Journal of Parallel and Distributed Computing **60**, 616–638 (2000)
doi:10.1006/jpdc.1999.1604, available online at http://www.idealibrary.com on IDEAL®

On Task Relocation in Two-Dimensional Meshes

Seong-Moo Yoo

Department of Computer Science, Columbus State University, Columbus, Georgia 31907
E-mail: syoo@colstate.edu

Hyunseung Choo

School of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, Korea
E-mail: choo@ece.skku.ac.kr

and

Hee Yong Youn, Chansu Yu, and Younghee Lee

School of Engineering, Information and Communications University, Taejon, Korea
E-mail: youn@icu.ac.kr; cyu@icu.ac.kr; yhlee@icu.ac.kr

Received February 2, 1998; accepted October 5, 1999

In parallel computer systems with a number of processors, *external fragmentation* is caused by continuous allocation and deallocation of processors to tasks which require exclusive use of several contiguous processors. With this condition, the system may not be able to find contiguous processors to be allocated to an incoming task even with a sufficient number of free processors. Relocation is an approach for alleviating this problem by reassigning the running tasks to other processors. In this paper, we examine two relocation schemes—*full relocation* and *partial relocation scheme*—for two-dimensional meshes. The full relocation scheme is desirable when the system is highly fragmented, while the partial relocation scheme is used for minimizing the number of relocated tasks. For the relocation process, we formally define and use two basic submesh movement operations—shifting and rotating. Comprehensive computer simulation reveals that the proposed schemes are beneficial when the relocation overhead is not high, which is machine dependent.

Key Words: external fragmentation; full and partial relocation scheme; mesh-connected systems; task allocation and deallocation; task relocation.

1. INTRODUCTION

Among several important interconnection topologies developed for parallel and distributed computing, two-dimensional (2D) mesh topology has become popular due to its simplicity and efficiency [1, 2]. There exist a number of commercial and experimental parallel computer systems built or under being development based on 2D mesh. Typical examples are Intel Paragon [3] and Intel/DARPA Touchstone

0743-7315/00 $35.00

Delta [4]. As in general parallel computer systems [5, 6], jobs submitted to a 2D mesh computer system are first placed in a waiting queue. Here, each job requires a submesh of a certain width and height for a certain time period. It is assumed that there exists a separate host processor keeping and processing *job dispatcher*, which consists of *job scheduler* and *processor allocator*. The job scheduler chooses the next job to be processed from the waiting queue according to the scheduling policy. The processor allocator finds a free submesh for the chosen job using a processor allocation scheme. As the size of the mesh grows, however, the efficient submesh allocation becomes an increasingly demanding task.

Li and Cheng [7] proposed a Buddy strategy for task allocation in 2D mesh applicable to only square meshes, where the length of one side must be power of 2. The strategy thus has the problem of overallocation (internal fragmentation) beyond what is actually needed because most jobs do not necessarily require square meshes. To solve this problem, Chuang and Tzeng [8] proposed the frame sliding (FS) strategy for meshes of arbitrary lengths and widths. The strategy allocates a free submesh which exactly matches the size of the incoming task. Hence, it eliminates the overallocation problem, but the searching process may result in allocation misses; i.e., it cannot recognize a free submesh for an incoming task even when one is available. Two schemes were proposed to solve this allocation miss problem; Zhu [9] proposed the First Fit and the Best Fit strategies, and Ding and Bhuyan [10] proposed the Adaptive Scan (AS) strategy. AS strategy not only solves the allocation miss problem but also increases the system utilization by employing an approach called *Address Translation*. Later, to improve the waiting delay and allocation time incurred in AS strategy, two schemes were proposed by Sharma and Pradhan [11] and Yoo *et al.* [12]. Lo *et al.* [13] also proposed noncontiguous allocation schemes.

Similar to the fragmentation phenomenon in a conventional memory system, however, continuous allocation and deallocation in the mesh system result in fragmented meshes. Then, even though a sufficient number of nodes are available, a submesh large enough for accommodating an incoming task may not be able to be found. This is called *external fragmentation*. Irrespective of the allocation

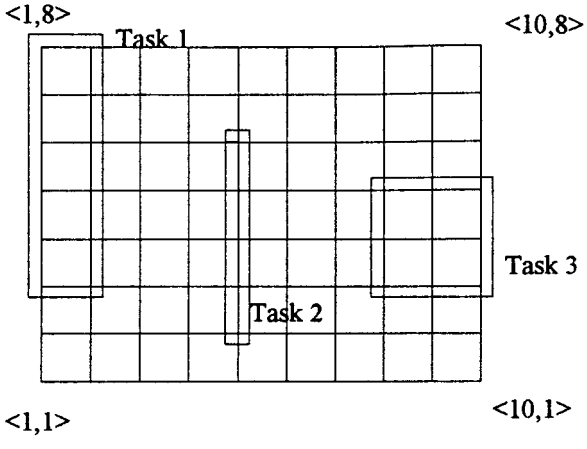

FIG. 1. An example of external fragmentation.

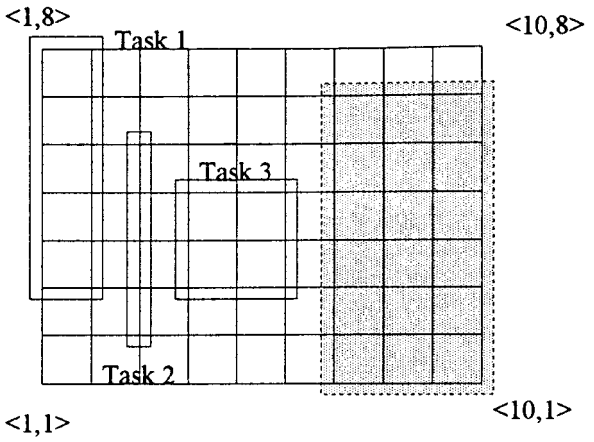

FIG. 2. The allocation of $T = (4, 7)$ after relocation.

strategy employed, the external fragmentation is unavoidable. Figure 1 shows an example of a fragmented mesh where the free 54 nodes cannot form a submesh for accommodating a task of only 4×7. If Tasks 2 and 3 are relocated to the left side, as shown in Fig. 2, then a submesh can be allocated to the task.

It is clear that fragmentation leads to the poor utilization of the nodes in the mesh. Ding and Bhuyan [10] showed from their experiment that the AS strategy still suffers from external fragmentation resulting in an approximately 30% performance degradation, even though it can be alleviated through *Address Translation*. As the fragmentation problem in the memory system is handled by memory compaction, *task relocation* can alleviate the external fragmentation problem in parallel computer systems. It relocates active tasks at one side of the structure in order to make a sufficiently large submesh at the other side for the incoming tasks. The task relocation approach was proposed and examined experimentally for hypercube in [14, 15].

In this paper we propose two relocation schemes for 2D meshes—full relocation (FR) and partial relocation (PR) scheme. The full relocation scheme relocates all previously allocated tasks, while the partial relocation scheme relocates only the tasks required to be moved to render a submesh for the incoming task. The performances of the proposed schemes are evaluated by computer simulation considering the *task relocation overhead*. To identify the relative effectiveness of the relocation schemes, the proposed relocation schemes are also compared with an efficient allocation scheme [10]. The computer simulation reveals that the proposed schemes improve the task completion time and processor utilization up to a certain degree of relocation overhead which is machine dependent. It was also found that the full relocation scheme outperforms the partial relocation scheme for most cases.

The rest of the paper is organized as follows. In Section 2, definitions and notations are introduced which will be used throughout the paper. In Section 3, we define and solve the submesh relocation problem in 2D mesh architecture. The proposed full and partial relocation schemes are presented in Section 4. In Section 5, the performances

of the proposed schemes are evaluated by computer simulation for various practical operational conditions. Finally, we conclude the paper in Section 6.

2. DEFINITIONS AND NOTATION

A two-dimensional mesh, $M(a, b)$, is an $a \times b$ rectangular grid consisting of ab nodes, where a and b represent the width and height of the mesh, respectively. Each node in the mesh refers to a processor and represented by the coordinate $\langle x, y \rangle (1 \leqslant x \leqslant a, 1 \leqslant y \leqslant b)$. It is assumed that the column and row indices increase from left to right and bottom to top starting from 1.

