태스크와 서브메쉬의 유형별 분류에 기반한 프로세서 할당방법

이원주⁰, 전창호
두원공과대학 인터넷프로그래밍과, 한양대학교 전자컴퓨터공학부
wonjoo⁰@doowon.ac.kr, chjeon@para1.hanyang.ac.kr

A Processor Allocation Scheme Based on Classification of Tasks and Submeshes

Won-Ju Lee⁰, Chang-Ho Jeon
Department of Internet Programming, Doowon Technical College,
School of Electrical and Computer Engineering, Hanyang University

요 약

본 논문에서는 메쉬 구조의 다중처리시스템을 위한 새로운 할당방법을 제안한다. 이 할당방법은 다른 가용 서브메쉬와 중첩되지 않는 독립 가용 서브메쉬를 유형에 따라 분류하여 유형별 가용 서브메쉬 리스트를 생성한다. 그리고 태스크의 유형에 따라 해당 유형별 가용 서브메쉬 리스트에서 최적할당이 가능한 서브메쉬를 찾음으로써 서브메쉬를 탐색하는데 소요되는 시간을 줄인다. 이 때 서브메쉬를 찾지 못하면 확장지수를 이용하여 더 큰 가용 서브메쉬를 형성한 후 할당함으로써 태스크의 대기시간을 줄이고, 이 결과로 외적단편화를 줄이는 효과도 얻는다. 또한 할당 해제시 독립 가용 서브메쉬는 다른 가용 서브메쉬의 크기에 변화를 주지 않기 때문에 그 유형에 따라 유형별 가용 서브메쉬 리스트에 삽입한다. 그럼으로써 할당 해제 후 유형별 가용 서브메쉬 리스트를 재생성하기 위해 전체 메쉬 구조를 탐색 할 필요가 없어진다.

1. 서 론

다중처리시스템은 입력되는 작업을 빠르게 처리하기 위해 서로 통신과 협동이 가능한 여러 개의 프로세서들로 구성된 시스템이다[1]. 다중처리시스템의 성능은 프로세서의 활용도에 따라 크게 달라지기 때문에 성능을 향상시키기 위해서는 프로세서의 활용도를 높일 수 있는 태스크 스케줄링 방법과 프로세서 할당방법이 필요하다. 태스크 스케줄링 방법은 입력된 작업들의 처리 순서를 결정하는 것이며, 프로세서 할당방법은 프로세서간의 통신비용을 최소화하기 위해 서로 인접한 가용 프로세서를 찾아 태스크에 할당하는 것이다[2].

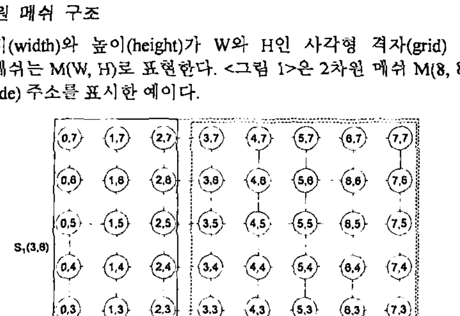
본 논문에서는 단순하고, 규칙적이며, 확장성이 뛰어난 2차원 메쉬 구조의 다중처리시스템을 위한 프로세서 할당 방법을 제안한다. 제안된 할당방법에서는 다른 가용 서브메쉬와 중첩되지 않는 독립 가용 서브메쉬의 유형에 따라 유형별 가용 서브메쉬 리스트를 생성 한다. 그리고 태스크의 유형에 따라 해당 유형별 가용 서브메쉬 리스트에서 최적할당이 가능한 서브메쉬를 찾음으로써 서브메쉬 탐색 시간을 줄인다. 그리고 할당이 가능한 서브메쉬를 찾지 못하면 확장지수를 사용하여 다른 서브메쉬와 중첩되는 영역으로 서브메쉬의 크기를 확장하여 할당함으로써 태스크의 대기시간을 줄이고, 그 결과 외적단편화를 줄이는 효과도 얻는다. 또한 서브 메쉬 할당 해제 시에 독립 가용 서브 메쉬를 그 유형에 따라 분류하여 해당 유형별 가용 서브 메쉬 리스트에 삽입한다. 그럼으로써 할당 해제 후 유형별 가용 서브메쉬 리스트를 재생성하기 위해 전체 메쉬 구조를 탐색할 필요가 없어진다. 본 논문의 2장에서는 기존의 프로세서 할당방법에 대하여 설명하고, 3장에서는 2차원 메쉬 구조에서 사용되는 기본 용어들을 정의한다. 4장 에서는 제안하는 프로세서 할당방법에 대하여 설명하고, 5장에서 결론을 맺는다.

