韓國軍事運營分析學會誌
第17卷, 第2號, 1991. 12. 31.

다중프로세서 시스템에 적합한 우선순위 할당 결정기법에 관한 연구

(Allocation Priority Scheme for Multiprocessor Systems)

박영선, 김화수*

Abstract

This paper presents the Allocation Priority Scheme (APS) for multiprocessor system. The objective of APS is to reduce the time-complexity on a Physical Mapping Scheme (PMS). The PMS is to allocate the nodes of the Data Dependency Graph (DDG) to the multprocessors efficiently and effectively.

The APS provides the priority to each node (vertex) in the DDG. In other words, the goal of the APS is to find a request resource mapping such that the total cost (time-complexity) is minimized. The special case in which all requests have equal priorities and all resoruces have equal precedences, and the comparisons between our APS and other schems are discussed in the paper.

The APS provides the heuristic rules which are based on maximum height (MH), number of children nodes (N_c), number of father nodes (N_f), and computation time (T_c). The estimation method of the computaion time is in the paper.

*國防大學院

I. 서론

우선순위 할당 기법은 어떤 응용 프로그램을 다수의 프로세서에 할당하기 전에 응용 프로그램의 각 타스크들에 대한 우선순위를 부여하여 효율적인 물리적 매핑(Physical Mapping)을 달성하기 위한 것으로, 한 순간에 다수의 타스크들이 다수의 프로세서에 동시에 스케줄링을 요청할시에 사용되는 기법이다.

일반적으로 많은 응용프로그램들은 동일한 Level에서 프로세서 할당 요청을 하는 타스크의 수가 프로세서의 수 보다 많기 때문에 우선순위 할당 기법 필요성이 대두되었다. 우선순위 할당기법은 전체적인 처리시간을 단축시키는데 커다란 영향을 미치는 중요한 기법중에 하나이다.

오늘날 하드웨어 가격의 급격한 하락에 따른 대량의 프로세서에 의한 다중처리 추세를 고려할 때, 기존 발표된 우선순위 할당기법 대부분이 한개 혹은 두개의 요소를 고려한 heuristic 알고리즘을 적용한 것으로 현재의 다중 프로세서 시스템에는 적합하지가 않다[8]. 기존의 관련 연구 논문에 대한 검토는 제2장을 참조 바란다.

제안하고자 하는 우선순위 할당 기법은 기존의 한개 혹은 두 개의 요소를 고려한 우선순위 할당 기법을 발전시켜 MH(Maximum Height), N_c(Child 블럭노드 수의 합), N_f(Father 블럭노드 수의 합) 및 T_c(계산시간)를 고려한 heuristic 알고리즘으로서 소프트웨어적으로 구현하기 쉽도록 각각의 블럭노드에 대한 종합적인 블럭노드정보(MH, N_c, N_f, T_c)를 구한 후 제안된 heuristic 규칙을 적용하여, 최종적으로 각각의 블럭노드에 대한 우선순위를 할당한다. 여기서의 블럭노드란 타스크 노드의 집합을 의미한다.

우선순위 할당 기법 적용시에 중요고려사항 중 하나는 계산시간(T_c)이며, 계산시간에 대한 정확한 예측은 어려운 실정이다[6]. 본 논문에서 계산시간 예측은 통계적으로 연구되어 나온 결과를 그대로 인용하였으며, 각 타스크 노드에 대한 계산시간을 알고 있을 때, 블럭 전체의 계산시간 예측 방법은 제2장에 제시하였다.

본 논문에서 제안하는 우선순위 할당 기법은 일반적인 Architecture Model에 적용할 수 있으며, Data Flow Model에 적용할 경우 보다 효과적이다.

본 논문의 구성은 제2장에서 기존의 관련된 연구와 제안하는 Heuristic Rule을 설명하였으며, 제3장에서는 우선순위 할당 기법에 대한 전체적인 알고리즘을 기술하고, 종속관계를 갖는 그래프를 예제로 선정한 다음 제안한 알고리즘을 적용하여 각 블럭노드들에 대한 최종적인 우선순위 할당에 대한 결과를 제시하였으며, 마지막 제4장 결론으로 끝맺었다.

