An Efficient Processor Allocation Scheme for Mesh Connected Parallel Computers

Jatin Upadhyay and Prasant Mohapatra
Department of Electrical and Computer Engineering
Iowa State University
Ames, Iowa 50011
E.mail: prasant@iastate.edu

Abstract

Several processor allocation schemes are proposed in the literature for mesh connected parallel computers. All these schemes aim at improving the system performance by reducing internal fragmentation or by enhancing the submesh recognition ability. In this paper, we propose an approach of system partitioning to reduce external fragmentation and thereby improve the system performance. The target systems considered here are two-dimensional meshes where the side lengths are powers of two. Processors are allocated to a partitioned mesh based on their submesh size requirements. The proposed scheme can be implemented in conjunction with any of the existing schemes and thereby can also exploit the advantages offered by those schemes. The performance measurements are done through simulation experiments. Completion time for a fixed number of jobs, internal and external fragmentation, and system utilization are obtained. It is observed that, in most cases, the proposed scheme demonstrates better performance. Time complexity of the proposed scheme is less by a factor of n compared to the corresponding allocation scheme without partitioning, where $n = log_2\{min$ *(width or height of the mesh)* $\}$.

1 Introduction

Mesh connected parallel computers are becoming increasingly popular because of their structural regularity, easy VLSI implementation, and application in numerical algorithms such as sorting, fast Fourier transform, and matrix operations [1, 2]. Examples of several experimental and commercial machines using the mesh topology include MasPar [3], Paragon [4], DASH [5], Touchstone Delta [6], and J Machine [7].

In this paper, we consider processor allocation in two-dimensional mesh connected parallel computers. Processor allocation is concerned with partitioning the whole system and allocating the parts to independent tasks so as to maximize the utilization. Incoming tasks require a specific size submesh and the processor allocator finds an available submesh satisfying the size requirement. The search for an available submesh is done using the underlying allocation algorithm. The allocation algorithms vary in terms of submesh recognition ability and time complexity. The performance of a system is dependent on the allocation algorithm employed for task assignment. Optimal allocation in a dynamic environment is known to be an NP - complete problem [8]. Usually, heuristics are used to devise efficient processor allocation schemes. The main objective of these schemes is to improve system performance by reducing system fragmentation.

There are three different types of fragmentation associated with the allocation policies. These are: internal fragmentation, external fragmentation, and fragmentation due to imperfect recognition. Internal fragmentation occurs when more than the required number of processors are allocated to a task. External fragmentation occurs when the required number of nodes are available but a submesh of the required size is not available in the fragmented system. Fragmentation due to the imperfect recognition ability arises when there is a submesh of the required size available but the allocation scheme is not able to identify it.

Three different schemes have been proposed in the literature for processor allocation in two-dimensional meshes. Li and Cheng have developed a two-dimensional buddy (2DB) strategy [9] which is applicable to square mesh systems and can allocate square submeshes. Furthermore, the side lengths of the square mesh and submeshes can only be powers of 2. The apparent drawbacks of 2DB scheme are high internal fragmentation and low submesh recognition ability. The Frame Sliding (FS) scheme proposed in [10] avoids internal fragmentation by allocating submeshes of exactly the required size. But it creates more external fragmentation, and the fragmentation due to imperfect recognition is also quite high. An allocation scheme proposed by Zhu [11] has the perfect submesh recognition ability (we have referred to this scheme as perfect recognition (PR) scheme) and has no internal fragmentation. It can be observed that while these schemes have tried to reduce the overall fragmentation by eliminating the internal fragmentation and the fragmentation due to imperfect recognition, no efforts have been made to reduce the external fragmentation.

In this paper, we propose an allocation scheme that concentrates on reducing the overall fragmentation by reducing the external fragmentation. We use a partitioning scheme where the job(task) allocation is seg-

regated based upon their submesh size requirements. A specific size job is allocated to a predetermined partition(s). The motivation behind such an allocation policy is to have similar size jobs allocated as close as possible. This heuristic helps in reducing external fragmentation. The novelty of the proposed allocation lies in the fact that any of the existing allocation policies can be used for allocation within the partition, and thereby besides reduction in external fragmentation, it can also take advantage of the simplicity or the recognition ability of the incorporated allocation scheme.

We have developed a simulation platform to evaluate and compare the performance of the proposed scheme with the existing schemes. Following inferences were derived from the simulation experiments.