2. 관련 연구

2차원 메쉬구조의 다중처리시스템을 위한 프로세서 할당방법에 대한 연구는 서브메쉬의 인식률을 높이면서 프로세서 단편화를 최소화하고, 할당 시간을 줄이는 방향으로 진행되어 왔으며, 그 결과 여러 프로세서 할당 방법들이 제안되었다[3-13]. 2DB(2 dimensional buddy) 할당방법[3]은 가용 서브메쉬 인식률이 낮으며 내적단편화가 발생한다. FS(frame sliding)[4]와 FF(first-fit)/BF(best-fit)[5] 할당방법은 가용 서브 메쉬 인식률이 낮다. AS(adaptive scan)[6]와 ADJ (adjacency)[7] 할당방법은 가용 서브메쉬 인식률이 높지만 할당 알고리즘 의 시간복잡도가 크다. LKH(look ahead) 할당방법[8]은 큐에 대기하는 태스크의 정보를 이용하여 할당한다. FL(free list) 할당방법[9]은 서로 중첩하는 서브메쉬의 리스트를 생성하여 할당하며 할당 및 해제 알고리즘의 시간복잡도가 작다. QA (quick allocation) 할당방법[10]은 가용 서브메쉬 인식률이 높고, 시간 복잡도가 낮지만 할당방법의 성능이 메쉬 크기에 따라 달라진다. FSL (free submesh list) 할당방법[11]은 가용 서브 메쉬 인식률이 높고, 가용 서브메쉬 리스트를 사용하여 최적할당을 구현한다. 그리고 다음 태스크에 할당 가능한 서브메쉬의 크기를 최대로 유지하기 위해 각 서브메쉬의 예약지수(*reservation factor*)를 구하고 그 값이 최대인 서브메쉬를 할당한다. 이 할당방법은 가용 서브메쉬의 수가 많아지면 예약지수를 구하는 빈도 수가 증가하기 때문에 할당 시간이 늘어난다. 또한, 가용 서브메쉬들 간에 중첩을 허용하고 있어 서브메쉬가 할당되면 이와 중첩되는 다른 가용 서브메쉬들의 크기가 변하기 때문에 할당 및 할당 해제 후 가용 서브메쉬 리스트를 재구성해야 한다.

3. 2차원 메쉬 구조

너비(width)와 높이(height)가 W와 H인 사각형 격자(grid) 구조의 2차원 메쉬는 M(W, H)로 표현한다. <그림 1>은 2차원 메쉬 M(8, 8)의 각 노드(node) 주소를 표시한 예이다.