II. 우선순위 할당 기법

2.1 관련 연구 검토

Coffman[3]은 단지 긴 Path만을 고려하여 우선순위 기법을 제안하였다. 그러나, 만약에 종속그래프가 한개 이상의 동일한 Path를 가지고 있을시 효율적인 방법이 되지 못한다.

Kasahara[4]는 두개의 요소, Level 및 Child 블럭노드의 수를 고려한 우선순위 기법을 제안하였다.

Sarkar[9]는 좋은 매핑기법을 제안하였으나 매핑시 우선순위 할당 기법을 무시하였다. 물론, 우선순위 할당 기법은 물리적 매핑기법에서 반드시 필요한 것은 아니지만, 우선순위 할당 기법을 사용하면 소프트웨어로 실행시 근본적으로 어려운 문제점을 해결하여 준다. 다시 말해서, 복잡한 종속관계를 갖고 있는 그래프상에서 우선순위 할당 기법을 사용하면, time-complexity를 줄일 수 있다는 것이 입증되었다.

Potkonjak[7]은 Level 및 Minimal Life-time을 고려하였으나, 상세한 절차는 제안하지 않았다.

상기와 같이 제안된 우선순위 할당 기법은 블럭노드정보의 일부만을 사용한 기법으로 특히 복잡한 종속관계를 갖는 과학적인 계산 (Scientific Computation) 방정식 및 프로세서가 증가하는 현재의 추세에는 적합하지 않는 기법이다.

2.2. 정 의

본 우선순위 할당 기법에 관련된 정의를 살펴보면 다음과 같다. 정의에서 사용되는 블럭노드는 타스크 노드의 집합으로 Minimum Cut-Set 및 계산시간을 포함한 Load-Balancing을 고려한 분할을 통해 형성된다[5].

▶정의 1-Root 블럭노드 : 현재 블럭노드가 Predecessor 블럭노드를 가지고 있지 않는 블럭노드를 Root 블럭노드라 한다.

▶정의 2-Leaf 블럭노드 : 현재 블럭노드가 Successor 블럭노드를 가지고 있지 않는 블럭노드를 Leaf 블럭노드라 한다.

▶정의 3-MH (Maximum Height) : 현재 블럭노드와 Leaf 블럭노드까지의 최대 거리를 의미하며, Level과 동일한 개념으로 사용한다. 본 논문에서는 Bottom-up 방법을 채택하였으므로 Leaf 블럭노드는 MH=1이 된다.

▶정의 4-N_c (Child 블럭노드의 수) : 현재 블럭노드에서 Leaf 블럭노드까지의 Child 블럭노드 전체의 수이다. Child 블럭노드 전체의 수는 direct child 블럭노드의 수 및 indirect child 블럭노드의 수를 합한 것이다. 여기에서 direct child는 현재 블럭노드에 직접적으로 연결된 Child를 의미하며, indirect child는 현재 블럭노드에 간접적으로 연결된 Child 블럭노드 (예, 손자, 증손자)를 의미한다.

▶정의 5-N_f (Father 블럭노드의 수) : 현재 블럭노드에서 Root 블럭노드까지의 Father 블럭노드 전체의 수이다. Father 블럭노드의 전체의 수는 direct father 블럭노드의 수 및 indirect father 블럭노드의 수를 합한 것이다. N_c의 경우와 마찬가지로 direct father는 현재 블럭노드에 직접적으로 연결된 Father를 의미하며, indirect father는 현재 블럭노드에 간접적으로 연결된 Father 블럭노드(예, 할아버지, 증조 할아버지)를 의미한다.

▶정의 6-T_c (계산시간) : 각 블럭노드는 그 자체의 계산시간을 가지고 있으며, 계산시간에 대한 예측방법은 2.3절에서 기술된다.

2.3 블럭노드정보

제안된 우선순위 할당 기법을 소프트웨어적으로 구현하기 용이하게 하기 위하여 각각의 블럭노드에 대한 블럭노드정보를 획득하는 과정을 수학적인 모형으로 제시하였다. 각 블럭노드에 대한 블럭노드정보를 얻기 위하여 기본적으로 Depth First 탐색 기법을 적용하였다. Depth First 탐색 기법 및 Breadth First 탐색 기법에 대한 각각의 특성에 대해서는 Barr [2]를 참조 바란다.