- The proposed partitioning approach greatly reduces the external fragmentation in the system. Though it introduces some amount of internal fragmentation, the total fragmentation is always less.
- The completion time of a fixed number of jobs using this scheme is always less when compared with the FS or the 2DB schemes without system partitions. PR scheme with and without partitions gives almost similar results.
- The proposed scheme results in higher system utilization.
- The time complexity of the proposed scheme is reduced by a factor of n for a mesh of size $2^i \times 2^j$, where $n = \min(i, j)$, when compared with the corresponding scheme used without partitioning the system.

The rest of the paper is organized as follows. The nomenclature and the previous allocation schemes are discussed in Section 2. In Section 3, we discuss the proposed allocation scheme. The performance evaluation through simulation is detailed in Section 4 along with the discussion of experimental results. The concluding remarks are outlined in Section 5.

2 Preliminaries

2.1 Nomenclature

A two dimensional mesh, denoted as $M_2(w, h)$, consists of $w \cdot h$ processor nodes arranged in $w \times h$ grid. A node in column i and row j is identified as $< i, j >$, and is connected through direct communication links to the nodes $< i \pm 1, j >$ and $< i, j \pm 1 >$, for $1 \leq i \leq w$ and $1 \leq j \leq h$. A two dimensional submesh in $M_2(w, h)$, denoted as $S_2(w', h')$, is a subgrid $M_2(w', h')$ such that $1 \leq w' \leq w$ and $1 \leq h' \leq h$. Figure. 1 shows a rectangular mesh $M_2(8, 4)$ and a submesh $S_2(3, 2)$. The submesh is shown by the shaded region.

In parallel computers, in order to achieve higher performance, all the processors can be space shared by the incoming tasks. An incoming task which requires a submesh of the size $S_2(w', h')$ is denoted as $T = (w', h')$.

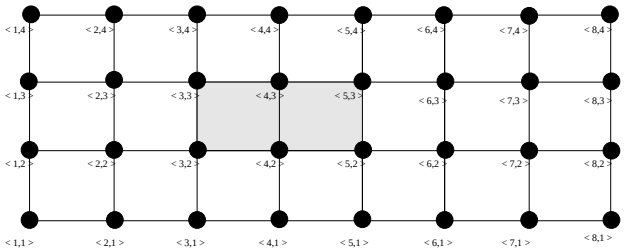

Figure 1: A rectangular mesh $M_2(8, 4)$.

2.2 Previous Allocation Policies

2.2.1 Two-Dimensional Buddy Scheme

The two-dimensional buddy (2DB) scheme proposed in [9] is a generalization of the one-dimensional buddy approach used for storage allocation [12]. This approach is applicable only to square meshes $M_2(w, w)$ and allocates only the square submeshes $S_2(w', w')$ where both w and w' are required to be powers of 2.

Assuming $w = 2^n$, the buddy strategy maintains a set of list of available square submeshes with all the possible side lengths. A separate list is maintained for each submesh of length $w' = 2^{n'}$ where $0 \leq n' \leq n$. An incoming request of task $T = (2^k, 2^k)$ is allocated the first element in the corresponding list if it is available. If the list is empty, an available submesh with $n' > k$ is decomposed and the task is allocated. One square submesh can be decomposed into four square submeshes which form four *buddies* amongst themselves.

The deallocation policy in buddy scheme requires searching for all available buddies of the submesh deallocated, and then merging them to form a bigger submesh. All the lists are updated accordingly.

The time complexity of the allocation and deallocation procedures in 2DB scheme is of $O(n)$ and $O(2^n)$ respectively. The major advantage of 2DB scheme is its simplicity of implementation. The major disadvantage on the other hand is its rigid requirements on mesh and submesh sizes. Under the buddy scheme a task which requires a submesh $S_2(w', h')$ is actually allocated a square submesh $S_2(w, w)$ where $w = 2^{\lceil \log_2 \max(w', h') \rceil}$. This results in large internal fragmentation and hence poor system utilization.

2.2.2 Frame Sliding Method

The frame sliding (FS) strategy is applicable to non-square meshes and allocates a submesh of the exact size to an incoming task [10]. For an incoming request $T = (w', h')$ a *frame* of the size (w', h') is considered. The frame is identified by the lower left corner, and status of the processors within the frame can be determined immediately. The FS strategy searches for an available frame for an incoming task by sliding the frame through the complete mesh. When the nodes in the currently examined frame are not all available, the frame is slid over the *plane* of the mesh for searching

the next candidates. The sliding is done by taking horizontal and vertical strides equivalent respectively to the width and height of the requested submesh. The process is continued until a frame involving only free nodes is found or all candidate frames are exhausted. In the latter case, the incoming task is queued till the deallocation of a job, when the allocation procedure is retried.

