

A Fast and Efficient Strategy for Sub-mesh Allocation with Minimal Allocation Overhead in 3D Mesh Connected Multicomputers

S. Bani-Mohammad¹, M. Ould-Khaoua¹, I. Ababneh² and Lewis M. Mackenzie¹

¹Department of Computing Science
University of Glasgow, Glasgow G12 8QQ, UK.
Email: {saad, mohamed, lewis}@dcs.gla.ac.uk

²Department of Computing Science
Al al-Bayt University, Mafraq, Jordan.
Email: ismail@aabu.edu.jo

Abstract-This paper presents a fast and efficient contiguous allocation strategy for 3D mesh multicomputers, 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. Turning means that the orientation of the allocation request is changed when no sub-mesh is available in the requested orientation. The *TBL* strategy relies on a new approach that maintains a list of allocated sub-meshes to determine all the regions consisting of 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 allocated sub-meshes to find the nodes that can be used as base nodes for the required sub-mesh size. Results from extensive simulations under a variety of system loads confirm that the *TBL* strategy incurs much less allocation overhead than all of the existing contiguous allocation strategies for 3D mesh multicomputers and delivers competitive performance in terms of parameters such as the average turnaround times and system utilization. Moreover, the time complexity of the *TBL* strategy is much lower than that of the existing strategies.

Keywords-Contiguous Allocation, Turnaround Time, Utilization, Allocation Overhead, Switching Request Orientation, Simulation.

I. INTRODUCTION

Multicomputers, consisting of many processing elements (or nodes) connected through a high-speed interconnection network, have been a prevalent computing platform for many real world scientific and engineering applications [3]. The mesh has been one of the most common networks for existing multicomputers due to its simplicity, scalability, structural regularity, and

ease of implementation [2, 3, 9, 15].

Efficient processor allocation and job scheduling are critical to achieve and harness the full computing power of a multicomputer [3, 7, 8, 23]. Processor allocation is responsible for selecting the set of processors on which parallel jobs are executed while job scheduling is responsible for determining the order in which the jobs are executed [3]. An incoming job specifies the side lengths of the sub-mesh it requires before joining the queue. The job scheduler selects the next job for execution using the underlying scheduling policy and then the processor allocator finds an available sub-mesh for the selected job.

In distributed memory multicomputers, jobs are allocated distinct contiguous processor sub-mesh for the duration of their execution [3, 7, 8, 9, 10, 15, 23, 24]. Most existing research studies [3, 7, 9, 14, 15, 24] 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 iWARP [4], the Cray XT3 [5], and Delta Touchstone [12], most practical multicomputers, like the MIT J-Machine [25], Cray T3D [18], the IBM BlueGene/L [1, 16], and Cray T3E [6], have used the 3D mesh as the underlying network topology due to its lower diameter and average communication distance [21].

The main shortcoming of existing contiguous allocation strategies for 3D mesh [8, 10, 23] is that they achieve complete sub-mesh recognition capability with high allocation overhead.

An efficient sub-mesh allocation strategy must be efficient in the time it takes for both allocation and deallocation (i.e., allocation overhead). In addition to its own efficiency, the strategy must deliver competitive performance. The performance is measured in terms of overall performance parameters such as the average turnaround time, mean allocation time and mean system utilization. In this paper, we propose a new fast and efficient contiguous allocation strategy that supports the rotation of job requests. The proposed strategy has lower time complexity than the existing strategies, yet the simulation results show that its system performance is as good as that of the promising previously proposed strategies. Moreover, the mean measured allocation time of the proposed strategy is much lower than that of the previous strategies.

The rest of the paper is organised as follows. Section 2 contains a brief summary of allocation strategies previously proposed for the 3D mesh. Section 3 contains a set of relevant preliminaries. Section 4 contains the proposed contiguous allocation strategy, and its complexity analysis is given in Section 5. Simulation results are presented in Section 6. Section 7 concludes this study.

II. RELATED WORK

Contiguous allocation has been investigated for 2D and 3D mesh-connected multicomputers [3, 7, 8, 9, 10, 14, 15, 23, 24]. The main shortcoming of the very few existing contiguous allocation strategies for the 3D mesh is that they achieve complete sub-mesh recognition capability with high allocation overhead. Below we describe some of these strategies.

Contiguous First Fit and Best Fit Allocation for the 3D Mesh: In [10], turning the allocation request is used to improve the performance of contiguous First Fit and Best Fit allocation in 3D mesh. Simulation results have shown that First Fit with rotation can greatly improve performance in terms of average turnaround time and scheduling effectiveness. Moreover, the performance of First Fit is almost identical to that of Best Fit.

First Fit (FF), Best Fit (BF), and Worst Fit (WF) for the 3D Torus Mesh: The contiguous allocation strategies First Fit, Best Fit and Worst Fit have been investigated and compared using simulation for the 3D torus (a torus is a mesh with wraparound links) and the mesh networks

when jobs are scheduled using First-Come-First-Served (*FCFS*). Changes to request orientation are allowed; that is, if for example, a request for $a \times b \times c$ processors could not be satisfied, it could be reoriented to become $a \times c \times b$ [23]. Simulation results in [23] have shown that these three strategies have comparable performance; the performance of *FF* is close to that of *BF*, and it is better than that of *WF*.

An Efficient First Fit on the 3D torus: An efficient processor allocation scheme for 3D torus which has complete recognition capability has been used to improve the performance of contiguous First Fit allocation on the 3D torus [8]. The results show that the average allocation time of the new scheme is much smaller than that of the earlier scheme [23] that is based on the Best Fit approach for practical ranges of input load. Moreover, the time complexity for the new scheme is much lower. This is achieved by an efficient search mechanism proposed for finding a free sub-mesh. In [8], different scheduling strategies, such as *FCFS* and *ScanAll*, have been studied to avoid potential performance loss due to blocking.

