A Minimal Fragmentation Algorithm for Task Allocation in Mesh-Connected Multicomputers

T. SRINIVASAN, Jayesh SESHADRI, Arvind CHANDRASEKHAR and J. B. SIDDHARTH JONATHAN

***Abstract*—Efficient allocation of processors to incoming tasks in tightly coupled systems is crucial for achieving high performance. A good allocation algorithm should identify available processors with minimum overhead. In addition, it should be submesh recognition complete and should minimize fragmentation as far as possible. In this paper, we propose an efficient task allocation mechanism called the Minimal Fragmentation Algorithm (MFA). By weighting the available nodes on the basis of their adjacency to existing busy submeshes or the mesh boundary, we identify nodes that, if chosen as the base for task allocation, would result in minimal external fragmentation. An analysis of the complexity of the proposed algorithm reveals that our scheme provides highly competitive performance.**

***Index Terms*— Complete submesh recognition, Efficient task allocation, Mesh connected architectures, Minimum fragmentation.**

I. INTRODUCTION

MULTICOMPUTER systems, consisting of many processing elements connected through a high speed network, have become widespread in engineering and scientific applications. Among various interconnection topologies developed for such systems, the two-dimensional (2D) mesh has become popular due to its simplicity and efficiency. Some experimental and commercial machines that have been developed uses a mesh are the Tera Computer System, Intel Touchstone Delta, Stanford DASH [1] and so on.

A multicomputer operating system (OS) should support a multiuser environment in which multiple parallel jobs are executed simultaneously. Efficient management of the system resources to achieve this functionality is the task of the processor management system of the OS. The OS consists of a job scheduler and a process allocator. The job scheduler selects the task to be processed from the waiting queue according to a job scheduling policy. The allocator finds a free submesh for the task using an appropriate allocation scheme. This paper deals with processor allocation.

The requirement here is to allocate incoming tasks to submeshes of appropriate size in the 2D mesh-based system. The size of the submesh can range from one node to the entire mesh. As the size of the mesh grows, efficient allocation becomes more demanding. The allocation scheme needs to maximize the processor utilization while minimizing allocation time. A critical attribute of such a scheme is its ability to find available submeshes for incoming requests if they exist; this is called submesh recognition ability. An allocation algorithm is said to have complete submesh recognition ability when it can always find a free submesh for an incoming job if one is available. Another important aspect of an allocation scheme is its speed. A fast allocation algorithm is essential for the minimization of runtime overhead.

Prior research on processor allocation has concentrated on perfecting the submesh recognition of allocation algorithms. Various schemes have been proposed. They however, either lack complete submesh recognition capability or achieve it at the expense of very high allocation overhead.

In this paper, we propose a simple, fast and efficient recognition-complete processor allocation scheme called the Minimum Fragmentation Algorithm (MFA) that attempts to minimize the external fragmentation to as great an extent as possible. By giving preference to nodes that are adjacent to existing busy submeshes, we try to find a submesh that fits as snugly as possible into the existing configuration of the mesh. We scan the nodes on the edge of each busy submesh to identify the candidate nodes. The allocation at each candidate node is assigned a weight called the optimality index which indicates the 'goodness of fit' of that submesh. This scheme ensures the best fit of the incoming task while still maintaining competitive time complexity and complete submesh recognition ability.

The rest of the paper is organized as follows. Section II presents the definitions and notations used throughout the paper. The existing task allocation schemes are reviewed in Section III. Section IV proposes our task allocation scheme in detail, and discusses two examples. In Section V, the time complexity of the proposed scheme is analysed and comparative studies with respect to the other allocation

Manuscript received October 31, 2004.

T. Srinivasan is with the Department of Computer Science in Sri Venkateswara College of Engineering, Sriperumbudur, India 602105. (telephone: 91-4111-262321, e-mail: tsrini@svce.ac.in).

J. Seshadri is with the Department of Computer Science in Sri Venkateswara College of Engineering, Sriperumbudur, India 602105.(e-mail: jayeshs2000@yahoo.co.in).

A. Chandrasekhar is with the Department of Computer Science in Sri Venkateswara College of Engineering, Sriperumbudur, India 602105.(e-mail: arvindcac@hotmail.com).

