# Non-contiguous Processor Allocation Strategy for 2D Mesh Connected Multicomputers based on Sub-meshes Available for Allocation

S. Bani-Mohammad
*Glasgow University, Computing Science, Glasgow G12 8RZ, UK.*
*saad@dcs.gla.ac.uk*

M. Ould-Khaoua
*Glasgow University, Computing Science, Glasgow G12 8RZ, UK.*
*mohamed@dcs.gla.ac.uk*

I. Ababneh
*Al al-Bayt University, Computing Science, Mafraq 25113, Jordan.*
*ismail@aabu.edu.jo*

Lewis M. Mackenzie
*Glasgow University, Computing Science, Glasgow G12 8RZ, UK.*
*lewis@dcs.gla.ac.uk*

## Abstract

*Contiguous allocation of parallel jobs usually suffers from the degrading effects of fragmentation as it requires that the allocated processors be contiguous and has the same topology as the network topology connecting these processors. 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 is available. Lifting the contiguity condition in non-contiguous allocation is expected to reduce processor fragmentation and increase processor utilization. However, the communication overhead is increased because the distances traversed by messages can be longer. The extra communication overhead depends on how the allocation request is partitioned and allocated to free sub-meshes. In this paper, a new non-contiguous processor allocation strategy, referred to as Greedy-Available-Busy-List, is suggested for the 2D mesh network, and is compared using simulation against the well-known non-contiguous and contiguous allocation strategies. To show the performance improved by proposed strategy, we conducted simulation runs under the assumption of wormhole routing and all-to-all communication pattern. The results show that the proposed strategy can reduce the communication overhead and improve performance substantially in terms of turnaround times of jobs and finish times.*

## Keywords

Multicomputers, Contiguous Allocation, Non-contiguous Allocation, Fragmentation, Turnaround Time, Finish Time, System Utilization, Performance Comparison, Simulation.

## 1. Introduction

In a multicomputer, processor allocation is responsible for selecting the set of processors on which parallel jobs are executed. Most strategies employed in a multicomputer are based on contiguous allocation [2, 5, 10, 13, 16]. These strategies often result in high external processor fragmentation, as has been shown in [16]. External fragmentation occurs when there are free processors sufficient in number to satisfy the number requested by a parallel job, but they are not allocated to it because the free processors are not contiguous or they do not have the same topology as the network topology connecting these processors.

Several studies have attempted to reduce such fragmentation [4, 6, 7, 12, 15]. One solution suggested is non-contiguous allocation [4, 7, 12, 15]. 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 is available. Although non-contiguous allocation increases message contention in the network, lifting the contiguity condition is expected to reduce processor fragmentation and to increase processor utilization [15].

Most of the existing studies [2, 5, 10, 13, 16] on allocation have been conducted in the context of the contiguous allocation. There has been comparatively very little work on non-contiguous allocation. Although contiguous allocation eliminates the communication overhead between processors by allocating them contiguously, non-contiguous allocation can alleviate the communication overhead as well as eliminate both internal and external fragmentation that result from contiguous allocation. Internal fragmentation occurs when more processors are allocated to a job than it requires. Existing research studies [2, 4, 5, 6, 10, 12, 13, 15, 16] on both contiguous and non-contiguous allocation have been

Proceedings of the 12th International Conference on Parallel and Distributed Systems (ICPADS'06)
0-7695-2612-8/06 $20.00 © 2006 IEEE

carried out in the context of the 2D mesh. The 2D mesh has been used as the underlying network in a number of practical and experimental parallel machines, such as iWARP [3] and Delta Touchstone [9]. In this study, we investigate the performance merits of non-contiguous allocation for reducing processor fragmentation on the 2D mesh and compare its performance to that of the previous allocation strategies. To do so, non-contiguous allocation strategy, notably Greedy-Available-Busy-List (GABL), is suggested. The GABL strategy combines the desirable features of both contiguous and non-contiguous allocation strategies. The performance of the GABL strategy will be compared against the existing well known non-contiguous allocation strategies Paging(0) and MBS [15], these two strategies have been selected because they have been shown to perform well when compared to existing strategies. Also, the performance of GABL is compared against the existing well known contiguous First Fit strategy [16]. First Fit has been used to represent the contiguous class of strategies as it has been found to perform well [16].

In this study, All-to-All communication pattern is considered because it causes much message collision and it was known as a weak point for the non-contiguous allocation strategies [12].

The rest of the paper is organized as follows. Section 2 contains a brief summary of allocation strategies previously proposed for 2D mesh. Section 3 contains the proposed non-contiguous allocation strategy. Section 4 compares the performance of the contiguous and non-contiguous allocation strategies. Section 5 concludes this study.