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 cause no inter-job contention in the network. However, the time complexities of the above allocation strategies grow with the size of the mesh so that they achieve complete sub-mesh recognition capability but with high allocation overhead.

In our proposed strategy, the mean measured allocation time is much lower than that of the previous strategies and also our strategy delivers competitive performance. Moreover, the time complexity of allocation and deallocation operation is lower than that of the previous strategies and do not grow with the size of the mesh as in previous strategies.

III. PRELIMINARIES

The target system is a $W \times D \times H$ 3D mesh, 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 $0 \leq x \leq W - 1$, $0 \leq y \leq D - 1$ and $0 \leq z \leq H - 1$ [19]. A processor is connected by bidirectional communication links to its neighbour

processors. The following definitions have been adopted from [7, 19].

Definition 1: A sub-mesh $S(w,d,h)$ of width w , depth d , and height h , where $0 \leq w \leq W-1$, $0 \leq d \leq D-1$ and $0 \leq h \leq H-1$ is specified by the coordinates (x,y,z) and (x',y',z') , where (x,y,z) are the coordinates of the base of the sub-mesh and (x',y',z') are the coordinates of its end, as shown in Fig. 1.

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

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 $S(x,y,z)$ is a free sub-mesh that satisfies the conditions: $x \geq a$, $y \geq b$ and $z \geq c$ assuming that the allocation of $S(a,b,c)$ 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.

Definition 8: 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 $\max(y_1-\beta+1,0) \leq y' \leq y_2$ and $\max(z_1-\gamma+1,0) \leq z' \leq z_2$. A RBP of sub-mesh S is a plane located just off the right boundary of S .

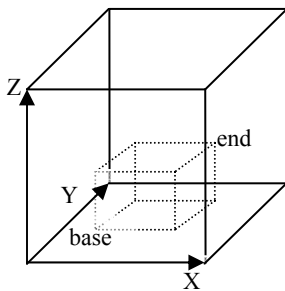


Fig. 1: A submesh inside the 3D mesh.

IV. PROPOSED ALLOCATION STRATEGY

The proposed allocation strategy is based on

maintaining a busy list of allocated sub-meshes. The list is scanned to determine all *prohibited regions*.

A *prohibited region* of job $J(\alpha \times \beta \times \gamma)$ with respect to an allocated sub-mesh $S(x_1,y_1,z_1,x_2,y_2,z_2)$ in the busy list 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 $J(2 \times 2 \times 2)$ with respect to the allocated sub-mesh $(1,1,0,2,2,1)$, is the sub-mesh $(0,0,0,2,2,1)$.

The 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 width and /or depth and /or height bounds 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.

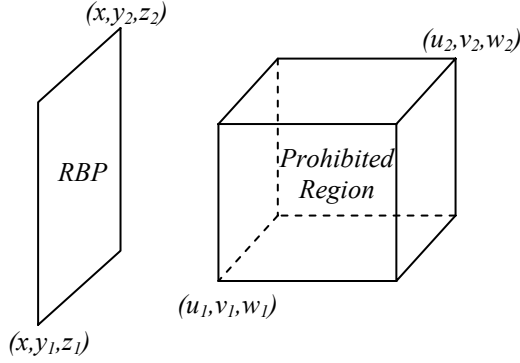
A job $J(\alpha \times \beta \times \gamma)$ is allocatable if there exists at least one node that does not belong to any of the prohibited regions and the three automatic prohibited regions of $J(\alpha \times \beta \times \gamma)$.

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. Fig. 2 shows all possible cases for subtracting prohibited regions from a RBP. The algorithm that is used to detect the base nodes for any new job request is formally presented in Fig. 3, and the allocation algorithm is presented in Fig. 4.

To facilitate the presentation of the algorithm, 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 of the mesh. The list *RBP_Nodes* contains a plane if its nodes are available for allocation to a job $J(\alpha \times \beta \times \gamma)$ selected for execution.

The proposed allocation algorithm supports the rotation of the job request. Let (a,b,c) be the width, depth and height of a sub-mesh allocation request. The six

permutations (a,b,c) , (a,c,b) , (b,a,c) , (b,c,a) , (c,a,b) and (c,b,a) are, in turn, considered for allocation using the proposed allocation strategy. 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 allocated until it is rotated to $(1, 3, 2)$.



