Communication Overhead in Non-Contiguous Processor Allocation Policies for 3D Mesh-Connected Multicomputers

Raed Almomani and Ismail Ababneh
Department of Computer Science, Al al-Bayt University, Jordan

Abstract: *Various contiguous and non-contiguous processor allocation policies have been proposed for two-dimensional mesh-connected multicomputers. Contiguous allocation suffers from high processor fragmentation because it requires that a parallel job be allocated a single contiguous processor subset of the exact shape and size requested. In non-contiguous allocation, a job may be allocated multiple dispersed processor subsets. This can reduce processor fragmentation, however it may increase the communication overhead because inter-processor distances can be longer and messages from different jobs can contend for communication resources. The extra communication overhead depends on how allocation requests are partitioned and assigned processors. In this paper, we investigate non-contiguous allocation for three-dimensional meshes. A greedy policy where partitioning is based on the processors available is proposed and compared, using simulation, to contiguous first-fit allocation, and to non-contiguous schemes adapted from previous two-dimensional schemes. In the detailed flit-level simulator, developed for this research, several common communication patterns are considered. The results show that non-contiguous allocation is expected to improve system performance in three-dimensional mesh-connected multicomputers substantially.*

Keywords: *Three-dimensional mesh multicomputers, non-contiguous processor allocation, processor fragmentation, external message contention.*

Received April 23, 2009; accepted March 9, 2010

1. Introduction

Three-dimensional mesh and torus interconnection networks have been used in recent research and commercial distributed memory parallel computers. Examples of such multicomputers are the Cray T3D [21], the Cray XT3 [15] and the IBM BlueGene/L [18]. Also, two-dimensional meshes have been used in several multicomputers, such as the Caltech Mosaic [6] and the Intel Paragon [19]. An important advantage of the 3D mesh over the 2D mesh is its lower diameter and higher bisection width. It can achieve reductions in communication delays given the same number of processors.

Typically, processor allocation proposed in the literature for mesh-connected multicomputers is contiguous and space-shared. A job is allocated a distinct contiguous subset of processors for the duration of its execution, and the subset normally has the same general shape as the multicomputer itself. Numerous contiguous processor allocation policies have been proposed in the literature for two-Dimensional (2D) meshes [3, 9, 12, 14, 16, 17, 22, 23, 27]. Also, a few contiguous allocation policies have been proposed for three-Dimensional (3D) meshes and tori [1, 8, 13, 25]. However, contiguous allocation suffers from high external processor fragmentation. There is external processor fragmentation when free processors are not allocated to a job because they are not contiguous or they do not include a submesh that has the same shape as that requested, although their number is sufficient to satisfy the job's allocation request.

The use of wormhole routing has lead researchers to investigate non-contiguousallocation in 2D mesh-connected multicomputers with the goal of reducing external processor fragmentation [2, 7, 11, 24]. An advantage of wormhole routing over earlier flow control schemes, mainly store-and-forward, is that message latency is not as sensitive to the distance between the source and destination. A message is subdivided into small parts, called flits, and the next hop on the path to the destination is determined when the header of the message is received. The remaining message parts follow the header over the same path to the destination in pipelined fashion.

In non-contiguous allocation, an allocation request can be split into parts that can be allocated non-adjacent available submeshes. Non-contiguous allocation suffers from two problems. First, messages exchanged between the processors of different submeshes allocated to the same job can compete with the messages of other jobs for communication resources. Second, the distances between the processors allocated to a job can be longer than when allocation is contiguous, which can increase the probability of contention and lengthen communication latency. However, the results of previous studies [2, 11, 24] indicate that non-contiguous allocation in 2D mesh-connected multicomputers can have better system performance than contiguous allocation, even when the communication load is very high. This suggests that the reduction in processor fragmentation that results from lifting the allocation contiguity condition in 2D meshes can outweigh the extra communication overhead associated with non-contiguous allocation.

Even though non-contiguous allocation in 2D meshes has been studied in detail, there exists little research on non-contiguous allocation for 3D meshes and tori. In [4], communication is not considered in detail. Instead, a simple equation, proposed in [11] for computing the communication overhead associated with non-contiguous allocation in 2D meshes, is used without modification to compute the execution time of a job when it is allocated non-contiguous submeshes in a 3D mesh-connected system. The specific locations of the submeshes allocated to jobs, contention among messages travelling between submeshes allocated to the same job, and contention with the messages of other jobs are not considered directly.

In this paper, we study non-contiguous allocation for 3D meshes more accurately. We simulate communication at the flit level for several non-contiguous allocation policies and communication patterns in the 3D mesh. Results obtained for 2D meshes may not be directly applicable to 3D meshes because of obvious topology differences. For example, the diameter of the 3D mesh is smaller than that of the 2D mesh, and an internal node in a 3D mesh has six neighbours, instead of four in the 2D mesh.

