments have been carried out for various system loads and system sizes to compare the performance of the proposed allocation strategy against well-known FF contiguous allocation strategy [34], with and without change of request orientation. We have implemented the proposed allocation and de-allocation algorithms, including the busy list

routines, in the C language, and integrated the software into ProcSimity; simulation tool that is widely used for processor allocation and job scheduling in parallel systems [50, 66].

The target mesh is a cube with width W, depth D and height H. Jobs are assumed to have exponential inter-arrival times. They are served on First-Come-First-Served (FCFS) basis to preserve fairness [33, 51, 52, 93]. We limit ourselves to FCFS scheduling because our main purpose here is to compare the allocation strategies. The execution times are assumed to be exponentially distributed with a mean of one time unit [6, 11, 33, 34, 74, 78, 85]. The time units are simulation time units, measured by floating point numbers, NOT hours, minutes, or seconds [66], where the numbers generated by the simulator, for some of the system parameters such as jobs' execution times, are real numbers. Two distributions are used to generate the width, depth and height of job requests. The first is the uniform distribution over the range from 1 to the mesh side length, where the width, depth and height of the job requests are generated independently. The second is the exponential distribution, where the width, depth and height of the job requests are exponentially distributed with a mean of half the side length of the entire mesh; the width, depth, and height of the job requests are rounded to the integer values using floor function and bounded by the dimensions of the mesh. The exponential distribution represents the case where most jobs are small relative to the size of the mesh system. These distributions have often been used in the literature [9, 11, 20, 27, 33, 34, 38, 51, 52, 77, 85, 94, 99]. Simulation parameters are illustrated in Table 3.1. It is worth noting that most of the values of these parameters have been adopted in the literature [9, 11, 20, 27, 33, 34, 38, 51, 52, 77, 85, 94, 99].

Table 3.1: The System Parameters Used in the Simulation Experiments

Simulator Parameter	Values
Dimensions of the Mesh Architecture	8 × 8 × 8, 10 × 10 × 10, and 12 × 12 × 12

Allocation Strategy	TBL, BL, TFF, and FF
Scheduling Strategy	FCFS
Job Size Distribution	Uniform: Job widths, depths, and heights are uniformly distributed over the range from 1 to the mesh side lengths.
	Exponential: Job widths, depths, and heights are exponentially distributed with a mean of half the side length of the entire mesh.
Execution Time Distribution	Exponential with a mean of one time unit.
Inter-arrival Time	Exponential with different values for the mean. The values are determined through experimentation with the simulator, ranged from lower values to higher values.
Number of Runs	The number of runs should be enough so that the confidence level is 95% that relative errors are below 5% of the means. The number of runs ranged from dozens to thousands.
Number of Jobs per Run	1000

Each simulation run consists of 1000 completed jobs. Simulation results are averaged over enough independent runs so that the confidence level is 95% that relative errors are below 5% of the means [7]. The method used to calculate confidence intervals is called batch means analysis [4, 66]. 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 3.2, which outlines the results depicted in Figure 3.8 for the load 5.8 jobs/time unit. However, as in existing studies [9, 11, 20, 27, 33, 34, 38, 51, 52, 77, 85, 94, 99], only the grand mean is shown in our figures. In most cases the error bars have been found to be quite small; the error bars have not been included in all the figures for

the sake of clarity and tidiness.

The main performance parameters observed are the *average turnaround time* of jobs, *mean system utilisation* and *average allocation overhead*. The turnaround time is the time that a parallel job spends in the mesh from arrival to departure. The utilisation is the percentage of processors that are utilized over time. The allocation overhead is the time that the allocation algorithm takes for allocation and de-allocation operations per job (i.e., It is the time a job at the head of the waiting queue takes to be allocated and de-allocated). The allocation overhead that is incurred for detecting the availability of a free sub-mesh for an incoming job request and de-allocating it is the realistic time. We recognize that these results are implementation dependent, but the trends shown by the results help to indicate the main features of the strategies. The important independent variable in the simulation is the *system load*. It is defined as the inverse of the mean inter-arrival time of jobs. Its range of values from low to heavy loads has been determined through experimentation with the simulator allowing each allocation strategy to reach its upper limits of utilisation. 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.

Table 3.2: The mean (i.e., mean turnaround time of job), 95% confidence interval, and relative error for the results shown in Figure 3.8 for the load 5.8 jobs/time unit

Algorithm	TBL	TFF	BL	FF
95% Confidence Interval	[95.87-97.28]	[95.58-97.59]	[158.85-160.06]	[156.03-158.43]
Mean (time unit)	96.580111	96.586394	159.457505	157.225758
Relative Error	0.007	0.01	0.004	0.008

Turnaround Time:

In Figures 3.8 and 3.9, the average turnaround time of jobs is plotted against the system load for both job size distributions considered in this research. It can be seen in the figures that

the strategies with rotation (TBL and TFF) have almost identical performance, and that they are superior to all other strategies. They are followed, in order, by the strategies BL and FF respectively. When compared to TBL and TFF in Figure 3.8, for example, BL increases the average turnaround times by about 160% and 65% for the loads 3.4 and 5.8 jobs/time unit, respectively. In Figure 3.9, the increases are by about 1017% and 143% for the loads 5.8 and 12.2 jobs/time unit, respectively. It can also be seen in the figures that the average turnaround times of the strategies that depend on a list of allocated sub-meshes for both allocation and de-allocation (as in TBL and BL) is very close to that of the strategies that depend on the number of processors in the mesh system (as in TFF and FF). For example, the average turnaround time of TBL is close to that of TFF and the average turnaround time of BL is close to that of FF. As has been reported above, the average turnaround time of the strategies with rotation (as in TBL and TFF) is substantially superior to the strategies without rotation (as in BL and FF) because it is highly likely that a suitable contiguous sub-mesh is available for allocation to a job when request rotation is allowed. Experiments that use large mesh system sizes ($10 \times 10 \times 10$ and $12 \times 12 \times 12$) have been also conducted. Their results lead to the same conclusion about the relative performance of the allocation strategies (please see Section 3.4.1.1).

Utilisation:

In Figures 3.10 and 3.11, the mean system utilisation of the contiguous allocation strategies is plotted against the system loads for the uniform and exponential job size distributions. The results reveal that switching request orientation improves performance substantially. This is indicated by the largely superior mean system utilisation of the allocation strategies that can switch the orientation of allocation requests (as in TBL and TFF) when they are compared to the allocation strategies without rotation (as in BL and FF). The allocation strategies TBL and TFF have comparable performance, and they are superior to the BL and

FF allocation strategies. This is because the rotation of the allocation request increases the probability of its allocation, which in turn improves system utilization. For both job size distributions, the allocation strategies with rotation TBL and TFF achieve system utilisation of 47% under the exponential distribution and 49% under uniform distribution, but the allocation strategies without rotation BL and FF cannot exceed 37% utilisation. Higher system utilisation is achievable under heavy loads because the waiting queue is filled very early, allowing each allocation strategy to reach its upper limits of utilisation.

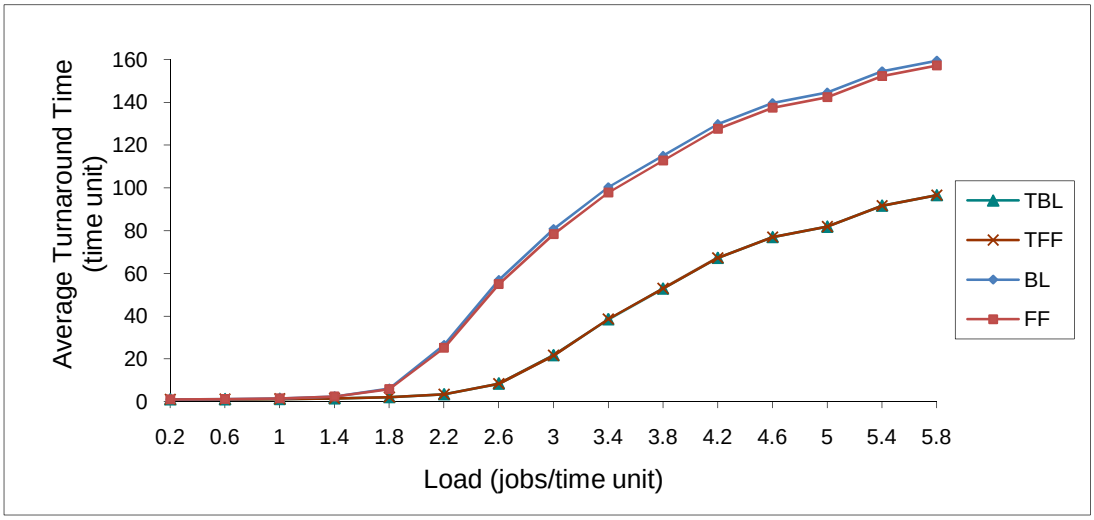


Figure 3.8: Average turnaround time vs. system load for the contiguous allocation strategies (BL, FF, TBL, TFF) and the uniform side lengths distribution in an 8 × 8 × 8 mesh.

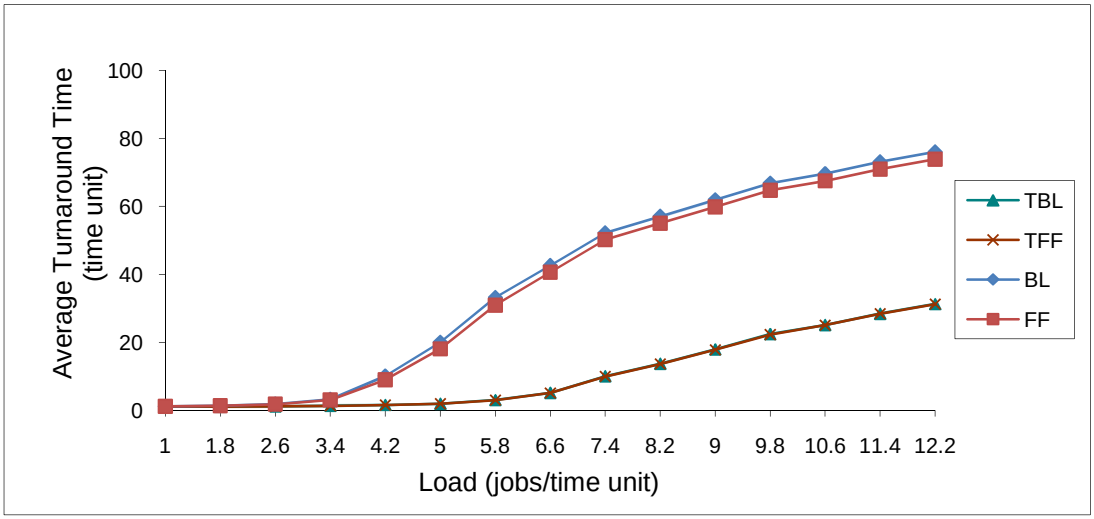


Figure 3.9: Average turnaround time vs. system load for the contiguous allocation strategies (BL, FF, TBL, TFF) and the exponential side lengths distribution in an 8 × 8 × 8 mesh.

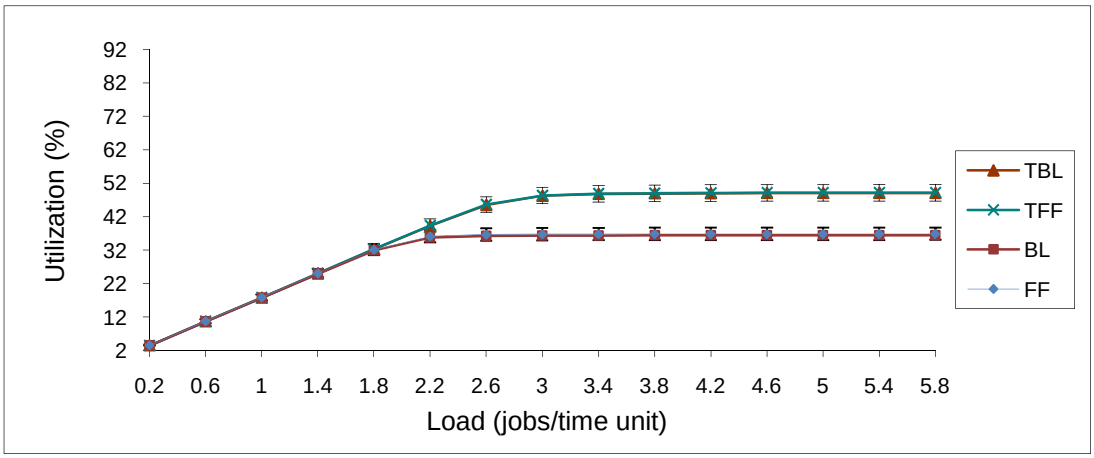


Figure 3.10: Mean System utilisation for the contiguous allocation strategies (BL, FF, TBL, TFF) and the uniform side lengths distribution in an 8 × 8 × 8 mesh.

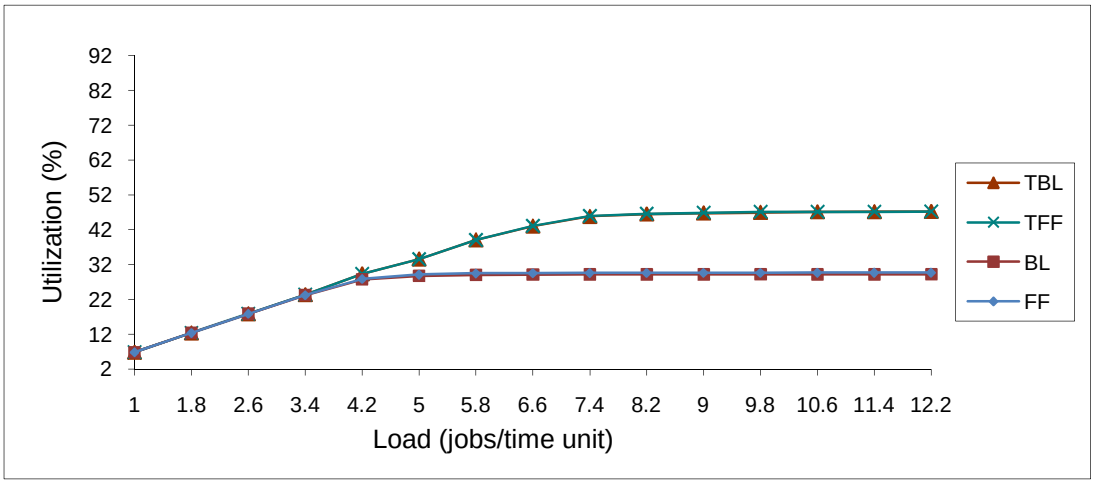


Figure 3.11: Mean System utilisation for the contiguous allocation strategies (BL, FF, TBL, TFF) and the exponential side lengths distribution in an 8 × 8 × 8 mesh.

Number of Allocated Sub-meshes (m):

In Figures 3.12~3.15, the average number of allocated sub-meshes (m) in the strategies that depend on a list of allocated sub-meshes for both allocation and de-allocation (TBL and BL) is plotted against the system load. Different mesh sizes (8 × 8 × 8, 10 × 10 × 10, and 12 × 12 × 12) are considered under both the uniform and exponential job size distributions. As expected, the average number of allocated sub-meshes is largest when the side lengths follow the exponential distribution. This is because the average sizes of jobs are smallest in this case. Moreover, the average number of allocated sub-meshes is lower than the number

of processors in the mesh system (n) for both job size distributions. It can be seen in the figures that m is often less sensitive with n. It can also be noticed that the average number of allocated sub-meshes for the strategy that use the rotation of the allocation request TBL is a little bit higher than that of the BL strategy which does not use the rotation of the allocation request. This is because it is highly likely that a suitable contiguous sub-mesh is available for allocation to a job when the request orientation is allowed, which in turn increases the number of allocated sub-meshes in the busy list. In Figures 3.12 and 3.13, for example, the average number of allocated sub-meshes of BL for all mesh sizes is 74% of that of TBL when the job arrival rate is 5.8 jobs/time unit.

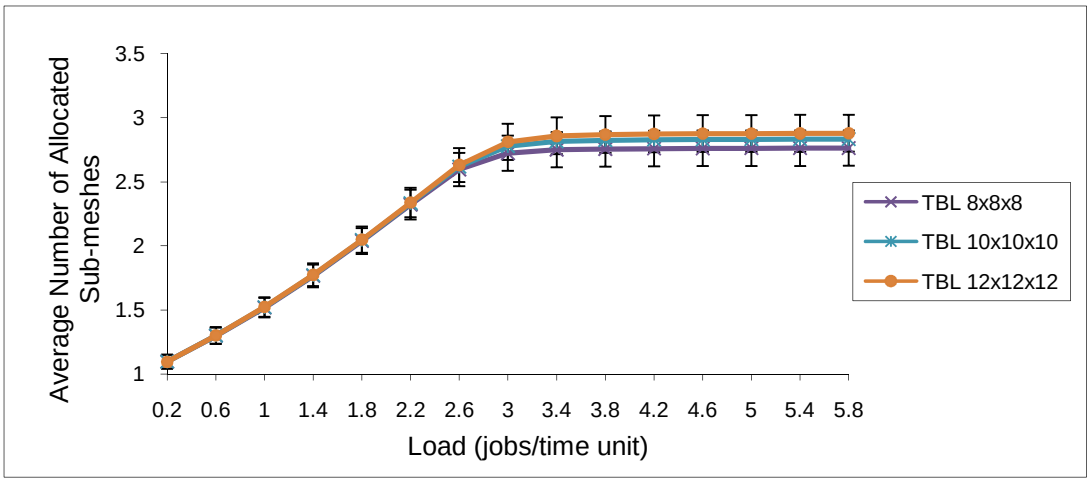


Figure 3.12: Average number of allocated sub-meshes (m) in TBL and the uniform side lengths distribution in 8 × 8 × 8, 10 × 10 × 10, and 12 × 12 × 12 meshes.

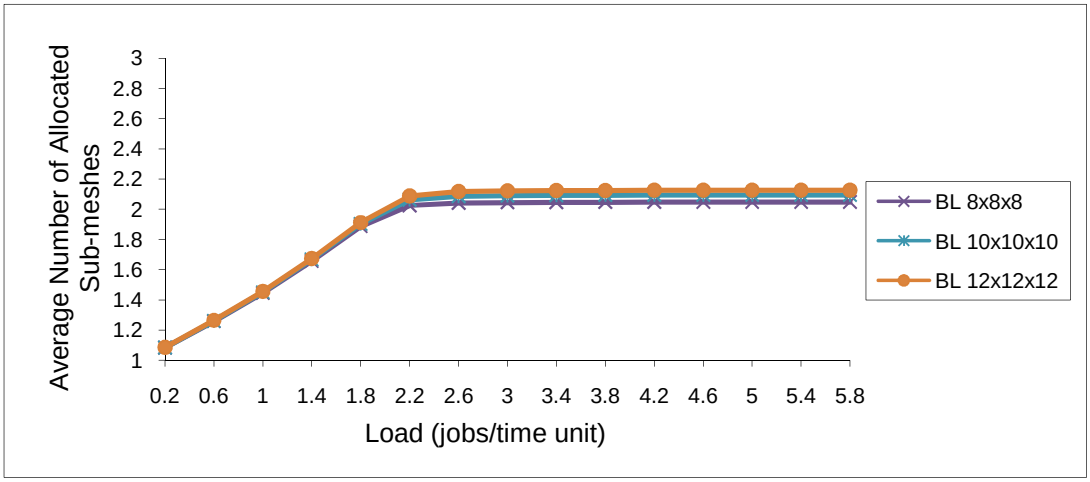


Figure 3.13: Average number of allocated sub-meshes (m) in BL and the uniform side lengths distribution in 8 × 8 × 8, 10 × 10 × 10, and 12 × 12 × 12 meshes.

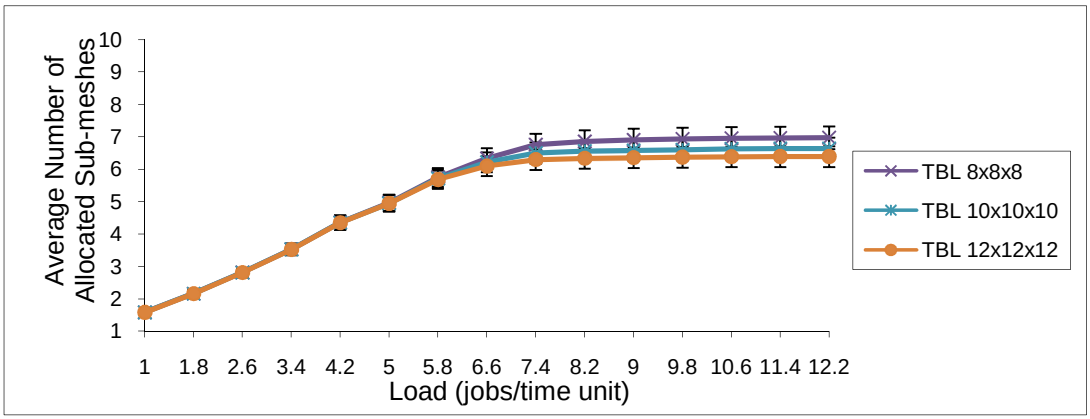


Figure 3.14: Average number of allocated sub-meshes (m) in TBL and the exponential side lengths distribution in 8 × 8 × 8, 10 × 10 × 10, and 12 × 12 × 12 meshes.

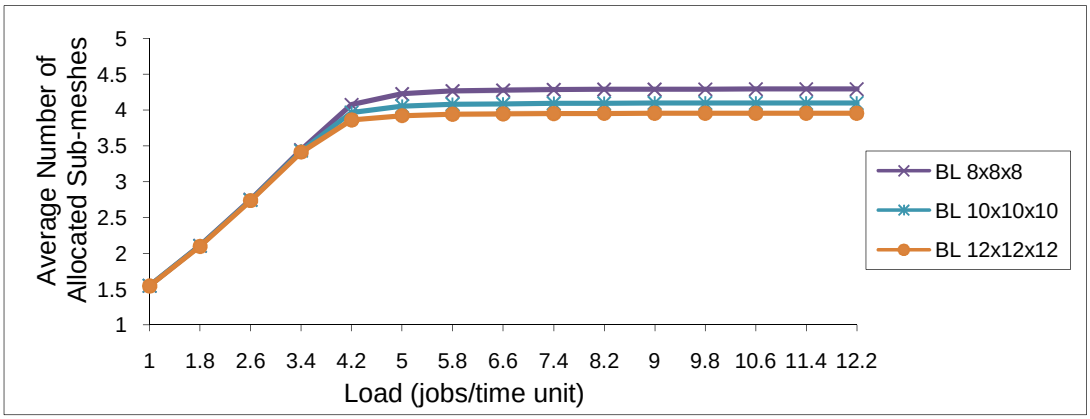


Figure 3.15: Average number of allocated sub-meshes (m) in BL and the exponential side lengths distribution in 8 × 8 × 8, 10 × 10 × 10, and 12 × 12 × 12 meshes.

Allocation Overhead (Allocation and De-allocation Time):

Before presenting the simulation results, let us first carry out a simple analysis of the time required for the allocation and de-allocation operations in the new TBL strategy. To do so, we need to examine the algorithm outlined in Figure 3.4 above. The RBP construction operation in Steps 2 and 2.1 of this algorithm requires $O(m)$ time, where m is the number of allocated sub-meshes. Subtracting a prohibited region from a RBP takes $O(1)$ time. As there are at most four RBP's and m prohibited regions, subtracting m prohibited regions from a RBP in step 2.3 of the algorithm takes $O(m)$ time. In total, the allocation operation takes $O(m^2)$ time since there are $4 \times m$ RBP's and m prohibited regions to be considered.

Typically, the average values of m are less sensitive with n, where n is the number of processors in the mesh, as has been seen in the simulation results above in Figures 3.12~3.15. The de-allocation operation requires m iterations to remove the allocated sub-mesh from the busy list. Therefore, the de-allocation operation takes $O(m)$ time. TBL maintains a busy list of m allocated sub-meshes. Thus, the space requirement of the TBL allocation strategy is $O(m)$. The space incurred by this strategy is small compared to the improvement in performance in terms of allocation overhead, as we will see in the simulation results.

As previously reported in Chapter 2, Section 2.5, the current version of ProcSimity ignores the overhead of allocation and de-allocation (i.e., the time that the allocation and de-allocation operations take per job). To compare the allocation strategies in terms of the allocation overhead associated with the allocation and de-allocation operations, we measured the average actual time taken by these operations on a Pentium machine running under Windows XP. The clock cycle of the machine is 3 GHz and the RAM size is 504 MB. The per-job average allocation overhead was computed in milliseconds over enough independent runs so that the confidence level is 95% that relative errors are below 5% of the mean.

In the remainder of this section, Figures 3.16~3.21 depict the average allocation overhead for the allocation strategies against the job arrival rate for different mesh sizes ($8 \times 8 \times 8$, $10 \times 10 \times 10$, and $12 \times 12 \times 12$), when request side lengths follow the uniform and exponential distributions. We observe that the strategies that depend on the busy list for both allocation and de-allocation (TBL, BL) have much smaller allocation overhead than the strategies that depend on the number of processors in the mesh system (TFF, FF). In Figure 3.16, for example, the allocation overhead of TBL strategy is 4% of that in TFF strategy under the job arrival rate 4.6 jobs/time unit. It can also be seen in the figures that the allocation overhead

for the strategies with rotation is higher than that of the strategies without rotation because in the worst case, the allocation process, in the strategies with rotation, is repeated for all possible permutations (6 permutations) of the job request while this process is repeated only one time for the other strategies.

The allocation overhead for the allocation strategies that depend on the list of allocated sub-meshes (TBL, BL) is little affected by changes in the system loads in our considered scenarios. This is because the average number of allocated sub-meshes in the busy list for these allocation strategies is much lower than the number of processors in the mesh system. In Figure 3.12 above, for example, the average number of allocated sub-meshes in the busy list varied from 1.09 to 2.76 from low to heavy loads. The allocation strategies, TBL and BL, depend on this small number of allocated sub-meshes in the busy list for both allocation and de-allocation. Consequently, the time needed for both allocation and de-allocation operations, for the allocation strategies that depend on a list of allocated sub-meshes, is little affected by changes in the system loads.

The average size of a requested sub-mesh is relatively small when the exponential distribution is used for generating job side lengths. Therefore, the number of allocated sub-meshes is larger in this case, meaning that the allocation choices are more numerous. Consequently, the allocation overhead of the strategies that depend on the busy list is largest when the side lengths follow the exponential distribution. Also and as shown in Figures 3.18~3.21, when the number of processors increases the allocation overhead increases for the allocation strategies that depend on the number of processors in the mesh system while it does not increase for the strategies that depend on a list of allocated sub-meshes. In Figures 3.16 and 3.20, for example, the allocation overhead of the TFF strategy for an $8 \times 8 \times 8$ mesh system size is 11% of that in TFF for a $12 \times 12 \times 12$ mesh system size under the job arrival rate 5.8 jobs/time unit. Moreover, the results reveal that the difference in allocation

overhead gets more noticeable as the system load increases. Thus, the strategies which depend on a list of allocated sub-meshes are more effective than the strategies that depend on the size of the mesh system.

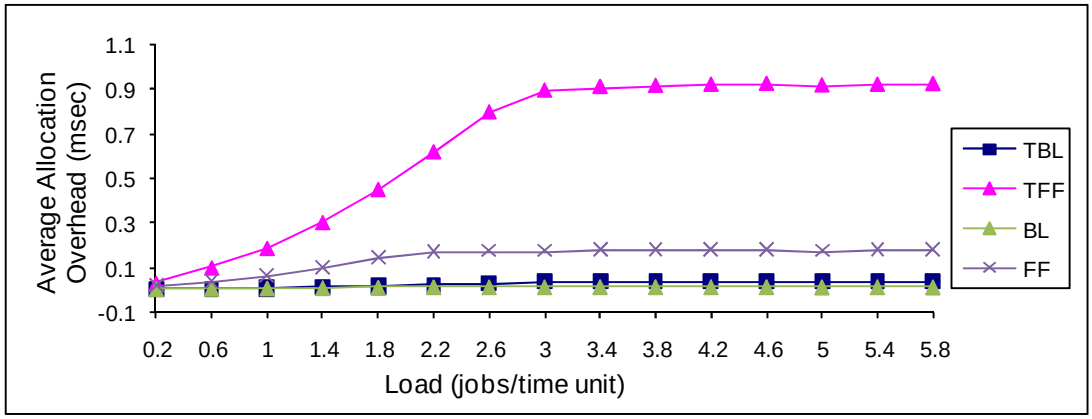


Figure 3.16: Average allocation overhead for the allocation strategies (TBL, TFF, BL, and FF) and uniform side lengths distribution in an 8 × 8 × 8 mesh.

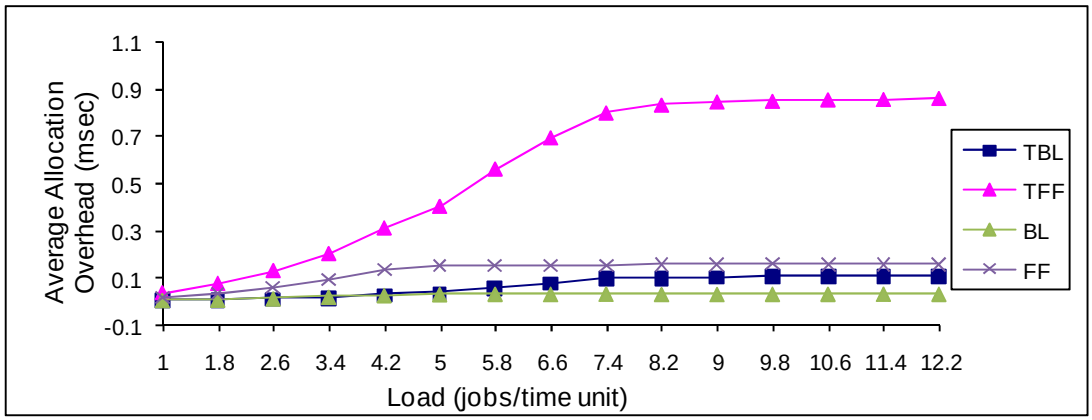


Figure 3.17: Average allocation overhead for the allocation strategies (TBL, TFF, BL, and FF) and exponential side lengths distribution in an 8 × 8 × 8 mesh.

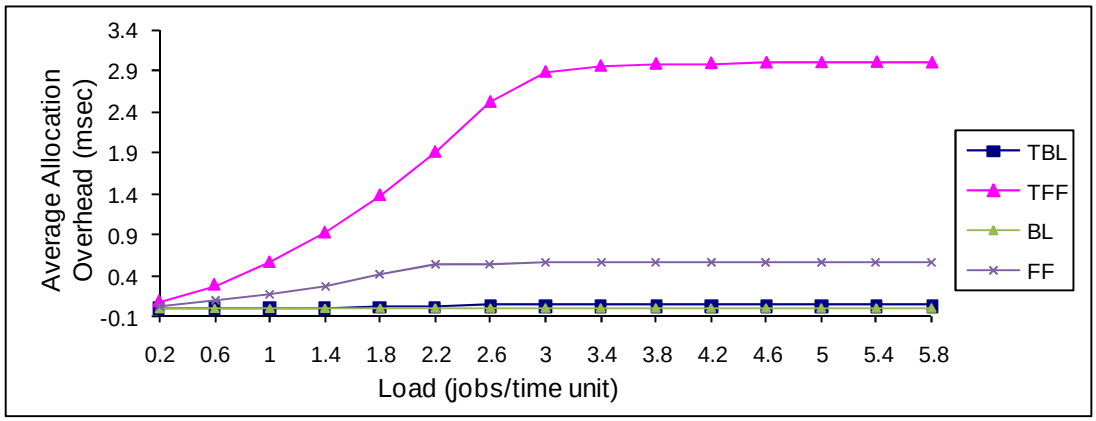


Figure 3.18: Average allocation overhead for the allocation strategies (TBL, TFF, BL, and FF) and uniform side lengths distribution in a 10 × 10 × 10 mesh.

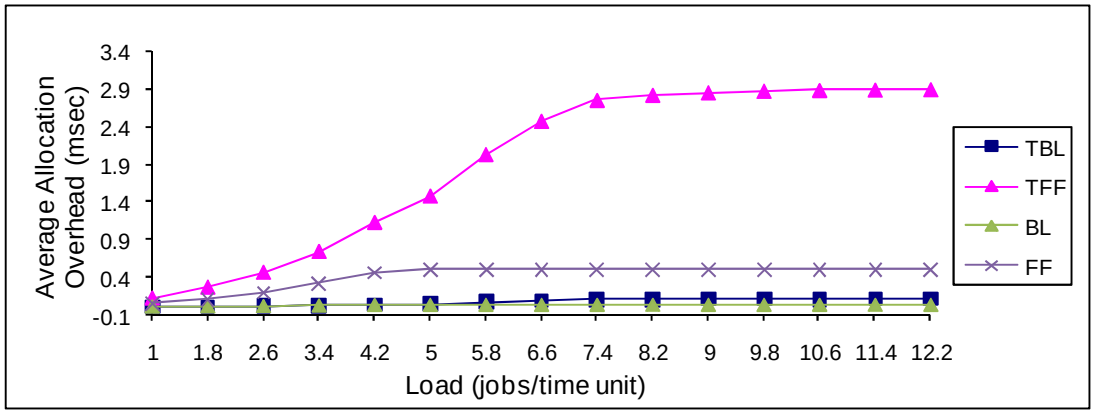


Figure 3.19: Average allocation overhead for the allocation strategies (TBL, TFF, BL, and FF) and exponential side lengths distribution in a 10 × 10 × 10 mesh.

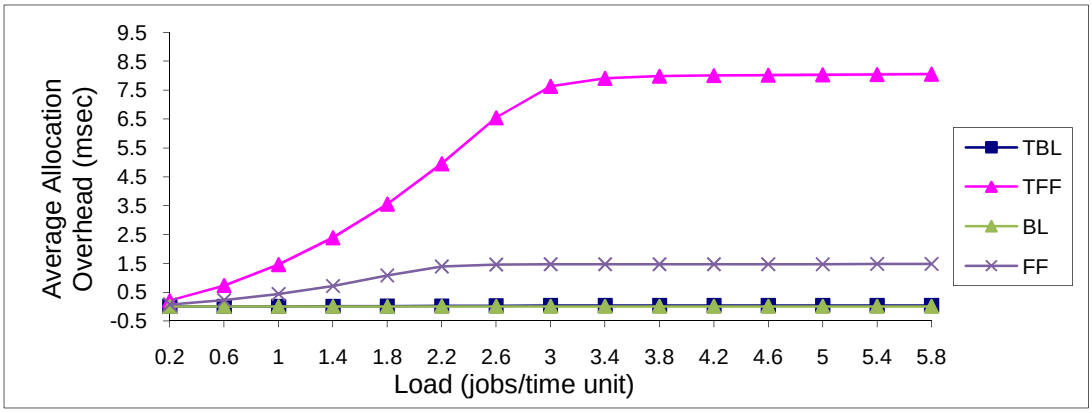


Figure 3.20: Average allocation overhead for the allocation strategies (TBL, TFF, BL, and FF) and uniform side lengths distribution in a 12 × 12 × 12 mesh.

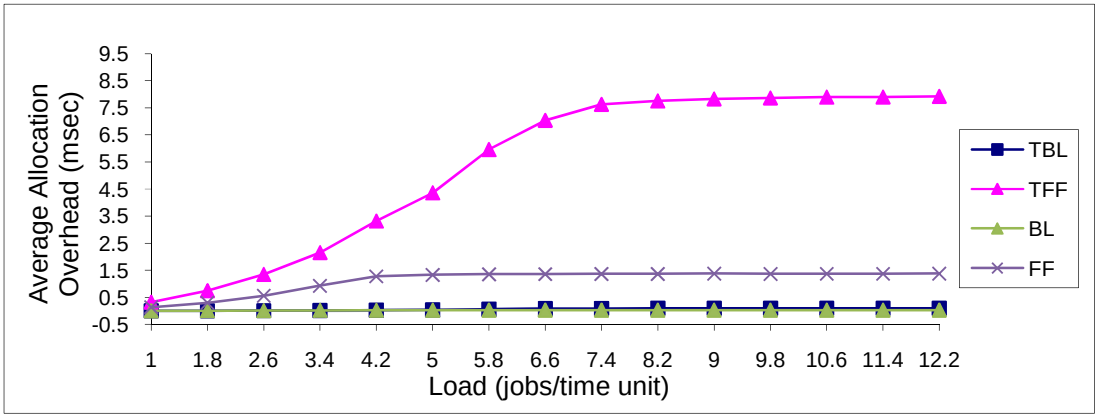


Figure 3.21: Average allocation overhead for the allocation strategies (TBL, TFF, BL, and FF) and exponential side lengths distribution in a 12 × 12 × 12 mesh.