- 2.1 $((x < u_1) \vee (x > u_2) \vee (z_2 < w_1) \vee (z_1 > w_2) \vee (y_2 < v_1) \vee (y_1 > v_2))$
In this case the result is RBP itself.
- 2.2 $(u_1 \leq x \leq u_2) \wedge (y_2 = v_1) \wedge (y_1 < v_1) \wedge (w_1 \leq z_2 \leq w_2) \wedge (z_1 < w_1)$
RBP1 $(x, y_1, z_1, x, y_2, w_1 - 1)$; RBP2 $(x, y_1, z_1, x, v_1 - 1, z_2)$
- 2.3 $(u_1 \leq x \leq u_2) \wedge (y_2 > v_2) \wedge (y_1 = v_2) \wedge (w_1 \leq z_2 \leq w_2) \wedge (z_1 < w_1)$
RBP1 $(x, y_1, z_1, x, y_2, w_1 - 1)$; RBP2 $(x, v_2 + 1, w_1, x, y_2, z_2)$
- 2.4 $(u_1 \leq x \leq u_2) \wedge (v_1 \leq y_1 \leq v_2) \wedge (v_1 \leq y_2 \leq v_2) \wedge (z_1 = w_2) \wedge (z_2 > w_2)$
RBP $(x, y_1, w_2 + 1, x, y_2, z_2)$
- 2.5 $(u_1 \leq x \leq u_2) \wedge (v_1 \leq y_1 \leq v_2) \wedge (y_2 > v_2) \wedge (z_1 = w_2) \wedge (z_2 > w_2)$
RBP1 $(x, y_1, w_2 + 1, x, y_2, z_2)$; RBP2 $(x, v_2 + 1, z_1, x, y_2, w_2)$
- 2.6 $(u_1 \leq x \leq u_2) \wedge (v_1 \leq y_2 \leq v_2) \wedge (y_1 < v_1) \wedge (z_1 = w_2) \wedge (z_2 > w_2)$
RBP1 $(x, y_1, w_2 + 1, x, y_2, z_2)$; RBP2 $(x, y_1, z_1, x, v_1 - 1, w_2)$
- 2.7 $(u_1 \leq x \leq u_2) \wedge (v_1 \leq y_1 \leq v_2) \wedge (v_1 \leq y_2 \leq v_2) \wedge (z_2 = w_1) \wedge (z_1 < w_1)$
RBP $(x, y_1, z_1, x, y_2, w_1 - 1)$
- 2.8 $(u_1 \leq x \leq u_2) \wedge (v_1 \leq y_1 \leq v_2) \wedge (y_2 > v_2) \wedge (z_2 = w_1) \wedge (z_1 < w_1)$
RBP1 $(x, y_1, z_1, x, y_2, w_1 - 1)$; RBP2 $(x, v_2 + 1, w_1, x, y_2, z_2)$
- 2.9 $(u_1 \leq x \leq u_2) \wedge (v_1 \leq y_2 \leq v_2) \wedge (y_1 < v_1) \wedge (z_2 = w_1) \wedge (z_1 < w_1)$
RBP1 $(x, y_1, z_1, x, y_2, w_1 - 1)$; RBP2 $(x, y_1, w_1, x, v_1 - 1, z_2)$
- 2.10 $(u_1 \leq x \leq u_2) \wedge (v_1 \leq y_1 \leq v_2) \wedge (v_1 \leq y_2 \leq v_2) \wedge (w_1 < z_2 \leq w_2) \wedge (z_1 < w_1)$
RBP $(x, y_1, z_1, x, y_2, w_1 - 1)$
- 2.11 $(u_1 \leq x \leq u_2) \wedge (v_1 \leq y_1 \leq v_2) \wedge (y_2 > v_2) \wedge (w_1 < z_2 \leq w_2) \wedge (z_1 < w_1)$
RBP1 $(x, y_1, z_1, x, y_2, w_1 - 1)$; RBP2 $(x, v_2 + 1, w_1, x, y_2, z_2)$
- 2.12 $(u_1 \leq x \leq u_2) \wedge (v_1 \leq y_2 \leq v_2) \wedge (y_1 < v_1) \wedge (w_1 < z_2 \leq w_2) \wedge (z_1 < w_1)$
RBP1 $(x, y_1, z_1, x, y_2, w_1 - 1)$; RBP2 $(x, y_1, w_1, x, v_1 - 1, z_2)$

