An Efficient Processor Allocation Strategy that Maintains a High Degree of Contiguity among Processors in 2D Mesh Connected Multicomputers

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

Two strategies are used for the allocation of jobs to processors connected by mesh topologies: contiguous allocation and non-contiguous allocation. In non-contiguous allocation, a job request can be split into smaller parts that are allocated to non-adjacent free sub-meshes rather than always waiting until a single sub-mesh of the requested size and shape is available. Lifting the contiguity condition is expected to reduce processor fragmentation and increase system utilization. However, the distances traversed by messages can be long, and as a result the communication overhead, especially contention, is increased. The extra communication overhead depends on how the allocation request is partitioned and assigned to free sub-meshes. This paper presents a new Non-contiguous allocation algorithm, referred to as Greedy-Available-Busy-List (GABL for short), which can decrease the communication overhead among processors allocated to a given job. The simulation results show that the new strategy can reduce the communication overhead and substantially improve performance in terms of parameters such as job turnaround time and system utilization. Moreover, the results reveal that the Shortest-Service-Demand-First (SSD) scheduling strategy is much better than the First-Come-First-Served (FCFS) scheduling strategy.

1. Introduction

In a multicomputer, 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. Most allocation strategies employed in a multicomputer 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 [1, 3, 4, 5, 7, 13, 22]. *Contiguous* strategies often result in high external processor fragmentation, as has been shown in [22]. External processor 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 multicomputer.

Several studies have attempted to reduce external processor fragmentation [2, 8, 10, 15, 17, 19]. One suggested solution is to adopt *non-contiguous* allocation [2, 8, 15, 19]. 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 utilization [2, 15, 19]. It is the introduction of wormhole routing [18] that has lead researchers to consider non-contiguous allocation on multicomputer networks with a long communication distances, such as the 2D mesh [2, 15, 19]. 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 depends less on the distance the message travels from source to destination [2, 18]. Nonetheless, most existing research studies have been conducted in the context of contiguous allocation [1, 3, 4, 5, 7, 13, 17, 22]. 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 external processor fragmentation that contiguous allocation suffers from.

Most existing research on contiguous and non-contiguous allocation has been carried out in the context of the 2D mesh [1, 2, 3, 4, 5, 7, 8, 10, 13, 15, 17, 19, 22]. The mesh network has been used as the underlying network in a number of practical and experimental parallel machines, such as IBM BlueGene/L [11, 21] and Delta Touchstone [6]. The method used for decomposing allocation requests in existing non-contiguous allocation schemes are not based on free contiguous sub-meshes. For example, allocation requests are subdivided into two equal parts in [2]. The subparts are successively subdivided in a similar fashion if allocation fails for any of them. In the study of [19], a promising strategy (MBS) expresses the allocation request as a base-4 number, and bases allocation on this expression. In this study, we propose a new non-contiguous allocation strategy, referred to here as Greedy-Available-Busy-List (GABL), for the 2D mesh. GABL strategy combines the desirable features of both contiguous and non-contiguous strategies and 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. The performance of GABL is compared against the performance of the non-contiguous allocation strategies Paging(0) and MBS [19]. These two strategies have been selected because they have been shown to perform well in [19]. Furthermore, GABL is also compared against the contiguous First Fit strategy [22] as this has been used in several previous related studies [2, 3, 19].

In addition to suggested allocation strategy, we use two job scheduling strategies, notably First-Come-First-Served (FCFS) and Shortest-Service-Demand-First (SSD) to compare the performance of allocation strategies. In FCFS, the allocation request that arrived first is considered for allocation first. Allocation attempts stop when they fail for the current FIFO queue head, while in SSD, the job with the shortest service demand is scheduled first [12].

The rest of the paper is organized as follows. Section 2 contains a brief summary of previous allocation strategies. Section 3 describes our proposed non-contiguous allocation strategy. Section 4 compares the performance of the contiguous and non-contiguous allocation strategies. Finally, Section 5 concludes this study.

2. Related Work

To conserve space, this section provides a brief overview of some existing non-contiguous allocation strategies, the existing contiguous allocation strategies are not included in this paper, but appear in [16].

