태스크와 서브메쉬의 유형별 분류에 기반한 서브메쉬 할당방법

이 원 주† · 전 창 호††

요 약

본 논문에서는 메쉬 구조 다중컴퓨터 시스템을 위한 새로운 서브메쉬 할당방법을 제안한다. 이 할당방법의 특징은 유형별 가용 서브메쉬 리스트를 사용하여 가용 서브메쉬의 탐색시간을 줄이는 것이다. 이 할당방법은 메쉬 구조에서 탐색한 독립 가용 서브메쉬들을 유형(정방형, 가로 직사각형, 세로 직사각형)에 따라 분류하여 유형별 가용 서브메쉬 리스트를 생성한다. 그리고 태스크의 유형과 동일한 유형별 가용 서브메쉬 리스트에서 먼저 최적의 서브메쉬를 찾음으로써 서브메쉬의 탐색시간을 줄인다. 만약 가용 서브메쉬를 찾지 못할 경우에는 각 독립 가용 서브메쉬의 속성으로 저장된 확장지수를 사용하여 서브메쉬의 크기를 확장한 후 최적의 가용 서브메쉬를 찾는다. 시뮬레이션을 통하여 본 논문에서 제안하는 서브메쉬 할당방법이 서브메쉬 탐색시간을 줄이는 면에서 기존의 할당방법에 비해 우수함을 보인다.

A Submesh Allocation Scheme Based on Classification of Tasks and Submeshes

Won Joo Lee† · Chang Ho Jeon††

ABSTRACT

This paper presents a new submesh allocation scheme for mesh-connected multicomputer systems. The key idea in the proposed allocation scheme is to reduce the submesh search time using classified free submesh lists (CFSL). This scheme reduces the submesh search time by classifying independent free submeshes according to their types (square, horizontal rectangle, vertical rectangle) and searching the best-fit submesh from the classified free submesh list. When no suitable submesh is found, the search can be continued by using the expansion index (EI), which is stored as an attribute of each submesh, is used to form a larger submesh. Through simulation, we show that the proposed strategy improves the performance compared to previous strategies with respect to submesh search time.

키워드 : 독립 가용 서브메쉬(IFS : Independent Free Submesh), 확장지수(EI : Expansion Index), 유형별 가용 서브메쉬 리스트(CFSL : Classified Free Submesh List)

1. Introduction

Mesh structure has been widely used in commercial or prototype multicomputer systems like Intel Touchstone Delta [1], Intel Paragon XP/S [2], Tera Computer system [3], Fusitsu AP1000 [4], Sanyo Edden/Cyberflow system [4], MP-1 [5], Parsytec GC [5], Data Transport computer [5], and PSAM [6], because of its simplicity, regularity, and expandability.

For better performance of multicomputer system, a submesh allocation scheme is needed, which can search for the best-fit submesh and allocate it to a task which can request submeshes of various sizes. Submesh allocation scheme is one that generates free submeshes by searching from the entire mesh structure and allocates the best-fit submesh to a given task. If it fails to find an available submesh, task should wait in a queue ; as a consequence, other tasks are delayed, which results in the performance deterioration of the multicomputer system.

The previous research efforts concerning submesh allocation for mesh structure have been mainly focused on searching for the best-fit submesh and reducing the submesh allocation delay. As a result, various submesh allocation schemes [7-15] have been proposed. These submesh allocation schemes can be classified as first-fit and best-fit allocation. In the first-fit allocation scheme [8, 9], a submesh is assigned immediately when it is found. The first-fit allocation scheme is simple and takes shorter submesh search time, but it can have a severe fragmentation and poor performance in comparison with the best-fit allocation. The

※ This work is supported by Doowon Technical College.
† 정 회 원 : The author is with the Department of Internet Programming, Doowon Technical College, Korea.
†† 종신회원 : The author is with the School of Electrical and Computer Engineering, Hanyang University, Korea.
논문접수 : 2003년 7월 5일, 심사완료 : 2003년 10월 7일

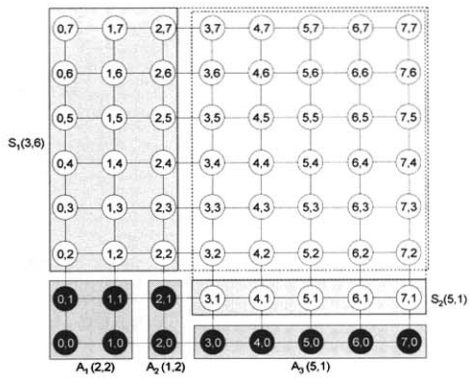
best-fit allocation scheme [7, 10-15] makes a list of free submeshes by searching over a mesh structure and allocates the best-fit submesh from the list for a given task. This scheme has less external fragmentation and better performance compared to the first-fit allocation scheme. In the best-fit allocation scheme, submesh allocation algorithm is complex and submesh search time can be long. Also, previously known best-fit allocation strategies search over the entire free submesh list to find the best-fit submesh for a given task. This approach has a problem that the submesh search time increases as the number of free submeshes increases.