<그림 1> 2차원 메쉬 M(8,8) 구조

메쉬 M(W,H)는 WxH개의 노드로 구성되며 각 노드는 하나의 프로세서를 의미한다. 각 노드의 주소 (address)는 <x,y> 형태로 표현하며, 노드 <x,y>는 메쉬의 좌측 하단(lower-leftmost) 노드 <0,0>를 기준으로 너비는 x만큼, 높이는 y만큼 떨어진 좌표에 위치한 노드이다. 이 때 x, y값은 $0 \le x \le W-1$와 $0 \le y \le H-1$조건을 만족한다. 메쉬 M(W,H)내에 포함된 서브메쉬 S(w,h)는 너비와 높이가 w 와 h인 사각형 격자 구조로 wxh개의 노드로 구성된다. S(w,h)는 $0 \le w \le W-1$와 $0 \le h \le H-1$ 조건을 만족한다. 서브메쉬의 주소는 두 개의 노드 $<x_1,y_1>$와 $<x_2,y_2>$를 사용하여 $S(<x_1,y_1>, <x_2,y_2>)$로 표현한다. $<x_1,y_1>$ 노드는 서브메쉬의 좌측하단 노드이고, $<x_2,y_2>$노드는 서브메쉬의 우측 상단 노드이다.

정의 1. 가용 서브메쉬(**free submesh**)는 서브메쉬를 구성하는 모든 노드들이 태스크에 할당되지 않은 유휴 프로세서들로 구성된 서브메쉬를 의미하며 S(w,h)로 표현한다.

정의 2. 할당 서브메쉬(***allocated submesh***)는 서브메쉬를 구성하는 모든 노드들이 태스크에 할당되어 실행중인 서브메쉬를 의미하며 A(w,h)로 표현한다.

<그림1>에서 $S_1(<0,2>,<2,7>)$의 주소를 가진 $S_1(3,6)$은 가용 서브메쉬이고, $A_1(<0,0>,<2,1>)$의 주소를 가진 $A_1(3, 2)$는 할당 서브메쉬이다.

정의 3. 독립 가용 서브 메쉬(independent free submesh)는 다른 가용 서브메쉬와 중첩되는 영역을 제외하고 형성 할 수 있는 최대 크기의 가용 서브메쉬를 의미한다.

<그림1>에서 실선으로 구분된 $S_1(<0,2>, <2,7>)$과 $S_2(<3,1>, <7,1>)$는 서로 중첩되는 영역($<3,2>,<7,7>$)을 제외하고 최대 크기를 형성하는 독립 가용 서브메쉬이다. 독립 가용 서브메쉬의 크기가 태스크 보다 작을 경우 서브메쉬를 할당 할 수 없기 때문에 태스크는 큐에 대기하게 된다. 이 때 서브메쉬의 크기를 확장하여 할당이 가능하다면 확장지수를 사용하여 확장한다. 확장지수는 다음과 같이 정의한다.

정의 4. 독립 가용 서브메쉬 $S(<x_1, y_1>, <x_2, y_2>)$의 크기를 확장하기 위해 두 노드 $<x_1,y_1>$, $<x_2,y_2>$에 더해 주는 변위값 ($<\alpha,\beta>,<\alpha',\beta'>$)을 독립 가용 서브메쉬의 확장지수(*expansion index*)라 하며, EI(IS)로 표현한다. EI(IS)는 독립 가용 서브메쉬 $S(<x_1, y_1>,<x_2, y_2>)$와 다른 가용 서브메쉬와 중첩을 허용하면서 구한 최대 크기의 가용 서브메쉬 $S(<x'_1, y'_1>,<x'_2, y'_2>)$를 이용하여 아래와 같이 구한다.

$$EI(IS) = (<\alpha,\beta><\alpha',B'>) = \begin{cases} \alpha = x'_1 - x_1 & (0 \le |\alpha| \le W) \\ \beta = y'_1 - y_1 & (0 \le |\beta| \le H) \\ \alpha' = x'_2 - x_2 & (0 \le |\alpha'| \le W) \\ \beta' = y'_2 - y_2 & (0 \le |\beta'| \le H) \end{cases}$$