J.B. Siddharth Jonathan is with the Department of Computer Science in Sri Venkateswara College of Engineering, Sriperumbudur, India 602105.(e-mail: jonathansiddharth@yahoo.co.in).

schemes are described. Section VI details our future work with respect to improving MFA. We finally conclude with Section VII.

II. Definitions and Notations

A 2D mesh, denoted as M(w, h) is a w x h rectangular grid of wh nodes. Each node in the mesh refers to a processor. A processor in column x and row y is represented by a coordinate <x ,y> (0 x w and 0 y h). A non-boundary node is connected to four nodes represented by <x 1, y> and <x, y 1>. A boundary node has 2 or 3 neighboring nodes based on its location within the entire mesh. It is assumed that the column and row indices increase from left to right and bottom to top.

Definition 1: A 2D submesh S(p,q) in the mesh M(w, h) is a subgrid M(p, q) such that $0 \leq p \leq w$ and $0 \leq q \leq h$. A task requesting a submesh is denoted by T(p,q). A submesh is identified by its lower left (base) and top right (end) nodes and is denoted by S(base, end).

Definition 2: A busy submesh β is one in which all the processors are currently allocated to a task. The busy set B is the union of all busy submeshes.

Fig. 1 depicts a mesh M(8,8) with four busy submeshes β_1 = (<1,2>, <4,4>), β_2 = (<5,5>, <6,6>) ,β_3 = (<0,5>, <1,6>) and β_4 = (<2,7>, <4,7>).

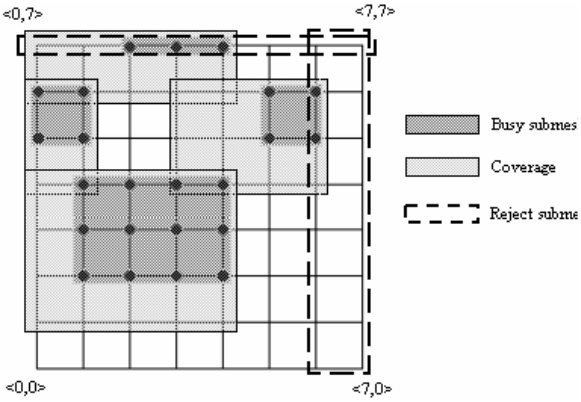

Fig. 1. An example of coverage and reject submeshes for T(3,2).

Definition 3: The coverage of a busy submesh β with respect to a task T, denoted by $\xi_{\beta,T}$, is a set of processors such that the use of any node in $\xi_{\beta,T}$ as the base of a free submesh for the allocation of T will make the task T overlap with β. The coverage set with respect to T, denoted as Ξ_T, is the set of the coverage of all busy submeshes. The symbols Ξ and C are sometimes used interchangeably to denote a coverage area or set. In Fig. 1, for example, with T(3,2), $\xi_{\beta_1,T}$ =(<0,1>, <4,4>), $\xi_{\beta_2,T}$ =(<3,4>, <6,6>), $\xi_{\beta_3,T}$ =(<0,4>, <1,6>) and $\xi_{\beta_4,T}$ =(<0,7>, <4,7>).

Definition 4: The reject submesh or area with respect to task T is a set of nodes Δ_T such that the use of any node in this set would make the task T cross the boundary of the mesh. For each T, there exist two reject submeshes – one in the horizontal direction (δ_H) and the other vertical (δ_V). In Fig. 1, for example, δ_H=(<0,7>,<7,7>) and δ_V=(<6,0>,<7,7>).

Definition 5: Internal fragmentation is the ratio of the number of overallocated processors to that of actually required number of processors. External fragmentation is the ratio of the number of free processors to the total number of processors in the mesh, when the allocation fails for the incoming task, even with a sufficient number of free processors.

Definition 6: The optimality index of an allocation is a measure of the adjacency of the allocation, i.e. how good a fit it is. It is the total number of nodes of other busy submeshes or the boundary of the mesh that the allocation will lie adjacent to. Fig. 2 indicates an allocation with optimality index 6 – it is adjacent to four nodes of other busy submeshes and has two nodes on the mesh boundary.

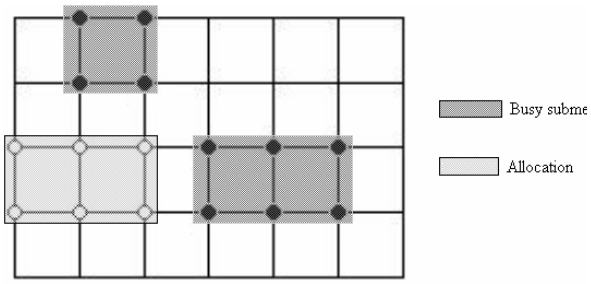

Fig. 2. An allocation with optimality index 6.