If there are n allocated tasks in the system, the allocation time complexity of the FS scheme is $O((w*h)/(w'*h'))$. However, since tasks are assigned exactly the same size submesh as per their requirements, the FS strategy has no internal fragmentation.

2.2.3 Allocation with Perfect Recognition

The 2DB and FS schemes do not always recognize a submesh even if one exists in the mesh. The 2DB scheme searches for an available submesh only at predetermined locations which are disjoint. Thus the overlapping regions are ignored. The imperfect recognition in FS scheme is due to the fact that the frame is slid in steps equal to its height and width of the required submesh. 2DB and FS schemes do not have perfect recognition ability and hence create extra fragmentation in the system. The inability of recognition can be defined as fragmentation due to imperfect recognition. An allocation scheme with perfect recognition (PR) was first proposed by Zhu[11].

Zhu proposed two different strategies for allocation using perfect recognition – the First Fit(FF) strategy and the Best Fit(BF) strategy. Both these schemes maintain a busy array representing the allocation state of the mesh. For each incoming task, the busy array is searched and a coverage array is determined. Coverage array denotes whether the corresponding processor can be the base of the incoming task or not. Once the coverage array is generated, the FF scheme picks up the first available processor which can serve as the base and allocates the job. The BF strategy on the other hand chooses the base in such a way that it has the maximum number of busy neighbors. If there are two positions with the same number of busy neighbors, the one with the minimum *area* is chosen Here *area* is defined as the multiplication of the numbers of consecutive free processors in each direction.

This scheme always recognizes a submesh if it exists. The time complexity of the BF strategy is $\Theta(w*h)$ [11]. Since the BF scheme searches through the busy array twice to generate the coverage array while the FF scheme stops as soon as it finds the first available base, the time complexity of the FF strategy is less than that of the BF strategy. Results show that in most of the cases the FF scheme gives better results than the BF scheme due to the dynamic nature of the system.

3 The Partitioned Allocation

3.1 Motivation

As discussed in the previous section, the 2DB scheme has all the three types of fragmentation - internal, external and fragmentation due to the imperfect recognition ability. At the cost of increased complexity, the FS scheme eliminates the internal fragmentation. External fragmentation and the fragmentation due to imperfect recognition however still exist in the FS scheme. The PR scheme searches all the possible locations for the required submesh and thereby eliminates the fragmentation due to the imperfect recognition ability.

Two observations can be made regarding above mentioned allocation policies. First, the main aim of these schemes is to reduce the overall system fragmentation by reducing the internal fragmentation and the fragmentation due the to imperfect recognition. No efforts are made to reduce the external fragmentation. Second, the maximum utilization achieved through these schemes is only around 50% [11].

The above two facts suggest that the external fragmentation plays a very critical role in the system performance. In this paper, we propose an allocation scheme that reduces the external fragmentation by statically partitioning the system. Jobs are allocated to the partitions corresponding to their sizes. Allocation within the partition can use any of above mentioned allocation policies. Though the proposed scheme introduces some internal fragmentation, it is observed that the overall fragmentation is always less and the system performance improves.

Another advantage of the proposed scheme is that due to the partitioning of the system, the search space and hence the allocation complexity of the algorithm reduces by a significant factor.

3.2 Partitioning Mechanism

The heuristic we propose for the system partitioning is derived from the main cause of the external fragmentation in mesh systems. External fragmentation occurs due to the distribution of the smaller jobs throughout the system which effectively blocks the larger jobs from being allocated, even if there are enough number of processors available. This problem can be eliminated if we can allocate the jobs of similar sizes together, in a predefined location in the system.

We propose to partition the system as follows. Initially assume that the mesh is square and has dimensions in powers of 2. Later in Section 4.3 we have generalized this scheme for rectangular meshes. Section 4.3 also investigates other methodologies for partitioning the system.

Consider the square mesh of width and height $w = 2^n$, as shown in Figure 2. The mesh is divided equally into 4 square partitions each with its width and height equal to half of the width and height of the original mesh. One of these four partitions is further divided into 4 more partitions, each with dimensions equal to half of the original dimensions of the partition. Since the original mesh size is assumed to be in powers of 2, all these partitions will also have their dimensions in powers of 2. The system is divided recursively in this fashion until the last partition is of the size 1×1.

