시뮬레이션 기법을 이용한 프로세서 할당 알고리즘들의 성능비교*

Performance Comparisons on Processor Allocation Algorithms by Using Simulation Techniques

최 준 구**, 박 기 현**
Jun-Gu Choi, Kee-Hyun Park

Abstract

With remarkable progress of hardware technologies, multiprocessor systems equipped with thousands of processors will be available in near future. In order to increase the performance of these systems, many processor allocation algorithms have been proposed. However, few studies have been conducted in order to compare the performance of these algorithms.

In this paper, simulation techniques are used in order to compare the performance of the processor allocation algorithms proved to be useful. These are: an algorithm using equipartion, an algorithm using average parallelism, an algorithm using execution signatures, and an algorithm using the number of tasks in a task precedence graph.

Simulation shows that the algorithm using execution signatures performs best while the algorithm using average parallelism performs worst with small allocated processors. Surprisingly, the algorithm using equipartition performs well despite the fact that it has smallest overhead. Overall, it can be recommended that the algorithm using equipartition be used without any execution history and that the algorithm using execution signatures be used with some execution history.

1. 서 론

하드웨어 기술의 발달로 여러 프로세서를 갖춘 다중프로세서 시스템에 대한 연구가 많이 수행되었고, 최근에는 수백 또는 수천개의 프로세서를 갖춘 초병렬(massively parallel) 다중프로세서에 대한 관심이 높아지고 있다

* 본 연구는 1993년도 계명대학교 비사연구기금으로 이루어 졌음.
** 계명대학교 전자계산학과

[10,18]. 이와 같이 많은 프로세서를 갖춘 시스템을 효율적으로 이용하기 위해서는 기존의 다중처리기 시스템을 위한 프로세서 스케쥴링 기법이 확장 및 보완되어야 할 필요가 있다. 즉, 기존의 프로세서 스케쥴링[3]과 타스크 할당(task assignment, 혹은 부하분산(load balancing)) [9,12,16,19] 외에도 프로세서 할당방법도 고려되어야 한다.

프로세서 할당이란 다중프로세서 시스템에서 다중사용자 환경일 경우, 각 응용 프로그램에 할당할 적절한 프로세서 수를 결정하여 성능향상을 이루고자 하는 문제를 일컫는다. 이상적인 다중프로세서 시스템의 성능은 입력되는 응용 프로그램에 할당되는 프로세서 수에 비례하여 증가할 것이다. 그러나 실제 성능은 프로세서 수에 비례하여 증가하지는 않는다. 프로세서간의 통신(inter-processor communication) 장애 및 동기화(synchronization)등의 여러 가지 과부담에 의해서 성능 증가율이 감소하기 때문에 이상적인 성능증가를 기대하기는 힘들다. 그러므로 많은 프로세서를 갖춘 시스템에서도 다중사용자 환경을 도입하는 것이 효율적이다.

동시에 실행되는 각각의 응용 프로그램에 몇 개의 프로세서를 할당할 것인가를 결정하는 것은 시스템의 성능을 결정하는 중요한 문제가 되며 이를 해결하기 위한 프로세서 할당에 대한 연구가 활발히 진행되고 있다[1,5-8,11,13-15,17,21-23]. 그러나 지금까지 제안된 프로세서 할당 알고리즘들은 각자 정의한 시스템에서만 성능이 논의되었을뿐, 이들을 종합적으로 비교 분석한 연구는 극히 적었다. 따라서 같은 시스템 환경하에서 이들을 비교 분석하여 장단점을 살펴보는 것도 의미가 있을것이다.

본 논문의 목적은 지금까지 제안된 중요한 프로세서 할당 알고리즘들을 같은 시스템 환경하에서 종합적으로 시뮬레이션을 통하여 비교 분석하는데 있다. 동등분할(equipartition) 알고리즘[22], 평균병렬도(average parallelism)를 이용한 알고리즘[11], 실행서명(execution signature)을 이용한 알고리즘[5,20] 및 응용 프로그램의 타스크 수를 이용한 알고리즘[1,2] 등을 시뮬레이션을 통하여 비교 분석하였다. 이중 일부 알고리즘들은 성능개선을 위하여 본 논문에서 약간씩 수정 보완하였다. 시뮬레이션에서 사용된 응용 프로그램들은 타스크 선행 그래프(task precedence graph)로 표현되며 이들은 임의로 작성되었다.

본 논문의 구성은 다음과 같다. 2절에서는 본 논문에서 비교 분석 대상인 프로세서 할당 알고리즘에 대하여 설명하고, 3절에서는 시뮬레이션 대상 시스템 모델을 제시하며, 4절에서는 시뮬레이션 결과 및 분석을 설명하겠다. 마지막으로 5절에서는 결론 및 앞으로의 연구과제를 제시할 것이다.