## 2. Related Work

This section provides a brief overview of some existing contiguous and non-contiguous allocation strategies.

### 2.1 Contiguous Allocation Strategies

Contiguous allocation has been investigated extensively for 2D mesh multicomputers [2, 5, 10, 13, 16]. Most of the previous studies have focused on reducing the degrading effects of high external fragmentation caused by the contiguous allocation strategies. Below we describe some of these strategies.

*Two Dimensional Buddy System (2DBS)*: The 2DBS allocation [10] 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 $a \times b$ such that $a \leq b$, the 2DBS allocates a sub-mesh of size $s \times s$, where $s = 2^{\lceil \log(\max(a,b)) \rceil}$. For example, if a job requests two processors it is allocated a square partition of processors with a side length of two, resulting in two idle processors and an internal fragmentation of 50%. This strategy suffers from high fragmentation [4, 15]. Also, it cannot be used for non-square meshes and does not have complete sub-mesh recognition ability [4, 15].

*First Fit (FF) and Best Fit (BF)*: The problem of missing an existing possible allocation explained above is solved using First Fit and Best Fit allocation strategies [16]. The free sub-meshes are scanned and First Fit allocates the first sub-mesh that is large enough to hold the job, whereas Best Fit allocates the smallest suitable sub-mesh by choosing the corner that has the largest number of busy neighbors. Bit arrays are used for scanning of available processors.

*Stack Based Allocation (SBA)*: This strategy finds a free sub-mesh quickly by reducing the search space drastically through the use of a simple coordinate calculation and spatial subtraction [2]. It could be implemented efficiently using a stack, as demonstrated in [2]. Simulation results have shown that the SBA strategy allocates processors faster than the previous allocation strategies and still delivers competitive performance [2].

The above allocation strategies consider only contiguous regions for the execution of a job. As a consequence, the distance of the communication paths is expected to be minimized in contiguous allocation. Only messages generated by the same process are expected within a sub-mesh and therefore cause 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 successfully allocating a job. It is possible to fail in the contiguous allocation strategies to allocate a job while a sufficient number of processors are available [4], i.e., fragmentation occurs in these strategies.

### 2.2 Non-Contiguous Allocation Strategies

Advances in switching techniques such as the proposal of wormhole routing [11], has made communication latency less sensitive to the distance among the communicating nodes [4]. This has made allocating a job to non-contiguous processors plausible. Allocation of jobs to non-contiguous nodes allows jobs to be executed without waiting if the number of available processors is sufficient [4]. Below we describe some of the non-contiguous allocation strategies that have been suggested in the literature.

*Paging*: In the Paging strategy [15], the entire 2D mesh is divided into pages that are sub-meshes with equal sides' length 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). A paging strategy is denoted as Paging( $size\_index$ ). For example, Paging(2) means that the pages are 4×4 sub-mesh. The number of pages a job requests is computed using $\lceil (a \times b)/Psize \rceil$, where $Psize$ is the size of the pages, and $a$ and $b$ are the side lengths of the requested sub-mesh. In this paper, we only consider the row-major indexing scheme because using the remaining indexing schemes has only a very slight impact on the performance of Paging [15].

*Multiple Buddy System (MBS)*: The MBS is an extension of the 2D buddy strategy. The mesh is divided into non-overlapped square sub-meshes with side lengths equal to the powers of two upon initialization. The number of processors, $p$, requested by an incoming job is factorized into a base of four representation of $\sum_{i=0}^{\lfloor \log_4 p \rfloor} d_i \times \left(2^i \times 2^i\right)$, where $0 \le d_i \le 3$. The request is then allocated to the mesh according to the factorized number in which $d_i$ number of $2^i \times 2^i$ blocks is required. If a required block is unavailable, MBS recursively searches for a bigger block and repeatedly breaks it down into buddies until it produces blocks of the desired size. If that fails, the requested block is then broken into four requests for smaller blocks and the searching process repeats [15].

*Adaptive Non-Contiguous Allocation (ANCA)*: In [4], ANCA always attempts to allocate a job contiguously. When contiguous allocation fails, it breaks a job request into two equal-sized sub-frames. These sub-frames are then allocated to available locations if possible; otherwise, each of these sub-frames is broken into two equal-sized sub-frames, and then ANCA try to allocate these sub-frames to available locations and thus take advantage of non-contiguous allocation, and so on.