2.3.1 MH(Maximum Height)

Top-down 접근 기법 대신에 Bottom-up 접근 기법을 사용하였으며, 각 블럭노드에 대한 MH정보를 얻기 위한 수학적인 모형은 다음과 같다.

(1) Leaf 블럭노드 : Leaf 블럭노드에는 MH=1를 지정하는데 그 이유는 상기에서 기술하였듯이 Bottom-up 기법을 사용하였기 때문이다.

(2) 기타 블럭노드

CASE 1 : 현재 블럭노드 i가 여러개의 Child 블럭노드들을 가지고 있으며, 그러한 Child 블럭노드들이 동일한 Level에 위치하고 있을시 현재 블럭노드의 MH를 구하는 수학적 모형은

$MH(n_i) = MH(n_d) + 1$이며, 단 n_d는 direct child이다.

CASE 2 : 현재 블럭노드 i가 여러개의 Child 블럭노드를 가지고 있으며, 각각의 Child 블럭노드들이 서로 다른 Level에 위치하고 있을 시, 현재 블럭노드의 MH를 구하는 수학적 모형은 $MH(n_i) = \max\{MH(c_1), MH(c_2)\} + 1$이며, 단 c_1, c_2는 두개의 서로 다른 Level에 위치한 direct child이다.

상기 (1) 및 (2)를 종합하여 현재 블럭노드의 MH를 구하는 수학적인 모형을 전체적으로 살펴보면 다음과 같다.

$$MH(n_i) = \begin{cases} 1 & : Child(n) = \phi \\ MH[child(n_1)] + 1 & : MH[child(n_1)] = MH[child(n_2)] \\ MH[child(n_1,\ n_2)] + 1 & : MH[child(n_1)] \neq MH[child(n_2)] \end{cases}$$

단, n_1, $n_2 \in n_i$

2.3.2 N_c(Child 블럭노드의 수)

현재 블럭노드의 N_c 블럭노드정보를 구하기 위해서는 크게 Leaf 블럭노드에 대한 N_c 및 기타 블럭노드에 대한 N_c를 구하는 방법으로 대별할 수 있으며, 그 방법은 아래와 같다.

(1) Leaf 블럭노드 : Leaf 블럭노드는 N_c=0이 되는데 그 이유는 Leaf 블럭노드는 Child 블럭노드를 가지고 있지 않기 때문이다.

(2) 기타 블럭노드 : 현재 블럭노드에서 Leaf 블럭노드까지의 direct 및 indirect child 블럭노드 수의 총합이다. 이것을 수학적인 모형으로 제시하면 다음과 같다.

$$N_c(n) = \begin{cases} 0 & \text{if child}(n) = \phi \\ \sum\{N_c(d) + N_c(i)\} & \text{if 0. W} \end{cases}$$

단, $N_c(d)$는 direct child 블럭노드의 총 수이며, $N_c(i)$는 indirect child 블럭노드의 총 수이다.

2.3.3 N_f(Father블럭노드의 수)

현재 블럭노드의 N_f 블럭노드정보를 구하는 방법도 N_c의 블럭노드정보를 구하는 것과 비슷한 방법으로 수행되며, 다만 N_f의 블럭노드정보는 현재 블럭노드에서 Root 블럭노드까지의 Father 블럭노드 수의 합을 의미한다.

(1) Root 블럭노드 : Root 블럭노드는 N_f=0로 지정하는데, 그 이유는 Root 블럭노드는 Father 블럭노드를 가지고 있지 않기 때문이다.

(2) 기타 블럭노드 : 각 블럭노드에서 Root 블럭노드까지의 direct 및 indirect father 수의 총 합이다. 이것을 수학적인 모형으로 제시하면 다음과 같다.

$$N_f(n) = \begin{cases} 0 & \text{if father}(n) = \phi \\ \sum\{N_f(d) + N_f(i)\} & \text{if 0. W} \end{cases}$$

단, $N_f(d)$는 direct child 블럭노드의 총 수이며, $N_f(i)$는 indirect child 블럭노드의 총수이다.

