A Dynamic Processor Allocation Strategy for Mesh-Connected Multicomputers

Geunmo Kim and Hyunsoo Yoon

Abstract

The processor allocation problem in mesh multicomputers is to recognize and locate a free submesh that can accommodate a request for a submesh of a specified size. An efficient submesh allocation strategy is required for achieving high performance on mesh multicomputers. In this paper, we propose a new best-fit submesh allocation strategy for mesh multicomputers. The proposed strategy maintains and uses a free submesh list to get global information for free submeshes. For an allocation request, the proposed strategy tries to allocate a best-fit submesh which causes the least amount of potential processor fragmentation so as to preserve the large free submeshes as many as possible for later requests. For this purpose, we introduce a novel function for quantifying the degree of potential fragmentation of submeshes. The proposed strategy has the complete submesh recognition capability. Extensive simulation is carried out to compare the proposed strategy with the previous strategies and experimental results indicate that the proposed strategy exhibits the best performance along with about 10% to 30% average improvement over the best previous strategy.

I. Introduction

Multicomputer architectures have been expected as the most promising way to construct massively parallel computers based on the interconnection of hundreds or thousands of microprocessors. Recently, the mesh has been drawing considerable attention as a topology of multicomputer due to its simple, regular, and scalable structure. Based on this topology, several prototypes and commercial systems have been built or marketed, such as Intel Touchstone Delta [1], Intel Paragon XP/S [2], Tera Computer System [3], Fujitsu AP-100 [4], Sanyo Edden/Cyberflow System [4], Parsytec GC [5], and PASM [6].

A mesh multicomputer can support multitasking environment efficiently due to its partitionable structure. For instance, the Intel Touchstone Delta [1] and Paragon XP/S [2] support a multitasking environment, and the PASM [6] provides a dynamic reconfiguration to operate one or more independent submachines of various sizes. In a multitasking environment, numerous tasks, each of which consists of a number of parallel modules, can be assigned to independent submeshes and executed simultaneously. Since tasks request the submeshes of various sizes, efficient submesh allocation, in order to make a mesh system to accommodate as many tasks as possible, is an important issue for achieving high performance on a mesh multicomputer.

The goal of submesh allocation in a mesh system is to maximize system utilization by minimizing *fragmentation* — *internal* and *external* fragmentation. Internal fragmentation occurs if a larger submesh is allocated to a task than that is required. External fragmentation occurs when even if a sufficient number of processors are available, they do not form a submesh large enough to accommodate the incoming request because they are scattered. Such fragmentation results in the degradation of system utilization.

In the last few years, several submesh allocation strategies have been reported in the literature [7,8,9,10,11,12]. Most of them can successfully eliminate internal fragmentation by allocating the submesh of the exact size requested. These strategies, however, quite suffer from external fragmentation, resulted from the characteristics of their allocation policies — incomplete (submesh) recognition capability and/or first-fit behavior. To solve this problem, a best-fit strategy with complete recognition capability is proposed recently [2], and this strategy shows better performance than all the previous strategies. This strategy, however, has the drawback that it is only applicable to somewhat specific situations owing to the limitation of its heuristic.

In this paper, we propose a new best-fit submesh

Manucript received July 31, 1995; accepted October 6, 1995.

The authors are with Center for Artificial Intelligence Research, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea.

allocation strategy, called *Free Submesh List(FSL)* strategy. The proposed strategy maintains and uses a free submesh list keeping all of the different free submeshes available in the system to get global information for free submeshes. The basic idea of our approach is to allocate a submesh which can preserve the large free submeshes as many as possible and to maintain the high availability of the wide range of submeshes, for later tasks. This approach would be able to reduce the external fragmentation, especially prevent the 'unnecessary' external fragmentation occurring by an inefficient allocation. Simulation study demonstrates that the proposed strategy outperforms the previous strategies, under meshes of various sizes and various loads. The experimental results indicate that the proposed strategy achieves about 30% and 10% average improvements over the best previous strategy under First Come First Served (FCFS) and Modified-FCFS job scheduling disciplines, respectively. While the proposed strategy can be easily extended to the meshes of higher dimensions, our discussion here is limited to the two-dimensional meshes, for simplicity.

The rest of this paper is organized as follows. Section 2 presents basic notations and briefly reviews the previous work on submesh allocation. Section 3 introduces the proposed allocation strategy, and provides the comparison with other strategies. Presented and discussed in Section 4 are the simulation results of the allocation strategies measured under various load distributions. Conclusion is given in Section 5.

II. Preliminaries and Related Work

A two dimensional mesh $M(L_X, L_Y)$ is an $L_X \times L_Y$ rectangular grid which consists of $L_X \cdot L_Y$ nodes. A *node* in a mesh system refers to a processor. So the terms *node* and *processor* are used interchangeably in this paper. A node in a mesh is identified by its coordinate $\langle x, y \rangle$ from the lower-leftmost position $\langle 0, 0 \rangle$ of the mesh, where $0 \le x \le L_X-1$ and $0 \le y \le L_Y-1$. A node $\langle x, y \rangle$ except the boundary node is connected to four adjacent nodes $\langle x-1, y \rangle$, $\langle x+1, y \rangle$, $\langle x, y-1 \rangle$, and $\langle x, y+1 \rangle$. Each boundary node has two or three of the above adjacent nodes depending on their positions. Fig. 1 presents a 7×5 mesh with all the addresses of its nodes.

A *submesh* $S(l_x, l_y)$ in the mesh $M(L_X, L_Y)$, $1 \le l_x \le L_X$ and $1 \le l_y \le L_Y$, is an $l_x \times l_y$ rectangular grid which belongs to $M(L_X, L_Y)$ with $l_x \cdot l_y$ nodes. The address of a submesh is denoted by a quadruple $(\langle x, y \rangle, \langle x', y' \rangle)$, where $\langle x, y \rangle$ indicates its lower-leftmost coordinate and $\langle x', y' \rangle$ its upper-rightmost one. For convenience' sake, we will use the terms both $S(l_x, l_y)$ and $S(\langle x, y \rangle, \langle x', y' \rangle)$ to denote a submesh. A *free submesh* is a submesh in which all the processors are currently free, and an *allocated submesh* is a submesh in which all the processors are currently allocated to a task. As an example, a free submesh $S(\langle 0, 2 \rangle, \langle 3, 4 \rangle)$ and an allocated submesh $S'(\langle 4, 0 \rangle, \langle 6, 3 \rangle)$ in Fig. 1 denote submeshes of 4×3 and 3×4, respectively. We specify an incoming request (task) for a rectangular submesh of size $a \times b$ or $b \times a$ by (a, b).

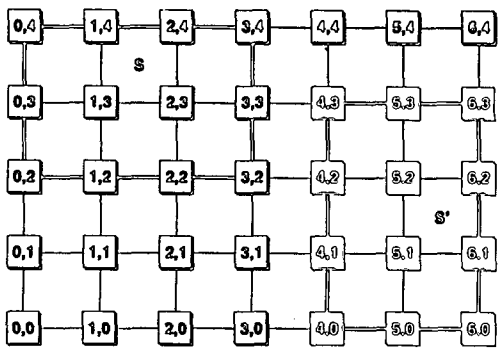


Fig. 1. Example of a 7×5 mesh $M(7, 5)$.