In this paper, we propose a new submesh allocation scheme which improves the performance of multicomputer systems. The key idea of this allocation scheme is to reduce free submesh search time of task by using the classified free submesh list (CFSL). This scheme reduces the submesh search time by classifying independent free submeshes according to their types (square, horizontal rectangle, vertical rectangle) and searching the best-fit free submesh from the classified free submesh list. If it fails to find a free submesh, the size of each independent free submesh is expanded using expansion index (EI) which is stored as an attribute of each independent free submesh and the best-fit free submesh is chosen. Through simulation, we show that the proposed strategy improves the performance compared to the previous strategies with respect to the free submesh search time.

The remainder of this paper is organized as follows : Section II presents two-dimensional mesh structure and submesh. In Section III, the previous strategies about submesh allocation are introduced. The proposed submesh allocation scheme is presented in Section IV. In Section V, the performance of the proposed scheme is evaluated by computer simulation for various practical operational conditions. Finally, the conclusion is given in Section VI.

2. Two-Dimensional Mesh Structure

A two-dimensional mesh, M(W, H), is a W×H rectangular grid consisting of W×H nodes, where W and H represent the width and height, respectively. Each node in the mesh refers to a processor. In this paper, address of a node is denoted by <x, y>, where $0 \le x \le W-1$ and $0 \le y \le H-1$. A node <x, y> is located on the coordinate position <x, y> on the base of the lowest-leftmost position <0, 0> of the mesh. An allocated node refers to a node that is allocated to a task and executing it. A free node refers to a node that is not allocated to a task.



(Figure 1) Example of a two-dimensional mesh M(8, 8)

In (Figure 1), allocated nodes are depicted as shaded circles and free nodes are depicted as empty circles. In M(W, H), a submesh denoted as S(w, h) is a rectangular lattice consisting of w×h nodes, where $1 \le w \le W-1$ and $1 \le h \le H-1$. A submesh is identified as S(<x_1, y_1>, <x_2, y_2>), where <x_1, y_1> and <x_2, y_2> are the addresses of lower- leftmost node and upper-rightmost node of the submesh, respectively. An allocated submesh is one that consists of allocated nodes and a free submesh is one that consists of free nodes. A free submesh is denoted as S_i(w, h) or S_i(<x_1, y_1>, <x_2, y_2>) and an allocated submesh are denoted as A_j(w, h) or A_j(<x_1, y_1>, <x_2, y_2>). In (Figure 1), allocated submeshes are A_1(<0, 0>, <1, 1>) and A_2(<2, 0>, <2, 1>), and free submeshes are S_1(<0, 2>, <2, 7>) and S_2(<3, 1>, <7, 1>).

3. Previous Submesh Allocation Strategies

3.1 First-fit allocation strategies

First-fit processor allocation allocates a free submesh whenever it searches one from the entire mesh structure.

Frame Sliding (FS) allocation scheme [8] improved the 2DB allocation scheme [7]. So, it can be applied to meshes having rectangular structure or tasks size that is not 2^n. This scheme solves the problem of internal fragmentation by finding and allocating the proper free submesh of each task size but still has the problem that it can't recognize free submeshes completely. The time complexity of this scheme is O(NB), where N is the number of all processors and B is the number of allocated submeshes.

Adaptive Scan (AS) allocation scheme [9] improved the FS allocation scheme by rotating the orientation of incoming task from T(w, h) to T(h, w) and therefore reduced the external fragmentation. It is same as the FS allocation strategy in that it moves towards x-axis as much as the width of frame to search for a free submesh. But the movement in the direction of y-axis is made by one, which makes the

rate of recognition of free submesh high unlike the FS allocation strategy where the movement is made by as much as the height of frame. The time complexity of this scheme is $O(N)$.

3.2 Best-fit allocation strategies

Best-fit allocation makes a free submesh list by using free submeshes after searching over the entire mesh structure. It searches for the best-fit submesh for a given task and allocates it to the task.

Two-Dimensional Buddy (2DB) allocation strategy [7] was applied for a square mesh of which the size is 2^n. This strategy can have internal fragmentation because it allocates only square free submesh having size of 2^n to tasks of various sizes. Then, it is possible to assign more processors to task than is required. During the deallocation step, submesh-merging is necessary to form bigger submeshes. This process has a time complexity of $O(\log N)$ for the worst case.