2.1 Non-Contiguous Allocation Strategies

Advances in routing techniques such as wormhole routing [18], has made communication latency less sensitive to the distance between communicating nodes [2]. This has made allocating a job to non-contiguous processors plausible in networks characterised by a long-diameter, such the 2D mesh. Non-contiguous allocation allows jobs to be executed without waiting if the number of available processors is sufficient [2, 15, 19]. Below, we describe some non-contiguous strategies that have been suggested in the literature.

Paging: In the Paging strategy [19], 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. The pages are indexed according to several indexing schemes (row-major, shuffled row-major, snake-like, and shuffled snake-like indexing). The number of pages a job requests is computed by: $\lceil (a \times b) / Psize \rceil$, 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 slight impact on the performance of Paging, as has been demonstrated in [19].

Multiple Buddy System (MBS): In MBS, the mesh network is divided into non-overlapped square sub-meshes with side lengths that are powers of 2. The number of processors, p, requested by an incoming job is factorized into a base-4 representation of the form: $\sum_{i=0}^{\lfloor \log_4 p \rfloor} d_i \times \left(2^i \times 2^i\right)$, where $0 \le d_i \le 3$. The request is then considered for allocation according to the factorized number, where d_i blocks of size $2^i \times 2^i$ are required. If a required block is unavailable, MBS recursively searches for a larger block and repeatedly breaks it down into four buddies until it produces blocks of the desired size. If that fails, the requested block is broken into four requests for smaller blocks and the searching process is repeated [19].

Adaptive Non-Contiguous Allocation (ANCA): ANCA first attempts to allocate a job contiguously. When contiguous allocation fails, it breaks the 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, then ANCA tries to assign these sub-frames to available locations and thus take advantage of non-contiguous allocation, and so on [2].

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 $size\_index$ [19]. An issue with MBS is that it may fail to 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. As for ANCA, it can disperse the allocated sub-meshes more than it is necessary. It requires that allocation to all sub-frames occur in the same decomposition 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 decomposition and search processes when a side length reaches 1, which can cause external fragmentation. The main goal of our proposed strategy is to achieve a larger degree of contiguity than the previous non-contiguous allocation strategies. This is so that the communication overhead is lower and the overall system performance is superior.

3. The Proposed Allocation Strategy

In the following, we present the system model assumed in this paper. The target system is a $W \times L$ 2D mesh, 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 $0 \leq x < W$ and $0 \leq y < L$ [15]. Each processor is connected by bidirectional communication links to its neighbour processors. The following definitions have been adopted from [15].

Definition 1: A sub-mesh $S(w,l)$ of width w and length l, where $0 \leq w < W$ and $0 \leq l < L$ is specified by the coordinates (x, y) and (x', y'), where (x, y) is the lower left corner of S 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.

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 not allocated.

An allocation request can be accommodated contiguously if and only if a suitable sub-mesh is available [3]. In this study, it is assumed that parallel jobs are selected for allocation and execution using FCFS and SSD scheduling strategies. The FCFS scheduling strategy is chosen because it is fair and it is widely used in other similar studies [2, 3, 4, 10, 15, 19, 20, 22], while the SSD scheduling strategy is used to avoid potential performance loss due to blocking [12]. In the next sub-section, we describe the non-contiguous allocation strategy.

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

The GABL strategy combines the desirable features of both contiguous and non-contiguous allocation, and partitions requests based on the sub-meshes available for allocation. In implementing GABL, we exploit an efficient approach, the Right of Busy Sub-meshes (RBS) approach proposed in [3], for the detection of such available sub-meshes. The basic idea of RBS is to maintain a list of the allocated sub-meshes sorted in the non-increasing order of the second coordinate of their upper right corners. The list is used to determine all forbidden regions consisting of the nodes that cannot serve as base nodes for the requested sub-mesh. The forbidden 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.

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 the 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 under the constraint that the number of processors allocated does not exceed $a \times b$. 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 the 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.