- 2.13 $(u_1 \leq x \leq u_2) \wedge (v_1 \leq y_1 \leq v_2) \wedge (v_1 \leq y_2 \leq v_2) \wedge (w_1 \leq z_1 \leq w_2) \wedge (z_2 > w_2)$
RBP $(x, y_1, w_2 + 1, x, y_2, z_2)$
- 2.14 $(u_1 \leq x \leq u_2) \wedge (v_1 \leq y_1 \leq v_2) \wedge (y_2 > v_2) \wedge (w_1 \leq z_1 \leq w_2) \wedge (z_2 > w_2)$
RBP1 $(x, v_2 + 1, z_1, x, y_2, w_2)$; RBP2 $(x, y_1, w_2 + 1, x, y_2, z_2)$
- 2.15 $(u_1 \leq x \leq u_2) \wedge (v_1 \leq y_2 \leq v_2) \wedge (y_1 < v_1) \wedge (w_1 \leq z_1 \leq w_2) \wedge (z_2 > w_2)$
RBP1 $(x, y_1, z_1, x, v_1 - 1, w_2)$; RBP2 $(x, y_1, w_2 + 1, x, y_2, z_2)$
- 2.16 $(u_1 \leq x \leq u_2) \wedge (v_1 \leq y_1 \leq v_2) \wedge (v_1 \leq y_2 \leq v_2) \wedge (z_1 < w_1) \wedge (z_2 > w_2)$
RBP1 $(x, y_1, z_1, x, y_2, w_1 - 1)$; RBP2 $(x, y_1, w_2 + 1, x, y_2, z_2)$
- 2.17 $(u_1 \leq x \leq u_2) \wedge (v_1 \leq y_1 \leq v_2) \wedge (y_2 > v_2) \wedge (z_1 < w_1) \wedge (z_2 > w_2)$
RBP1 $(x, y_1, z_1, x, y_2, w_1 - 1)$; RBP2 $(x, v_2 + 1, z_1, x, y_2, z_2)$
RBP3 $(x, y_1, w_2 + 1, x, v_2, z_2)$
- 2.18 $(u_1 \leq x \leq u_2) \wedge (v_1 \leq y_2 \leq v_2) \wedge (y_1 < v_1) \wedge (z_1 < w_1) \wedge (z_2 > w_2)$
RBP1 $(x, y_1, z_1, x, v_1 - 1, z_2)$; RBP2 $(x, v_1, z_1, x, y_2, w_1 - 1)$
RBP3 $(x, v_1, w_2 + 1, x, y_2, z_2)$
- 2.19 $(u_1 \leq x \leq u_2) \wedge (y_2 > v_2) \wedge (y_1 < v_1) \wedge (z_1 < w_1) \wedge (z_2 > w_2)$
RBP1 $(x, y_1, z_1, x, v_1 - 1, z_2)$; RBP2 $(x, v_2 + 1, z_1, x, y_2, z_2)$
RBP3 $(x, v_1, z_1, x, v_2, w_1 - 1)$; RBP4 $(x, v_1, w_2 + 1, x, v_2, z_2)$
- 2.20 $(u_1 \leq x \leq u_2) \wedge (y_2 > v_2) \wedge (y_1 < v_1) \wedge (z_1 < w_1) \wedge (z_2 = w_2)$
RBP1 $(x, y_1, z_1, x, v_1 - 1, z_2)$; RBP2 $(x, v_2 + 1, z_1, x, y_2, z_2)$
RBP3 $(x, v_1, z_1, x, v_2, w_1 - 1)$
- 2.21 $(u_1 \leq x \leq u_2) \wedge (y_2 > v_2) \wedge (y_1 < v_1) \wedge (z_1 = w_1) \wedge (z_2 > w_2)$
RBP1 $(x, y_1, z_1, x, v_1 - 1, z_2)$; RBP2 $(x, v_2 + 1, z_1, x, y_2, z_2)$
RBP3 $(x, v_1, w_2 + 1, x, v_2, z_2)$
- 2.22 $(u_1 \leq x \leq u_2) \wedge (y_2 > v_2) \wedge (y_1 < v_1) \wedge (z_1 = w_1) \wedge (z_2 = w_2)$
RBP1 $(x, y_1, z_1, x, v_1 - 1, z_2)$; RBP2 $(x, v_2 + 1, z_1, x, y_2, z_2)$
- 2.23 $(u_1 \leq x \leq u_2) \wedge (y_2 > v_2) \wedge (y_1 < v_1) \wedge (z_1 = w_1) \wedge (z_2 < w_2)$
RBP1 $(x, y_1, z_1, x, v_1 - 1, z_2)$; RBP2 $(x, v_2 + 1, z_1, x, y_2, z_2)$
- 2.24 $(u_1 \leq x \leq u_2) \wedge (y_2 > v_2) \wedge (y_1 < v_1) \wedge (z_1 > w_1) \wedge (z_2 = w_2)$
RBP1 $(x, y_1, z_1, x, v_1 - 1, z_2)$; RBP2 $(x, v_2 + 1, z_1, x, y_2, z_2)$
- 2.25 $(u_1 \leq x \leq u_2) \wedge (y_2 > v_2) \wedge (y_1 < v_1) \wedge (z_1 < w_1) \wedge (w_1 \leq z_2 < w_2)$
RBP1 $(x, y_1, z_1, x, v_1 - 1, z_2)$; RBP2 $(x, v_2 + 1, z_1, x, y_2, z_2)$
RBP3 $(x, v_1, z_1, x, v_2, w_1 - 1)$
- 2.26 $(u_1 \leq x \leq u_2) \wedge (y_2 > v_2) \wedge (y_1 < v_1) \wedge (z_2 > w_2) \wedge (w_1 \leq z_1 < w_2)$
RBP1 $(x, y_1, z_1, x, v_1 - 1, z_2)$; RBP2 $(x, v_2 + 1, z_1, x, y_2, z_2)$
RBP3 $(x, v_1, w_2 + 1, x, v_2, z_2)$
- 2.27 $(u_1 \leq x \leq u_2) \wedge (v_1 \leq y_1 \leq v_2) \wedge (w_1 \leq z_1 \leq w_2) \wedge (w_1 \leq z_2 \leq w_2)$
No RBP in this case.
- 2.28 $(u_1 \leq x \leq u_2) \wedge (v_1 \leq y_1 \leq v_2) \wedge (y_2 > v_2) \wedge (w_1 \leq z_1 \leq w_2) \wedge (w_1 \leq z_2 \leq w_2)$
RBP $(x, v_2 + 1, z_1, x, y_2, z_2)$
- 2.29 $(u_1 \leq x \leq u_2) \wedge (v_1 \leq y_2 \leq v_2) \wedge (y_1 < v_1) \wedge (w_1 \leq z_1 \leq w_2) \wedge (w_1 \leq z_2 \leq w_2)$
RBP $(x, y_1, z_1, x, v_1 - 1, z_2)$

Fig. 2: 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 $B = \{b_0, b_1, b_2, \dots, b_m\}$ where b_0 is a hypothetical allocated sub-mesh and $b_i, 1 \leq i \leq m$, are the m already allocated sub-meshes; Both sub-meshes $(0, d-\beta+1, 0, w-1, d-1, h-1)$, $(w-\alpha+1, 0, 0, w-1, d-1, h-1)$, and $(0, 0, h-\gamma+1, w-1, d-1, h-1)$ 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. $RBP_Nodes \leftarrow NULL$.

Step 2. for each allocated sub-mesh b_i 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 where $x_r = x_2 + 1$, $y_{r1} = \max(y_1 - \beta + 1, 0)$, $z_{r1} = \max(z_1 - \gamma + 1, 0)$, $y_{r2} = y_2$ and $z_{r2} = z_2$.

Step 2.2. if RBP_i is within any automatic prohibited region then goto Step 2.

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 $F_j = (x_{f1}, y_{f1}, z_{f1}, x_{f2}, y_{f2}, z_{f2})$ where $x_{f1} = \max(x_1 - \alpha + 1, 0)$, $y_{f1} = \max(y_1 - \beta + 1, 0)$, $z_{f1} = \max(z_1 - \gamma + 1, 0)$, $x_{f2} = x_2$, $y_{f2} = y_2$ and $z_{f2} = z_2$.

Subtract F_j from RBP_i as follows:

Determine the case to which the subtraction belongs by comparing the coordinates of RBP_i and F_j as in Fig. 2.