In our system, it is assumed that more than one process are allowed to arrive at or leave a node simultaneously if they use different links. For example, in Fig. 3, assume that a process p_1 arrives at $\langle i, j \rangle$ from $\langle i-1, j \rangle$, while another process p_2 does that from $\langle i, j+1 \rangle$. If p_1 moves to $\langle i+1, j \rangle$, then p_2 can move to either $\langle i-1, j \rangle$ or $\langle i, j-1 \rangle$ without a contention. If a contention occurs, the process with the higher priority (the process with the longer relocation distance) moves first.

DEFINITION 1. Internal fragmentation is the ratio of the number of over-allocated processors to that of actually required processors. External fragmentation is the ratio of the number of available processors to the total number of processors in the system, when allocation failure occurs even with a sufficient number of free processors for the incoming task.

DEFINITION 2. Let w_S and h_S denote the width and height of a submesh, S, respectively. Let also ul_S, ur_S, ll_S, and lr_S denote the upper-left, upper-right, lower-left, and lower-right corner of S, respectively. The *address* of S is a quadruple $\langle x, y, x', y' \rangle$, where $\langle x, y \rangle$ and $\langle x', y' \rangle$ indicate ll_S and ur_S, respectively. Here $w = x' - x + 1$ and $h = y' - y + 1$. The *base* of S refers to ll_S, while the *area* of a submesh $S(w, h)$ is the number of nodes in it, and clearly $w_S h_S$.

DEFINITION 3. The node distance of two nodes $\langle x, y \rangle$ and $\langle x', y' \rangle$ is defined as $|x - x'| + |y - y'|$. Here $|x - x'|$ and $|y - y'|$ denote the *horizontal* and *vertical node distance*, respectively.

DEFINITION 4. Residence time of a task is the time between starting execution and completion. The relocation overhead factor, α, is the ratio of the time required for a process to be relocated from a node to a physically adjacent node to the

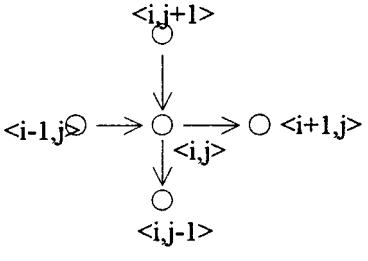

FIG. 3. Routing model.

residence time of the process. In other words, the time for moving a process one position is obtained by multiplying α and the residence time. α is a function of several factors dependent on the actual implementation, and thus it is given as an input parameter.

For example, the peak unidirectional network bandwidth of Paragon [16] is 200 MB/s, which means that the time required for a process with 200 bytes of data to be relocated is 1 μs. If the residence time of the process is 100 μs, then α is 0.01.

DEFINITION 5. For the two same size submeshes, $S_S \langle x_s, y_s, x'_s, y'_s \rangle$ and $S_D \langle x_d, y_d, x'_d, y'_d \rangle$, the *corresponding corner* of ul_{SS}, ur_{SS}, ll_{SS}, and lr_{SS} is ul_{SD}, ur_{SD}, ll_{SD}, and lr_{SD}, respectively. The *shortest* matching corner is the corner for which the node distance between the two corresponding corners is minimal among the four corners. If $w_{SS} = w_{SD}$ and $h_{SS} = h_{SD}$, then the four distances will be all the same. In this case, *ll* is assumed to be the shortest matching corner.

For example, for S_S and S_D in Fig. 4, *ll* is the shortest matching corner since the distances of corresponding corners are 6, 7, 5, and 6, respectively.

DEFINITION 6. Assume two same size submeshes S_S and S_D at different locations. Shifting S_S to S_D is defined as the continuous movement of S_S horizontally (vertically) first and then vertically (horizontally) so that the shortest matching corner of S_S overlaps with that of S_D. Assume the new shifted submesh is S_{ID} $\langle x_{id}, y_{id}, x'_{id}, y'_{id} \rangle$. If the orientations of S_S and S_D are same, S_{ID} is same as S_D and the relocation is finished. Otherwise, *rotating* S_{ID} to S_D is required to complete the relocation.

For example, in Fig. 4, during shifting, each process in S_S is moved to S_{ID} in parallel which takes 5 time units (2 for horizontal and 3 for vertical movements). Now each node in S_{ID} should be mapped to the corresponding node in S_D, whose procedure is defined as rotating.

DEFINITION 7. Assume that a task was relocated from S_S to S_D. The relocation distance between S_S and S_D, RD (S_S, S_D), is the maximum routing time between all pairs of two corresponding nodes of S_S and S_D.

We next discuss two main submesh movement operations in 2D mesh structure which are required in our task relocation scheme.

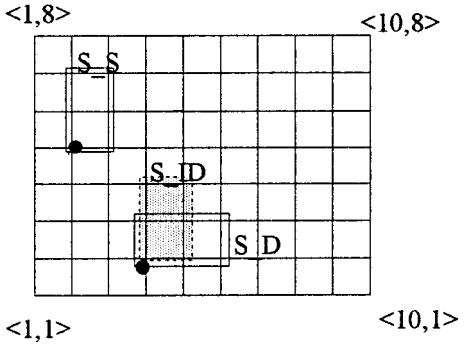

FIG. 4. The shortest matching corner.

3. SUBMESH RELOCATION IN 2D MESHES

In this section two main submesh movement operations in a 2D mesh—shifting and rotating—are studied. Here, all the processes of a task allocated to S_S are moved in parallel to the nodes in another submesh, S_D. According to the orientations of S_S and S_D, the relocation involves shifting and sometimes rotating also as mentioned above.

Both the shifting and rotating process consist of two steps—*node mapping* and *routing*. Node mapping is a logical operation which maps each node in the source submesh to the corresponding node in the destination submesh. Routing is the actual movement of the processes to the destination nodes. Shifting is discussed first.

3.1. Shifting

When the two meshes, S_S and S_D, have the same orientation, the node-mapping is straightforward since the widths and the heights of the source and destination submeshes are same. The node mapping, thus, for a source node in S_S, $\langle s_i, s_j \rangle$, the intermediate destination node in S_{ID}, $\langle id_i, id_j \rangle$, and the destination node in S_D, $\langle d_i, d_j \rangle$, is that $d_i = id_i = s_i + x_d - x_s$ and $d_j = id_j = s_j + y_d - y_s$. The following is the actual routing algorithm from $\langle s_i, s_j \rangle$ to $\langle d_i, d_j \rangle$ in the shifting process.

ALGORITHM-S: Shifting between two nodes.
$i \leftarrow s_i; j \leftarrow s_j$
vertical_movement( )
horizental_movement( )

Procedure vertical_movement( )
  repeat
    if $j < d_j$, move from $\langle i, j \rangle$ to $\langle i, j+1 \rangle$ and $j = j+1$ /*up*/
    else if $j > d_j$, move from $\langle i, j \rangle$ to $\langle i, j-1 \rangle$ and $j = j-1$ /*down*/
    else stop
  until $(j = d_j)$

Procedure horizontal_movement( )
  repeat
    if $i < d_i$, move from $\langle i, j \rangle$ to $\langle i+1, j \rangle$ and $i = i+1$ /*right*/
    else if $i > d_i$, move from $\langle i, j \rangle$ to $\langle i-1, j \rangle$ and $i = i-1$ /*left*/
    else stop
  until $(i = d_i)$

For example, for the shifting from S_S to S_{ID} in Fig. 4, the process of the source node $\langle 2, 5 \rangle$ moves vertically down to $\langle 2, 2 \rangle$, then horizontally to the destination node $\langle 4, 2 \rangle$.

LEMMA 3.1. *No contention occurs between the processes of a task while they are shifted.*

Proof. According to Algorithm-S, all processes in our model move together in only one direction in the pipelined fashion during shifting. Therefore, no more than

one process need to use the same link at the same time. Certainly, no contention can occur.

LEMMA 3.2. *Shifting can be made either horizontally - first or vertically - first. In either case, the routing time for one process is* $|x_s - x_d| + |y_s - y_d|$ *assuming that the routing time for a process from a node to a physically adjacent node is a unit time when no contention exists for the link, and it is minimum.*

Proof. In shifting, the paths between every source and destination node are the shortest paths. Due to Lemma 3.1, no delay occurs due to contention. Therefore, the routing time is the node distance of the shortest matching corners.