Related research is surveyed and discussed in the next section. This is followed by a presentation of the non-contiguous schemes in section 3. The system model is specified in section 4, and the performance evaluation of the allocation schemes is presented and discussed in section 5. The paper is concluded in section 6.

2. Previous Related Research

As wormhole-routed mesh and torus interconnection networks have been used in many recent highly-parallel multicomputers, processor allocation for such multicomputers has attracted the interest of many researchers over the past two decades.

2.1. Contiguous Allocation Schemes for Mesh-Connected Multicomputers

Most contiguous processor allocation schemes investigated in the literature have been for 2D meshes [3, 9, 12, 14, 16, 17, 22, 23, 27]. In addition, a few contiguous allocation schemes have been proposed for 3D meshes and tori. In these schemes, it is assumed that a job requests upon arrival the allocation of a submesh of a specified width, depth and height.

2.1.1. Maximal Free List Scheme for 3D Tori

In this scheme [25], all maximal free submeshes are detected and placed in a free list. A free submesh is maximal if it is not a proper subset of any free submesh; that is, it can not be expanded in any of the three dimensions (x, y and z) to form a larger free submesh. For allocation, several heuristics that select a large enough maximal free submesh from the free list were considered.

A submesh is large enough for a request if it has enough free processors in each of the dimensions of the request. In the first-fit heuristic, the first free submesh that can accommodate the request is selected as allocation submesh. In the best-fit heuristic, a submesh that is closest in size to the request is selected for allocation. The size of a submesh is the number of processors it contains.

In worst-fit, the selected allocation submesh is one with the largest size. In addition to these allocation schemes, a *k*-look ahead scheduling heuristic proposed in [9] was adopted. In this heuristic, the waiting requests are examined so that allocation to the current request would leave free submeshes for a maximum number of the *k* largest waiting requests. Also, changing the orientation of requests, proposed in [17] for 2D meshes, is supported so as to reduce processor fragmentation. As example, an $a \times b \times c$ request can be changed to a request for an $a \times c \times b$ submesh.

2.1.2. Scan Search Scheme for 3D Tori

This scheme [13] was proposed with the goal of reducing the allocation time as compared with the best-fit variant proposed in [25]. It is a contiguous recognition-complete first-fit allocation scheme for 3D tori. Using simulations, it is shown in [13] that the measured allocation time of this scan search scheme is smaller than that of the earlier scheme. In addition, in order to study the impact of scheduling, a non-FCFS job scheduling policy is considered along with FCFS. In the non-FCFS policy, the jobs in the waiting queue are considered for allocation in their arrival sequence if the job at the head of the queue can not be accommodated. However, looking inside the queue is halted if the queue head has spent some preset time period waiting. The allocation time complexity of the scan search scheme is $O(WD^2H^2)$. This is superior to the time complexity of the previous scheme proposed in [25], which is $O(W^4D^4H^4)$.

2.1.3. Folding Contiguous Allocation for 3D Meshes

In [1], a folding processor allocation scheme was proposed for 3D meshes. In this scheme, a job is allocated the submesh it has requested if such submesh is available. Otherwise, the largest request side is decremented by one and allocation is re-attempted. This process is repeated until either allocation succeeds or the number of processors requested reaches a load-dependent fraction of the size of the request. This folding fraction increases dynamically with the processor demand of the jobs currently in the system. In addition to folding, the job scheduling policies FCFS and out-of-order scheduling were considered. In out-of-order job scheduling, multiple waiting jobs may be considered for allocation in their arrival order. Simulation results show that allocation request folding and out-of-order job scheduling both improve system utilization and mean job turnaround times.

2.2. Non-Contiguous Allocation Schemes for 2D Meshes

Several non-contiguousallocation policies that differ in the degree of contiguity they maintain were proposed for 2D meshes, and they were evaluated using simulation.

2.2.1. Random Allocation

A request for n processors is satisfied with n randomly selected free processors [24]. There is no processor fragmentation in this policy however, the communication overhead can be high because contiguity among allocated processors is not sought.

2.2.2. Paging

This scheme [24] is denoted as Paging$_{\text{index-scheme}}(m)$, where m is a nonnegative integer. It divides the processors of the multicomputer into square pages of side lengths equal to 2^m, and the page is the allocation unit. Pages are numbered according to several indexing schemes (row-major, shuffled row-major, snake-like, and shuffled snake-like indexing). However, the indexing scheme had little influence on performance. A request for n processors is satisfied with the first free $\lceil n/2^{2m} \rceil$ pages. There is some degree of contiguity because of the indexing schemes used. Contiguity can also be enhanced by increasing m. However, this produces internal fragmentation for $m \geq 1$, and this internal fragmentation increases with m.