The above partitioning mechanism results in exactly three partitions of identical size except the last partitions of size 1×1 which are four. Jobs will be allocated only to the partitions corresponding to their

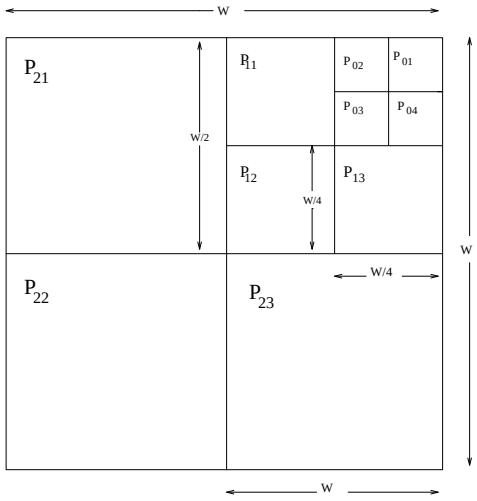

Figure 2: Partitioning Mechanism in a Mesh $M_2(w, w)$

sizes.

Static partitioning of the system as described above may result in two problems. First, it may cause large internal fragmentation. Second, if the job size distribution is not uniform, it may results in jobs unnecessarily waiting for their partitions while other partitions in the system are idle.

The proposed partitioning scheme can handle these problems. To reduce the internal fragmentation, instead of allocating the whole partition to a job as done in the normal partitioning approach, we propose to use the existing allocation strategies such as 2DB, FS or the PR scheme for allocation within the partition. Secondly since the proposed partitioning forms four equal, smaller size partitions from one larger partition, 4 jobs corresponding to one particular size can always be merged to form a single job corresponding to one higher size partition. Similarly, a single job can be moved to all lower size partitions if all of them are free. This dynamic movement of jobs is described in detail in the next subsection.

3.3 Allocation algorithm

Consider a mesh $M_2(w, w)$ where $w = 2^n$. The system is partitioned as explained in the previous section, so there would be $(3\log(n) + 1)$ partitions, 3 partitions, each of size 2^i where $1 \leq i \leq n-1$ and 4 partitions of size 1×1. Without the loss of generality, we will assume that there are exactly three partitions for all sizes. We shall denote all the partitions of size $2^i \times 2^i$ collectively as P_i and the individual partitions amongst P_i as P_{ij} where $j = 1, 2, 3$. An example of this notational structure is indicated in Figure 2 by considering $w = 8$.

We would limit the incoming task sizes to half of the mesh size. When a task requests a larger submesh than this, it has to wait until almost all the jobs in the system are deallocated. This step is independent of the employed allocation scheme. The size restriction of the incoming tasks can also be removed as discussed in Section 4.3. The algorithm for processor allocation and deallocation can be formally described as follows.

Processor Allocation

Let the incoming task be $T = (w', h')$, $1 \leq w', h' \leq w/2$. For notational convenience, we shall denote the allocation policy used within the partition as AP, which can be either 2DB, FS, or PR scheme.

Step 1 Identify the corresponding system partitions P_i for an incoming task $T = (w', h')$ where i is the smallest value that satisfies the inequalities $w' \leq 2^i$ and $h' \leq 2^i$.

Step 2 If P_{ij} is entirely free for some j, allocate T in partition P_{ij}. Stop.

Step 3 For $j = 1$ to 3, if allocation is possible in partition P_{ij} using AP, allocate T in that partition. Stop.

Step 4 If job queue for partitions $P_i > 4$ and P_{i+1} is free, combine 4 jobs as one with service time as maximum of all of them and allocate it in partition P_{i+1}. Stop.

Step 5 If job queue of partitions $P_i > 4$ and P_j is free, for all $j < i$, move the job at the top of the queue and allocate it to all the partitions P_j, $j < i$. Stop.

Step 6 Attach the job in the queue of partitions P_i and wait until a job in the partition is deallocated. Go to Step 2.

Processor Deallocation

Processor deallocation procedure will depend upon the allocation scheme used in the partition. In case of buddy scheme, it will involve merging the deallocated submesh with its three buddies, if they exist, and accordingly updating the lists of each submesh size. In case of FS method, deallocation steps will have to remove the deallocated submesh from the busy list of that partition. In PR scheme, the busy array entries corresponding to the deallocated submesh are updated and the submesh is removed from the list of allocated submeshes.