THEOREM 3.1. *The routing of nodes in Algortihm-S is deadlock free.*

Proof. There is no circular wait in the paths because all processes move together in only one direction in the pipelined fashion.

Let d be the distance between the two corresponding nodes of S_S and S_D where $w_{SS} = w_{SD}$ and $h_{SS} = h_{SD}$. As a direct consequence of Lemmas 3.1 and 3.2 and Theorem 3.1, it is clear that all processes in S_S can be relocated to S_D in d time units. We next consider the rotating process.

3.2. Rotating

When the orientations of S_S and S_D are different from each other, first an optimal shifting needs to be found. Assume an upright shape ($w < h$) S_S and thus the lying shape ($w \geqslant h$) S_D. Then, according to the relative position of S_D with respect to S_S, there exist four different cases as shown in Fig. 5. For the lying shape S_S and upright shape S_D, there exist another four different combinations. For each of these eight different combinations, the source submesh is shifted first such that the shortest matching corners overlap each other. Table I lists the eight combinations and the corresponding shortest matching corners. It also lists the corresponding $\langle d_i, d_j \rangle$'s where $\langle id_i, id_j \rangle$ denotes the node before rotation and $\langle x, y \rangle$ denotes the shortest matching corner of S_D. Here $id_i = s_i + x_d - x_s$ and $id_j = s_j + y_d - y_s$ as mentioned in Section 3.1. Once the shortest matching corners overlap, next the shifted submesh, S_{ID}, needs to be rotated to complete the relocation. Here an optimal rotation is achieved by rotating the submesh toward the destination mesh as shown in Fig. 6.

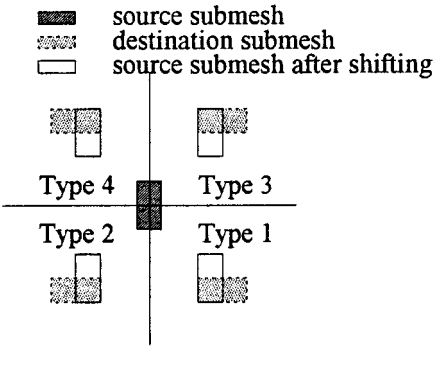

FIG. 5. Relative location of destination submesh.

TABLE I

The Shortest Matching Corners For Eight Different Combinations of the Relative Location of S_S and S_D

Type	Location of S_D	Shape of source task	The shortest matching corner	d_i	d_j
1	*lr*	$w \leqslant h$	*ll*	$x+y+h-1-id_j$	$y-x+id_i$
2	*ll*	$w \leqslant h$	*lr*	$x-y-h+1+id_j$	$x+y-id_i$
3	*ur*	$w \leqslant h$	*ul*	$x-y+h-1+id_j$	$x+y-id_i$
4	*ul*	$w \leqslant h$	*ur*	$x+y-h+1-id_j$	$y-x+id_i$
5	*lr*	$w > h$	*ur*	$x-y+id_j$	$x+y-w+1-id_i$
6	*ll*	$w > h$	*ul*	$x+y-id_j$	$y-x-w+1+id_i$
7	*ur*	$w > h$	*lr*	$x+y-id_j$	$y-x+w-1+id_i$
8	*ul*	$w > h$	*ll*	$x-y+id_j$	$x+y+w-1-id_i$

THEOREM 3.2. *For Type 1 of Table I, a source node* $\langle s_i, s_j \rangle$ *is mapped to* $\langle d_i, d_j \rangle$ *where* $d_i = x+y+h-1-id_j$ *and* $d_j = y-x+id_i$.

Proof. $\langle s_i, s_j \rangle$ is mapped to $\langle id_i, id_j \rangle$ after shifting. Refer to Fig. 7. Since S_{ID} is rotated 90° counter-clockwise, it is easy to see that $d_i = (x+h-1)-(id_j-y) = x+y+h-1-id_j$, and $d_j = y+(id_i-x) = y-x+id_i$.

Assume $S_S = \langle 2, 5, 3, 7 \rangle$ and $S_D = \langle 4, 2, 6, 3 \rangle$ as in Fig. 4. It is Type 1, and thus $S_{ID} = \langle 4, 2, 5, 4 \rangle$ and $\langle x, y \rangle = \langle 4, 2 \rangle$. The node-mappings are $\langle 2, 5 \rangle \rightarrow \langle 4, 2 \rangle$ from shifting, then $\langle 6, 2 \rangle$ from rotating. This is because $d_i = 4+2+3-1-2 = 6$ and $d_j = 2-4+4 = 2$. $\langle d_i, d_j \rangle$ for other seven types are obtained by the same way, and they are also listed in Table I.

Now consider an exceptional case shown in Fig. 8. Observe that the shortest matching corner between S_S $\langle 2, 1, 3, 7 \rangle$ and S_D $\langle 4, 4, 10, 5 \rangle$ is *ul*, which is *P* in the figure. In a 2D mesh where no end-around links exist, S_S cannot be shifted to S_{ID1} because S_{ID1} is out of bound. In this case, S_S is shifted to only one direction to S_{ID2}. Table II lists the adjusted destination nodes of Table I considering this

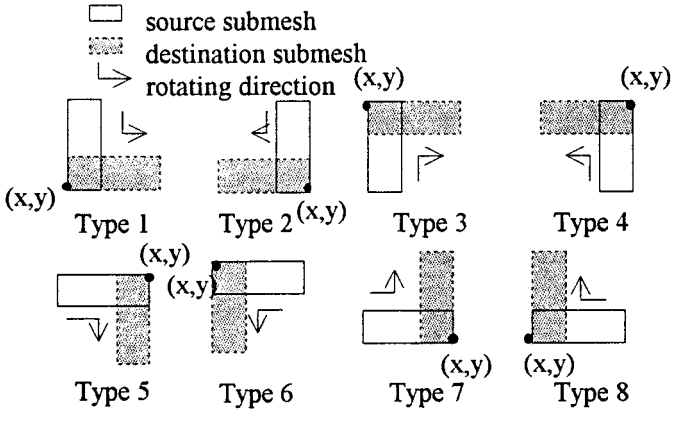

FIG. 6. Eight rotating types.

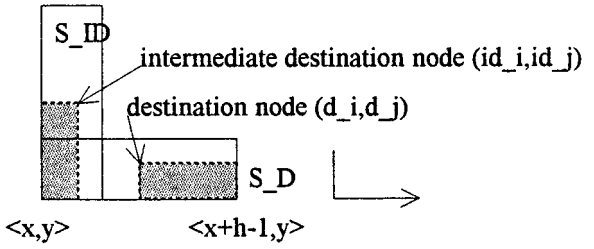

FIG. 7. Node mapping of Type 1.

condition, where d is the distance of two corresponding shortest matching corners between S_S and S_{ID}. Note that this exceptional case is irrelevant to a 2D torus.

After the node-mapping, actual routing of each process is done according to the following algorithm.

```
ALGORITHM-R: Rotating between two nodes.
if (Type = 1, 4, 5, or 8)
    if (d_i ≥ s_i and d_j ≥ s_j) or (d_i < s_i and d_j < s_j)
        horizontal_movement( )
        vertical_movement( )
    else
        vertical_movement( )
        horizontal_movement( )
else /* Type = 2, 3, 6, or 7 */
    if (d_i ≥ s_i and d_j ≥ s_j) or (d_i < s_i and d_j < s_j)
        vertical_movement( )
        horizontal_movement( )
    else
        horizontal_movement( )
        vertical_movement( )
```

LEMMA 3.3. *No contention occurs between the processes of a task while they are rotated.*

Proof. Here only Type 1 is considered since all eight types are symmetric. The main property of Algorithm-R is that the nodes in the source submesh move along

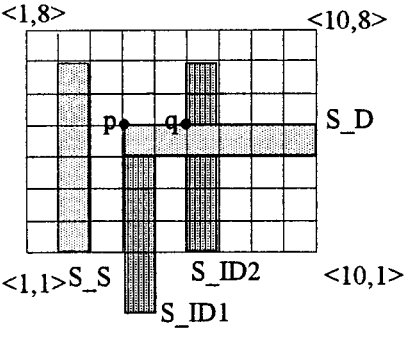

FIG. 8. Example of out-of-mesh-bound.