In Paging strategy, there is some degree of contiguity because of the indexing schemes used. Contiguity can also be increased by increasing *size_index*. However there is an internal processor fragmentation for *size_index* ≥ 1, and it increases with *size_index* [15]. An issue with MBS strategy is that it may not allocate a contiguous sub-mesh although one exists. 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. ANCA strategy can disperse the allocated sub-meshes more than it is necessary. It requires that allocation to all sub-frames occurs in the same decomposition and allocation iteration, skipping over the possibility of allocating larger sub-frames for a large part of the request in a previous iteration. Moreover, stopping when a side length reaches one can cause external fragmentation.

Our proposed strategy maintains a degree of contiguity between processors larger than that of the previous non-contiguous allocation strategies if the number of requested processors is available in the mesh so that the communication between processors in GABL is lower than that of previous non-contiguous allocation strategies. Moreover, it eliminates both internal and external fragmentation.

## 3. Proposed Allocation Strategy

The target system is a $W \times L$ 2D mesh, where $W$ is the width of the square mesh and $L$ its length. Every processor is denoted by a coordinate $(x, y)$, where $1 \le x \le W$ and $1 \le y \le L$ [7]. Each processor is connected by bidirectional communication links to its neighbor processors, as depicted in Fig. 1. This figure shows an example of a 4×4 2D mesh, where allocated processors are denoted by shaded circles while the free processors are denoted by white circles.

If a job requests the allocation of 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 can be allocated to the job if allocation is non-contiguous. In what follows we assume that a parallel job requests, when it arrives, the allocation of a 2D sub-mesh $S(a,b)$ of width $a \le W$ and length $b \le L$. The following definitions have been adopted from [7].

Definition 1: A sub-mesh $S(w,l)$ of width $w$ and length $l$, where $1 \le w \le W$ and $1 \le l \le L$ is specified by the coordinates $(x, y)$ and $(x', y')$, where $(x, y)$ is the lower left corner of $S$, $(x', y')$ is the 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. For example (0, 0, 2, 1) represents the 3 × 2 sub-mesh $S$ in Fig. 1. The base node of the sub-mesh is (0, 0), and its end node is (2, 1).

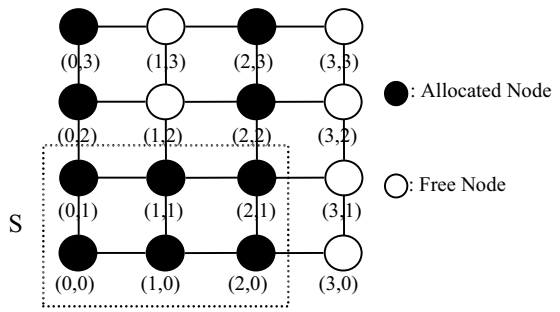


Fig. 1: An example of a 4×4 2D mesh

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

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

In this study, it is assumed that parallel jobs are by the scheduler on a First-Come-First-Served (FCFS) basis. The FCFS scheduling strategy is chosen because it is fair and it is widely used in other studies [4, 5, 7, 15, 16]. In the next sub-section, we describe the non-contiguous allocation strategy.

### 3.1 Greedy-Available-Busy-List Strategy (GABL)

In GABL strategy, we use an efficient approach [5], Right of Busy Sub-meshes (RBS) contiguous allocation strategy, to facilitate the detection of such available sub-meshes with low allocation overhead. The basic idea in this approach [5] is to maintain a list of allocated sub-meshes sorted in non-increasing order of the second coordinated of their upper right corners. The list is scanned to determine all the nodes that cannot be used as base nodes for the requested sub-mesh. These nodes are then subtracted from the right border line of the allocated sub-meshes to find the nodes that could be used as base nodes for the required sub-mesh size.

In GABL strategy, when a parallel job is selected for allocation a sub-mesh suitable for the entire job is searched for. If such a sub-mesh is found it is allocated to the job and allocation is done. Otherwise, the largest free sub-mesh that can fit inside $S(a,b)$ is allocated. Then, the largest free sub-mesh whose side lengths do not exceed the corresponding side lengths of the previous allocated sub-mesh is searched for and allocated if this does not result in allocating more processors than $a \times b$. This last step is repeated until $a \times b$ processors are allocated. Allocated sub-meshes are kept in a busy list. Each element in this list includes *id* of the job the sub-mesh is allocated to. When a job departs the sub-meshes it is allocated are removed from the busy list and the number of free processors is updated.