There are always at least three partitions corresponding the size of each incoming job. All the jobs for the same size partitions are allocated using 2DB, FS or PR scheme in the first come first served(FCFS) discipline. The job combination and dynamic movement is considered only when all the destination partitions are entirely empty. Thus the proposed scheme is totally fair to jobs of all sizes. The dynamic movement also enhances system utilization. Since jobs are allowed to move from one partition to the higher or lower size partitions if they are entirely free, this scheme seldom results in any part of the system being idle for a long time while a job of equal or lesser size is waiting in the queue.

Since the proposed scheme uses existing allocation policies for job allocation within the system partitions,

it can be viewed as one level higher than the existing allocation schemes and can be implemented easily. The processor allocator can maintain the status of each partition in the system and is responsible for implementation of the allocation algorithm. When a request arrives, it first decides the appropriate partition for the job. Once the partition is decided the allocation within the partition can be implemented using the chosen allocation scheme. The processor allocator takes care of job movements, if required.

The time complexity of the allocation step in the proposed scheme can be calculated in the following manner. Assuming that we use PR scheme for allocation within the partitions, the complexity of allocation within the partitions would be $\Theta(w' \times w')$ if the partition size is $(w' \times w')$ [11]. Let the mesh $M_2(w, w)$, $w = 2^n$, be partitioned as described previously. So there would be three partitions of size $P_{n-1} = 2^{n-1} \times 2^{n-1}$, three partitions of size $P_{n-2} = 2^{n-2} \times 2^{n-2}$ and so on till partitions of size $P_0 = 1 \times 1$ which would be 4.

If the probability that a job is in partition P_i is p_i, the total time complexity of our scheme assuming uniform distribution of jobs can be calculated as:

$$
\begin{aligned}
\mathrm{T} &= 3p_{n-1}(\frac{w}{2} \times \frac{w}{2}) + 3p_{n-2}(\frac{w}{4} \times \frac{w}{4}) + \\
&\quad \cdots + 4p_0(\frac{w}{w} \times \frac{w}{w}) \\
&= 3p_{n-1}(2^{n-1} \times 2^{n-1}) + 3p_{n-2} \\
&\quad (2^{n-2} \times 2^{n-2}) + \cdots 4p_0(2^0 \times 2^0) \\
&= 3p_{n-1}(2^{2(n-1)}) + 3p_{n-2}(2^{2(n-2)}) \\
&\quad + \cdots + 4p_0(2^0) \\
&\approx 3 \cdot 2^{2n} \cdot p[2^{-2} + 2^{-4} + \cdots + 2^{-2n}] \\
&\quad \text{(neglecting the 4th partition of size } 1 \times 1) \\
&= 3 \cdot 2^{2n} \cdot p\left[\frac{2^{-2}(1 - 2^{-2n})}{1 - 2^{-2}}\right] \\
&\approx 3 \cdot 2^{2n} \cdot p\frac{1}{3} \\
&= 2^{2n} \cdot p \\
&= \Theta(w \times w) \cdot p \\
&= \Theta(w \times w) \cdot \frac{1}{n} \qquad (1)
\end{aligned}
$$

Similar derivations can be carried out for FS and 2DB schemes. In all the cases, it can be proved that the proposed scheme reduces the search space by the factor of n where $n = \log_2 w$.

4 Performance Evaluation

Extensive simulation was carried out to evaluate the performance of the proposed allocation scheme. Results for all the three schemes, namely 2DB, FS and PR schemes under the proposed strategy were collected and are presented. Simulation was carried out on mesh systems of various sizes. For comparison purposes we have presented the results for (64×64) and (32×16) meshes. Other results also follow the same trend.

4.1 Simulation Environment

The system was simulated by using a discrete event driven model. The mesh was logically partitioned as described in Section 3.2. The implementation within a partition was according to the allocation policy used. The input to the system are the *mesh width* and the *mesh height*, the *mean inter-arrival time and mean service time* of the jobs and the *job side length distribution*. The output parameters of the model are *completion time, system utilization, and system fragmentation*.

Initially the whole system is assumed to be free. Total 10000 jobs are then generated and allocated until all the jobs are complete. Job inter-arrival time is assumed to follow exponential distribution with a mean of 1 time unit. The performance was studied for exponential distribution of the service time (execution time) with a mean of 10 time units. The results were averaged over many simulation runs and were found to be consistent with maximum deviation of only 1%.