The submesh allocation problem is to recognize and locate a free submesh accommodating a request for a submesh of an arbitrary size. An allocation strategy is said to be *complete recognition* if it can recognize and allocate all the possible submeshes of any size. In the last few years, several submesh allocation strategies have been reported in the literature [7,8,9,10,11,12], with/without complete recognition capability. These strategies are classified into two approaches, *first-fit* and *best-fit*. In the first-fit approach, submeshes are searched in a fixed sequence and the first free submesh is allocated. On the other hand, the best-fit approach is intended to allocate an appropriate submesh by considering the contributions of submeshes to the external fragmentation for the mesh system. In the hypercube, due to the symmetric property of topology, the processor (subcube) allocation strategies show the fairly similar performance regardless of their allocation approach, first-fit or best-fit with/without complete recognition capability [13]. However, in the submesh allocation, as observed in the previous works, the performances of allocation strategies are subject to their recognition capabilities and allocation properties.

In what follows, we briefly review previous allocation strategies. The details of each strategy can be found in the references [7,8,9,10,11,12].

1. Two-Dimensional Buddy Strategy

The Buddy strategy [7,8] was proposed for a square mesh with side lengths being exactly powers of 2. This strategy can recognize and allocate the specific square submeshes whose side length is a power of 2. For a request of arbitrary side lengths, it allocates the smallest square submesh accommodating the request, and hence serious internal

fragmentation can be involved owing to over-allocated nodes. Thus, this strategy cannot be incorporated in general meshes with arbitrary side lengths.

2. Frame Sliding (FS) Strategy

To solve the internal fragmentation of the Buddy strategy and to improve flexibility in identifying free submeshes, the FS strategy was proposed in [9]. This strategy allocates a submesh of the size identical to a request and can be applied to meshes with arbitrary side lengths. This strategy, however, still cannot completely recognize all the submeshes because of its fixed orientation along with horizontal and vertical strides. That is, for a request (a, b), it searches a first free submesh of $a \times b$ but not $b \times a$ with horizontal stride a and vertical stride b from the first (lower-leftmost) free processor.

3. First-Fit (FF) and Best-Fit (BF) Strategies

To solve the problem associated with fixed strides in the FS strategy, two strategies were proposed in [10]. For a request (a, b), the FF strategy allocates the first free submesh of $a \times b$ without fixed strides. Hence, this strategy recognizes all available $a \times b$ submeshes. However, the FF strategy still has the problem of fixed orientation, that is, this strategy can not recognize $b \times a$ submeshes like the FS strategy. On the other hand, the BF strategy tries to allocate a submesh in the smallest region of free processors, reserving the large regions for later tasks. For the best-fit submesh, the BF strategy selects a submesh whose *base node*, the lower-leftmost node of the submesh, has the largest number of busy (allocated) neighbors (a boundary node of mesh is considered as having a busy neighbor beyond that boundary). Like the FF strategy, the BF strategy does not have complete recognition capability. Moreover, the BF strategy did not show clear performance improvement over the FF strategy on simulation.

4. Adaptive Scan (AS) Strategy

The AS strategy [11] is the first one providing the complete recognition capability. For an allocation request (a, b), this strategy first finds an $a \times b$ free submesh from the coordinate $\langle 0, 0 \rangle$ to $\langle L_X-1, L_Y-1 \rangle$ by increasing x-coordinate prior to y-coordinate. If an $a \times b$ free submesh does not exist, the strategy tries to find a $b \times a$ free submesh by the same way as the above. Although this strategy has the complete recognition capability, it may suffer from the significant external fragmentation because of its first-fit behavior.

5. Busy-List Strategy

The Busy-List strategy [12] is the first best-fit strategy with the complete recognition capability. This strategy allocates a submesh with the largest *boundary value* either adjacent to allocated submeshes or on one of the four corners of mesh to reduce the external fragmentation occurred in the first-fit approach. The boundary value of a submesh is the sum of the number of busy neighbors of its nodes, where a boundary node of mesh is considered as having a busy neighbor beyond that boundary. For example, the boundary value of a submesh S shown in Fig. 1 is 9 since the node $\langle 0, 4 \rangle$ has two busy neighbors beyond upper and left boundaries, each of nodes $\langle 0, 2 \rangle$, $\langle 0, 3 \rangle$, $\langle 1, 4 \rangle$, $\langle 2, 4 \rangle$, and $\langle 3, 4 \rangle$ has one busy neighbor beyond the associated boundary, and each of nodes $\langle 3, 2 \rangle$ and $\langle 3, 3 \rangle$ has one busy neighbor which belongs to the allocated submesh S'. For an allocation request, this strategy generates candidate submeshes on the boundary of allocated submeshes and the corners of the mesh, if possible. Then, this strategy evaluates the boundary values of the candidate submeshes and allocates one which has the largest boundary value. This heuristic is similar to that of the BF strategy which considers only the boundary value of base node. Although this strategy provides the better performance than the other strategies, its heuristic using boundary value is only applicable to somewhat specific situations because of its own limitation of selecting best-fit submesh, and so this strategy does not fully resolve the unnecessary external fragmentation (described in Section 3).

III. The Proposed Strategy

In the following, we propose a new recognition-complete allocation strategy based on the *free submesh list*, termed the *FSL* strategy. The proposed strategy offers the best solution to date which can effectively reduce the external fragmentation. In the various environment, the proposed strategy exhibits the best performance and the least run-time overhead.

1. Free Submesh List

The free submesh list (FSL) consists of the *dominant free submeshes* covering all possible free submeshes. A free submesh is said to be *dominant* if it is not covered by (i.e., not a submesh of) any other free submeshes. Those dominant submeshes may not be disjoint each other, that is, they may share common regions (said to be *overlapping*).

Definition 1. The FSL is a sorted list of dominant free submeshes in non-increasing order of size, i.e., the number of nodes. If two free submeshes are of the same size, one of them closer to the square submesh is prior to the other.

For example, in a 10×10 mesh as shown in Fig. 2, there are three dominant free submeshes, $S_1(\langle 0, 0 \rangle, \langle 6, 4 \rangle)$, $S_2(\langle 4, 0 \rangle, \langle 6, 9 \rangle)$, and $S_3(\langle 4, 5 \rangle, \langle 9, 7 \rangle)$ of sizes $35(7 \times 5)$, $30(3 \times 10)$, and $18(6 \times 3)$, respectively. In this case, FSL is sorted in sequence S_1, S_2, and S_3, from the largest free submesh to the smallest one.

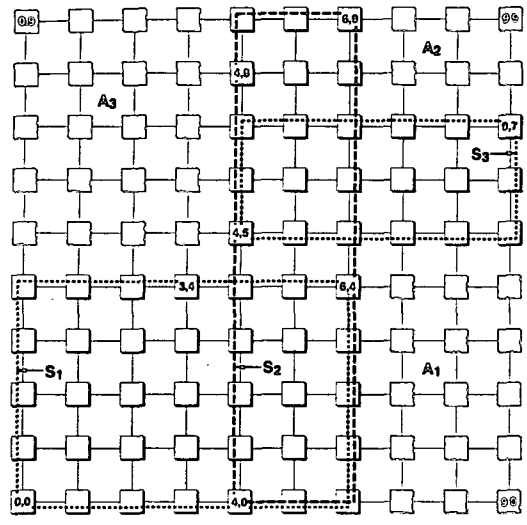


Fig. 2. Dominant Free Submeshes.