2.3.4 T_c(계산시간)

본 논문에서는 다음과 같은 기존의 Operation Timing을 인용하여 사용한다[6].

Operation	Clock Cycle
+, -, >, <	3
*	11
/	25
=	4

블럭은 노드의 집합으로 구성된다고 할때 응용프로그램은 순차적 블럭, Loop 블럭, Conditional 블럭으로 대별할 수 있는데, 각각의 블럭에 대한 계산시간은 다음과 같이 예측할 수 있다.

(1) 순차적 블럭노드 : 블럭내에 있는 모든 노드의 T_c의 합을 구하면 된다.

(2) Loop 블럭 노드 : Loop 블럭노드에 대한 계산시간 예측을 하는 공식은 $T_c = (T_f - T_s) * N_l$이며, 단 T_f는 Loop이 끝나는 시간, T_s는 Loop이 시작되는 시간, N_l은 예상되는 Loop의 횟수를 의미한다.

(3) Conditional 블럭노드 : Conditional 블럭에서는 조건에 따른 분기(True, False)중에서 가장 긴 계산시간을 선정한다. 이것은 계산시간을 부족하게 예측하는 것보다는 충분한 계산시간을 예측하는 것이 합리적이기 때문이다.

2.4 우선순위 할당 기법의 규칙

제안된 우선순위 할당 기법은 전체적으로 다섯개의 heuristic 규칙으로서 구성되어 있으며, 각각에 대한 규칙은 다음과 같다.

▶ 규칙 1 : Root 및 Leaf 블럭노드를 선정하여 우선순위 Queue의 첫번째 및 마지막에 각각 위치시킨 다음 블럭노드정보를 검사해서 최대 MH를 가진 블럭노드가 유일한 경우, 최대 MH를 가진 블럭노드를 Queue에 할당한다. 블럭노드의 분할은 Minimum Cut-Set 및 계산시간(T_c)를 기본으로 한 Load Balancing을 고려하였으므로 각 블럭노드간의 계산시간 차이는 크지 않게 된다[5]. 따라서 우선순위

결정시 종속관계가 무엇보다도 중요하게 되며, MH는 종속관계의 척도가 되므로 MH가 큰 블럭노드를 먼저 할당한다.

▶ 규칙 2 : 블럭노드정보로부터 MH를 검사했을 때 만약 동일한 MH를 가진 1개이상의 블럭노드가 존재할 시에는 N_c가 큰 블럭노드를 Queue에 먼저 할당하는 이유는 N_c가 크다는 것은 그 블럭노드를 기다리는 블럭노드들이 많다는 의미이기 때문이다.

▶ 규칙 3 : 만약 어떤 블럭노드가 MH 및 N_c가 동일시에는 N_f를 검사한다. 그리고 N_f가 적은 블럭노드를 먼저 Queue에 할당한다. N_f가 적은 블럭노드를 할당하는 이유는 N_f가 적다는 것은 N_f가 큰 블럭노드 보다 father 블럭노드가 조기에 처리되기 때문이다.

▶ 규칙 4 : 만약 어떤 블럭노드가 MH, N_c 및 N_f가 동일시에는 T_c를 검사한다. 그리고 T_c가 큰 블럭노드를 먼저 Queue에 할당한다. 실제적으로 T_c가 큰 블럭노드를 먼저 처리해야 하는가, 혹은 T_c가 적은 블럭노드를 먼저 처리해야 하는가의 문제는 현재까지 많은 연구를 해왔으나 아직도 통일된 결론을 얻지 못하고 있는 실정이다. 현재까지의 소결론은 응용프로그램의 종속관계 특성에 따라 T_c가 큰 블럭노드 혹은 적은 블럭노드를 먼저 처리해야 한다는 학설이 있다.[1][6]. 본 우선순위 할당 기법은 복잡한 종속관계를 나타내는 수학적인 계산에 적합한 기법에 적용하고자 하므로 T_c가 큰 블럭노드를 먼저 처리하는 기법을 채택하였다.

▶ 규칙 5 : 상기 1)-4)의 규칙이 만족치 못할시에는 임의로 Left-to-Right 순서대로 Queue에 할당한다.