III. Existing Allocation Schemes

A. Two dimensional buddy scheme

The two-dimensional buddy (2DB) scheme was proposed in [2]. It requires the overall mesh, as well as the incoming request, to be a square with each side being a power of two. A non-square request of size pxq is rounded up, and the scheme allocates a 2^i x 2^i submesh where i = $\log_2(\max(p,q))$. The scheme thus has the problem of overallocation (internal fragmentation), and is not submesh recognition complete.

B. Frame Sliding

To overcome the limitations of the 2DB scheme, the frame sliding technique was proposed [3]. The scheme is based on the fact that, for any T, none of the nodes inside Ξ_T or Δ_T can serve as the base node of a free submesh accommodating T. For an incoming request T(p,q), a frame of size p x q is considered. The sliding starts from the lowest leftmost available node. When nodes in the currently examined frame are not all available, the frame is slid over the plane of the mesh to the next candidate frame by taking horizontal and vertical strides equal to the breadth and height and of the frame respectively. The sliding continues until a free submesh is allocated or there are no more candidate frames to examine. The scheme eliminates the problem of internal fragmentation,

but does not have complete submesh recognition because of the fixed strides. External fragmentation can also be very high.

C. First Fit (FF) /Best Fit (BF) approach

These were proposed in [4] to improve the FS strategy. The FF strategy is applicable to any mesh. FF maintains a busy array, which is a bit map representation of the allocation status of the mesh where the element [i, j] has value 1(0) if the node <i, j> is (is not) in Ξ_T . The FF algorithm scans all busy arrays from left to right and top to bottom and returns the first available node that is not part of the coverage set. The BF strategy selects a base in such a way that it has maximum number of busy neighbors. Experimental analysis shows that FF does better than BF.

The FF scheme is simple and efficient and can provide relatively good performance, but is not recognition complete as it considers only a fixed orientation of the submesh request. In addition, the high runtime overhead due to the manipulation of the arrays renders the algorithm unattractive, especially when the mesh system is large or a fast submesh allocation is required.

D. Adaptive Scan (AS) strategy

The adaptive scan technique as proposed in [5] is very similar to the FS scheme, but the AS scheme uses scanning instead of the sliding operation. That is, it moves a frame with a stride distance of 1. It also considers all possible orientations of each submesh request to achieve recognition completeness. However, the shorter stride distance increases search overhead and hence AS is not suitable for large meshes.

E. Adjacency strategy

For allocation, the Adjacency Strategy (ADJ), a list based algorithm, considers only submeshes residing on the periphery of busy submeshes, as well as any free submeshes at the corners of the mesh system. A free submesh that has the least number of adjacent busy nodes is selected in order to reduce external fragmentation. This strategy, while being quite fast, is hard to use for higher-degree meshes, since the number of nodes the lists need to maintain increases exponentially with the dimensional increase.

F. Free List (FL)

The key idea of this scheme proposed in [7] is to maintain a list of free submeshes in the system called a free list. To allocate a job, the list is searched to find a free submesh that is large enough to accommodate the job. The allocation is simple and fairly fast. However, updating the free list when a submesh is released is extremely complicated. In addition, this algorithm is not recognition complete.

G. Quick Allocation (QA)

The basic idea of the QA scheme is similar to that of the FF strategy. To reduce the search overhead, the scheme uses a last_covered array for each row in the mesh, indicating the last busy node on that row. With this array, it is not necessary to check the state of each node in the mesh since the array indicates whether a row has a potential base for a free submesh This overhead of this scheme increases in proportion to the number of rows in the mesh. The construction of the row-wise arrays can be a performance bottleneck.

H. Stack based algorithm (SBA)

This algorithm maintained a list of allocated submeshes. Using this list, the size of the requested submesh and the architectural limitations, the algorithm determines all nodes that cannot be assigned to the job. Then, these nodes are spatially subtracted from the entire mesh system to find a free submesh for the job. The algorithm uses a stack to speed up the search process. Recognition-completeness is achieved by considering all orientations of the requested submesh.