It was assumed that all the incoming jobs are of rectangular (some of them could be square). Since in the proposed scheme, the largest system partitions have lengths and widths equal to half of the mesh length and width respectively, the jobs are also limited to this size. Three distributions were chosen to represent the side lengths of these jobs – uniform distribution, increasing distribution and decreasing distribution. Uniform distribution represents the cases when the side is uniformly distributed between 1 and maximum allowable value as discussed above. Increasing and decreasing distributions on the other hand assume probability of larger submesh being requested as very high and less respectively. These probabilities are shown in Table 1 for a 64×64 system. Here P[a,b] represents the probability that the side length is within the interval a and b. It should be noted that both the height and the width of a job are always considered under the same distribution. However two different random numbers are used, one for each side, to generate the lengths.

Once a job is generated, depending on its size it is queued at an appropriate server. The server attempts to allocate the job at the head of the queue using the algorithm presented in the earlier section. If a job can not be allocated, it is kept waiting in the queue until a submesh is deallocated. The allocation is reattempted at this time. Within a partition the jobs are allocated using the chosen allocation policy.

4.2 Results and Discussions

The following performance parameters were collected for various job side length and service time distributions. (1) Completion time (2) System utilization and (3) Internal, external and total fragmentation. Completion time is defined as the time taken to complete execution of all the jobs. System utilization is calculated as the average of the percentage of the the busy time of all processor. The fragmentation are calculated as defined in [9]. Internal fragmentation (F_{int}) is the percentage of overallocated processors

Table 1: Performance comparison for FS scheme.

Performance		Submesh size distribution		
		Unif.	Incr. [a]	Decr. [b]
Completion Time	FS	1758	3021	1004
	FS/P	1595	2354	993
System Utilization	FS	37.3	44.3	22.3
	FS/P	41.4	56.6	23.0
External Fragment.	FS	29.3	25.5	14.3
	FS/P	13.6	6.1	10.9
Internal Fragment.	FS	0	0	0
	FS/P	9.9	14.4	0
Total Fragment.	FS	29.3	25.5	14.3
	FS/P	22.1	19.6	10.9

[a] P[1,16] = 0.2, P[17,24] = 0.2, P[25,28] = 0.2,P[29,32] = 0.4
[b] P[1,4] = 0.4, P[5,8] = 0.2, P[9,16] = 0.2,P[17,32] = 0.2

Table 2: Performance comparison for FS scheme, square Jobs.

Performance		Submesh size distribution		
		Unif.	Incr. [a]	Decr. [b]
Completion Time	FS	2186	3137	1166
	FS/P	1564	2314	1014
System Utilization	FS	39.8	48	35.5
	FS/P	56.5	65.2	40.3
External Fragment.	FS	27.8	23.1	28.5
	FS/P	11.5	6.2	13.3
Internal Fragment.	FS	0	0	0
	FS/P	3.2	4.6	0.2
Total Fragment.	FS	27.8	23.1	13.3
	FS/P	14.4	10.51	13.1

[a] P[1,16] = 0.2, P[17,24] = 0.2, P[25,28] = 0.2,P[29,32] = 0.4
[b] P[1,4] = 0.4, P[5,8] = 0.2, P[9,16] = 0.2,P[17,32] = 0.2

over the allocated mesh processors. External fragmentation (F_{ext}) is the percentage of total available mesh processor to the system size at each allocation failure, averaged over all allocation attempts. The total fragmentation, F_{tot}, is equal to $(1 - F_{ext}) \times F_{int} + F_{ext}$.

Table 1 shows comparison between the normal FS scheme and the proposed scheme with FS used within the partitions for a 64×64 mesh. All the three distributions for the job side lengths are considered. The results show clear performance improvement in all the three types of job side length distributions. The partitioning scheme results in less completion time and higher utilization in all the three cases. As evident from the table this is mainly due to the significant reduction in the external fragmentation. Note that the proposed scheme introduces some amount of internal fragmentation due to the job movement in which the whole partition is occupied by a single job. However, the combined effect of internal and external fragmentation is always less.

It can be observed that the performance improvement is least significant for the *decreasing* distribution. The reason for this behavior is the fact that, under this distribution, most of the requests are for smaller submeshes. The system although fragmented can accommodate incoming jobs of smaller sizes without many allocation failures. On the other hand, in the proposed scheme, since the system is partitioned and no jobs are allowed to be allocated across the partition boundaries, there is always a minimum "built-in" fragmentation in the scheme irrespective of the size of the incoming jobs. This puts an upper limit on the performance improvement using this scheme.