3.4.1.1 Performance Impact of Mesh System Size

In this section, we investigate the effect of the size of the mesh system on the performance of the allocation strategies considered in this chapter in terms of average turnaround time of

jobs. Figures 3.22 and 3.23 plot the average turnaround time of jobs against the size of the mesh system, assuming a heavy system load of 5.8 and 12.2 jobs/time unit for the uniform and exponential side lengths distribution, respectively. The results show that the performance of the allocation strategies is little affected by changes in the system size in our considered scenarios. In Figure 3.22, for example, the average turnaround time of the TBL strategy for a 12 × 12 × 12 mesh system size is 93% of that of the TBL strategy for an 8 × 8 × 8 mesh system size. Moreover, the allocation strategies that use the orientation of the allocation request perform much better than the allocation strategies that do not use the orientation of the allocation request regardless of the mesh system size. For instance, Figure 3.23 shows that the average turnaround time of the TBL strategy is 44% of that of the BL strategy for a 12 × 12 × 12 mesh system size.

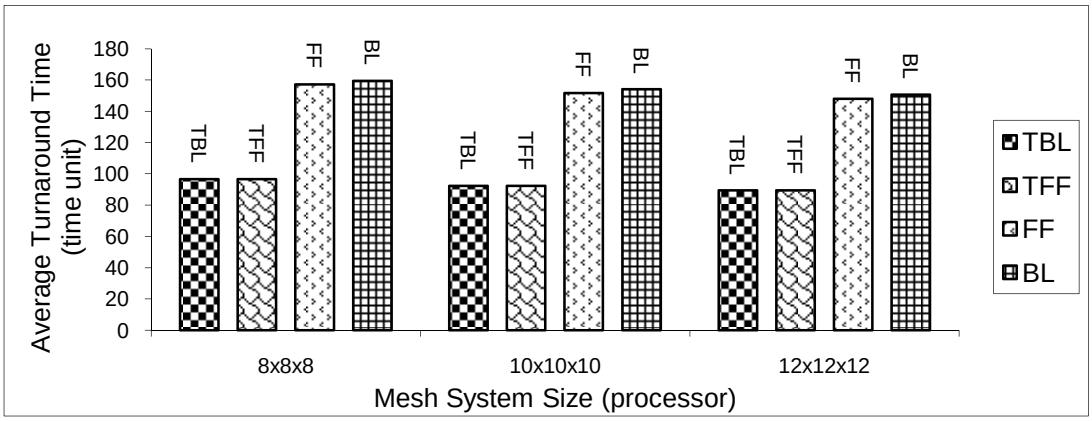


Figure 3.22: Average turnaround time vs. size of the mesh system for the contiguous allocation strategies (BL, FF, TBL, TFF) and the uniform side lengths distribution.

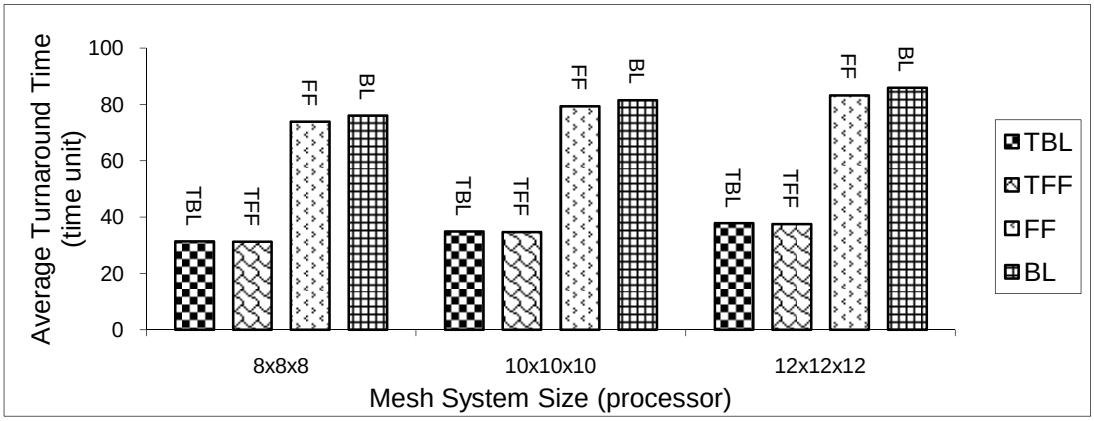


Figure 3.23: Average turnaround time vs. size of the mesh system for the contiguous allocation strategies (BL, FF, TBL, TFF) and the exponential side lengths distribution.

3.5 Conclusions

While the existing contiguous allocation strategies for the 3D mesh-connected multicomputers achieve complete sub-mesh recognition capability but with a high allocation overhead, this chapter has suggested an efficient contiguous allocation strategy, referred to as the Turning Busy List strategy (TBL for short), which can overcome the limitations of the existing strategies. The performance of the new strategy has been compared against that of the existing contiguous allocation strategies which have been suggested for the 3D mesh-connected multicomputers. Simulation results have shown that the performance of the TBL proposed allocation strategy is at least as good as that of the previously promising proposed strategies in terms of average turnaround time and mean system utilisation. Moreover, the allocation overhead of the TBL strategy is much lower than that of the existing strategies. The scenarios that have been examined in our simulation experiments have also revealed that system performance is affected only a little by a change in the network size.

The performance impact of the switching of request orientations has been also evaluated. The results have revealed that in general the rotation of the job request improves the performance of the contiguous allocation strategies. Moreover, TBL can be efficient because it is implemented using a busy list approach. This approach can be expected to be efficient in practice because when the mesh system size increases the requirement of applications in terms of the number of requested processors often increases and in such a case our algorithm is expected to exhibit competitive performance levels.

The subsequent chapter will describe a new non-contiguous allocation algorithm for the 2D mesh-connected multicomputers which can exhibit better performance in terms of the turnaround time than the previous non-contiguous allocation strategies in most of the cases

considered. Moreover, in the presence of high message contention due to heavy network traffic, the proposed strategy exhibits superior performance over the previous contiguous and non-contiguous allocation strategies; in particular, it exhibits high system utilisation as it manages to eliminate both internal and external processor fragmentation.

Chapter 4

Greedy Available Busy List (GABL): A New Non-contiguous Allocation Algorithm for Mesh-Connected Multicomputers

4.1 Introduction

Most allocation strategies [9, 27, 28, 33, 34, 38, 41, 48, 52, 65, 74, 75, 99] suggested for mesh-connected multicomputers are based on contiguous allocation, where the processors allocated to a parallel job are physically contiguous and have the same topology as that of the interconnection network of the multicomputer. Contiguous allocation strategies often result in high processor fragmentation, leading to a degradation in system performance in terms of average turnaround time of jobs and mean system utilisation, as has been shown in [99] (please refer to Section 2.1 in Chapter 2 for the definition of processor fragmentation).

The main goal of a processor allocation strategy is to reduce the job turnaround time and at the same time maximize the system utilisation by alleviating the processor fragmentation problem. Several studies have attempted to reduce processor fragmentation [18, 24, 28, 35, 51, 77, 81, 85]. One of the suggested solutions is to adopt *non-contiguous* allocation [18, 24,

49, 72, 85]. In non-contiguous allocation, a job can execute on multiple disjoint smaller sub-networks rather than always waiting until a single sub-network of the requested size and shape is available. Although non-contiguous allocation increases message contention in the network, lifting the contiguity condition is expected to reduce processor fragmentation and increase processor utilisation [18, 72, 85]. It is the introduction of wormhole routing [2, 11, 83] that has lead researchers to consider non-contiguous allocation on multicomputer networks with a long communication distances, such as the 2D mesh [2, 18, 49, 77, 85]. This is due to the fact that one of main advantages of wormhole routing over earlier communication schemes, e.g., store-and-forward, is that message latency is less dependent on the message distance.

Most existing research studies have been conducted in the context of contiguous allocation [9, 27, 28, 33, 38, 48, 65, 81, 99]. There has been comparatively very little work on non-contiguous allocation. Whereas contiguous allocation eliminates contention among the messages of concurrently executing jobs, non-contiguous allocation can eliminate processor fragmentation that contiguous allocation suffers from. Furthermore, most existing research on contiguous and non-contiguous allocation has been carried out in the context of the 2D mesh [9, 18, 27, 28, 33, 35, 38, 48, 49, 51, 65, 77, 81, 85, 99]. The mesh network has been used as the underlying network in a number of practical and experimental parallel machines, such as the iWARP [15], IBM BlueGene/L [10, 55, 98], Cplant [84], and Delta Touchstone [40]. Examples of current generation mesh-connected systems that use non-contiguous allocation are the Cplant [84] and Cray XT3 [19, 60].

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 [18, 24, 49, 84, 85]. Also, the allocation for job requests is not

based on free contiguous sub-meshes [18, 85]. Instead, it is often based on artificial predefined geometric or arithmetic patterns [18, 85]. For example, in the study of [18], ANCA subdivides job requests into two equal parts. The subparts are successively subdivided in a similar fashion if allocation fails for any of them. In the study of [85], MBS strategy bases partitioning on a base-4 representation of the number of processors requested, and partitioning in Paging [85] 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 and which in turn can cause degradation in system performance in terms of turnaround times [18, 72, 85].

Motivated by the above observations, this chapter makes the following contributions. We describe a new non-contiguous allocation strategy, referred to here as Greedy Available Busy List (GABL for short), for the 2D mesh, and compare its performance properties using detailed simulations against those of the previous non-contiguous allocation strategies Paging(0) and Multiple Buddy Strategy (MBS) [85]. These two strategies have been selected because they have been shown to perform well in [85]. The MBS and Paging(0) have been discussed in detail in Chapter 2 (please see Section 2.2.2). To show the superiority of non-contiguous allocation against contiguous allocation with respect to fragmentation, the GABL strategy is compared against the contiguous First Fit strategy (FF) [99] as this has been used in previous related studies [18, 85].

This chapter also conducts a performance evaluation of the non-contiguous allocation strategies in terms of overall performance parameters such as the average turnaround time, average waiting time, and mean system utilisation. Furthermore, the contention in the network that results from the communication among allocated processors has been measured using two metrics. These are the contiguous ratio and average blocks per job. The

contiguous ratio measures the ratio of jobs that allocated contiguously. The average blocks per job is defined as the average number of non-contiguous blocks allocated to a job. Message contention decreases when the number of blocks allocated to a job deceases. This study is the first to examine the non-contiguous allocation based on the sub-meshes available for allocation. The results show that the proposed strategy has lower turnaround times than the previous non-contiguous allocation strategies of [85]. When message contention increases inside the network, the proposed strategy exhibits superior performance in terms of job turnaround times over the previous contiguous and non-contiguous allocation strategies. Furthermore, the proposed strategy exhibits high system utilisation as it manages to eliminate both internal and external fragmentation.

The remainder of the chapter is organized as follows. Section 4.2 describes our proposed non-contiguous allocation strategy. Section 4.3 compares the performance of the contiguous and non-contiguous allocation strategies. Finally, Section 4.4 concludes this chapter.

4.2 The Proposed Greedy Available Busy List Allocation Strategy (GABL)

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 (for the sake of conciseness please refer to the description of 3D mesh in Section 3.2 in Chapter 3, as the adaptation of the description to the 2D mesh is straightforward).

The GABL strategy partitions requests based on the sub-meshes available for allocation. A major goal of the partitioning process is to maintain a high degree of contiguity among the processors allocated to a given parallel job. Furthermore, the GABL strategy combines the desirable features of both contiguous and non-contiguous allocation strategies. For example, the desirable features of any ideal contiguous allocation strategy are to eliminate the

communication overhead among processors allocated to a parallel job and to achieve complete sub-mesh recognition capability with low allocation overhead. The desirable feature of an ideal non-contiguous allocation strategy is to alleviate communication overhead among processors allocated to a job by maintaining a degree of contiguity between them. Moreover, GABL 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 strategy which has been suggested for the 3D mesh network.

In implementing GABL, we exploit an efficient approach, the Turning Busy List (TBL) approach described in Chapter 3, for the detection of such available sub-meshes. As previously discussed in Chapter 3, the basic idea of TBL is to maintain a list of the allocated sub-meshes. The list is used to determine all prohibited regions, which are sub-meshes consisting of the nodes that cannot serve as base nodes for the requested sub-mesh. The prohibited regions are then subtracted from the right border lines of the allocated sub-meshes so as to locate nodes that could be used as base nodes for the required sub-mesh. The TBL algorithm in Chapter 3 builds the busy list in order to detect the free sub-meshes in the target mesh. The detection of available sub-meshes and the allocation process for 2D mesh are implemented by the algorithms illustrated in Figures 4.1 and 4.2 respectively.

Procedure Detect (α, β):

Begin *{*

{Mesh M(W, L); incoming job J requests for an α×β free sub-mesh;
Busy list B = {b_0, b_1, b_2, ….., b_m} where b_0 is a hypothetical allocated sub-mesh and b_i, $1 \le i \le m$, are the m already allocated sub-meshes; Both sub-meshes (W-α+1, 0, W-1, L-1) and (0, L-β+1, W-1, L-1) are automatic prohibited regions and

automatically not available for accommodating the base node of a free α×β sub-mesh for J.}

Step 1. RBL_Nodes←NULL.

Step 2. for each allocated sub-mesh $b_i(x_1, y_1, x_2, y_2)$ from i = 0 to m

Step 2.1. Construct RBL of b_i, denoted as $RBL_i = (x_r, y_{r1}, x_r, y_{r2})$, with respect to J where $x_r=x_2+1$, $y_{r1}=max(y_1-\beta+1, 0)$, and $y_{r2}=y_2$.

Step 2.2. if RBL_i is within an automatic prohibited region then goto Step 2.

Step 2.3. for each allocated sub-mesh b_j (x_1,y_1,x_2,y_2) from j = 1 to m

Construct prohibited region of J with respect to b_j, denoted as $P_j = (x_{p1}, y_{p1},x_{p2},y_{p2})$ where $x_{p1}=max(x_1-\alpha+1, 0)$, $y_{p1}=max(y_1-\beta+1, 0)$, $x_{p2}=x_2$, and $y_{p2}=y_2$.

subtract P_j from RBL_i as follows:

Determine the case to which the subtraction belongs by comparing the coordinates of RBL_i and P_j as the following:

1. $((x_r < x_{p1}) \,||\, (x_r > x_{p2}) \,||\, (y_{r2} < y_{p1}) \,||\, (y_{r1} > y_{p2}))$.

2. $((x_r >= x_{p1}) \,\&\&\, (x_r <= x_{p2}) \,\&\&\, (y_{r2} >= y_{p1}) \,\&\&\, (y_{r2} <= y_{p2}) \,\&\&\, (y_{r1} < y_{p1}))$

3. $((x_r >= x_{p1}) \,\&\&\, (x_r <= x_{p2}) \,\&\&\, (y_{r1} >= y_{p1}) \,\&\&\, (y_{r1} <= y_{p2}) \,\&\&\, (y_{r2} > y_{p2}))$

4. $((x_r >= x_{p1}) \,\&\&\, (x_r <= x_{p2}) \,\&\&\, (y_{r1} < y_{p1}) \,\&\&\, (y_{r2} > y_{p2}))$

5. $((x_r >= x_{p1}) \,\&\&\, (x_r <= x_{p2}) \,\&\&\, (y_{r1} >= y_{p1}) \,\&\&\, (y_{r2} <= y_{p2}))$

Switch (subtraction case)

{

case (1): if $(y_{r1} > y_{p2})$ then

begin

add the whole RBL_i to RBL_Nodes.

goto Step 2.

end

break.

case (2): adjust RBL_i such that $y_{r2} \leftarrow y_{p1}-1$.

break.

case (3): add line segment $(x_r, y_{p2}+1, x_r, y_{r2})$ to RBL_Nodes.

goto Step 2.

case (4): add line segment $(x_r, y_{p2}+1, x_r, y_{r2})$ to RBL_Nodes.

adjust RBL_i such that $y_{r2} \leftarrow y_{p1}-1$.

break.

case (5): goto Step 2.

}

goto Step 2.3.

TBL_Allocate(RBL_Nodes, α, β)

} ***End.***

Figure 4.1: Outline of the Detect Procedure in TBL Contiguous Allocation Strategy for 2D Mesh

Procedure TBL_Allocate (RBL_Nodes, α, β):

Begin *{*

int botx, boty;

botx=RBL_Nodes.botx;

boty=RBL_Nodes.boty;

Add the sub-mesh represented by the address (botx, boty, botx + α -1, boty + β – 1) to the busy list by setting sub-mesh's ID to the job ID.

} ***End.***

Figure 4.2: Outline of the TBL Contiguous Allocation Strategy for 2D Mesh

To explain how the detection of the available sub-meshes and the allocation process on the 2D mesh works; consider the example of Figure 4.3 in which a 6 × 6 mesh is illustrated. There are 4 allocated sub-meshes in this example. These allocated sub-meshes are denoted by $\{b_1, b_2, b_3, b_4\}$ and represented by the addresses b_1(1, 4, 5, 5), b_2(0, 2, 1, 3), b_3(4, 3, 5, 3), and b_4(5, 2, 5, 2), respectively. Assume that an incoming job J requests a 2 × 4 sub-mesh. Now, consider the sub-mesh b_2(0, 2, 1, 3). The RBL of b_2(0, 2, 1, 3) with respect to the job request $J(2\times4)$ is (2, 0, 2, 3). The automatic prohibited regions with respect to the job request $J(2\times4)$ are calculated resulting in the regions (5, 0, 5, 5) and (0, 3, 5, 5). The automatic prohibited regions are subtracted from the RBL (2, 0, 2, 3) resulting in (2, 0, 2, 2). Now, the prohibited region of the first allocated sub-mesh in the busy list b_1(1, 4, 5, 5) with respect to the job request $J(2\times4)$ is calculated resulting in (0, 1, 5, 5), which when subtracted from the RBL (2, 0, 2, 2) results in (2, 0, 2, 0). Then, the prohibited region of the second allocated sub-mesh b_2(0, 2, 1, 3) with respect to the job request $J(2\times4)$ is calculated resulting in (0, 0, 1, 3), which when subtracted from the RBL (2, 0, 2, 0) results in RBL (2, 0, 2, 0). The prohibited region of the third allocated sub-mesh b_3(4, 3, 5, 3) with respect to the job request $J(2\times4)$ is calculated resulting in (3, 0, 5, 3), which when subtracted from RBL (2, 0, 2, 0) results in (2, 0, 2, 0). Finally, the prohibited region of the last allocated sub-mesh b_4(5, 2, 5, 2) with respect to the job request $J(2\times4)$ is calculated resulting in (4, 0, 5, 2), which when subtracted from the RBL (2, 0, 2, 0) results in (2, 0, 2, 0). Now, the node (2, 0, 2, 0) will be used as a base node for the sub-mesh requested by the job request $J(2\times4)$ and the sub-mesh (2, 0, 3, 3) is allocated to the job request $J(2\times4)$

and then it is added to the busy list.

In GABL, 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 the job using the above TBL contiguous allocation strategy. Otherwise, the largest free sub-mesh that can fit inside $S(\alpha,\beta)$ 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 $\alpha\times\beta$. This last step is repeated until $\alpha\times\beta$ processors are allocated. For example, given the system state shown in Figure 4.3 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 using TBL contiguous allocation strategy. 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.

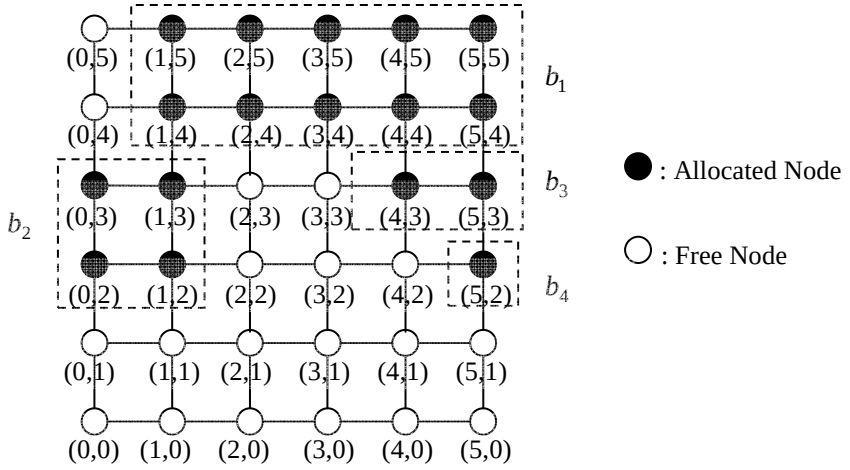


Figure 4.3: A 6 × 6 sub-mesh with 19 free processors forming several free sub-meshes

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 4.4, while the de-allocation algorithm is outlined in Figure 4.5. Note that allocation always succeeds if the number of free processors is $\geq \alpha \times \beta$. 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 = $\alpha \times \beta$

*Step*1. *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.
}
*Step*3. $\alpha_{new} = \alpha$ *and* $\beta_{new} = \beta$
*Step*4. *Subtract* 1 *from max* (α_{new}, β_{new}) *if max* > 1
*Step*5. *If (Total _allocated* + $\alpha_{new} \times \beta_{new}$ > *Job_Size) go to step 4*
*Step*6. *If there is a free S (w, l) suitable for S*(α_{new}, β_{new})

```
        {
                Allocate it using TBL contiguous allocation algorithm.
                Total_allocated = Total_allocated + α_new × β_new
        }
Step7. If (Total_allocated == Job_Size)
                return success.
        else
                go to Step 5.
} End.
```

Figure 4.4: 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 4.5: Outline of the Greedy Available Busy List de-allocation algorithm

4.3 Performance Evaluation

In this section, the allocation and de-allocation time, in addition to the space requirement in the proposed allocation strategy, are presented first. Then, the results from simulations that have been carried out to evaluate the performance of the proposed algorithm are presented and compared against those of Paging(0), MBS and FF.

4.3.1 Allocation and De-allocation Time in GABL

When a sub-mesh is allocated, TBL takes $O(m^2)$ time, where m is the number of allocated sub-meshes. Therefore, the time of Step 6 in GABL's allocation algorithm is in the order of $O(bm^2)$, where b is the number of allocation attempts carried out in this step. The worst case for TBL occurs when the free and busy processors alternate in the same way as the

light and dark positions on a chessboard, and a job requires the allocation of $n/2$ processors, where n is the number of processors in the mesh system. As b is in $O(n)$ in such a case, the worst-case time for Step 6 of the allocation algorithm is in $O(n^3)$. However, as we shall show in the simulation results below, the average values of b and m are less sensitive to n. The number of times Steps 4 and 5 are executed is in $O(n)$ in the worst case. These steps exhibit their worst case behaviour when all free sub-meshes are of size equal to one. The simulation results show that Step 6 dominates Steps 4 and 5 for the typical cases considered in this study. When a job departs, the busy list is scanned so as to determine the sub-meshes to be released. Therefore, the de-allocation algorithm takes $O(m)$ time. The proposed algorithm maintains a busy list. Therefore, its space requirement is in $O(m)$.

4.3.2 Simulation Results

In addition to simulation results for GABL, we will show below the results for Paging(0), MBS and FF. We have implemented the proposed allocation and de-allocation algorithms, including the busy list routines, 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 [50, 66].

The target mesh modelled 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 at which a job completes (i.e., a job completes when the messages it should send have been sent [85]) minus the time at which allocation succeeds for the job and the job starts

execution. 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. As previously reported in Chapter 3, 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 exponential distribution, where the width and length of job requests are exponentially distributed with a mean of half the side length of the entire mesh; where the width and length of the job requests are rounded to the integer values using floor function and bounded by the dimensions of the mesh. The exponential distribution represents the case where most jobs are small relative to the size of the system. These distributions have often been used in the literature [20, 27, 77, 85, 99].

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}. As previously reported in Chapter 2, Section 2.3.3, processors allocated to a job communicate with each other using one of three common communication patterns [49, 83, 85]. 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 all-to-all, where each processor allocated to a job sends a packet to all other processors allocated to the same job. This communication pattern causes much message collision and is known as the weak point for non-contiguous allocation algorithms [49]. In the third communication pattern, randomly selected processors send packets to randomly selected destinations within the set of processors allocated the same job. 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. As in [85], the number of messages that are actually generated by a given job is exponentially distributed with a mean num_mes. Unless specified otherwise, the performance figures shown below are for a 16 × 16 mesh, $t_s = 3$ time units, $P_{len} = 8$ flits and $num\_mes = 5$ packets. 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 [20, 27, 49, 77, 85, 99] and have been recommended in [66].

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	GABL, MBS, Paging(0), and FF
Scheduling Strategy	FCFS
Job Size Distribution	Uniform: Job widths and lengths are uniformly distributed over the range from 1 to the mesh side lengths.
	Exponential: Job widths and lengths are exponentially distributed with a mean of half the side length of the entire mesh.
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, All-to-All, and Random
Messages per Job	Messages per Job are exponential distributed with a mean = 5.0.
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% [7]. The method used to calculate confidence intervals is called the batch means analysis [4, 66]. This method has been discussed in detail in Chapter 3 (please see Section 3.4.1). Table 4.2 shows the grand means, confidence intervals, and relative errors that outline the results depicted in Figure 4.6 for the load 0.0185 jobs/time unit. In most of the cases the error bars are quite small. These error bars are not shown on all the figures for the sake of clarity.

Table 4.2: The mean (i.e., mean turnaround time of job), 95% confidence interval, and relative error for the results shown in Figure 4.6 for the load 0.0185 jobs/time unit

Algorithm	GABL	MBS	Paging(0)	FF
95% Confidence Interval	[5019.37-5329.85]	[8177.79-8342.99]	[9079.11-9449.688]	[18661.92-19038.93]
Mean (time unit)	5174.610807	8260.392389	9264.400494	18850.428350
Relative Error	0.03	0.01	0.02	0.01

The main performance parameters used are the *average turnaround time* of jobs, *average waiting time*, *mean system utilisation*, and *contiguous ratio*. The *turnaround time* of a job is the time that the job spends in the mesh from arrival to departure. The *waiting time* is the time that the job spends in the queue before it is allocated the requested sub-mesh. The *system utilisation* is the percentage of processors that are utilized over time. The *contiguous ratio* is the ratio of jobs which are allocated contiguously. The important independent variable in the simulation is the *system load*. It is defined as the inverse of the mean inter-arrival time of jobs. Its range of values from low to heavy loads has been determined through experimentation with the simulator allowing each allocation strategy to reach its upper limits of utilisation. 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.

Turnaround Time:

In Figures 4.6 and 4.7, the average turnaround times of jobs are plotted against the system load for the one-to-all communication pattern. The results reveal that GABL performs better than all other contiguous and non-contiguous allocation strategies for both job size distributions considered in this research. Furthermore, GABL is substantially superior to the contiguous allocation FF strategy for both job size distributions. In Figure 4.6, for example, the difference in performance in favour for GABL could be as large as 65% compared to FF, and 36% to Paging(0), and 30% to MBS under the job arrival rate 0.0205 jobs/time unit. Experiments that use larger packet sizes (16, 32, and 64 flits) have been also conducted. Their results lead to the same conclusion on the relative performance of the allocation strategies (please see Section 4.3.2.2). Moreover, the results indicate that the relative performance merits of the non-contiguous GABL strategy over the remaining contiguous and non-contiguous allocation strategies become more noticeable as the packet length increases.

In Figures 4.8 and 4.9, the average turnaround times of jobs are plotted against the system load for the all-to-all communication pattern. Again, GABL performs much better than all other allocation strategies for both job size distributions. Moreover, GABL is substantially superior to FF for both job size distributions. Figure 4.8, for example, shows that when the job arrival rate is 0.0305 jobs/time unit, the average turnaround times of GABL are 20%, 24%, and 38% of that of FF, Paging(0), and MBS, respectively. Experiments that use larger packet sizes (16, 32, and 64 flits) have lead to the same conclusion as to the relative performance of the allocation strategies (please see Section 4.3.2.2).

In Figures 4.10 and 4.11, the average turnaround times are plotted against the system load for the random communication pattern. The results in Figure 4.10 reveal that the non-

contiguous GABL strategy outperforms the other non-contiguous allocation strategies for the uniform side lengths distribution. It can also be noticed from Figure 4.11 that GABL performs better than the non-contiguous Paging(0) strategy for the exponential side lengths distribution. However the performance of GABL is very close to that of the non-contiguous MBS strategy. For instance, Figure 4.11 reveals that the average turnaround times of GABL are 44%, 89%, and 99% of that of FF, Paging(0), and MBS, respectively, under the job arrival rate 0.1 jobs/time unit.

GABL is overall better than the 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 all-to-all 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 Paging(0), the performance 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. As expected, the results show that GABL is substantially superior to the contiguous FF strategy. 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.

Experiments that use large system sizes (32 × 32 and 64 × 64) have been also conducted for the three communication patterns. The results lead to the same conclusion about the relative performance of the allocation strategies (please see Section 4.3.2.1).

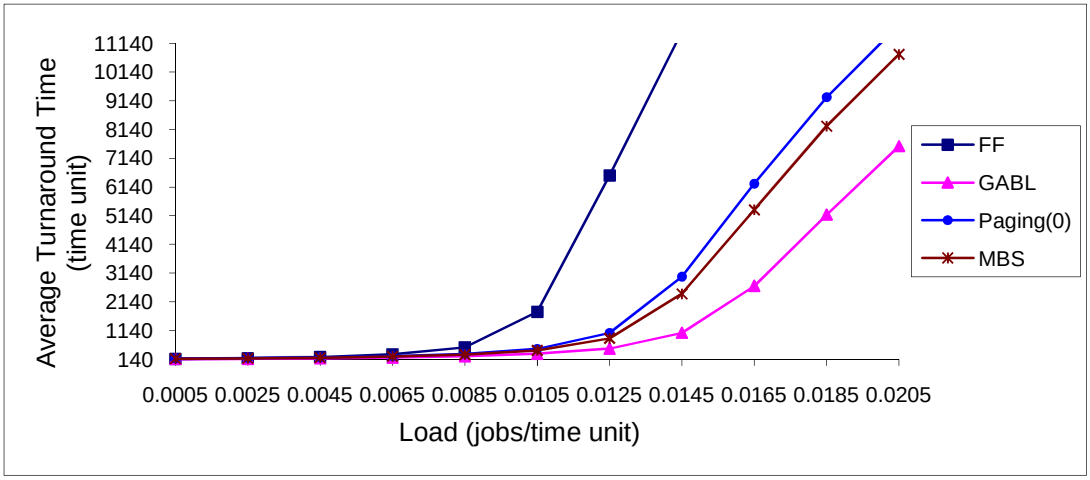


Figure 4.6: Average turnaround time vs. system load for the one-to-all communication pattern and uniform side lengths distribution in a 16 × 16 mesh.

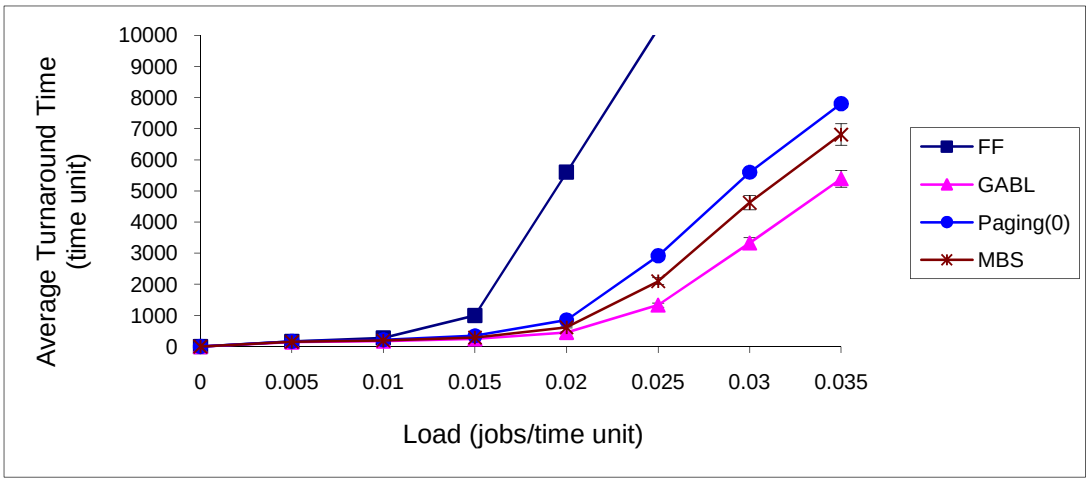


Figure 4.7: Average turnaround time vs. system load for the one-to-all communication pattern and exponential side lengths distribution in a 16 × 16 mesh.

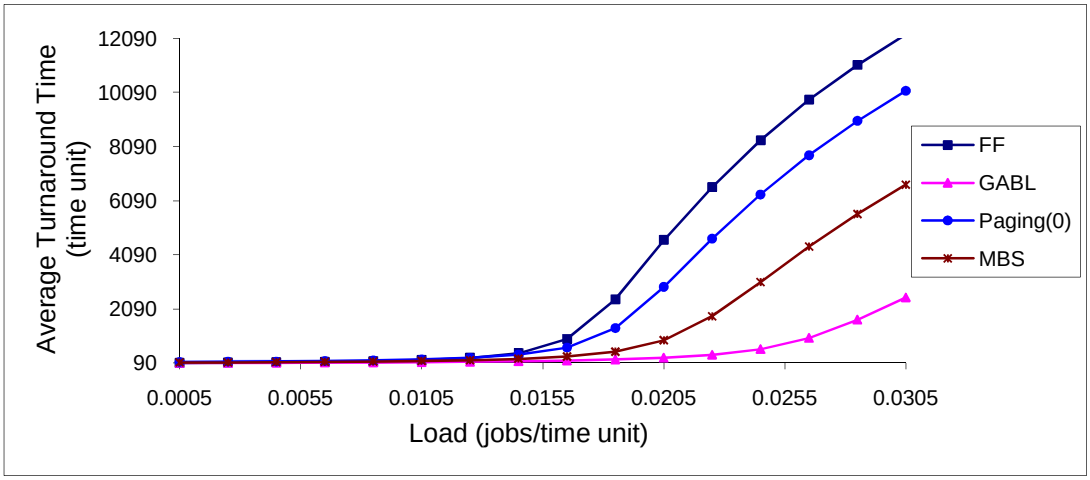


Figure 4.8: Average turnaround time vs. system load for the all-to-all communication pattern and uniform side lengths distribution in a 16 × 16 mesh.

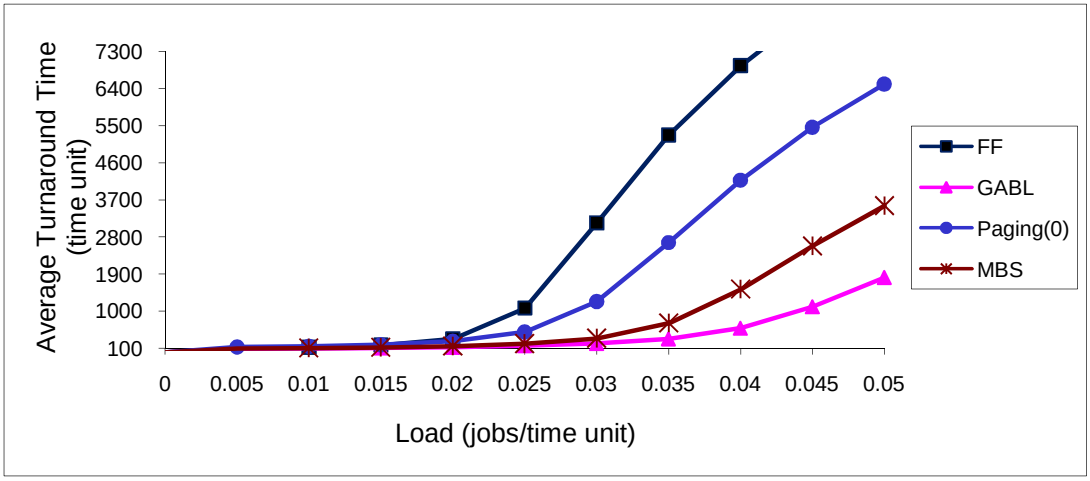


Figure 4.9: Average turnaround time vs. system load for the all-to-all communication pattern and exponential side lengths distribution in a 16 × 16 mesh.