2.2.3. Multiple Buddy Strategy

In this scheme, the number, n, of processors requested is converted to a base-4 number of the following form:

$$n = d_k \times 2^k \times 2^k + \ldots + d_0 \times 2^0 \times 2^0 \qquad (1)$$

The allocation policy of (MBS) attempts to satisfy every term i in the request with d_i free $2^i \times 2^i$ processor blocks. If a needed block is not available, the algorithm tries locating a free larger block that it repeatedly breaks down into four buddies until a block of the needed size is obtained. The buddies of a $2^j \times 2^j$ block are $2^{j-1} \times 2^{j-1}$ processor blocks. If the attempts to satisfy a term i fail, the algorithm breaks the term itself into four smaller requests for $2^{i-1} \times 2^{i-1}$ blocks and repeats the process described above. Allocation always succeeds when the number of free processors is sufficient because the request or parts of it can be decomposed into requests for 1×1 processor blocks [24].

2.2.4. Adaptive Non-Contiguous Allocation

This policy attempts first to allocate a contiguous processor submesh of the requested shape and size. If this fails, the request is decomposed into two equal subframes and the allocation algorithm attempts to allocate submeshes for these subframes. If this fails again, the request is split into smaller subframes, and allocation is attempted for the new subframes. The size of a subframe in a step is half its size in the previous step. This process terminates if allocation succeeds for all subframes in the same step, or if it has repeated a specified number of times, denoted by A. Additionally, allocation attempts are halted if a side length of the subframes reaches one [11].

This policy can disperse the allocated submeshes more than it is necessary. Moreover, its recognition capability is incomplete. Allocation can fail although a sufficient number of processors are available. For example, assume that an allocation request for an 8×3 submesh arrives while the state of processors is as shown in Figure 1 and A = 1. The request is subdivided into two 4 × 3 requests, and allocation fails although a 6 × 3 and a 2 × 3 submeshes are available.

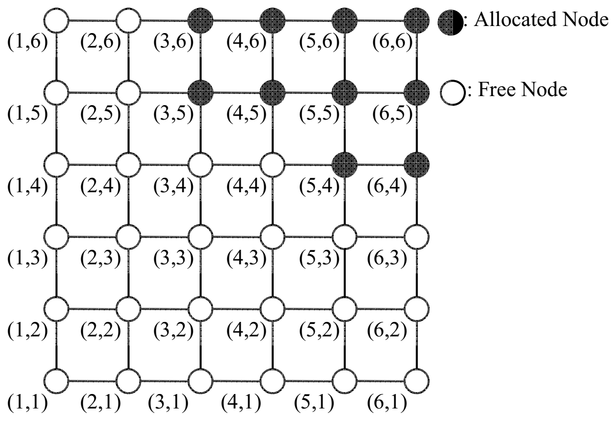

Figure 1. A 6×6 2D mes.

2.3. Non-Contiguous Allocation Schemes for 3D Meshes

There exists little published research on non-contiguous allocation for 3D meshes and tori. In [4], several non-contiguous allocation schemes for the 3D mesh were evaluated using simulation. However, communication was not considered in detail. Instead, a simple equation, proposed in [11] for computing the execution times of jobs when they are allocated non-contiguous submeshes in 2D meshes, was used, as is, to compute the execution times of jobs when they are allocated non-contiguous submeshes in 3D mesh. Communication was not simulated at the flit level. That is, the specific locations of the submeshes allocated to jobs and contention for communication resources were not considered directly. Also, it is not obvious that approximate mathematical models based on flit-level simulation results for 2D meshes are directly applicable to 3D meshes.

3. Three-Dimensional Non-Contiguous Allocation Policies

The target system in this paper is a three-dimensional mesh-connected multicomputer $M(W, D, H)$, where W is the width of the mesh, D is its depth, and H is its height. A processor (node) in $M(W, D, H)$ is represented by the coordinates (x, y, z), where $1 \le x \le W$, $1 \le y \le D$, and $1 \le z \le H$. An internal node is directly connected via bidirectional links to six neighbours: $(x-1, y, z)$, $(x+1, y, z)$, $(x, y-1, z)$, $(x, y+1, z)$, $(x, y, z+1)$ and $(x, y, z-1)$. The eight mesh corner nodes have three neighbours each, other edge nodes have four neighbours, and the remaining peripheral nodes have five neighbours. The size of the 3D mesh, N, is the number of processors it contains, where $N = WDH$. A job is assumed to request the allocation of an $a \times b \times c$ submesh when it arrives, where $1 \le a \le W$, $1 \le b \le D$ and $1 \le c \le H$. A $w \times d \times h$ submesh $S(w, d, h)$ is represented by $(x1, y1, z1, x2, y2, z2)$, where $(x1, y1, z1)$ is the lower left corner of S, $(x2, y2, z2)$ is its upper right corner, $w=x2-x1+1$, $d\ y2-y1+1$ and $h=z2-z1+1$. The size of the submesh is wdh processors. The non-contiguous allocation policies considered in this paper are as follows.