Ⅲ. 알고리즘

본장에서는 2장에서 기술된 규칙을 근거로 하여 소프트웨어적으로 구현할 수 있는 알고리즘을 제시하고 예제를 선정하여 실제적인 우선순위 할당 기법을 적용한 결과를 기술하였다.

3.1 알고리즘

본 알고리즘을 적용하기 위한 입력으로는 응용프로그램으로부터 소프트웨어 Tool을 통해 형성된 종속그래프[3.2절 참조]를 사용하며, 알고리즘의 출력 결과는 우선순위 할당 List로 나타난다.

(1) 각 블럭노드는 블럭노드정보를 가지고 있으며, 블럭노드정보 형태는 (MH, N_c, N_f, T_c) 이다.

(2) Root 블럭노드를 찾아서, Queue의 첫 번째에 할당한다.

(3) 만약 최대 MH를 갖는 블럭노드가 유일하면, 최대 MH를 갖는 블럭노드를 Queue에 할당한다.

(4) 만약 동일 MH를 갖는 블럭노드가 1개 이상이면, N_c를 검사하고 N_c가 큰 블럭노드를 Queue에 먼저 할당한다.

(5) 만약 동일 MH, N_c를 갖는 블럭노드가 1개 이상이면, N_f를 검사하고 N_f가 작은 것을 Queue에 먼저 할당한다.

(6) 만약 동일 MH, N_c 및 N_f를 갖는 블럭노드가 1개 이상이면, T_c를 검사하고 T_c가 큰

것을 Queue에 먼저 할당한다.

(7) 만약 모든 것이 동일하면, Left-to-Right순으로 Queue에 할당한다.

(8) 상기 (1)-(7)의 단계를 모든 블럭노드가 Queue에 할당될 때까지 반복한다.

3.2 실행단계 및 예제

본 예제는 하나의 종속그래프를 선정하여 제안된 우선순위 할당 알고리즘을 이용하여 각 블럭노드들에 대한 우선순위를 결정하고, 실제적으로 프로세서들에게 할당될 때 전체적인 실행시간을 단축시키는 결과를 보여준다.

이러한 종속그래프는 실제적으로 어떻게 형성할 수 있는가 하는 실행단계를 먼저 살펴본 다음, 선정된 종속그래프에 제안된 우선순위 할당 알고리즘을 적용하여 보도록 한다.

3.2.1 실행단계

실행은 몇 단계로 대별할 수 있으며, 그 절차는 다음과 같다.

첫째, 사용자는 프로프래밍 언어를 이용하여 프로그램을 작성한다. 본 논문에서 사용된 프로그래밍 언어는 C언어이며, 서로 다른 프로그래밍 언어를 사용하여 프로그램을 작성할 수 있다.

둘째, 적합한 컴파일러를 사용하여 작성한 응용프로그램을 컴파일시켜 중간코드 (intermediate code)를 생성한다.

셋째, 컴파일러로 컴파일시켜서 얻은 중간코드를 가지고 종속그래프를 생성할 수 있는 소프트웨어 도구(software tool)를 이용하여 종속그래프를 형성한다.

넷째, 형성된 종속그래프를 가지고 제안된 우선순위 할당 알고리즘을 적용하여 각 블럭노드에 대한 프로세서 할당 우선순위를 결정한다.

3.2.2 예제

3.2.1절에서 제시한 실행단계를 거쳐서 종속그래프가 형성되면 각 블럭노드에 대한 블럭노드정보는 Depth First 탐색방법 및 제안된 수학적인 모델을 가지고 구할 수 있다. 앞에서 이미 기술하였지만 블럭노드정보는 MH, N_c, N_f 및 T_c로 구성되어 있다.

〈그림 3-1〉 종속그래프를 통해 각 블럭노드들에 제2장의 수학적 모형을 적용하여 우선순위를 결정하기 위한 각각의 블럭노드정보를 계산하는 과정을 살펴보면 다음과 같다.