독립 가용 서브메쉬인 $S_i(<x_1, y_1>, <x_2, y_2>)$에 확장지수 ($<\alpha,\beta>,<\alpha',\beta'>$)를 합산하여 확장된 서브메쉬 $S_i(<x_1+\alpha, y_1+\beta>, <x_2+\alpha', y_2+\beta'>)$를 구한다. 예를 들어 태스크 T(6,6)에는 <그림 1>의 독립 가용 서브메쉬 $S_1(<0, 2>, <2, 7>)$이나 $S_2(<3, 1>, <7, 1>)$를 할당 할 수 없다. 그러나 $S_1(<0, 2>, <2, 7>)$을 점선으로 표현된 영역($<3, 2>, <7, 7>$)만큼 확장하면 할당이 가능하다. 이 때 $S_1(<0, 2>, <2, 7>)$에 확장지수 $EI(S_1)=(<0, 0><5,0>)$을 합산하여 $S_1(<0, 2>, <7, 7>)$으로 확장한다.

정의 5. 유형별 가용 서브메쉬 리스트(**CFSL**: *classified free submesh list*)는 독립 가용 서브메쉬 S(w,h)들을 w와 h값에 따라 정방형(square)과 가로 직사각형(horizontal rectangle), 세로 직사각형(vertical rectangle) 유형으로 분류하여 생성한다.

- w = h: 정방형
  S-flist={ $S_a(w, h), \ldots, S_p(w, h)$} ($1 \le a \le n,\ 1 \le p \le n,\ a \le p$)
- w > h: 가로 직사각형
  H-flist={ $S_b(w, h), \ldots, S_q(w, h)$} ($1 \le b \le n,\ 1 \le q \le n,\ b \le q$)
- w < h: 세로 직사각형
  V-flist={ $S_c(w, h), \ldots, S_r(w, h)$} ($1 \le c \le n,\ 1 \le r \le n,\ c \le r$)

이 CFSL내의 서브메쉬들은 그 크기에 따라 내림차순으로 정렬된다. S-flist와 H-flist, V-flist는 각각 (p-a+1), (q-b+1), (r-c+1)개의 서브메쉬들로 구성되며 S-flist $\cap$ H-flist $\cap$ V-flist= Φ 조건을 만족한다.

<그림 1>에서 탐색된 $S_1(3,6)$은 h가 w보다 크기 때문에 세로 직사각형으로 분류되어 V-flist에 삽입된다. 그리고 독립 가용 서브메쉬 $S_2(5,1)$는 w가 h보다 크기 때문에 가로 직사각형으로 분류되어 H-flist에 삽입된다. <그림 1>에서 생성된 할당 서브메쉬 리스트와 CFSL은 다음과 같다.

- Alloc_List={$A_1(<0,0>,<2,1>)$, $A_2(<3,0>,<7,0>)$}
- SQ_flist = {Φ }
- HR_flist = { $S_2(<3,1>,<7,1>)$ }
- VR_flist = { $S_1(<0,2>,<2,7>)$ }

4. 태스크 유형별 프로세서 할당방법

CFSL 할당 방법은 독립 가용 서브메쉬를 유형에 따라 분류하여 CFSL을 생성한다. 그리고 태스크의 유형에 따라 해당 CFSL에서 최적할당이 가능한 서브메쉬를 찾아 할당함으로써 서브메쉬 탐색 시간을 줄인다. 그리고 할당이 가능한 서브메쉬를 찾지 못하면 확장지수를 사용하여 다른 서브메쉬와 중첩되는 영역으로 서브메쉬의 크기를 확장하여 할당함으로써 태스크의 대기시간을 줄이고, 그 결과 외적단편화를 줄이는 효과도 얻는다.