3.1. Greedy Available Allocation

When a parallel job is selected for allocation and the number of free processors is sufficient, a free submesh, $S(w, d, h)$, that is suitable for the request (i.e., $w \ge a$, $d \ge b$, and $h \ge c$) is searched for. If one is found, the submesh $S(a, b, c)$ located in the left-lower corner of $S(w, d, h)$ is allocated to the job and allocation is done. Otherwise, the largest free submesh, $S_l(w, d, h)$, that can fit inside $S(a, b, c)$ is allocated, and it is subtracted from $S(a, b, c)$. The subtraction operation used is one that produces the largest possible non-overlapping submeshes successively. For example, the submeshes produced by the subtraction of a $2\times2\times3$ submesh from a $3\times4\times3$ submesh are $3\times2\times3$ and $1\times2\times3$. The subtraction results are added at the tail of a request-list. The fragments in the request-list are processed until they are all accommodated; that is, until the current job is allocated the number of processors it has requested. The allocation steps are shown in Algorithm 1. This allocation policy maintains some contiguity by giving preference to allocating large free submeshes.

Algorithm 1: Greedy Available Allocation Algorithm

```
Greedy_Available_Allocation (a, b, c){
 Step 1. If (number_of_free_processors < job_size)
             return failure
         else insert requestR(a,b,c) for a × b × c submesh
         in request-list
 Step 2. While there are elements R(aa,bb,cc) in
         request-list do
          if (a free S(x,y,z) is suitable for R(aa,bb,cc)){
              allocate its lower-left S(aa,bb,cc) corner
              submesh to the job and remove R(aa,bb,cc)
              from the request-list
          }
          else {
              find all free submeshes that fit in R(aa,bb, cc)
              allocate largest such submesh L(x,y,z) and
              remove R(aa,bb,cc) from the request-list
              subtract L(x,y,z) form R(aa,bb,cc) and add
              resulting fragments in the decreasing order of
              their size to the tail of request-list
          }
}
```

3.2. Paging Strategy

The 3D mesh is divided into pages that are subcubes with equal side lengths of 2^m, where m is an integer greater than or equal to zero. A page is the allocation unit, and its size, $Psize$, is equal to 2^{3m}. The pages are ordered according to the row-major indexing scheme, as illustrated in Figure 2. If the number of free pages is greater than or equal to the current request, the free pages are scanned starting with the first page until the requested number of pages is allocated. A paging policy is denoted as paging(m). For example, paging (2) means that the pages are $4\times4\times4$ processor blocks. The number of pages requested by a job of size job_size is computed using the equation:

$$Prequest = \lceil job\_size/Psize \rceil \tag{2}$$

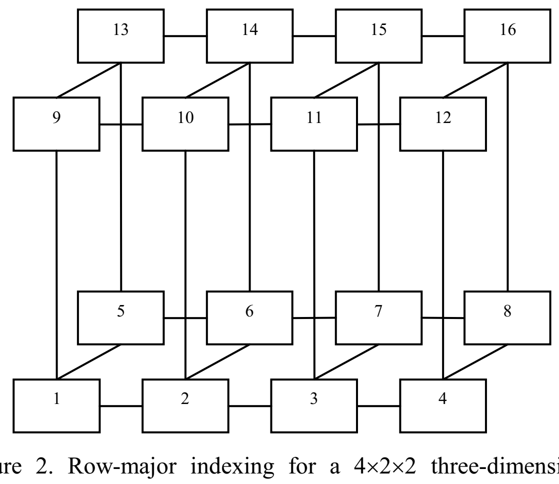

Figure 2. Row-major indexing for a 4×2×2 three-dimensional mesh.

The paging allocation algorithm is as follows:

1. The mesh is initially divided into 3D pages of side lengths equal to 2^m, and the pages are numbered using the row-major indexing scheme, as shown in Figure 2.
2. When the number of free pages is greater than or equal to the requested number of pages, *Prequest*, allocation succeeds. The first *Prequest* free pages are allocated to the job being served.

3.3. Random Allocation Strategy

As in the scheme Random proposed previously for 2D meshes [24], a request for *n* processors is satisfied with *n* randomly selected free processors.

4. System Model

We use simulation to evaluate and compare the greedy available, paging(0), random and contiguous First-Fit allocation policies. We have selected paging(0) as representative of the paging policies because it has no internal processor fragmentation. Moreover, it performed well in [4, 24].

As in previous works, we adopt the simulation performance methodology because it can take into account the dynamic nature of the interactions between applications, operating systems and architectures [10].