2. 프로세서 할당 알고리즘

2.1 동등분할(Equipartition)을 이용한 프로세서 할당

Tucker 등[22]에 의해 제안된 동등분할 알고리즘에서는 입력되는 응용 프로그램에 전체 프로세서를 같은 비율로 할당한다. 최대병렬도 이상의 프로세서를 할당하는 것은 무의미하므로 본 논문에서는 할당된 프로세서 수를 각 응용 프로그램이 가지고 있는 최대병렬도보다 많지 않도록 수정하였다.

본 논문에서 이용한 수정된 동등분할 알고리즘은 (알고리즘 1)과 같다.

2.2 평균병렬도(average parallelism)를 이용한 프로세서 할당

Eager 등[11]에 의해 제안된 평균병렬도는 무한 개의 프로세서가 이용 가능할 때 해당 응용 프로그램을 수행하는 동안에 사용한 평균 프로세서 수로 나타낸다. 이때 무한 개의 프로세서란 해당 응용 프로그램이 동시에 요구하는 최대 프로세서 수(즉, 최대병렬도(maximum parallelism))라고 이야기 할 수 있다. Eager 등은 응용 프로그램의 speedup과 efficiency의 trade-off가 고려된 적절한 프로세서 수는 평균병렬도에 근접되어 있다는 것을 증명하였다.

본 논문에서는 다중사용자 환경을 가정하였으므로 단순히 각 응용 프로그램에 각각의 평균병렬도만큼 프로세서를 할당하는 것은 문제가 발생한다. 왜냐하면 시스템이 가진 프로세서 수가 평균병렬도의 합과 같지 않을 수 있기 때문이다. 따라서 평균병렬도를 이용한 프로세서 할당 알고리즘에 다음과 같이 약간의 수정을 하였다.

첫째, 각 응용 프로그램에 할당할 기본 프로세서 수를 그 응용 프로그램의 평균병렬도와 같게하고, 할당된 기본

```
Tot_pro = N                                  /* 이용 가능한 프로세서 수 */
Sum = 0                                      /* 할당된 프로세서 수 */
While (Sum < Tot_pro)
  begin
    for every 응용 프로그램i
        if (P_noi < Maxi) AND (Sum < Tot_pro)
          begin
             P_noi = P_noi + 1 /* 응용 프로그램 i에 프로세서 한개를 할당 */
            Sum = Sum + 1
          end
 end
```

〈알고리즘 1〉 동등분할을 이용한 프로세서 할당

프로세서 수가 시스템이 가진 프로세서 수보다 작으면(크면) 사용하는 프로세서 수의 분산(즉 병렬분산(variance in parallelism))에 비례하여 더하도록 (감하도록) 하였다. 이 유는 사용하는 프로세서 수의 분산이 클수록 할당된 프로세서를 효율적으로 사용할 수 없기 때문이다.

둘째, 할당할 최소 프로세서 수를 1로 하였다. 그 이유는 어떤 응용 프로그램에게도 무한 연기(indefinite postponement) 현상이 일어나지 않도록 하기 위해서이다. 그리고 할당할 최대 프로세서 수를 최대병렬도로 하였다.

이상의 수정을 통하여 시뮬레이션에 이용된 알고리즘은 (알고리즘 2)와 같다. 이 알고리즘에서 Avg_i, Max_i, 그리고 Dev_i는 각각 응용 프로그램 i의 평균병렬도, 최대병렬도, 그리고 병렬분산을 의미한다.

2.3 실행서명(execution signature)을 이용한 프로세서 할당

병렬 응용 프로그램에서의 실행서명[5,20]은 해당 응용 프로그램이 홀로 실행 되었을 때의 실행비율이다. p_i개의 프로세서가 응용 프로그램 i에 할당되었을 때의 실행서명 $E_i(p_i)$는 식 (2.1)과 같다. (2.1)식의 X와 Y는 주어진 응용 프로그램에 따라 다르며, 할당된 프로세서 수가 다른 여러 번의 실험을 통하여 얻을 수 있다. Least-square 방법을 이용하여 근사적인 실행서명을 구하기 때문에 많은 실행역사를 가질수록 정확한 실행서명을 얻을 수 있다.

$$E_i(p_i) = p_i / (X \cdot p_i + Y) \quad (2.1)$$

실행서명은 평균병렬도와 마찬가지로 실험을 통하여 얻은 결과를 기초로 하여 얻어진다. 그러므로 소프트웨어와 하드웨어의 병렬특성을 모두 내포하고 있다. 그리고 응용 프로그램의 타스크들 사이의 통신비용도 포함한다.