Switch (subtraction case)

{

case (1): if ($z_{r1} > z_{f2}$) then

begin

add the RBP in Fig. 2.1 to RBP_nodes .

goto Step 2.

end

break.

case (2): adjust RBP_i as in Fig. 2.2; break. case (3): adjust RBP_i as in Fig. 2.3; break.

case (4): add the RBP in Fig. 2.4 to RBP_nodes .; goto Step 2.

case (5): adjust RBP_i as in Fig. 2.5; break. case (6): adjust RBP_i as in Fig. 2.6; break.

case (7): adjust RBP_i as in Fig. 2.7; break. case (8): adjust RBP_i as in Fig. 2.8; break.

case (9): adjust RBP_i as in Fig. 2.9; break. case (10): adjust RBP_i as in Fig. 2.10; break.

case (11): adjust RBP_i as in Fig. 2.11; break. case (12): adjust RBP_i as in Fig. 2.12; break.

case (13): add the RBP in Fig. 2.13 to RBP_nodes .; goto Step 2.

case (14): adjust RBP_i as in Fig. 2.14; break. case (15): adjust RBP_i as in Fig. 2.15; break.

case (16): adjust RBP_i as in Fig. 2.16; break. case (17): adjust RBP_i as in Fig. 2.17; break.

case (18): adjust RBP_i as in Fig. 2.18; break. case (19): adjust RBP_i as in Fig. 2.19; break.

case (20): adjust RBP_i as in Fig. 2.20; break. case (21): adjust RBP_i as in Fig. 2.21; break.

case (22): adjust RBP_i as in Fig. 2.22; break. case (23): adjust RBP_i as in Fig. 2.23; break.

case (24): adjust RBP_i as in Fig. 2.24; break. case (25): adjust RBP_i as in Fig. 2.25; break.

case (26): adjust RBP_i as in Fig. 2.26; break. case (27): go to Step 2.

case (28): adjust RBP_i as in Fig. 2.28; break. case (29): adjust RBP_i as in Fig. 2.29; break.

}

goto Step 2.3.

$TBL_Allocate(RBP_Nodes, \alpha, \beta, \gamma)$

}

End.

Fig. 3: Outline of the Detect Procedure in *TBL* Contiguous Allocation Strategy

```

Procedure TBL_Allocate (RBP_Nodes,  $\alpha$ ,  $\beta$ ,  $\gamma$ ):
Begin
{
  int botx, boty, botz, w, d, h;
  botx=RBP_Nodes.botx; boty=RBP_Nodes.boty;
  botz=RBP_Nodes.botz;
  for each  $w_i$  from  $i = \text{botx}$  to  $\text{botx} + \alpha$ 
    for each  $d_j$  from  $j = \text{boty}$  to  $\text{boty} + \beta$ 
      for each  $h_k$  from  $k = \text{botz}$  to  $\text{botz} + \gamma$ 
        Allocate the node ( $w_i, d_j, h_k$ ) for the incoming job.
  }
End.

```

Fig. 4: Outline of the *TBL* Contiguous Allocation Strategy

Example: This example shows how the allocation algorithm works. In this example, we assume the mesh is free, and three requests for the allocation of a $2 \times 4 \times 4$, $2 \times 1 \times 2$ and $1 \times 2 \times 1$ sub-meshes arrive in this order. Fig. 5 gives the states of the processors of a $4 \times 4 \times 4$ mesh. Assume the job request $2 \times 4 \times 4$ is allocated the sub-mesh $(0,0,0,1,3,3)$ as shown in Fig. 5, and then the allocation algorithm is called with a $2 \times 1 \times 2$ allocation request. In this case, the busy list contains the allocated sub-meshes $b_0 : (-1,0,0,-1,3,3)$ and $b_1 : (0,0,0,1,3,3)$ respectively. As for the second job request $2 \times 1 \times 2$, the first *RBP* (*RBP* for the hypothetical allocated sub-mesh b_0 in the busy list) is calculated resulting in $(0,0,0,0,3,3)$. The automatic prohibited regions $(0,4,0,3,3,3)$, $(3,0,0,3,3,3)$ and $(0,0,3,3,3,3)$ with respect to the second allocation request are subtracted from the first *RBP*, resulting in the plane $(0,0,0,0,3,2)$, and 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 $(0,0,0,1,3,3)$ prohibited region, which when subtracted from the plane $(0,0,0,0,3,2)$ results in the NIL value, meaning 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 the new *RBP*, resulting in $(2,0,0,2,3,2)$, and after that the prohibited region of the allocated sub-mesh b_1 is subtracted from the *RBP* $(2,0,0,2,3,3)$, resulting in $(2,0,0,2,3,2)$. Now, any node on the plane $(2,0,0,2,3,2)$ can be used as a base node for the second allocation request, in this example, the node $(2,0,0,2,0,0)$ is used as a base node

for the second allocation request and the sub-mesh $(2,0,0,3,0,1)$ is allocated to the job and then it is added to the busy list, resulting in $\{b_0 : (-1,0,0,-1,3,3), b_1 : (0,0,0,1,3,3), b_2 : (2,0,0,3,0,1)\}$. The same procedure is repeated to allocate a sub-mesh to the third job. The allocated sub-mesh for the allocation request $2 \times 4 \times 4$ is denoted by black circles, the allocated sub-mesh for the allocation request $2 \times 1 \times 2$ is denoted by shaded circles and the allocated sub-mesh for the allocation request $1 \times 2 \times 1$ is denoted by dotted circles.

In the deallocation operation, the allocated sub-mesh is deallocated by removing its corresponding entry in the busy list. The deallocation algorithm is formally presented in Fig. 6.