Definition 2. The reservation factor of a submesh $S(\langle x_1, y_1\rangle, \langle x_2, y_2\rangle)$ against a submesh $S'(\langle x_1', y_1'\rangle, \langle x_2', y_2'\rangle)$ is referred to $rf(S, S')$ and is defined as the size of the largest submesh formed from S' except the overlapping region as follows (see Fig. 3):

$$rf(S, S') = \begin{cases} \max\{(x_1 - x_1') \cdot l_y', (x_2' - x_2) \cdot l_y', l_x' \cdot (y_1 - y_1'), l_x' \cdot (y_2' - y_2)\}, & \text{if } S \text{ and } S' \text{ are overlapping,} \\ l_x' \cdot l_y' \ (= \text{size of } S'), & \text{otherwise.} \end{cases}$$

where $l_x' = x_2' - x_1' + 1$, $l_y' = y_2' - y_1' + 1$, and $\max\{*\}$ is a function that returns the largest non-negative value.

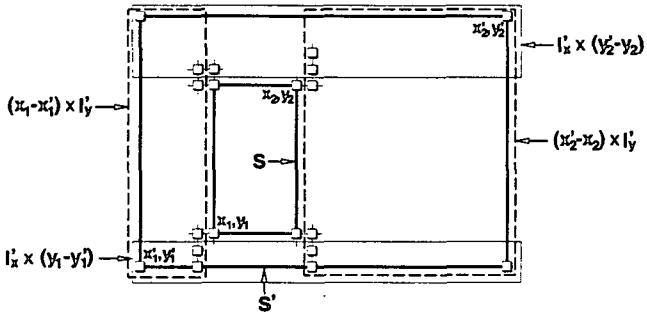


Fig. 3. Reservation Factor : Submeshes reserved by S.

In Fig. 2, the rf of the submesh $S_2(\langle 4, 0\rangle, \langle 6, 9\rangle)$ against the submesh $S_1(\langle 0, 0\rangle, \langle 6, 4\rangle)$, $rf(S_2, S_1)$, is 20 $(= \max\{4 \cdot 5, 0, 0, -5 \cdot 7\})$, since the largest submesh constructed from S_1 except the overlapping region $(\langle 4, 0\rangle, \langle 6, 4\rangle)$ is $(\langle 0, 0\rangle, \langle 3, 4\rangle)$ of size 4×5. Also, $rf(S_3, S_2) = 15$, i.e., the size of submesh $(\langle 4, 0\rangle, \langle 6, 4\rangle)$ which is larger than the submesh $(\langle 4, 8\rangle, \langle 6, 9\rangle)$.

Definition 3. For a request (a, b), a *candidate submesh* is a free submesh of size $a \times b$ or $b \times a$.

Definition 4. A candidate submesh is said to be *dominant* against a free submesh S if it has the largest rf against S than any other candidates.

2. Allocation Algorithm

For an allocation request (a, b), first, the proposed allocation algorithm generates candidates of the size either $a \times b$ or $b \times a$ from free submeshes in FSL. Consider a free submesh $S(\langle x_1, y_1\rangle, \langle x_2, y_2\rangle)$ in FSL accommodating the request. Candidate submeshes are generated from four corners of free submesh S, $\langle x_1, y_1\rangle$, $\langle x_1, y_2\rangle$, $\langle x_2, y_1\rangle$, and $\langle x_2, y_2\rangle$. As shown in Fig. 4, candidate submesh(es) of types $a \times b$ and/or $b \times a$ can be generated at each corner. When candidates of both types can be generated, one approach, we adopt, is to consider only the candidates of a type with the larger rf value against S, i.e., dominant candidates against S. Then, the algorithm evaluates the rf's of candidates of both types against S and chooses the candidates (of a type) which are dominant against S. Although many other candidates may be possible, considering only the dominant candidates is sufficient because, with regard to the submesh S, they have the largest rf value and reserve the largest submeshes than any other candidates. This approach is expected to cause less external fragmentation in the choice of candidate submeshes by preventing the large free submesh from being fragmented to several smaller submeshes. Fig. 5 presents all candidate submeshes for the request (3,2) on the case of Fig. 2, where C_k^* denote candidate submeshes originated from the free submesh S_k in Fig. 2.

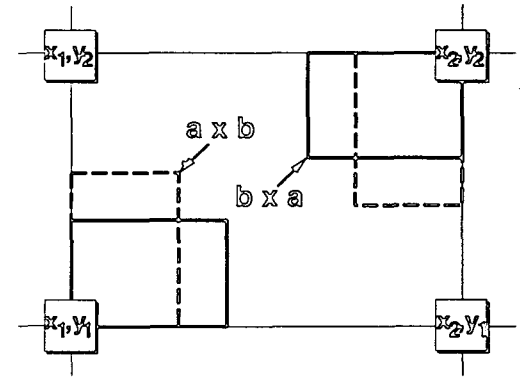


Fig. 4. Candidate submeshes from a free submesh for the request (a, b).

On the completion of generating candidate submeshes, the algorithm evaluates the rf of each candidate submesh against the first free submesh of FSL and then rejects candidate submeshes except those which have the largest rf value, and, evaluates candidate submeshes not rejected against the second free submesh of FSL and then rejects candidate submeshes except those which have the largest rf value, and, so on. The rf is used for evaluating the potential external fragmentation of candidate submeshes. The candidate submesh with large rf values can conserve larger free

submeshes than the one with small rf values. Since the free submeshes in FSL are sorted in non-increasing order of size, the above evaluation process using rf chooses the dominant candidate against all the free submeshes which can reserve large free submeshes as many as possible for later requests. This may prevent 'unnecessary' external fragmentation and alleviate the overall external fragmentation. For example, suppose in Fig. 5 that the two tasks $t_1=(3,2)$ and $t_2=(7,5)$ are submitted to the system in that order. If either one of C_1^* or C_2^1 is allocated to the task t_1, the subsequent task t_2 cannot be allocated due to the external fragmentation. This is an instance of 'unnecessary' external fragmentation arising because of an inefficient allocation. On the other hand, allocating one of C_2^2 or C_3^* to the task t_1 can accommodate the task t_2 in the submesh $(\langle 0,0\rangle,\langle 6,4\rangle)$. In this case, the proposed algorithm allocates C_3^2 to the task t_1, since C_3^2 has the largest rf values against all the free submeshes (see Example 1).

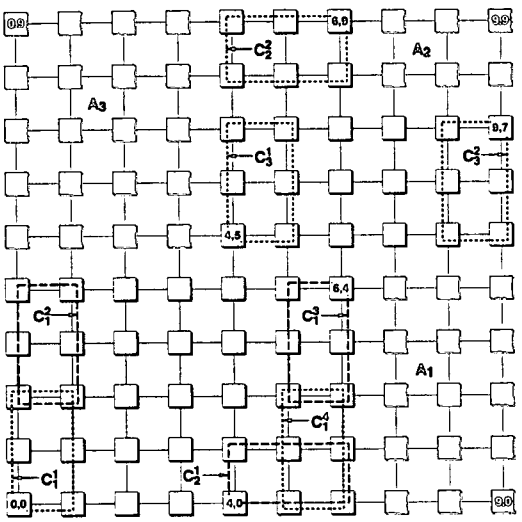


Fig. 5. Candidate Submeshes for the request (3,2).