B9 블럭노드는 Leaf 블럭노드로서 MH=1을 지정하며, B5 및 B8 블럭노드는 2.3.1절의 기타 블럭노드 CASE 1에 해당 되므로 MH(B9)+1=2, B6 및 B7 블럭노드는 B5, B8 블럭노드와 같은 경우에 해당되므로 MH(B8)+1=3, B4 블럭노드는 2.3.1절의 기타 블럭노드 CASE 2에 해당되므로 max(MH(B5), MH(B6), MH(B7))+1=4가 된다. 비슷한 방법으로 계산하면 나머지 블럭노드에 대한 MH값은 〈그림 3-1〉의 각 블럭노드정보 중 첫번째와 같이 된다.

N_c의 경우 Root블럭노드인 B1 블럭노드는 direct child 블럭노드의 수 2, indirect child 블럭노드 수 6으로 N_c는 8이 된다. 비슷한 방법으로 나머지 블럭노드에 대한 N_c를 구할 수

있으며, 그 결과는 〈그림 3-1〉의 각 블럭노드 정보 중 두번째와 같이 된다(제2.3.2절 참조).

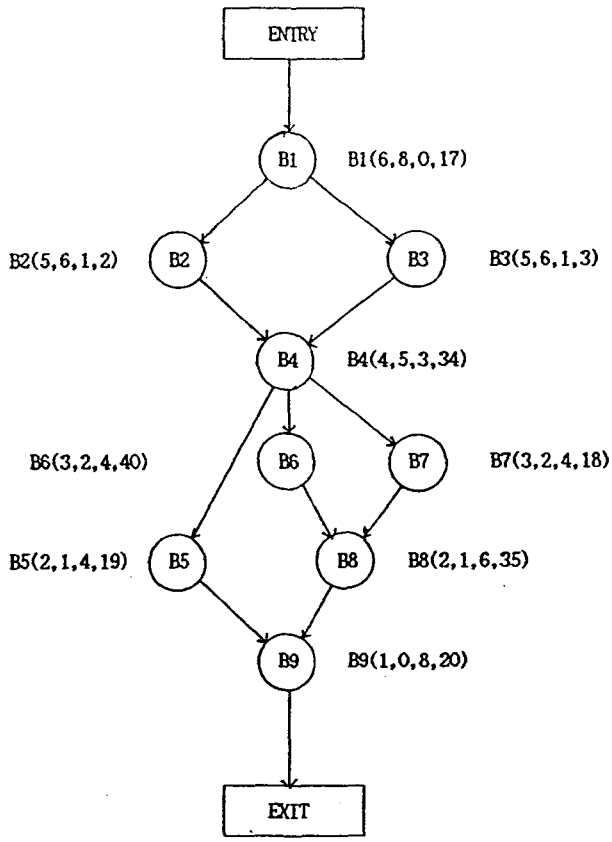
N_f의 경우, B5 블럭노드는 direct father 블럭노드의 수 1, indrect father 블럭노드의 수가 3으로서 N_f는 4가 된다. 비슷한 방법으로 나머지 블럭노드에 대한 N_f를 구할 수 있으며, 그 결과는 〈그림 3-1〉의 각 블럭노드정보 중 세번째와 같이 된다(제2.3.3절 참조).

각 블럭노드에 대한 T_c 값은 2.3.4절과 같은 방법으로 그할 수 있으며, 그 결과는 〈그림 3-1〉의 각 블럭노드정보 중 마지막에 위치한다.

이와 같이 계산된 각 블럭노드정보를 기초로 3.1절의 알고리즘을 적용하여 보면 출력되는 우선순위 List는 B1, {B2 : B3}, B4, {B6 : B7}, {B5 : B8}, B9이 된다.

상기의 출력 결과는 B1→{B2, B3 병행처리}→B4→{B6, B7 병행처리}→{B5, B8 병행처리}→B9의 순서로 처리될 수 있음을 의미한다.

상기 종속그래프에서 B5 블럭노드의 처리시간이 B6 블럭노드나 B7 블럭노드와 B8 블럭



〈그림 3-1〉 종속그래프

노드로 이어지는 Path의 전체 처리시간 보다 긴 경우 B5 블럭노드는 B8 블럭노드와 병행처리되는 것보다 B6 블럭노드나 B7 블럭노드와 병행처리되는 것이 더 효율적이다. 그러나 종속그래프를 형성하는 블럭노드를 분할할시, 사용된 분할기법에서 계산시간을 포함한 Load Balancing이 고려되므로 블럭노드들간의 계산시간에는 큰 차이가 없음을 생각하면 본 논문에서 제안한 Heuristic Rule이 효율적임을 알 수 있다[5].