Allocation in GABL strategy is implemented by the algorithm outlined in Fig. 1, while the deallocation algorithm is outlined in Fig. 2. Note that allocation always succeeds if the number of free processors $\geq a \times b$. Moreover, it can be noticed that the methodology used for maintaining contiguity is greedy. GABL strategy attempts to allocate large sub-meshes first.

Procedure Greedy-Available-Busy-List (a, b):
{Total_Allocated = 0; Job_Size = $a \times b$
*Step*1. *If (number of free processors < Job_Size)*
return failure
*Step*2. *If (there is a free S(x, y) suitable for S(a, b))*
allocate it using RBS contiguous allocation algorithm and return success.
*Step*3. *α = a and β = b*
*Step*4. *Subtract 1 from max (α, β) if max > 1*

*Step*5. *If (Total _allocated + α × β > Job_Size)*
go to step 4
*Step*6. *If there is a free S (x, y) suitable for S(α, β){*
allocate it using RBS algorithm.
Total_allocated = Total_allocated + α × β.
}
*Step*7. *If (Total_allocated = Job_Size)*
return success.
else
go to Step 5.
} ***end procedure***

Figure 1: Outline of GABL allocation algorithm

Procedure GABL_Deallocate ():
{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***

Figure 2: Outline of GABL deallocation algorithm

4. Performance Evaluation

In this section, 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. To conserve space, the complexity analysis of the proposed algorithm is not included in this paper, but appears in [16].

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 job scheduling in highly parallel systems [9, 14].

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 scheduled using FCFS and SSD scheduling strategies. The execution times of jobs are assumed to be exponential distributed with a mean of one time unit. 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 distribution is uniform-decreasing distribution. It is determined by four probabilities $p1$, $p2$, $p3$, and $p4$, respectively. The side lengths within a range are equally likely to occur. For the simulation results shown below, $p1 = 0.4$, $p2 = 0.2$, $p3 = 0.2$, $p4 = 0.2$, $l1 = L/8$, $l2 = L/4$, $l3 = L/2$, and $l4 = L$. These distributions have often been used in the literature [1, 3, 4, 15, 16, 19, 22]. 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%.

The interconnection network uses wormhole, *XY* 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 one of the two common communication patterns [8, 18, 19]. 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 [8].

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 the 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 [19], 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 $num\_mes = 5$ messages. The main performance parameters used are the average turnaround time of jobs and the mean system utilization. The turnaround time of a job is the time that the job spends in the mesh from arrival to departure. The mean system utilization is the percentage of processors that are utilized over time. The independent variable in the simulation is the system load. It is defined as the inverse of the mean inter-arrival time of jobs.

The notation <allocation strategy>(<scheduling strategy>) is used to represent the strategies in the performance figures. For example, GABL(SSD) refers to the Greedy-Available-Busy-List allocation strategy under the scheduling strategy Shortest-Service-Demand-First.

In Figs. 3 and 4, the average turnaround time of jobs is plotted against the system load for the one-to-all communication pattern and the two scheduling strategies FCFS and SSD. The results reveal that GABL strategy performs better than all other strategies for both job size distributions and scheduling strategies considered in this paper. Furthermore, GABL strategy is substantially superior to the FF strategy for both job size distributions and scheduling strategies. In Fig. 3, for example, the difference in performance in favour for GABL(FCFS)

strategy could be as large as 64% compared to FF(FCFS) strategy, 36% to Paging(0)(FCFS) strategy, and 31% to MBS(FCFS) strategy under the job arrival rate 0.0205 jobs/time unit. 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 allocation strategies. Moreover, the results indicate that the relative performance merits of GABL strategy over the remaining strategies become more noticeable as the message length increases.