First-Fit (FF)/Best-Fit (BF) allocation strategy [10] allocates a free submesh of exactly the same size as that of task to solve the internal fragmentation problem. This strategy uses allocation array and range array to allocate a free submesh. FF searches for a submesh that consists of nodes of which the range array values are zeros and allocates it to the task. BF chooses the biggest free submesh having the largest number of adjacent nodes. The submesh recognition rate of this strategy is not perfect because it fixes the orientation of the incoming task. Also, the management of both allocation array and range array results in a big overhead. The time complexity of FF and BF allocation is $\theta(N)$.

Adjacency (ADJ) allocation strategy [11] considers submeshes that are adjacent to one of four corners of submeshes that are already allocated. Boundary values of free submeshes are calculated and a submesh having the largest boundary value is chosen and is allocated to a given task. This strategy has an advantage that allocation time is constant, regardless of the mesh size and external fragmentation is reduced. The time complexity of this allocation algorithm is, in its worst case, $O(B^3)$, where B is the number of allocated submeshes.

Look-ahead (LKH) allocation strategy [12] selects the biggest task from the waiting queue. It predicts whether there exists a submesh which is enough for the biggest task chosen, if there exists a free submesh, the task is assigned to a submesh, otherwise, another free submesh is chosen and a next prediction is made. The time complexity of this prediction is $O(1)$. To save the intermediate results when searching for a free submesh, a space of $O(B)$ size is required. Time complexity of algorithm for free submesh is $O(B^2)$.

Free List (FL) allocation strategy [13] is to search over the free submesh list for a free submesh whose size is equal to or larger than that of an incoming task. A submesh in a free submesh list could be overlapped ; therefore after allocating a new task, free submesh list needs to be reconstructed. The number of free submeshes in a free submesh list can be increased, thus resulting in a significant overhead. The time complexity for both allocation and deallocation is $O(F^2)$, where F is the number of free submeshes.

Quick Allocation (QA) allocation strategy [14] reduced the submesh search time using an array that manages the states of each row in a mesh. QA does not search over the entire mesh structure. QA has a high recognition rate and a low time complexity but has a problem that the performance is dependent on the size of the mesh. Time complexity of QA is $O(hB)$, where h means the height of mesh.

Free Submesh List (FSL) allocation strategy [15] has a high submesh recognition rate and searchs for a submesh with which best-fit allocation is possible from the free submesh list. And to keep the size of free submesh as biggest as possible, FSL calculates a reservation index of each submesh and chooses a submesh having the largest reservation index value. Then, FSL allocates it to the task. The more the number of free submeshes is, the longer the allocation time becomes, because the frequency of calculating the reservation index increases. Also FSL needs to reconstruct the free submesh list after allocation and deallocation because the sizes of other free submeshes changes since allows overlapping among the free submeshes. The time complexity of this allocation algorithm is $O(F^2)$.

Previous best-fit allocation strategies search over the entire free submesh list to find the best-fit submesh for a given task. This approach has a problem that the submesh search time increases as the number of free submeshes increases. In this paper, we propose Classified Free Submesh List (CFSL) allocation scheme to reduce the free submesh search time. The free submeshes from the entire submesh structure are classified according to the corresponding types (square, horizontal rectangle, vertical rectangle), from which a CFSL is constructed. For a given task, by searching for the best-fit submesh of the same type with that of given task, submesh search time is reduced. When it fails to find a submesh, after the size of each submesh is expanded using expansion index, which is stored as an attribute of each submesh. Then, the best-fit submesh is chosen.

4. The Proposed Submesh Allocation Strategy

4.1 Basic Idea

Some basic terms will be defined prior to introducing the CFSL allocation scheme.

Definition 1 : *An Independent free submesh (IFS) is a free submesh of maximum size that does not overlap with other free submeshes.*

For example, in (Figure 1), S(<0, 2>, <7, 7>) and S(<3, 1>, <7, 7>) overlap in the submesh S(<3, 2>, <7, 7>). The overlapping submeshes are shown by using dotted rectangle. Excluding overlapped submeshes, we obtain S_1(<0, 2>, <2, 7>) and S_2(<3, 1>, <7, 1>). These two submeshes are IFS that is shown by using solid rectangle. Since IFSs don't overlap, allocation of specific IFS does not affect the size of other IFSs. Therefore, CFSL can be made without searching over the entire mesh structure by inserting submeshes that are divided after task allocation into their corresponding free submesh list. CFSL is defined as follows.