After the evaluation for the candidates, if two or more candidates are remained, the algorithm performs the evaluation for the candidates against the entire mesh $(\langle 0,0\rangle,\langle L_X-1, L_Y-1\rangle)$. The rf value of a candidate against the entire mesh indicates the size of a potential free submesh reserved by the candidate. The algorithm chooses the dominant candidate against the entire mesh, so that the already allocated submeshes on being released may be merged each other or together with free submeshes into large free submesh(es). The formal allocation algorithm is presented in Fig. 6. The symbols used by the algorithm mean that: *cand_list* denotes the list of candidate submeshes being considered; *free_list* denotes FSL, the list of free submeshes; *alloc_list* denotes the list of allocated submeshes; *best* denotes the best candidate submesh.

```
Alloc( a, b ) /* Request (a, b) */
{
    /* Step 1 */
    cand_list = φ;
    for all free submesh S ∈ free_list do
        if (S can accommodate the request a × b or b × a)
            Generate candidate submeshes from the four corners of S and
            store them to cand_list;
    if (cand_list == φ)
        Stop and wait until a submesh is released;
    /* Step 2 */
    for all free submesh S ∈ free_list do {
        Evaluate the rf of each candidate submesh in cand_list against S;
        Reject candidate submeshes except those which have the largest rf value;
        if (There is only one candidate submesh in cand_list)
            Assign it to best and goto Step 3;
    }
    Select the candidate submesh which has the largest rf value against the mesh
    (< 0,0 >, < L_X − 1, L_Y − 1 >) and assign it to best;
    /* Step 3 */
    for all free submesh S ∈ free_list do
        if (S is overlapping with best) {
            Delete S from free_list;
            Generate new submeshes by decomposing S and
            insert them into free_list if they are dominant submeshes;
        }
    Allocate submesh best to the request and insert it into alloc_list;
}
```

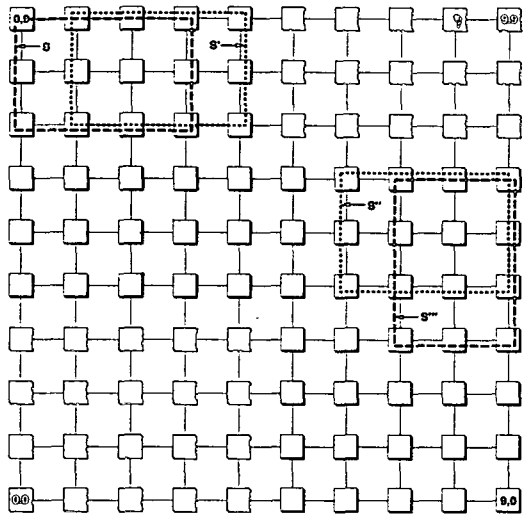
Fig. 6. Allocation Algorithm.

The following example illustrates allocation process for a request (3,2) in the situation of Fig. 2.

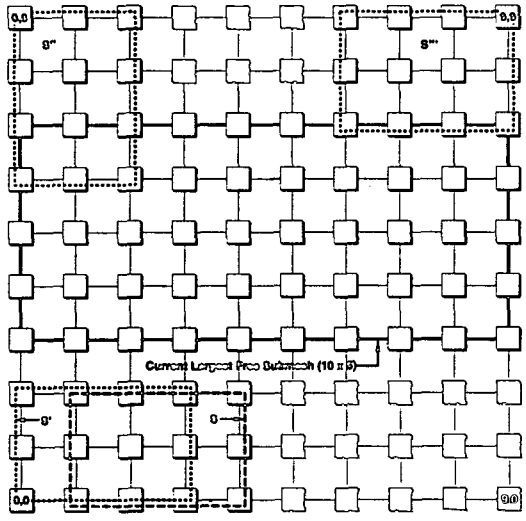
Example 1. Initially, FSL = $\{S_1(\langle 0,0\rangle,\langle 6,4\rangle), S_2(\langle 4,0\rangle,\langle 6,9\rangle), S_3(\langle 4,5\rangle,\langle 9,7\rangle)\}$. The algorithm, from free submeshes S_1, S_2, and S_3, generates candidate submeshes $\{C_1^1, C_1^2, C_1^3, C_1^4, C_2^1, C_2^2, C_3^1, C_3^2\}$ as shown in Fig. 5. Then, the algorithm evaluates the rf's of candidate submeshes against the first free submesh S_1: $rf(C_1, S_1) = rf(C_1^2, S_1) = 25$ (= size of $(\langle 2,0\rangle,\langle 6,4\rangle)$ of 5×5), $rf(C_1^3, S_1) = rf(C_1^4, S_1) = 25$ (= size of $(\langle 0,0\rangle,\langle 4,4\rangle)$ of 5×5), $rf(C_2^1, S_1) = 21$ (= size of $(\langle 0,2\rangle,\langle 6,4\rangle)$ of 7×3), and, $rf(C_2^2, S_1) = rf(C_3^1, S_1) = rf(C_3^2, S_1) = 35$ (= size of S_1). Excepting dominant candidates C_2^2 and C_3^* with the largest rf value of 35, the algorithm rejects all other candidates C_1^* and C_2^1. Again, the algorithm evaluates the rf's of candidates C_2^2 and C_3^* against the second free submesh S_2: $rf(C_2^2, S_2) = 24$ (= size of $(\langle 4,0\rangle,\langle 6,7\rangle)$ of 3×8), $rf(C_3^1, S_2) = 15$ (= size of $(\langle 4,0\rangle,\langle 6,4\rangle)$ of 3×5), and, $rf(C_3^2, S_2) = 30$ (= size of S_2). Then, the algorithm rejects C_2^2 and C_3^1. Now that only one candidate submesh C_3^2 remains (which is the best under the current situation), the algorithm does not perform evaluation for free submesh S_3. In step 3, because S_3 is overlapping with the best candidate submesh C_3^2, the algorithm decomposes S_3 into free region(s) and an allocated one, and generates a submesh $(\langle 4,5\rangle,\langle 7,7\rangle)$ (denoted as S'). Since S' is a dominant submesh which is not covered by existing free submeshes S_1 and S_2, S' is inserted into FSL. The final FSL is $\{S_1(\langle 0,0\rangle,\langle 6,4\rangle), S_2(\langle 4,0\rangle,\langle 6,9\rangle), S'(\langle 4,5\rangle,\langle 7,7\rangle)\}$.

The main advantage of the proposed strategy is that it can conserve the large free submeshes of the wide range of size

as many as possible after allocation, and thus, prevent available processors from being fragmented. Moreover, the proposed strategy can successfully resolve the external fragmentation occurred by allocating from a large free submesh although a smaller free submesh can accommodate a request. The previous strategies have the difficulty to resolve this type of fragmentation effectively, on account of the first-fit behaviors of the AS strategy and the un-robust heuristic of the Busy-List strategy which cannot discriminate between a candidate submesh located at a large free region and one at a small free region if they have the same boundary value. On the above example, the AS strategy allocates submesh either $(\langle 0,0\rangle,\langle 2,1\rangle)$ of 3×2 or C_1^1 of 2×3 in Fig. 5, since those are the first free ones of size 3×2 and 2×3, respectively.



(a)



(b)

Fig. 7. The best candidate submeshes (S and S') and the worst (S'' and S''').