Ⅳ. 결론

우선순위 할당 기법은 종속관계 그래프의 각 블럭노드들에 대해 우선순위를 부여함으로써 물리적 매핑을 보조하는 중요한 역할을 수행한다. 동일한 MH를 갖는 수십개, 수백개의 블럭노드들이 동시에 프로세서를 할당받으려고 할 때, 동일 MH에 있는 각 블럭노드들에 대한 우선순위를 부여함으로써 전체적인 실행시간을 단축시키기 위한 것이 본 논문의 목적이며, 제안된 우선순위 할당 기법은 궁극적으로 보다 효과적이고 효율적인 다중 프로세서 시스템이 달성되도록 하는데 있다.

종속관계 그래프 상에 존재하는 각 블럭노드는 Maximum Height(MH), Child 블럭노드의 수(N_c), Father 블럭노드의 수(N_f), 계산시간(T_c)의 네가지 블럭노드정보를 가지고 있는데, 각 블럭노드에 대한 우선순위는 각각의 블럭노드가 가지고 있는 네 가지의 블럭노드정보를 사용한 heuristic 알고리즘을 통해 결정된다.

기존의 우선순위 할당 기법 연구는 한개 또는 두개의 요소를 고려해서 우선순위를 결정한 것으로 하드웨어 가격의 저하와 기술 발전에 따른 대량의 프로세서를 이용하는 현재의 다중 프로세서 시스템에는 미흡하다. 반면에, 본 논문에서 제안한 네 가지의 블럭노드 정보를 사용한 Heuristic 알고리즘은 가용한 블럭노드정보를 최대한 활용함으로써 보다 합리적인 우선순위를 결정할 수 있으므로 매우 복잡한 종속관계를 갖고 있는 그래프 처리 및 대량의 프로세서가 참여하는 다중 프로세서 시스템에 적합한 우선순위 할당 기법이라 할 수 있다.

제안한 우선순위 할당 기법의 이해를 돕기위하여 종속그래프를 형성하는 실행절차를 설명하였고, 예제를 통해 우선순위 할당 기법을 적용하고 그 출력 결과가 병렬처리를 달성할 수 있음을 제시하였다. 제안된 기법을 완벽하게 실용화하기 위해서는 정확한 계산시간 예측이 선행되어야 하므로 이러한 예측방법에 대한 추가적인 연구가 필요하다.

V. 참고문헌

1. 조유근, 고건, 운영체제, 홍릉과학출판사, 1984.
2. Edited by Barr, A. and Feigenbaum, E. A., *The Handbook of Artificial Intelligence*, Vol. 1, 1981.
3. Coffman, E. and Graham, R., "Optimal Scheduling for Two Processor System," *Acta Informatica 1*, 1972, pp. 200-213.
4. Kasahara, H. and Narita, S., "Practical Multiprocessor Scheduling Algorithm for Efficient Parallel Processing," *IEEE Trans. on Computer*, Vol. c-33, No. 11, 1984, pp. 1023-1029.
5. Kim, Hwa-Soo, "A Large-Grain Mapping Approach for Multiprocessor System through Data Flow Model," *Ph. D Thesis*, CWRU, 1990.
6. Koren, I. and Silverman, G., "A Direct Mapping of Algorithms onto VLSI Processing Arrays based on the Data Flow Approach," *IEEE Parallel Processing*, 1983, pp. 335-337.
7. Potkonjak, M. and Rabaey, J., "A Scheduling and Resource Allocation Algorithm for Hierarchical Signal Flow Graph," *26th ACM/IEEE DAC*, 1989, pp. 7-12.
8. Shih, L., "Automatic Synthesis of Control for Multiprocessor Systems through Fine-Grain Mapping," *Ph. D Thesis*, of Computer Eng. and Sci., CWRU, 1990.
9. Sarker, V., "Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors," *TR CSL-TR-87-328*, Dept. of Electrical Engineering and Computer Science, Standford University, 1988.