The interconnection network assumed uses all-port routers and wormhole switching. The all-port router model specifies that up to six (the maximum number of neighbours) messages can be relayed simultaneously when they require distinct outgoing channels [26]. In wormhole switching, message flits move in a pipeline fashion. If a header flit encounters a busy channel, it is blocked until that channel becomes free. We assume that a flit takes one time unit to move between two neighbouring routers. Neighbouring routers are interconnected by bidirectional channels, and message routing is dimension-ordered and deterministic.

An application is assumed to be composed of a computation phase followed by a communication phase. The computation to communication ratio, *R*, of jobs is assumed to be uniformly distributed over [1, 9] when the jobs do not interfere with each other. That is, the interference-free efficiencies of jobs are distributed over the interval [50%, 90%]. To determine the interference-free communication time for a job of a given size, its simulated communication time for the communication pattern assumed (e.g., one-to-all) is determined by executing the job alone on the simulated target system.

The interference-free communication times for all possible job sizes are pre-determined and stored in a file. There is a separate file for each communication pattern. During the simulation of the allocation algorithms, the interference-free communication time for the current job is read from the appropriate file based on the job size, and a value for *R* is generated. Then, the job's computation time is determined by multiplying these values. After delaying for this computation time, the job enters the communication phase. In this phase, a job sends messages according to the communication pattern being simulated. Note that the new communication time can be longer than the stored communication time because of possible contention with other jobs, for example.

We expect the performance of non-contiguous allocation to depend on the contention that results from *external message interference*. This interference occurs when the messages of two or more jobs need to use the same communication channel at the same time. In addition, non-contiguous allocation can generate *internal message interference*. This interference occurs when messages travelling between submeshes allocated to the same job contend for communication resources.

In this paper, three communication patterns are considered. They are the one-to-all, ring and random communication patterns. In one-to-all, a randomly-selected processor allocated to a job sends a message to all other processors allocated to the same job. In the ring communication pattern, the processors allocated to a job are considered to form a torus linear array and each processor sends a message to its successor; the successor of the last processor is the first one. In the random communication pattern, each processor allocated to a job sends a message to another processor allocated to the same job and selected randomly.

The main performance parameters measured in the simulation experiments are the mean turnaround times of jobs and the allocation effectiveness. The turnaround time of a job is the time that it spends in the system, from arrival to departure. The allocation effectiveness, A_e, measures the ability of an allocation algorithm to avoid processor fragmentation [20]. At the simulation time *t*, $A_e(t)$ is defined by the equation:

$$A_e(t) = P_a/min(N, P_d) \quad (3)$$

In this equation, P_a is the number of allocated processors, *N* is the number of processors in the target multicomputer, and P_d is the total processor demand of the jobs in the system, running or waiting. For example, for N=1000, P_d=1200 and P_a=750, the allocation effectiveness is 75% and processor fragmentation is 25% (100% minus 75%). However, for P_d=400 and P_a=400 the allocation effectiveness is 100% and there is no processor fragmentation. The mean allocation effectiveness is computed over the entire simulation time, and the average turnaround time is computed for completed jobs.

The size of the system assumed in this paper is 1000 processors. It is organized as a $10\times10\times10$ cube of processors. For communication, it is assumed that the length of messages is 250 flits, and the start up latency is 30 time units. Other values were considered in [5], but their results lead to the same conclusions. Therefore, these results are not shown here so as to conserve space. The side lengths of allocation requests are integers that are uniformly distributed over the range [1, 10], and they are generated independently. Jobs arrive from a Poisson source at a rate of λ jobs per time unit.

This rate is varied from low values representing light loads to high values that produce turnaround times that are past the knees of the job turnaround time performance curves. In the simulation experiments, the system load is defined as $\lambda*\mu$, where μ is the mean job service time. Every simulation run executes 1000 parallel jobs, and the runs are repeated enough times so that the mean turnaround times obtained have relative errors that do not exceed 10% with 90% confidence.

5. Results and Comparison

Figures 3 and 4 show the allocation effectiveness and turnaround times for the contiguous First-Fit allocation policy and the non-contiguous allocation policies when the communication pattern is one-to-all. The results for the ring communication pattern are displayed in Figures 5 and 6, and those for the random communication pattern are displayed in Figures 7 and 8. In these Figures, the scheduling policy is FCFS, where only the job at the head of the queue is considered for allocation. We limit ourselves to this policy because it is typically assumed in related studies (e.g., [3, 12, 13, 24]), and our primary goal is to compare allocation policies.

It can be noticed in the Figures that the non-contiguous policies are substantially better than First-Fit. This is because contiguous allocation suffers from high external processor fragmentation. The allocation effectiveness of First-Fit is high when the load is very low because it is then highly likely that a suitable contiguous submesh is available for allocation to a job when it arrives to the system. However, this effectiveness drops rapidly as the load increases. This outcome is compatible with the results of previous works, where contiguous First-Fit allocation achieves only low system utilization for 3D mesh-connected multicomputers [1, 13]. Overall, the additional communication overhead associated with non-contiguous allocation is less significant than the performance advantage that results from the reduction in processor fragmentation that is produced when the allocation contiguity condition is lifted.