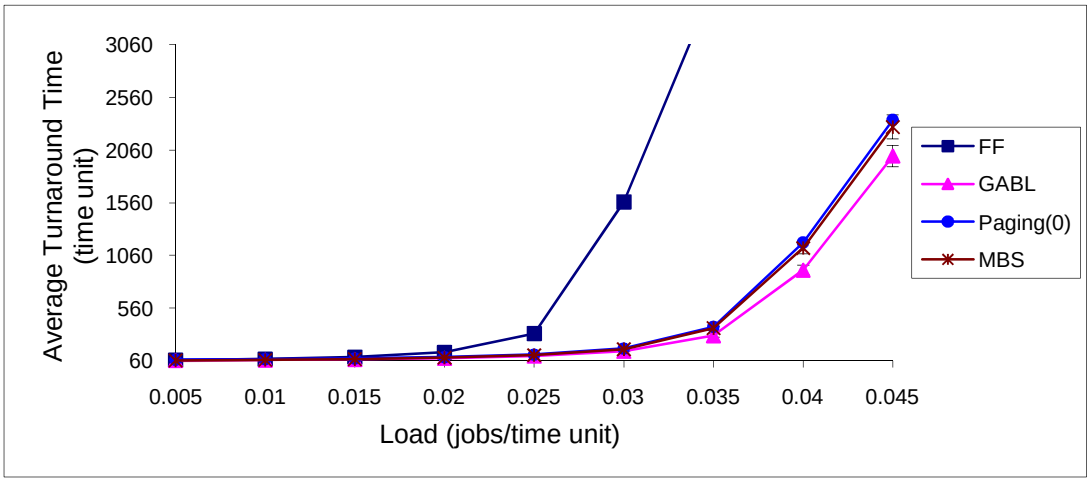


Figure 4.10: Average turnaround time vs. system load for the random communication pattern and uniform side lengths distribution in a 16 × 16 mesh.

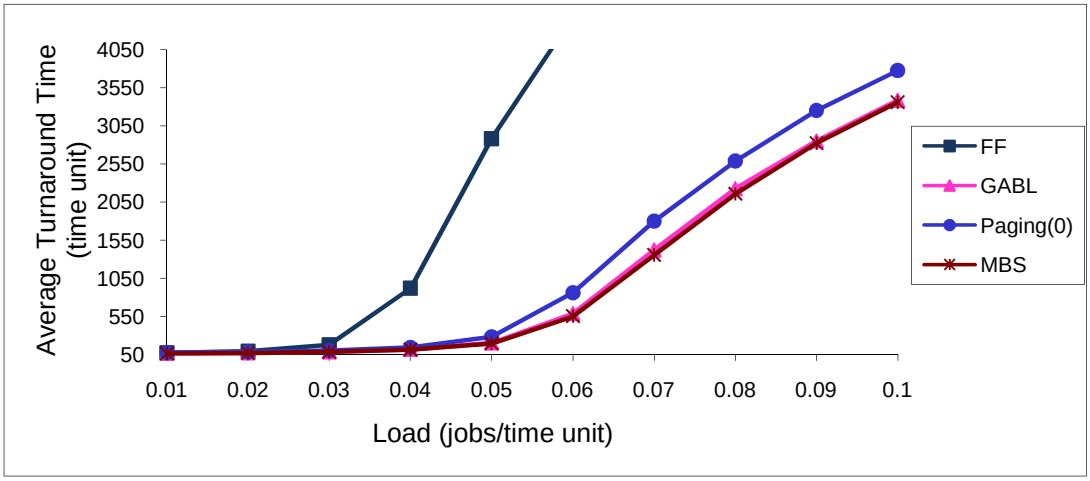


Figure 4.11: Average turnaround time vs. system load for the random communication pattern and exponential side lengths distribution in a 16 × 16 mesh.

Waiting Time:

In Figures 4.12 and 4.13, the average waiting times of jobs are plotted against the system load for the one-to-all communication pattern. The results reveal that GABL performs better than all other contiguous and non-contiguous allocation strategies for both job size distributions. This is because the degree of contiguity between allocated processors in GABL is higher than that of the previous non-contiguous allocation strategies, and thus decreases the distance traversed by messages. This in turn decreases the communication overhead, which means that the allocation in the GABL strategy is more likely to succeed. As a consequence, the waiting time is lower. Furthermore, GABL is substantially superior to FF for both job size distributions. In Figure 4.12, for example, the average waiting times of GABL are 35%, 64%, and 70% of that of FF, Paging(0), and MBS, respectively, under the job arrival rate 0.0205 jobs/time unit.

In Figures 4.14 and 4.15, the average waiting times of jobs are plotted against the system load for the all-to-all communication pattern. Again, GABL outperforms all other strategies for both job size distributions. Moreover, GABL is substantially superior to FF for both job size distributions. Figure 4.15, for example, depicts that when the job arrival rate is 0.05 jobs/time unit, the average waiting times of GABL are 19%, 27%, and 50% of that of FF, Paging(0), and MBS, respectively.

In Figures 4.16 and 4.17, the average waiting times are plotted against the system load for the random communication pattern. Figure 4.16 depicts that GABL has a better performance than the other non-contiguous allocation strategies for the uniform side lengths distribution. It can also be noticed from Figure 4.17 that GABL performs better than the non-contiguous Paging(0) strategy for the exponential side lengths distribution. But GABL's performance is

comparable to that of MBS strategy. For instance, Figure 4.17 shows that the average waiting times of GABL are 43%, 89%, and 99% of that of FF, Paging(0), and MBS, respectively, under the job arrival rate 0.1 jobs/time unit.

Overall, GABL is better than the previous non-contiguous allocation strategies at decreasing waiting times in the waiting queue. This conclusion is compatible with the values of the average turnaround times shown above.

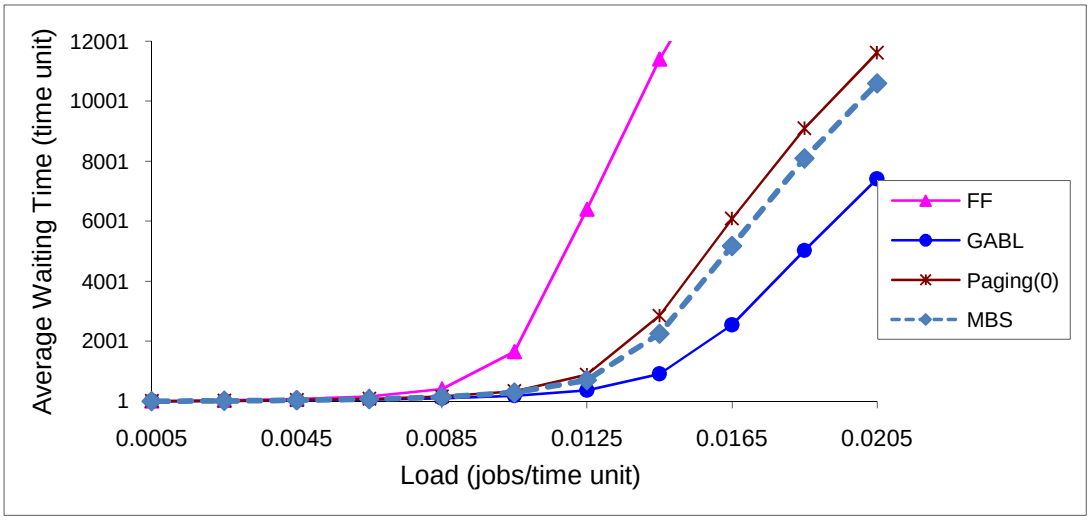


Figure 4.12: Average waiting time vs. System load for the one-to-all communication pattern and uniform side lengths distribution in a 16 × 16 mesh.

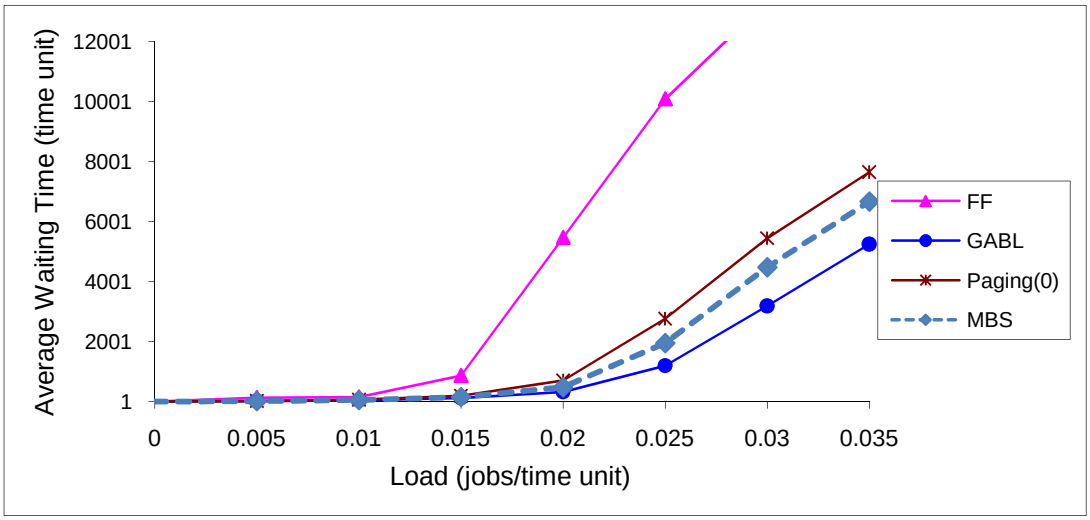


Figure 4.13: Average waiting time vs. System load for the one-to-all communication pattern and exponential side lengths distribution in a 16 × 16 mesh.

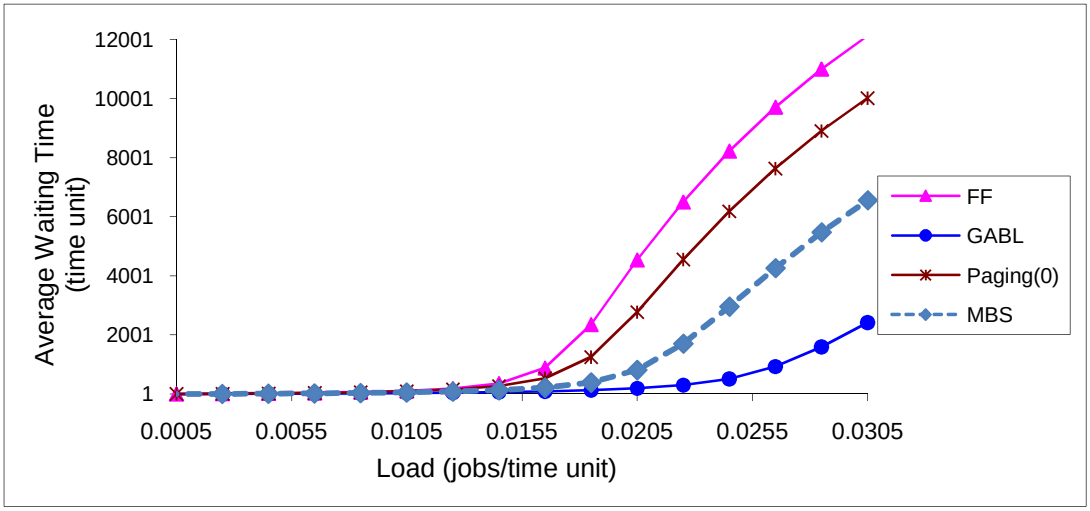


Figure 4.14: Average waiting time vs. System load for the all-to-all communication pattern and uniform side lengths distribution in a 16 × 16 mesh.

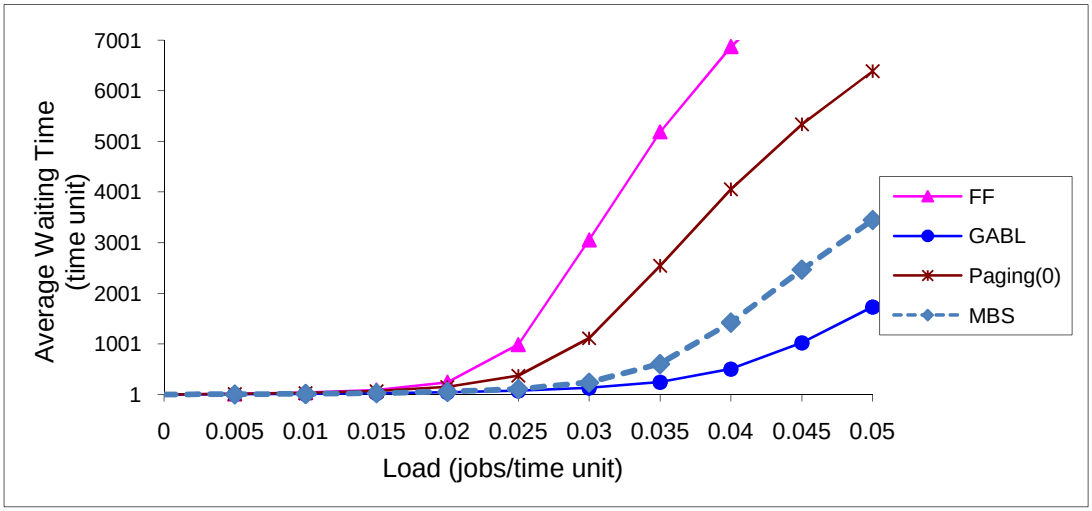


Figure 4.15: Average waiting time vs. System load for the all-to-all communication pattern and exponential side lengths distribution in a 16 × 16 mesh.

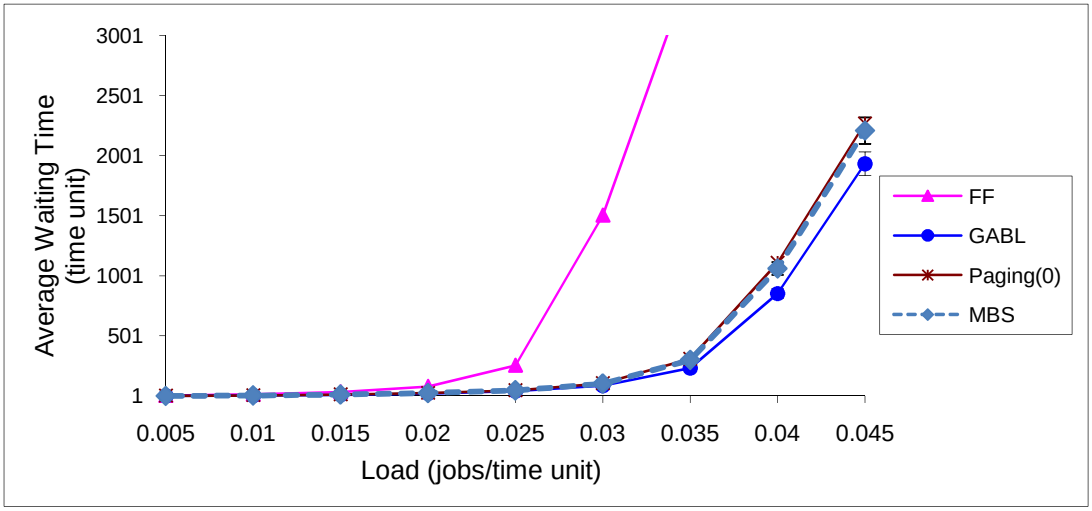


Figure 4.16: Average waiting time vs. System load for the random communication pattern and uniform side lengths distribution in a 16 × 16 mesh.

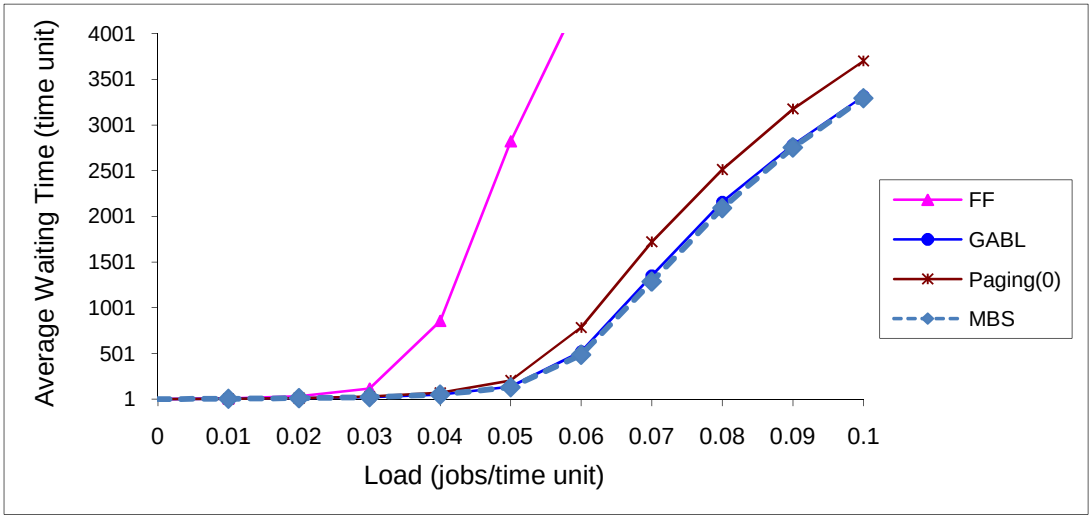


Figure 4.17: Average waiting time vs. System load for the random communication pattern and exponential side lengths distribution in a 16 × 16 mesh.

Utilisation:

Figures 4.18 and 4.19 depict the mean system utilisation of the allocation strategies (GABL, MBS, Paging(0), and FF) for the three communication patterns tested and job size distributions considered in this study. 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 utilisation. For both job size distributions, the non-contiguous allocation strategies achieve a mean system utilisation of 71% to 75%, but the contiguous FF strategy cannot exceed 50% utilisation. This is because contiguous allocation produces high external fragmentation, which makes allocation less likely to succeed. As a consequent, the mean system utilisation is lower. The utilisation of the three non-contiguous allocation strategies is approximately the same for both job size distributions. This is because the non-contiguous allocation strategies have the same ability to eliminate internal and external processor fragmentation. They always succeed to allocate processors to a job when the number of free processors is greater than or equal to the allocation request.

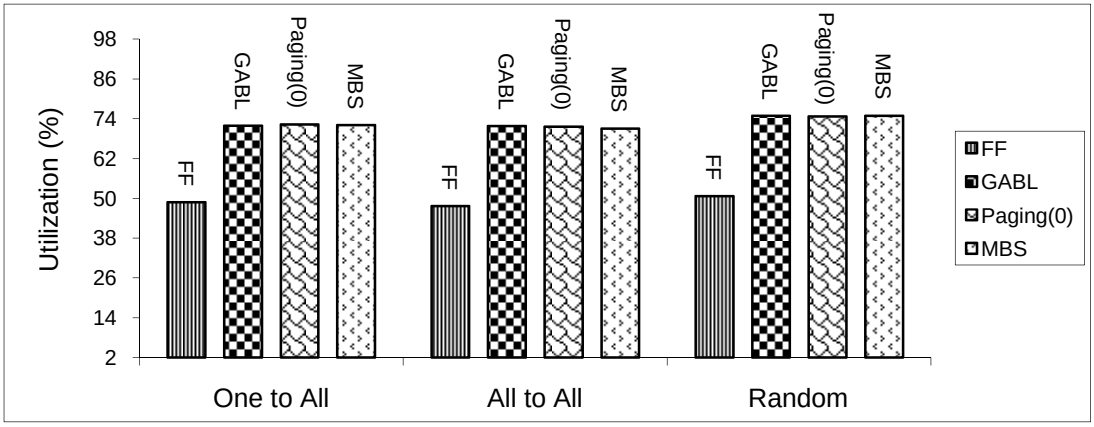


Figure 4.18: System utilisation of the non-contiguous allocation strategies (GABL, MBS, Paging(0)) and contiguous allocation strategy FF, for the three communication patterns tested, and uniform side lengths distribution in a 16 × 16 mesh.

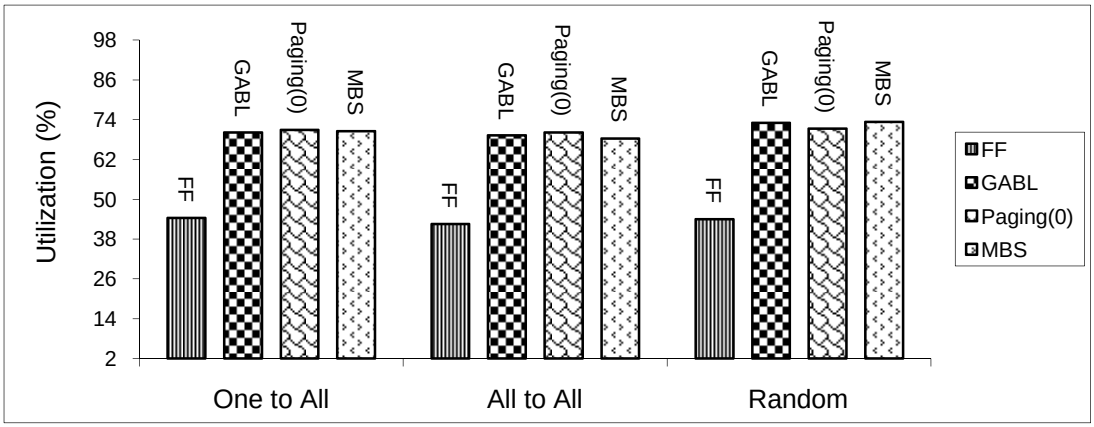


Figure 4.19: System utilisation of the non-contiguous allocation strategies (GABL, MBS, Paging(0)) and contiguous allocation strategy FF, for the three communication patterns tested, and exponential side lengths distribution in a 16 × 16 mesh.

Contiguous Ratio:

Figures 4.20 and 4.21 display the ratio of contiguous jobs of the non-contiguous allocation strategies (GABL, MBS, and Paging(0)) for the three communication patterns tested and heavy system loads that allow each allocation strategy to reach its upper limits of utilisation under both the uniform and exponential job size distributions. When the number of jobs that are allocated contiguously increases, the contention in the network decreases. This is because only messages generated by the same job are expected within a sub-mesh and therefore cause no inter-job contention in the network. The results reveal that GABL performs better than both MBS and Paging(0) strategies. For example, Figure 4.21 shows

that the ratio of jobs which allocated contiguously in GABL is 60% approximately while it is less than 5% for Paging(0) and less than 19% for MBS, so that GABL has a greater ability than the remaining strategies, MBS and Paging(0), to alleviate message contention in the network and hence achieves better performance than the previous non-contiguous allocation strategies in terms of average turnaround time. This conclusion is compatible with the values of the performance parameters shown above.

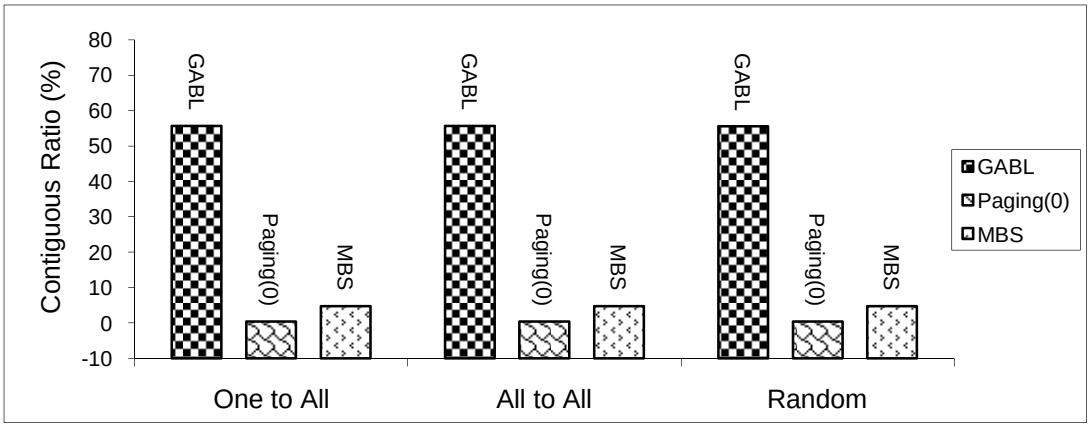


Figure 4.20: Percent of jobs allocated contiguously in the non-contiguous allocation strategies (GABL, MBS, Paging(0)), for the three communication patterns tested, and uniform side lengths distribution in a 16 × 16 mesh.

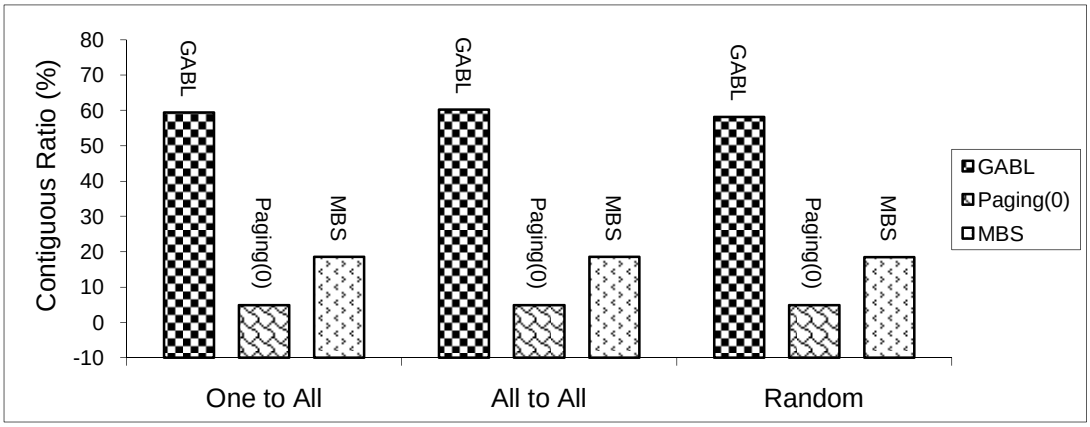


Figure 4.21: Percent of jobs allocated contiguously in the non-contiguous allocation strategies (GABL, MBS, Paging(0)), for the three communication patterns tested, and exponential side lengths distribution in a 16 × 16 mesh.

Average Blocks per Job:

In addition to the performance parameters shown above, we have measured another performance parameter for the non-contiguous allocation strategies that gave the best

performance (GABL and MBS), and that is the average blocks per job. It is defined as the average number of non-contiguous blocks allocated to a job in each strategy. The higher the average number of blocks the more likely it is that the job's messages visit nodes allocated to other jobs, potentially causing higher contention inside the network [85].

In Figures 4.22~4.27, the average blocks per job is plotted against the system load for the three communication patterns tested and for both job size distributions. The results reveal that GABL has a lower average blocks per job than MBS over all loads. In Figure 4.25, for example, the average blocks per job of GABL is 39%, 53%, and 75% of that of MBS when the job arrival rates are 0.015, 0.03, and 0.05 jobs/time unit, respectively. This conclusion is compatible with the values of the average turnaround times shown above.

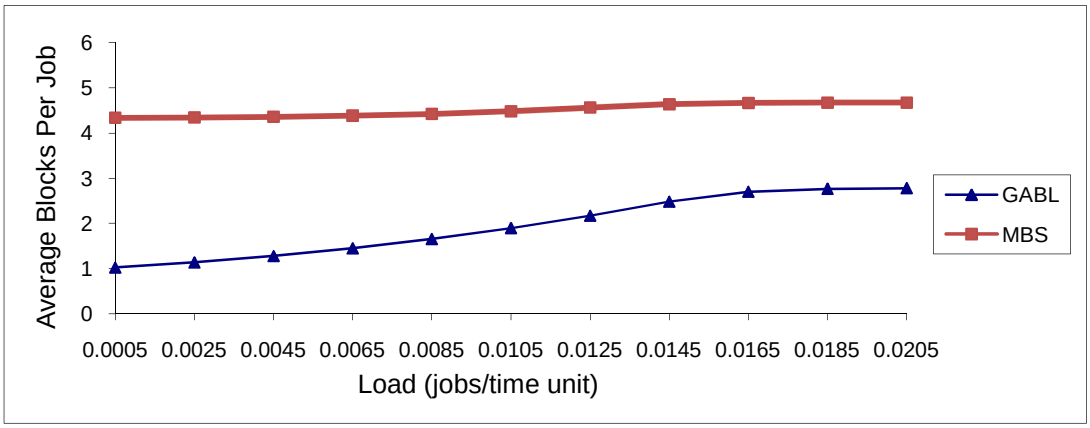


Figure 4.22: Average blocks per job vs. system load for the one-to-all communication pattern and uniform side lengths distribution.

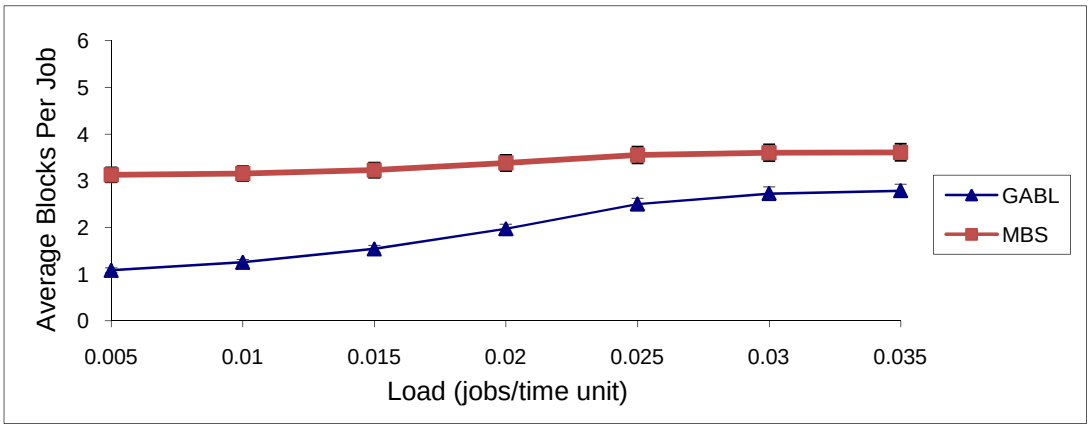


Figure 4.23: Average blocks per job vs. system load for the one-to-all communication pattern and exponential side lengths distribution.

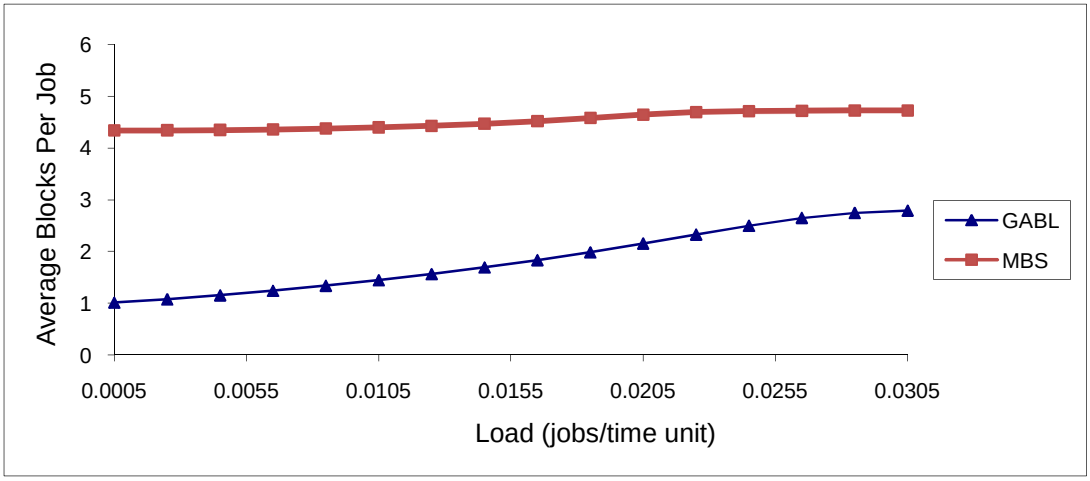


Figure 4.24: Average blocks per job vs. system load for the all-to-all communication pattern and uniform side lengths distribution.

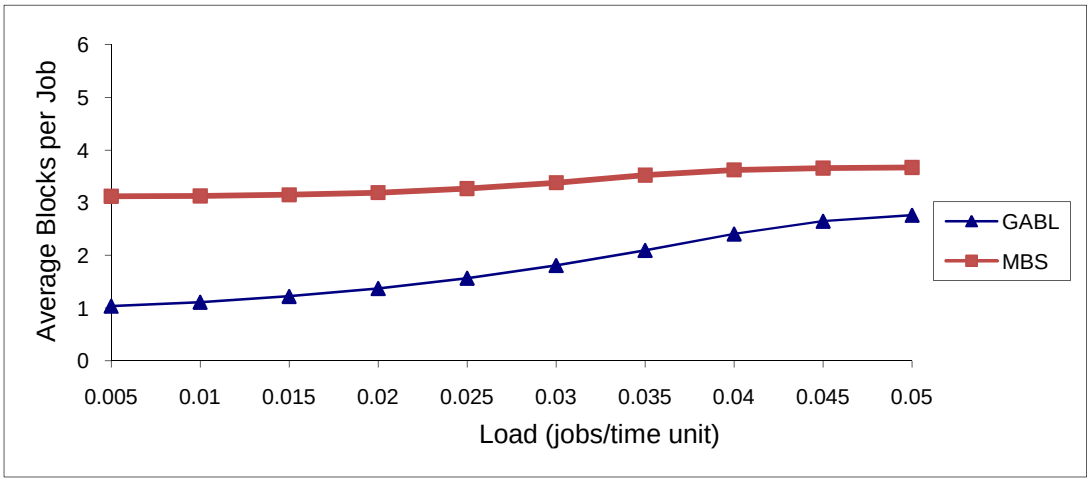


Figure 4.25: Average blocks per job vs. system load for the all-to-all communication pattern and exponential side lengths distribution.

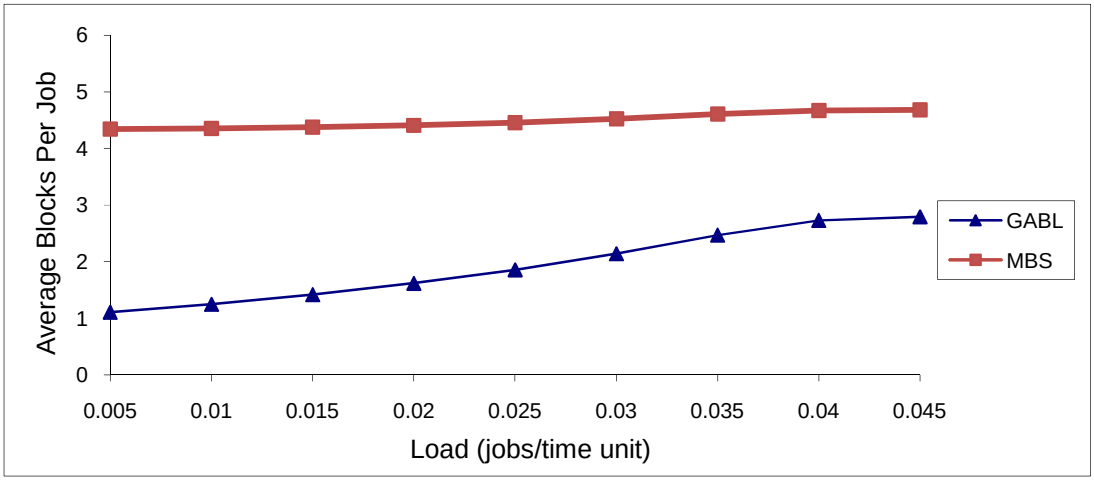


Figure 4.26: Average blocks per job vs. system load for the random communication pattern and uniform side lengths distribution.