TABLE II

Adjustment of Destination Nodes in Table I in Exceptional Cases

Type	d_i	d_j
1	$x+y+h-1-id_j-d$	$y-x+id_i-d$
2	$x-y-h+1+id_j+d$	$x+y-id_i-d$
3	$x-y+h-1+id_j-d$	$x+y-id_i+d$
4	$x+y-h+1-id_j+d$	$y-x+id_i+d$
5	$x-y+id_j+d$	$x+y-w+1-id_i+d$
6	$x+y-id_j-d$	$y-x-w+1+id_i+d$
7	$x+y-id_j+d$	$y-x+w-1+id_i-d$
8	$x-y+id_j-d$	$x+y+w-1-id_i-d$

the disjoint paths as shown in Fig. 9. Therefore, no contention can occur. The same property holds for the out-of-bound case as shown in Fig. 10.

THEOREM 3.3. *All processes in S_{ID} can be rotated to S_D in max $(w-1, h-1)$ time units.*

Proof. In rotating phase, the process movement requiring the longest time is clearly due to the nodes at boundaries. Observe from Fig. 9 that ul_{SID}, which is mapped to ll_{SD}, is one of such nodes. Clearly the distance is $h-1$. Due to Lemma 3.3, no contention occurs during rotating, and all processes in Type 1, 2, 3, and 4 can be rotated in $h-1$ $(>w-1)$ time units. Similarly, the maximum rotating time for Type 5, 6, 7, and 8 is $w-1$ $(>h-1)$. Therefore, the rotating time is max $(w-1, h-1)$. A similar proof can be applied to the out-of-bound case.

THEOREM 3.4. *The routing of nodes in Algorithm-R is deadlock free.*

Proof. Due to Lemma 3.3, no contention occurs during rotating. Therefore, there is no circular wait in the paths.

We next present the proposed relocation schemes which are based on these two basic routing processes.

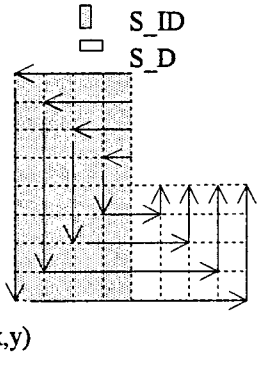

FIG. 9. Routing paths of processes in Type 1.

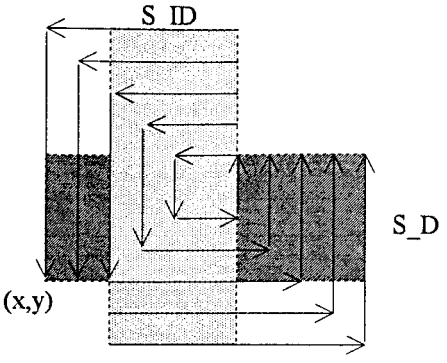

FIG. 10. Routing paths of processes in Type 1 for out-of-bound case.

4. PROPOSED RELOCATION SCHEMES

In this section, two proposed relocation schemes—full and partial relocation—are introduced. *Full relocation* is the relocation involving all tasks in the mesh, and thus it is desirable when the system is highly fragmented. Assume, however, that allocation of an incoming task becomes possible by relocating only a portion of allocated tasks. In this situation, it may be preferable to relocate only those tasks. This is called *partial relocation.* We first introduce the full relocation.

4.1. Full Relocation

For the task relocation in 2D mesh structures, three steps are involved: they are

(i) *submesh-mapping* from the source submesh to the destination submesh,

(ii) *node-mapping* between the nodes in the source submesh and the destination submesh, and finally

(iii) *task movement* through the shortest deadlock-free paths. The submesh mapping and node mapping are logical steps for finding the new locations of the previously allocated tasks, while the actual task relocation takes place in the final step of task movement. For submesh mapping, *c-list* and *r-list* are maintained.

DEFINITION 8. *c-list* is an ordered list which keeps the task id and the x-coordinate of the base of the submesh allocated to the task by the increasing order of x-coordinates. *r-list* is based on y-coordinate. For example, *c-list* of Fig. 1 is (1, 1), (2, 5), (3, 8), and *r-list* is (2, 2), (1, 3), (3, 3) assuming Task 3 was allocated later than Task 1. An entry is made and deleted from them at the allocation and deallocation time, respectively.

The following explains each of the three steps required in the full relocation scheme. In the submesh-mapping step, all tasks are shifted (of course logically) to the left (bottom) if the height of the incoming task is greater (smaller) than the width. Then it is checked if a submesh large enough to accommodate the incoming task is available. If not, they are shifted to the bottom (left), and then the availability is checked again. This process is repeated until the desired size submesh is obtained or no more movement can be possible. We call this interleaved rowwise

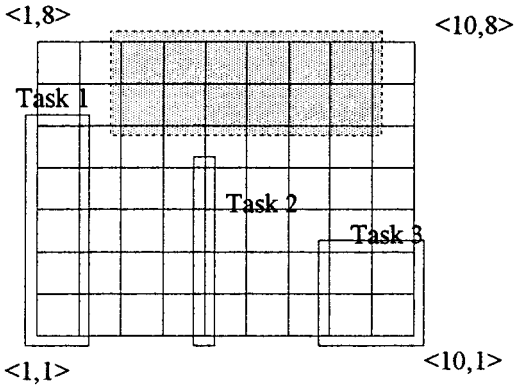

FIG. 11. The allocation of $T=(7, 3)$ after down movement.

and columnwise shift *interleaved compaction*. For example, in Fig. 1, assume that $T=(4, 7)$ has arrived. As the height of it is greater than the width, all tasks are moved left as shown in Fig. 2. *c-list* now contains $(1, 1)$, $(2, 3)$, $(3, 4)$. After the movements, the system finds a submesh $\langle 7, 1, 10, 7\rangle$ which is large enough to accommodate $T=(4, 7)$. Instead of $T=(4, 7)$, assume that $T=(7, 3)$ was arrived. Then all tasks are moved down as shown in Fig. 11. *r-list* is updated as $(2, 1)$, $(1, 1)$, $(3, 1)$. The system can find a submesh $\langle 3, 6, 9, 8\rangle$ large enough to accommodate $T=(7, 3)$. If the incoming task was $T=(7, 5)$, even with the movements of Fig. 11, the required size submesh is not available yet. Thus all the tasks are then moved left as shown in Fig. 12. Now the system finds a submesh $\langle 4, 4, 10, 8\rangle$. As mentioned above, these are logical operations for finding the submesh mapping of each task.

Now we will show a case in which more than one horizontal and one vertical movement occur. In Fig. 13 a, assume that $T=(10, 2)$ has arrived. Since no task can be initially moved down, left movement is initiated. First, Tasks 3 and 4 are moved left. Second, only Task 3 is moved down. Third, Task 4 is moved left. Next, Task 4 is moved down. Now the system can find a submesh $\langle 1, 7, 10, 8\rangle$ for accommodating $T=(10, 2)$ as shown in Fig. 13b. Here Tasks 1 and 2 are not moved. In this example two horizontal and two vertical movements occurred.

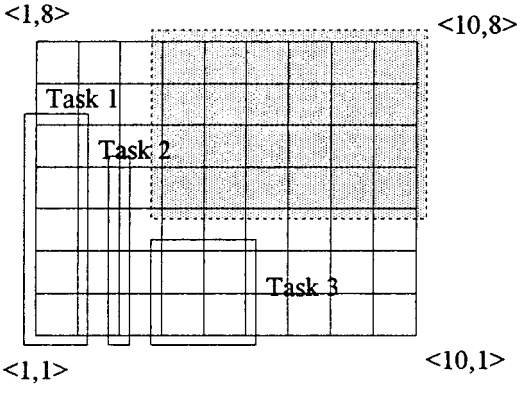

FIG. 12. The allocation of $T=(7, 5)$ after left movement.