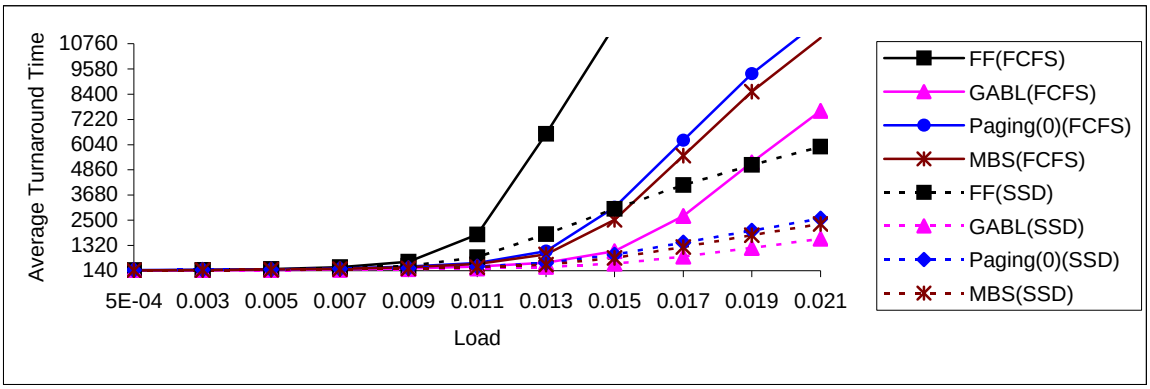

Figure 3: Average turnaround time vs. system load for the one-to-all communication pattern and uniform side lengths distribution.

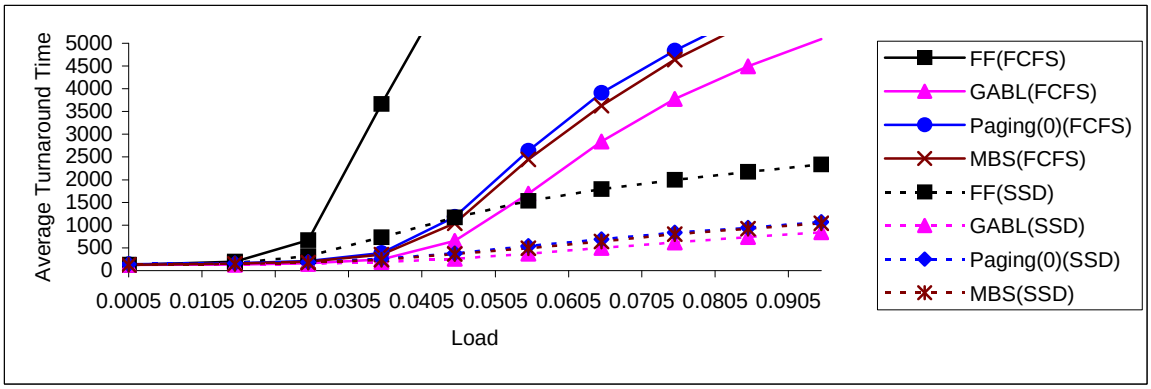

Figure 4: Average turnaround time vs. system load for the one-to-all communication pattern and uniform-decreasing side lengths distribution.

In Figs. 5 and 6, the average turnaround time of jobs is plotted against the system load for the all-to-all communication pattern and the two scheduling strategies FCFS and SSD. Again, GABL strategy performs much better than all other strategies for both job size distributions and scheduling strategies. Moreover, GABL strategy is substantially superior to FF strategy for both job size distributions and scheduling strategies. Experiments that use larger messages sizes (16, 32, and 64 flits) have lead to the same conclusion as to the relative performance of the strategies. Fig. 6, for example, depicts that when the job arrival rate is 0.1 jobs/time unit, the average turnaround time of GABL(FCFS) are 0.17, 0.28, and 0.30 of the average turnaround time of FF(FCFS), Paging(0)(FCFS), and MBS(FCFS) respectively.