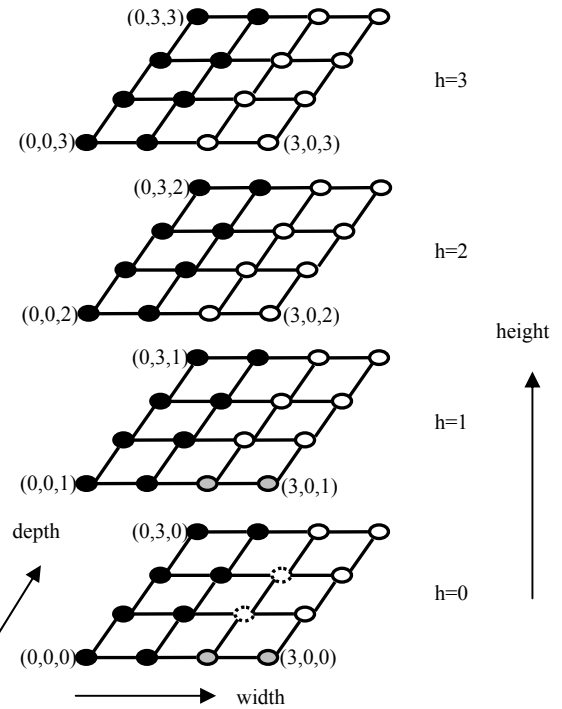


Fig. 5: Allocation Example

```

Procedure TBL_Deallocate ():
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 procedure
End.

```

Fig. 6: Outline of the deallocation algorithm

V. ALLOCATION AND DEALLOCATION TIME COMPLEXITIES

Firstly, we analyse the allocation algorithm. Assume that there are m allocated sub-meshes in the busy list, the *RBP* construction operation in steps 2 and 2.1 requires $O(m)$ time. Scanning the busy list to subtract a prohibited region from a *RBP* requires $O(1)$ time for each subtraction operation. In the worst case there are at most four *RBP*'s and m prohibited regions for subsequent subtraction, therefore the operation of subtracting m prohibited regions from a *RBP* in step 2.3 takes $O(m)$ time. There are a total of $4 \times m$ *RBP*'s and m prohibited regions to be considered, therefore the allocation algorithm takes $O(m^2)$ time. Typically, the average values of m do not grow with n where n is the number of processors in the mesh, as we will see in the simulation results. The deallocation algorithm requires m iterations to remove the allocated sub-mesh from the busy list. Therefore, the deallocation algorithm takes $O(m)$ time. The proposed algorithm maintains a busy list of m allocated sub-meshes. Therefore, the space complexity of the allocation algorithm is $O(m)$. The space incurred by this strategy is small compared to the improvement in performance in terms of allocation time, as we will see in the simulation results.

VI. SIMULATION RESULTS

Extensive simulation experiments have been carried out to compare the performance of the proposed *TBL* allocation strategy against well-known contiguous allocation strategy First Fit [10], with and without change of request orientation. The First Fit strategy allocates an incoming job to the first available sub-mesh that is found [8, 10, 23, 24]. In this study, First Fit has been used to represent the contiguous class of strategies as it has been found to perform well [8, 10, 23, 24]. Switching request orientation has been used in [8, 10, 23].

We have implemented the proposed allocation and deallocation algorithms, including the busy list routines, in the C language, and integrated the software into the ProcSimity simulation tool for processor allocation and scheduling in highly parallel systems [13, 17].

The target mesh is 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 [9, 14, 15, 22]. The execution times are assumed to be exponentially distributed with a mean of one time unit. 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 distribution 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. These distributions have often been used in the literature [3, 7, 9, 10, 11, 14, 15, 19, 20, 23, 24]. 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. The main performance parameters observed are the average turnaround time of jobs, mean system utilization and mean allocation time. The turnaround time is the time that a parallel job spends in the mesh from arrival to departure. The utilization is the percentage of processors that are utilized over time. The allocation time is the time that the allocation algorithm takes to assign a set of jobs to the mesh system. The allocation time that is incurred for detecting the availability of a free sub-mesh for an incoming job request is the realistic time. We recognize that these results are implementation dependent, but the trends shown by the results indicate the features of the strategies. The independent variable in the simulation is the system load. The system load is defined as the inverse of the mean inter-arrival time of jobs. Unless specified otherwise, the performance figures shown below are for $8 \times 8 \times 8$ mesh.

Figures 7 and 8 show simulation results for average allocation time against job arrival rates in a $8 \times 8 \times 8$ mesh when request side lengths follow the uniform and the exponential distributions. We observe that *TBL* is superior to *TFF* in the two figures. In figure 7, for example, the average allocation time of *TBL* is 0.33 of the average allocation time of *TFF* under the arrival rate 4.6 jobs/time unit. It can also be seen in the figures that our proposed *TBL* strategy consistently takes much smaller allocation time than *TFF* strategy regardless of the system load. Moreover, the difference in allocation time gets much more significant as the system load increases. Thus, our proposed strategy can be said to be more effective

than the other strategies represented by *TFF* in these figures.

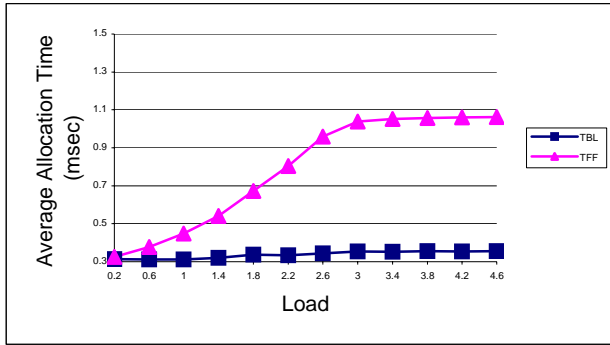


Fig. 7: Average allocation times for the contiguous allocation strategies (*TBL*, *TFF*) and uniform side lengths distribution in a $8 \times 8 \times 8$ mesh.