On the other hand, the Busy-List strategy allocates one of submeshes C_2^2 and C_3^2 since these two submeshes have the largest boundary value of 7. The Busy-List strategy has two main disadvantages: (1) Many candidate submeshes have the same boundary values; (2) To choose the candidate submesh(es) with the largest boundary value may be the worst choice on a situation. For example, consider two situations for a 10×10 mesh shown in Fig. 7. For a request (4,3) on the situation of Fig. 7 (a), there exist 11 candidate submeshes having the same boundary value of 7 (but we present four submeshes among them). In this case, although both submeshes S (or S') and S'' (or S''') have the same boundary value, the submesh S (or S') is the best choice because it can reserve submeshes of 5×7 and 10×3, but the submesh S'' (or S''') is the worst choice. On the other hand, on the situation of Fig. 7 (b), only two submeshes S'' and S''' have the largest boundary value of 9. However, selecting one of them is the worst choice because it cannot conserve the largest free submesh of size 10×5 and may cause more external fragmentation than other candidates, S and S', with the smaller boundary value of 7. In the above two situations, our strategy allocates either S or S', which are the best choice since they conserve the largest free submesh after allocation. As another advantage of our strategy, our allocation algorithm can quickly determine whether a request can be accommodated by searching FSL, and hence, can avoid exhaustive search unlike most of the previous strategies basically using the frame-sliding method.

3. Deallocation Algorithm

Once a submesh is released after the corresponding task is completed, the deallocation algorithm starts from the initial mesh $(\langle 0,0\rangle,\langle L_X-1, L_Y-1\rangle)$ and generates a new set of dominant free submeshes by decomposing the initial mesh for the allocated submeshes. Fig. 8 describes the deallocation algorithm. For a released submesh R, first (step 1), if FSL (*free_list*) is empty, the released submesh is inserted into FSL; otherwise, if the allocation list (*alloc_list*) is empty, all the submeshes in FSL are removed and FSL is set to the initial list $(\langle 0,0\rangle,\langle L_X-1, L_Y-1\rangle)$ because all the nodes of mesh system are free. Otherwise, that is, if both FSL and the allocation list are not empty, FSL is set to the initial list $\{(\langle 0,0\rangle,\langle L_X-1, L_Y-1\rangle)\}$, and the algorithm decomposes submesh(es) in FSL for all the allocated submeshes in the allocation list with the same way as in step 3 of allocation algorithm. After completion of step 2, FSL is set to the dominant free submeshes including all the nodes of the released submesh R.

Example 2. Consider the situation of Fig. 2. Initially, FSL = $\{S_1, S_2, S_3\}$ and *alloc_list* = $\{A_1(\langle 7,0\rangle,\langle 9,4\rangle), A_2(\langle 7,8\rangle,\langle 9,9\rangle), A_3(\langle 0,5\rangle,\langle 3,9\rangle)\}$. When the allocated submesh A_3 is released, A_3 is removed from *alloc_list*, and *alloc_list* = $\{A_1, A_2\}$. In step 2, FSL is reset to the initial

state { $S(\langle 0,0\rangle,\langle 9,9\rangle)$ }. For the allocated submesh A_1, S is decomposed into two submeshes S_1' and S_2' as shown in Fig. 9, and so FSL = { $S_1'(\langle 0,0\rangle,\langle 6,9\rangle)$, $S_2'(\langle 0,5\rangle,\langle 9,9\rangle)$ }. For the allocated submesh A_2, since S_2' is overlapping with A_2, S_2' is removed from FSL and is decomposed into two free submeshes $S_3'(\langle 0,5\rangle,\langle 9,7\rangle)$ and $S_4'(\langle 0,5\rangle,\langle 6,9\rangle)$. Since S_3' is dominant and S_4' is covered by S_1', S_3' is inserted into FSL and finally FSL = { $S_1'(\langle 0,0\rangle,\langle 6,9\rangle)$, $S_3'(\langle 0,5\rangle,\langle 9,7\rangle)$ }.

```
Dealloc( R ) /* R is a released submesh */
{
  /* Step 1 */
  if (free_list == φ) {
    free_list = {R};
    return;
  }
  if (alloc_list == φ) {
    free_list = {(< 0,0 >,< L_X − 1, L_Y − 1 >)};
    return;
  }
  /* Step 2 */
  free_list = {(< 0,0 >,< L_X − 1, L_Y − 1 >)};
  for all allocated submesh S ∈ alloc_list do
  for all free submesh S' ∈ free_list do
    if (S is overlapping with S') {
      Delete S' from free_list and generate new submeshes by decomposing S';
      Insert the new submeshes into free_list if they are currently dominant;
    }
}
```

Fig. 8. Deallocation Algorithm.

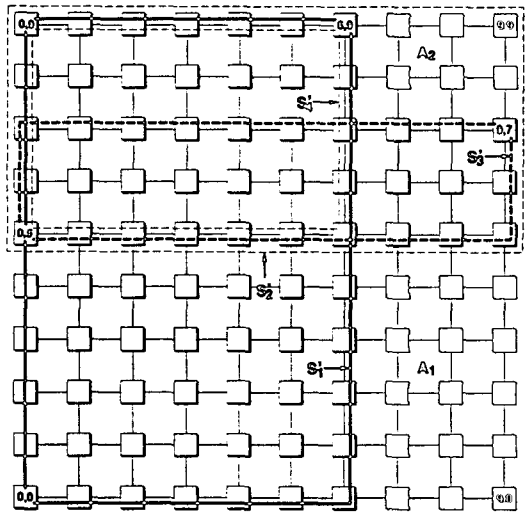


Fig. 9. Example of Deallocation Algorithm.

4. Algorithm Analysis

In the following, we analyze the theoretical time complexity of the proposed strategy in the worst case. We first analyze the time complexity of the allocation algorithm. In step 1 of the allocation algorithm, at most N_f free submeshes each of which accommodates a request are considered to generate the candidate submeshes, where N_f is the number of free submeshes in FSL. Thus, the time complexity of step 1 is $O(N_f)$. In step 2, at most $4 \cdot N_f$ candidate submeshes are evaluated for N_f submeshes in FSL. Hence, step 2 has the time complexity of $O(N_f^2)$. In step 3, at most N_f submeshes in FSL may be overlapping with the *best* submesh, and they are decomposed into at most four free submeshes which are inserted into FSL if they are dominant. Thus, the time complexity of step 3 in the worst case is $O(N_f^2)$, too. Therefore, the allocation algorithm takes $O(N_f^2)$ time complexity in the worst case.

Let us analyze the time complexity of the deallocation algorithm in the worst case. The time complexity of step 1 is $O(1)$ when either FSL or the allocation list is empty. In step 2, if we assume that the number of allocated submeshes in *alloc_list* is N_a, then from the following theorem, the maximum number of (dominant) free submeshes (N_f) is less than $4 \cdot N_a$. Since the time complexity of the body of *if*-statement of step 2 is $O(N_f)$, that of step 2 in the worst case is $O(N_a \cdot N_f^2) = O(N_a^3)$. Therefore, the deallocation algorithm takes $O(N_a^3)$ time complexity.

Theorem 1. *If there exist* $k(\geq 1)$ *allocated submeshes in a mesh, then the maximum number of dominant free submeshes is less than or equal to* $4 \cdot k$.

