論文2000-37SD-9-9

BILI-하드웨어/소프트웨어 분할 휴리스틱
(BILI-Hardware/Software Partition Heuristic)

吳 鉉 玉 * · 河 舜 會 *
(Hyu Nok Oh and Soon Hoi Ha)

요 약

이 논문에서는 Best Imaginary Level-Iterative(BILI) 분할 방법이라 부르는 새로운 하드웨어/소프트웨어 분할 알고리즘을 제안한다. 이 분할 알고리즘은 BIL 이질 다중 프로세서 스케줄링을 반복적으로 적용하는 방법이다. 이 분할 알고리즘은 여러 개의 하드웨어와 소프트웨어로 이루어진 시스템에 대해서 분할을 할 수 있을 뿐만 아니라, 여러 가지의 구현 가능성 중에서 적은 비용의 구현을 선택하는 문제까지 해결한다. 이 분할 알고리즘은 기존의 분할 알고리즘인 GCLP와 비교하여 약 15%의 비용 감소를 가지고, 항상 최적의 해를 찾는 정수 선형 프로그래밍과 비교하여 약 5%정도의 비용 증가를 가진다.

Abstract

This paper presents a fast partitioning heuristic for hardware/software codesign called Best Imaginary Level-Iterative(BILI) partitioning which iteratively applies BIL heterogeneous multiprocessor scheduling heuristic to minimize the cost within the given time constraint. The proposed algorithm solves the partitioning problem with the implementation bin selection problem as well as architectures with multiple software modules. It costs about 15% less than the GCLP and at most about 5% more than the optimal solution obtained by the Integer Linear Programming(ILP) algorithm.

I. 서 론

내장형 시스템은 크게 주문형 반도체(Application specific IC : ASIC)와 FPGA등의 하드웨어와 프로세서등(DSP, CPU)의 소프트웨어를 구동하기 위한 부분으로 이루어져 있다[1]. 그리고 이러한 시스템 설계의 중요한 목표중의 하나는 주어진 제약조건을 만족하면서 최소의 비용으로 시스템을 구현하는데 있다. 그래서 시스템 설계 자동화의 중요한 부분 중의 하나가 기술된 알고리즘에서 어느 부분을 어떠한 형태로 구현해야 되는 지에 대한 결정이다. 즉 각각의 부분을 하지에 대한 결정과, 어떤 순서로 각 부분을 수행해야 좋은 지에 대한 판단이 중요한 문제이다. 이러한 일련드웨어로 구현해야 할 지 소프트웨어로 구현해야 할의 판단 과정을 분할이라 부르며, 특히 하드웨어/소프트웨어 분할이라 부른다. 이러한 분할은 단지 각 부분의 수행 순서와 하드웨어 또는 소프트웨어로 수행되는지에 대한 결정뿐만 아니라, 만일 하드웨어로 구현된다면 어떤 형태로 구현되어야 하는지에 대해서도 결정해야 한다. 예를 들어, 어떤 덧셈기가 하드웨어로 구현되는 것으로 결정되었을 때에, 실제 구현에 따라서 자리올림전파 덧셈기(ripple-carry adder), 자리올림선택 덧셈기(carry-select adder), 조건합 덧셈기(conditional sum adder)등 여러 가지 구현 가능성을 가질 수 있다. 본 논문에서 다루는 분할 알고리즘은 이러한 문제까지 포함한다.

* 正會員, 서울大學校 電氣 및 컴퓨터工學部
(School of Electrical Engineering & Computer Science, Seoul National University)
接受日字:1999年3月8日, 수정완료일:2000年7月29日

1. 시스템 수준에서의 설계 방법론

시스템 설계에서는 분할되는 대상의 기본적인 크기에 따라 다른 접근 방식을 취한다. 이 논문에서 취하는 설계 방법론은 시스템 수준의 큰 단위의 태스크(또