This allocation process is implemented by the algorithm bellow, illustrated in Fig. 2. Note that allocation always succeeds if the number of free processors $\geq a \times b$, and scanning for free sub-meshes uses the RBS contiguous allocation strategy [5].

```
Procedure Greedy-Available-Busy-List (a, b):
Begin
{
  Total_Allocated = 0
  Job_Size = a × b
    Step1. If (number of free processors < Job_Size)
                return failure
    Step2. If (there is a free S(x, y) suitable for S(a, b))
                allocate it using RBS contiguous allocation
                algorithm and return success.
    Step3. α = a and β = b
    Step4. Subtract 1 from max (α, β) if max > 1
    Step5. If (Total _allocated + α × β > Job_Size)
                go to step 4
    Step6. If there is a free S (x, y) suitable for S(α, β)
      {
          allocate it using RBS algorithm.
          Total_allocated = Total_allocated + α × β.
    }
    Step7. If (Total_allocated = Job_Size)
                return success.
          else
                go to Step 5.
}
End.
```

Fig. 2: Outline of the GABL algorithm

## 4. Performance Evaluation

In this section, the time and space complexities of the proposed allocation strategy are presented first. Then, the results of detailed simulations that have been carried out to evaluate the performance of the proposed algorithm and compare it to Paging(0) and MBS are presented; These two strategies have been selected because they have been shown to perform well when compared to existing strategies in [15].

### 4.1 Allocation and Deallocation Complexities

When a sub-mesh is allocated, the busy list can be accessed using the algorithm proposed in [5] in $O(m^2)$ time, where $m$ is the number of allocated sub-meshes in the system (i.e., for all jobs). In the worst case, the new proposed algorithm allocates $bm^2$ to a job, where $b$ is the number of allocated sub-meshes for an incoming job. Therefore, the time complexity of allocation in GABL algorithm is in $O(bm^2)$. When a job departs, the busy list

is accessed for each released sub-mesh. The number of released sub-meshes is in $O(m)$, where $m$ is the number of allocated sub-meshes. Consequently the time complexity of the deallocation algorithm is in $O(m)$.

The proposed algorithm maintains a busy list. As a result, its space complexity is in $O(m)$. Such space incurred by this strategy is small compared to the improvement in performance (i.e., low turnaround times and low finish times compared to the previous non-contiguous allocation strategies as can be seen in the next subsection).

### 4.2 Simulation Results

In addition to simulation results for GABL, Paging(0), and MBS, we also show the results for the contiguous First Fit allocation strategy in this subsection. This strategy was chosen because it is a good representative of traditional contiguous allocation strategies (First Fit, Best Fit, and Worst Fit) [5, 15, 16]. This strategy allocates an incoming job to the first available sub-mesh that is found [16].

We have implemented the proposed allocation and deallocation algorithms, including the busy list routines, in C language, and integrated the software into the ProcSimity simulation tool for processor allocation and job scheduling in highly parallel systems [14].

The target mesh modelled in the simulation experiments discussed bellow is square with side lengths $L$. Jobs are assumed to have exponential inter-arrival times. They are served on First-Come-First-Served (FCFS) basis. The execution times of jobs are assumed to be exponentially distributed with a mean of one time unit. We have used uniform distribution to generate the lengths and widths of requests. The uniform distribution is used over $[1, L]$, where the width and length of a request are generated independently. This distribution has often been used in the literature [5, 7, 15, 16]. Each simulation run consists of one thousands 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%.

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. Message sizes are represented by $P_{len}$. Processors allocated to a job communicate with each other using all-to-all communication pattern [11, 12, 15]. In all-to-all communication pattern, each processor allocated to a job sends a packet to all other processors allocated to the same job. The number of messages that are actually sent is exponentially distributed with a mean *num-mes*.

Unless specified otherwise, the performance figures shown bellow are for a 16 ×16 mesh, $t_s$ = 3 time units, $P_{len}$ = 8 flits and *num_mes* = 5 messages. The main performance parameters used are the average turnaround time of jobs, the finish time for all jobs and the system utilization. The turnaround time of a job is the time that the job spends in the mesh from arrival to departure. The finish time is the time required for completion of all the jobs. The utilization is the percentage of processors that are utilized over time. 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.

In Fig. 3, the finish times of all jobs are plotted against the system load for the all-to-all communication pattern. It can be seen in the figure that all strategies have the same finish times for the low loads because it is highly likely that a suitable contiguous sub-mesh is available for allocation to a job when it arrives to the mesh. However, there is a difference in finish times between the strategies for medium and high loads. The GABL could show better performance than all other strategies when the load is increased. Moreover, GABL is substantially superior to all other strategies for high loads.

In Fig. 4, the average turnaround times of jobs are plotted against the system load for the all-to-all communication pattern. This figure reveals that GABL performs much better than all other strategies. Also, GABL is substantially superior to the First Fit contiguous allocation strategy. Experiments that use larger messages sizes (16, 32, and 64 flits) have been also conducted. Their results lead to the same conclusion on the relative performance of the strategies. Moreover, the results indicate that the relative advantage of GABL over the remaining strategies increases with the message length.