Proof. If k allocated submeshes exist in a mesh, the mesh can be divided into k arbitrary independent (i.e., non-overlapped) regions (submeshes), each of which must include only one allocated submesh. Then, each region includes at most four free submeshes, each of which may not be really a dominant free submesh. If a free submesh in a region is not a dominant free submesh, it is a part (submesh) of a dominant free submesh. Since there exist at most $4 \cdot k$ free submeshes in k regions and a dominant free submesh may be counted more than once if it is divided into the parts of one or more regions, the maximum number of dominant free submeshes is less than or equal to $4 \cdot k$.□

Table 1 summarizes the time complexities of the strategies described in Section 2 and the proposed strategy along with the other characteristics, where N is the number of nodes in a mesh.

Table 1. Characteristics of submesh allocation strategies.

Strategies	Allocation Complexity	Deallocation Complexity	Complete Recognition	Internal fragmentation	Type
Buddy	$O(N)$	$O(N)$	No	Yes	Best-Fit
FS	$O(N)$	$O(1)$	No	No	First-Fit
FF	$\Theta(N)$	$O(N)$	No	No	First-Fit
BF	$\Theta(N)$	$O(N)$	No	No	Best-Fit
AS	$O(N)$	$O(1)$	Yes	No	First-Fit
Busy-List	$O(N^3)$	$O(1)$	Yes	No	Best-Fit
Proposed	$O(N_f^2)$	$O(N_a^3)$	Yes	No	Best-Fit

IV. Simulation Study

A simulation study is conducted to evaluate the per-

formance of the proposed strategy and for the comparison with other strategies. Two previous strategies, the Busy-List[1] and AS strategies, are compared with our FSL strategy, since they show the better performance than any other strategies [11,12].

1. Workload and Job Scheduling

The workload considered on simulation is characterized by the job arrival distribution, distribution of the job size (submesh size), and distribution of the job residence (service) time. The job arrival pattern is assumed to follow Poisson distribution [14] with an arrival rate λ. The job residence time is assumed to follow exponential distribution [15] with a given mean residence time. The job size (the side lengths of submesh) is assumed to follow a given distribution: *uniform*, *normal*, and *exponential* distribution [15]. Under normal distribution, the mean of each side length distribution was selected as $(1+L_X)/2$ and $(1+L_Y)/2$, and the variance as half of the mean, i.e., $(1+L_X)/4$ and $(1+L_Y)/4$ for $L_X \times L_Y$ mesh. The exponential distribution has the same mean as the normal distribution. Values drawn from the distributions outside of the range were ignored. Under a given *system load* ρ $(0.0 < \rho \leq 1.0)$ and residence time, the job arrival rate (λ) is determined as follows:

$$\textit{job arrival rate } (\lambda) = \frac{\rho \cdot N}{m \cdot r}$$

where N is the number of processors contained in the mesh $(N = L_X \cdot L_Y)$, m is the mean number of processors in a submesh request, and r is the mean residence time.

We consider the two job scheduling disciplines, the First-Come-First-Served (FCFS) and Modified FCFS (M-FCFS). In the FCFS scheduling, only the job at the head of queue (*head job*) has the chance to be allocated. In the M-FCFS scheduling, when the head job is blocked, that is, it is failed to be allocated, subsequent jobs are considered in arrived sequence and allocated if suitable submeshes are found. The head job may suffer indefinite postponement under certain distribution of workload. To eliminate indefinite postponement, we impose a *threshold value* to the blocked head job, which is maximum queuing delay that a job can tolerate at the head of queue. The threshold could be predefined or computed dynamically. A dynamic threshold is derived by $d_t \cdot \lambda$, where d_t is average queueing delay for allocated jobs until now. The average queueing delay and the arrival rate are monitored by the scheduler. The scheduler updates the threshold value everytime a task is allocated to the system. When the threshold of the head job is reached, it gets priority over all other jobs. No jobs are allocated until the head job is allocated.

1) The source code of the busy-list strategy was provided by D.D.Sharma and D. K. Pradhan [12].

2. Simulation Result

The simulator was developed in C and compiled with the highest level of optimization on a SUN SPARCStation 10/514 multiprocessor system. The performance of strategies is measured in terms of the *mean waiting delay*, *allocation miss*, and *mean search time* for 100,000 requests per run of simulation under 95% confidence level with the error range of $\pm 3\%$, which are defined as follow:

- *waiting delay*, the time that elapses from the moment a job initially arrives to the system until it is allocated to a submesh.
- *allocation miss*, the percentage of allocation miss for valid requests, where a request is valid if there are enough nodes available.
- *mean search time*, the average cpu time elapsed on allocation(s) and deallocation per job (the allocation attempt for a job may be done more than once).

We have assumed the mesh system to be a square one, done for simplicity to plot curves, though the trends should remain the same for any arbitrary mesh systems.

We first present the simulation results based on FCFS scheduling. The first experiment was performed to measure the mean waiting delay by changing the size of mesh from 16×16 to 512×512. The offered load and the mean residence time were given 0.47 and 10 seconds, respectively. Fig. 10 gives the results of experiment for the uniform and exponential distributions. The experimental result for the normal distribution exhibited similar trend as that for exponential distribution and therefore, is omitted. For the exponential distribution, the result of the AS strategy for over 64×64 mesh is eliminated to obtain better scaling factors, since the AS strategy is saturated at the load 0.47. As can be seen in the figures, the proposed strategy outperforms the other strategies. As the mesh size increases, the performance difference between our strategy and the other strategies increases. Moreover, the proposed strategy shows the nearly similar performance for both uniform and exponential distributions, while the Busy-List and AS strategies show the more or less dependent results on the type of load. For the uniform distribution, the proposed strategy improves the mean waiting delay about 31% to 56% over the AS strategy and 14% to 26% over the Busy-List strategy. For the exponential distribution, the proposed strategy achieves 46% to 91% and 16% to 39% improvements over the AS and the Busy-List strategies, respectively. Also, under the normal distribution, the proposed strategy achieves 41% to 83% and 19% to 36% improvements over the AS and the Busy-List strategies, respectively. These results are due to relative efficiency of the heuristic of the proposed strategy over the first-fit policy of the AS strategy and the heuristic of the Busy-List strategy.

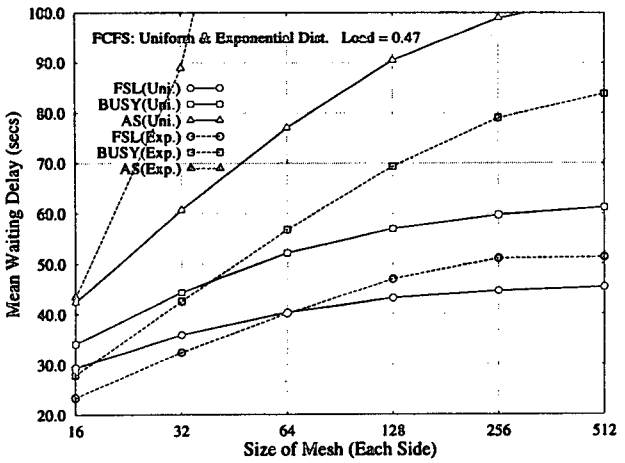


Fig. 10. Mean Waiting Delay vs. Size of Mesh.