During the simulation runs it was observed that under uniform distribution, the probability that a submesh of the size say, 32×1 is requested is equal to the probability of any other size submesh being requested. This assumption of equal probability would be highly unlikely in the actual systems. So we also collected the same system performance measures assuming all the jobs to be square, i.e by generating only one dimension and assuming the other dimension to be same. These results are presented in Table 2. The results clearly show that FS scheme with partitions gives superior performance over the normal FS strategy.

The same simulation runs were carried out for all the three – 2DB, FS and the PR allocation schemes. Figure 3(a) compares the completion time of all the three schemes with the proposed scheme. Figure 3(b) shows the same results for square jobs. The results presented here are for the uniform job side length distribution. Increasing and decreasing distributions also follow the same trend. The plots show that the proposed scheme results in improved system performance in almost all the cases. This improvement is minimum in case of 2DB scheme because of its large internal fragmentation. FS scheme with partitions results in significant reduction in the completion time of the jobs while PR scheme with and without partitions gives almost the same value for the completion time. It should be noted that the time complexity of PR scheme with partitioning is significantly less than that of the PR scheme without partitioning. Figure 3(b) shows the same results for square jobs. The 2DB scheme is very well suited for the square jobs and hence shows better performance as compared to the FS strategy. It is interesting to note that all the three schemes 2DB, FS and PR with partitioning gives almost same completion time. Since jobs are square in nature, and the way system is partitioned, there can not be more than one job in a partition. Thus no matter what scheme we use within the partition, the allocation remains same.

Figures 4(a) and 4(b) plot fragmentation caused by each scheme and compares them with the fragmentation of the proposed scheme. The results shown are for the uniform job side length distribution. It is apparent that using the 2DB, FS or PR scheme, the partitioning approach results in significantly less fragmentation. In the case of the buddy scheme, the internal fragmenta-

Figure 3: Completion Time of allocation Schemes (a) Rectangle jobs (b) Square jobs.

Figure 4: Fragmentation Vs. allocation Schemes for uniform Job side length distribution (a)Rectangle jobs (b) Square jobs

tion remains almost the same while the external fragmentation is reduced. For the FS and PR schemes, the partitioning approach introduces some amount of internal fragmentation but the reduction in external fragmentation overweighs this small increase and the total fragmentation is always less. Note that the completion time as shown in Figure 3(b) are almost same in case of square jobs for all the three schemes when used with partitioning. The fragmentation values for 2DB however is too large. This is due to the fact that 2DB needs subcubes with side lengths in powers of two. While this introduces internal fragmentation, the completion time is not affected because in any case, one partition can have only one job.

The total fragmentation of a scheme is defined as $F_{tot} = F_{int} + F_{ext} - F_{int}F_{ext}$. For clarity, however, we have plotted the actual values of internal and external fragmentation and hence the column height does not necessarily represent the total fragmentation. It can be observed from the graphs that since the sum of internal and external fragmentation is always less for the proposed scheme than the corresponding allocation schemes without partitioning, the total fragmentation would be even less. The internal fragmentation in buddy scheme is due to dilation of jobs to powers of 2. Internal fragmentation in partitioned FS and PR schemes is only due to the job movements.

4.3 Extensions of the Partitioning Approach

The allocation algorithm presented here is suited not only for square meshes but also for the rectangle meshes. The partitioning of the rectangle meshes can be done in a similar way as for the square meshes. Partitions in these cases would not be square and would have their widths and heights in proportion respectively to the mesh width and mesh height. Since within a partition our policy allows use of any of the 2DB, FS or the PR schemes, rectangular partitions would be equally acceptable.

For example, for a mesh $M_2(32, 16)$, which is used in Touchstone Delta system, the partitions can be made of size 16×8, 8×4, 4×2 and 2×1. The incoming jobs can be allocated to any of these partitions depending upon their sizes. Here we have assumed that the side lengths of the rectangular meshes are in powers of 2. This assumption is based on the sizes of the practical rectangle systems like Touchstone Delta and Intel Paragon.