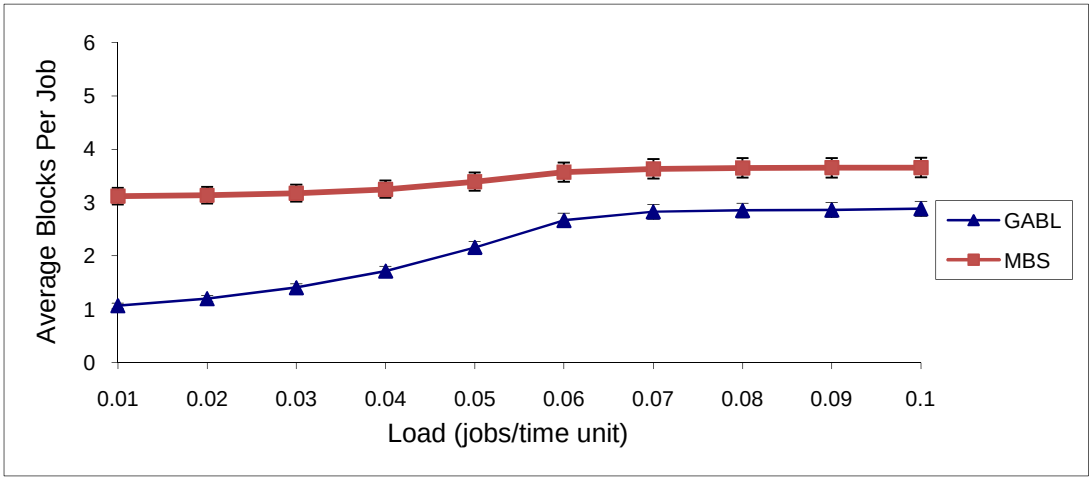


Figure 4.27: Average blocks per job vs. system load for the random communication pattern and exponential side lengths distribution.

Number of Allocated Sub-meshes (m) in the Busy List and the Number of Allocation Attempts (b) that Carried out in Step 6 in GABL Algorithm:

We have calculated the average number of allocated sub-meshes in the busy list (m) and the average number of allocation attempts (b) that were carried out in Step 6 in the GABL allocation algorithm. These experiments have been conducted to show that m and b are less sensitive to the size of the mesh system. In such experiments, different mesh sizes have been considered under both the uniform and exponential job size distributions.

In Figures 4.28~4.33, the average number of allocated sub-meshes (m) is plotted against the system load for the three communication patterns tested and for both job size distributions considered in this research. As expected, the average number of allocated sub-meshes is largest when the side lengths follow the exponential distribution. This is because the average sizes of jobs are smallest in this case. Moreover, and as discussed in Section 4.3.1 on the allocation and de-allocation time, the average number of allocated sub-meshes (m) is lower than n for both job size distributions and the three communication patterns tested under different mesh system sizes.

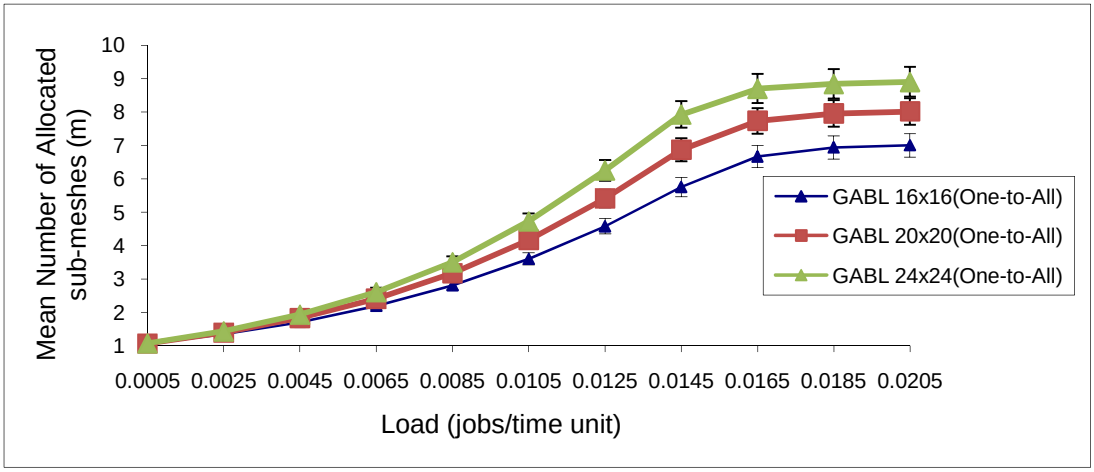


Figure 4.28: Average number of allocated sub-meshes (m) in GABL for the one-to-all communication pattern and uniform side lengths distribution in a 16 × 16 mesh, a 20 × 20 mesh, and a 24 × 24 mesh.

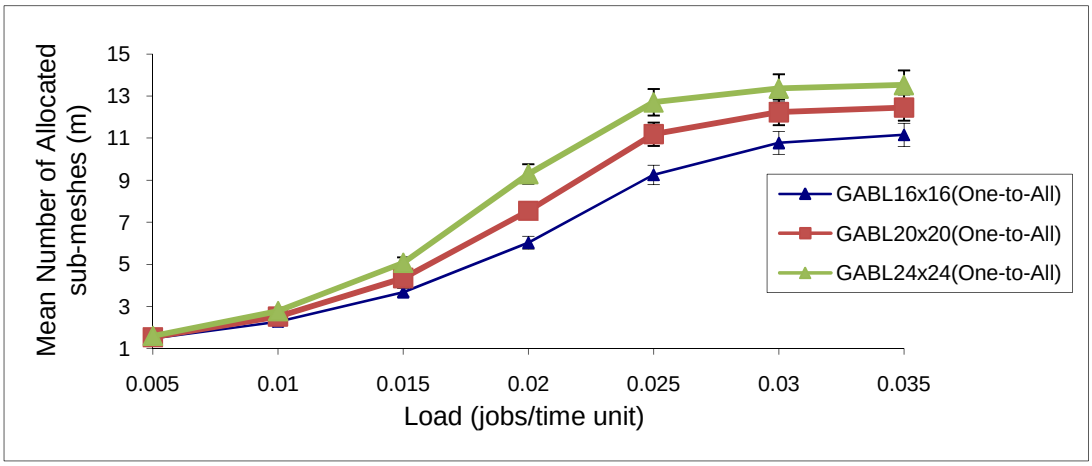


Figure 4.29: Average number of allocated sub-meshes (m) in GABL for the one-to-all communication pattern and exponential side lengths distribution in a 16 × 16 mesh, a 20 × 20 mesh, and a 24 × 24 mesh.

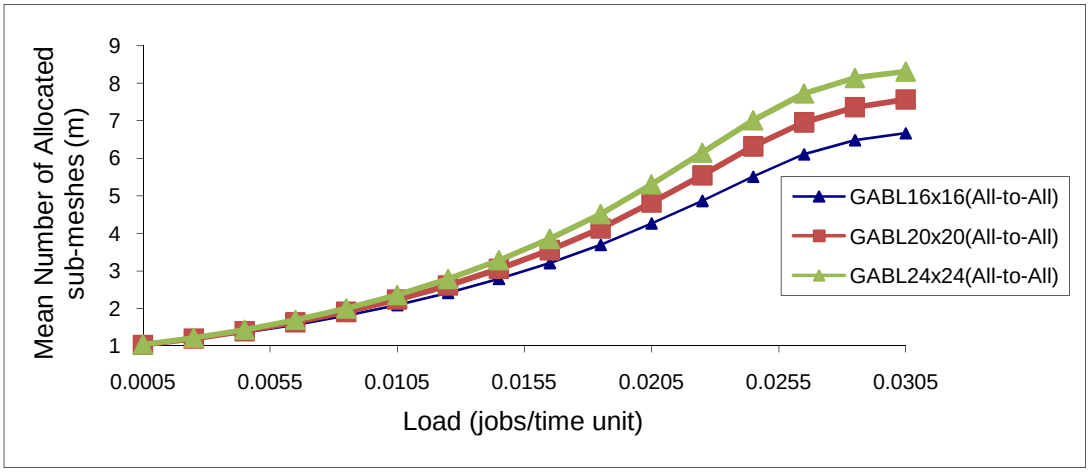


Figure 4.30: Average number of allocated sub-meshes (m) in GABL for the all-to-all communication pattern and uniform side lengths distribution in a 16 × 16 mesh, a 20 × 20 mesh, and a 24 × 24 mesh.

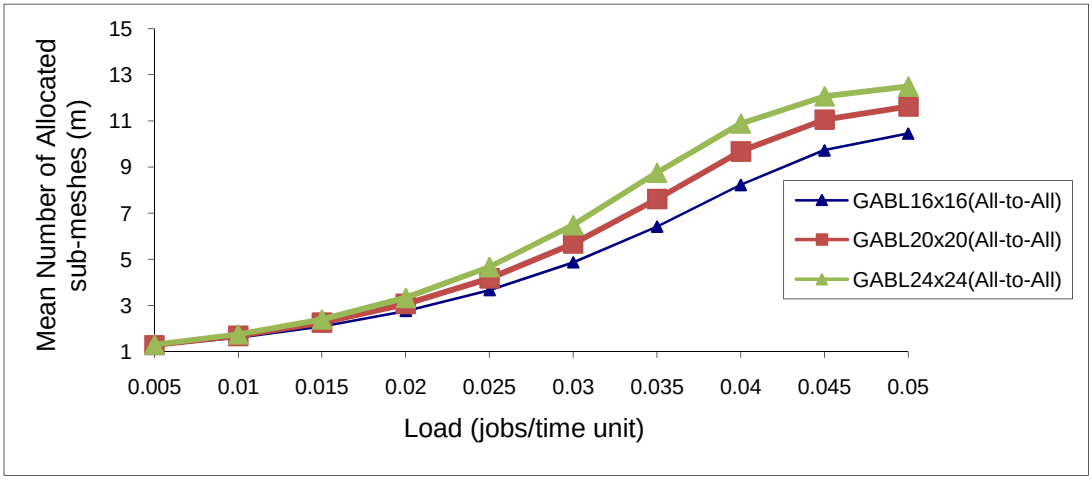


Figure 4.31: Average number of allocated sub-meshes (m) in GABL for the all-to-all communication pattern and exponential side lengths distribution in a 16 × 16 mesh, a 20 × 20 mesh, and a 24 × 24 mesh.

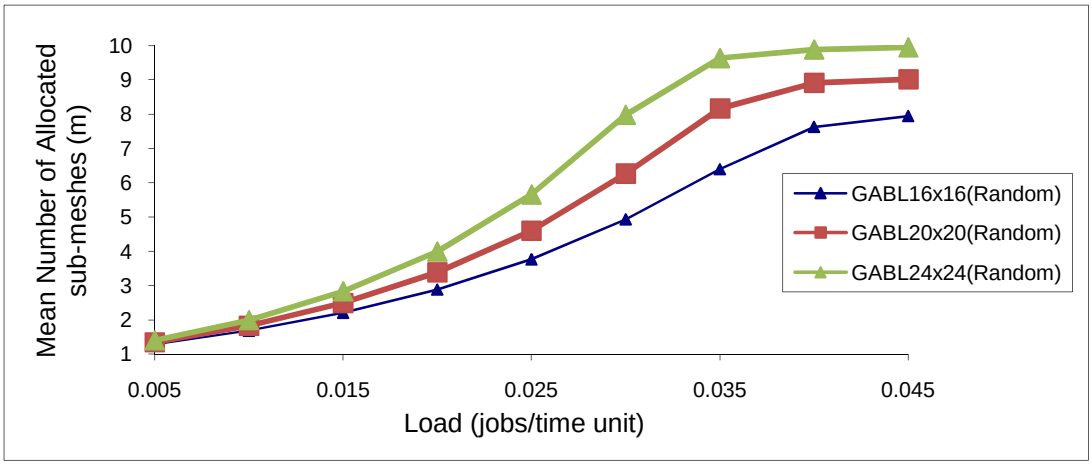


Figure 4.32: Average number of allocated sub-meshes (m) in GABL for the random communication pattern and uniform side lengths distribution in a 16 × 16 mesh, a 20 × 20 mesh, and a 24 × 24 mesh.

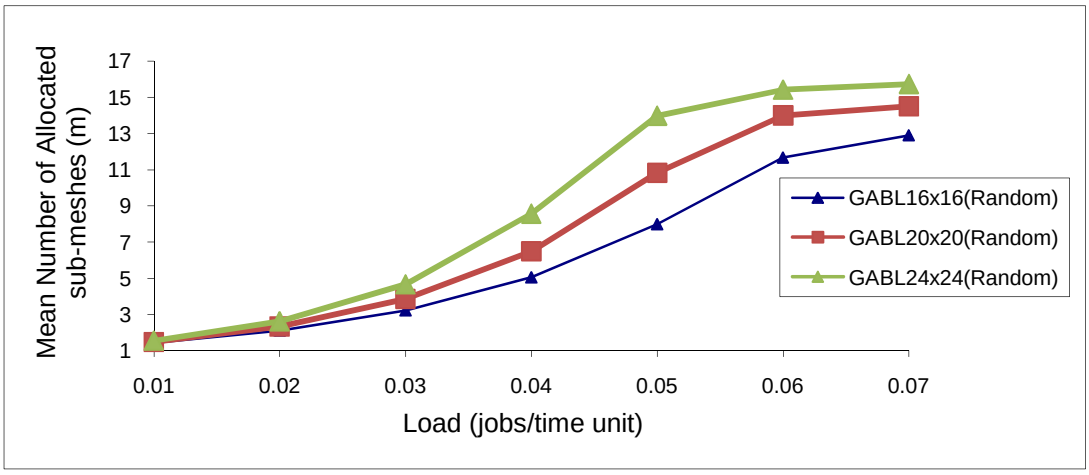


Figure 4.33: Average number of allocated sub-meshes (m) in GABL for the random communication pattern and exponential side lengths distribution in a 16 × 16 mesh, a 20 × 20 mesh, and a 24 × 24 mesh.

In Figures 4.34~4.39, the average number of allocation attempts (b) is plotted against the system load for both job size distributions and communication patterns tested. The results reveal that the average number of allocation attempts is lower than n for both job size distributions and the communication patterns considered in this study. Moreover, experiments conducted for larger mesh system sizes have revealed that b is less sensitive to the size of the mesh system (n) for the common job size distributions used in this study. Experiments that compute the average number of times Steps 4 and 5 are repeated have also been conducted. Their results lead to the conclusion that Step 6 dominates Steps 4 and 5 when the average case behaviour of the allocation algorithm is considered.

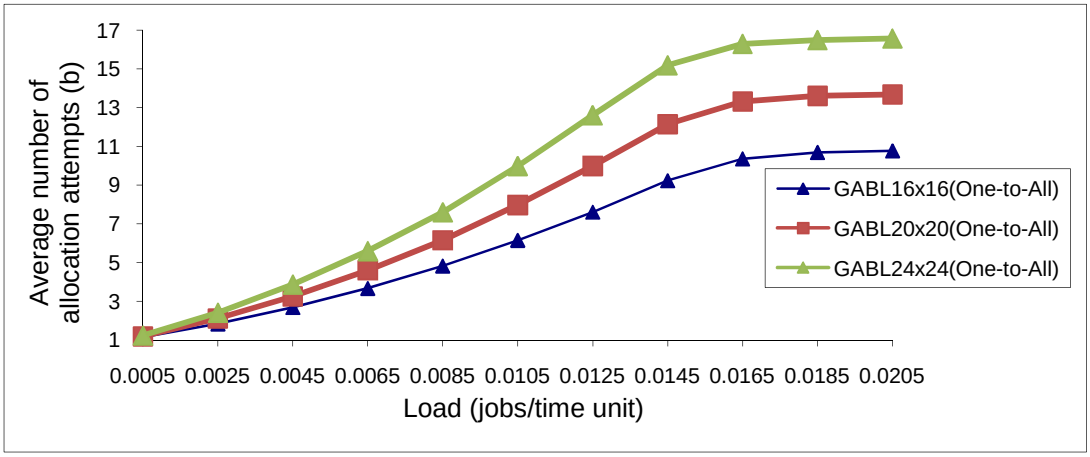


Figure 4.34: Average number of allocation attempts (b) in GABL for the one-to-all communication pattern and uniform side lengths distribution in a 16 × 16 mesh, a 20 × 20 mesh, and a 24 × 24 mesh.

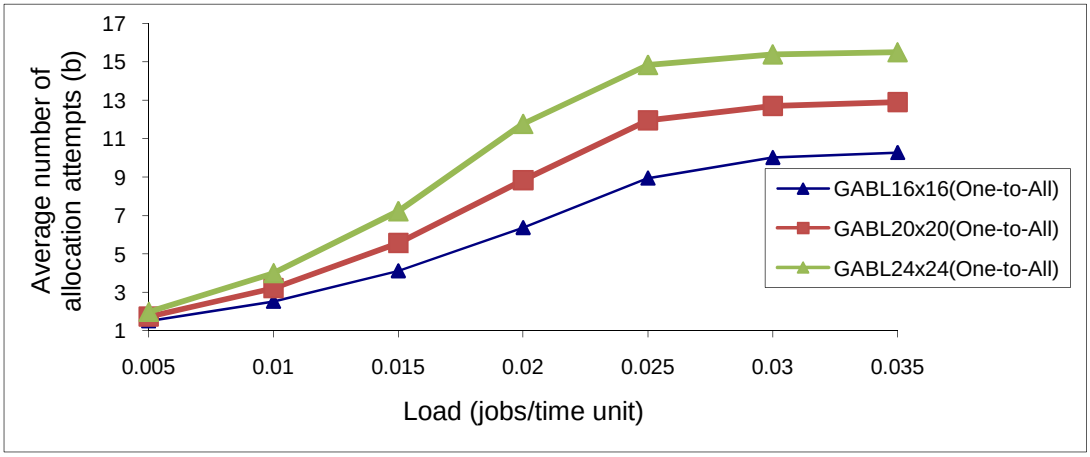


Figure 4.35: Average number of allocation attempts (b) in GABL for the one-to-all communication pattern and exponential side lengths distribution in a 16 × 16 mesh, a 20 × 20 mesh, and a 24 × 24 mesh.

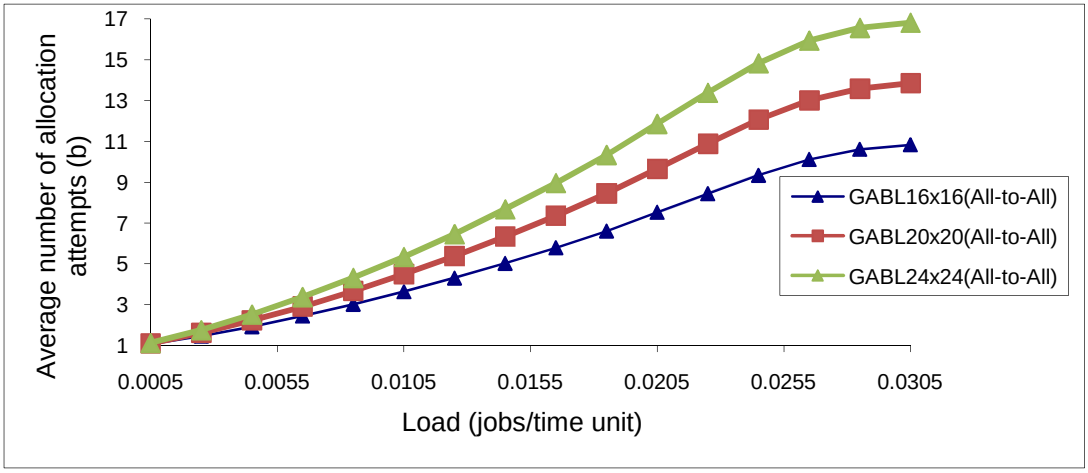


Figure 4.36: Average number of allocation attempts (b) in GABL for the all-to-all communication pattern and uniform side lengths distribution in a 16 × 16 mesh, a 20 × 20 mesh, and a 24 × 24 mesh.

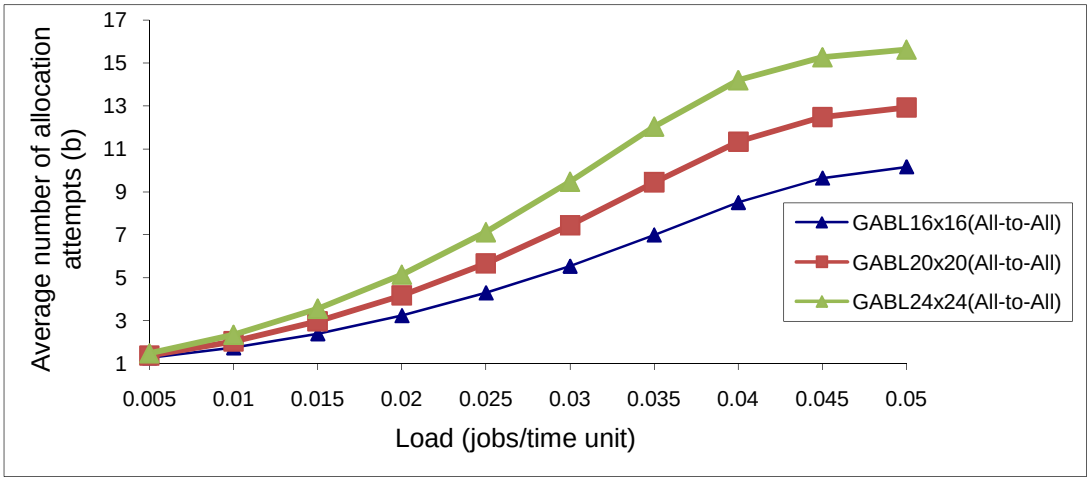


Figure 4.37: Average number of allocation attempts (b) in GABL for the all-to-all communication pattern and exponential side lengths distribution in a 16 × 16 mesh, a 20 × 20 mesh, and a 24 × 24 mesh.

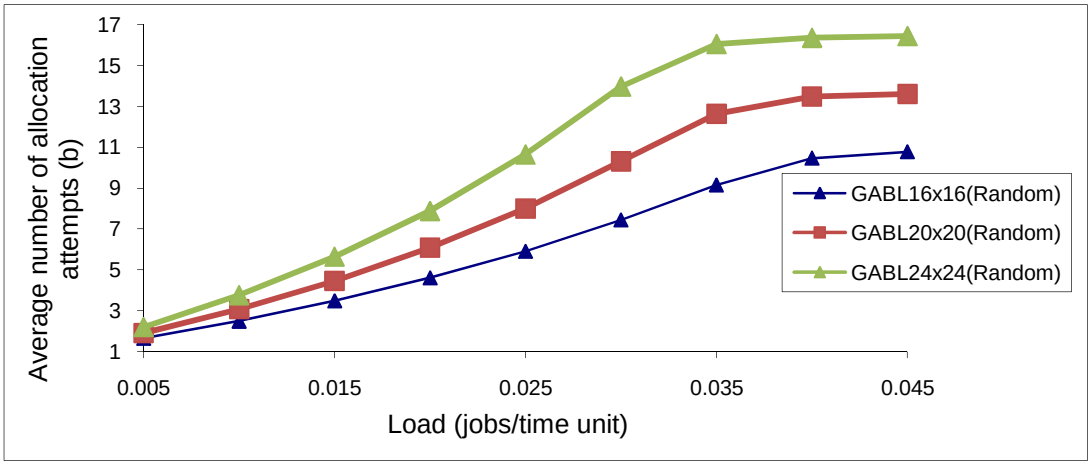


Figure 4.38: Average number of allocation attempts (b) in GABL for the random communication pattern and uniform side lengths distribution in a 16 × 16 mesh, a 20 × 20 mesh, and a 24 × 24 mesh.

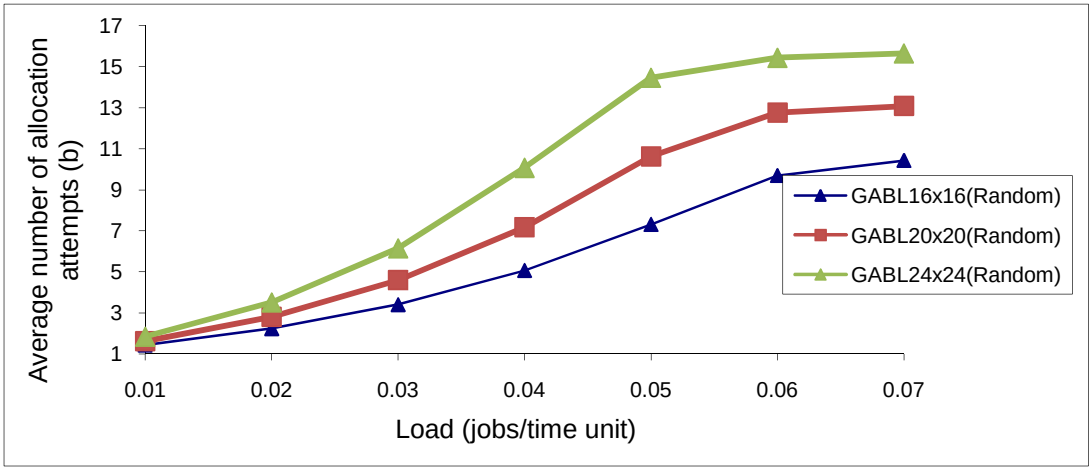


Figure 4.39: Average number of allocation attempts (b) in GABL for random communication pattern and exponential side lengths distribution in a 16 × 16 mesh, a 20 × 20 mesh, and a 24 × 24 mesh.

4.3.2.1 Performance Impact of Mesh System Size

In this section, we analyse the effects of the mesh system size on the performance of the allocation strategies in terms of average turnaround time of jobs. For the sake of conciseness, we have only concentrated on turnaround time in this Section because it is usually a good estimate of the performance of processor allocation strategies and it has been used in the existing allocation strategies [9, 18, 20, 27, 33, 34, 51, 52, 65, 78, 85, 99]. The parameters used in Section 4.3.2 are recalled here except the change regarding the mesh system size that is set to 16 × 16, 32 × 32, and 64 × 64 processor.

Figures 4.40~4.45 plot the average turnaround time of jobs against the size of the mesh system for both job size distributions considered in this chapter and all communication patterns tested assuming heavy system loads that allow each allocation strategy to reach its upper limits of utilisation. Figures 4.40 and 4.41 assume the one-to-all communication pattern. Figures 4.42 and 4.43 assume the all-to-all communication pattern, while Figures 4.44 and 4.45 assume a random communication pattern. The side lengths of the requested sub-meshes in these figures follow uniform and exponential distributions, respectively.

The results show that GABL performs better than all of the existing contiguous and non-contiguous allocation strategies for all mesh system sizes, except that in Figures 4.44 and 4.45 where the random communication pattern is examined. This is because the contention for the random communication pattern is smaller than that for the one-to-all and all-to-all communication patterns, as the destinations are chosen randomly and paths are less likely to overlap. Message contention that results from a random communication pattern is not sufficient for differentiating among the non-contiguous allocation strategies. For instance, Figure 4.42 shows that the average turnaround times of GABL are 20%, 24%, and 37% of that of FF, Paging(0), and MBS, respectively, for high loads and a 16 × 16 mesh system size, while for a 64 × 64 mesh system size and high loads, the average turnaround times of GABL are 23%, 34%, and 45% of that of FF, Paging(0), and MBS, respectively. Moreover, the results have shown that significant drops in performance with increasingly larger systems. In Figure 4.40, for instance, the average turnaround time of GABL for a 16 × 16 mesh system size is 34% of that for a 64 × 64 mesh system size. This is because when the system size increases, the allocated processors might be far from each other. This increases the distance traversed by messages, and as a result increases the communication overhead, leading to an increases in the turnaround time of jobs.

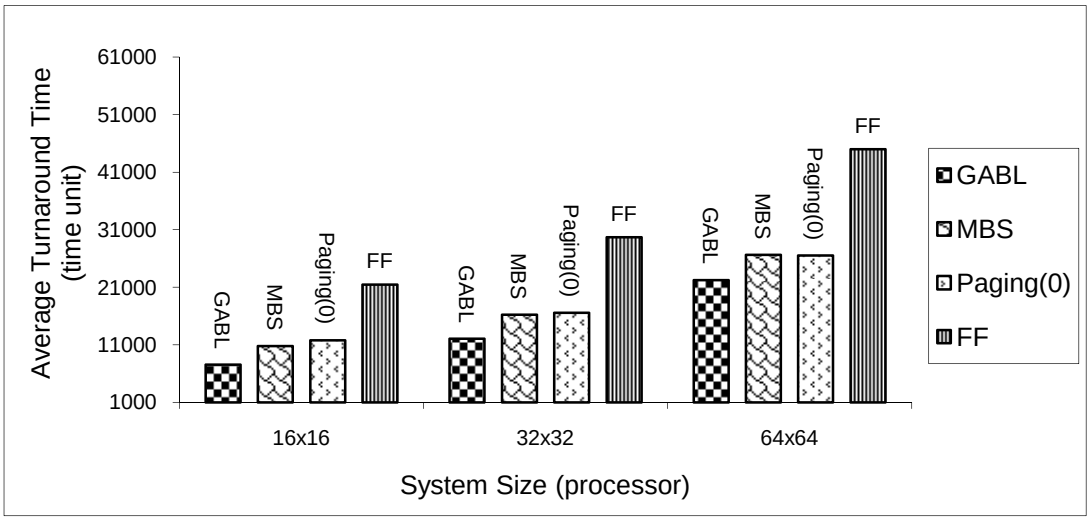


Figure 4.40: Average turnaround time vs. mesh system size for the one-to-all communication pattern and the uniform side lengths distribution.

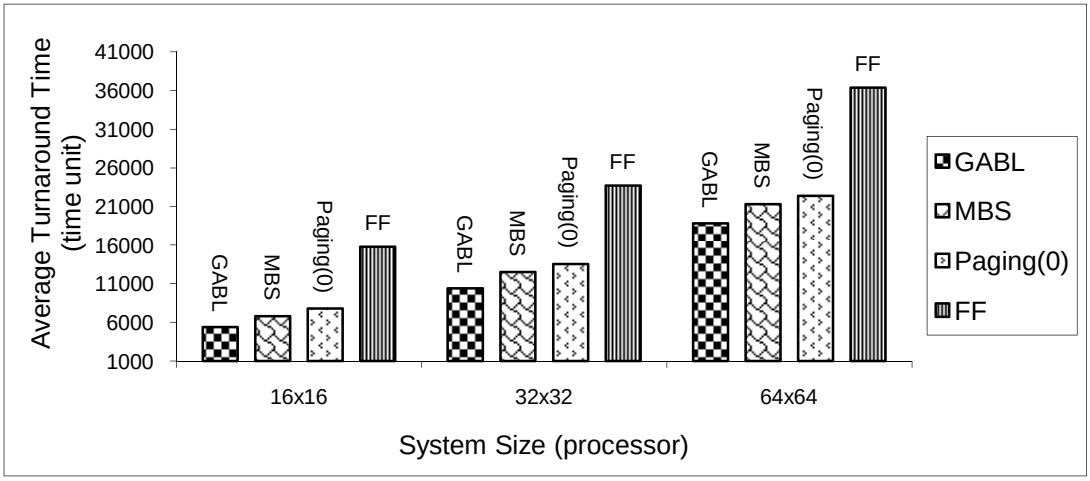


Figure 4.41: Average turnaround time vs. mesh system size for the one-to-all communication pattern and the exponential side lengths distribution.

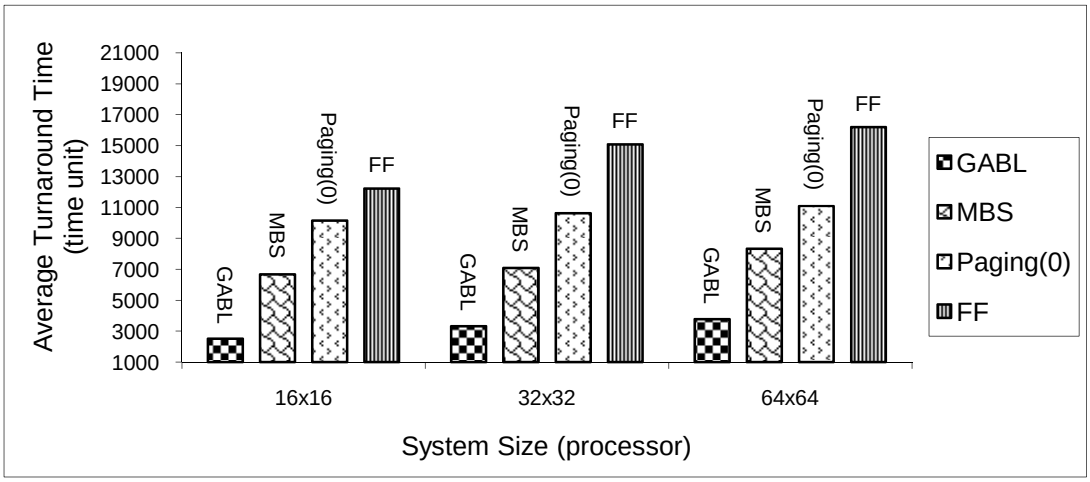


Figure 4.42: Average turnaround time vs. mesh system size for the all-to-all communication pattern and the uniform side lengths distribution.

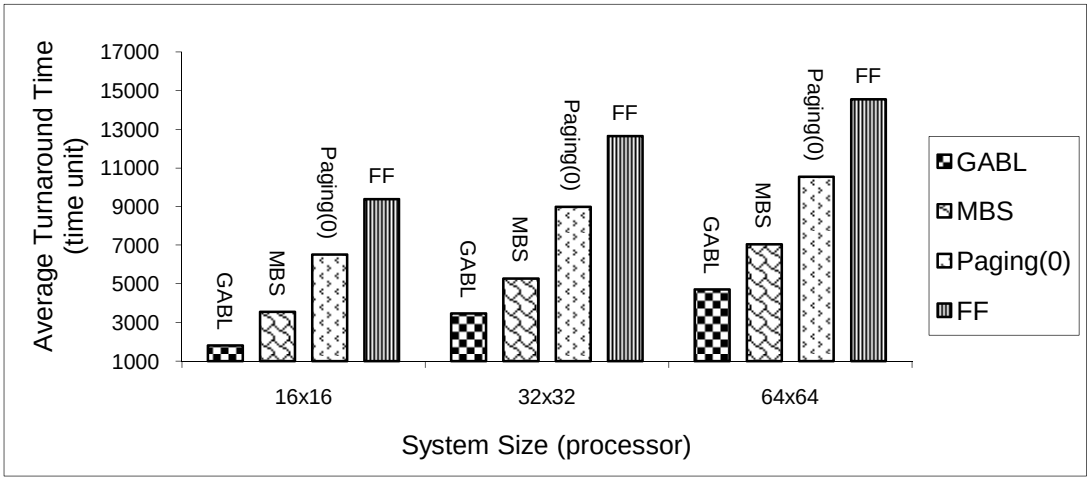


Figure 4.43: Average turnaround time vs. mesh system size for the all-to-all communication pattern and the exponential side lengths distribution.

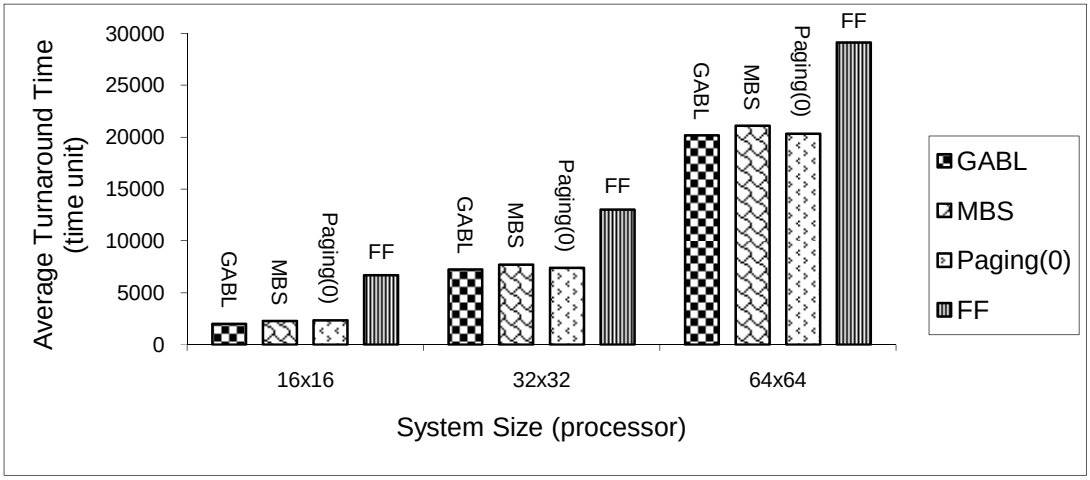


Figure 4.44: Average turnaround time vs. mesh system size for the random communication pattern and the uniform side lengths distribution.