박기현 등[5,20]이 제안한 실행서명을 이용한 프로세서 할당방법은, 각 응용 프로그램에 프로세서 한 개를 더 할당하였을 경우에 실행서명의 역수가 가장 많이 감소하는 응용 프로그램에게 할당하도록 하였다. 왜냐하면 실행서명의 역수는 실행시간이기 때문에 되도록 실행시간을 줄일 수 있는 응용 프로그램에 우선적으로 할당하기 위함이다. 이와 같은 방법으로 이용가능한 프로세서가 모두 할당될 때까지 반복한다. 실행서명을 이용한 알고리즘은 2가지로 구분되어 있다. 하나는 완벽한 실행역사를 알지 못하고 단지 몇개의 실행역사만을 가지고 있는 경우로서, 이 경우 least-square 방법을 사용하여 근사적 실행서명을 구하고 이를 이용하여 할당한다. 다른 하나는 최대병렬도 만큼의 서로 다른 실행역사를 모두 알고 있을 경우로서, 정확한 실행서명을 이용하여 할당한다. 일단 실행서명이 얻어지면 (알고리즘 3)을 이용하여 프로세서를 할당한다. 본 논문에서는 두가지(정확한, 근사적인) 실행서명을 이용한 방법 모두에 대하여 시뮬레이션하였다.

본 논문에서는 최소 프로세서 수를 앞에서 설명한 알고리즘과 같이 1로 결정하였다. 실행서명을 이용한 알고리즘은 평균병렬도 알고리즘이나 동등분할 알고리즘에서와 같이 최대병렬도 이상 할당되지 않도록 주의할 필요가 없다. 왜냐하면 최대병렬도 이상의 프로세서가 할당 될 경우 실행서명은 증가되지 않기 때문이다.

```
Tot_pro = N                                  /* 이용 가능한 프로세서 수 */
Sum = 0
for every 응용 프로그램 i
begin
   P_noi = Avgi                              /* 최소 프로세서 수        */
   Sum   = Sum + Avgi                        /* 할당된 프로세서 수      */
end
if ( Sum < Tot_pro)
   while (Sum < Tot_pro) do
      Devi가 다음으로 작은 응용 프로그램 i 선택
               (Devi가 가장 작은것에서 부터 가장 큰 것까지 순회)
       if (P_noi < Maxi)
          begin
            P_noi = P_noi + 1                /* 프로세서 한개를 할당 */
            Sum = Sum + 1
          end
 else
    while (Sum > Tot_pro) do
      Devi가 다음으로 큰 응용 프로그램 i 선택
                (Devi가 가장 큰것에서부터 가장 작은것까지 순회)
      if (P_noi > 2)
         begin
           P_noi = P_noi - 1                 /* 프로세서 한개를 회수 */
           Sum = Sum - 1
         end
```

〈알고리즘 2〉 평균병렬도를 이용한 프로세서 할당

```
Tot_pro = N                                  /* 이용 가능한 프로세서 수 */
Sum = 0                                      /* 할당된 프로세서 수 */
for every 응용 프로그램 i
  begin
    P_noi = 1                                /* 최소 프로세서 할당 */
    Sum = sum + P_noi
  end
while (Sum < Tot_pro) do
  begin
    (1/(Ei(P_noi+1)) - 1/(Ei(P_noi)) 값이
                        가장 큰 응용 프로그램 i선택
    P_noi = P_noi + 1                        /* 프로세서 한개를 할당 */
    Sum = Sum + 1
  end
```

〈알고리즘 3〉 실행서명을 이용한 프로세서 할당

2.4 응용프로그램의 타스크 수를 이용한 프로세서 할당

최준구 등[1,2]은 응용 프로그램의 타스크 선행그래프에서 얻은 타스크의 수를 이용한 프로세서 할당방법을 제안하였다. 이 알고리즘의 기본개념은 각 응용 프로그램의 타스크가 많으면 많을수록 프로세서 이용률이 높을 가능성이 있다는 가정을 기초로 하여, 전체 응용 프로그램의 타스크 수에 비례하여 각 응용 프로그램에 할당할 초기 프로세서 수를 결정한다. 일단 초기 프로세서 수가 결정되면 응용 프로그램의 병렬특성인 최대병렬도와 평균병렬도의 차(Gap)를 이용하여 프로세서 수를 미세 조정한다.

응용 프로그램 i에 할당할 기본적인 프로세서수 P_no_i는 식(2.2)과 같다. 식 (2.2)에서 No_task_i는 응용 프로그램 i에서 생성되는 타스크의 수이며, Sum_task는 입력된 응용 프로그램들의 task 합이고, Tot_pro는 시스템이 가진 전체 프로세서 수이다.

$$P\_no_i = (No\_task_i / Sum\_task) * Tot\_pro \qquad (2.2)$$

각 응용 프로그램에 할당된 프로세서 합이 이용가능한 프로세서 수보다 클 경우에는 Gap이 제일 큰 응용 프로그램에서 하나의 프로세서를 회수한다. 이 과정을 할당된 프로세서 수가 이용가능한 프로세서 수와 같아질 때까지 반복한다. 또, 응용 프로그램의 Gap이 같을 때는 각 응용 프로그램의 타스크 수에 따라 할당할 프로세서 수가 결정된다. 만약 최대(최소) Gap을 가진 응용 프로그램이 여러 개 존재할 때는 타스크 수가 가장 큰(작은) 응용 프로그램을 선택한다. 시뮬레이션에 이용된 알고리즘은 다음과 같다.