4.1 할당 알고리즘

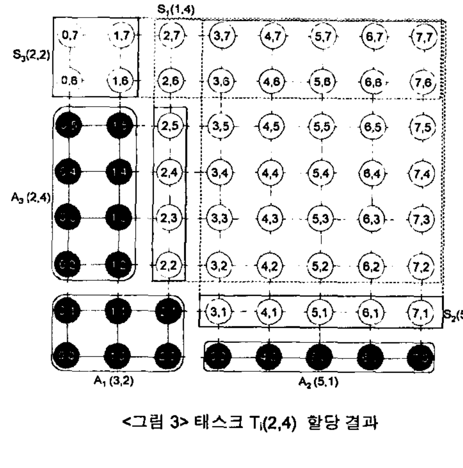
CFSL할당방법의 할당 알고리즘은 <그림 2>와 같다.

```
CFSL_Allocation   //Request(T<w, h>)
{
  Make_CFSL( ); //가용_서브메쉬를 탐색하여 CFSL 생성
  Make_Allocate( ) //유형별 할당 과정 선택
  {
   Ti = Dispatch_Qi(T<w, h>); //대기 큐에서 태스크 dispatch
   If (Ti.type == type-S ){     //정방형(square) 태스크
      for ( i=a; i<p+1; i++)
         SQ-flist에서T<w, w>에 최적할당 가능한 가용 서브메쉬 S 선택
   }else If (Ti.type == type-H ){ //가로 직사각형 태스크
      for ( i=b; i<q+1; i++ )
        HR-flist에서T<w, h>에 최적할당 가능한 가용 서브메쉬 S 선택

   }else If (Ti.type == type-V ){ //세로 직사각형 태스크
      for ( i=c; i<r+1; i++ )
         VR-flist에서T<w, h>에 최적할당 가능한 가용 서브메쉬 S 선택
   }
   Do_Allocate ( ) //서브메쉬 할당 과정
   {
     if(C-flist == φ){
         Waiting();     //할당 가능한 가용 서브메쉬가 생성될 때까지 대기
     }else{
        Allocate();     //후보 가용 서브메쉬를 할당하고 Alloc-list에 삽입;
        Delete();       //가용 서브메쉬 S를 CFSL에서 삭제;
        Insert-CFSL(); // 할당 후 생성된 가용 서브메쉬를 CFSL에 삽입;
     }
   }
  }
}
```

<그림 2> 할당 알고리즘

<그림 1>에서 태스크 $T_3(2,4)$가 입력되면 그 유형에 따라 VR_flist에서 최적할당이 가능한 서브메쉬를 찾는다. VR_flist에서 $S_1(<0,2>,<2,7>)$의 크기가 태스크 보다 크기 때문에 할당이 가능하다. <그림 3>과 같이 $S(<0,2>,<2,7>)$을 $T_3(2,4)$에 할당하면 새로운 가용 서브메쉬들이 생성되며 이 서브메쉬들은 그 유형에 따라 CFSL에 삽입된다.



<그림 3> 태스크 $T_3(2,4)$ 할당 결과

<그림 3>에서 생성된 할당 서브메쉬 리스트와 CFSL은 다음과 같다.

- Alloc_List={$A_1(<0,0>,<2,1>)$, $A_2(<3,0>,<7,0>)$, $A_3(<0,2>,<1,5>)$}
- SQ_flist = { $S_3(<0,6>,<1,7>)$ }
- HR_flist = { $S_2(<3,1>,<7,1>)$ }
- VR_flist = { $S_1(<2,2>,<2,5>)$ }

CFSL할당 알고리즘은 $T_3(2,4)$의 유형에 따라 해당 유형인 VR_flist 만을 탐색하여 최적할당이 가능한 서브메쉬를 찾기 때문에 서브메쉬 탐색시간을 줄일 수 있다.

4.2 할당 해제 알고리즘

CFSL 할당방법의 할당 해제 알고리즘은 <그림 4>와 같다.

```
CFSL_Deallocation ( )    //Deallocation(A<w, h>)
{
  If(Alloc-list = φ){
        S-flist={ M(W,H)};  H-flist = V-flist = φ;
  } else {
    if (A(w,h)  == 독립 가용 서브메쉬 ){
      if (A(w,h) == type-S){ //정방형(square)
          S-flist={ Sa(w, h), ... , Sp(w, h)} U { A(w,h)};
      }else if (A(w,h) == type-H){ //가로 직사각형(horizontal rectangle)
          H-flist ={ Sb(w, h), ... , Sq(w, h)} U { A(w,h)};
      }else if (A(w,h) == type-V){ 세로 직사각형(vertical rectangle)
          V-flist ={ Sc(w, h), ... , Sr(w, h)} U { A(w,h)};
      }
    } else{
      Make_CFSL( ); // 가용_서브메쉬를 탐색하여 CFSL 재생성
    }
  }
}
```