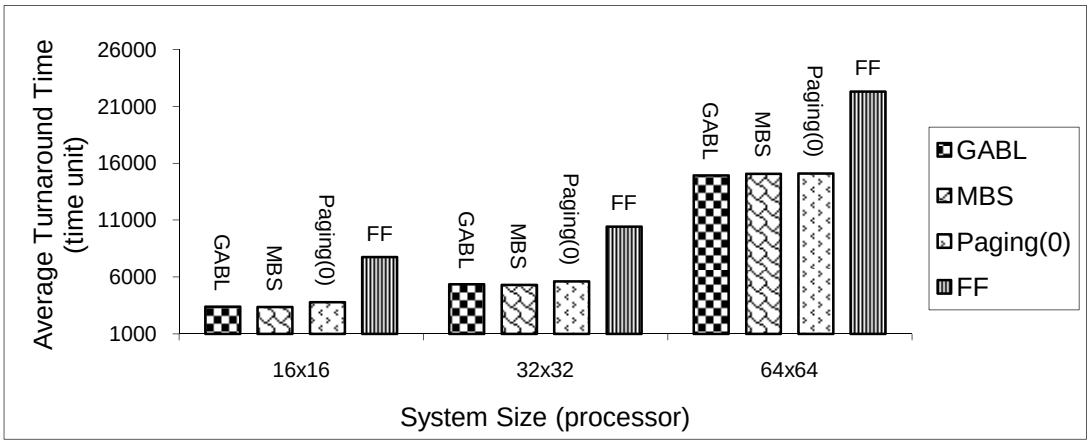


Figure 4.45: Average turnaround time vs. mesh system size for the random communication pattern and the exponential side lengths distribution.

4.3.2.2 Performance Impact of Packet Length

In this section, we investigate the effect of varying the packet length on the performance of the allocation strategies in terms of average turnaround time of jobs. As previously reported in Section 4.3.2.1, turnaround time has been chosen in this Section because it is usually a good estimate of the performance of processor allocation strategies and it has been used in the existing allocation strategies [9, 18, 20, 27, 33, 34, 51, 52, 65, 78, 85, 99]. The parameters used in Section 4.3.2 are recalled here, except for the change regarding the packet length that is set to 64 flits.

Figures 4.46 and 4.47 depict the performance of the allocation strategies in terms of turnaround times of jobs for the one-to-all communication pattern. The results have revealed that GABL has a lower turnaround time than all other contiguous and non-contiguous allocation strategies for both the exponential and uniform job size distributions. As previously reported in Section 4.3.2, the relative performance merits of the non-contiguous GABL strategy over the remaining contiguous and non-contiguous allocation strategies become more noticeable as the packet length increases. For example, in Figure 4.6 in Section 4.3.2 and for 8-flits packet length, the difference in performance in favour for GABL could be as large as 36% over Paging(0) and 30% over MBS for high loads while in Figure 4.46 and for 64-flits packet length, the difference in performance in favour for GABL could be as large as 45% over Paging(0) and 40% over MBS for high loads.

In Figures 4.48 and 4.49, the average turnaround times of jobs are plotted against the system load for the all-to-all communication pattern. Again, GABL performs much better than all other allocation strategies when the packet length increases for both job size distributions. Moreover, the difference in performance between GABL and the remaining non-contiguous strategies increases when the packet length increases. For example, in Figure 4.9 in Section 4.3.2 and for 8-flits packet length, the difference in performance in favour for GABL could be as large as 72% over Paging(0) and 49% over MBS for high loads while in Figure 4.49 and for 64-flits packet length, the difference in performance in favour for GABL could be as large as 85% over Paging(0) and 55% over MBS for high loads.

In Figures 4.50 and 4.51, the average turnaround times are plotted against the system load for the random communication pattern. As previously reported in Section 4.3.2, the contention for the random communication pattern is smaller than that for the one-to-all and all-to-all communication patterns. This is because destinations are chosen randomly and paths are less likely to overlap. Again, for larger packet sizes, the contention that results

from the random communication pattern is not sufficient for differentiating among the non-contiguous allocation strategies. As a consequence, the difference in performance between the non-contiguous strategies considered in this study is not changed by increasing the packet length.

To sum up, the above performance results demonstrate that GABL is the most flexible allocation strategy. Overall, it is superior to all other allocation strategies considered in this research; including when contention is heavy (the communication pattern is all-to-all).

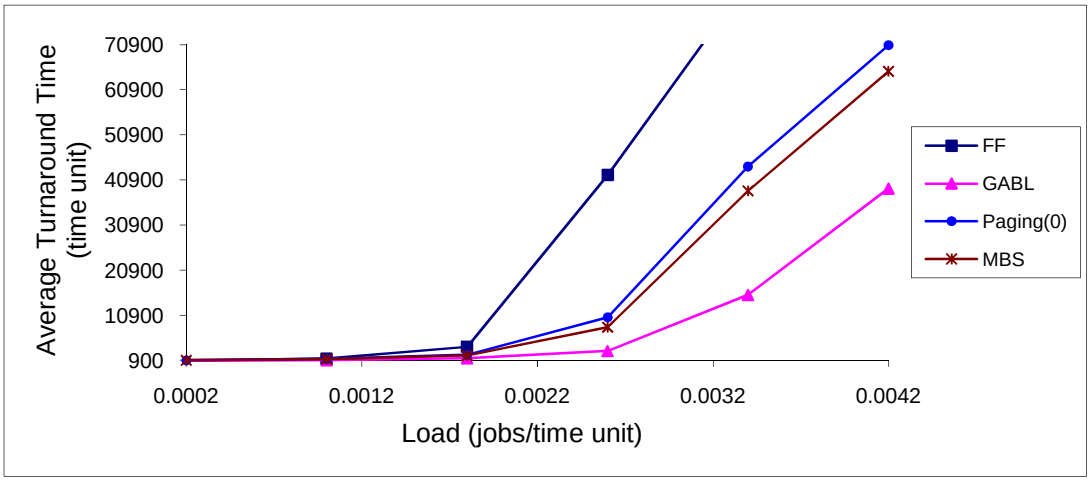


Figure 4.46: Average turnaround time vs. system load for the one-to-all communication pattern and uniform side lengths distribution with a 64-flits packet length in a 16 × 16 mesh.

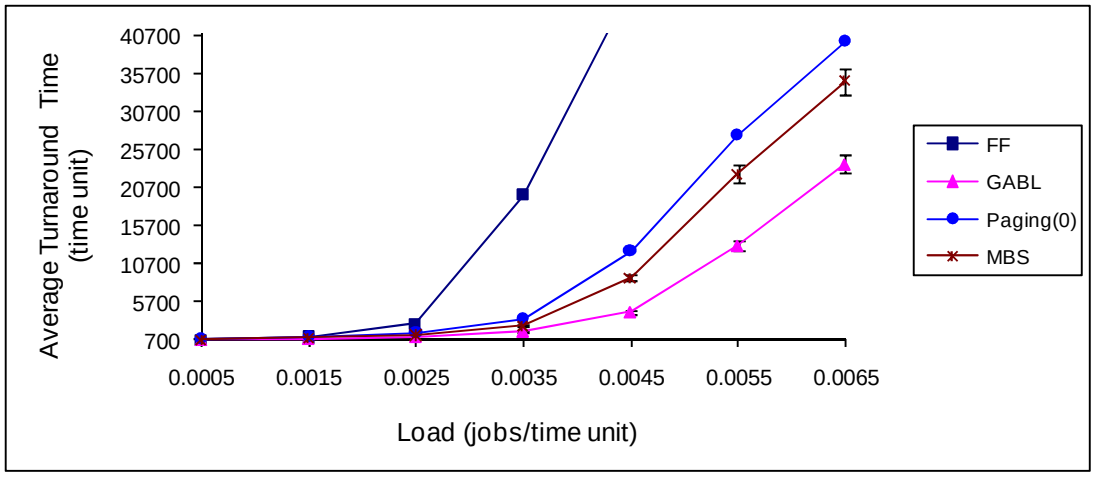


Figure 4.47: Average turnaround time vs. system load for the one-to-all communication pattern and exponential side lengths distribution with a 64-flits packet length in a 16 × 16 mesh.

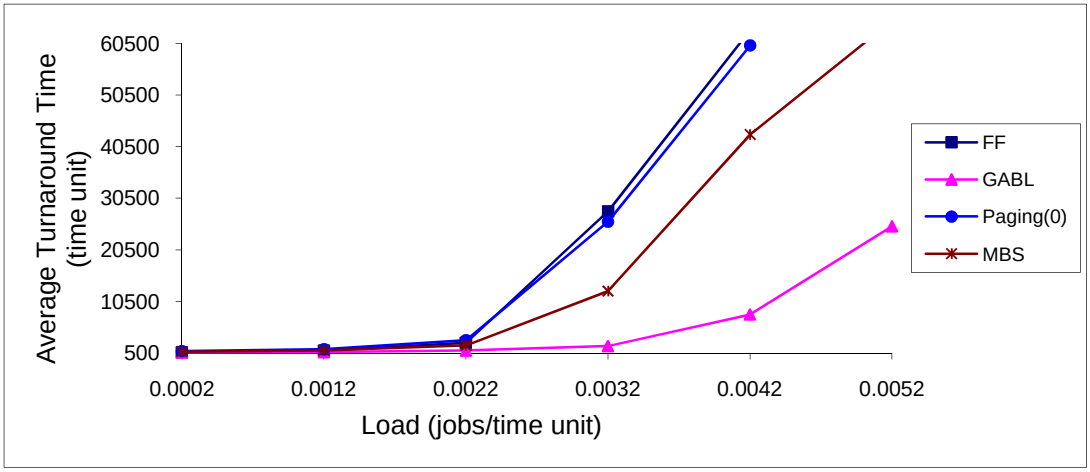


Figure 4.48: Average turnaround time vs. system load for the all-to-all communication pattern and uniform side lengths distribution with a 64-flits packet length in a 16 × 16 mesh.

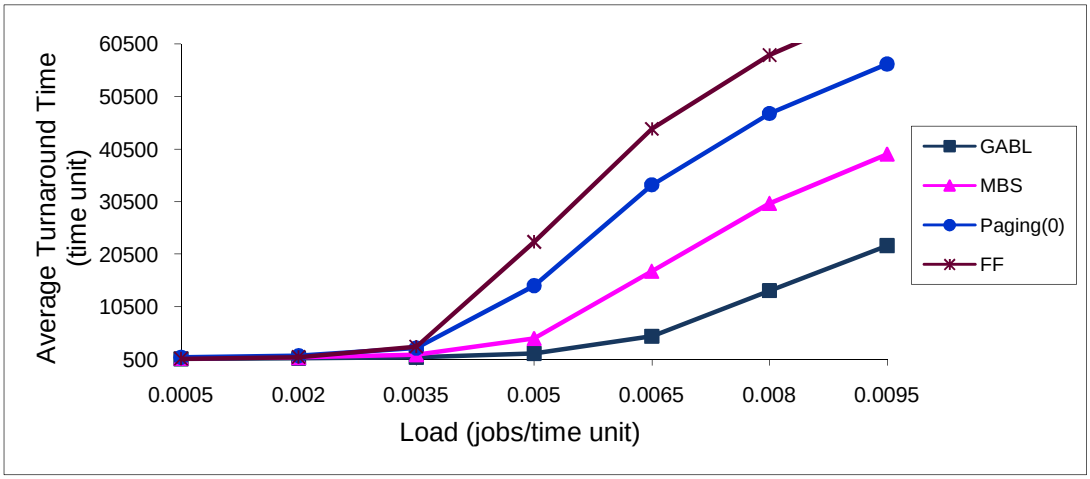


Figure 4.49: Average turnaround time vs. system load for the all-to-all communication pattern and exponential side lengths distribution with a 64-flits packet length in a 16 × 16 mesh.

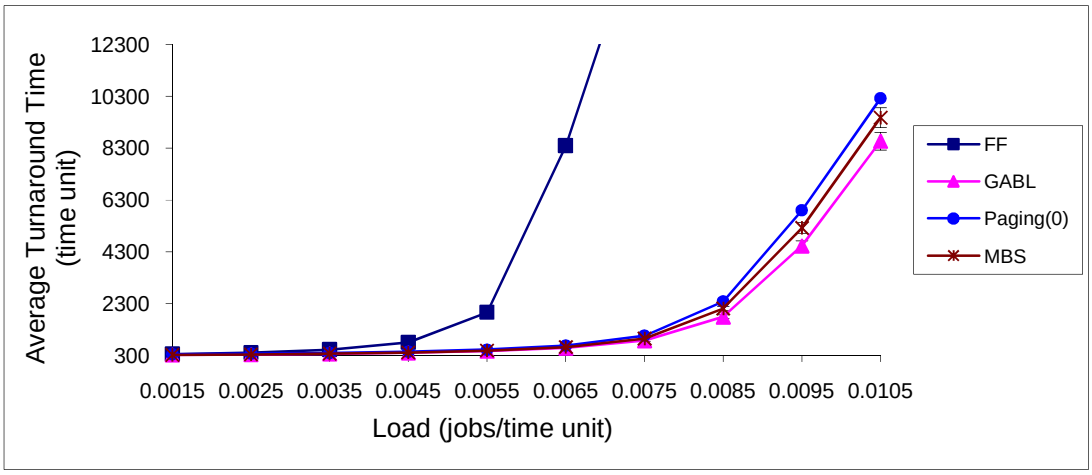


Figure 4.50: Average turnaround time vs. system load for the random communication pattern and uniform side lengths distribution with a 64-flits packet length in a 16 × 16 mesh.

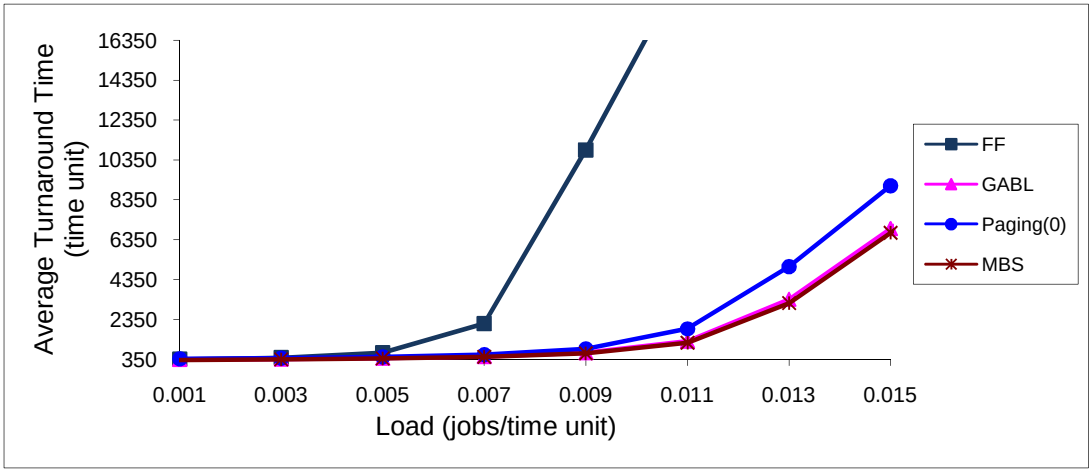


Figure 4.51: Average turnaround time vs. system load for the random communication pattern and exponential side lengths distribution with a 64-flits packet length in a 16 × 16 mesh.

4.4 Conclusions

This chapter has investigated the performance merits of the non-contiguous allocation in the 2D mesh network. To this end, we have suggested a new non-contiguous allocation strategy, referred to as Greedy Available Busy List (GABL for short), which differs from the earlier non-contiguous allocation strategies in the method used for partitioning allocation requests. The GABL strategy partitions the allocation requests based on the sub-meshes available for allocation. The major goal of the partitioning process is to maintain a high degree of contiguity among processors allocated to a job. This decreases the number of sub-meshes allocated to a job, and hence decreases the distance traversed by a message. This in turn decreases the communication overhead. GABL achieves this by using a busy list whose length is often small even when the size of the mesh scales up.

The performance of GABL has been compared against that of the existing non-contiguous and contiguous strategies. Simulation results have shown that GABL can greatly improve performance despite the additional message contention inside the network that results from the interference among the messages of different jobs. GABL also produces superior system

utilisation than its contiguous counterpart as it manages to eliminate both internal and external processor fragmentation. The results have also revealed that GABL is substantially superior over the previous well known non-contiguous allocation strategies, such as MBS and Paging(0), in terms of turnaround times. Furthermore, experiments for larger packet sizes and larger mesh system sizes have shown that GABL outperforms the previous contiguous and non-contiguous allocation strategies. Moreover, GABL can be efficient because it is implemented using a busy list approach. This approach can be expected to be efficient in practice because when the mesh system size increases the requirement of applications, in terms of the number of requested processors, often increases and in such a case our algorithm is often expected to exhibit competitive performance levels.

Chapter 5

Comparative Evaluation of Contiguous Allocation Strategies on Mesh-Connected Multicomputers

5.1 Introduction

The performance of contiguous allocation strategies can be significantly affected by the type of distribution adopted for job execution times [59]. The efficiency of the existing contiguous allocation strategies has typically been assessed under the assumption of exponentially distributed job execution times [27, 31, 33, 34, 35, 38, 48, 51, 52, 74, 78, 94, 99], which may not reflect all possible practical scenarios. For instance, a number of measurement studies [22, 47, 57, 58, 59, 88, 96] have convincingly shown that the execution times of certain computational jobs are better characterised by heavy-tailed execution times; that is, many jobs are short and fewer are long. The fewer jobs that have long execution time account for more than half of the total jobs' execution time [59]. Heavy-tailed distributions can capture this variability and have been shown to behave quite differently from the distributions more commonly used to evaluate the performance of allocation strategies (e.g., the exponential distribution) [22, 57, 58]. In particular, when sampling random variables that

follow a heavy-tailed distribution, the probability of large observations occurring is non-negligible. In order to gain a deeper understanding of the performance of the allocation strategies under various job execution time distributions, this chapter conducts an extensive comparison of the contiguous allocation strategies for 3D mesh-connected multicomputers, considering different mesh system sizes and various system loads.

Existing allocation strategies have typically been evaluated with the assumption of First-Come-First-Served (FCFS) job scheduling strategy [9, 11, 18, 20, 27, 31, 33, 34, 51, 52]. In this chapter, in addition to FCFS, a Shortest-Service-Demand (SSD) scheduling strategy is also adopted because it is expected to reduce performance loss due to FCFS blocking. SSD considers the shortest job to be the one having the shortest total processors service demand [63]. This strategy was found to improve system performance in a some previous studies [50, 73, 79].

Motivated by the above observations, this chapter makes the following contributions. We first compare the performance of the contiguous allocation strategy proposed in Chapter 3 as well as the existing contiguous allocation strategies for 3D mesh-connected multicomputers when subjected to heavy-tailed and exponential job execution times, respectively, under the FCFS strategy. We assess the effects of the heavy-tailed distribution on the performance of the contiguous allocation strategies for various system loads and different scheduling strategies and system sizes are investigated. To the best of our knowledge, this study is the first to consider heavy-tailed distributions in the context of processor allocation in mesh-connected multicomputers.

The performance of the allocation strategies is measured in terms of the usual performance parameters [27, 31, 33, 35, 73, 74, 77, 78, 79, 94, 99] including the average turnaround time

and mean system utilisation, as well as the mean measured allocation overhead, that accounts for the time required for the allocation and de-allocation of processors to jobs. The results presented below will reveal that the performance of the allocation strategies degrades when the distribution of job execution times are heavy-tailed. As a consequence, an appropriate scheduling strategy is required to deal with heavy-tailed distributions. Our analysis reveals that SSD exhibits superior performance than FCFS in terms of average turnaround time and mean system utilization.

The remainder of the chapter is organised as follows. Section 5.2 provides a brief overview of the allocation strategies whereas Section 5.3 provides a brief overview of the scheduling strategies considered in this chapter. Section 5.4 presents the results of the comparative performance study. Finally, Section 5.5 concludes this chapter.

5.2 Processor Allocation Strategies

The allocation strategies compared in this chapter cover a wide range of choices, including traditional First Fit (FF), Turning First Fit (TFF), a Busy List allocation strategy (BL) and the Turning Busy List allocation strategy (TBL).

The FF strategy [34] allocates the first available sub-mesh that is found, but it does not permit changing the orientation of the allocation requests, hence it suffers from high external processor fragmentation. The TFF strategy [34] improves performance by considering all possible orientations of the allocation request when needed, however its allocation overhead (i.e., allocation and de-allocation time) is high; FF and TFF strategies have been discussed in detail in Chapter 2 (please see Section 2.2.1). The BL strategy maintains a list of allocated sub-meshes to determine the nodes that cannot be used as base nodes for the requested sub-meshes and it reduces the allocation overhead that FF and TFF

suffer from, but it does not permit the orientation of the allocation request, hence it suffers from high processor fragmentation. The TBL strategy attempts to maintain good performance in terms of mean system utilisation and average turnaround time, by considering all the orientations of the allocation request when needed, with little allocation overhead. BL and TBL strategies have been discussed in detail in Chapter 3 (please see Section 3.3).

5.3 Job Scheduling Strategies

The order in which jobs are scheduled can have considerable effect on system performance [34, 73, 79]. The scheduling strategies used in this chapter include FCFS and SSD. In FCFS scheduling, the allocation request that arrives first is considered for allocation first. Allocation attempts stop when they fail for the current FIFO queue head. In SSD scheduling, the job with the shortest service demand is scheduled first [50, 73, 79].

Job scheduling is an important factor of processor allocation in multicomputers. For meshes, the results in [50, 73, 79] have shown that the SSD strategy results in significantly better performance than FCFS. In this chapter we show that SSD could be used with other mesh processor allocation strategies to yield improvement in performance in terms of average turnaround time and mean system utilisation.

The performance of the contiguous allocation can be significantly affected by both the type of the distribution adopted for job execution times and the scheduling strategy adopted for determining the order in which jobs are selected for execution. To illustrate this, the performance of the allocation strategies considered in this chapter has been evaluated in the context of a heavy-tailed distribution and both the FCFS and SSD strategies.

5.4 Simulation Results

Extensive simulation experiments have been carried out in order to compare the performance of the allocation strategies considered in this chapter, with and without change of request orientation. The performance analysis has been conducted using the same simulation model as outlined in Chapter 3 (please see Section 3.2).

The allocation and de-allocation algorithms, including the busy list routines, have been implemented in the C language, and integrated into the software ProcSimity; a simulation tool that is widely used for processor allocation and job scheduling in parallel systems [50, 66]. The target mesh is a cube with width W, depth D and height H. Jobs are assumed to have exponential inter-arrival times. They are scheduled using the FCFS and SSD strategies. FCFS is chosen because it is fair and it is widely used in other similar studies [6, 33, 51, 52, 73, 74, 77, 78, 79, 93], while SSD is used to avoid potential performance loss due to FCFS blocking [73, 79]. We assume that job execution times show some maximum values. As a consequence, job execution times are modelled by a Bounded Pareto [53] (exhibiting a heavy-tailed property but has an upper bound), which is defined as follows:

$$f(x) = \frac{\alpha k^{\alpha}}{1-(k/q)^{\alpha}} x^{-\alpha-1} (k \le x \le q) \quad \text{..... (5.1)}$$

where k and q are the lower and upper limits of the job execution time, and α is a factor that reflects the variability of job execution times. In the experiments, these parameters are set to $k = 15.0$, $q = 4241.0$ and $\alpha = 1.0$, as suggested in [53]. A Bounded Pareto distribution shows very high variability when $k << q$ and $\alpha \approx 1.0$. So, the values of k, q, and α have been chosen as above to show this variability. However, when α increases the

probability of large values decreases. For instance, when $\alpha = 3.0$ and $k = 1.0$ the Bounded Pareto distribution approaches the exponential distribution with a mean of 1 time unit.

As has been mentioned in the previous Chapters, two distributions are used to generate the width, depth and height of job requests. The first is the uniform distribution over the range from 1 to the mesh side length, where the width, depth and height of the job requests are generated independently. The second is the exponential distribution, where the width, depth and height of the job requests are exponentially distributed with a mean of half the side length of the entire mesh; the width, depth, and height of the job requests are rounded to the integer values using floor function and bounded by the dimensions of the mesh. These distributions have often been used in the literature [9, 20, 27, 33, 34, 35, 51, 52, 73, 74, 76, 77, 78, 79, 85, 94, 99]. Simulation parameters are illustrated in Table 5.1. It is worth noting that most of the values of these parameters have been adopted in the literature [9, 11, 20, 27, 33, 34, 38, 50, 51, 52, 53, 73, 77, 79, 85, 94, 99].

Table 5.1: The System Parameters Used in the Simulation Experiments

Simulator Parameter	Values
Dimensions of the Mesh Architecture	8 × 8 × 8, 10 × 10 × 10, and 12 × 12 × 12
Allocation Strategy	TBL, BL, TFF, and FF
Scheduling Strategy	FCFS and SSD
Job Size Distribution	Uniform: Job widths, depths, and heights are uniformly distributed over the range from 1 to the mesh side lengths.
	Exponential: Job widths, depths, and heights are exponentially distributed with a mean of half the side length of the entire mesh.

Execution Time Distribution	Bounded Pareto with the following parameters: $k = 15.0$, $q = 4241.0$ and $\alpha = 1.0$ [53].
Inter-arrival Time	Exponential with different values for the mean. The values are determined through experimentation with the simulator, ranged from lower values to higher values.
Number of Runs	The number of runs should be enough so that the confidence level is 95% that relative errors are below 5% of the means. The number of runs ranged from dozens to thousands.
Number of Jobs per Run	1000

Each simulation run consists of one thousand completed jobs. Simulation results are averaged over enough independent runs so that the confidence level is 95% that relative errors are below 5% of the means [7]. The batch means analysis has been used to calculate confidence intervals [4, 66]. This method has been discussed in detail in Chapter 3 (please see Section 3.4.1). Table 5.2 shows the grand means, confidence intervals, and relative errors that outline the results depicted, for example, in Figure 5.3 for the load 0.035 jobs/time unit under SSD. In most of the cases, the error bars are quite small. For the sake of clarity of the figures, the error bars are not shown on all the subsequent figures.

Table 5.2: The mean (i.e., mean turnaround time of job), 95% confidence interval, and relative error for the results shown in Figure 5.3 for the load 0.035 jobs/time unit and the SSD scheduling strategy

Algorithm	TBL	TFF	BL	FF
95% Confidence Interval	[572.11- 585.45]	[569.01- 588.22]	[657.04- 669.14]	[640.43- 660.82]
Mean (time unit)	578.781626	578.614877	663.090303	650.626269
Relative Error	0.011	0.016	0.009	0.015

The main performance parameters observed are the *average turnaround time* of jobs, *mean system utilisation* and *average allocation overhead*. As previously reported in Chapter 3, the turnaround time is the time that a parallel job spends in the mesh from arrival to departure. The utilisation is the percentage of processors that are utilized over time. The allocation overhead is the time that the allocation algorithm takes for the allocation and de-allocation operations per job. The important independent variable in the simulation is the *system load*. It is defined as the inverse of the mean inter-arrival time of jobs. Its range of values from low to heavy loads has been determined through experimentation with the simulator allowing each allocation strategy to reach its upper limits of utilisation.

In what follows, the notation <allocation strategy>(<scheduling strategy>) is adopted to represent the strategies in the performance figures. For instance, TBL(SSD) refers to the Turning Busy List allocation strategy under the Shortest-Service-Demand scheduling strategy.

5.4.1 Performance Comparison under Heavy-Tailed and Exponential Job Execution Times with the FCFS Scheduling Strategy.

To evaluate the impact of heavy-tailed distribution on the performance of the allocation strategies, its performance is compared, in terms of the average turnaround time of jobs and mean system utilisation when the job execution times follow heavy-tailed distribution according to the values specified in Table 5.1, against that of the exponential job execution times with a mean of 83 time units. Figure 5.1 depicts the average turnaround time of the allocation strategies (TBL, TFF, BL, and FF) for the heavy-tailed and exponential job execution times and FCFS scheduling strategy under uniform side lengths distribution. The simulation results in this figure are presented for a heavy system load that allows each allocation strategy to reach its upper limits of utilisation. The results reveal that the performance of the allocation strategies degrades when the distribution of job execution

times is heavy-tailed. This is because the long jobs' execution times, resulting from the heavy-tailed distribution, increase the average turnaround time and consequently lead to a degradation in system performance. For example, the average turnaround time of TBL(FCFS) under exponential job execution time distribution is 49% of that of TBL(FCFS) under heavy-tailed job execution time distribution.

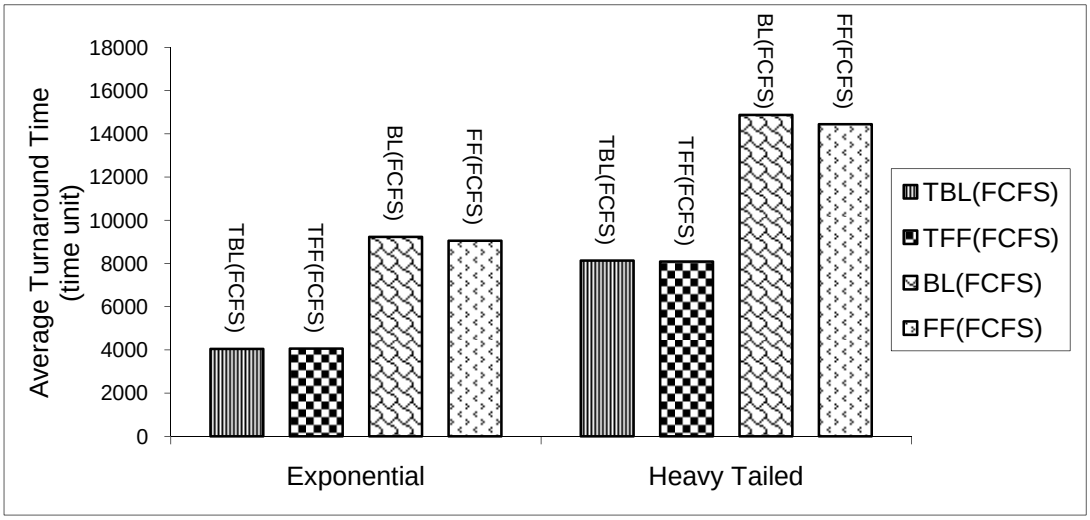


Figure 5.1: Turnaround time in BL, FF, TBL, and TFF under the exponential and heavy-tailed job execution times with FCFS scheduling strategy and the uniform side lengths distribution in an 8 × 8 × 8 mesh.

Figure 5.2 depicts the mean system utilisation of the strategies (TBL, TFF, BL, and FF) for the heavy-tailed and exponential job execution times with FCFS and uniform side lengths distribution. The simulation results in this figure 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 utilisation. The results reveal that the utilisation of the allocation strategies degrades when job execution times follow heavy-tailed distribution, while it is better for the exponential job execution times. This is because the long jobs' execution times due to the heavy-tailed distribution decrease the probability of successful allocation to other jobs, and this in turn degrades system performance. For example, the allocation strategies with rotation, as in TBL and TFF, achieve a mean system utilisation of 49% for exponential

job execution times, but cannot exceed 39% for heavy-tailed job execution times.

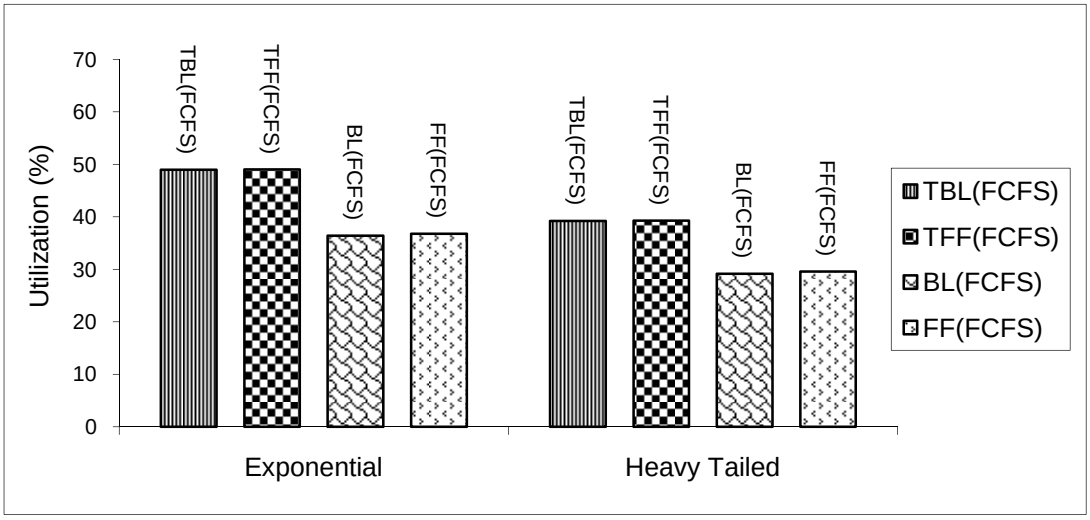


Figure 5.2: Mean system utilisation in BL, FF, TBL, and TFF under the exponential and heavy-tailed job execution times with FCFS scheduling strategy and the uniform side lengths distribution in an 8 × 8 × 8 mesh.

5.4.2 Performance Comparison under Different System Loads and Scheduling Strategies

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. The results obtained have been found to be similar to those observed when other mesh system sizes are considered (please see Section 5.4.3).

Turnaround Time:

In Figures 5.3 and 5.4, the average turnaround time of jobs is plotted against the system load for both job size distributions and the two scheduling strategies considered. The results reveal that the allocation strategies with rotation under SSD scheduling (TBL(SSD) and TFF(SSD)) have comparable performance, and that they are superior to all other strategies. They are followed, in order, by the strategies BL(SSD), FF(SSD), TBL(FCFS), TFF(FCFS), BL(FCFS), and FF(FCFS). When compared to TBL(SSD) and TFF(SSD) in Figure 5.3,

BL(SSD) increases the average turnaround times by about 13% and 48% for the loads 0.03 and 0.045 jobs/time unit, respectively. In Figure 5.4, the increases are by about 21% and 32% for the loads 0.075 and 0.105 jobs/time unit, respectively.

It can also be seen in the figures that the average turnaround times of the strategies that depend on a list of allocated sub-meshes for both allocation and de-allocation, as in TBL and BL, is very close to that of the strategies that depend on the allocation states of processors, as in TFF and FF, assuming that the same scheduling strategy is used. For example, the average turnaround time of TBL(SSD) is very close to that of TFF(SSD). It can also be seen in the figures that the average turnaround time of the strategies with rotation, as in TBL and TFF, is substantially superior to that of the strategies without rotation, as in BL and FF, because it is more likely that a suitable contiguous sub-mesh is available for allocation to a job when request rotation is allowed. It can also be noticed in the figures that SSD is much better than FCFS. In Figure 5.3, for instance, the average turnaround time of TBL(SSD) is 7% of that of TBL(FCFS) in the presence of high loads. This finding demonstrates that the scheduling and allocation strategies both have substantial effect on the performance of the contiguous allocation strategies in the 3D mesh.

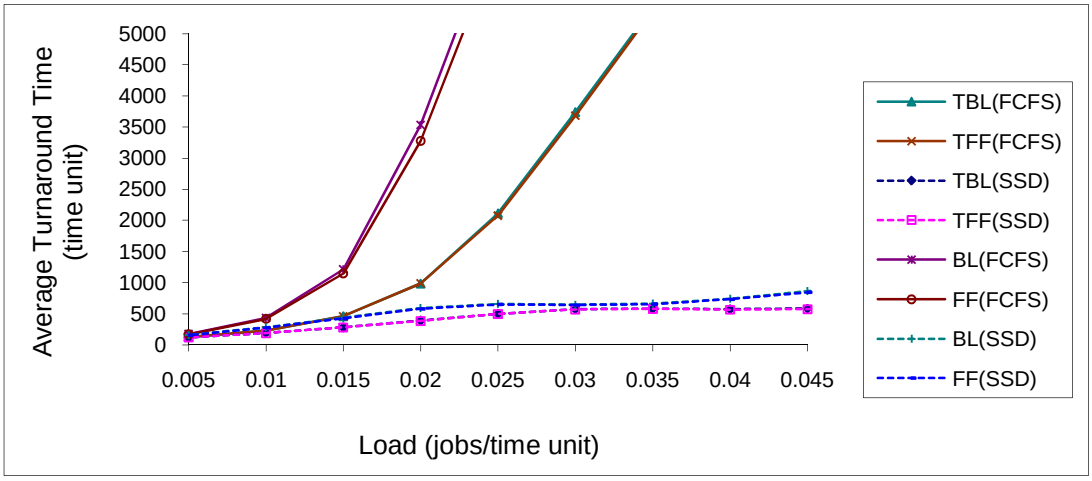


Figure 5.3: Average turnaround time vs. system load for the contiguous allocation strategies (BL, FF, TBL, TFF) under the scheduling strategies (FCFS and SSD) and the uniform side lengths distribution in an 8 × 8 × 8 mesh.