The second experiment was performed to measure the mean waiting delay by changing the offered load from 0.1 to 1.0 for 64×64 mesh. Fig. 11 gives the results of experiment for the uniform and normal distributions with the range of the load from 0.3 to 0.55, to obtain better scaling factor because all the strategies are saturated over the load 0.55. The proposed strategy performs the best under all the loads. As the load increases, the AS and Busy-List strategies are saturated more rapidly than the proposed strategy and also the performance difference grows. Only the proposed strategy can sustain the load over 0.5, nearly up to 0.55, for both uniform and normal distributions. Similar result was observed for the exponential distribution, and therefore, the results are not presented.

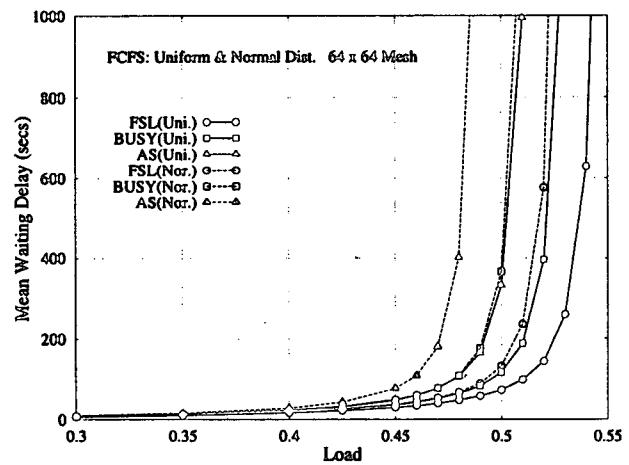


Fig. 11. Mean Waiting Delay vs. Load.

To observe a picture of the optimality of the strategies, the third experiment was performed to gauge the percentage of the allocation miss for valid requests. Figs. 12 and 13 present the results of experiment for the uniform and normal distributions for the load 0.47 and 64×64 mesh, respectively. The proposed strategy exhibits the lowest miss percentage compared to the other strategies. The allocation miss for a valid request is due to the external fragmentation including the unnecessary external fragmentation resulted from the decision in the previous allocation step, as already explained in the previous section. Therefore, the experimental results demonstrate that our strategy provides more optimal allocation than the other strategies.

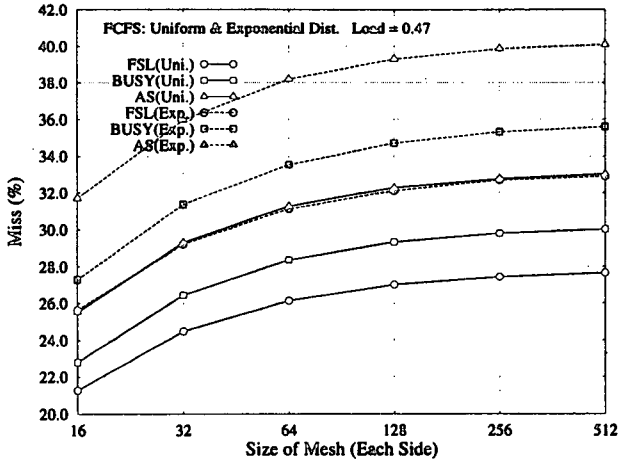


Fig. 12. Allocation Miss vs. Size of Mesh.

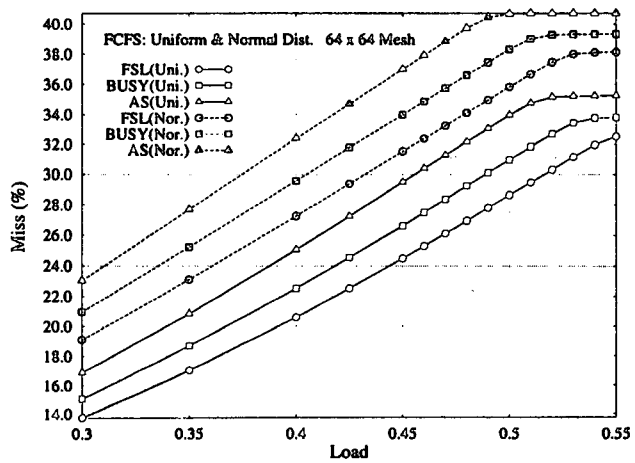


Fig. 13. Allocation Miss vs. Load.

The fourth experiment was performed to compare the run-time overhead, i.e., the mean search time, of strategies. The experimental results are given in Fig. 14 for the exponential and normal distributions. Fig. 14 shows that the Busy-List and our strategy give the nearly constant mean search times, regardless of the size of mesh, while our strategy exhibits slightly lower than the Busy-List strategy. The AS strategy, however, provides the mean search time increasing in proportion to the size of mesh.

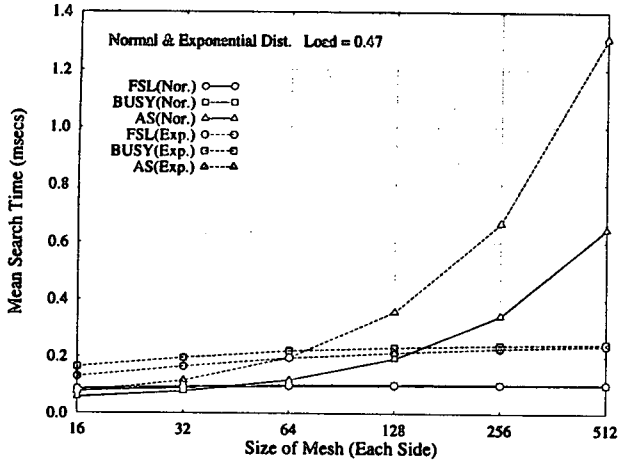


Fig. 14. Mean Search Time vs. Size of Mesh.

Next, we present the experimental results based on M-FCFS scheduling. The experimental results are given Figs. 15, 16, and 17. Figs. 15 and 16 show that the M-FCFS discipline gives better performance compared to the FCFS scheduling of Figs. 10 and 11. The proposed strategy shows better performance than the other two strategies. As shown in Fig. 15, the proposed strategy improves mean waiting delay about 16% to 28% over the AS strategy and 7% to 13% over the Busy-List strategy under uniform distribution at load 0.57. For the exponential distribution, the proposed strategy achieves 15% to 25% improvement over the AS strategy and 5% to 11% over the Busy-List strategy. Fig. 16 shows that the proposed strategy performs the best under all the loads. Fig. 17 presents mean search times of strategies. The proposed strategy shows the lowest mean search time which is nearly constant regardless of both the size of mesh and distribution.

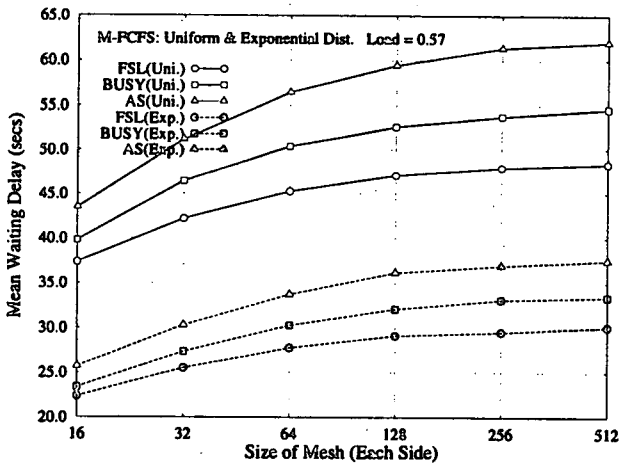


Fig. 15. Mean Waiting Delay vs. Size of Mesh under M-FCFS Scheduling.