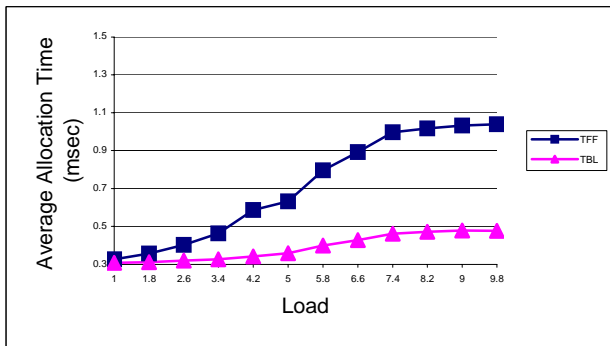


Fig. 8: Average allocation times for the contiguous allocation strategies (*TBL*, *TFF*) and the exponential side lengths distribution in a $8 \times 8 \times 8$ mesh.

In figures 9 and 10, the average turnaround time of jobs are plotted against the system load in a $8 \times 8 \times 8$ mesh for both job size distributions. It can be seen in these figures that the average turnaround times of our strategy *TBL* are very close to those of *TFF*. However, the time complexity of *TBL* is in $O(m^2)$, whereas it is in $O(W \times D \times H)$ for *TFF* [9]. Also the complexity of *TBL* strategy does not grow with the size of the mesh as in *TFF* strategy. It can also be seen in the figures that *TBL* is substantially superior to the strategies *BL* and *FF* without rotation because it is highly likely that a suitable contiguous sub-mesh is available for allocation to a job when request rotation is allowed. In figure 9, for example, the average turnaround times of *TBL* are 0.47, 0.53, and 0.56 of the average turnaround times of *FF* and *BL* under the arrival rates 3.8, 4.2, and 4.6 jobs/time unit, respectively.

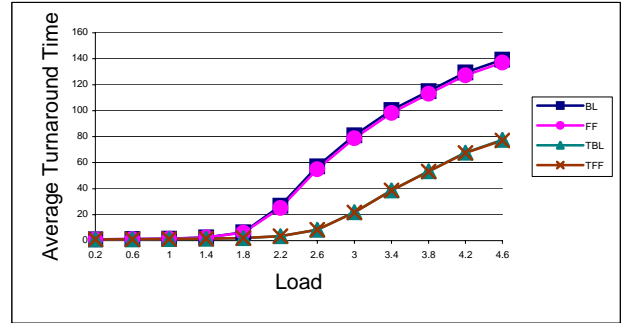


Fig. 9: Average turnaround time vs. system load for the contiguous allocation strategies (*BL*, *FF*, *TBL*, *TFF*) and the uniform side lengths distribution in a $8 \times 8 \times 8$ mesh.

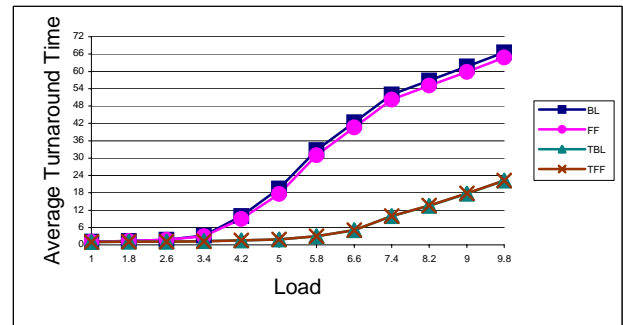


Fig. 10: Average turnaround time vs. system load for the contiguous allocation strategies (*BL*, *FF*, *TBL*, *TFF*) and the exponential side lengths distribution in a $8 \times 8 \times 8$ mesh.

In figures 11 and 12, the mean system utilization is plotted against the system load for both job size distributions. The simulation results show that all strategies have the same utilization under the low loads. For higher loads, the utilization of the strategies that use the rotation of job requests is better than that of the strategies that do not use the rotation of the job requests. For both job size distributions, the contiguous allocation strategies that use the rotation of job requests achieve system utilization of 47% to 49%, but the contiguous allocation strategies that do not use the rotation of job requests can not exceed 36%.

In figures 13 and 14, the average number of allocated sub-meshes (m) in *TBL* is plotted against the system load for both 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, and as clarified in

the allocation and deallocation time complexity section, the average number of allocated sub-meshes (m) is much lower than n for both job size distributions. Moreover, experiments for larger mesh sizes show that m does not grow with n for the job size distributions considered in this paper, as expected.

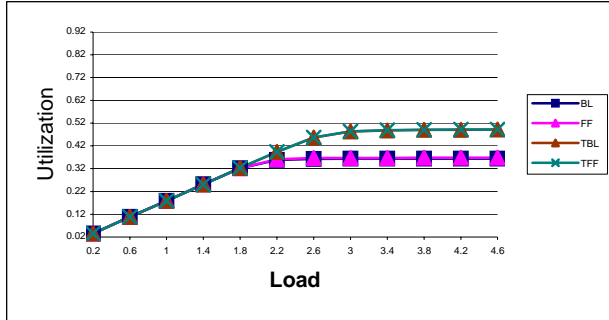


Fig. 11: Mean System utilization for the contiguous allocation strategies (BL , FF , TBL , TFF) and the uniform side lengths distribution in a $8 \times 8 \times 8$ mesh.

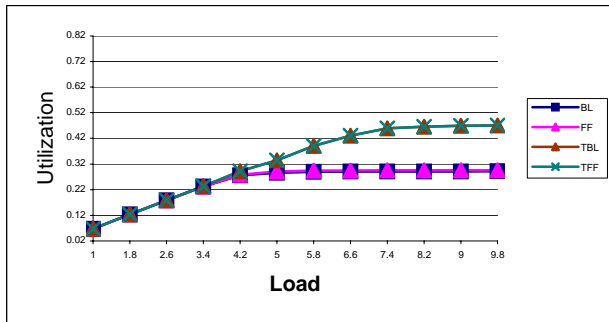


Fig. 12: Mean System utilization for the contiguous allocation strategies (BL , FF , TBL , TFF) and the exponential side lengths distribution in an $8 \times 8 \times 8$ mesh.