<그림 4> 할당 해제 알고리즘

<그림 4>의 할당 해제 알고리즘에서는 할당 해제할 서브메쉬가 독립 가용 서브메쉬인지 아닌지를 먼저 판별한다. 독립 가용 서브메쉬가 아니면 전체 메쉬 구조를 탐색하여 CFSL을 재생성한다. 그러나 <그림 3>의 할당 서브메쉬 $A_2(<3,0>,<7,0>)$와 같이 독립 가용 서브메쉬가 할당 해제 된다면 다른 가용 서브메쉬의 크기에 변화를 주지 않기 때문에 $A_2(<3,0>, <7,0>)$를 그 유형에 따라 HR_flist에 삽입한다.

4.3 할당 및 할당 해제 알고리즘의 시간복잡도

할당 알고리즘에서 Make_CFSL() 모듈은 2차원 메쉬 M(W, H)를 탐색하여 CFSL을 생성하기 때문에 시간복잡도는 O(F)이다. 여기서 F는 가용 서브 메쉬의 수이다. Make_Allocate() 모듈은 태스크의 유형에 따라 해당 CFSL에서 최적할당이 가능한 하나의 가용 서브메쉬를 찾는 과정으로 시간 복잡도는 O(N)이다. 여기서 N는 해당 유형의 CFSL의 가용 서브 메쉬의 수이다. SQ_flist, HR_flist, VR_flist의 N값은 각각 p-a+1, q-b+1, r-c+1이다. Do_Allocate() 모듈은 최적할당을 한 후 분할되는 가용 서브메쉬의 유형을 분류하여 유형별 가용 서브메쉬 리스트에 삽입하는 모듈로써 시간복잡도는 O(1)이다. CFSL 할당 알고리즘의 시간복잡도는 최악의 경우 O(F)이다. 할당 해제 알고리즘에서는 유형별 가용 서브메쉬 리스트가 φ 이거나 할당 서브메쉬 리스트가 φ 일 경우 시간복잡도는 O(1)이다. 그리고 독립 가용 서브메쉬가 할당 해제 될 경우 CFSL을 재생성하지 않기 때문에 시간복잡도는 O(1)이다. 이외의 경우는 Make_CFSL() 모듈에서 할당된 서브메쉬를 이용하여 CFSL을 재생성하기 때문에 시간복잡도는 $O(B^2)$이다. 여기서 B는 할당된 서브메쉬의 수 이다. 할당 해제 알고리즘의 시간복잡도는 최악의 경우 $O(B^2)$이다.

5. 결론

본 논문에서는 2차원 메쉬구조의 다중처리시스템을 위한 새로운 할당방법을 제안하였다. 기존 프로세서 할당방법들과 제안한 CFSL 할당방법의 특징을 비교하면 <표 1>과 같다. CFSL 할당방법은 독립 가용 서브메쉬를 유형에 따라 분류하여 CFSL을 생성한다. 그리고 태스크의 유형에 따라 해당 CFSL에서 최적할당이 가능한 서브메쉬를 찾아 할당함으로써 서브메쉬 탐색 시간을 줄였다. 할당 과정에서 최적할당이 가능한 서브메쉬를 찾지 못하면 확장지수를 사용하여 다른 서브메쉬와 중첩되는 영역으로 서브메쉬의 크기를 확장하여 할당함으로써 태스크의 대기시간을 줄이고, 그 결과 외적단편화를 줄이는 효과도 얻을 수 있었다. 또한, 할당 해제 시에 독립 가용 서브메쉬는 다른 가용 서브메쉬의 크기에 변화를 주지 않기 때문에 서브메쉬의 유형에 따라 해당 CFSL에 삽입한다. 그럼으로써 CFSL을 재생성하기 위해 전체 메쉬구조를 탐색 할 필요가 없었다.