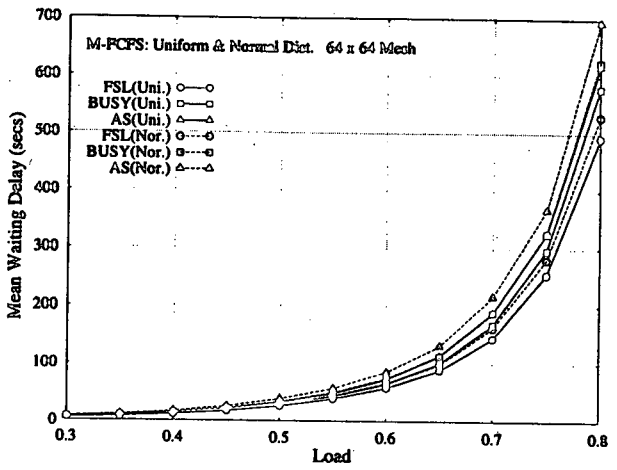


Fig. 16. Mean Waiting Delay vs. Load under M-FCFS Scheduling.

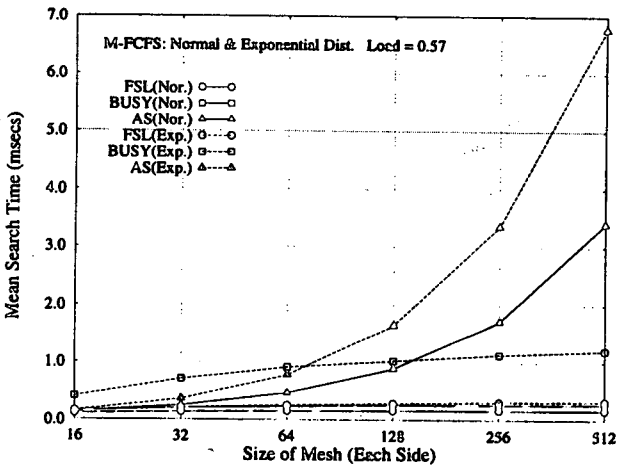


Fig. 17. Mean Search Time vs. Size of Mesh under M-FCFS Scheduling.

From the simulation study, it should be noted that, on both FCFS and M-FCFS scheduling, the proposed strategy exhibits the best performance compared to the other strategies and obvious improvement for all the performance measures under various distributions, regardless of both the size of mesh and the load.

V. Conclusion

The performance of mesh multicomputer possessing hundreds or thousands of processors is subject to the efficient use of its processors. Therefore, an efficient submesh allocation strategy is essential for achieving the high performance on mesh multicomputers. In this paper, we

introduced a best-fit submesh allocation strategy for the mesh multicomputers. The proposed strategy allocates a best-fit submesh which can preserve the large free submeshes of wide range of size as many as possible and thus causes the minimal fragmentation by using *reservation factor* quantifying the potential fragmentation of candidate submeshes. To evaluate the quality of the proposed strategy, we compared with the other two complete recognition strategies by simulation. Simulation results show that the proposed strategy leads to the lowest waiting delay and allocation miss compared to other strategies for meshes of various sizes under various kinds of system loads. The proposed strategy achieves about 30% and 10% average improvements over the best previous strategy under the FCFS and Modified FCFS job scheduling disciplines, respectively. As the size of meshes and the loads increase, the performance difference becomes more obvious. Moreover, our strategy shows the least run-time overhead which is fairly constant regardless of both the size of mesh and the load. Therefore, the proposed strategy offers the most efficient and practical solution for mesh multicomputers.

References

[1] Intel Corp., *A Touchstone DELTA System Description*, 1991.

[2] Intel Corp., *Paragon XP/S Product Overview*, 1991.

[3] R. Alverson et al., "The Tera computer system," *Proc. 1990 Int'l Conf. on Supercomputing*, pp.1-6, 1990.

[4] D. K. Kahaner and U. Wattenberg, "Japan: a competitive assessment," *IEEE Spectrum*, pp.42-47, Sept. 1992.

[5] G. Zorpette, "The power of parallelism," *IEEE Spectrum*, pp.28-33, Sept. 1992.

[6] T. E. Bell, "Beyond today's supercomputers," *IEEE Spectrum*, pp.72-75, Sept. 1992.

[7] K. Li and K. H. Cheng, "A Two Dimensional Buddy System for Dynamic Resource Allocation in A Partitionable Mesh Connected System," *Proc. ACM Computer Science Conf.*, pp.22-28, Feb. 1990.

[8] K. Li and K. H. Cheng, "A Two Dimensional Buddy System for Dynamic Resource Allocation in A Partitionable Mesh Connected System," *J. Parallel and Distributed Computing*, pp.79-83, 1991.

[9] P. Chuang and N. Tzeng, "An Efficient Submesh Allocation Strategy for Mesh Computer Systems," *Proc. 11th Int'l Conf. on Distributed Computing Systems*, pp.256-262, 1991.

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

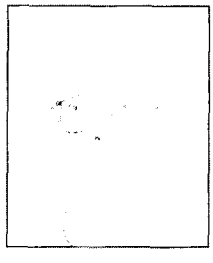
[11] J. Ding and L. N. Bhuyan, "An Adaptive Submesh Allocation Strategy for Two-Dimensional Mesh Connected Systems," *Proc. 1993 Int'l Conf. on Parallel Processing*, vol.II, pp.193-200, 1993.

[12] D. D. Sharma and D. K. Pradhan, "A Fast and Efficient Strategy for Submesh Allocation in Mesh-Connected Parallel Computers," *IEEE Symp. on Parallel and Distributed Processing*, pp.682-689, Dec. 1993.

[13] P. Krueger, T.-H. Lai, and V. A. Radiya, "Processor Allocation vs. Job Scheduling on Hypercube Computers," *Proc. 11th Int'l Conf. on Distributed Computing Systems*, pp.394-401, 1991.

[14] S. M. Ross, *Introduction to Probability Models: Third Edition*, Orlando, Florida:Academic Press, Inc., 1985.

[15] W. H. Press et al., "Numerical Recipes in C: The Art of Scientific Computing," Cambridge: Cambridge University Press, 1988.



Geunmo Kim received the B.S. degree in computer science from Yonsei University, Korea, in 1989, and the M.S. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Korea, in 1991. Since 1991, he has been working towards the Ph.D. degree in the Department of Computer Science, KAIST, Korea. His major research areas are parallel and distributed proecssing, multiprocessor systems, and interconnection network.

Hyunsoo Yoon received the B.S. degree in electronics engineering from the Seoul National University, Korea, in 1979, the M.S. degree in computer science from the Korea Advanced Institute of Science and Technology, in 1981, and the Ph.D. degree in computer and information science from the Ohio State University, Columbus, Ohio, in 1988. From 1978 to 1980, he was with the Tongyang Broadcasting Company, Korea, from 1980 to 1984, with the Samsung Electronics company, Korea, and from 1988 to 1989, with the AT & Bell Labs. as a Member of Technical Staff. He joined the faculty of KAIST in 1989. His research interests include parallel computer architecture, parallel computing, interconnection network, protocol engineering, and neural network.