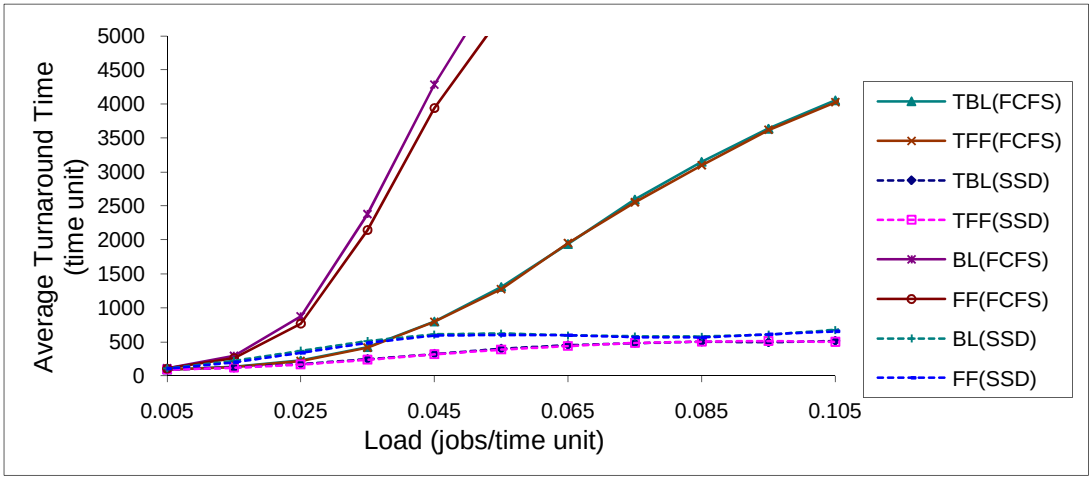


Figure 5.4: Average turnaround time vs. system load for the contiguous allocation strategies (BL, FF, TBL, TFF) under the scheduling strategies (FCFS and SSD) and the exponential side lengths distribution in an 8 × 8 × 8 mesh.

Utilisation:

In Figures 5.5 and 5.6, the mean system utilisation of the allocation strategies is plotted against the system loads under the uniform and exponential job size distributions, respectively, and both scheduling strategies considered. In these two figures, TBL(SSD) and TFF(SSD) have almost identical performance, and they are superior to the other strategies. In Figure 5.5, for example, TBL(SSD) achieves system utilisation of 52%, but TBL(FCFS) cannot exceed 39% system utilisation. Also, the results show that the switching request orientation improves performance substantially. This is indicated by the largely superior mean system utilisation of the allocation strategies that can switch the orientation of allocation requests when they are compared to the strategies without rotation. The strategies with rotation, as in TBL(SSD) and TFF(SSD), achieve system utilisation of 44% under the exponential distribution and 52% under uniform distribution. But the strategies without rotation, as in BL(SSD) and FF(SSD), cannot exceed 42% utilisation. Higher system utilisation is achievable under heavy loads because the waiting queue is filled very early, allowing each allocation strategy to reach its upper limits of utilisation.

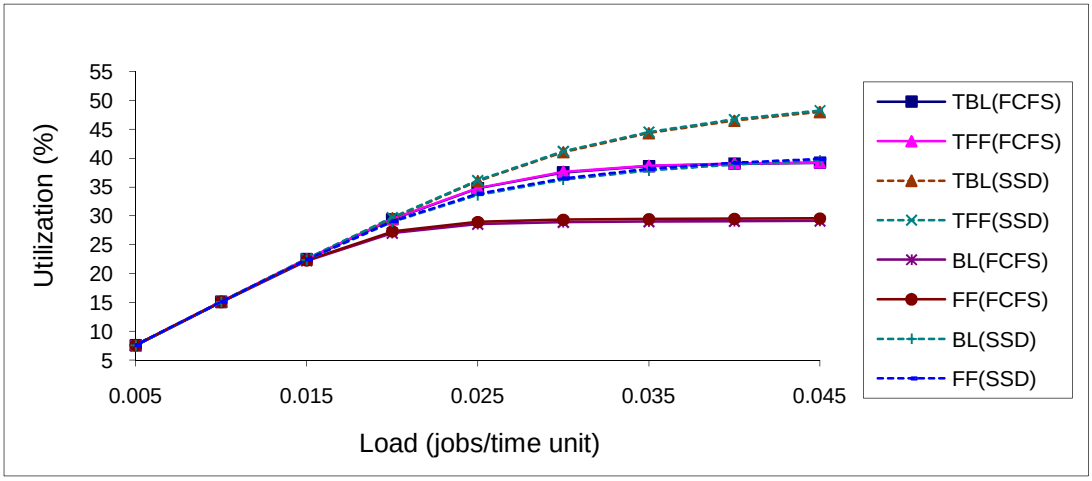


Figure 5.5: Mean System utilisation for the contiguous allocation strategies (BL, FF, TBL, TFF) under the scheduling strategies (FCFS and SSD) and the uniform side lengths distribution in an 8 × 8 × 8 mesh.

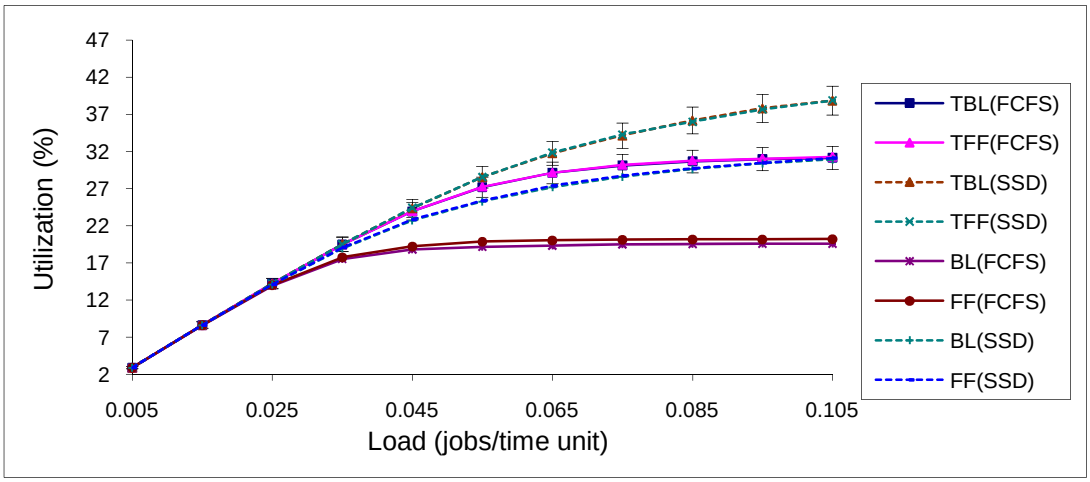


Figure 5.6: Mean System utilisation for the contiguous allocation strategies (BL, FF, TBL, TFF) under the scheduling strategies (FCFS and SSD) and the exponential side lengths distribution in an 8 × 8 × 8 mesh.

Number of Allocated Sub-meshes (m):

In Figures 5.7~5.10, the average number of allocated sub-meshes in the strategies that depend on a list of allocated sub-meshes for both allocation and de-allocation (TBL and BL) is plotted against the system load. Different mesh sizes are considered under both job size distributions and scheduling strategies examined in this study. As expected, the average number of allocated sub-meshes is largest when the side lengths follow the exponential distribution. This is because the average sizes of jobs are smallest in this case. Moreover, the

average number of allocated sub-meshes is much lower than the number of processors in the mesh system (n) for both job size distributions. Figure 5.7 depicts that the average number of allocated sub-meshes in the busy list varied from 1.19 to 2.22 for the uniform side lengths distribution and FCFS scheduling, and from 1.19 to 3.03 for the uniform side lengths distribution and SSD scheduling. In Figure 5.8, the average number of allocated sub-meshes varied from 1.22 to 4.72 for the exponential side lengths distribution and FCFS, and from 1.22 to 6.62 for the exponential side lengths distribution and SSD. It can be seen in the figures that m is often less sensitive with n. It can also be noticed that the average number of allocated sub-meshes under SSD is higher than that under FCFS. In Figure 5.7, for example, the average number of allocated sub-meshes of TBL(FCFS) for all mesh sizes are 84% and 75% of that of TBL(SSD) under the job arrival rates 0.04 and 0.105 jobs/time unit, respectively. This is because in SSD, the job with the shortest service demand is scheduled first, meaning that allocation and de-allocation operations are more numerous within a given time period, resulting in more allocated sub-meshes in the busy list.

As previously reported in Chapter 3, the average number of allocated sub-meshes for the TBL strategy that use the rotation of the allocation request is a bit higher than that of the BL strategy that does not use the rotation of the allocation request. This is because it is highly likely that a suitable contiguous sub-mesh is available for allocation to a job when the request orientation is allowed, which in turn increases the number of allocated sub-meshes in the busy list. In Figures 5.7 and 5.9, the average number of allocated sub-meshes of BL(FCFS) for all mesh system sizes is 74% of that of TBL(FCFS) under the job arrival rate 0.105 jobs/time unit, and the average number of allocated sub-meshes of BL(SSD) for all mesh system sizes is 80% of that of TBL(SSD) when the job arrival rate is 0.105 jobs/time unit.

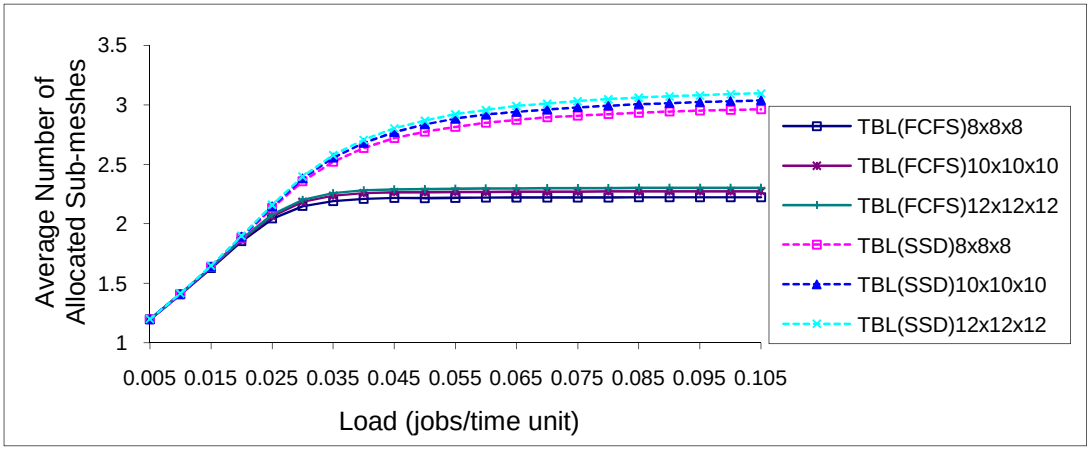


Figure 5.7: Average number of allocated sub-meshes (m) in TBL under the scheduling strategies (FCFS and SSD) and the uniform side lengths distribution in 8 × 8 × 8, 10 × 10 × 10 and 12 × 12 × 12 meshes.

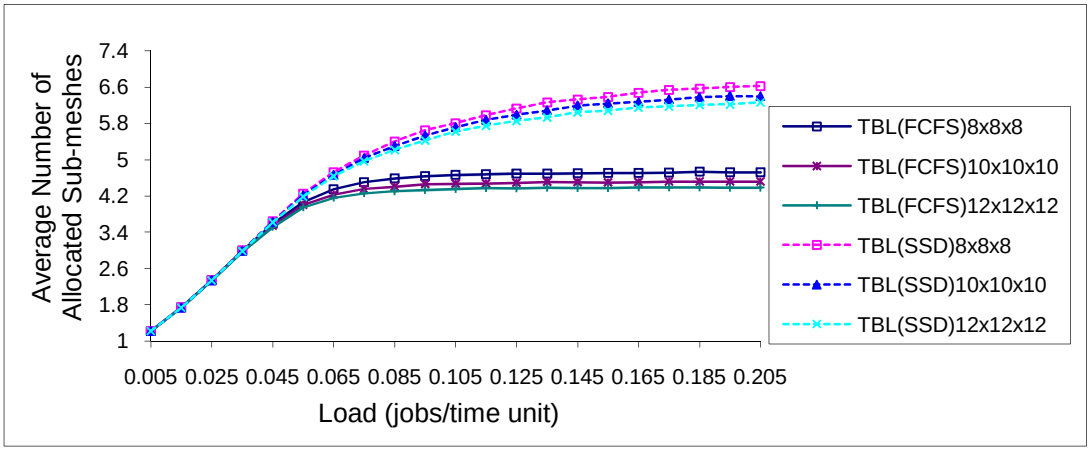


Figure 5.8: Average number of allocated sub-meshes (m) in TBL under the scheduling strategies (FCFS and SSD) and the exponential side lengths distribution in 8 × 8 × 8, 10 × 10 × 10 and 12 × 12 × 12 meshes.

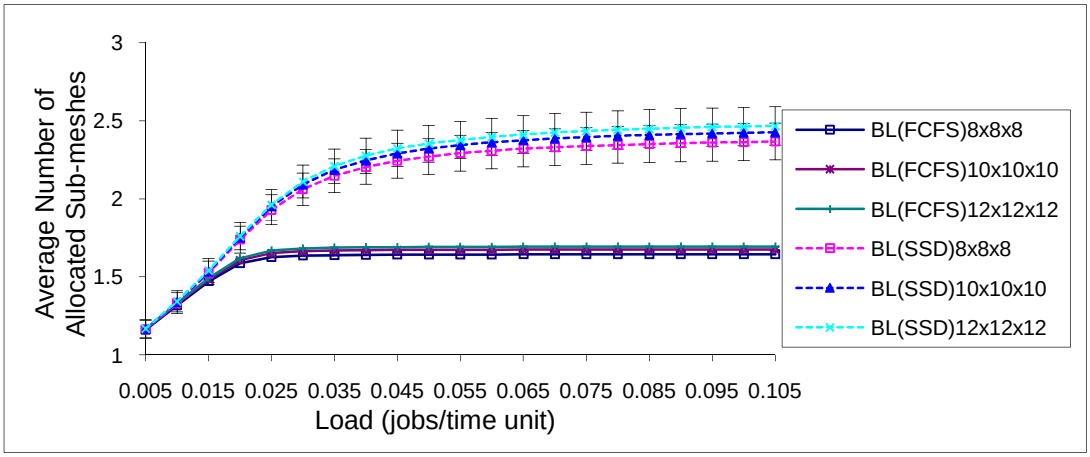


Figure 5.9: Average number of allocated sub-meshes (m) in BL under the scheduling strategies (FCFS and SSD) and the uniform side lengths distribution in 8 × 8 × 8, 10 × 10 × 10 and 12 × 12 × 12 meshes.

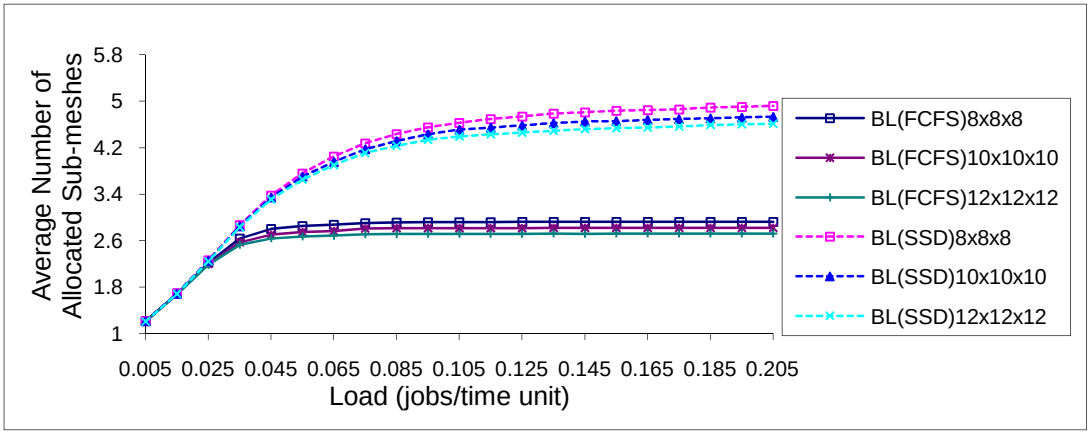


Figure 5.10: Average number of allocated sub-meshes (m) in BL under the scheduling strategies (FCFS and SSD) and the exponential side lengths distribution in 8 × 8 × 8, 10 × 10 × 10 and 12 × 12 × 12 meshes.

Allocation Overhead (Allocation and De-allocation Time):

Figures 5.11~5.18 show the average allocation and de-allocation time (*allocation overhead*) for the allocation strategies considered against the job arrival rate for an 8 × 8 × 8, a 10 × 10 × 10, and a 12 × 12 × 12 system sizes, when the request side lengths follow the uniform and exponential distributions, respectively. We observe that the strategies that depend on a list of allocated sub-meshes for both allocation and de-allocation, as in TBL and BL, have much smaller allocation overhead than the strategies that depend on the number of processors in the mesh system, as in TFF and FF, under both scheduling strategies considered.

In Figure 5.11, for example, the allocation overhead of TBL(FCFS) is 4% of that in TFF(FCFS) under the job arrival rate 0.075 jobs/time unit. It can also be seen in the figures that the time needed for both allocation and de-allocation for the strategies with rotation, as in TBL and TFF, is higher than that of the strategies without rotation, as in BL and FF. This is because in the worst case, the allocation process, in the allocation strategies with rotation, is repeated for all possible permutations (6 permutations) of the job request while this process is repeated only one time for the strategies without rotation. In Figures 5.11 and 5.13, for example, the allocation overhead of BL(SSD) is 37% of that in TBL(SSD) under

the job arrival rate 0.105 jobs/time unit.

The average size of a requested sub-mesh is relatively small when the exponential distribution is used for generating job side lengths. Therefore, the number of allocated sub-meshes is larger in this case, meaning that the allocation choices are more numerous. Consequently, the time needed for both the allocation and de-allocation operations of the allocation strategies that depend on a list of allocated sub-meshes is largest when the side lengths follow the exponential distribution.

Also and as shown in Figures 5.15~5.22, when the number of processors increases the allocation overhead increases for the strategies that depend on the number of processors in the mesh system, as in TFF and FF, while it does not increase for the strategies that depend on a list of allocated sub-meshes, as in TBL and BL. In Figures 5.12 and 5.20, for example, the allocation overhead of TFF(SSD) for an 8 × 8 × 8 mesh system size is 11% of that in TFF(SSD) for a 12 × 12 × 12 mesh system size under the job arrival rate 0.205 jobs/time unit. Moreover, it can be noticed in the figures that the difference in allocation and de-allocation time becomes more significant as the system load increases. Thus, the allocation strategies that depend on a list of allocated sub-meshes are more effective than the strategies that depend on the size of the mesh system.

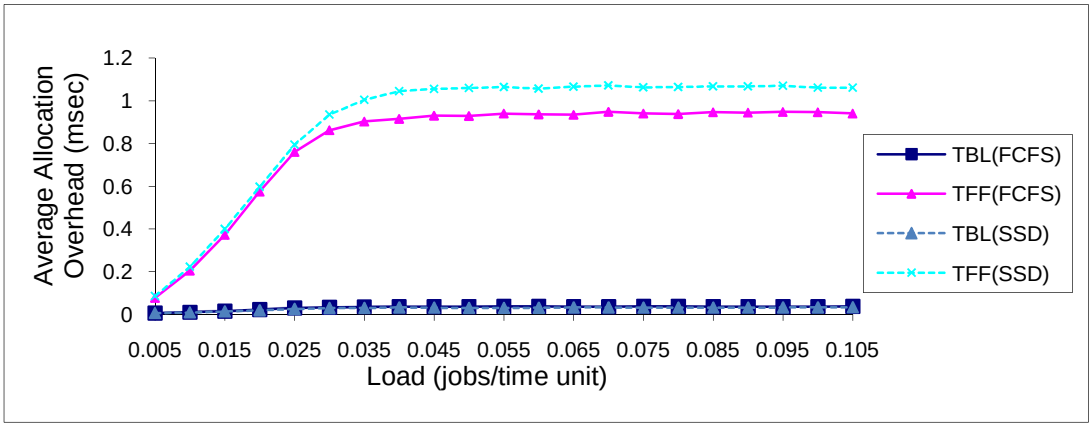


Figure 5.11: Average allocation overhead for the contiguous allocation strategies (TBL and TFF) under the scheduling strategies (FCFS and SSD) and uniform side lengths distribution in an 8 × 8 × 8 mesh.

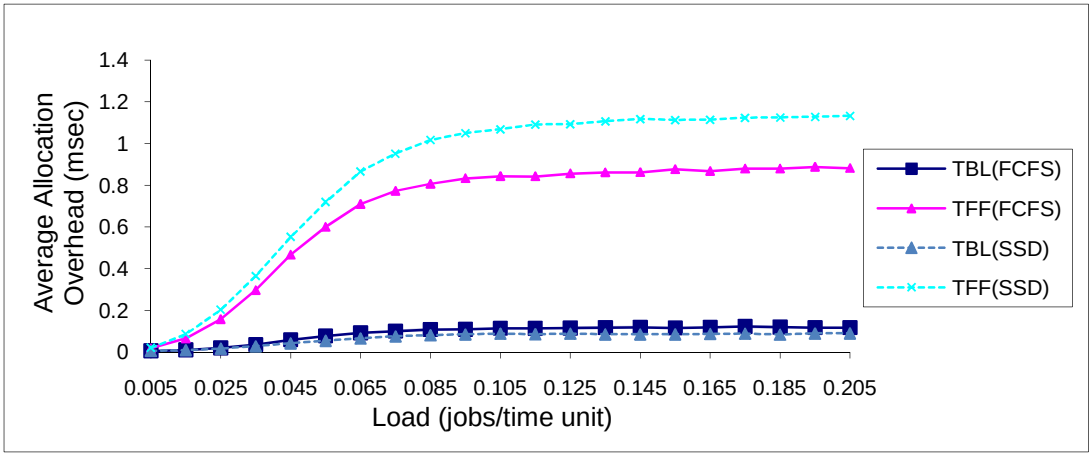


Figure 5.12: Average allocation overhead for the contiguous allocation strategies (TBL and TFF) under the scheduling strategies (FCFS and SSD) and exponential side lengths distribution in an 8 × 8 × 8 mesh.

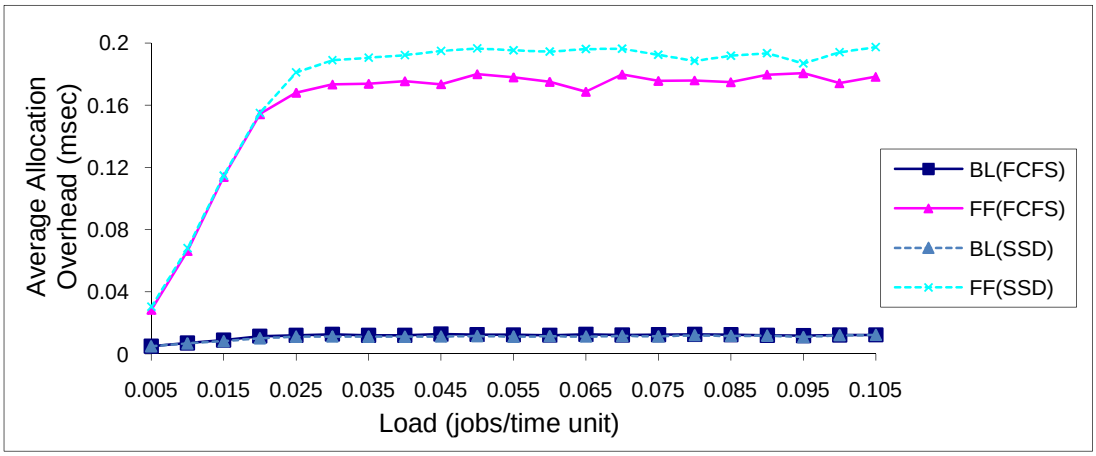


Figure 5.13: Average allocation overhead for the contiguous allocation strategies (BL and FF) under the scheduling strategies (FCFS and SSD) and uniform side lengths distribution in an 8 × 8 × 8 mesh.

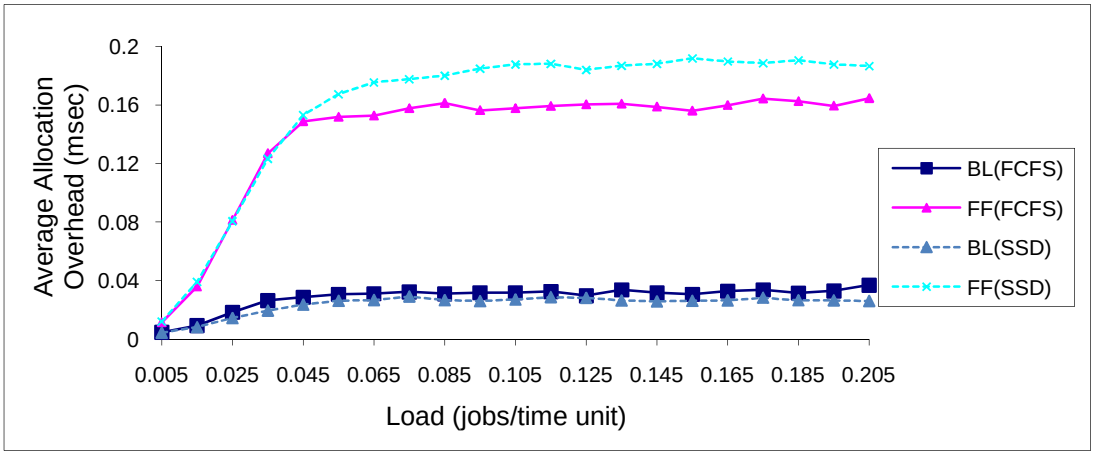


Figure 5.14: Average allocation overhead for the contiguous allocation strategies (BL and FF) under the scheduling strategies (FCFS and SSD) and exponential side lengths distribution in an 8 × 8 × 8 mesh.

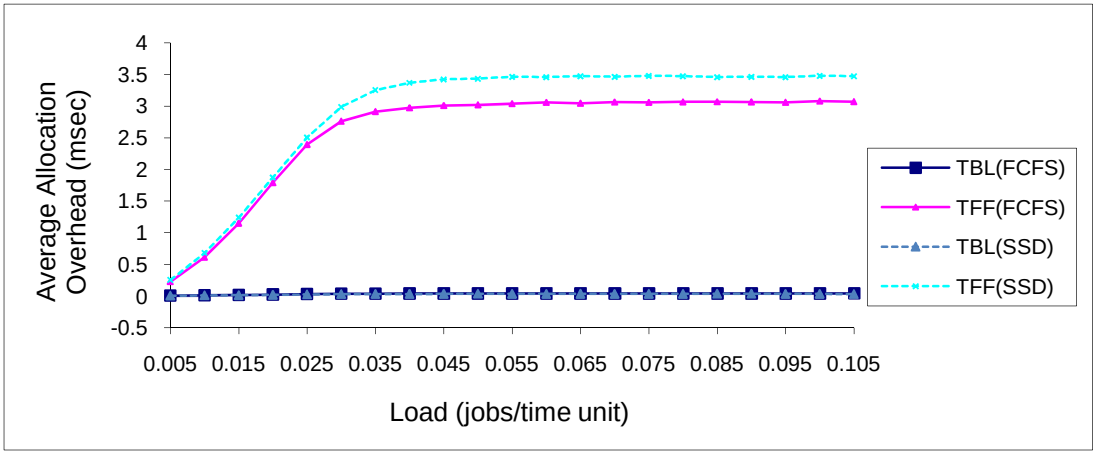


Figure 5.15: Average allocation overhead for the contiguous allocation strategies (TBL and TFF) under the scheduling strategies (FCFS and SSD) and uniform side lengths distribution in a 10 × 10 × 10 mesh.

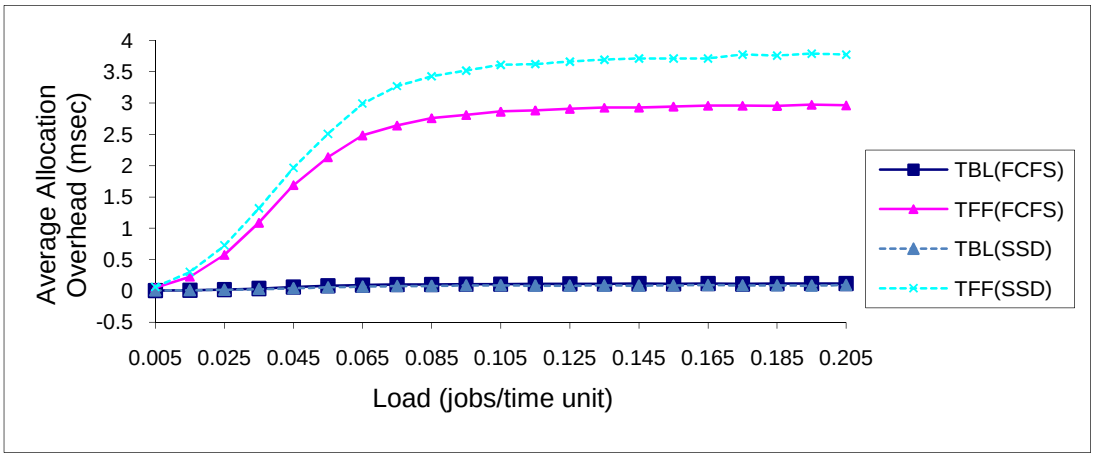


Figure 5.16: Average allocation overhead for the contiguous allocation strategies (TBL and TFF) under the scheduling strategies (FCFS and SSD) and exponential side lengths distribution in a 10 × 10 × 10 mesh.

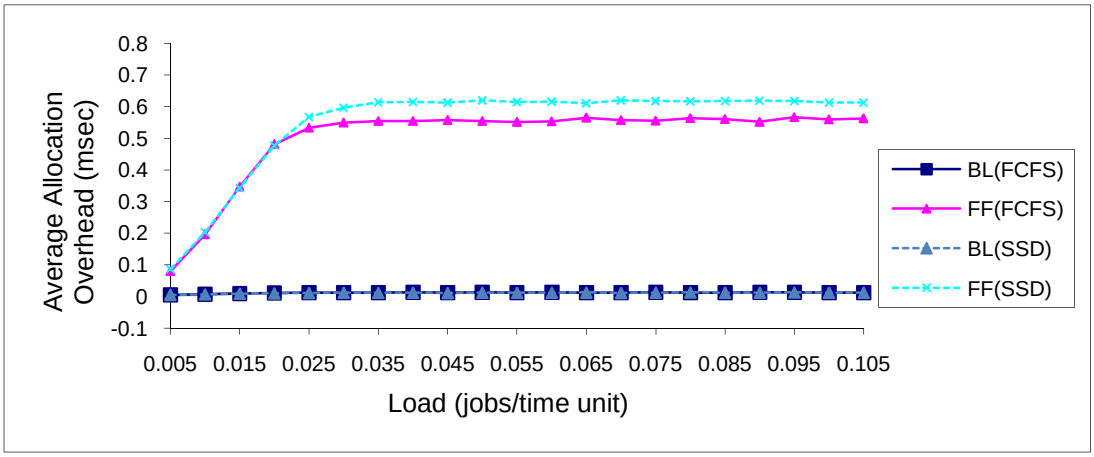


Figure 5.17: Average allocation overhead for the contiguous allocation strategies (BL and FF) under the scheduling strategies (FCFS and SSD) and uniform side lengths distribution in a 10 × 10 × 10 mesh.

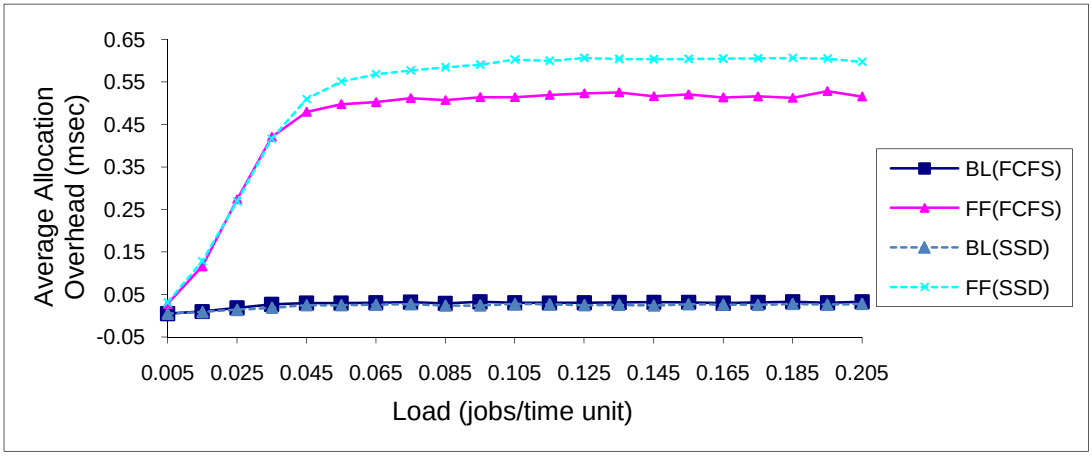


Figure 5.18: Average allocation overhead for the contiguous allocation strategies (BL and FF) under the scheduling strategies (FCFS and SSD) and exponential side lengths distribution in a 10 × 10 × 10 mesh.

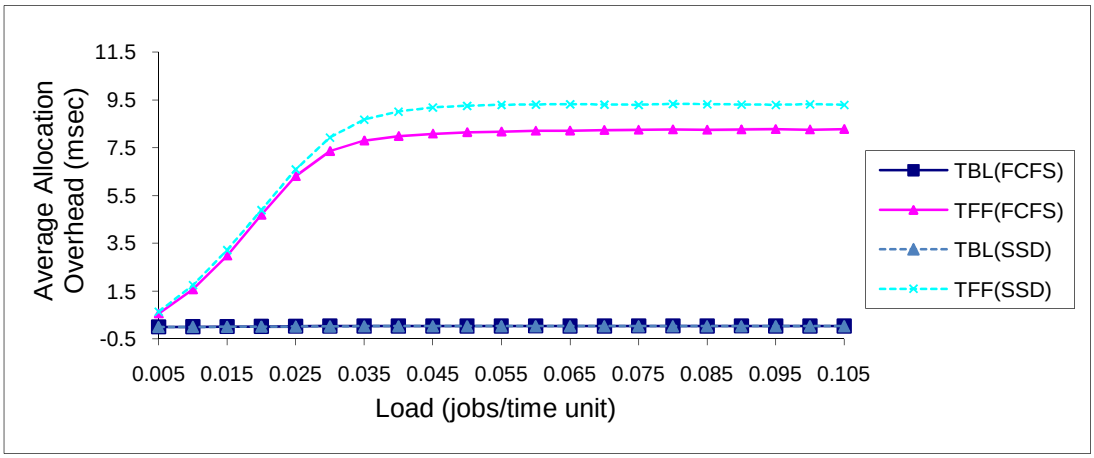


Figure 5.19: Average allocation overhead for the contiguous allocation strategies (TBL and TFF) under the scheduling strategies (FCFS and SSD) and uniform side lengths distribution in a 12 × 12 × 12 mesh.