3. 시스템 모델

본 논문에서 가정한 시스템 환경은 그림 3-1과 같다. 그림 3-1의 시스템은 하나의 호스트(host) 프로세서와 n개의 독립적인 프로세서로 구성된 프로세서 풀(pool)을 가지며, 각 프로세서는 동일하다고 가정한다. 호스트는 응용 프로그램을 컴파일/로드하며 스케쥴링을 담당한다. 응용 프로그램의 실행은 프로세서 풀에서 이루어진다.

시스템내에서의 작업처리 방법은 일괄처리를 가정하였다. 즉 여러 개의 응용 프로그램이 대기큐에 도착하면 호스트는 일정한 크기의 윈도우(window)를 설정한다. 윈도우의 크기는 시스템의 성능을 최대화 할 수 있도록 조절할 수 있다. 큐에서 윈도우 크기만큼의 응용 프로그램을 로드하면서 앞절에서 설명한 여러가지 알고리즘을 이용하여 각각의 응용 프로그램에게 적절한 프로세서 수를 계산하여 할당하여 준다. 본 논문에서는 윈도우의 크기를 3, 4, 5로 하였다. 각각의 응용 프로그램은 할당된 프로세서를 이용하여 실행을 한 후 프로세서를 반납하고 시스템을 빠져 나가게 된다. 프로세서를 이용하여 실행할 때의 통신 과부담은 모든 할당 알고리즘에서 동일하다고 가정한다. 로드된 모든 프로그램이 실행을 마치면 호스트는 대기큐에서 다음에 실행할 응용 프로그램들을 윈도우 크기만큼 다시 로드하여 프로세서를 할당하게 된다.

할당 알고리즘들의 성능은 평균 반환시간(average turnaround time)으로 비교하였다. 반환시간이란 응용 프로그램이 시스템에 도착해서 프로세서 풀을 빠져 나갈때 까지의 시간간격이다. 여기서 할당 방법 자체의 과부담은 고려되지 않았으나 할당 과부담을 고려하는 것은 어렵지 않다.

4. 시뮬레이션 및 비교분석

이 절에서는 본 논문에서 수행한 시뮬레이션 방법과 그 결과에 대하여 설명하고, 각각의 알고리즘의 성능을 비교분석하겠다. 본 논문의 시뮬레이션에 이용되는 응용 프로그램은 그림 4-1과 같이 타스크 선행 그래프를 이용하여 표현하였다. 각각의 응용 프로그램은 임의로 얻어진다. 응용 프로그램의 생성방법은 일단, 응용 프로그램의 깊이(depth)를 구한 후 각 레벨(level)에서 파생된 노드의 수를 난수발생 함수를 이용하여 구하였다. 깊이와 노드가 구해졌으면 노드를 순회하며 전체적인 선행관계를 정의하여 준다.

선행관계의 정의는 부모노드는 최소 1개에서 3개까지의 자식 노드를 가질 수 있도록 하였다. 부모노드가 바로 다음 레벨의 자식을 가질 확률은 70%로 하였으며, 30%는 다른 레벨의 자식을 갖도록 하였다. 이렇게 선행관계가 형성된 후, 마지막 노드로부터 역으로 순회하며 부모노드가 없는 노드에게 난수 발생함수를 이용하여 바로 위의 레벨의 노드중 하나를 부모노드로 지정하게 하였다.

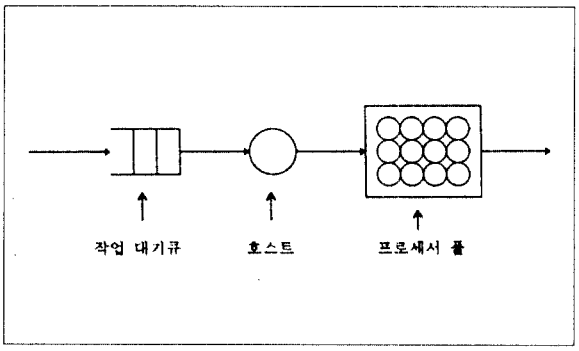
이렇게 생성된 응용 프로그램의 타스크 그래프를 이용하여 여러가지 병렬특성을 계산하였다. 병렬특성을 계산

```
Tot_pro = N                                    /* 이용가능한 프로세서 수 */
Sum = 0
for every 응용 프로그램 i
      begin
        Gap_i = Max_i - Avg_i                                  /* Gapi 계산 */
        Sum_task = Sum_task + No_task_i                /* 전체 타스크의 수 */
      end
   for every 응용 프로그램 i
      P_no_i = (No_task_i / Sum_task) * Tot_pro   /* 초기 프로세서 수 */
   while (Sum < Tot_pro) do
     begin
       각 응용 프로그램 중 Gap이 제일 작은 응용 프로그램 i를 선택
        if (Gap 이 같다면) then
            타스크의 수가 많은 응용 프로그램 i을 선택
        if (P_no_i < Max_i)
          begin
            P_no_i = P_no_i + 1                          /* 프로세서 할당 */
            Sum = Sum + 1
          end
        Gap_i = Gap_i + 1
     end
   while (Sum > Tot_pro) do
      begin
        각 응용 프로그램 중 Gap이 제일 큰 응용 프로그램 i을 선택
        if (Gap이 같다면) then
           타스크의 수가 적은 응용 프로그램 i을 선택
        if (P_no_i < Max_i)
           begin
             P_no_i = P_no_i - 1                         /* 프로세서 회수 */
             Sum = Sum - 1
           end
        Gap_i = Gap_i - 1
      end
```