는 노드)를 분할의 기본 단위로 생각한다. 이러한 시스템 수준에서의 기술은 시스템 설계에 있어서 두 가지 장점을 준다. 첫째는 분할의 대상이 되는 노드들이 알고리즘 레벨에서 기술되기 때문에 실제적인 큰 크기의 응용을 설계하는 것이 용이하다. 두번째로 시스템 수준에서의 기술은 기술한 알고리즘이 실제 시스템으로 구현될 때에 대한 제약을 두지 않고 있기 때문에 여러 가지 형태의 시스템 설계가 가능하다는 점이다. 즉 소프트웨어로만 또는 하드웨어로만 이루어진 시스템뿐만 아니라, 그 중간쯤 되는 소프트웨어와 하드웨어와 섞여 있는 시스템으로도 구현 가능하게 된다. 이러한 유연성은 구현 비용을 적게 하면서, 주어진 제약 조건을 만족하는 성능을 가지는 시스템의 설계를 가능하게 해 준다.

2. 시스템 설계 과정

이 논문에서 취하는 설계 자동화의 과정이 그림 1에 나타나 있다. 이 논문에서 사용하는 시스템 설계 자동화 틀에서는 각 디자인 요소가 소프트웨어와 하드웨어로 구현될 때의 성능과 비용이 미리 계산되어 라이브러리화 되어 있다고 가정한다. 그리고 다양한 구조의 시스템에 대해서 분할을 수행함으로써, 적은 비용으로 주어진 알고리즘을 구현할 수 있는 시스템을 찾을 수 있도록 한다.

이러한 시스템 설계 자동화의 과정에서부터 분할 알고리즘이 갖추어야 할 두가지 요구 사양을 생각할 수 있다. 첫째로, 분할 알고리즘은 성능의 큰 희생없이 빠르게 수행될 수 있어야 한다. 대개의 시스템 설계의 경우 분할은 사용자가 입력 요소값을 달리 하면서 여러번 수행될 것이고, 또 현재의 상황에서 볼 때 앞으로 더욱 더 시스템의 복잡도가 증가하면서 문제의 크기도 같이 증가할 것으로 예상되기 때문이다. 두 번째로, 분할 알고리즘은 어떤 특정한 시스템 구조만을 지원해서는 안되며, 다양한 시스템 구조를 모두 지원할 수 있어야 한다. 이전의 분할 알고리즘은 대개가 하나의 소프트웨어 모듈과 하드웨어 요소로 이루어진 시스템을 가정했다. 즉 한 개의 소프트웨어, 한 개의 하드웨어로 이루어진 시스템에서의 분할을 생각했다. 그러나 본 논문에서 제안하는 분할 알고리즘은 그러한 제약 조건이 없이 여러 개의 소프트웨어 요소와, 여러 개의 하드웨어 요소로 이루어진 시스템에 대해 분할을 수행할 수 있다. 그 뿐만 아니라, 주어진 함수 블록에 대한 다양한 구현들 중 적절한 구현을 선택하는 기능도 제공한다. 이러한 알고리즘의 유연성은 이 알고리즘이 일반적인 임의의 이종 다중 프로세서 스케줄러를 이용하는 데 기인하다. 분할이 끝나면 분할된 그래프는 각각 소프트웨어 요소와 하드웨어 요소들에 맞게 합성될 것이다. 이렇게 합성된 코드를 실제 시스템에 내장하면 하나의 시스템이 완성되는 것이다.

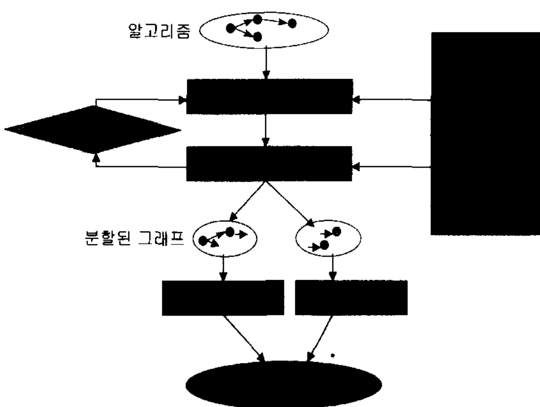


그림 1. 시스템 설계의 과정
Fig. 1. The flow of system design.