IV. Minimal Fragmentation Algorithm

A. The main approach

The processor allocation algorithm proposed in this paper has complete submesh recognition. The proposed algorithm is particularly attractive for large meshes. Like earlier recognition complete allocation algorithms, the proposed scheme achieves recognition completeness by manipulating the orientation of the submesh request. In allocating a task T (p,q), the scheme first tries to allocate the task using the given orientation (pxq). If this allocation fails, the algorithm creates a new request T(q,p) by rotating the original orientation and tries to allocate the new request. If this attempt also fails, the allocation of the job fails.

The basic idea of the proposed allocation technique is to find a node that, if chosen as the base node for allocating the requested task, would result in minimal external fragmentation. Initially all nodes are given a status of 0 (free). The algorithm first calculates the reject set Δ_T and the coverage set Ξ_T with respect to the task T and sets the status of these nodes to 1 (unavailable). The next step is to systematically scan the prospective base nodes surrounding each busy submesh. Once a prospective base node has been considered, its status is set to 2 (checked) to avoid checking it again if it is adjacent to another busy submesh.

For each busy submesh, the scheme considers the adjacent nodes along each of the four sides – right, top, left and bottom – as shown in Fig. 3. The nodes along the right and top are potential base nodes, and hence are directly under consideration. For a busy submesh (<a,b>, <c,d>), the nodes to be considered are given by

Right : <c+1, b-q+1> to <c+1, d>
Top : <c, d+1> to <a-p+1, d+1>

The nodes along the left and bottom of the busy submesh are potential end (upper right) nodes, and hence the scheme subtracts the task size dimensions from the coordinates of the nodes. The nodes to be considered are

Left: <a-p, d> to <a-p, b-q+1>
Bottom: <a-p+1, b-q> to <c, b-q>

The algorithm considers each of the sets of nodes – right, top, left, bottom – in order. For each node <x, y> under

consideration, the algorithm calculates the optimality index of an allocation at <x, y>. After each such check, the highest optimality index and the corresponding base node is retained. This process is repeated for all busy submeshes. If the optimality index of an allocation at a particular node is 2(p+q), the algorithm returns this node as the base node for task

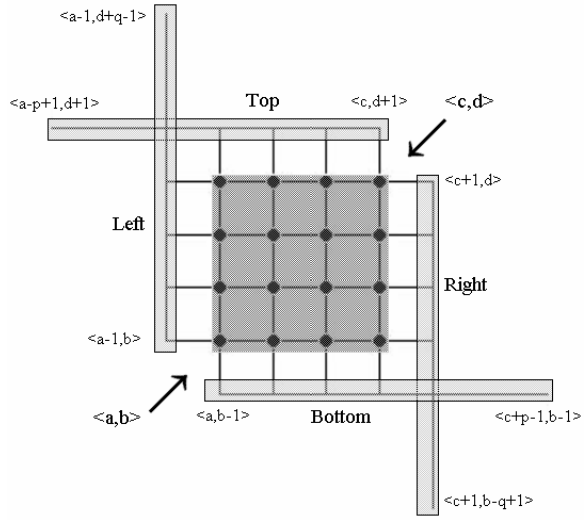

Fig. 3. Checking nodes adjacent to a busy submesh.

allocation. This step eliminates unnecessary calculations as there is no possible assignment that can be better i.e. yielding lesser fragmentation. The final node returned (the allocation at which has highest optimality index) is chosen as the base node for task allocation.

B. Algorithm

1) Notations

M(w,h) represents the mesh of width w and height h.

T(p, q) indicates a task of width p and height q.

β (<a, b>, <c, d>) indicates a busy submesh with lower left node <a, b> and upper right node <c, d>

2) Algorithm

Set status of all nodes to 0.