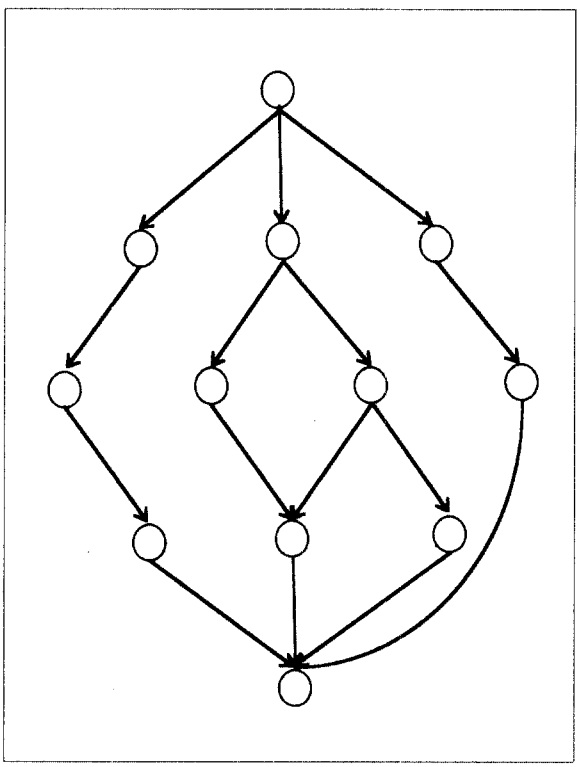
〈알고리즘 4〉 응용프로그램의 타스크 수를 이용한 프로세서 할당

하기 위하여 타스크 그래프의 노드 실행시간은 1 단위시간으로 가정하였다. 각 응용 프로그램에 할당된 프로세서 수에 따른 실행시간은 Hu의 레벨링 알고리즘[4]을 이용하여 계산하였다. Hu의 알고리즘은 자식노드를 하나만 가지는 경우 최적의 실행시간을 계산할 수 있으며, 본 논문에서 이용하는 타스크 선행그래프와 같은 경우에는 최적에 근사한 실행시간을 계산할 수 있다. 본 논문에서는 위와 같은 방법으로 10개의 타스크 선행 그래프를 생성하고 이들을 조합하여 시뮬레이션 하였다.

윈도우 크기는 3, 4, 5로 변화시켜, 각각에 대하여 시스템의 전체 프로세서 수를 윈도우 크기의 배수로 하여 시뮬레이션 하였다. 윈도우의 크기가 3인 경우 총 120회의 시뮬레이션을, 이용가능한 전체 프로세서 수를 3의 배수로 하여 시뮬레션 하였다. 또 윈도우의 크기가 4인 경우 총 210회의 시뮬레이션을, 이용 가능한 전체 프로세서 수를 4의 배수로 하여 시뮬레이션 하였으며, 윈도우의 크기



〈그림 3-1〉 시스템 모델



〈그림 4-1〉 타스크 선행 그래프

가 5인 경우 252회의 실험을, 이용가능한 전체 프로세서 수를 5의 배수로 하여 시뮬레이션 하였다. 이용 가능한 프로세서 수를 윈도우 크기의 배수로 한 이유는 동등분할 알고리즘을 시뮬레이션함에 있어 각 응용 프로그램에게 똑같은 비율의 프로세서를 할당하기 위해서이다. 그리고 무한대기를 방지하기 위하여 각 알고리즘에서는 최소한 한개의 프로세서는 할당 받도록 하였다.

윈도우의 크기를 3으로하여 시뮬레이션한 결과는 표 4-1과 같다. 표 4-2와 4-3은 각각 윈도우의 크기를 4와 5로 하여 시뮬레이션한 결과이다. 이 결과에서 최적 할당에서의 반환시간은, 각 응용 프로그램에 할당할 수 있는 모든 경우를 계산하여 반환시간이 제일 짧은 것을 구하였다. 이 결과를 토대로 최적 할당과 각 알고리즘을 이용한 결과와의 상대오차를 그림 4-2, 그림 4-3, 그림 4-4 와 같은 그래프로 그릴 수 있다. 표와 그림에서 동등분할 알고리즘은 "Equ"로, 평균병렬도를 이용한 알고리즘은 "Avg"로, 정확한(근사적) 실행서명을 이용한 알고리즘은 "E.S." ("App—E.S.")로 나타내었다. 타스크의 수를 이용한 알고리즘은 "Task"로 표현하였다. 그림에서 Y축의 상대오차는 최적할당과 각 알고리즘을 이용한 할당과의 반환시간에 관한 상대오차이다.