Definition 2 : *A Classified Free Submesh List (CFSL) is a set of IFSs, classified as square, horizontal rectangle or vertical rectangle, depending on the values of w and h of IFS.*

In this paper, we denote sets of square, horizontal rectangle, and vertical rectangle as SQ-flist, HR-flist, and VR-flist, respectively. Set of allocated submeshes is denoted as Alloc-list. IFS in CFSL are ordered in decreasing order according to its size. It is satisfied that SQ-flist ∩ HR-flist ∩ VR-flist = ∅. In (Figure 1), S_1(3, 6) is inserted into VR-flist and S_2(5, 1) is inserted into HR-flist. Alloc-list includes A_1 (<0, 0>, <1, 1>), A_2(<2, 0>, <2, 1>), and A_3(<3, 0>, <7, 0>).

A task is denoted as T(w, h), where w and h are width and height of the task, respectively. To execute T(w, h), it needs a free submesh that have node of w×h. T(w, h) is classified as square, horizontal rectangle, and vertical rectangle depending on h and w. In this paper, these types of tasks are used. That is, we reduce the submesh search time by searching for the best-fit submesh from the CFSL of the same type with that of a task. For example, if the type of a task is horizontal rectangle, HR-flist is searched over for the best-fit submesh.

Definition 3 : *An Expansion Index (EI) is a displacement value such that an IFS can be expanded as much as possible if overlapping is allowed.*

EI of an IFS S(<x_1, y_1>, <x_2, y_2>), is denoted as EI(S) = (<α, β>, <α', β'>). A free submesh, formed by overlapping S(<x_1, y_1>, <x_2, y_2>) with other free submesh is denoted as S'(<x_1', y_1'>, <x_2', y_2'>). EI(S) = (<α, β>, <α', β'>) is calculated using the lower-leftmost and upper-rightmost coordinate values of S(<x_1, y_1>, <x_2, y_2>) and S'(<x_1', y_1'>, <x_2', y_2'>). That is, <α, β> is <x_1'- x_1, y_1'- y_1> and <α', β'> is <x_2'- x_2, y_2'- y_2>

For example, in (Figure 1), IFS S_1(<0, 2>, <2, 7>) can form S_1'(<0, 2>, <2, 7>), by including S(<3, 2>, <7, 7>) which overlaps with other free submeshes. If we calculate EI value using the coordinate values of lower-leftmost and upper-rightmost nodes, <α, β> is <0-0, 2-2> and <α', β'> is <7-2, 7-7>. Therefore, EI[S_1(<0, 2>, <2, 7>)] = (<0, 0>,<5, 0>). In (Figure 1), T(8, 6) cannot be allocated either S_1(<0, 2>, <2, 7>) or S_2(<3, 1>, <7, 1>) and the allocation is delayed. S_1(<0, 2>, <2, 7>) is expanded using the EI(<0, 0>, <5, 0>) into S_1'(<0, 2>, <7, 7>). We can allocate S_1'(<0, 2>, <7, 7>) to the T(8, 6).

4.2 CFSL Submesh Allocation Algorithm

In the following, we propose a new submesh allocation scheme, called CFSL allocation scheme. (Figure 2) describes the CFSL submesh allocation algorithm.

```
CFSL_Allocation() // Request(Ti<w, h>)
{   //dispatch a task from waiting queue
  Ti = Dispatch_Qi(Ti<w, h>);
  Make_CFSL();      // construct CFSL
  Select_Submesh(); // select the best-fit free submesh step
  {
    switch (Ti type) {
      case type-S : // Square task
        for (i = a ; i <= p ; i++)
          • Search for the best-fit free submesh from
            SQ-flist={Sa,···,Sp};
        break ;
      case type-H : // Horizontal rectangle task
        for (i = b ; i <= q ; i++ )
          • Search for the best-fit free submesh from
            HR-flist={Sb,···,Sq};
        break ;
      case type-V : // Vertical rectangle task
        for (i = c ; i <= r ; i++ )
          • Search for the best-fit free submesh from
            VR-flist={Sc,···,Sr};
        break ;
    }
    if (best free submesh = ∅) {
      Waiting(); //wait until a free submesh becomes available
    }
  }
  Do_Allocate(); //submesh allocation step
  {
    Allocate(); //Allocate free submesh to a task
    Insert();   //insert the allocated submesh into Alloc-list
    Delete();   // Delete a free submesh from CFSL
  }
}
```

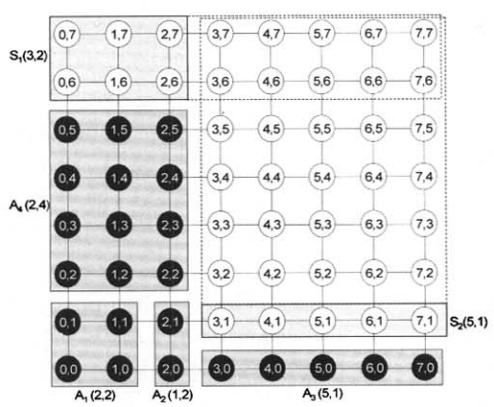
(Figure 2) CFSL submesh allocation algorithm

In Make_CFSL() module, CFSL is constructed according to the type of IFS. In Select_Submesh() module, it searches for the best-fit submesh with the same type as that of given task, which results in a reduced submesh search time. If it fails to find a free submesh, the size of each independent free submesh is expanded using expansion index (EI) which is stored as an attribute of each independent free submesh and the best-fit free submesh is chosen.