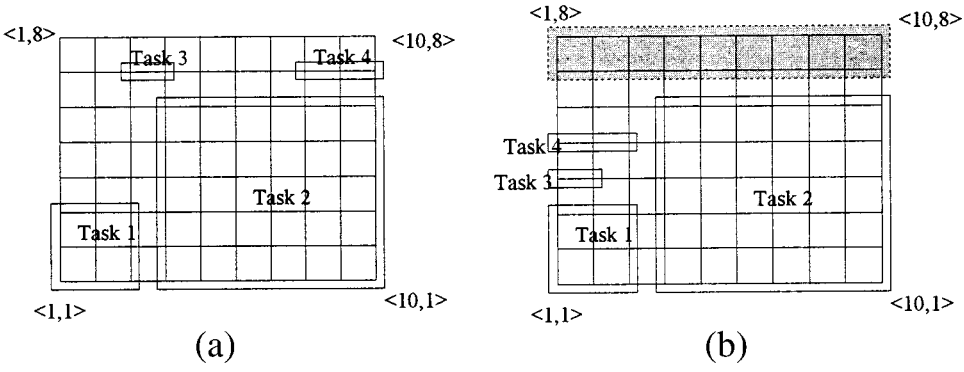

FIG. 13. (a) An allocation of $T = (10, 2)$ before task movement, (b) an allocation of $T = (10, 2)$ after task movement.

Note that only shifting is necessary for full relocation. Therefore, once the source and destination submeshes are determined from the submesh mapping step, node mapping and finally Algorithm-S presented in Section 3 are used for completing the relocation of the submeshes. The following is the procedure of full relocation for incoming task $T = (w, h)$.

PROCEDURE: FULL RELOCATION

- Step 1: /* Step 1–4: Decision of relocatability */
  direction = (down, left)
  list = (*c-list*, *r-list*)
  if $w > h$
  then *direction* ← down and *list* ← *r-list*
  else *direction* ← left and *list* ← *c-list*
  prev-move ← true
- Step 2: Move all tasks toward *direction* using *list*. If no task has been moved, go to Step 4.
- Step 3:
  Check whether a submesh can be allocatable to the incoming task.
  /*use original allocation scheme*/
  If allocatable
  then go to Step 5
  else *prev-move* ← true, change *direction* and *list*, and go to Step 2.
- Step 4:
  If *prev-move* is true
  then *prev-move* ← false, change *direction* and *list*, and go to Step 2
  else go to Step 6.
- Step 5: Node-mapping; Task movement by shifting; Allocation of the submesh to the incoming task. Stop.
- Step 6: No relocation can be made. Stop.

It is intuitively clear that the proposed interleaved compaction scheme always results in a larger submesh than either column-wise-only compaction or row-wise-only

compaction. The following theorem determines the relocation time of the full relocation scheme.

THEOREM 4.1. *Suppose that n tasks, T_1, T_2, T_3, ..., T_n, are relocated. Let RT_i be the relocation time of T_i with no contention. Then max (RT_1, RT_2, ..., RT_n) is the time for completing the full relocation.*

Proof. If no contention occurs between any two tasks, the proof is trivial because all tasks are relocated in parallel. Thus we need to show the case when contentions occur among the tasks. In the full relocation scheme, each submesh is mapped into a distinct submesh, and the tasks are relocated through the shifting operation. Recall that the shifting operations in Algorithm-S are all downward first and then leftward while the source and destination submesh pairs are all disjoint. Refer to Fig. 14, where $n = 2$. Note that no more than two submeshes can overlap at the same time. Let $S_{S1}\langle x_{s1}, y_{s1}, x'_{s1}, y'_{s1}\rangle$ and $S_{D1}\langle x_{d1}, y_{d1}, x'_{d1}, y'_{d1}\rangle$ be the source and destination submeshes for T_1, respectively. Similarly, $S_{S2}\langle x_{s2}, y_{s2}, x'_{s2}, y'_{s2}\rangle$ and $S_{D2}\langle x_{d2}, y_{d2}, x'_{d2}, y'_{d2}\rangle$ are for T_2. In our scheme, a contention occurs when a task changes the direction of its movement from downward to leftward while the other task continuously moves leftward and the two intermediate submeshes for the two tasks become overlapped. Let $S_{ID1}\langle x_{id1}, y_{id1}, x'_{id1}, y'_{id1}\rangle$ and $S_{ID2}\langle x_{id2}, y_{id2}, x'_{id2}, y'_{id2}\rangle$ be the intermediate submeshes for T_1 and T_2, respectively, when a contention just occurs. Without the loss of generality, assume that $RT_1 < RT_2$. Then if RT'_1 and RT'_2 are the remaining relocation time of T_1 and T_2 respectively when the contention occurs, $RT'_1 < RT'_2$ since the tasks have spent the same time so far. Therefore, T_2 has the higher priority, and T_1 should wait until T_2 passes, i.e., until S_{ID1} does not overlap S_{ID2} any more. The overlap occurs if either $x_{id1} \leqslant x_{id2} \leqslant x'_{id1}$ or $x_{id1} \leqslant x'_{id2} \leqslant x'_{id1}$. In either case, the maximum contention time (overlapping time) is $x'_{id2} - x_{id1} + 1$ which represents the time for the right-hand side of T_2 passes the left-hand side of T_1. Now we show that $RT'_1 + (x'_{id2} - x_{id1} + 1) \leqslant RT'_2$. $RT'_1 + (x'_{id2} - x_{id1} + 1) = (x_{id1} - x_{d1}) + (x'_{id2} - x_{id1} + 1) = x'_{id2} - (x_{d1} - 1)$. Since $RT'_1 < RT'_2$, $x_{d1} - 1 \geqslant x'_{id2}$. Therefore, $x'_{id2} - (x_{d1} - 1) \leqslant x'_{id2} - x'_{d2} = RT'_2$. Consequently, the task requiring the longer routing time determines the time for the full relocation.

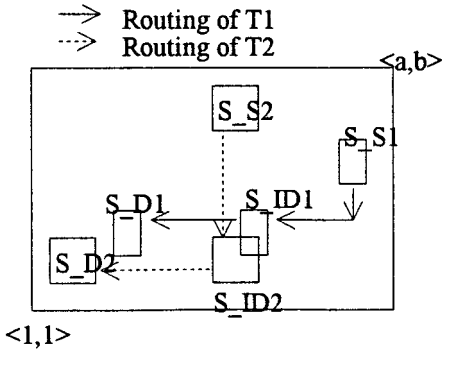

FIG. 14. Contention between two tasks.

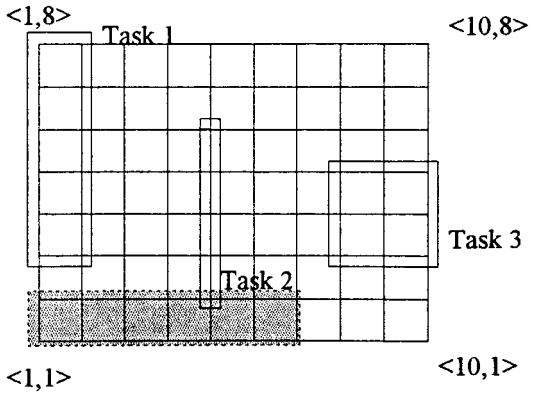

FIG. 15. Range of relocation array for $T=(4, 7)$.

4.2. Partial Relocation

In full relocation, all tasks in the mesh are involved. Thus some tasks may be relocated unnecessarily. For example, in Fig. 1, assume that $T=(4, 7)$ has arrived. To find a free submesh for T, full relocation has relocated Task 2 (5 processes) and Task 3 (9 processes) as shown in Fig. 2. However, the system can find a free submesh if only Task 1 is relocated as shown in Fig. 16b. The basic idea of partial relocation is that tasks are relocated such that the number of processes relocated is minimized. For this, *relocation array* is manipulated for finding such submesh.

DEFINITION 9. In a mesh $M(a,b)$, for every incoming task $T=(w, h)$ requiring relocation of already allocated submesh, S_A, a relocation array R is formed. Here, $R\ [i,j]\ (1\leqslant i\leqslant a-w+1, 1\leqslant j\leqslant b-h+1)$ represents the total number of processors required to be relocated when T is allocated to candidate submesh $S_C\langle i, j, i+w-1, j+h-1\rangle$.

Refer to Fig. 15, and assume that $T=(4, 7)$ arrives. Observe that the shaded region denotes the nodes on which the base of the submesh for T can be put. If the task is allocated to candidate submesh $\langle 2, 1, 5, 7\rangle$, Task 1 and 2 should be relocated. Therefore, $R\ [2, 1]$ is 17 because the area of submesh $\langle 1, 3, 2, 8\rangle$ for Task 1 is 12 and the area of submesh $\langle 5, 2, 5, 6\rangle$ for Task 2 is 5. The relocation array for T is thus as follows.

$$R=\begin{bmatrix} 12 & 17 & 5 & 5 & 14 & 9 & 9 \\ 12 & 17 & 5 & 5 & 14 & 9 & 9 \end{bmatrix}$$

R is set up as explained below.