Ⅱ. 이전 연구들

하드웨어-소프트웨어 분할 문제는 이미 잘 알려진 NP-hard 문제이며 시스템 설계에서 중요한 부분이므로, 많은 분할 알고리즘의 개발되었다. 분할을 수행하는 방식 중 하나의 형태가 긴 시간동안 수행하여 최적 또는 최적에 가까운 결과를 얻는 방식이다. Henkel등의 시뮬레이티드 어닐링(simulated annealing)[3], Kumar등의 수학적인 프로그래밍[4], Hu등의 가지치기(branch and bound)[5], Niemann과 Marwedel의 혼성 선형 정수 프로그래밍(mixed integer linear programming)[6]등이 이러한 접근 방식에 속한다. 일반적으로 이러한 방식은 주어진 시스템에 맞는 분할을 적절한 시간내에서 수행하기 위해서는 입력 노드의 수가 수십 개 이내이어야 한다. 즉 이러한 접근 방법은 시스템이 고정되어 있고, 문제의 크기도 작은 경우에만 적용 가능한 알고리즘들이다. 이러한 알고리즘은 분할하는데 시간이 많이 걸리기 때문에 시스템 개발자가 시스템 구조를 여러 가지로 바꾸고 분할하고 평가하는 방식으로 시스템을 개발하는 경우에는 적절하지

않은 방법이다.

반면, 또 다른 접근 방법은 약간의 성능 저하를 감수하고 수행 시간을 대폭적으로 줄이는 방법이다. Gupta[7][8]는 데이타 의존 지연 연산(data dependent delay operation)을 제외한 모든 연산을 초기에는 하드웨어로 설정한 후, 주어진 시간 제약 조건을 만족하는 범위에서 그리디(greedy) 알고리즘을 사용하여 소프트웨어로 한 연산씩 보내는 방법을 고안하였다. 또 다른 분할 방법은 우선 대강의 분할을 수행한 다음, 조금씩 반복해서 성능을 향상시키는 방법이다[9][10]. 이러한 알고리즘은 복잡한 시스템 설계와 반복적인 시스템 구조 변경에 대해서 여러번 분할을 실시해도 그리 많은 시간이 걸리지 않는 알고리즘이기 때문에 실제 시스템 설계에 적용하는 데 좋다. 그러나 이러한 접근 방법에서도 각 노드의 구현 방식을 다양화할 수 없다는 한계점을 가지고 있다. 이에 대해서 Kalavade는 global critical/local phase(GCLP)[11]라는 전역적인 정보와 지역적인 정보를 이용하여 분할을 수행하는 선형 스케줄링 알고리즘을 제안하였다. 이 방법은 DFG로부터 분할을 시도한다. 이 방법도 주어진 시간 제약 조건을 만족하면서 하드웨어 자원을 적게 사용하기 위한 방법이다. GCLP는 선형 스케줄링에 기반한 방법으로 분할할 노드가 하드웨어로 구현할 지 소프트웨어로 구현할 지 결정하는 과정에서 판단 기준으로 삼는 목적 함수를 GC에 따라 다르게 적용하는 방법이다. GC란 분할되지 않은 노드 중 주어진 시간 제약 조건을 만족하기 위해 하드웨어로 매핑되어야하는 노드의 비율을 나타내다. LP란 GC의 단점을 보완하기 위한 개념으로 각 노드의 특성(하드웨어로 구성하기 어렵다거나, 소프트웨어로 구현하기 어렵다는 정도)을 수치화한 값이다. 이 방법에서는 GC+LP값에 따라 목적 함수를 바꾼다. 만일 이 값이 주어진 임계값보다 크게 되면 시간 제약 조건을 만족하는 것이 중요하므로 분할할 노드를 하드웨어로 매핑하는 함수를 수행한다. 그렇지 않은 경우에는 소프트웨어로 매핑하는 함수를 수행한다. 이 알고리즘은 다시 각 노드에 대해 하드웨어와 소프트웨어로 매핑하는 과정 뿐만 아니라 하드웨어로 매핑된 노드에 대한 구현 방법까지를 생각하는 확장된 분할 문제를 푸는 알고리즘으로 향상되었다[12]. 그러나 이 GCLP알고리즘도 대상이 되는 시스템의 구조를 한 개의 소프트웨어와 한 개의 하드웨어(여러 개의 하드웨어 요소를 포함한 경우를 고려하다더지, 통신 시간, 어떤 하드웨어 요소로 할당되는 지에 대한 언급이 없다)로 구성된 경우로 한정하고 있기 때문에 여러 소프트웨어와 하드웨어들을 포함한 시스템에 적용할 수 없는 한계를 가진다.