In this paper, CFSL submesh algorithm given in (Figure 2) is illustrated using three examples of submesh allocation.

The first example is searching the best-fit submesh from CFSL and allocating it to a task. In (Figure 1), if a task $T_4(3, 4)$ requiring a submesh of size(3×4), it would be able to be allocated a submesh. Make_CFSL() module generates CFSL according to the type of IFS that has been searched from the entire mesh structure. Alloc-list and CFSL generated from (Figure 1) are as follows :

- Alloc-list = { A_1(<0, 0>, <1, 1>), A_2(<2, 0>, <2, 1>), A_3(<3, 0>, <7, 0>) },
- SQ-flist = {∅},
- HR-flist = { S_2(<3, 1>, <7, 1>) },
- VR-flist = { S_1(<0, 2>, <2, 7>) }.

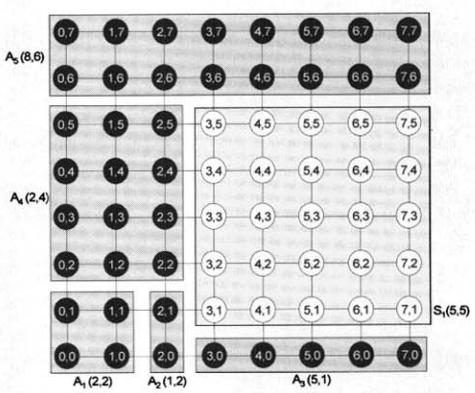


(Figure 3) The result of $T_4(3, 4)$ allocation

In Select_Submesh(), the best-fit free submesh is searched from CFSL according to the type of task. Since $T_4(3, 4)$ is a vertical rectangle, S_1(<0, 2>, <2, 7>) from VR-flist is chosen to be the best-fit submesh. In Do_Allocate() module, S_1(<0, 2>, <2, 7>) is allocated to $T_4(3, 4)$. After the allocation, the type of divided S_1(<0, 6>, <2, 7>) is horizontal rectangle. Therefore, it is inserted into HR-flist. A_4(<0, 2>, <2, 5>) is inserted into Alloc-list. The result of $T_4(3, 4)$ allocation is illustrated in (Figure 3). Alloc-list and CFSL generated from (Figure 3) are as follows :

- Alloc-list = { A_1(<0, 0>, <1, 1>), A_2(<2, 0>, <2, 1>), A_3(<3, 0>, <7, 0>) }, A_4(<0, 2>, <2, 5>) },
- SQ-flist = VR-flist = {∅},
- HR-flist = { S_1(<0, 6>, <2, 7>), S_2(<3, 1>, <7, 1>) }.

The second example is about expanding submesh size using expansion index and allocating it to a task. In (Figure 3), if a task $T_5(8, 2)$ requiring a submesh of size (8×2), it would not be able to be allocated a submesh right away. Since, there is no available free submesh that can accommodate the task in CFSL. To reduce this submesh allocation delay, our scheme selects S_1(<0, 6>, <2, 7>) from HR-flist as the best-fit submesh and expands S_1(<0, 6>, <2, 7>) into S_1'(<0, 6>, <7, 7>) using EI(S_1) = (<0, 0>, <5, 0>). And, Do_Allocate() module allocates S_1'(<0, 6>, <2, 7>) to $T_5(8, 2)$. The result of $T_5(8, 2)$ allocation is illustrated in (Figure 4).



(Figure 4) The result of $T_5(8, 2)$ allocation

In (Figure 4), newly generated submesh S_1(<3, 1>, <7, 5>) is a square type after the allocation of $T_5(8, 2)$. Therefore, it is inserted into SQ-flist. Allocated submesh A_4(<0, 6>, <7, 7>) is inserted into Alloc-list. Alloc-list and CFSL generated from (Figure 4) are as follows :

- Alloc-list ={ A_1(<0, 0>, <1, 1>), A_2(<2, 0>, <2, 1>), A_3(<3, 0>, <7, 0>) }, A_4(<0, 2>, <2, 5>) },
- SQ-flist = { S_2(<3, 1>, <7, 5>) },
- HR-flist = VR-flist ={∅}.