Construct the reject set Δ_T and the coverage set Ξ_T with respect to the task T(p, q).

reset status of all nodes in reject and coverage set to 2.

max_optimality_index ← p + q

best_node_so_far ← <0, 0>

For each busy submesh β (<a,b> ,<c,d>) in the mesh do,

for every candidate node <x, y> obtained from the adjacent nodes along the right, top, left and bottom of β and belonging to M(w,h),

if (status of <x, y> is not 2) then

current_optimality_index=weight (<x, y>, p, q)

if(current_optimality_index=2(p+q)) then

return <x, y> as the best fit node

end if

if(current_optimality_index>max_optimality_index) then

max_optimality_index ← current_optimality_index

best_node_so_far ← <x, y>

end if

end if

end for

end for

if both orientations have been tried

if (best_node_so_far=<0,0>) and (<0,0> is in Δ_T or Ξ_T)

Allocation fails

else

return best_node_so_far

end if

else

repeat the algorithm with the alternate orientation

function weight(<x, y>, p, q)

/* computes optimality index for a particular sub mesh */

index_of_this_allocation ← 0

for every node <i, j> in the perimeter of the submesh of size (p, q) with base node <x, y>

if <i, j> belongs to an element in Δ_T or Ξ_T then

index_of_this_allocation = -1

set status of <x, y> to 2

return index_of_this_allocation

end if

if (i = 0 or w – 1) then

increment index_of_this_allocation by 1

end if

if (j = 0 or h – 1) then

increment index_of_this_allocation by 1

end if

if <i, j> is adjacent to *n* busy nodes then

increment index_of_this_allocation by *n*

end if

end for

set status of <x, y> to 2

return index_of_this_allocation

C. Examples

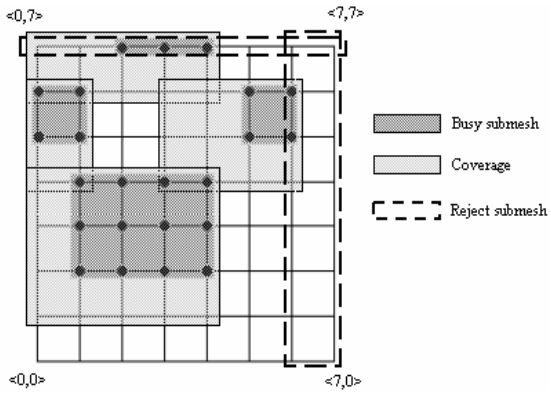

Fig. 4. An example of allocation for T(3,2).

1) Example 1

Consider the example for a task T(3,2) given in Fig. 1. Fig. 1 is reproduced as Fig. 4.

The first busy submesh is given by β_1 = (<1,2>, <4,4>). The nodes to be considered for adjacency are

Right: <5, 1> to <5, 4> Top : <4, 5> to <0, 5>
Left : no nodes Bottom: <0, 0> to <4, 0>

TABLE I
EXAMPLE 1, SUBMESH β_1

R/T/ L/B	Base	Optimality index	Boundary or adjacency
R	<5,1>	3	<7,1> <7,2> <4,2>
R	<5,2>	4	<7,2> <7,3> <4,2> <4,3>
R	<5,3>	4	<7,3> <7,4> <4,3> <4,4>
R	<5,4>	-1	-not applicable-
T	<4,5>	-1	-not applicable-
T	<3,5>	-1	-not applicable-
T	<2,5>	10	<1,5> <1,6> <2,4> <3,4> <4,4> <5,5> <5,6> <2,7><3,7> <4,7>

The nodes along the boundary of the submesh are considered in order as shown in Table I. The table indicates which direction (right R, top T, left L or bottom B) is under consideration, which node is taken as potential base, the optimality index of the allocation and the nodes contributing to the optimality index of that allocation.

In this case, we come across a node such that an allocation at it gives an optimality index of 10, which is the maximum possible. Hence, the algorithm returns the node <2,5> as the base node for the task allocation.

2) Example 1

Consider the example for a task T(3,2) given in Fig. 5.

The first busy submesh is given by β_1 = (<0,0>, <3,2>). The nodes to be considered for adjacency are

Right: <4, 0> to <4, 2> Top : <3, 3> to <0, 3>
Left : no nodes Bottom: no nodes

The nodes along the boundary of the submesh are considered in order as shown in Table II. At this point, node <0,3> with an optimality index of 7 is retained as the best node. The next busy submesh is given by β_2 = (<5,5>, <6,6>).