Fig. 7 depicts the mean system utilization of the allocation strategies GABL(FCFS), MBS(FCFS), Paging(0)(FCFS), and FF(FCFS) for the two communication patterns considered and FCFS scheduling strategy under uniform-decreasing 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 utilization. For both job size distributions, uniform and uniform-decreasing, the non-contiguous allocation strategies achieve a mean system utilization of 72% to 79%, but FF can not exceed 49%. This is because contiguous allocation produces high external fragmentation, which means that allocation is less likely to succeed. As a consequent, the mean system utilization is lower. The

utilization of the three non-contiguous allocation strategies is approximately the same for both job size distributions. This is because the non-contiguous allocation strategies, considered in this paper, have the same ability to eliminate both internal and external processor fragmentation. They always succeed to allocate processors to a job when the number of free processors is greater than or equal the allocation request. Experiments that compute the utilization based on SSD strategy have been also conducted. Their results lead to the same conclusion as in FCFS strategy. To conserve space, the results of the mean system utilization under uniform side length distribution are not included in this paper, but appear in [16].

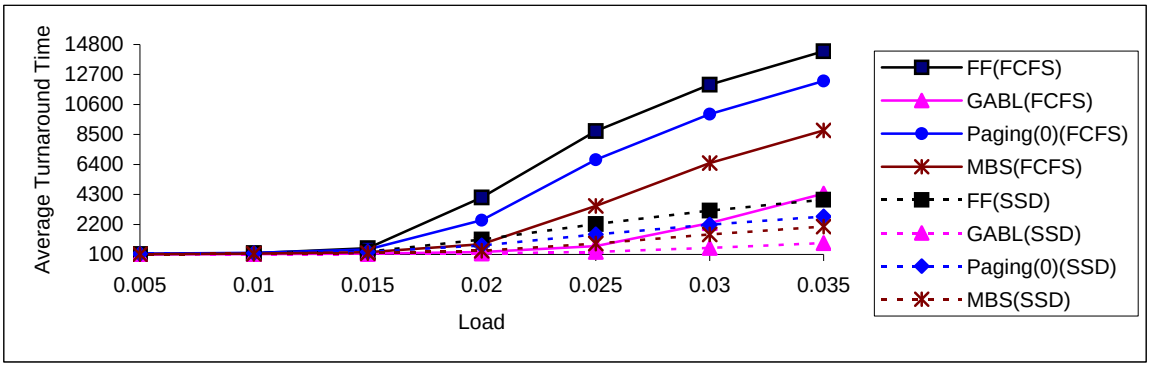

Figure 5: Average turnaround time vs. system load for the all-to-all communication pattern and uniform side lengths distribution.

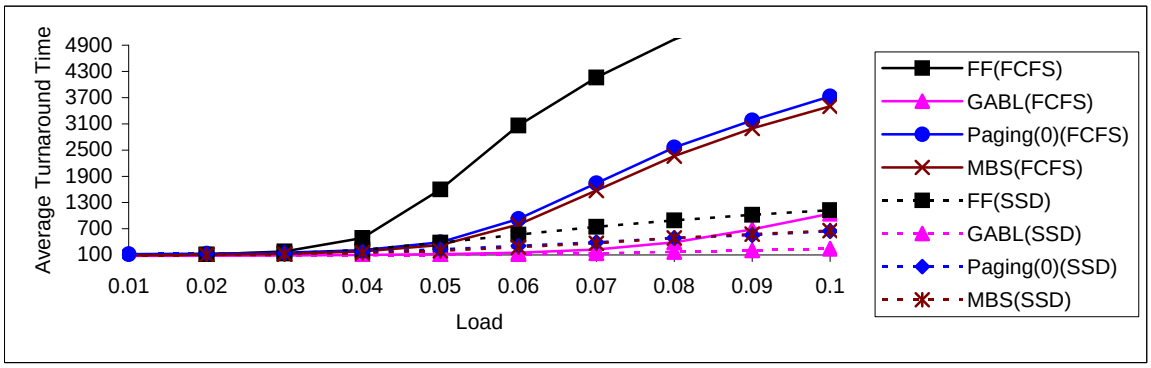

Figure 6: Average turnaround time vs. system load for the all-to-all communication pattern and uniform-decreasing side lengths distribution.