The third example is one that even if submesh size is expanded using expansion index, submesh allocation is delayed, because there is no available free submesh that can accommodate a task in CFSL. In (Figure 3), If $T_5(8, 3)$ is given, submesh allocation for $T_5(8, 3)$ is delayed because there is no available free submesh that can accommodate the $T_5(8, 3)$. Therefore, $T_5(8, 3)$ waits until a free submesh is formed.

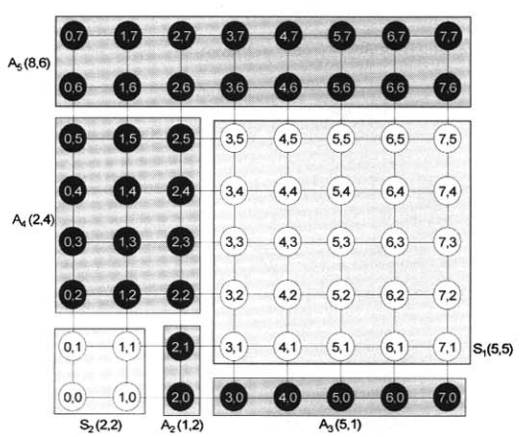
4.3 Submesh Deallocation Algorithm

After finishing the execution of the allocated task, a submesh that is set free is sent back to the related CFSL and is arranged. (Figure 5) describes the submesh deallocation algorithm. In (Figure 5), Submesh deallocation algorithm distinguishes whether an allocated submesh to be freed is IFS or not. If it is not an IFS, Deallocation() module deallocates the allocated submesh. Make_CFSL() module reconstructs CFSL, searching over the entire mesh structure. If it is an IFS, Insert_CFSL() module inserts the submesh into CFSL of which the corresponding type is same as that of IFS.

```
CFSL_Deallocation ( ) // Deallocation(A<w, h>)
{
  if ( Alloc-list = ∅){ // Alloc-list : allocated submesh list
     SQ-flist = {M(W, H)} ;
     HR-flist = VR-flist =∅ ;
  } else {
    if (A(w, h) == IFS ){
     //insert allocated submesh into CFSL according to IFS type
       Insert_CFSL() ;
    } else{
       Deallocation() ; //deallocate allocated submesh
       // reconstruct CFSL by searching for free submesh
       Make_CFSL() ;
    } //end if (A(w, h) == IFS)
  } //end if( Alloc-list = ∅)
} //end
```

(Figure 5) Submesh deallocation algorithm

In (Figure 4), since A_1(<0, 0>, <1, 1>) is an IFS that does not affect the size of other free submeshes, it is inserted into SQ-flist according to its type. The result of A_1(2, 2) deallocation is illustrated in (Figure 6).



(Figure 6) The result of A_1(2, 2) deallocation

Alloc-list and CFSL generated from (Figure 6) are as follows :

- Alloc-list = {A_1(<2, 0>, <2, 1>), A_3(<3, 0>, <7, 0>), A_4(<0, 2>, <2, 5>) }, A_5(<0, 6>, <7, 7>) },
- SQ-flist = { S_1(<3, 1>, <7, 5>, S_2(<0, 0>, <1, 1>) },
- HR-flist = VR-flist = {∅}.

5. Simulation Study

Through simulation, we showed that CFSL allocation strategy performs better compared to the FSL allocation strategy with respect to submesh search time. Kim and Yoon [15] showed that FSL allocation method is better than ADJ [11] and FL [13] allocation methods. In this paper, therefore, the comparison of CFSL, ADJ [11], and FL [13] is omitted.

5.1 Simulation Environment

In simulation, we used a simulator, which was implemented by C# that is an object oriented programming language. Submesh and task having various attributes can be implemented as objects in our simulator. The workload considered for simulation is characterized by the task arrival distribution, the task size distribution, and distribution of the task execution (service) time. The task arrival pattern is assumed to follow the Poisson distribution [16], with an arrival rate λ. Under a given system load ($0 < \rho \leq 1$), the task arrival rate (λ) is determined as follows [17] :

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

where N is the number of processors in the mesh (N = W×H), m is the mean number of processors in a submesh request, and r is the mean execution time. Attributes of task include size and execution time, and FCFS scheduling decides the allocation order. The execution time is assumed to follow exponential distribution with a given mean execution time. The task size (the side lengths of a required submesh) is assumed to follow a given distribution : either uniform, normal, or exponential. Under normal distribution, the mean of task size is assumed to be (H+1)/2 · (W+1)/2 and the variance as the half of the mean, i.e., (H+1)/4 · (W+1)/4 for a mesh(W×H), where W and H means width and height of the entire mesh structure. In exponential distribution, the mean of task size is assumed to be same as that of normal distribution.