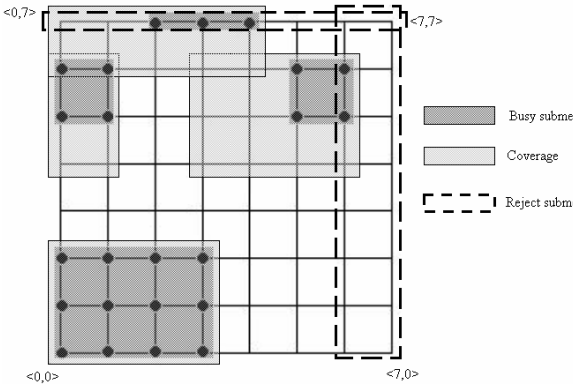

Fig 5. Another example of allocation for T(3,2)

TABLE II
EXAMPLE 2, SUBMESH β_1

R/T/ L/B	Base	Optimality index	Boundary or adjacency
R	<4,0>	5	<3,0> <3,1> <4,0> <5,0> <6,0>
R	<4,1>	2	<3,1> <3,2>
R	<4,2>	1	<3,2>
T	<3,3>	2	<3,2> <5,5>
T	<2,3>	2	<3,1> <3,2>
T	<1,3>	4	<1,5> <1,2> <2,2> <3,2>
T	<0,3>	7	<1,5> <1,6> <2,4> <3,4> <4,4> <5,5> <5,6> <2,7> <3,7> <4,7>

The nodes to be considered for adjacency are

Right: <7,4> to <7,6> Top : <6,7> to <3,7>
Left : <2,6> to <2,4> Bottom: <3,3> to <5,3>

The nodes along the boundary of the submesh are considered in order as shown in Table III.

TABLE III
EXAMPLE 2, SUBMESH β_2

R/T/ L/B	Base	Optimality index	Boundary or adjacency
R	<7,4>	-1	-not applicable-
R	<7,5>	-1	-not applicable-
R	<7,6>	-1	-not applicable-
T	<6,7>	-1	-not applicable-
T	<5,7>	-1	-not applicable-
T	<4,7>	-1	-not applicable-
T	<3,7>	-1	-not applicable-
L	<2,6>	-1	-not applicable-
L	<2,5>	7	<2,7> <3,7> <4,7> <1,6> <1,5> <5,5> <5,6>
L	<2,4>	2	<1,5> <5,5>
B	<3,3>	-	Computed earlier
B	<4,3>	2	<5,5> <6,5>
B	<5,3>	4	<5,5> <6,5> <7,4> <7,3>
B	<6,3>	-1	

Node <0,3> is still the best base node found so far. Although <2,5> also has an optimality index of 7, <0,3> is retained since it was encountered first.

The third busy submesh is given by β_3 = (<0,5>, <1,6>).

The nodes to be considered for adjacency are: (Table IV)

Right: <2,4> to <2,6> Top : <1,7> to <0,7>

TABLE IV
EXAMPLE 2, SUBMESH β_3

R/T/ L/B	Base	Optimality index	Boundary or adjacency
R	<2,4>	-	Computed earlier
R	<2,5>	-	Computed earlier
R	<2,6>	-1	
T	<1,7>	-1	
T	<0,7>	-1	
B	<0,3>	-	Computed earlier
B	<1,3>	-	Computed earlier

Left : no nodes Bottom: <0,3> to <1,3>

Node <0,3> is still the most optimal base node.

The last busy submesh is given by β_4 = (<2,7>, <4,7>).

The nodes to be considered for adjacency are

Right: <5,6> to <5,7> Top : no nodes

Left : no nodes Bottom: <0,5> to <4,5>

The nodes along the boundary of the submesh are considered in order as shown in Table V.

TABLE V

EXAMPLE 2, SUBMESH β_4

R/T/ L/B	Base	Optimality index	Boundary or adjacency
R	<5,6>	-	Computed earlier
R	<5,7>	-	Computed earlier
R	<0,5>	-1	
B	<1,5>	-1	
B	<2,5>	-1	
B	<3,5>	-	Computed earlier
B	<4,5>	-	Computed earlier

After considering all busy submeshes as above, the algorithm identifies node <0,3> to be the best base node for allocating the task T(3,2).

V. ANALYSIS

A. Comparative study of existing approaches

A comparative study of the existing approaches documents allocation and deallocation complexities, internal and external fragmentation, and complete submesh recognition capabilities of the approaches.