We simulated our scheme for the mesh $M_2(32, 16)$ and collected the results. Table 3 shows these results for normal FS and FS with partitions. The interarrival time and the service time of the tasks are again assumed to be exponential with a mean of 1 and 10

Table 3: Performance comparison for FS scheme on $M_2(32, 16)$

Performance		Submesh size distribution		
		Unif.	Incr. [a]	Decr. [b]
Completion Time	FS	1704	2742	998
	FS/P	1563	2352	996
System Utilization	FS	42.8	52.2	26.1
	FS/P	46.7	60.9	26
External Fragment.	FS	26.2	21.7	8.8
	FS/P	16.2	9.3	10.2
Internal Fragment.	FS	0	0	0
	FS/P	7.2	10.9	0
Total Fragment.	FS	26.2	21.7	8.8
	FS/P	22.2	19.18	10.2

[a] width: P[1,16] = P[17,24] = P[25,28] = 0.2, P[29,32] = 0.4
height: P[1,4] = P[5,12] = P[13,14] = 0.2, P[15,16] = 0.4
[b] width: P[1,4] = 0.4, P[5,8] = P[9,16] = P[17,32] = 0.2
height: P[1,2] = 0.4, P[3,4] = P[5,8] = P[9,16] = 0.2

time units respectively. Probabilities for the increasing and decreasing job side length distributions are mentioned below the table.

The performance measure follows the same trend as was observed for the square meshes. The proposed scheme performs better than the FS scheme for the uniform and increasing distributions while the results are almost the same for the decreasing distribution.

The time complexity of the proposed scheme for rectangle systems is also less by a factor of n than the corresponding normal allocation scheme without partitioning. Here, $n = \log_2\{\min(w, h)\}$ for the mesh $M_2(w, h)$.

The proposed scheme is based on one way of partitioning the system. There could be other ways also. For example, the whole mesh can be divided in two equal but nonsquare parts by partitioning the system horizontally or vertically. One of these parts can again be divided into two parts, this time dividing horizontally if the division in the first case was vertical or vice-versa. This process can be repeated till we get the smallest desirable partition size. This partitioning scheme can allow jobs larger than the jobs allowed in our scheme but the combination and movement of jobs in this case would be complicated.

5 Conclusions

An efficient processor allocation strategy using a partitioning scheme for two dimensional mesh connected computers is reported in this paper. The previous allocation algorithms aimed at reducing internal fragmentation and improving the submesh recognition ability. This paper presents an effort in reducing external fragmentation by partitioning the system and allocating similar size jobs together. The existing schemes such as 2DB, FS or PR can be used within the system partitions. The results indicate that the performance is considerably improved in almost all the cases by using the partitioning allocation scheme proposed here. The time complexity using the proposed scheme is also reduced by a factor of n with respect to the incorporated allocation algorithm, where $n = \log_2\{min(w, h)\}$ for the mesh $M_2(w, h)$. Since the proposed scheme is implemented on top the existing allocation schemes - 2DB, FS or PR, it can also exploit the inherent advantages of these schemes.

The work reported in this paper will instigate researchers for devising more efficient allocation policies which can reduce or eliminate all the three types of fragmentation.

References

[1] D. Nassimi and S. Sahni, "Bitonic Sort on a Mesh-Connected Parallel Computer," *IEEE Trans. Comput.*, pp. 2-7, Jan. 1979.

[2] R. Miller, V. P. Kumar, D. Reisis, Q. Stout, "Parallel Computations on Reconfigurable Meshes," *IEEE Trans. Comput.*, pp. 678-692, June 1993.

[3] T. Blank, "The MasPar MP-1 Architecture," Proceedings of the *IEEE Compcon*, Feb. 1990.

[4] Intel Corporation, *Paragon XP/S Product Overview*, 1991.

[5] D. Lenoski, J. Laudon, et. al., "The Stanford DASH Multiprocessor," *IEEE Trans. Comput.*, pp. 63-79, March 1992.

[6] Intel Corporation, *A Touchstone DELTA System Description*, 1990.

[7] M. Noakes, D. A. Wallach and W. J. Dally, "The J-Machine Multicomputer: An Architectural Evaluation," *Int. Symp. on Computer Architecture*, pp. 224-235, 1993.

[8] D. E. Knuth, *The Art of Computer Programming, Vol.1 Fundamental Algorithms*, Addison-Wesley Publishing Co., 1973.

[9] K. Li and K. H. Cheng, "Job Scheduling in a Partitionable Mesh Using a Two Dimensional Buddy System Partitioning Scheme," *IEEE Trans. on Parallel and Distributed Systems*, pp. 413-422, Oct. 1991.

[10] P. J. Chuang and N. F. Tzeng, "An Efficient Submesh Allocation Strategy for Mesh Computer Systems," *Int. Conf. on Distributed Computing Systems*, pp.256-263, May 1991.

[11] Y. Zhu, "Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers," *Journal of Parallel and Dist. Computing*, 16, pp. 328-337, 1992.

[12] K. C. Knowlton, "A Fast Storage Allocator," *Commun. ACM*, Vol.8 pp.623-625, Oct.1965.