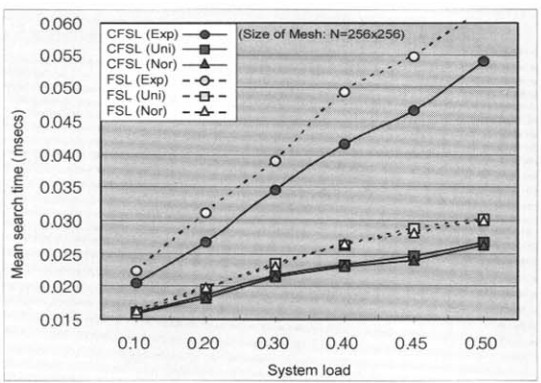
5.2 Performance Analysis

The simulation was conducted under 95 percent confidence level with an error range of ±3 percent and 300,000 tasks for each run of the simulation. We have assumed a

square mesh system for simplicity of plotting. The performance is measured in terms of the mean search time. We define mean search time as follows.

- Mean search time : The average time elapsed on searching for the best-fit submesh for a task.

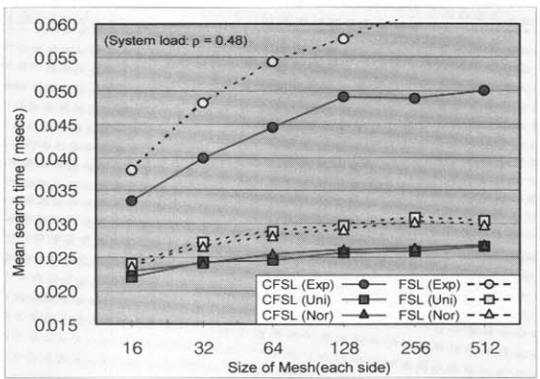
The first simulation was performed to measure the mean search time by varying the offered system load in the range from 0.1 to 1.0 for the 256×256 mesh. (Figure 7) gives the results for exponential (Exp), uniform (Uni), and normal (Nor) distributions, when the system load ranges from 0.1 to 0.55, to obtain better scaling factor because all the strategies are saturated for loads greater than 0.55.



(Figure 7) Mean search time vs. System load (N=256×256).

In (Figure 7), CFSL allocation scheme outperforms in all system loads. When the system load is small, the difference between mean search times of two allocation schemes is not large, but as the system load increases, the difference becomes larger. If the system load is small, the submesh search time is also small because the number of submeshes being divided is small. As a consequence, the difference of mean search time between CFSL and FSL allocation scheme is not large. But as the system load increases, because the number of submeshes being divided after submesh allocation also increases, the submesh search time increases. In this case, CFSL allocation scheme can reduce the submesh search time using the CFSL with the same type as that of given task, compared to FSL. In (Figure 7) for the Exp case, we can see that the mean search time of FSL allocation scheme saturates more rapidly than the CFSL allocation scheme, and also the mean search time difference grows. Therefore, we can conclude that CFSL allocation scheme is better than FSL allocation scheme as the system load increases.

The second simulation was performed to measure the mean search time by varying the size of the mesh from 16×16 to 512×512. (Figure 8) shows the results for system loads 0.48.



(Figure 8) Mean search time vs. size of the mesh (ρ=0.48).

As can be seen from the (Figure 8), CFSL allocation scheme performs well in every mesh size. For Uni and Nor cases, the difference of the mean search time does not become notably larger as the mesh size increases. By the way, for the Exp case, the difference of the mean search time becomes larger as mesh size increases. When the system load is 0.48, CFSL allocation scheme reduced mean search time, compared to FSL allocation scheme, as much as 10.4~19.2%, 8.2~13%, and 2.9~10.3% for Exp, Uni, and Nor cases, respectively. In (Figure 8), for the Exp case, when the mesh size is above 256×256, the mean search time of FSL allocation scheme increases dramatically. Aslo, for the Exp case, the mean search time is larger compared to other probability distributions. For the Uni, and Nor cases, the rate of tasks out of the whole tasks of which the sizes are above half the size of entire mesh is about 15%, but for the Exp case, the rate is about 3%. This implies that there are many smaller tasks in the Exp case, compared to Nor or Uni case. If the size of task is small, it is likely that they can be processed simultaneously. As the number of free submeshes that is divided in allocated submesh increases, the mean search time for the best-fit free submesh increases also. But CFSL allocation scheme reduces the submesh search time by classifying IFS according to the types (square, horizontal rectangle, vertical rectangle) and searching the best-fit submesh from the CFSL.