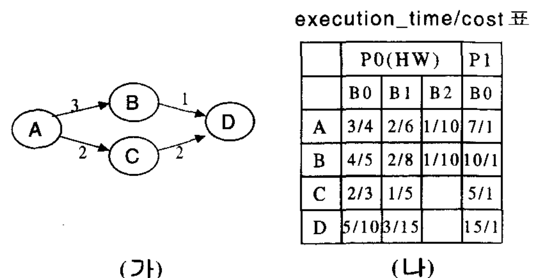
Ⅲ. 하드웨어/소프트웨어 분할 문제 정의

이장에서는 본 논문에서 제안하는 분할 알고리즘이 상위 수준의 알고리즘 기술과 각 노드의 다양한 구현의 선택, 넓은 범위의 시스템 구조를 지원하기 위한 분할 문제에 대해서 정의한다.

1. 입력 그래프

분할 알고리즘은 방향성 비순환 그래프(direct acyclic graph : DAG)를 입력으로 한다. 입력 예가 그림 2에 나타나 있다. 방향성 비순환 그래프의 노드는 실행되어야 할 태스크를 나타내고 간선은 두 노드 간의 순서관계를 지시한다. 각 간선 위의 숫자는 프로세서간 통신 시간(interprocessor communication : IPC)을 나타낸다. 만일 두 노드가 같은 프로세서에 스케줄된다면 프로세서간 통신 시간은 무시된다.

N	$=\{n_1, n_2, \ldots, n_n\}$: 입력 그래프상의 노드 집합.
E	$=\{e_{i,j} \mid n_i, n_j \in N$, 노드 n_i 에서 n_j로 가는 간선이 있을 때$\}$: 에지 집합.
CO	$=\{comm_{i,j} \mid e_{i,j} \in E\}$: 노드들 사이의 통신 시간 집합.



	P0(HW)			P1
	B0	B1	B2	B0
A	3/4	2/6	1/10	7/1
B	4/5	2/8	1/10	10/1
C	2/3	1/5		5/1
D	5/10	3/15		15/1

그림 2. 분할 알고리즘의 입력 예
Fig. 2. The partition input example.

2. 실행 시간과 비용

프로세서-구현 j-k(j번째 프로세서의 k번째 구현을 가르킨다)에서의 노드 i의 수행시간과 비용은 각각 수

행시간(i,j,k), 비용(i,j,k)로 나타내며, 각 노드-프로세서-구현 쌍에 대해서 컴파일 시에 정적으로 알려져 있다고 가정한다. 그림 2 (나)에 그러한 정보가 표의 형태로 나타나 있다. 예를 들어 수행시간(A,P0,B0) = 3, 비용(A,P0,B0) = 4이다. 노드 i가 프로세서-구현 j-k에서 수행될 수 없다면 무한대로 표시한다. 예를 들어 노드 D가 프로세서 P1에서는 스케줄 될 수 없다면 수행시간(D,P1,B0) = ∞이다. 이러한 경우는 노드를 수행하기 위한 명령어나 자원이 프로세서에 없는 경우에 발생한다. 비용이라 함은 대개 하드웨어에서의 면적(또는 전력 소비)과 소프트웨어에서의 메모리 요구량으로 생각할 수 있다.