PROCEDURE: CONSTRUCT-R

$R[i,j]\ (1\leqslant i\leqslant a-w+1, 1\leqslant j\leqslant b-h+1)\leftarrow 0.$
For each i
    For each j
        Construct S_C for T
        For each S_A

If S_A overlaps with S_C
Add area of S_A to $R[i, j]$

Once the relocation array is constructed, the base $\langle i, j \rangle$ is chosen for which $R[i, j]$ is minimum. If $R[i, j] \geqslant wh$, no relocation is allowed. This is for guaranteeing that any task which is larger than or equal to the incoming task is not relocated. If there exist multiple $\langle i, j \rangle$'s with the same minimal values, choose the $\langle i, j \rangle$ of the smallest value of $i+j$. The purpose of this is to minimize the fragmentation by putting the submesh at one side (actually the lower left side).

For the example of Fig. 15 and the relocation array above, the minimum value is 5, which is smaller than 28 (the size of the incoming task). Any of the four nodes, $\langle 3, 1 \rangle$, $\langle 4, 1 \rangle$, $\langle 3, 2 \rangle$, or $\langle 4, 2 \rangle$, can be the base, but $\langle 3, 1 \rangle$ is chosen as explained above. Fig. 16a shows that submesh $\langle 3, 1, 6, 7 \rangle$ is selected to be allocated to the incoming task. Task 2 then needs to be relocated. Using the original task allocation scheme, a submesh $\langle 7, 1, 7, 5 \rangle$ can be allocated to Task 2 as shown in Fig. 16b. Thus, the submesh $\langle 7, 1, 7, 5 \rangle$ is the destination for Task 2 to be relocated. As can be seen in this example, when all the tasks overlapped with the incoming task can be successfully allocated to other submeshes using the original task allocation scheme, then the actual relocation of them is started. If the allocation is not possible, the relocation is not allowed. When the submesh mapping is finished for the overlapped submeshes, the subsequent operation is exactly the same as for the full relocation scheme. The only difference here is that rotating is also required in addition to shifting when the orientations of the source and destination submeshes are different. The following is the procedure for partial relocation for an incoming task $T = (w, h)$.

PROCEDURE: PARTIAL RELOCATION

- Step 1. $flag \leftarrow false$. /* the flag representing the orientation */
- Step 2. Decide the orientation of T as follows. If ($flag = false$), then $T \leftarrow T(w, h)$, else $T \leftarrow T(h, w)$.
- Step 3. Based on current S_A's and T, construct R as explained above.
- Step 4. Choose the base of the candidate submesh as explained above.
  If minimum $R[i, j] \geqslant wh$
  if ($flag = false$)
  then $flag \leftarrow true$ and go back to Step 2
  else go to Step 7.
- Step 5. Submesh mapping of the submeshes overlapped with the incoming task using the allocation mechanism. If any of them cannot be mapped, go to Step 7.
- Step 6. Node-mapping; Task movement by shifting and/or rotation; Allocation of the submesh to the incoming task. Stop.
- Step 7. No relocation is allowed. Stop.

We next evaluate the performance of the proposed schemes.

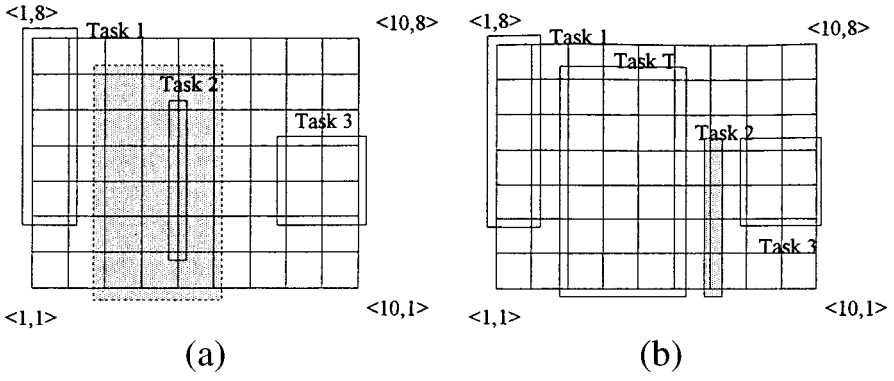

FIG. 16. An allocation for $T = (4, 7)$: (a) before relocation, (b) after relocation.

5. PERFORMANCE EVALUATION

In this section, the proposed schemes are evaluated in terms of task completion time and processor utilization. The study is done first by evaluating the time complexity of the proposed schemes. Then computer simulation evaluates the performance. We also compare our schemes with a well-known task allocation scheme [10].

5.1. Time Complexity of Relocation

Assume that N_A is the number of allocated submeshes in $M(a, b)$. The time complexity of the task allocation of AS scheme is $O(abN_A)$ and that of QA scheme is $O(bN_A)$ as analyzed in [12].

First, we analyze the time complexity of full relocation scheme. Clearly Steps 1, 4, and 6 take only $\Theta(1)$, respectively. Step 2 takes $O(N_A)$ because all allocated submeshes are moved in the worst case. Step 3 takes $O(bN_A)$ when the QA scheme is used as the original allocation scheme. Step 5 takes $O(ab)$ because ab is the number of nodes to be mapped and shifted in the worst case. Since Step 5 dominates, the time complexity of full relocation scheme is $O(ab)$.

Next, we analyze the time complexity of partial relocation scheme. Steps 1, 2, and 7 take only $\Theta(1)$, respectively. Step 3 takes $O(abN_A)$ to construct R because $i = a$ and $j = b$ in the worst case. Step 4 takes $O(ab)$ to choose the best base. Step 5 takes $O(bN_A^2)$ when the QA scheme is used as the original allocation scheme because each extracted task should check the possibility to find a free submesh elsewhere and the number of extracted task is N_A in the worst case. Step 6 takes $O(ab)$ because ab is the number of nodes to be mapped and shifted in the worst case. Since Step 5 dominates, the time complexity of partial relocation scheme is$O(bN_A^2)$. Since $N_A = ab$ in the worst case, the time complexity of partial allocation scheme in the worst case is $O(a^2b^3)$.

Note that the time complexity of full relocation scheme is much lower than that of partial relocation scheme.

5.2. Simulation Result

In this subsection the proposed two schemes are evaluated by computer simulation for various operational conditions. Since the AS scheme [10] is one of the

most efficient allocation schemes, it is employed as the allocation scheme. Simulations are conducted for the meshes ranging from 4×4 to 64×64. All the simulation use 95% confidence level with an error range of $\pm 3\%$. The simulator was developed in C language running on a Sun 4/490. Two different relocation schemes are studied; the full relocation scheme with the parallel task-relocation (FR-p) and the partial relocation scheme with the sequential task-relocation (PR-s). Notice that, due to the possibility of contention, sequential movement is assumed for the partial relocation.

The time for a task relocation [14] consists of the time for message startup, suspending/resuming the tasks, synchronization, forwarding in-transit messages, and context switch. The time for suspending/resuming a task and context switch is negligible compared to the time for actual task relocation. The relocation time is thus determined by the routing time from the original to the new sites which is obtained by multiplying the routing distance and the time for one position movement. As mentioned earlier, the time for one position movement is machine dependent, which is obtained by the product of the relocation overhead factor (α) and the residence time. The performance of our relocation schemes is evaluated for several different values of α as others do [14]. The relocation time is added to the task service time (residence time) when the overall performance is evaluated. The relocation may result in the increase of the performance by decreasing the external fragmentation, or degradation because of the excessive relocation overhead.