시뮬레이션 결과, 이용가능한 프로세서 수가 윈도우 크기의 4배 이하인 경우 정확한 실행서명을 이용한 할당은 윈도우의 크기에 관계없이 최적할당과 일치하였다. 그러나 평균병렬도를 이용한 알고리즘은 최고 19%까지 성능이 떨어짐을 알 수 있다. 평균병렬도는 단순하기 때문에 응용프로그램의 수가 많을 경우에는 큰 효과를 보지만, 응용 프로그램의 수가 적을 경우에는 큰 효과를 볼 수 없다. 만약 응용 프로그램의 수가 적을 경우 병렬도의 분산이 크다면 평균병렬도 자체만으로는 병렬특성을 잘 나타내지 못하기 때문이다. 전체 프로세서 수가 윈도우 크기의 4배 이상 8배 이하에서는 동등분할 알고리즘이 2-3% 정도 성능이 떨어짐을 알 수 있었으며, 실행서명을 이용한 알고리즘은 다른 알고리즘보다 성능이 좋음을 알 수 있다. 프로세서 수가 윈도우 크기의 8배 이상이 되면 대부분의 알고리즘이 비슷한 성능을 나타내고 있다. 대체로 근사적 실행서명을 이용한 알고리즘은 정확한 실행서명을 이용한 알고리즘과 많은 차이를 나타내지 않고 있다. 전체적으로 평균병렬도를 제외한 대부분의 알고리즘이 5% 이내의 상대오차를 보여줌으로써 이용가능한 프로세서 수와 관계없이 일정한 성능을 나타냄을 알 수 있다. 또한 윈도우의 크기와는 무관하게 대부분 일정한 성능을 유지함을 확인할 수 있다.

여기서 놀라운 점은 동등분할 할당이 비록 단순하지만 전체적으로 좋은 성능을 보이고 있는 것이다. 이것은 실

〈표 4-1〉 시뮬레이션 결과 (윈도우 크기 = 3)
(단위: 1 단위시간)

프로세서 수	최적 할당	Equ	Avg	Es	App—Es	Task
3	4560	4560	4560	4560	4560	4560
6	2423	2460	2740	2423	2477	2443
9	1718	1800	1888	1718	1723	1434
12	1404	1428	1470	1404	1411	1250
15	1220	1251	1254	1228	1246	1252
18	1099	1148	1132	1122	1121	1126
21	1029	1069	1053	1043	1052	1050
24	982	998	1003	996	994	999
27	943	961	961	959	956	958
30	918	930	934	934	931	927
33	902	912	910	914	909	909
36	893	899	896	902	897	899
39	884	888	886	890	888	887
41	880	882	882	882	881	881
44	878	879	880	879	879	878
47	877	877	878	877	877	877
51	876	876	876	876	876	876

〈표 4-2〉 시뮬레이션 결과 (윈도우 크기 = 4)
(단위: 1 단위시간)

프로세서 수	최적 할당	Equ	Avg	Es	App—Es	Task
4	7980	7980	7980	7980	7980	7980
8	4221	4305	5029	4221	4266	4238
12	2992	3150	3375	2992	3001	3044
16	2446	2499	2581	2446	2461	2495
20	2121	2188	2187	2144	2173	2171
24	1915	2008	1973	1953	1968	1979
28	1793	1865	1847	1819	1827	1850
32	1702	1740	1747	1737	1736	1740
36	1632	1672	1673	1669	1664	1664
40	1592	1617	1616	1623	1613	1612
44	1568	1583	1578	1591	1578	1585
48	1550	1558	1555	1564	1559	1558
52	1540	1546	1546	1545	1544	1541
56	1537	1539	1540	1538	1539	1537
60	1534	1534	1534	1534	1534	1534

행시간이 짧은 응용 프로그램에게 상대적으로 많은 프로세서를 할당시켜주므로써, 보다 빨리 실행을 마치도록 하기 때문이다.

이상의 시뮬레이션 결과를 토대로 바람직한 할당방법을 생각한다면 다음과 같다: 첫째, 실행역사가 전혀 없는 새로운 응용 프로그램들에 대해서는 동등분할 알고리즘을 적용하여 스케쥴링 오버헤드를 줄이도록 하고, 둘째, 실행역사가 있는 응용 프로그램들에 대해서는 근사적(혹은 정확한) 실행서명을 이용한 알고리즘을 사용하는 것이 바람직하겠다.