6. Conclusions

In this paper, a new processor allocation strategy called CFSL allocation scheme is proposed so submesh search time can be reduced for the purpose of improving the performance of mesh-connected multicomputer system. This scheme reduces the submesh search time by classifying independent

free submeshes according to the types (square, horizontal rectangle, vertical rectangle) and searching the best-fit submesh from the CFSL. When it fails to find a submesh, after the size of each submesh is expanded using expansion index, which is stored as an attribute of each submesh. Then, the best-fit submesh is chosen.

Through simulation, we can conclude that CFSL performs better than the previous strategies, regardless of system loads and mesh sizes. When the system load was 0.48, CFSL reduced the mean search time as much as 10.4~19.2%, 8.2~13%, and 2.9~10.3% for exponential, uniform, and normal distribution cases, respectively.

CFSL allocation scheme was proved to be efficient compared to the previous method in reducing the submesh search time. Our future work is to develop a new scheduling method other than FCFS which has been used for this work and to test it.

Reference

[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. Supercomputing, pp.1-6, Nov., 1990.

[4] D. K. Kahaner and U. Wattenberg, "Japan : A Competitive Assessment," IEEE Spectrum, Vol.29, No.9, pp.42-47, Sep., 1992.

[5] G. Zorpette, "The Power of Parallelism," IEEE Spectrum, Vol.29, No.9, pp.28-33, Sep., 1992.

[6] T. E. Bell, "Beyond Today's Supercomputers," IEEE Spectrum, Vol.29, No.9, pp.72-75, Sep., 1992.

[7] K. Li and K.H. Cheng, "A Two-Dimensional Buddy System for Dynamic Resource Allocation in a Partitionable Mesh Connected system," IEEE Journal of Parallel and Distributed Computing, Vol.12, pp.79-83, May, 1991.

[8] P. J. Chuang and N. F. Tzeng, "An Efficient Submesh Allocation Strategy for Mesh Computer Systems," Proc. Int'l Conf. Distributed Computing Systems, pp.256-263, Aug., 1991.

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

[10] Y. Zhu, "Efficient Processor Allocation Strategies for Mesh Connected Parallel Computers," IEEE Journal of Parallel and Distributed Computing, Vol.16, pp.328-337, Dec., 1992.

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

[12] S. Bhattacharya and W. T. Tsai, "Lookahead Processor Allocation in Mesh-Connected Massively Parallel Multicomputer," Proc. Int'l Parallel Processing Symp., pp. 868-875, Apr., 1994.

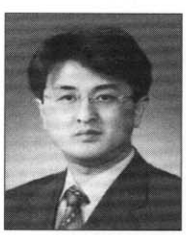
[13] T. Liu, W. K. Huang, F. Lombardi and L. N. Bhuyan, "A Submesh Allocation Scheme for Mesh-Connected Multiprocessor Systems," Proc. Int'l Conf. Parallel Processing, pp.II-159-II-163, Aug., 1995.

[14] S. M. Yoo, H. Y. Youn and B. Shirazi, "An Efficient Task Allocation Scheme for 2D Mesh Architecture," IEEE Trans. on Parallel and Distributed Systems, Vol.8, No.9, pp.934-942, Sep., 1997.

[15] G. M. Kim and H. S. Yoon, "On Submesh Allocation for Mesh Multicomputers : A Best-Fit Allocation and a Virtual Submesh Allocation for Faulty Meshes," IEEE Trans. on Parallel and Distributed Systems, Vol.9, No.2, pp.175-185, Feb., 1998.

[16] S. M. Ross, *Introduction to Probability Models*, sixth edition, Academic Press, 1985.

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



이 원 주

e-mail : wonjoo@doowon.ac.kr

1989년 한양대학교 전자계산학과(학사)

1991년 한양대학교 전자계산학과 (공학석사)

1997년 한양대학교 컴퓨터공학과 (박사과정 수료)

1991년~1995년 (주)큐닉스컴퓨터 응용연구소 선임연구원

1999년~현재 두원공과대학 인터넷프로그래밍과 조교수

관심분야 : 병렬처리시스템, 그리드컴퓨팅, 모바일 컴퓨팅 등

전 창 호

e-mail : chjeon@cse.hanyang.ac.kr

1977년 한양대학교 전자공학과(학사)

1982년 Cornell University 컴퓨터 공학과(공학석사)

1986년 Cornell University 컴퓨터 공학과(공학박사)

1977년~1979년 한국전자통신연구원 연구원

1997년~1998년 한국정보과학회 학회지 편집위원장

1989년~현재 한양대학교 전자컴퓨터공학부 교수

관심분야 : 컴퓨터구조, 병렬처리, 성능분석, 그리드컴퓨팅 등