In Fig. 5, the system utilization is plotted against the system load for the all-to-all communication pattern. It can be seen in the figure that all non-contiguous allocation strategies are better than First Fit for all the system loads. This is due to the fact that contiguous strategies, represented by First Fit, produce high external fragmentation under such loads. It can also be noticed that the system utilization for GABL strategy is slightly better than all other non-contiguous allocation strategies.

Overall, using the busy list approach enables the GABL to be the most flexible allocation strategy. Moreover, when the contention is increased using all-to-all communication pattern the GABL is superior to the other strategies as is depicted in Figs. 3 and 4.

## 5. Conclusions

This study has investigated the performance merits of non-contiguous allocation in the 2D mesh. To this end,

we have proposed a new non-contiguous allocation strategy, notably Greedy-Available-Busy-List, the GABL strategy decompose the allocation request based on the sub-meshes available for allocation. The major goal of partitioning process is maintaining a high degree of contiguity among processors allocated to a job. Therefore, the number of sub-meshes allocated to a job is reduced and hence the distance traversed by the messages is reduced. The performance of GABL strategy has been compared against the existing non-contiguous as well as contiguous allocation strategies. Simulation results have revealed that non-contiguous allocation greatly improves performance despite the additional message contention inside the network that results from the interference among the messages of different jobs and it produces superior utilization than its contiguous allocation counterpart. Results have also shown that the GABL strategy is substantially superior to the previous promising non-contiguous allocation strategies MBS and Paging(0). 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 job sizes typically grow with the size of the mesh.

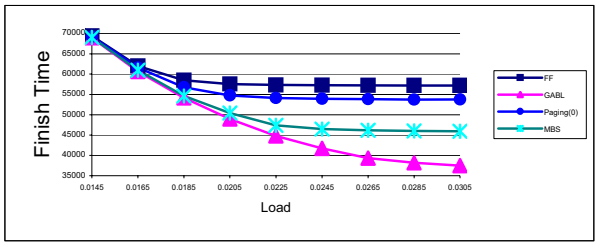


Fig. 3: Finish time vs. system load for the all-to-all communication in a 16 × 16 mesh.

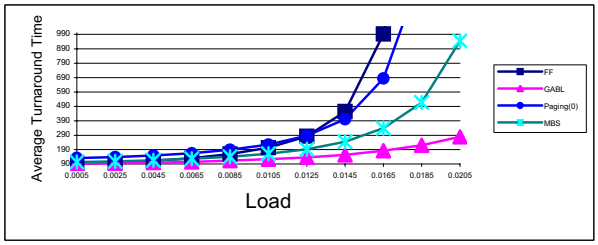


Fig. 4: Average turnaround time vs. system load for the all-to-all communication in a 16 × 16 mesh.

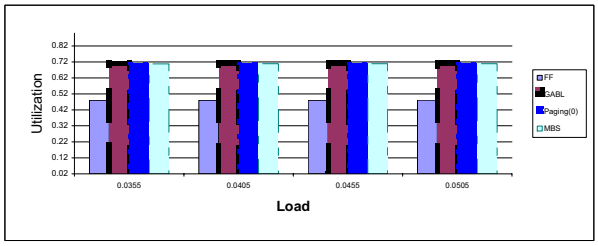


Fig. 5: System utilization vs. system load for the all-to-all communication in a 16 × 16 mesh.

## 6. References


[1] ″Blue Gene Project″, http://www.research.ibm.com/bluegene/index.html, 2005.

[2] 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.

[3] C. Peterson, J. Sutton, P. Wiley, iWARP: a 100-MPOS VLIW microprocessor for multicomputers, *IEEE Micro*, vol. 11, no. 13, 1991.

[4] C.-Y. Chang, P. Mohapatra, Performance improvement of allocation schemes for mesh-connected computers, *Journal of Parallel and Distributed Computing*, vol. 52, no. PC981459, pp. 40-68, 1998.

[5] 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.

[6] I. Ababneh, 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.

[7] 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.

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

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

[10] K. Li, 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.

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

[12] Kuniyasu Suzaki, Hitoshi Tanuma, Satoshi Hirano, Yuuji Ichisugi, Chris Connelly, Michiharu Tsukamoto, Multi-tasking Method on Parallel Computers which Combines a Contiguous and Non-contiguous Processor Partitioning Algorithm. *PARA 1996*: 641-650.

[13] P.-J. Chuang, 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.

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

[15] V. Lo, K. Windisch, W. Liu, 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.

[16] 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.