TABLE VI

COMPARISON OF DIFFERENT TASK ALLOCATION APPROACHES

Allocation scheme	Allocation complexity	Deallocation complexity	Internal/External Fragmentation
2DB	O(log √N)	O(N)	Yes/Yes
FS	O(NB)	Θ(1)	No/Yes
FF/BF	O(N)	O(N)	No/No
AS	O(NB)	Θ(1)	No/Yes
ADJ	$O(B^3)$	Θ(1)	No/Yes
FL	$O(F^2)$	$O(F^2)$	No/Yes
QA	O(hB)	Θ(1)	No/Yes
SBA	$O(B^2)$	Θ(1)	No/Yes

Table VI represents the time complexities of some of the approaches considered earlier. In our analyses we shall use the following notations:

B : Length of busy list

F : Length of Free list

h : Number of rows in mesh

w: Number of columns in mesh

Table VI depicts the parallels drawn between various approaches. It is evident that most of these approaches do not consider external fragmentation as a factor. Further, another common characteristic visible is the exchange of time complexity for complete submesh recognition.

Apart from considering internal and external fragmentation the MFA gives complete submesh recognition significant importance, as portrayed in Section IV.

B. Time complexity

The algorithmic time complexity of the algorithm is calculated based on the flow of the algorithm proposed in Section IV. We use the following notations in addition to existing notations for calculating the algorithmic time complexity of the MFA.

B: Number of busy submeshes

N: Number of nodes in mesh (N = w x h)

K: Average number of nodes in a submesh

1) Calculation of Time Complexity

Calculation of the reject set Δ_T

This calculation takes constant time and is independent of the parameters B and N, since it is a raw calculation (subtraction and addition) from the dimensions of the mesh and incoming task. Hence, let us take time taken as a constant ‘a’.

Calculation of the coverage set Ξ_T

This calculation takes time proportional to the number of busy submeshes, since each busy submesh requires a coverage calculation done in constant time ‘b’. Hence, let the time taken for this calculation be bB.

Setting up variables

Setting up the variables in the algorithm such as max_optimality_index, best_node_so_far takes constant time, and hence, we have an additional constant ‘c’ in the complexity calculation.

Calculation of minimal fragmentation seed nodes:

The time complexity of the core of the algorithm can be calculated as follows. For each busy submesh from which adjacent nodes need to be checked, we require a constant time, let us take as‘t’ per node. Since the average size of each busy submesh is K, the maximum number of nodes possible along the fringes of a busy submesh is 4 t √K. The calculation for adjacent nodes requires that this calculation be performed for each busy submesh. Since there are B busy submeshes, we have the time complexity for this calculation to be 4 t √K. Further, condition checking and updating variables requires a

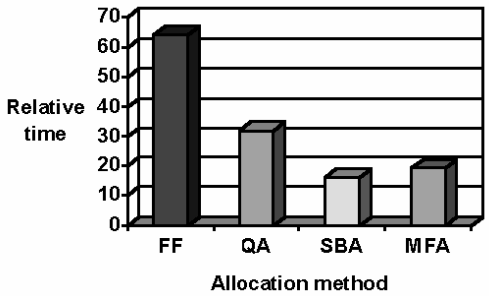

Fig. 6. Running time of some approaches for Examples 1 and 2

constant time per busy submesh, say u. Hence, the calculation

per busy submesh takes 4 t √K + u. Thus, we have effective running time calculation for the core of the algorithm as (4 t √K + u) B. MFA therefore runs in an average time of

$$a + bB + c + (4\ t\ \sqrt{K} + u)\ B$$

The asymptotic complexity of the MFA is therefore in the order of Θ (√K B).

Fig. 7 provides a summary review of the complexity comparisons among different algorithms including MFA. We find as the mesh size and allocated number of meshes increase, the running time of MFA falls below that of the SBA as well. The parameters used in this assessment are h=8 and K=5.75.

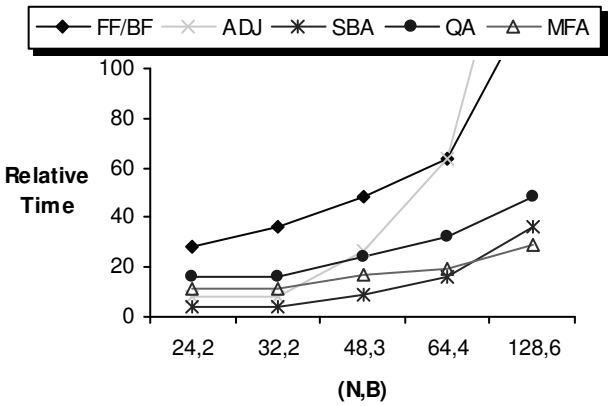

Fig. 7 Algorithmic Running times

2) Comparing Optimality Indices

We compare the optimality indices (as prescribed in Section III) from the allocation of incoming tasks of dimensions (3,2) , (3,3) and (4,2) respectively. For our examples 1 and 2, the optimality is measured using the optimality index specified in