> T = { $t_{i,j,k}$ | $t_{i,j,k} = execution_time(i,j,k)$ } : 수행시간 집합. $t_{i,j,k}$ 는 노드-프로세서-구현 i-j-k의 수행 시간을 나타낸다.
> C = { $cost_{i,j,k}$ | $cost_{i,j,k} = cost(i,j,k)$ } : 비용 집합. $cost_{i,j,k}$ 는 노드-프로세서-구현 i-j-k의 비용을 나타낸다.

3. 제약 조건과 분할 목표

이 논문에서의 분할의 목표는 주어진 시간 제약 조건을 만족하면서 적은 비용의 시스템 구현을 찾는데 있다. 즉 주어진 그래프의 스케줄 결과에서 가장 마지막에 끝나는 시각(스케줄 길이 또는 makespan이라 한다)이 주어진 시간 제약 조건보다 작거나 같게 하면서 비용이 적게 드는 노드-프로세서-구현의 매핑을 찾는 것이 분할의 목표이다.

> I = { $i_{i,j,k}$ | $i_{i,j,k} \in \{0,1\}$ }
> 이진 변수 $i_{i,j,k}$ 은 구현을 나타낸다. 노드 i가 프로세서 j에 스케줄되고 k형태로 구현된다고 하면, $i_{i,j,k}=1$, 아니면 0. 즉 $\sum_{j,k} i_{i,j,k}=1$.
> ST = { st_i | $st_i \in \{0,1,2,\ldots,D\}$, $n_i \in N$ } : 노드 i의 스케줄 가능한 시작 시각을 나타낸다. D는 제약 시간이다.
> $$st_f + \sum_{j,k}(t_{f,j,k} \times i_{f,j,k}) \le D$$
> 노드 f는 끝단의 노드 집합에 속하는 노드들이다. 이러한 노드 모두에 대해서 시간 제약 조건을 만족해야 한다.
> 분할 목적 함수는 다음과 같다.
> minimize ($\sum_{i,j,k} cost_{i,j,k} \times i_{i,j,k}$)

4. 대상 아키텍처

이종 다중 프로세서 스케줄러가 지원한다면 어떠한 형태의 구조도 분할 알고리즘이 지원한다. 이 논문의 분할 알고리즘에서 사용하는 이종 다중 프로세서 스케줄링 알고리즘으로 Best Imaginary Level(BIL) 스케줄링 알고리즘[2]을 선택하였다. 스케줄링 알고리즘에서 프로세서는 통신 중에 계산을 병행해서 수행할 수 있다. 대상이 되는 아키텍처는 여러 종류의 하드웨어와 소프트웨어 요소로 이루어 질 수 있다. 그리고, 통신에 필요한 하드웨어 또는 소프트웨어 면적은 분할의 결과에 크게 영향을 주지 않으므로 무시한다.

Ⅳ. BIL-Iterative (BILI) 분할 알고리즘

1. 구현을 고려한 분할 알고리즘

이 논문에서 제안하는 분할 알고리즘은 이종 다중 프로세서 스케줄링 알고리즘을 중요한 부분으로 사용한다. 하지만, 이종 다중 프로세서 스케줄링 알고리즘들[2][13][14]은 스케줄 할 때 비용의 요소는 무시하고 단지 가장 짧은 스케줄 길이만을 생성하려고 하기 때문에 바로 분할 알고리즘에 적용할 수는 없다. 그래서 이 논문에서 제시하는 분할 알고리즘은 분할 문제를 반복적인 스케줄링 문제로 바꾸어 문제를 해결하고자 한다.