The performance differences among the non-contiguous policies are small, however greedy available is overall slightly better than the remaining policies under the heaviest loads. It ranks first or second for the three common communication patterns investigated. The mean turnaround times of paging(0) for the one-to-all and random communication patterns are longer than those of Greedy Available by about 23% and 8% under the heaviest loads considered. The performance of paging(0) is good for the ring communication pattern because paging(0) allocates neighbours located on the same X-axis when they are free. Random performs worse than Greedy Available by about 13.5% under the heaviest load in Figure 6, where the ring communication pattern is assumed. This is because random allocation is not compatible with the near neighbour property of the ring pattern.

It can be seen in Figure 4 that Random performs well under heavy loads when the communication pattern is one-to-all.

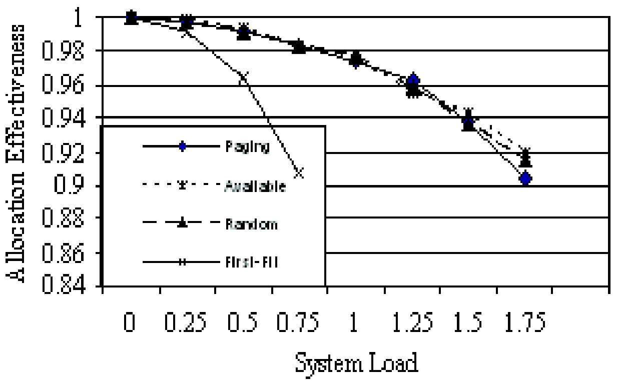

Figure 3. Mean allocation effectiveness of the policies for the one-to-all communication pattern.

The reason is that this communication pattern can make use of multiple communication links simultaneously as the processors allocated to a job are dispersed randomly across the system. In contrast, a node needs to use only a single outgoing link for the ring communication pattern. In this case, Random performs poorly because the distance between communicating neighbours is expected to be relatively large, which increases both the message transit time and the probability of communication.

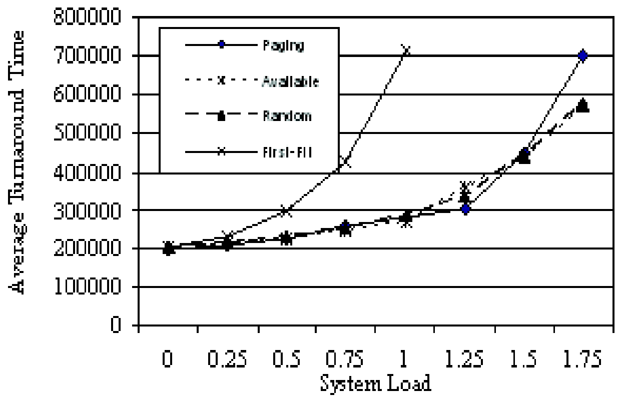

Figure 4. Average turn around time of the policies for the one-to-all on communication pattern.

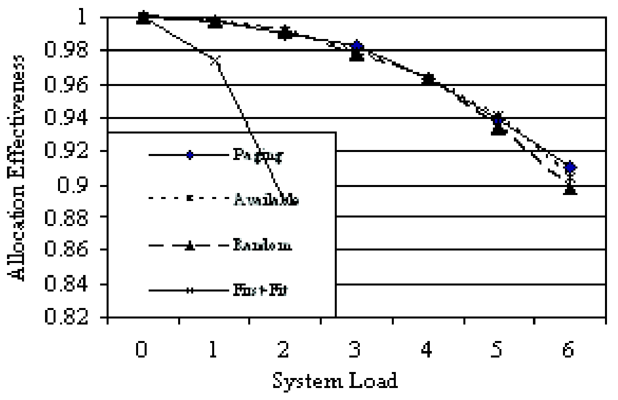

Figure 5. Mean allocation effectiveness of the policies for the ring communication pattern.

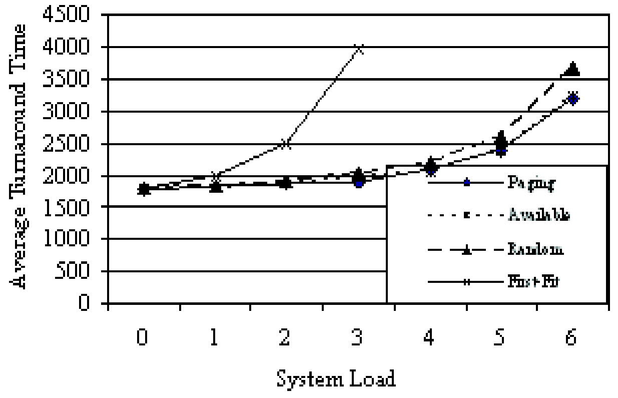

Figure 6. Average turnaround time of the policies.