We employ the same simulation model used in [8, 10, 12, 14, 15]. Task allocation is carried out by a separate processor which functions as a task dispatcher. Initially the entire mesh is free, and 1000 tasks are generated and queued at the task dispatcher. Each task has a residence time requirement, which is assumed to be uniformly distributed between 5 to 10 time units. The tasks are assumed to arrive at each time unit. The time unit is large enough such that the time needed for task dispatcher to scan the whole mesh plane is negligible. The side lengths (width or height) of incoming tasks are assumed to follow one of the four distributions: uniform, exponential, decreasing, and increasing. For the uniform distribution, the side lengths of incoming tasks are uniformly distributed between 1 and the side length of the mesh (L). For the exponential distribution, the mean is selected as the half of L. Those values outside the range [1, side length + 1) were discarded. For the decreasing distribution, the probability that a side length of an incoming task falls into the range $[1, L/8]$ is 0.4, $[L/8+1, L/4]$ is 0.2, $[L/4+1, L/2]$ is 0.2, and $[L/2+1, L]$ is 0.2. For example, the probability that a side length falling into the range $[s_1, s_2]$ of $M(16, 16)$, denoted as $P_{[s1, s2]}$ is distributed as follows; $P_{[1, 2]} = 0.4$, $P_{[3, 4]} = 0.2$, $P_{[5, 8]} = 0.2$, and $P_{[9, 16]} = 0.2$. For the increasing distribution, the distributions are the opposite of the decreasing distribution. For $M(16, 16)$, again, $P_{[1, 8]} = 0.2$, $P_{[9, 12]} = 0.2$, $P_{[13, 14]} = 0.2$, and $P_{[15, 16]} = 0.4$. The widths and heights of tasks are generated separately based on the above distributions.

The task dispatcher is assumed to follow the First-Come-First-Serve (FCFS) discipline, i.e., the dispatcher always tries to find a free submesh for the first task in the queue. If it fails to find a free submesh, the dispatcher simply waits for a submesh to be released to allow the allocation. After a task is assigned to a submesh, it is removed from the queue and the next task in the queue is served in

the next time unit. Simulation is done for various values of α from 0.01 to 0.05. For every relocation, the routing time between the two submeshes is computed. Then the relocation time, the routing time multiplied by α, is added to the remaining residence time of the task.

We collect the following five performance metrics: (i) the allocation completion time (T_c), (ii) the average processor utilization over T_c, (iii) the number of relocations done, (iv) average number of tasks relocated per relocation occurred, and (v) average size of relocated task in terms of the number of processors. Table III compares the proposed relocation schemes and AS scheme under various values of α. Note that $\alpha = 0$ for AS scheme since it does not involve any relocation. From it, we observe the following.

- FR-p outperforms AS scheme (no relocation) when $\alpha \leqslant 0.03$. It means that the relocation schemes are effective only for smaller relocation overhead factors.

TABLE III

Performance Comparisons under Various Values of α of $M(16, 16)$ Scheduling: FCFS, $T = 1000$

Side length distribution	Performance measure	Scheme	α = 0	0.01	0.02	0.03	0.04	0.05
Uniform	Task completion time	AS	3482					
		FR-p		3365	3394	3421	3449	3476
		PR-s		3343	3410	3504	3584	3625
	Processor utilization (%)	AS	61.36					
		FR-p		63.39	62.85	62.35	61.85	61.36
		PR-s		63.81	62.56	60.97	59.62	58.85
Exponential	Task completion time	AS	1740					
		FR-p		1652	1688	1719	1749	1782
		PR-s		1650	1767	1905	2095	2254
	Processor utilization (%)	AS	60.90					
		FR-p		64.11	62.75	61.64	60.58	59.45
		PR-s		64.19	59.96	55.63	50.59	46.99
Decreasing	Task completion time	AS	1235					
		FR-p		1163	1192	1226	1250	1287
		PR-s		1174	1352	1561	1749	2003
	Processor utilization (%)	AS	60.49					
		FR-p		64.23	62.70	60.93	59.75	58.09
		PR-s		63.65	55.25	47.88	42.74	37.31
Increasing	Task completion time	AS	5899					
		FR-p		5855	5867	5879	5891	5903
		PR-s		5851	5867	5882	5898	5914
	Processor utilization (%)	AS	69.91					
		FR-p		70.44	70.30	70.15	70.01	69.87
		PR-s		70.48	70.29	70.11	69.92	69.73

Note. Residence time distribution: Uniform [5, 10].

• Among the four distributions, the decreasing distribution is the most sensitive to the value of α, and the increasing distribution is the least.

• Compared to the full relocation scheme, the partial relocation scheme performs worse if $\alpha > 0.01$. This is because the relocation overhead is more significant than the reduced number of process relocation.

From Table IV, which studies the schemes for various size meshes when $\alpha = 0.01$, we observe the following.

• Our relocation schemes are relatively more effective for smaller size meshes. Notice that the performance difference between ours and AS scheme gets smaller as the size of mesh increases.

• The full relocation scheme outperforms the partial relocation scheme for relatively large meshes. This is because the relocation distance of partial relocation is longer than that of full relocation in the worst case.

TABLE IV
Performance Comparisons for Various Sizes When $\alpha = 0.01$
Scheduling: FCFS, T = 1000

Side length distribution	Performance measure	Scheme	Mesh size				
			4×4	8×8	16×16	32×32	64×64
Uniform	Task completion time	AS	3987	3625	3482	3358	3356
		FR-p	3940	3530	3365	3336	3288
		PR-s	3892	3467	3343	3343	3378
	Processor utilization (%)	AS	73.49	65.59	61.36	58.96	58.21
		FR-p	74.5	67.09	63.39	59.95	59.00
		PR-s	75.42	68.31	63.81	59.83	57.42
	Number of recolation occured	FR-p	30	54	70	76	79
		PR-s	55	90	96	102	106
	Number of tasks relocated	FR-p	1.16	1.35	1.39	1.39	1.50
		PR-s	1.05	1.15	1.20	1.20	1.21
	Size of tasks relocated	FR-p	3.85	12.65	45.48	178.61	701.95
		PR-s	2.58	7.90	27.53	110.73	391.04
Exponential	Task completion time	AS	2291	1864	1775	1677	1626
		FR-p	2207	1774	1652	1610	1638
		PR-s	2137	1700	1650	1692	1961
	Processor utilization (%)	AS	74.66	66.20	60.67	57.58	56.13
		FR-p	77.48	69.55	64.11	59.97	55.73
		PR-s	80.03	72.57	64.19	57.03	46.55
	Number of relocations occured	FR-p	67	93	103	113	122
		PR-s	135	177	171	174	187
	Number of task relocated	FR-p	1.61	2.04	2.31	2.49	2.82
		PR-s	1.15	1.35	1.40	1.51	1.51
	Size of tasks relocated	FR-p	2.65	7.84	28.61	109.55	401.08
		PR-s	1.86	4.77	16.64	61.97	233.22

Note. Residence time distrubution: Uniform [5, 10].

• FR-p requires fewer relocations than PR-s. This is expected since the partial relocation scheme does not fully compact the array.

• As expected the average number of relocated tasks per relocation and the size of tasks relocated of partial relocation are much smaller than that of full relocation.

To check the effect of the side lengths in the two relocation schemes, we have simulated two more cases. In the first case, all requests are squares whose side lengths are power of two and uniformly distributed. For example, in $M(16, 16)$, side length is 1, 2, 4, 8, or 16, and those five numbers are uniformly distributed. In the second case, all requests are rectangles whose side lengths are power of two and also uniformly distributed. The other conditions are same as the previous simulations. However, in these two cases, we do not observe any particular difference from the above-mentioned observations.

In addition, another simulation is conducted in order to study the external fragmentation. The simulation results for mesh systems of various sizes follow a similar trend, and thus we report the simulation results for only the 16×16 mesh system. The simulation uses a 90% confidence level with an error range of $\pm 5\%$. Task residence time and interarrival time are assumed to have the exponential distribution with the means of MTRT (mean task residence time) and MIAT (mean task interarrival time), respectively. Here *system load* is defined as $(n \times \text{MTRT})/(N \times \text{MIAT})$ where n is the average size of the requested submesh in terms of the number of processors and N is the total number of processors in the 2D mesh system. In the simulation, we fix MTRT to be 7.5 time units and adjust MIAT according to the desired load.

Figure 17 plots the external fragmentation of the studied relocation schemes under different workloads of $M(16, 16)$ when $\alpha = 0.01$ for exponential distribution of side lengths. We observe the following from the simulation result.