5. 결 론

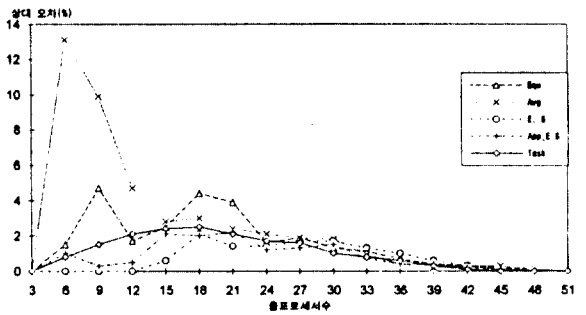
프로세서 수가 많은 다중처리 시스템의 중요성이 부각됨에 따라 프로세서 할당방법에 대한 관심도 높아지고 있다. 본 논문에서는 지금까지 제안된 중요한 프로세서 할당 알고리즘들을 시뮬레이션을 통하여 비교 분석하였다. 시뮬레이션 결과, 전체적으로 평균병렬도를 이용한 프로세서 할당 알고리즘의 성능이 다른 알고리즘들에 비해 성능이 떨어짐을 알 수 있었다. 또, 실행서명을 이용한 알고리즘은 정확한 실행서명을 이용하든, 근사적인 실행 서명을 이용하든, 전체적으로 성능이 매우 좋음을 알 수 있었다. 동등분할의 경우 각 응용 프로그램의 병렬특성과 무관하게 할당함으로서 별도의 계산을 필요치 않아도 되며, 전체적으로 좋은 성능을 보여 주고 있다. 따라서 실행역사가 없는 응용프로그램을 위해서는 동등분할 알고리즘을 사용하고, 실행역사가 생기면 실행서명을 이용한 알고리즘을 사용하는 것이 바람직할 것이다.

본 연구에서의 시뮬레이션 환경은 응용 프로그램의 각 노드가 같은 수행시간을 갖는다는 제약을 같고 있다. 또 실제 컴퓨팅에 존재하는 여러가지 과부담들을 고려하지 않았다. 이와 같은 문제점을 고려하여 각 노드의 실행시간이 다른 경우와, 과부담(통신 과부담등)을 고려한 경우에 대해서도 연구하여야 할 것이다. 또한 시뮬레이션 외에 실제의 다중처리 시스템에서의 성능평가도 앞으로의

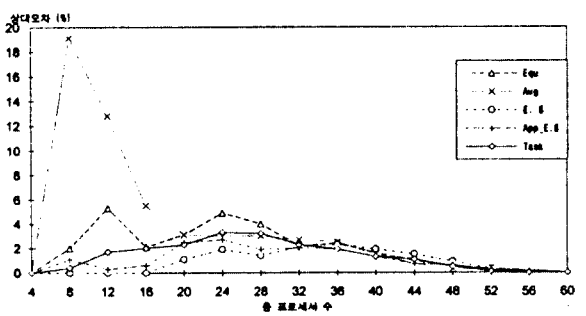
〈표 4-3〉 시뮬레이션 결과 (윈도우 크기 = 5)

(단위: 1 단위시간)

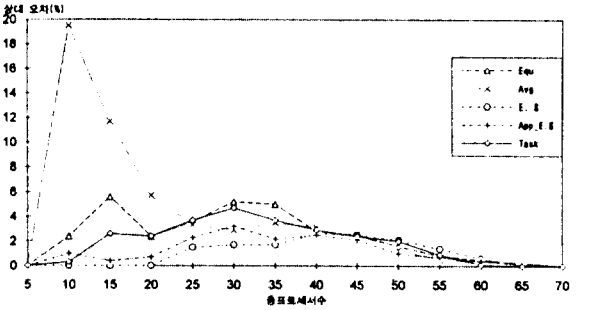
프로세서 수	최적 할당	Equ	Avg	Es	App—Es	Task
5	9576	9576	9576	9576	9576	9576
10	5045	5165	6030	5045	5098	5058
15	3578	3779	3998	3578	3593	3672
20	2927	2998	3093	2927	2949	2998
25	2534	2625	2618	2571	2592	2628
30	2293	2411	2360	2331	2367	2400
35	2144	2251	2218	2181	2190	2224
40	2028	2095	2090	2081	2079	2087
45	1947	1996	1998	1995	1988	1993
50	1900	1930	1924	1939	1920	1937
55	1871	1887	1881	1897	1885	1887
60	1854	1862	1861	1864	1860	1856
65	1845	1848	1850	1846	1860	1846
70	1841	1841	1841	1841	1841	1841



〈그림 4-2〉 프로세서 할당 알고리즘들의 성능비교(윈도우크기=3)



〈그림 4-3〉 프로세서 할당 알고리즘들의 성능비교(윈도우크기=4)



〈그림 4-4〉 프로세서 할당 알고리즘들의 성능비교(윈도우크기=5)

연구과제이다.

참고문헌

[1] 최준구, *Pessimistic* 실행환경에서의 근사적 프로세서 할당, 석사학위논문, 계명대학교, 1991.

[2] 최준구, 박기현, "Pessimistic 실행환경에서의 병렬특성을 이용한 근사적 프로세서 할당," 한국정보과학회 학술발표 논문집, 18권, 1호(1992), pp.569-571.

[3] Deitel, H. M., *Operating Systems second edition*, Canada, 1990.