<표 1> 기존의 프로세서 할당방법과 제안된 CFSL 할당방법 비교

할당방법	태스크 T(w,h) 방향 전환	할당 알고리즘의 시간복잡도	할당 해제 알고리즘의 시간복잡도	서브메쉬 완벽 인식	내적 단편화	서브메쉬 할당
2DB [3]	No	O(logN)	O(logN)	No	Yes	BF
FS [4]	No	O(NB)	O(1)	No	No	FF
FF/BF [5]	No	Θ(N)	O(N)	No	No	FF / BF
AS [6]	Yes	O(N)	O(1)	Yes	No	FF
ADJ [7]	No	$O(B^3)$	O(1)	Yes	No	BF
LKH [8]	No	$O(B^2)$	O(B)	Yes	No	FF, BF, WF
FL [9]	No	$O(F^2)$	$O(F^2)$	Yes	No	BF
QA [10]	Yes	O(hB)	Θ(1)	Yes	No	BF
FSL [11]	Yes	$O(F^2)$	$O(B^3)$	Yes	No	BF
LBA [12]	Yes	$O(B^2)$	O(1)	Yes	No	FF, BF, WF
SBA [13]	Yes	$O(B^2)$	Θ(1)	Yes	No	BFF/EFF
CFSL	Yes	O(F)	$O(B^2)$	Yes	No	BF

• N: 전체 프로세서의 수 • F: 가용 서브메쉬의 수 • B: 할당 서브메쉬의 수 • h: 2D 메쉬의 높이 • FF: First-Fit • BF: Best-fit • WF: Worst-Fit • BFF: Bit-map based FF • EFF: emulated FF

6. 참고문헌

1) Almasi and Gottleb, *Highly Parallel Computing*, The Benjamin / Cummings Publishing Company, 1994. 2nd Edition
2) Catherine M. McCann., *Processor Allocation Policies for Message-Passing Parallel Computers*, Technical Report, no. 95-07-03, Sep. 1994
3) 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
4) P.J. Chuang and N.F. Tzeng, "An Efficient Submesh Allocation Strategy for Mesh Computer Systems", Proc. Intl Conf. Distributed Computing Systems, pp.256-263, Aug. 1991
5) 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
6) J. Ding and L.N. Bhuyan, "An Adaptive Submesh Allocation Strategy for Two-Dimensional Mesh Connected Systems", Proc. Intl Conf. Parallel Processing, pp.II-193-200, Aug. 1993
7) 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
8) S. Bhattacharya, W.-T. Tsai, "Lookahead Processor Allocation in Mesh-Connected Massively Parallel Multocomputer", Proc. Intl Parallel Processing Symp., pp. 868-875, 1994
9) T. Liu, W.-K. Huang, F. Lombardi, and L.N.Bhuyan, "A Submesh Allocation Scheme for Mesh-Connected Multiprocessor Systems", Proc. Intl Conf. Parallel Processing, pp. II-159-II-163, 1995
10) S. M. Yoo, H. Y. Youn, Behrooz 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
11) Geunmo Kim, Hyunsoo 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
12) Ge-Ming Chiu, Shin-Kung Chen, "An Efficient Submesh Allocation Scheme for Two-Dimensional Meshes with Little Overhead", IEEE Trans. on Parallel and Distributed Systems, vol. 10, no 5. pp. 471-486, May 1999
13) Byung S. Yoo, Chita R. Das, "A Fast and Efficient Processor Allocation Scheme for Mesh-Connected Multicomputers", IEEE Trans. on Computers, vol. 51, no 1. pp. 46-60, Jan. 2002