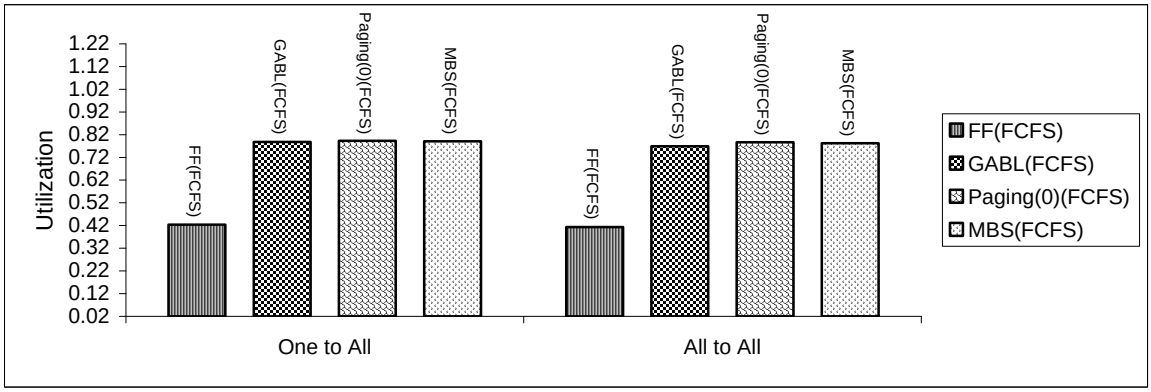

Figure 7: System utilization of the non-contiguous allocation strategies (GABL, MBS, Paging(0)) and contiguous allocation strategy FF, for the two communication patterns tested, and uniform-decreasing side lengths distribution.

In addition to the turnaround time and system utilization we have measured another performance parameter for the non-contiguous allocation strategies. This is the average blocks per job, which is defined as the average number of non-contiguous blocks allocated to a job. The higher the average blocks the more likely it is that the job's messages will go through nodes allocated to other jobs, potentially causing more contention in the interconnection network [19]. Fig. 8 shows that the average blocks per job for the non-contiguous allocation strategies that gave the best performance (GABL and MBS strategies), for all-to-all communication pattern and uniform side length distribution under both scheduling strategies FCFS and SSD. It can be seen that GABL strategy has lower average blocks per job than MBS strategy over all loads under the two scheduling strategies FCFS and SSD. For example, the average blocks per job of GABL(FCFS) allocation strategy are 0.54, 0.59, and 0.60 of the average blocks per job of MBS(FCFS) allocation strategy when the job arrival rates are 0.025, 0.03, and 0.035 jobs/time unit, respectively. This conclusion is compatible with the values of the mean turnaround time shown above. To conserve space, the results of the average blocks per job for the one-to-all communication pattern under both job size distributions and also for all-to-all communication pattern and uniform-decreasing side lengths distribution are not included in this paper, but appear in [16].

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

5. Conclusions and Future Directions

This paper has investigated the performance merits of non-contiguous allocation in the 2D mesh network. To this end, we have suggested a new non-contiguous allocation strategy, referred to as Greedy-Available-Busy-List, which differs from the earlier non-contiguous allocation strategies in the method used for decomposing allocation requests. The GABL strategy decomposes 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, hence decreases the distance traversed by messages, and which in turn decreases the communication overhead. GABL strategy 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 strategy was compared against that of existing non-contiguous and contiguous allocation strategies using both FCFS and SSD scheduling strategies. Simulation results have shown that GABL strategy can greatly improve performance despite the additional message contention inside the network that results from the interference among the messages of different jobs. GABL strategy also produces superior system utilization than its contiguous counterpart. The results have also revealed that GABL strategy is substantially superior over the previous well-known non-contiguous allocation strategies considered in this paper. Results have also shown that the effects of the SSD scheduling strategy on the performance of the allocation strategies is substantially better than that of the FCFS scheduling strategy in terms of mean turnaround time. Moreover, GABL strategy 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.

As a continuation of this research in the future, it would be interesting to assess the suggested 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.