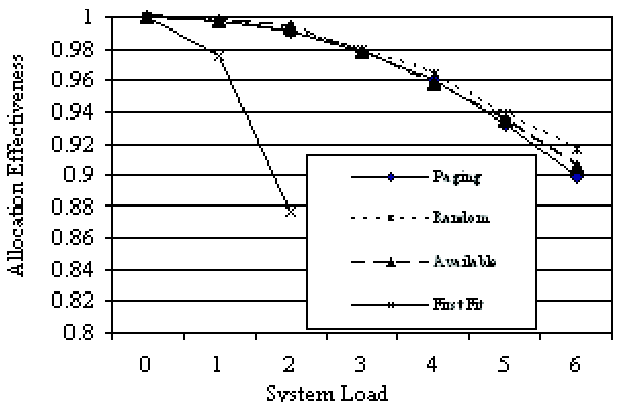

Figure 7. Mean allocation effectiveness of the policies for the random communication pattern.

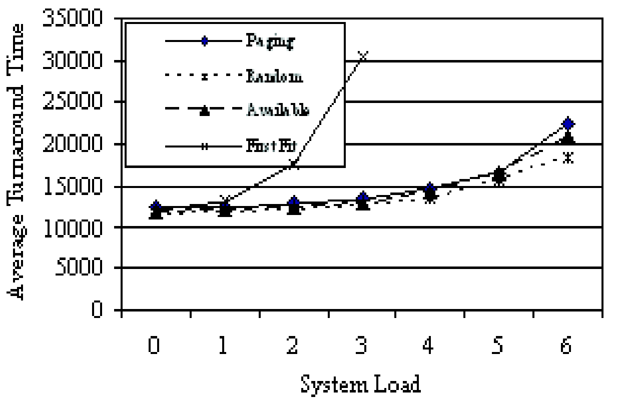

Figure 8. Average turnaround time of the policies for the random communication pattern.

6. Conclusions

In this research, we have studied non-contiguous allocation in 3D-mesh multicomputers using detailed flit-level simulation. Three non-contiguous allocation strategies were considered: greedy available, random and paging(0). The performance of these strategies was compared to the performance of contiguous First-Fit allocation for common communication patterns.

The aim of simulating communication in detail is to evaluate the effect of contention in the interconnection network on the performance of non-contiguous allocation. Simulation results show that non-contiguous allocation can greatly improve performance despite the additional contention in the interconnection network that can result from the interference among messages. This is because non-contiguous allocation is capable of superior utilization (i.e., lower processor fragmentation and superior allocation effectiveness) when it is compared with contiguous allocation, represented in this study by First-Fit. The performance differences among the non-contiguous policies investigated are small; however, greedy available is promising. It ranks first or second for the common communication patterns considered in this paper.

References

[1] Ababneh I. and Bani S., "Non-Contiguous Processor Allocation for 3D Mesh Multicomputers," *Journal of AMSE Advances in Modeling and Analysis*, vol. 8, no. 2, pp. 51-63, 2003.

[2] Ababneh I., "Availability-Based Non-Contiguous Processor Allocation Policies for 2D Mesh-Connected Multicomputers," *Journal of Systems and Software*, vol. 81, no. 7, pp. 1081-1092, 2008.

[3] Ababneh I., "Job Scheduling and Contiguous Processor Allocation for Three-Dimensional Mesh Multicomputers," *Journal of AMSE Advances in Modelling and Analysis*, vol. 6, no. 4, pp. 43-58, 2001.

[4] Ababneh I., "On Submesh Allocation for 2D Mesh Multicomputers Using the Free-List Approach: Global Placement Schemes," *Journal of Performance Evaluation*, vol. 66, no. 2, pp. 105-120, 2009.

[5] Almomani R., *Communication Overhead in Non-Contiguous Allocation for 3D Meshes, Unpublished MS Thesis*, AL Albayt University, 2002.

[6] Athas W. and Seitz C., "Multicomputers Message-Passing Concurrent Computers," *in Proceedings of IEEE Computer*, California, pp. 9-24, 1988.

[7] Bani S., OuldKhaoua M., Ababneh, I., and Mackenzie L., "Comparative Evaluation of Contiguous Allocation Strategies on 3D Mesh Multicomputers," *Journal of Systems and Software*, vol. 82, no. 2, pp. 307-318, 2009.

[8] Bani S., Ould-Khaoua M., and Ababneh I., "Greedy-Available Non-Contiguous Processor Allocation Strategy and Job Scheduling for 2D Mesh Connected Multicomputers," *Computer Journal of International and their Applications*, vol. 15, no. 4, pp. 283-296, 2008.