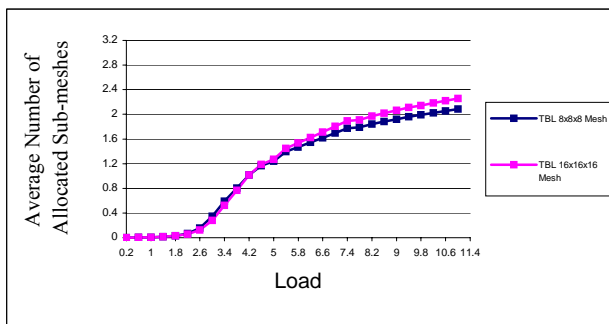


Fig. 13: Average number of allocated sub-meshes (m) in TBL under the uniform side lengths distribution in a $16 \times 16 \times 16$ mesh and $8 \times 8 \times 8$ mesh.

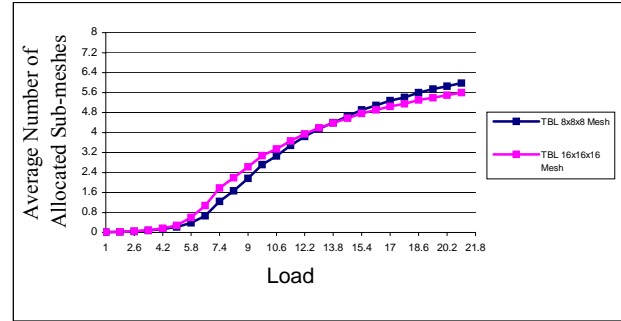


Fig. 14: Average number of allocated sub-meshes (m) in TBL under the exponential side lengths distribution in a $16 \times 16 \times 16$ mesh and $8 \times 8 \times 8$ mesh.

VII. CONCLUSION AND FUTURE DIRECTIONS

While the existing contiguous allocation strategies for 3D mesh achieve complete sub-mesh recognition capability but with high allocation overhead, this study has suggested a fast and efficient contiguous allocation strategy, which overcomes the limitations of the existing strategies. To this end, we have proposed a new efficient contiguous allocation strategy, notably Turning Busy List (TBL) strategy. The TBL strategy can maintain good performance with little allocation overhead. The performance of the TBL strategy has been compared against the existing contiguous allocation strategies. Simulation results have shown that the performance of proposed allocation strategy TBL is at least as good as that of the previously proposed allocation strategies. Moreover, the mean measured allocation time of the TBL strategy is much lower than that of the previous strategies. We also evaluated the switching request orientation. The results have revealed that 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 job sizes typically grow with the size of the mesh. The length of the busy list can be expected to be small, even when the size of the mesh grows.

As a continuation of this research in the future, it would be interesting to evaluate the performance of the contiguous allocation strategies with different scheduling approaches. It would be also interesting to assess the proposed allocation strategy in other common multicomputer networks, such as torus networks. Another possible line for future research is to implement our strategy based on real workload

traces from different parallel machines and compare it with our results obtained by means of simulations.

REFERENCES

- [1] "Blue Gene Project", <http://www.research.ibm.com/bluegene/index.html>, 2005.
- [2] A. Al-Dubai, M. Ould-Khaoua, L. M. Mackenzie, An efficient path-based multicast algorithm for mesh networks, *Proc. 17th Int. Parallel and Distributed Processing Symposium (IPDPS)*, Nice, France, IEEE Computer Society Press, pp. 283-290, 22-26 April, 2003.
- [3] B.-S. Yoo, 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.
- [4] C. Peterson, J. Sutton, P. Wiley, iWARP: a 100-MPOS VLIW microprocessor for multicomputers, *IEEE Micro*, vol. 11, no. 13, 1991.
- [5] Cray, Cray XT3 Datasheet, 2004.
- [6] E. Anderson, J. Brooks, C. Grassl, S. Scott, Performance of the Cray T3E multiprocessor, *Proc. Supercomputing Conference*, pp. 19, 1997.
- [7] G.-M. Chiu, S.-K. Chen, An efficient submesh allocation scheme for two-dimensional meshes with little overhead, *IEEE Transactions on Parallel & Distributed Systems*, vol. 10, no. 5, pp. 471-486, 1999.
- [8] H. Choo, S. Yoo, 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.
- [9] 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.
- [10] 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.
- [11] I. Ababneh, S. Bani Mohammad, Noncontiguous Processor Allocation for Three-Dimensional Mesh Multicomputers, *AMSE Advances in Modelling & Analysis*, vol. 8, no. 2, pp. 51-63, 2003.
- [12] Intel Corporation, A Touchstone DELTA system description, 1991.
- [13] K. Windisch, J. V. Miller, and V. Lo, ProcSimity: an experimental tool for processor allocation and scheduling in highly parallel systems, *Proceedings of the Fifth Symposium on the Frontiers of Massively Parallel Computation (Frontiers'95)*, Washington, DC, USA, IEEE Computer Society Press, pp. 414-421, 1995.
- [14] 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.
- [15] K.-H. Seo, 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.
- [16] 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.
- [17] ProcSimity V4.3 User's Manual, University of Oregon, 1997.
- [18] R.E. Kessler, J.L Swarszmeier, Cray T3D: a new dimension for Cray research, *Proc. CompCon*, pp. 176-182, 1993.
- [19] 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.
- [20] 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.
- [21] W. Athas, C. Seitz, Multicomputers: message-passing concurrent computers, *IEEE Computer*, vol. 21, no. 8, pp. 9-24, 1988
- [22] W. Mao, J. Chen, 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.
- [23] W. Qiao, L. Ni, Efficient processor allocation for 3D tori, *Technical Report*, Michigan State University, East Lansing, MI, 48824-1027, 1994.
- [24] 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.

- [25] Y.-J. Tsai, 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.