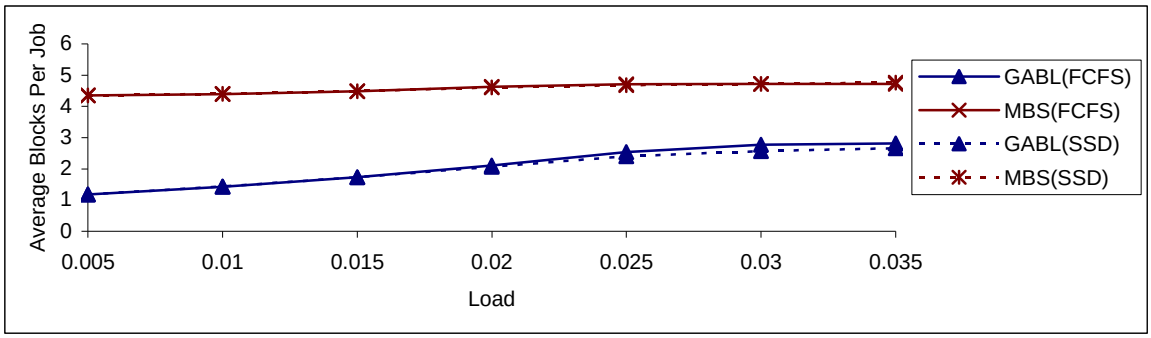

Figure 8: Average blocks per job vs. system load for the all-to-all communication pattern and uniform side lengths distribution.

6. References

[1] 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*, IEEE Computer Society, Washington, USA, January 2002, pp. 46-60.

[2] C.-Y. Chang, P. Mohapatra, "Performance improvement of allocation schemes for mesh-connected computers", *Journal of Parallel and Distributed Computing, vol. 52, no. 1*, Academic Press, Inc. Orlando, FL, USA, July 1998, pp. 40-68.

[3] 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*, IEEE Press, Piscataway, NJ, USA, May 1999, pp. 471-486.

[4] I. Ababneh, "An efficient free-list submesh allocation scheme for two-dimensional mesh-connected multicomputers", *Journal of Systems and Software, vol. 79, no. 8*, Elsevier Science Inc., New York, NY, USA, August 2006, pp. 1168-1179.

[5] I. Ismail, J. Davis, "Program-based static allocation policies for highly parallel computers", *Proc. IPCCC* 95, IEEE Computer Society Press, Scottsdale, AZ, USA, 28-31 Mar 1995, pp. 61-68.

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

[7] 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*, Elsevier Science, CA, USA, May 1991, pp. 79-83.

[8] 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 Third International Workshop on Applied Parallel Computing, Industrial Computation and Optimization*, Springer-Verlag, UK, 1996, pp. 641-650.

[9] 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)*, IEEE Computer Society Press, Washington, USA, 6-9 Feb 1995, pp. 414-421.

[10] 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, Washington, DC, USA, 7-9 December, 2005, pp. 318-323.

[11] 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.

[12] P. Krueger, T. Lai, 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*, IEEE Press, Piscataway, NJ, USA, May 1994, pp. 488-497.

[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*, IEEE Press, USA, February 1994, pp. 211-217.

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

[15] 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), vol. 2*, IEEE Computer Society Press, USA, 2006, pp. 41-48.

[16] 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.

[17] T. Srinivasan, J. Seshadri, A. Chandrasekhar, J. Jonathan, "A Minimal Fragmentation Algorithm for Task Allocation in Mesh-Connected Multicomputers", *Proceedings of IEEE International Conference on Advances in Intelligent Systems – Theory and Applications – AISTA 2004 in conjunction with IEEE Computer Society, ISBN 2-9599-7768-8*, IEEE Press, Luxembourg, Western Europe, 15-18 Nov 2004.

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

[19] 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*, IEEE Press, Piscataway, NJ, USA, July 1997, pp. 712-726.

[20] 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, Washington, DC, USA, 30 November - 3 December, 2005, pp. 53-60.

[21] 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)*, Springer Berlin / Heidelberg, Cambridge, MA, June 19, 2005, pp. 91-107.

[22] Y. Zhu, "Efficient processor allocation strategies for mesh-connected parallel computers", *Journal of Parallel and Distributed Computing, vol. 16, no. 4*, Elsevier, San Diego, CA, 1992, pp. 328-337.