[9] Bhattacharya S. and Tsai T., "Look Ahead Processor Allocation in Mesh-Connected Massively Parallel Multicomputer," *in Proceedings of International Parallel Processing Symposium*, Cancun, pp. 868-875, 1994.

[10] Chang Y. and Mohapatra P., "Performance Improvement of Allocation Schemes for Mesh-Connected Computers," *Journal of Parallel and Distributed Computing*, vol. 52, no. 1, pp. 40-68, 1998.

[11] Chhabra A. and Singh G., "Knowledge-Based Modeling Approach for Performance Measurement of Parallel Systems," *The International Arab Journal of Information Technology*, vol. 6, no. 1, pp. 77-84, 2009.

[12] Chiu M. and Chen K., "An Efficient Submesh Allocation Scheme for Two-Dimensional Meshes with Little Overhead," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 5, pp. 471-486, 1999.

[13] Choo H., Yoo S., and Youn Y., "Processor Scheduling and Allocation for 3D Torus Multicomputer Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 5, pp. 475-484, 2000.

[14] Chuang J. and Tzeng F., "Allocating Precise SubMeshes in Mesh Connected Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 2, pp. 211-217, 1994.

[15] Cray, Cray XT3 Datasheet, available at: http://www.craysupercomputers.com/downloads/CrayXT3/CrayXT3_Datasheet.pdf, last visited 2004.

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

[17] Ding J. and Bhuyan N., "An Adaptive Submesh Allocation Strategy for 2D Mesh Connected Systems," *in Proceedings of International Conference. Parallel Processing II*, Seoul, pp. 193-200, 1993.

[18] Gara A., Blumrich M., Chen D., Chiu G., Coteus P., Giampapa M., Haring R., Heidelberger P., Hoenicke D., Kopcsay G., Liebsch T., Ohmacht M., Steinmacher B., Takken T., and Vranas P., "Overview of the Blue Gene/L System Architecture," *IBM Journal of Research and Development*, vol. 49, no. 2, pp. 195-212, 2005.

[19] Intel Crop, Paragon XP/S Product Overview, available at: http://books.google.com/books/about/Paragon_XP_S_product_overview.html?id=qkGNkgAACAAJ, last visited 1991.

[20] Ismail I. and Davis J., "Program-Based Static Allocation Policies for Highly Parallel Computers," *in Proceedings of International Phoenix Conference on Computers and Communications*, Scottsdale, pp. 61-68, 1995.

[21] Kessler R. and Schwarzmeier J., "CRAY T3D: A New Dimension for Cray Research," *in Proceeding of COMPCON*, pp. 176-182, 1993.

[22] Kim G. and Yoon H., "On SubMesh Allocation for Mesh Multicomputers: A Best-Fit Allocation and a Virtual SubMesh Allocation for Faulty Meshes," *Computer Journal of IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 2, pp. 175-185, 1998.

[23] Liu T., Huang K., Lombardi F., and Bhuyan N., "A SubMesh Allocation Scheme for Mesh-Connected Multiprocessor Systems," *in Proceedings of International Conference Parallel Processing II*, California, pp. 159-163, 1995.

[24] Lo V., Windisch K., Liu W., and Nitzberg B., "Non-Contiguous Processor Allocation Algorithms for Mesh-Connected Multicomputers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 7, pp. 712-726, 1997.

[25] Qiao W. and Ni L., "Efficient Processor Allocation for 3D Tori," *IEEE International Parallel Processing Symposium*, pp. 466-471, available at: http://citeseer.nj.nec.com/qiao94efficient.html, last visited 1995.

[26] Tsai J. and McKinley P., "An Extended Dominating Node Approach to Broadcast and Global Combine in Multiport Wormhole-Routed Mesh Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 1, pp. 41-58, 1997.

[27] Zhu Y., "Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers," *Journal of Parallel and Distributed Computing*, vol. 16, no. 4, pp. 328-337, 1992.

Raed Almomani received his BS and MSc degrees in computer science from Al al-Bayt University in 1999 and 2002, respectively. From 2003 to 2009, he was an instructor in the Department of Computer Science at Al al-Bayt University. Presently, he is pursuing a PhD degree at Wayne State University, USA. His main research interests are distributed systems, image processing and computer vision.

Ismail Ababneh received his Engineer degree from the National Superior School of Electronics and Electro-Mechanics of Caen, France in 1979. He received the MS degree in software engineering from Boston University in 1984, and the PhD degree in computer engineering from lowa State University in 1995. From 1984 to 1989, he was a software engineer with DAS, Boston, Massachusetts. He is an associate professor in the Department of Computer Science at Al al-Bayt University, Jordan, and a member of Tau Beta Pi and Eta Kappa Nu. Presently, he is a visiting associate professor in the Department of Computer Science at Jordan University of Science and Technology. His main research interests are processor allocation in multicomputers, and routing algorithms for mobile ad hoc networks.