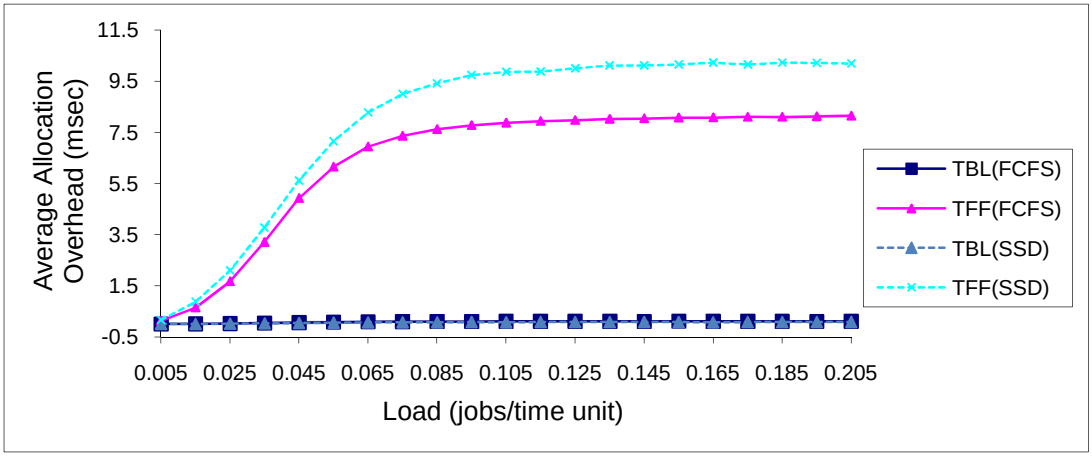


Figure 5.20: Average allocation overhead for the contiguous allocation strategies (TBL and TFF) under the scheduling strategies (FCFS and SSD) and exponential side lengths distribution in a 12 × 12 × 12 mesh.

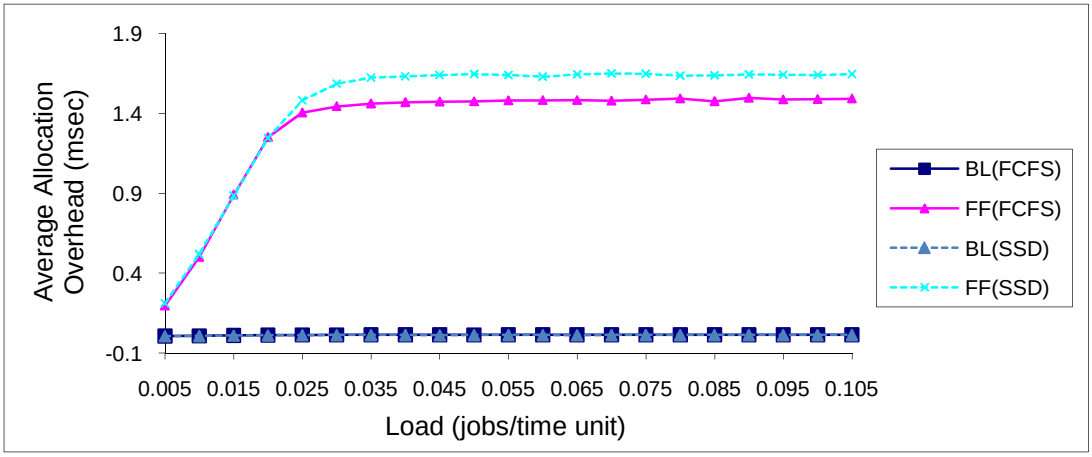


Figure 5.21: Average allocation overhead for the contiguous allocation strategies (BL and FF) under the scheduling strategies (FCFS and SSD) and uniform side lengths distribution in a 12 × 12 × 12 mesh.

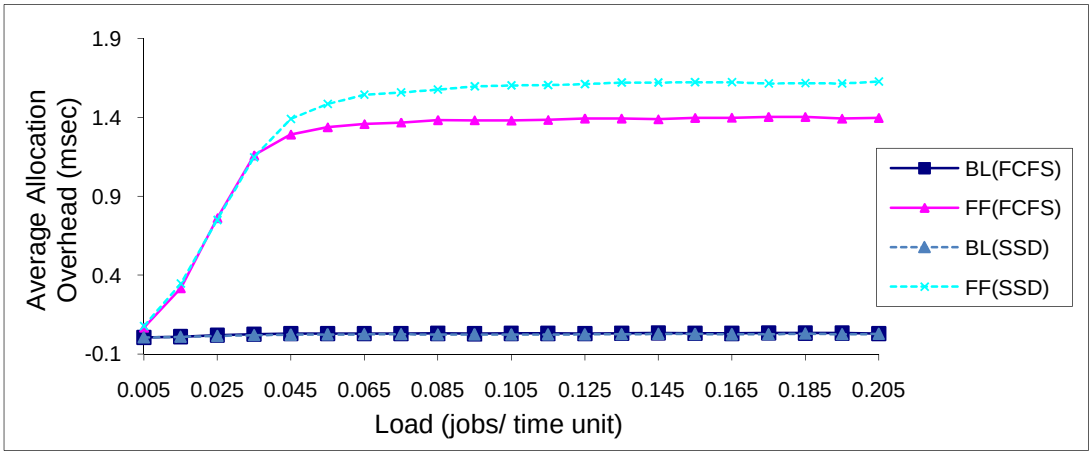


Figure 5.22: Average allocation overhead for the contiguous allocation strategies (BL and FF) under the scheduling strategies (FCFS and SSD) and exponential side lengths distribution in a 12 × 12 × 12 mesh.

5.4.3 Impact of System Size

In this section, we investigate the effect of the size of the mesh system on the performance of the allocation strategies considered in terms of average turnaround time of jobs under both FCFS and SSD when job execution times follow heavy-tailed distributions. For the sake of conciseness, we have only concentrated on job turnaround time in this section because it is usually a good estimate of the performance of processor allocation strategies and it has been used in the existing allocation strategies [9, 18, 20, 27, 33, 34, 51, 52, 65, 78,

85, 99].

Figure 5.23 assumes that the side lengths of the requested sub-meshes follow a uniform distribution, while an exponential distribution is assumed in Figure 5.24. The results reveal that the performance of the allocation strategies is little affected by changes in the system size in our considered scenarios. In Figure 5.24, the average turnaround time of the TBL(SSD) strategy for an 8 × 8 × 8 mesh system size is 98% of that of TBL(SSD) for a 10 × 10 × 10 mesh system size and 91% of that of TBL(SSD) for a 12 × 12 × 12 mesh system size. Moreover, the allocation strategies that use the rotation of the allocation request, as in TBL and TFF, perform much better than the allocation strategies that do not use the rotation of the allocation request, as in BL and FF, regardless of the mesh system size. Figure 5.23 shows that the average turnaround time of TBL(SSD) is 34% of that of BL(SSD) for a 12 × 12 × 12 mesh system size. The results also show that the SSD scheduling strategy improves the performance of the allocation strategies compared to FCFS scheduling. In Figure 5.23, the average turnaround time of TBL(SSD) is 8% of that of TBL(FCFS) for a 12 × 12 × 12 mesh system size.

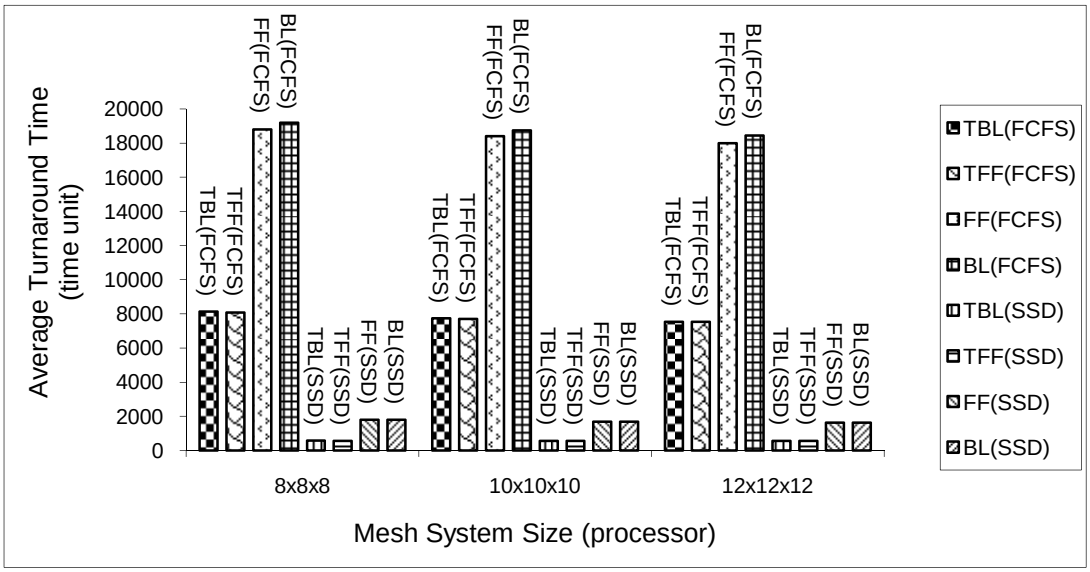


Figure 5.23: Average turnaround time vs. size of the mesh system for the contiguous allocation strategies (BL, FF, TBL, TFF) and the uniform side lengths distribution under FCFS and SSD scheduling strategies.

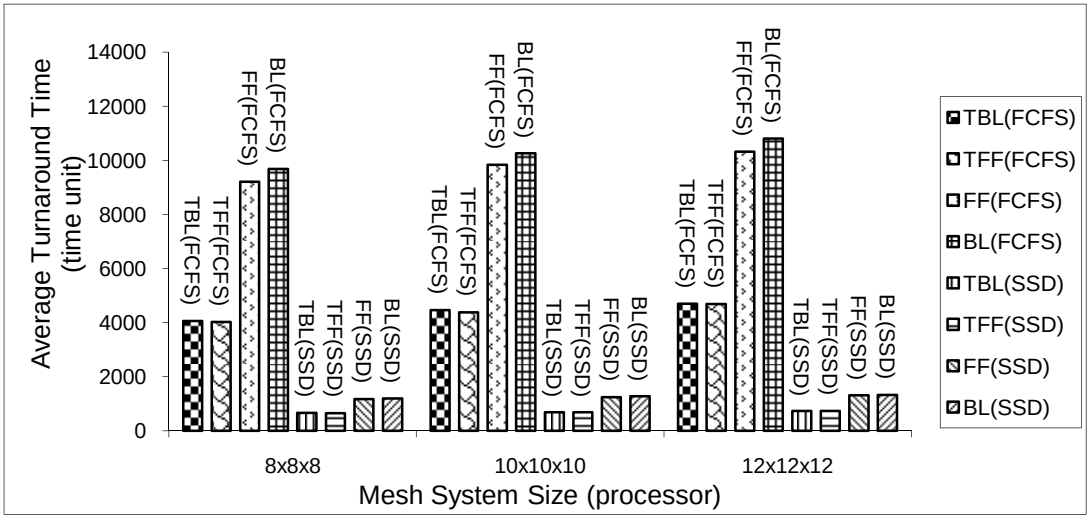


Figure 5.24: Average turnaround time vs. size of the mesh system for the contiguous allocation strategies (BL, FF, TBL, TFF) and the exponential side lengths distribution under FCFS and SSD scheduling strategies.

5.5 Conclusions

We have compared the performance of processor allocation strategies proposed for 3D mesh-connected multicomputers for a wide range of system loads and system sizes when the distribution of job execution times is heavy-tailed (e.g., Bounded Pareto distribution). The strategies examined in this chapter include First Fit (FF), Turning First Fit (TFF), a Busy List strategy (BL) and the Turning Busy List strategy (TBL). BL maintains a list of allocated sub-meshes to determine the nodes that cannot be used as base nodes for the requested sub-meshes, whereas TBL attempts to maintain a good performance in terms of mean system utilisation and average turnaround time with little allocation overhead.

The heavy-tailed distribution has been adopted in this study because many measurement studies have convincingly demonstrated that the execution times of certain computational jobs can be characterised by heavy-tailed distributions; that is, many jobs are short and fewer are long. Heavy-tailed distributions can capture this variability and have been shown to behave quite differently from the exponential distribution which may not reflect all

possible practical scenarios when compared to the heavy-tailed distribution.

The performance of the allocation strategies is measured in terms of usual performance parameters that have been used in the existing strategies including the average turnaround time and mean system utilisation, as well as the measured allocation overhead, that the allocation and de-allocation operations take per job. Moreover, the SSD scheduling strategy has been used to deal with heavy-tailed job execution times to avoid performance loss due to FCFS blocking that results from large jobs.

The simulation results have shown that the performance of the allocation strategies in terms of average turnaround time and mean system utilisation degrades considerably when the distribution of job execution times is heavy-tailed. This is because the long jobs' execution times that have been resulted from heavy-tailed distribution increase the average turnaround time of those jobs and which consequently degrade the system performance. Our analysis has shown that when job executions times follow a heavy-tailed distribution, SSD improves the performance of the allocation strategies compared to FCFS in terms of the performance metrics measured in this study.

The simulation results have also shown that the performance of TBL(SSD) is almost identical to that of TFF(SSD) and is superior over that of the other allocation strategies. Moreover, the performance of the TBL and BL strategies that depend on a list of allocated sub-meshes for both allocation and de-allocation is at least as good as that of the TFF and FF strategies that depend on the number of processors in the mesh system, assuming that the same scheduling strategy is used. The results have also shown that the average allocation overhead of the TBL and BL strategies is lower than that of the TFF and FF strategies that depend on the states of processors in the mesh system.

Chapter 6

Conclusions and Future Directions

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 [43, 70, 90]. Distributed-memory multicomputers are an important class of parallel computers for building large-scale parallel systems [83]. 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 [9, 18, 20, 21, 27, 31, 33, 35, 51, 52, 77, 78, 85, 99]. Meshes are suited to a variety of practical applications including matrix computation, image processing and problems whose task graphs can be embedded naturally into the mesh [89, 95]. It has been used as the underlying network in a number of commercial and experimental multicomputers, including the Intel Paragon [39], Cray XT3 [19, 60], MIT J-machine [61], Cray T3D [67], Cray T3E [25], iWARP [15], IBM BlueGene/L [10, 55, 97, 98], and Delta Touchstone [40].

Processor allocation in distributed-memory multicomputers, particularly those based on the

mesh network, has been the focus of a lot of research over the past years [9, 11, 16, 24, 26, 28, 31, 33, 34, 35, 45, 51, 52, 71, 72, 73, 74, 75, 76, 77, 78, 79, 81, 93, 97]. Several commercial and experimental parallel machines have used space sharing for processor allocation [10, 15, 19, 25, 39, 40, 55, 61, 67, 97, 98]. 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 [6, 11, 17, 37]. Processor allocation strategies based on space sharing can be divided into two broad categories: *contiguous* and *non-contiguous*. In contiguous allocation [9, 20, 21, 26, 27, 31, 33, 34, 35, 38, 48, 52, 65, 74, 75, 78, 94, 99], the allocated processors are physically contiguous and have the same topology as the underlying network, e.g. the mesh, in order to maintain low communication overhead among the allocated processors. The direct consequence of contiguous allocation is that good system utilisation is often difficult to achieve due to the *fragmentation* problem which results from contiguous allocation [18, 85]. The fragmentation problem could be of two types: *internal* and *external*. Internal fragmentation occurs when more processors are allocated to a job but not used, whereas external fragmentation occurs when there are a sufficient number of free processors are available to satisfy a job request but they are not allocated to it because they are not contiguous.

To solve the fragmentation problem, a number of researchers have adopted non-contiguous allocation [18, 24, 49, 71, 72, 84, 85] 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 the past, non-contiguous allocation has not attracted considerable research attention because the communication latency was sensitive to the distance in the network employed in the first generation of multicomputers [11]. However, the advances in routing technique such as *wormhole routing* [2, 4, 11, 29, 71, 72, 83] have made non-contiguous allocation plausible in networks characterised by long diameters such as the mesh. Wormhole routing

has been widely adopted in the second generation of multicomputers [25, 39, 40, 54, 67, 91]. An advantage of wormhole routing over earlier communication schemes, mainly store-and-forward, is that message latency has become less dependent on message distance [2, 43].

The main goal of a processor allocation strategy is to reduce job turnaround times and maximize system utilisation [72]. A given allocation strategy may have a partial or full sub-mesh recognition ability [85, 99]. Having a full sub-mesh recognition ability increases the time to allocate a sub-mesh to a new job, as has been shown in the studies of [26, 31, 34, 94, 97]. With increased system size, the time to search for free processors to satisfy an incoming job might be comparable to the job's execution time [46]. Hence it is important to develop allocation strategies that minimize the search time (also referred to as the allocation time), and as a result decrease the turnaround time of jobs. Furthermore, the method used for partitioning allocation requests in non-contiguous allocation has a considerable impact on the performance of non-contiguous allocation strategies [18, 71, 72, 85]. Hence, the partitioning process in non-contiguous allocation should aim to maintain a high degree of contiguity between the processors allocated to a given parallel job. This is so that the communication overhead is kept to a minimum without adversely affecting the overall system performance [71, 72].

6.1 Summary of the Results

The major focus of the present research has been the development of new efficient contiguous and non-contiguous allocation strategies for mesh-connected multicomputers that overcome the limitations of the existing strategies suggested for the 2D and the 3D mesh networks. Summarised below are the major contributions made in this research study.

- There have been relatively few contiguous allocation strategies that have been

suggested for the 3D mesh-connected multicomputers. These strategies achieve a complete sub-mesh recognition capability at the expense of a high allocation overhead [31, 34, 94], that accounts for the time required to allocate and de-allocate a set of processors to an incoming job. Furthermore, the allocation overhead in the previously proposed contiguous allocation strategies has been shown to grow with the system size [26, 31, 34, 94]. Motivated by these observations, the first part of this dissertation has proposed a new contiguous allocation strategy, referred to as Turning Busy List (TBL for short), for the 3D mesh-connected multicomputers. The TBL strategy exhibits a low allocation overhead and can identify a free sub-mesh of the requested size as long as it exists in the mesh system. It can do so because it relies on a new approach that maintains a list of allocated sub-meshes to determine all the regions consisting of the nodes that cannot be used as base nodes for the requested sub-mesh. These nodes are then subtracted from the right border plane of the already allocated sub-meshes in order to determine the nodes that can be used as base nodes for the required sub-mesh size.

- Extensive simulation experiments under a variety of system loads have been carried out in order to compare the performance of the proposed TBL allocation strategy against well-known contiguous allocation strategies [34], with and without change of request orientation. Our analysis has shown that in most circumstances TBL strategy exhibits a lower allocation overhead than the previous strategies [34]. For instance, simulation results have revealed that the allocation overhead in the TBL strategy can be as low as 4% of that in the existing Turning First Fit (TFF) strategy [34] in the presence of high loads. Moreover, when the number of processors increases the allocation overhead increases for the allocation strategies that depend on the number of processors in the mesh system (as in TFF and FF) while it does not

increase for the allocation strategies that depend on a list of allocated sub-meshes (as in TBL and BL). For example, the allocation overhead of the existing TFF strategy for a 12 × 12 × 12 mesh system size can increase by up to 773% of that for an 8 × 8 × 8 mesh system size. The allocation overhead in the proposed TBL strategy is kept low when the mesh system size increases, while its performance, in terms of the turnaround times and system utilisation, is still as good as that of the existing competing TFF strategy [34]. The new TBL strategy is efficient because it is implemented using a busy list approach. In practice it is often the case that when the system size scales up, the requirement of applications in terms of the number of requested processors often increases to exploit the available computational power, and in such scenarios our suggested strategy is expected to exhibit competitive performance levels.

- Our results have also revealed that the contiguous TBL and TFF strategies that employ request rotation have comparable performance, and are both superior to the other strategies that do not employ rotation (e.g., BL and FF). When compared against TBL and TFF, BL increases the average turnaround times by up to 65% in the presence high loads. The allocation strategies with rotation, notably, TBL and TFF, achieve system utilisation of 47% under the exponential distribution and 49% under uniform distribution. On the other hand, the BL and FF strategies that do not employ rotation cannot exceed 37% utilisation for both job size distributions.

- 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 [18, 24, 84, 85]. Moreover, the allocation of

processors to job requests is not based on free contiguous sub-meshes in the existing strategies [18, 85]. Instead, it is often based on artificial predefined geometric or arithmetic patterns [18, 85]. Hence these strategies may fail to allocate an available large sub-mesh, which in turn cause degradation in system performance in terms of turnaround times [18, 72, 85]. Motivated by these observations, the second part of this dissertation has suggested a new non-contiguous allocation algorithm, referred to as Greedy Available Busy List (GABL for short), for mesh-connected multicomputers. The GABL strategy combines the main desirable features of both the contiguous and non-contiguous allocation strategies. Moreover, GABL is general enough in that it could be applied to either the 2D or 3D mesh. However, 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; it is worth pointing out that we have opted to discuss our new allocation strategy in the context of the 2D mesh network because there has been hardly any non-contiguous allocation strategy which has been suggested for the 3D mesh network.

- The proposed GABL strategy relies on a new approach that maintains a higher degree of contiguity among the processors than that of the previous non-contiguous allocation strategies. This decreases the number of sub-meshes allocated to a job, hence decreases the distance traversed by messages, which in turn decreases communication overhead. Extensive simulation experiments under a variety of system operating conditions have been carried out to compare the performance of the proposed GABL strategy against that of the existing non-contiguous and 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

contiguous and non-contiguous allocation strategies of [85]. Moreover, when message contention increases inside the network due to using the all-to-all communication pattern, for example, GABL exhibits superior performance over the previous contiguous and non-contiguous allocation strategies. For instance, under high loads, the average turnaround times in GABL are 20%, 24%, and 38% of that of the contiguous First Fit (FF) [85, 99], non-contiguous Paging(0) [85], and non-contiguous Multiple Buddy Strategy (MBS) [85], respectively. Furthermore, the proposed strategy exhibits high system utilisation as it manages to eliminate both internal and external fragmentation. For instance, under high loads, GABL achieves a mean system utilisation of 71% to 75% under the exponential and uniform side lengths distributions, respectively, but system utilisation in the contiguous FF allocation strategy cannot exceed 50%.

- Experiments for large packet sizes have been also conducted. The results have shown that under most system loads GABL outperforms the previous contiguous and non-contiguous allocation strategies. For instance, when the packet length is 8-flits, the difference in performance in terms of average turnaround times in favour for the GABL strategy could be as large as 72% over Paging(0) and 49% over MBS under high loads. Similarly, when packet length is increased to 64 flits, the difference in performance in terms of average turnaround times in favour for the GABL strategy could be as large as 85% over Paging(0) and 55% over MBS.

- Experiments for large system sizes in terms of average turnaround times have also been carried out. GABL has been found to perform better than the existing contiguous and non-contiguous allocation strategies for all system sizes. For instance, for a 16 × 16 mesh system size, the average turnaround times of GABL can be 20%, 24%, and 37% lower of that of FF, Paging(0), and MBS, respectively. For a

64 × 64 mesh system size, the average turnaround times of GABL can be 23%, 34%, and 45% lower of that of FF, Paging(0), and MBS, respectively. Moreover, the results have shown a significant drop in performance as the system scales up. For instance, the average turnaround time of GABL for a 64 × 64 mesh system size could increase by as much as 194% of that for a 16 × 16 mesh system size. This is because when the system size increases, the allocated processors might be far from each other. This increases the distance traversed by messages, and as a result increases the communication overhead, leading to an increases in the turnaround time of jobs.

- The performance evaluation of most allocation strategies, including those described here [6, 11, 18, 27, 31, 33, 34, 35, 38, 48, 51, 52, 74, 78, 85, 94, 99] have assumed an exponential distribution for job execution times. However, many measurement studies [22, 47, 56, 57, 58, 59, 88, 96] have convincingly demonstrated that the execution times of certain computational jobs could be better characterised by heavy-tailed distributions; that is, many jobs are short and fewer are long. The few jobs that have long execution times can account for more than half of the total jobs' execution time [59]. Heavy-tailed probability distributions (e.g., Bounded Pareto) can capture this variability in job execution times and have been shown to behave quite differently from the traditional exponential probability distribution, which has been widely used to evaluate the performance of allocation strategies [22, 57, 58, 75]. Most importantly, when sampling random variables that follow a heavy-tailed distribution, the probability of large generated values is non-negligible [22, 47, 56, 57, 58, 59, 88, 96].

- In the final part of this dissertation, the performance of the existing contiguous allocation strategies for 3D mesh-connected multicomputers, including the ones

developed in this research, has been revisited in the context of heavy-tailed job execution times. To the best of our knowledge, this study is the first to consider heavy-tailed distributions in the context of processor allocation on mesh-connected multicomputers. As in [6, 9, 18, 21, 27, 31, 33, 34, 35, 38, 71, 72, 73, 74, 75, 77, 78, 79, 85, 94, 99], in this part, the performance of allocation strategies is measured in terms of the average turnaround time and mean system utilisation, as well as the measured allocation overhead, that is, the time that the allocation and de-allocation operations take per job. It is worth noting that we have limited our investigation to contiguous allocation strategies in this research due to time and resource limitations.

- Our study has revealed that in general the performance of the allocation strategies degrades considerably when the distribution of job execution times is heavy-tailed (e.g., Bounded Pareto). This is because the long jobs' execution times due to the heavy-tailed distribution increase the average turnaround time of those jobs, and consequently degrade system performance. For instance, the average turnaround time of TBL(FCFS) (i.e., TBL with the FCFS scheduling strategy) under the exponential job execution time distribution is 49% of that of TBL(FCFS) under the heavy-tailed job execution time distribution and high loads. Our analysis has also shown that when job executions times follow a heavy-tailed distribution the Shortest-Service-Demand (SSD) scheduling strategy improves the performance of the allocation strategies compared to the FCFS scheduling strategy. For instance, the average turnaround time of TBL(SSD) (i.e., TBL with the SSD scheduling strategy) is 7% of that of TBL(FCFS) in the presence of high loads. Also, TBL(SSD) achieves system utilisation of 52%, but TBL(FCFS) cannot exceed 39% system utilisation.

- Having said the above, the allocation overhead of the TBL and BL allocation

strategies is still much lower than that of the TFF and FF allocation strategies when the job execution times follow a heavy tailed distribution. For instance, the allocation overhead in the TBL(FCFS) strategy for an 8 × 8 × 8 mesh system size is 4% of that in the TFF(FCFS) strategy. Moreover, when the number of processors increases the allocation overhead increases in the allocation strategies that depend on the number of processors in the mesh system, as in TFF and FF, while it does not increase in the allocation strategies that depend on a list of allocated sub-meshes, as in TBL and BL. For instance, the allocation overhead in the TFF(FCFS) strategy for an 8 × 8 × 8 mesh system size is 11% of that in the TFF(FCFS) strategy for a 12 × 12 × 12 mesh system size.

- Experiments to measure the average turnaround times have also been conducted for large system sizes. However, the main conclusions on the performance of the allocation strategies remain unchanged. For example, the average turnaround time of the TBL(SSD) strategy for an 8 × 8 × 8 mesh system size is 98% of that for a 10 × 10 × 10 mesh system size and 91% for a 12 × 12 × 12 mesh system size.

6.2 Directions for the Future Work

There are several interesting issues and open problems that require further investigation. These are briefly outlined below.

- In this research, the performance of the allocation strategies proposed in Chapters 3 and 4 has been evaluated assuming the First-Come-First-Served (FCFS) scheduling strategy. A natural extension of this work would be to evaluate the performance of our allocation strategies with other possible scheduling approaches, such as smallest job first (SJF) [66], Last Come First Served (LCFS) [66], Out of Order (OO) [34],

and backfilling [93]. Backfilling allows a later job in the waiting queue to be chosen to schedule as long as its execution does not delay the earliest possible execution of the earliest arriving job in the queue [93]. This requirement imposes the need for an estimation of job execution times.

- The results in Chapter 4 and in [85] have shown that non-contiguous allocation strategies dramatically outperform contiguous allocation strategies in the 2D mesh network. Greedy Available Busy List strategy (GABL) proposed in Chapter 4 can be applied to either the 2D or 3D mesh network. It can be adapted to 3D mesh by exploiting an efficient approach, the Turning Busy List (TBL) approach described in Chapter 3 for 3D mesh, for the detection of such available sub-meshes. It would be interesting to investigate the performance of the non-contiguous allocation against that of the contiguous allocation in 3D mesh network by comparing the performance of the proposed GABL non-contiguous allocation algorithm described in Chapter 4 against that of the TBL contiguous allocation algorithm described in Chapter3.

- The study conducted in Chapter 5 has examined the performance of the contiguous allocation strategies in the context of heavy-tailed distributions. It would be interesting to conduct a similar performance study on the non-contiguous allocation strategies.

- The results in Chapter 5 have revealed that the performance of the allocation strategies degrades considerably when the distribution of job execution times is heavy-tailed. A challenging continuation of this work would be to develop new allocation strategies that can efficiently support heavy-tailed job execution times.

- There have been a number of interconnection networks such as torus and hypercube

networks which have been suggested for multicomputers over the past years [93]. It would be interesting to adapt the proposed allocation strategies to other well-known network topologies and assess their performance on these networks.

- Throughout this research, it has been assumed that messages are routed according to deterministic routing. Even though this form of routing is simple to implement it cannot react to a change in network conditions. In adaptive routing, intermediate nodes take current network conditions, such as the presence of congestions or failures, into account to determine a route that a message should select to cross the network. It would be interesting to extend the proposed allocation strategies to this type of routing.

- Irregular networks have received considerable attention from the research community due to the emergence of clusters of workstations as a cost-effective method for achieving parallel processing. A new direction of research along the broad lines of this dissertation would be to investigate the development of efficient contiguous and non-contiguous allocation algorithms for this class of network topologies.

- The performance of the proposed allocation strategies, as well as the existing strategies, has been traditionally carried out by means of simulation based on stochastic workload models to generate a stream of incoming jobs. To validate the findings of the existing research, including that outlined in this thesis, on the performance properties of the existing allocation algorithms, there is a need to examine the performance of these strategies using real workload traces. Hence, it would be very interesting to analyse the performance of our strategies based on real workload traces collected from practical parallel systems and contrast the results

obtained against those obtained by means of simulation.

- Research efforts on processor allocation have relied on the simulation method to analyse the performance behaviour of most suggested strategies. As in other research endeavours, simulation cannot (due to time and complexity considerations) predict results and provide insight for all possible scenarios. A natural extension to the research efforts described in this dissertation would be to develop analytical models that can capture the performance behaviour of the proposed allocation strategies for cases that cannot be investigated by simulations.

- There has been little research activity in the performance measurement of actual parallel systems. Provided sufficient resources were available to materialise an actual multicomputer, it would be useful to conduct measurements to verify the conclusions that have been reported in the literature and which have largely been reached by means of simulations. Apart from instilling confidence in the existing work, such an investigation might reveal issues ignored in the assumptions of the simulation model or otherwise not captured by present simulation tools.

Appendix A

The Components of the MBS Allocation Algorithm

A.1 Introduction

In the MBS allocation strategy, a job request for p processors is represented as a base 4 number of the following form: $p = d_i \times 2^i \times 2^i + d_{i-1} \times 2^{i-1} \times 2^{i-1} + ... + d_0 \times 2^0 \times 2^0$. MBS is composed of the following five parts [85]: system initialisation, request factoring algorithm, buddy generating algorithm, allocation algorithm, and de-allocation algorithm.

A.2 System Initialisation

In this part, the mesh system is divided into *initial blocks* (i.e., sub-meshes), which are non-overlapped square sub-meshes with side lengths equal to powers of 2. The concept of *free block records* (FBR) extends the notion of the free block lists in the 2DBS strategy [48]. FBR[i] records the number (FBR[i].block_num) of available blocks of size $2^i \times 2^i$ and an ordered list

(FBR[i].block_list) of the locations of such blocks. Another global variable, AVAIL, keeps track of the current number of available processors in the mesh system, and is initialised to the number of processors in the system (N).

A.3 The Request Factoring Algorithm

The number of processors requested by an incoming job request has a base 4 representation of the form $\sum_{i=0}^{\lfloor \log_4 N \rfloor} d_i \times (2^i \times 2^i)$ where $0 \le d_i \le 3$. Thus any job request can be accommodated by d_i blocks of size $2^i \times 2^i$. At most $\lceil \log_4 N \rceil$ distinct blocks are needed with a maximum of 3 blocks of a given size. The *Maximum distinct blocks* (*MaxDB*) of a given mesh system is defined as $\lceil \log_4 N \rceil$. The factoring algorithm needs to take as an input the job size and produces as output a request array (Request_Array[0..*MaxDB*]). Request_Array[i] is the number of size $2^i \times 2^i$ blocks that the job needs.

A.4 The Buddy Generating Algorithm

The buddy breaks a large block into 4 smaller adjacent blocks to satisfy the $2^i \times 2^i$ requests. For example, the 4 buddies of a large block $2^j \times 2^j$ are $2^{j-1} \times 2^{j-1}$ blocks. The algorithm operates in two phases. In the first phase, an available block is searched by examining the FBRs in increasing order of block size from $2^{i+1} \times 2^{i+1}$ to $2^{\max} \times 2^{\max}$. During the second phase, the block is repeatedly broken down into smaller buddies until the desired size blocks are found. If no block is found during the search phase, the algorithm breaks the request for a $2^i \times 2^i$ block into 4 smaller requests for $2^{i-1} \times 2^{i-1}$ blocks.

A.5 The Allocation Algorithm

First, the request is factored and stored in Request_Array. This strategy attempts to satisfy each request for a block of size $2^i \times 2^i$ from FBR[i]. Otherwise, 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 $2^j \times 2^j$ block are $2^{j-1} \times 2^{j-1}$ blocks. If that fails, MBS breaks the request for a $2^i \times 2^i$ block into 4 smaller requests for $2^{i-1} \times 2^{i-1}$ blocks, which are stored in Request_Array[i-1], and repeats the allocation process. In MBS, 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.

A.6 The De-allocation Algorithm

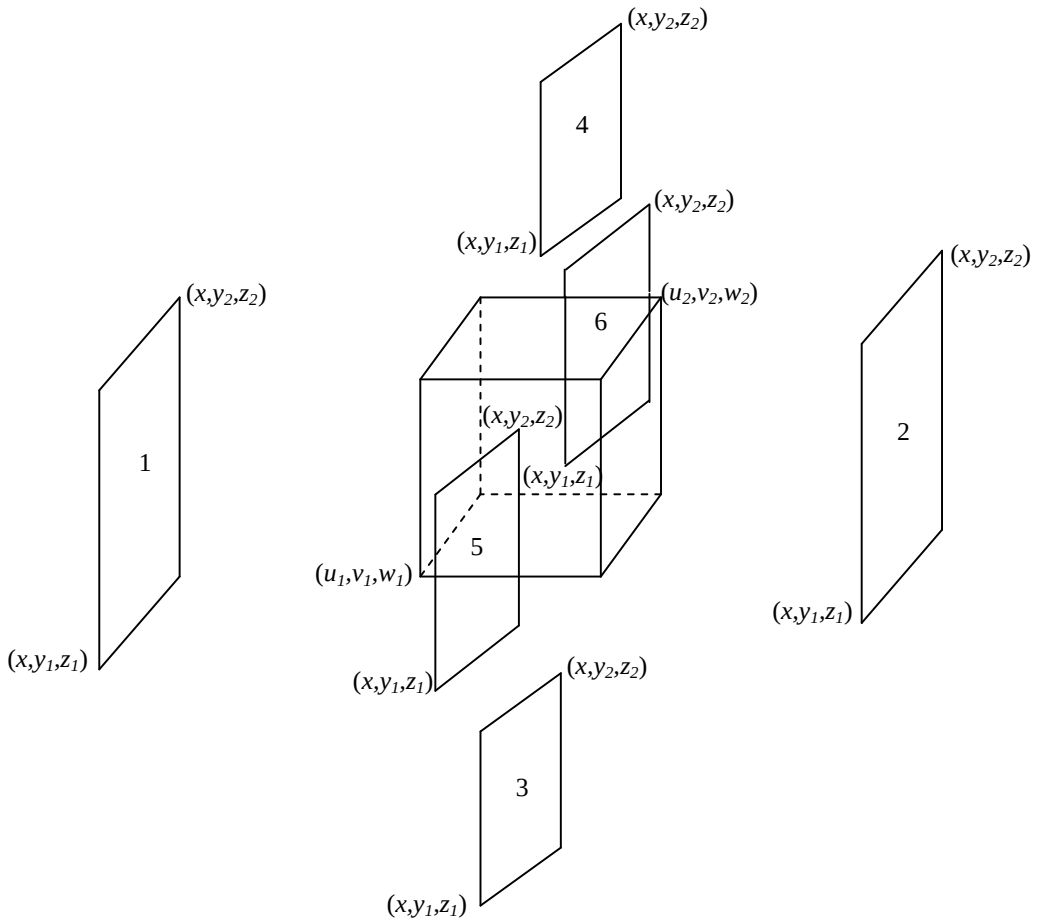
The MBS strategy needs to return all the blocks owned by the job to the system, and merge the buddies up to restore the larger blocks.