[4] Baker, K. R., *Introduction to Sequencing and Scheduling*, New York, 1975., pp. 114-118.

[5] Park, K. H., *Dynamic Processor Partitioning for Multiprogrammed Multiprocessor Systems*, Ph. D. Dissertation, Vanderbilt Univ., Aug. 1990.

[6] Al-Dhelaan A. and B. Bose, "A New Strategy for processors allocation in an N-Cube Multiprocessors," *Proc. Phoenix Conf. Comp. and Comm.*, Mar. 1989

[7] Chen M. and K. H. Schultz, "Topological Properties of hypercubes," *IEEE Trans. on Comp.*, July 1988, Vol. 37, No. 7.

[8] Chuang P. J. and N. F. Tzeng, "Dynamic processor allocation in Hypercube Computers," *Proc. of 17th Annual International Symposium on Computer Architecture*, 1990, pp. 40-49.

[9] Black, D. L., "Processor, Priority, and Policy : Mach Scheduling for New Environment," *Proc. of the Winter 1991 USENIX Conf.*, 1991, pp. 1-12.

[10] Duncan, R., "A Survey of Parallel Computer Architecture," *IEEE Computer*, Feb. 1990, pp. 5-16.

[11] Eager, D. L. and J. Zahorhjan and E. D. Lazowska, "Speedup Versus Efficiency in Parallel Systems," *IEEE Trans. on S/W Engineering*, Vol. SE-12, Vo. 5, May 1986, pp. 408-423.

[12] Gajski, D. D. and J. K. Peir, "Essential Issues in Multiprocessor System," *IEEE Computer*, June 1985, pp. 9-27.

[13] Ghosal, D., "The Processor Working Set and Its Use in Scheduling Multiprocessor Systems," *IEEE Trans. on S/W Engineering*, Vol. 17, No. 5, May 1991, pp. 443-353.

[14] Kim G. and H. Yoon, "A processor Allocation Strategy using Cube Coalescing in Hypercube multicomputers," *Proc. of Fifth Symposium on Parallel and Distributed Processing*, Dec. 1993, pp. 596-605.

[15] Lentenegger, S. T. and M. K. Vernon, "Performance of Multiprogrammed Multiprocessor Scheduling Algorithms," *ACM SIGMETRICS*, Vol. 18, No. 1, May 1990, pp. 226-236.

[16] Lin, F. C. H. and R. M. Keller, "The Gradient Model Load Balancing Method," *IEEE Trans. on S/W Engineering*, Vol. SE-13, No. 1, January 1987, pp. 32-38

[17] Mccann, C. and R. Vaswani and J. Zahorjan, "A dynamic Processor Allocation Policy for Multiprogrammed Shared-Memory Multiprocessors," *ACM Trans. on Computer Systems*, [Vol. 11, No. 2, May 1993, pp. 146-178.

[18] Murakami, K. and T. Kakuta and R. Onai and N. Ito, "Research on Parallel Machine Architecture for Fifth-Generation Computer Systems," *IEEE Computer*, June 1985, [pp. 76-92.

[19] Ousterhout, J. K., "Scheduling techniques for concurrent systems," *In Third International Conference on Distributed Computing Systems*, 1982, pp. 22-30.

[20] Park, K. H. and L. W. Dowdy, "Dynamic Partitioning of Multiprocessor Systems," *International Journal of Parallel Programming*, Vol. 18, No. 2, 1989, pp. 91-120.

[21] Sevcik, K. C., "Characterization of Parallelism in Applications and Their Use On Scheduling," *Perform. Eval. Review*, Vol. 17, No. 1, May 1989, pp. 171-180.

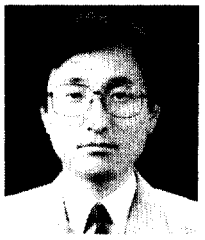
[22] Tucker, A. and A. Gupta, "Process Control and Scheduling Issues for Multiprogrammed Shared-Memory Multiprocessor," *Proc. 12th ACM Symposium on Operating System Principles*, Dec. 1989, pp. 159-166.

[23] Zahorjan, J. C. and McCann, "Processor scheduling shared memory multiprocessors," *ACM SIGMETRICS*, Vol. 18, No. 1, May 1990, pp. 214-225.

● 저자소개 ●

최준구

1990.2 계명대학교 공과대학 전자계산학과 졸업(공학사)
1992.2 계명대학교 대학원 전자계산학과 졸업(공학석사)
1994.3 계명대학교 대학원 전자계산학과 박사과정 재학중
현재 경산대학교 정보처리학과 강의조교
관심분야: 컴퓨터 시뮬레이션, 병렬 분산 운영체제



박기현

1979.2 경북대학교 전자공학과 졸업(공학사)
1981.2 한국과학기술원 전자계산학과 졸업(이학석사)
1990.8 미국 Vanderbilt 대학교 전자계산학과(공학박사)
현재 계명대학교 전자계산학과 부교수
관심분야: 병렬 분산 운영체제, 성능분석/예측