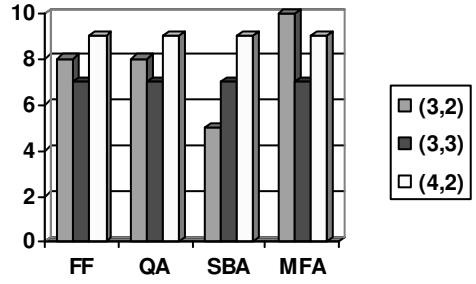

Fig. 8. Optimality Index for Example 1 with different approaches for different task sizes.

Section IV(A), shown here in Fig.8 and Fig. 9 respectively.

This comparison reveals that the MFA provides an allocation such that external fragmentation is lower than the other approaches.

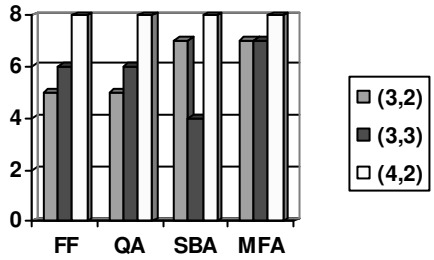

Fig. 9. Optimality Index for Example 2 with different approaches for different task sizes.

VI. FUTURE WORK

We plan to extend MFA by exploiting parallelism in the optimality index calculations to improve the algorithmic time complexity of the algorithm. Parallelization of the MFA algorithm is currently underway and testing should be complete in a short time.

VII. CONCLUSION

The development of efficient processor allocation algorithms for different multicomputers has been an active area of research. In this paper, we have proposed the Minimal Fragmentation Algorithm (MFA) for task allocation in mesh-connected multicomputers. The proposed algorithm is particularly attractive for large meshes. The MFA identifies nodes that, if chosen as the base for task allocation, would result in minimal external fragmentation. This is clearly established by simulations and comparative studies. Further, the asymptotic time complexity of MFA is quite competitive and complete submesh recognition is achieved. For task allocation in a large mesh system requiring minimum fragmentation in a relatively small time, our approach provides an ideal solution.

REFERENCES

[1] Alverson, et al. (1990): "The Tera Computer System," *Proc. 1990 Int'l Conf. Supercomputing*

[2] K. Li and K.H. Cheng (1991): "A Two-Dimensional Buddy System for Dynamic Resource Allocation in a Partitionable Mesh Connected System," *Journal of Parallel and Distributed Computing,* vol. 12, pp. 79-83

[3] P. Chuang, and N. Tseng (1991): "An Efficient Submesh Allocation Strategy for Mesh Computer Systems," *Proc. 1991 Int'l Conf. Distributed Computer Systems,* pp. 256-263

[4] Y. Zhu (1992): "Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers," *J. Parallel and Distributed Computing,* vol. 16, pp. 328-337

[5] J. Ding and L.N. Bhuyan (1993): "An Adaptive Submesh Allocation Strategy for Two-Dimensional Mesh Connected Systems," *Proc. International Conference on Parallel Processing,* vol. II, pp. 193-200

[6] D. Das Sharma and D.K. Pradhan (1993): "A fast and efficient strategy for submesh allocation in mesh-connected parallel computers," *IEEE Symposium on Parallel and Distributed Processing,* pp. 682-689

[7] T. Liu, W. Huang, F. Lombardi and L.N. Bhuyan (1995): "A Submesh Allocation Scheme for Mesh Connected Multiprocessor Systems," *Proc. International Conference on Parallel Processing*, vol. II, pp. 193-200

[8] S. Yoo, H.Y. Youn and B. Shirazi (1997): "An Efficient Task Allocation Scheme for 2D Mesh Architectures," *IEEE Transactions on Parallel and Distributed Systems,* vol. 8, no. 9, pp. 934-938

[9] B.S. Yoo and Chita R. Das (2002): "A Fast and Efficient Allocation Scheme for Mesh-Connected Multicomputers," *IEEE Transactions on Computers*, vol. 51, no. 9, pp. 46-55