Appendix B

The Possible Cases for Subtracting Prohibited Regions from RBP's in the TBL Allocation Algorithm

The figures for all possible cases of subtracting Prohibited Regions (PR) from a Right Border Plane (RBP) introduced in Chapter 3 are presented for each case. In all of the figures presented in this Appendix, the coordinates of the RBP are represented by the address $(x, y_1, z_1, x, y_2, z_2)$ while the coordinates of PR are represented by the address $(u_1, v_1, w_1, u_2, v_2, w_2)$.

For example, Figure B.1 shows 6 possible situations for subtracting PR from RBP (please see Case 3.3.1 in Figure 3.3, Chapter 3); in all of these situations the subtraction process results in the same RBP. As a consequence, all processors on the RBP can be used as base processors for an allocation sub-mesh. The 6 possible situations for the RBP in Figure B.1 are: $x < u_1$, $x > u2$, $z_2 < w_1$, $z_1 > w_2$, $y_2 < v_1$, $y_1 > v_2$.

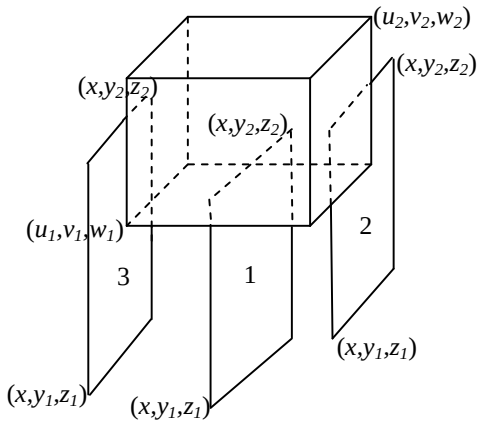


1. $x < u_1$
2. $x > u2$
3. $z_2 < w_1$
4. $z_1 > w_2$
5. $y_2 < v_1$
6. $y_1 > v_2$

((x< u1) or (x> u2) or (z2< w1) or (z1> w2) or (y2< v1) or (y1> v2))

In this figure the result is RBP itself.

Figure B.1: Subtracting PR from RBP (Case 3.3.1 from Figure 3.3 in Chapter 3)



1. $(u_1 \le x \le u_2)$ *and* $(v_1 \le y_1 \le v_2)$ *and* $(v_1 \le y_2 \le v_2)$ *and* $(w_1 \le z_2 \le w_2)$ *and* $(z_1 < w_1)$

RBP $(x, y_1, z_1, x, y_2, w_1\text{-}1)$

2. $(u_1 \le x \le u_2)$ *and* $(v_1 \le y_1 \le v_2)$ *and* $(y_2 > v_2)$ *and* $(w_1 \le z_2 \le w_2)$ *and* $(z_1 < w_1)$

RBP1 $(x, y_1, z_1, x, y_2, w_1\text{-}1)$

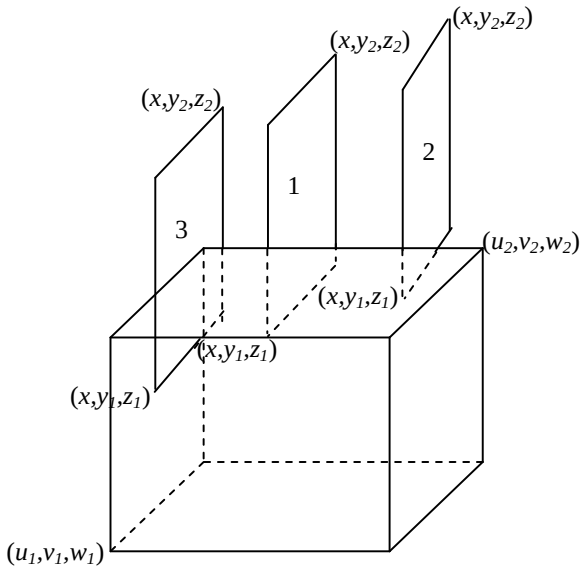
RBP2 $(x, v_2\text{+}1, w_1, x, y_2, z_2)$

3. $(u_1 \le x \le u_2)$ *and* $(v_1 \le y_2 \le v_2)$ *and* $(y_1 < v_1)$ *and* $(w_1 \le z_2 \le w_2)$ *and* $(z_1 < w_1)$

RBP1 $(x, y_1, z_1, x, y_2, w_1\text{-}1)$

RBP2 $(x, y_1, w_1, x, v_1\text{-}1, z_2)$

Figure B.2: Subtracting PR from RBP (Cases 3.3.2, 3.3.3, 3.3.4 in Figure 3.3 in Chapter 3).



1. ($u_1 \le x \le u_2$) and ($v_1 \le y_1 \le v_2$) and ($v_1 \le y_2 \le v_2$) and ($w_1 \le z_1 \le w_2$) and ($z_2 > w_2$)

RBP ($x, y_1, w_2+1, x, y_2, z_2$)

2. ($u_1 \le x \le u_2$) and ($v_1 \le y_1 \le v_2$) and ($y_2 > v_2$) and ($w_1 \le z_1 \le w_2$) and ($z_2 > w_2$)

RBP1 ($x, v_2+1, z_1, x, y_2, w_2$)

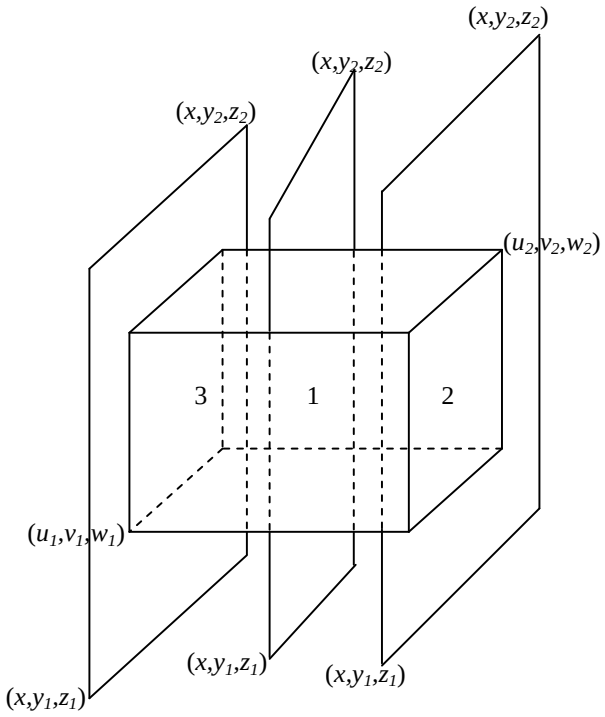
RBP2 ($x, y_1, w_2+1, x, y_2, z_2$)

3. ($u_1 \le x \le u_2$) and ($v_1 \le y_2 \le v_2$) and ($y_1 < v_1$) and ($w_1 \le z_1 \le w_2$) and ($z_2 > w_2$)

RBP1 ($x, y_1, z_1, x, v_1-1, w_2$)

RBP2 ($x, y_1, w_2+1, x, y_2, z_2$)

Figure B.3: Subtracting PR from RBP (Cases 3.3.5, 3.3.6, 3.3.7 in Figure 3.3 in Chapter 3).



1. $(u_1 \leq x \leq u_2)$ *and* $(v_1 \leq y_1 \leq v_2)$ *and* $(v_1 \leq y_2 \leq v_2)$ *and* $(z_1 < w_1)$ *and* $(z_2 > w_2)$

RBP1 $(x, y_1, z_1, x, y_2, w_1\text{-}1)$

RBP2 $(x, y_1, w_2\text{+}1, x, y_2, z_2)$

2. $(u_1 \leq x \leq u_2)$ *and* $(v_1 \leq y_1 \leq v_2)$ *and* $(y_2 > v_2)$ *and* $(z_1 < w_1)$ *and* $(z_2 > w_2)$

RBP1 $(x, y_1, z_1, x, v_2, w_1\text{-}1)$

RBP2 $(x, v_2\text{+}1, z_1, x, y_2, z_2)$

RBP3 $(x, y_1, w_2\text{+}1, x, v_2, z_2)$

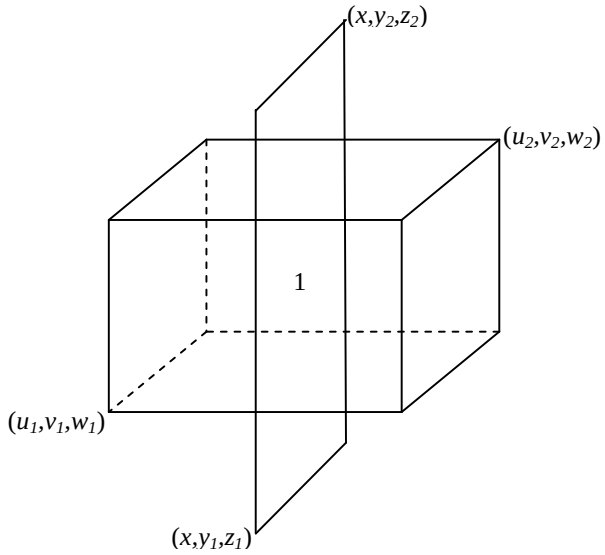
3. $(u_1 \leq x \leq u_2)$ *and* $(v_1 \leq y_2 \leq v_2)$ *and* $(y_1 < v_1)$ *and* $(z_1 < w_1)$ *and* $(z_2 > w_2)$

RBP1 $(x, y_1, z_1, x, v_1\text{-}1, z_2)$

RBP2 $(x, v_1, z_1, x, y_2, w_1\text{-}1)$

RBP3 $(x, v_1, w_2\text{+}1, x, y_2, z_2)$

Figure B.4: Subtracting PR from RBP (Cases 3.3.8, 3.3.9, 3.3.10 in Figure 3.3 in Chapter 3).

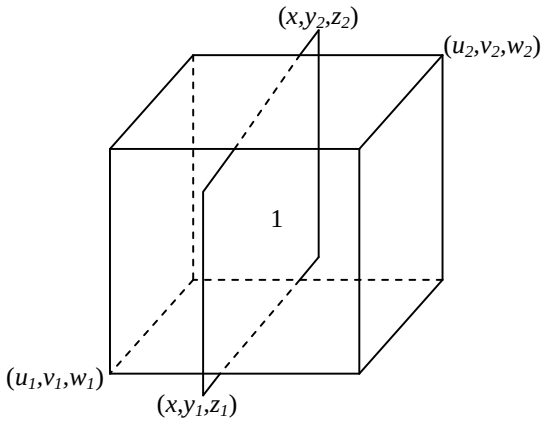


1. $(u_1 \leq x \leq u_2)$ *and* $(y_2 > v_2)$ *and* $(y_1 < v_1)$ *and* $(z_1 < w_1)$ *and* $(z_2 > w_2)$

RBP1 $(x, y_1, z_1, x, v_1\text{-}1, z_2)$ *;* *RBP2* $(x, v_2\text{+}1, z_1, x, y_2, z_2)$

RBP3 $(x, v_1, z_1, x, v_2, w_1\text{-}1)$ *;* *RBP4* $(x, v_1, w_2\text{+}1, x, v_2, z_2)$

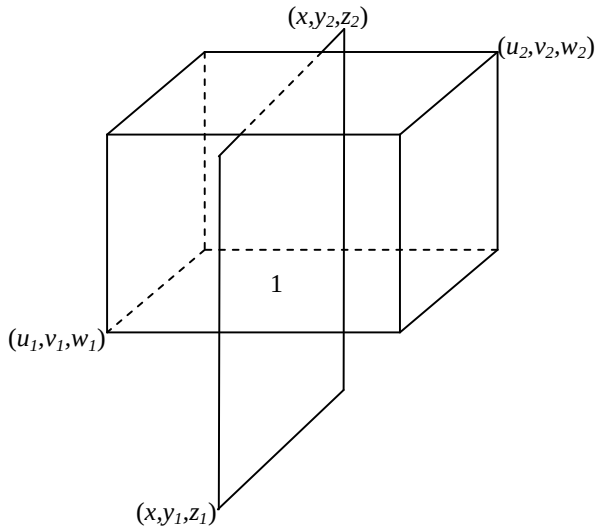
Figure B.5: Subtracting PR from RBP (Case 3.3.11 in Figure 3.3 in Chapter 3).



1. $(u_1 \leq x \leq u_2)$ *and* $(y_2 > v_2)$ *and* $(y_1 < v_1)$ *and* $(z_1 \geq w_1)$ *and* $(z_2 \leq w_2)$

RBP1 $(x, y_1, z_1, x, v_1\text{-}1, z_2)$ *;* *RBP2* $(x, v_2\text{+}1, z_1, x, y_2, z_2)$

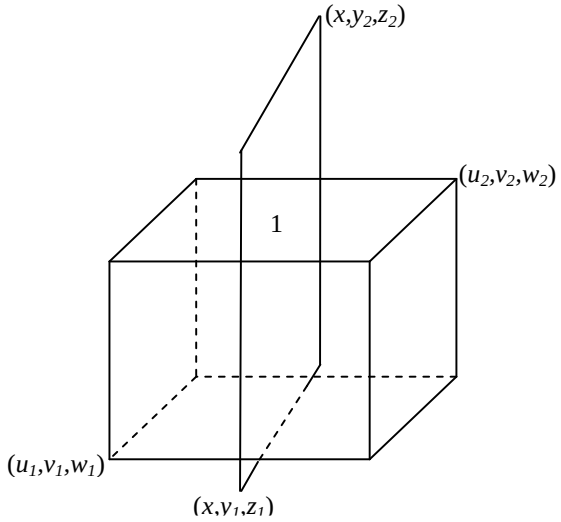
Figure B.6: Subtracting PR from RBP (Case 3.3.12 in Figure 3.3 in Chapter 3).



1. $(u_1 \leq x \leq u_2)$ and $(y_2 > v_2)$ and $(y_1 < v_1)$ and $(z_1 < w_1)$ and $(w_1 \leq z_2 \leq w_2)$

RBP1 $(x, y_1, z_1, x, v_1\text{-}1, z_2)$; RBP2 $(x, v_2\text{+}1, z_1, x, y_2, z_2)$; RBP3 $(x, v_1, z_1, x, v_2, w_1\text{-}1)$

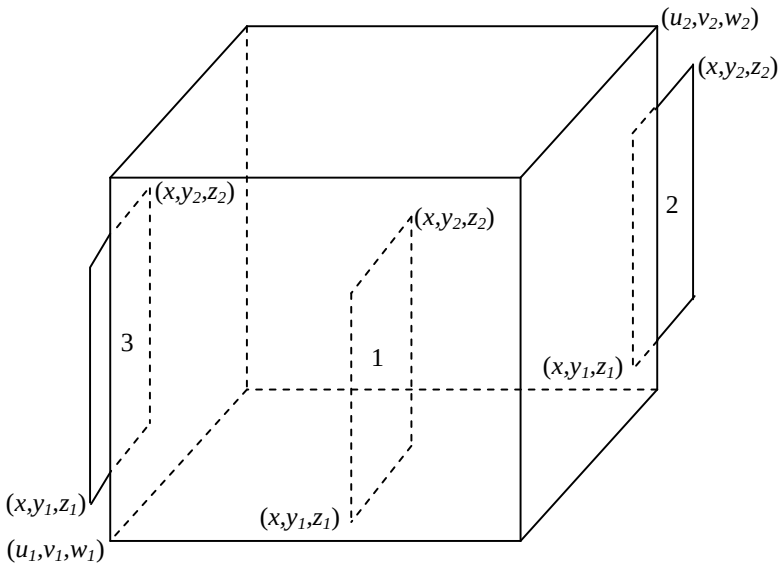
Figure B.7: Subtracting PR from RBP (Case 3.3.13 in Figure 3.3 in Chapter 3).



1. $(u_1 \leq x \leq u_2)$ and $(y_2 > v_2)$ and $(y_1 < v_1)$ and $(z_2 > w_2)$ and $(w_1 \leq z_1 \leq w_2)$

RBP1 $(x, y_1, z_1, x, v_1\text{-}1, z_2)$; RBP2 $(x, v_2\text{+}1, z_1, x, y_2, z_2)$; RBP3 $(x, v_1, w_2\text{+}1, x, v_2, z_2)$

Figure B.8: Subtracting PR from RBP (Case 3.3.14 in Figure 3.3 in Chapter 3).



1. $(u_1 \le x \le u_2)$ and $(v_1 \le y_1 \le v_2)$ and $(v_1 \le y_2 \le v_2)$ and $(w_1 \le z_1 \le w_2)$ and $(w_1 \le z_2 \le w_2)$

No RBP in this case.

2. $(u_1 \le x \le u_2)$ and $(v_1 \le y_1 \le v_2)$ and $(y_2 > v_2)$ and $(w_1 \le z_1 \le w_2)$ and $(w_1 \le z_2 \le w_2)$

RBP $(x, v_2+1, z_1, x, y_2, z_2)$

3. $(u_1 \le x \le u_2)$ and $(v_1 \le y_2 \le v_2)$ and $(y_1 < v_1)$ and $(w_1 \le z_1 \le w_2)$ and $(w_1 \le z_2 \le w_2)$

RBP $(x, y_1, z_1, x, v_1-1, z_2)$

Figure B.9: Subtracting PR from RBP (Cases 3.3.15, 3.3.16, 3.3.17 in Figure 3.3 in Chapter 3).

Appendix C

Publications during the Course of this Research

Journal Papers

- S. Bani-Mohammad, M. Ould-Khaoua and I. Ababneh, *An Efficient Non-Contiguous Processor Allocation Strategy for 2D Mesh Connected Multicomputers*, Journal of Information Sciences - Elsevier (INS), Elsevier, Vol. 177, No. 14, pp. 2867-2883, 15 July 2007.

- S. Bani-Mohammad, M. Ould-Khaoua and I. Ababneh, *A New Processor Allocation Strategy with a High Degree of Contiguity in Mesh-Connected Multicomputers*, Journal of Simulation Modelling, Practice & Theory (SIMPRA), Elsevier Science, Vol. 15, No. 4, pp. 465-480, April 2007.

- S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh and Lewis M. Mackenzie, *Processor Allocation and Job Scheduling on 3D Mesh Interconnection Networks*, International

Journal of Computers and Applications, (ACTA), Vol. 29, No. 3, pp. 309-317, Canada, ACTA Press, 2007.

- S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh and Lewis M. Mackenzie, *A Fast and Efficient Strategy for Sub-mesh Allocation with Minimal Allocation Overhead in 3D Mesh Connected Multicomputers*, Ubiquitous Computing and Communication Journal (UBICC), Vol.1, No. 1, pp. 26-36, ISSN 1992-8424, 2006.

- S. Bani-Mohammad, M. Ould-Khaoua and I. Ababneh, *Greedy-Available Non-contiguous Processor Allocation Strategy and Job Scheduling for 2D Mesh Connected Multicomputers*, Accepted to appear in International Journal of Computers and their Applications (IJCA), International Society for Computers and Their Applications (ISCA), 2008.

- S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, and Lewis M. Mackenzie, *Comparative Evaluation of Contiguous Allocation Strategies on 3D Mesh Multicomputers*, Revised version under review for Journal of Systems and Software, Elsevier Publishing, 2008.

Conference Papers

- S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, Lewis M. Mackenzie and J. D. Ferguson, *The Effect of Real Workloads and Stochastic Workloads on the Performance of Allocation and Scheduling Algorithms in 2D Mesh Multicomputers*, Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS 2008). Hyatt Regency Hotel, Miami, Florida, USA. IEEE Computer Society Press, April

14-18, 2008.

- S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh and Lewis M. Mackenzie, *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), Hsinchu, Taiwan, IEEE Computer Society Press, Volume 2 , pp. 1-7, December 5-7, 2007.

- S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh and Lewis M. Mackenzie, *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 2007), Niagara Falls, ON, CANADA, Springer-Verlag Berlin Heidelberg, LNCS 4742, pp. 645-656, August 29-21, 2007.

- S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh and Lewis M. Mackenzie, *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), Amman, Jordan, IEEE Computer Society Press, pp. 934-941, May 13-16, 2007.

- S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh and Lewis M. Mackenzie, *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, Volume 2 , pp. 41-48, 12-15 July, 2006.

- S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh and Lewis M. Mackenzie, *Greedy-Available Non-contiguous Processor Allocation Strategy and Job Scheduling for 2D Mesh Connected Multicomputers*, Proceedings of the 11th International CSI Computer Conference (CSICC 2006), School of Computer Science, IPM, Tehran, Iran, pp. 122-130, January 24-26, 2006. This paper has been selected for the special issue in International Journal of Computers and their Applications, ISCA Press.

- S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh and Lewis M. Mackenzie, *An Efficient Turning Busy List Sub-mesh Allocation Strategy for 3D Mesh Connected Multicomputers*, Proceedings of the 7th Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking & Broadcasting, (PGNET 2006), Liverpool John Moores University, UK, pp. 37-43, 26-27 June 2006.

- Bani-Mohammad S., Ould-Khaoua M., and Ababneh I., *Performance Evaluation of Processor Allocation Strategies in the 2-Dimensional Mesh Network*, N. Thomas (editor), Proceedings of the 21st UK Performance Engineering Workshop (UKPEW 2005), School of Computing Science, Technical Report Series, CS-TR-916, University of Newcastle, UK, ISSN 1368-1060. pp. 177-188, 14-15 July 2005.

- Bani-Mohammad S., Ould-Khaoua M., and Ababneh I., *A Simulation Study of Allocation Strategies on the Mesh Interconnection Networks*, Proceedings of the 6th Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking & Broadcasting, (PGNET 2005), Liverpool John Moores University, UK, ISBN 1-902-56011-6, pp. 197-202, 27-28 June 2005.

Technical Reports

- S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh and Lewis M. Mackenzie, *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, January 2007.

- Bani-Mohammad S., Ould-Khaoua M., and Ababneh I., *Performance Analysis of Processor Allocation Strategies on 2D-Mesh Interconnection Networks*, Technical Report; TR-2005-202, DCS Technical Report Series, Department of Computing Science, University of Glasgow, June 2005.

- Bani-Mohammad S., Ould-Khaoua M., and Ababneh I., *A Simulation Study of Allocation Strategies on the Mesh Interconnection Networks*, Technical Report; TR-2005-194, DCS Technical Report Series, Department of Computing Science, University of Glasgow, April 2005.

References

[1] A. A. Chien and J. K. Kim, Planar adaptive routing: low cost adaptive networks for multiprocessors, *Proceedings of the 19th International Symposium on Computer Architecture*, pp. 268-277, 1992.

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

[3] A. Al-Dubai, M. Ould-Khaoua, and L. M. Mackenzie, An efficient path-based multicast algorithm for mesh networks, *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS)*, Nice, France, IEEE Computer Society Press , pp. 283-290, 22 -26 April, 2003.

[4] A. Al-Dubai, Towards Efficient Collective Communication in Multicomputer Interconnection Networks, Ph.D. Thesis, Department of Computing Science, University of Glasgow, 2004.

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

[6] A. I. D. Bucur and D. H. J. Epema, Scheduling Policies for Processor Coallocation in Multicluster Systems, *IEEE Transaction on Parallel and Distributed Systems*, vol. 18, no. 7, pp. 958-972, 2007.

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

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

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

[10] Blue Gene Project, *http://www.research.ibm.com/bluegene/index.html*, 2007.

[11] C. A. F. De Rose, H.-U. Heiss, and B. Linnert, Distributed Dynamic processor Allocation for Multicomputers, *Parallel Computing*, vol. 33, no. 3, pp. 145-158, 2007.

[12] C. G. Glass and L. M. Ni, The turn model for adaptive routing, *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pp. 278-287, 1992.

[13] C. J. Drewes, 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, 1996.

[14] C. P. Kruskal and M. Snir, The performance of multistage interconnection networks for multiprocessors, *IEEE Trans. Computers*, vol. 32, no. 12, pp. 1091-1098, 1983.

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

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

[17] C.-S. Wu, Processor scheduling in multiprogrammed shared memory NUMA multiprocessors, M.Sc. Thesis, Department of Computer Science, University of Toronto, 1993.

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

[19] Cray, Cray XT3 Datasheet, 2005.

[20] D. Babbar and P. Krueger, A performance Comparison of Processor Allocation and Job Scheduling Algorithms for Mesh-Connected Multiprocessors, *Proceedings of the 6th IEEE Symposium on Parallel and Distributed Processing*, pp. 46-53, 1994.

[21] D. Das Sharma and D. K. Pradhan, 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, 1996.

[22] D. G. Feitelson, Workload Modeling for Computer Systems Performance Evaluations, *http://www.cs.huji.ac.il/~feit/wlmod/wlmod.pdf*, 2007.

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

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

[25] E. Anderson, J. Brooks, C. Grassl, S. Scott, Performance of the Cray T3E multiprocessor, *Proceedings of the ACM/IEEE Supercomputing Conference*, pp. 1-17, 1997.

[26] E. Krevat, J. G. Castannos, and J. E. Moreira, Job Scheduling for the BlueGene/L System, *Proceedings of the Job Scheduling Strategies for Parallel Processing Workshop (JSSPP)*, pp. 38-54, 2002.

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

[28] G. Gabrani and T. Mulkar, A quad tree-based algorithm for processor allocation in 2D mesh-connected multicomputers, *Journal of Computer Standards and Interfaces*, vol. 27, no. 2, pp. 133-147, 2005.

[29] G. Min, Performance Modelling and Analysis of Multicomputer Interconnection Networks, Ph.D. Thesis, Department of Computing Science, University of Glasgow, 2003.

[30] G.-M. Chiu, The odd-even turn model for adaptive routing, *IEEE Transaction on Parallel and Distributed Systems*, vol. 11, no. 7, pp. 729-738, 2000.

[31] H. Choo, S. Yoo, and H.-Y. Youn, Processor scheduling and allocation for 3D torus multicomputer systems, *IEEE Transactions on Parallel & Distributed Systems*, vol. 11, no. 5, pp. 475-484, 2000.

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

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

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

[35] I. Ababneh and F. Fraij, Folding contiguous and non-contiguous space sharing

policies for parallel computers, *Mu'tah Lil-Buhuth wad-Dirasat, Natural and Applied Sciences Series*, vol. 16, no. 3, pp. 9-34, 2001.

[36] I. Foster, Designing and Building Parallel Programs, Concepts and Tools for Parallel Software Engineering, Addison-Wesley, 1995.

[37] I. Ismail, Space sharing job scheduling policies for parallel computers, Ph.D. Thesis, Department of Electrical and Computer Engineering, Iowa State University, 1995.

[38] I. Ismail and J. Davis, Program-based static allocation policies for highly parallel computers, *Proceedings of the IPCCC 95*, IEEE Computer Society Press, pp. 61-68, 1995.

[39] Intel Corp., Paragon XP/S product overview, Supercomputer Systems Division, Beaverton, Oregon, 1991.

[40] Intel Corporation, A Touchstone DELTA system description, 1991.

[41] J. Ding and L.-N. Bhuyan, 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, 1993.

[42] J. Duato, A new Theory of Deadlock-Free Adaptive Routing in Wormhole Networks, *IEEE Transaction on Parallel and Distributed Systems*, vol. 4, no. 12, pp. 1320-1331, 1993.

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

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

[45] J. Mache, V. Lo, and S. Garg, Job Scheduling that Minimizes Network Contention due to both Communication and I/O, *Proceedings of* the *14th International Parallel*

and Distributed Processing Symposium (IPDPS'00), pp. 457-463, 2000.

[46] J. Sua, Processor Allocation in Hypercube Computers, M.Sc. Thesis, Department of Computer Engineering, Faculty of Engineering, Florida Atlantic University, 1993.

[47] J. Wei, X. Zhou, and C-Z. Xu, Robust Processing Rate Allocation for Proportional Slowdown Differentiation on Internet Servers, *IEEE Transactions on Computers*, vol. 54, no. 8, pp. 964-977, 2005.

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

[49] K. Suzaki, H. Tanuma, S. Hirano, Y. Ichisugi, C. Connelly, and M. Tsukamoto, 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, 1996.

[50] K. Windisch, J. V. Miller, and V. Lo, 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, 1995.

[51] K.-H. Seo, 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, 7-9 December, 2005.

[52] K.-H. Seo and S.-C. Kim, 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, 2003.

[53] L. He, S. Jarvis, D. Spooner, H. Jiang, D. Dillenberger, and G. Nudd, Allocating Non-Real-Time and Soft Real-Time Jobs in Multiclusters, *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 2, pp. 99-112, 2006.

[54] L. M. Ni and P. K. McKinley, A survey of wormhole routing techniques in direct networks. *IEEE Computer*, vol. 26, no. 2, pp. 62-76, 1993.

[55] M. Blumrich, D. Chen, P. Coteus, A. Gara, M. Giampapa, P. Heidelberger, S. Singh, B. Steinmacher-Burow, T. Takken, and P. Vranas, Design and Analysis of the BlueGene/L Torus Interconnection Network, *IBM Research Report RC23025*, IBM Research Division, Thomas J. Watson Research Center, Dec. 3, 2003.

[56] M. E. Crovella and A. Bestavros, Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes, *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 835-846, 1997.

[57] M. E. Crovella and L. Lipsky, Long-Lasting Transient Conditions in Simulations with Heavy-Tailed Workloads, *Proceedings of the 1997 Winter Simulation Conference*, pp. 1005-1012, 7-10 Dec., 1997.

[58] M. H.-Balter, M. E. Crovella, and C. D. Murta, On Choosing a Task Assignment Policy for a Distributed Server System, *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 204-228, 1999.

[59] M. H.-Balter, The Effect of Heavy-Tailed Job Size Distributions on Computer System Design, *Proceedings of ASA-IMS Conference on Applications of Heavy Tailed Distributions in Economics, Engineering and Statistics*, Washington, DC, June 1999.

[60] M. Levine, CRAY XT3 at the Pittsburgh Supercomputing Centre, *DEISA Symposium*, Bologna, 4-5 May 2006.

[61] M. Noakes, D. A. Wallach, and W. J. Dally, The J-machine multicomputer: an architecture evaluation, *Proceedings of the 20th International Symposium Computer*

Architecture, pp. 224-235, 1993.

[62] N. Alzeidi, Performance Analysis of Wormhole Switched Interconnection Networks with Virtual Channels and Finite Buffers, Ph.D. Thesis, Department of Computing Science, University of Glasgow, 2007.

[63] P. Krueger, T. Lai, and V. A. Radiya, Job scheduling is more important than processor allocation for hypercube computers, *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 5, pp. 488-497, 1994.

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

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

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

[67] R. E. Kessler and J. L. Swarszmeier, Cray T3D: a new dimension for Cray research, *Proceedings of the 38th Annual International Computer Conference (COMPCON SPRING'93)*, pp. 176-182, IEEE Computer Society Press, 1993.

[68] R. Jan, The Art of Computer Systems Performance Analysis, John Wiley & Sons, Inc., New York, 1991.

[69] R. V. Boppana and S. Chalasani, Framework for designing deadlock-free wormhole routing algorithms, *IEEE Transaction on Parallel and Distributed Systems*, vol. 7, no. 2, pp. 169-183, 1996.

[70] S. A. Ghozati and H. C. Wasserman, The *k*-ary *n*-cube network: modelling, topological properties and routing strategies, *Computers and Electrical Engineering*, vol. 25, no. 3, pp. 155-168, May 1999.

[71] S. Bani-Mohammad, M. Ould-Khaoua, and I. Ababneh, 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, 2007.

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

[73] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, and L. Machenzie, 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, January 2007.

[74] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, and L. Machenzie, 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, 2006.

[75] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, and L. Machenzie, 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, 2007.

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

[77] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, and L. Machenzie, Non-

contiguous Processor Allocation Strategy for 2D Mesh Connected Multicomputers Based on Sub-meshes Available for Allocation, *Proceedings of the 12th International Conference on Parallel and Distributed Systems (ICPADS'06)*, Minneapolis, Minnesota, USA, IEEE Computer Society Press, vol. 2 , pp. 41-48, 2006.

[78] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, and Lewis M. Mackhenzie, An Efficient Turning Busy List Sub-mesh Allocation Strategy for 3D Mesh Connected Multicomputers, *Proceedings of the 7th Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking & Broadcasting, (PGNET 2006)*, Liverpool John Moores University, UK, pp. 37-43, 26-27 June 2006.

[79] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, and Lewis M. Mackhenzie, 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, 13-16 May 2007.

[80] T. Liu, W.-K. Huang, F. Lombardi, and L. N. Bhuyan, A Submesh Allocation Scheme for Mesh-Connected Multiprocessor Systems, *Proceedings of the International Conference Parallel Processing II*, pp. 159-163, 1995.

[81] T. Srinivasan, J. Seshadri, A. Chandrasekhar, and J. Jonathan, A Minimal Fragmentation Algorithm for Task Allocation in Mesh-Connected Multicomputers, *Proceedings of the IEEE International Conference on Advances in Intelligent Systems – Theory and Applications – AISTA'04*, ISBN 2-9599-7768-8, 15-18 Nov, Luxembourg, Western Europe, IEEE Computer Society, IEEE Press, 2004.

[82] V. Adve and M. K. Vernon, Performance analysis of mesh interconnection networks with deterministic routing, *IEEE Trans. Parallel & Distributed Systems*,

vol. 5, no. 3, pp. 225-246, 1994.

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

[84] V. Leung, E. Arkin, M. Bender, D. Bunde, J. Johnston, A. Lal, J. Mitchell, C. Phillips, and S. Seiden, 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, 2002.

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

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

[87] V. Naik, S. Setia, and M. Squillante, Scheduling of large scientific applications on distributed memory multiprocessor systems. *Proceedings of the 6th SIAM Conference on Parallel Processing for Scientific Computing*, pp. 913-922, 1993.

[88] V. Tabatabaee, A. Tiwari, and J. K. Hollingsworth, Parallel Parameter Tuning for Applications with Performance Variability, *SC'05*, Seattle WA, November 2005.

[89] V. Varavithya, Multicasting in wormhole routed multicomputers, Ph.D. Thesis, Department of Electrical and Computer Engineering, Iowa State University, 1998.

[90] W. Athas and C. Seitz, Multicomputers: message-passing concurrent computers, *IEEE Computer*, vol. 21, no. 8, pp. 9-24, 1988.

[91] W. J. Dally and C. L. Seitz, Deadlock-Free message routing in multiprocessor interconnection networks, *IEEE Transaction on Computers*, vol. 36, no. 5, pp. 547-

553, 1987.

[92] W. J. Dally and H. Aoki, Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels, *IEEE Transaction on Parallel and Distributed Systems*, vol. 4, no. 4, pp. 466-475, 1993.

[93] W. Mao, J. Chen, and W. Watson, Efficient Subtorus Processor Allocation in a Multi-Dimensional Torus, *Proceedings of the 8th International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA'05)*, IEEE Computer Society Press, pp. 53-60, 30 November - 3 December, 2005.

[94] W. Qiao and L. Ni, Efficient processor allocation for 3D tori, *Proceedings of the 9th International Conference on Parallel Processing Symposium*, IEEE Computer Society Press, pp. 466-471, 1995.

[95] X. Lin, P. McKinley, and L. M. Ni, Deadlock-free multicast wormhole routing in 2D mesh multicomputers, *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 8, pp. 793-804, 1994.

[96] X. Tang and S. T. Chanson, Optimizing Static Job Scheduling in a Network of Heterogeneous Computers, *Proceedings of the 2000 International Conference on Parallel Processing (ICPP)*, IEEE Computer Society Press, pp. 373-382, 2000.

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

[98] Y. Aridor, T. Domany, O. Goldshmidt, Y. Kliteynik, J. Moreira, and E. Shmueli, Open Job Management Architecture for the Blue Gene/L Supercomputer, *Proceedings of the 11th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'05)*, June 19, Cambridge, pp. 91-107, 2005.

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

337, 1992.

[100] Y.-J. Tsai and P. McKinley, An extended dominating node approach to broadcast and global combine in multiport wormhole-routed mesh networks, *IEEE Transactions on Parallel & Distributed Systems*, vol. 8, no. 1, pp. 41-58, 1997.