• Regardless of the system load, both relocation schemes decrease the external fragmentation. For example, under the system load of 0.5, the external fragmentation

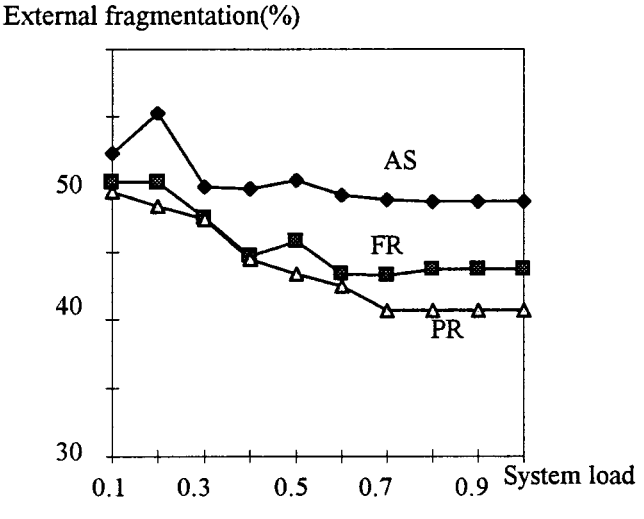

FIG. 17. External fragmentation versus offered system load.

without relocation is 50.2%, whereas those under the full relocation and the partial relocation are 45.8% and 43.4%, respectively.

- As expected, the partial relocation scheme decreases the external fragmentation more than the full relocation scheme. This agrees with the information in Table IV, where the number of relocations occurred in the partial relocation scheme is much higher than that in the full relocation scheme.

6. CONCLUSION

In this paper, we have presented two relocation schemes—*full relocation* and partial relocation—for enhancing the performance of 2D mesh architecture by reducing external fragmentation. The partial relocation scheme tries to minimize the relocated tasks while the full relocation scheme involves simple shift operations of all tasks. For general relocation processes, we also developed and modeled two basic submesh relocation operations—shifting and rotating. Simulation results show that both schemes increase the performance of the system in terms of task completion time and system utilization up to a certain value of relocation overhead factor which is machine dependent. The full relocation scheme demonstrated better performance than the partial relocation scheme for relatively large meshes and relocation overhead. The relocation scheme is also useful when high–priority jobs arrive but are not allocatable in a real-time environment. The performance of a mesh system is expected to be further enhanced if every task allocation is determined such that the external fragmentation is less likely. This issue is currently being investigated.

REFERENCES

1. P. Muzumdar, Evaluation of on-chip static interconnection networks, *IEEE Trans. Comput.* **36** (March 1987), 365–369.
2. G. Randade and S. L. Johnsson, The communication efficiency of meshes, boolean cubes and cube connected cycles for wafer scale integration, *in* "Proc. of the International Conf. on Parallel Processing," pp. 477–482, Aug. 1987.
3. "Paragon XP/S Product Overview," Intel Corporation, 1991.
4. "A Touchstone DELTA System Description," Intel Corporation, 1991.
5. J. Kim, C. R. Das, and W. Lin, A top-down processor allocation scheme for hypercube computers, *IEEE Trans. Parallel Distrib. Systems* **2** (Jan. 1991), 21–30.
6. P. J. Chuang and N. F. Tzeng, A fast recognition-complete processor allocation strategy for hypercube computers, *IEEE Trans. Comput* **41**, 4 (April 1992), 467–479.
7. K. Li and K. H. Cheng, A two-dimensional buddy system for dynamic resource allocation in a partitionable mesh connected system, *J. Parallel Distrib. Comput.* **12** (May 1991), 79–83.
8. P. J. Chuang and N. F. Tzeng, An efficient submesh allocation strategy for mesh computer systems, *in* "Proc. of the International Conf. on Distributed Computing Systems," pp. 256–263, Aug. 1991.
9. Y. Zhu, Efficient processor allocation strategies for mesh-connected parallel computers, *J. Parallel Distrib. Comput.* **16** (Dec. 1992), 328–337.
10. J. Ding and L. N. Bhuyan, An adaptive submesh allocation strategy for two-dimensional mesh connected systems, *in* "Int'l Conf. on Parallel Processing," pp. II-193–200, Aug. 1993.

11. D. D. Sharma and D. K. Pradhan, Job scheduling in mesh multicomputers, *IEEE Trans. Parallel Distrib. Systems* (Jan. 1998), 57–70.

12. S. M. Yoo, H. Y. Youn, and B. Shirazi, An efficient task allocation scheme for 2D mesh architectures, *IEEE Trans. Parallel Distrib. Systems* (Sep. 1997), 934–942.

13. V. Lo, K. J. Wndish, W. Liu, and B. Nitzberg, Non-contiguous processor allocation algorithms for mesh-connected multicomputers, *IEEE Trans. Parallel Distrib. Systems* (July 1997), 712–726.

14. C. H. Huang and J. Y. Juang, A Partial compaction scheme for processor allocation in hypercube multiprocessors, *in* "Int'l Conf. on Parallel Processing," pp. I-211–217, 1990.

15. M. S. Chan and K. G. Shin, Subcube allocation and task migration in hypercube multiprocessors, *IEEE Trans. Comput.* **39**, 9 (Sep. 1990), 1146–1155.

16. K. Hwang and H. Xu, "Scalable Parallel Computing: Technology, Architecture, Programming," WCB/McGraw–Hill, Boston, 1998.

SEONG-MOO YOO received the B.A. in economics from Seoul National University, Seoul, Korea, and the M.S. and Ph.D. in computer science from The University of Texas at Arlington in 1989 and 1995, respectively. He is currently an assistant professor in the Department of Computer Science, Columbus State University, Columbus, Georgia. His research interests include parallel computing, multiprocessor systems, computer security, and mobile computing. Dr. Yoo is a member of the Association for Computing Machinery and the IEEE Computer Society.

HYUNSEUNG CHOO received the B.S. in mathematics from Sungkyunkwan University, Suwon, Korea in 1988, the M.S. in computer science from the University of Texas at Dallas, Richardson, TX in 1990, and the Ph.D. in computer science from the University of Texas at Arlington (UTA), Arlington, TX in 1996. In 1997, he was a faculty associate of the Department of Computer Science and Engineering at UTA. From 1997 to 1998, he was a patent examiner at Korean Industrial Property Office, Seoul, Korea. Dr. Choo is currently an assistant professor of the School of Electrical and Computer Engineering at Sungkyunkwan University, Suwon, Korea. His research interests include networking, ATM switching, system performance evaluation, parallel and distributed computing, and design of algorithms.

HEE YONG YOUN received the B.S. and M.S. in electrical engineering from Seoul National University, Seoul, Korea, in 1977 and 1979, respectively, and the Ph.D. in computer engineering from the University of Massachusetts at Amherst, in 1988. From 1979 to 1984, he was on the research staff of Gold Star Precision Central Research Laboratories, Korea. He is an associate professor in the Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington, Texas. He is currently on a leave of absence to Information and Communications University, Korea. He received the Outstanding Paper Award from the 1988 International Conference on Distributed Computing Systems and 1992 Supercomputing, respectively. He also served as a lecturer of the ACM Lectureship Series from 1993 to 1997. His research interests include parallel and distributed computing, mobile computing, performance modeling and evaluation, and fault-tolerant computing. Dr. Youn is a senior member of the IEEE Computer Society.

CHANSU YU received the B.S. and M.S. in electrical engineering from Seoul National University, Seoul, Korea in 1982 and in 1984, respectively. He worked as a research engineer at GoldStar Company until 1989. He received the Ph.D. in computer engineering from the Pennsylvania State University in 1994. Since 1997, he has been an assistant professor with the School of Engineering, Information and Communications University, Taejon, Korea. His areas of interest are computer architecture, parallel and cluster computing, performance evaluation, and mobile systems. Dr. Yu is a member of the IEEE and IEEE Computer Society.

YOUNGHEE LEE received B.S. and M.S. in electronics from Seoul National University, Korea, in 1976 and 1980, respectively and Ph.D. in computer science from Université de Technologie de Compiegne, France in 1984. Since he joined Electronics and Telecommunications Research Institute (ETRI), Korea in 1984, he has been working on the fields of high speed networks, CCS systems, ATM, gigabit networks, and next generation Internet. He is currently Professor of Information and Communications University in Korea. He also serves as one of the vice chairmen of SG 7 in ITU-T. His research interests include parallel processing, next generation Internet technologies, and high-speed networks.