이 논문에서 제안하는 알고리즘을 설명하기 전에 먼저 몇가지 용어에 대해서 정의하도록 한다. 어떤 노드가 어떤 프로세서-구현 쌍에 스케줄될 수 없을 때, 이 노드는 그 쌍에 대해 잠겨(locked)있다라고 하고, 반대의 경우는 풀려(unlocked)있다라고 한다. 제안하는 알고리즘은 소프트웨어에서부터 시작하는 알고리즘으로써, 모든 노드가 처음에는 각각 가장 적은 비용의 프로세서-구현 쌍에 대해서만 스케줄될 수 있고, 다른 프로세서-구현 쌍에 대해서는 잠겨있다. 이렇게 노드-프로세서-구현 쌍을 잠그는 것은 해당되는 쌍의 수행시간을 무한대로 만드는 것으로 가능하다. 다음 단계에서 이종 다중 프로세서 스케줄링 알고리즘을 수행한다. 만약 스케줄된 결과가 사용자가 입력으로 한 시간 제약 조건을 만족한다면, 그것으로 시간 제약 조건을 만족하면서 최소 비용이 되는 분할을 수행한 것이다. 시간 제약 조건을 만족시키지 않는다면, 스케줄 길이를 줄이기 위해서 어떤 잠긴 쌍을 선택해서 풀어놓도록 한다. 풀어놓았을 때 비용이 작게 증가하면서 스케

줄되는 길이는 많이 감소하는 잠긴 노드-프로세서-구현 쌍을 찾아야 한다. 그렇게 하기 위해서 각 쌍의 스케줄 길이 감소 기대값(the expected makespan decrement:EMD)과 비용 증가 기대값(the expected cost increment : ECI)을 정의한다. 그리고, 주어진 제약 시간과 현재의 상황에서의 스케줄된 길이의 차이를 여유분(Slack)으로 정의한다. 여유분은 필요이상으로 스케줄 길이를 줄이는 것을 막는 역할을 한다. 예를 들어 스케줄 길이를 약간만 줄이면 되는 경우에 EMD와 ECI값이 둘 다 큰 것을 선택하면 불필요한 비용 증가가 있을 수 있기 때문이다.

정의 1

Lock(i,j,k) : 원래시간(i,j,k)=수행시간(i,j,k)
수행시간(i,j,k) = ∞
Unlock(i,j,k) : 수행시간(i,j,k) = 원래시간(i,j,k)
EMD(i,j,k) = 수행시간(i,j*,k*) - 수행시간(i,j,k)
ECI(i,j,k) = 비용(i,j,k) - 비용(i,j*,k*)
여유분 = 스케줄시간 - 제약시간
(i : 노드, j : 프로세서, k : 구현)
(j* : 최소 비용 프로세서, k* : 최소 비용 구현)

EMD(i,j,k)와 ECI(i,j,k)는 각각 노드 i가 프로세서-구현 쌍(j,k)에서 풀렸을 때 (스케줄 가능하게 되었을 때) 시간 감소의 기대값과 비용 증가의 기대값을 나타낸다. 스케줄 길이 감소 기대값은 노드의 병렬도가 큰 경우에 그리고 노드가 할당될 수 있는 프로세서가 한 개뿐이 경우에는 통신시간이 감추어져서 정확한 기대값이 된다. 하지만 병렬도가 그리 크지 않은 경우에는 통신 지연 시간이 스케줄 길이에 영향을 미쳐서 실제로 EMD에서 계산한 만큼의 시간 감소 효과를 기대할 수는 없다. 비용 증가 기대값의 경우는 노드가 스케줄될 수 있는 프로세서가 두가지뿐이고 방금 풀린 프로세서-구현으로 스케줄이 되는 경우에는 정확한 값을 나타내지만, 통신 지연 시간등을 고려해서 스케줄에서 다른 결론을 내리면 ECI값은 정확하지 않은 값이 된다.

모든 노드-프로세서-구현 쌍의 EMD와 ECI값을 계산한 후에 잠긴 쌍 중에서 $\frac{\min(EMD, Slack)}{ECI}$ 값이 가장 크게 되는 쌍을 선택한다. 왜냐하면 그 경우가 비용이 적게 들면서 스케줄 길이는 많이 줄일 수 있다고 여겨지기 때문이다. 여기서 $\min(EMD, Slack)$은 필요 이상의 스케줄 길이 감소와 높은 비용 증가를 가지는 쌍의 선택을 막아 준다. 이렇게 선택된 쌍을 Unlock(i,j,k)함수에 의해 푼다. 그러면, 재조정된 수행시간/비용 표로 이종 다중 프로세서 스케줄링을 수행한다. 한 개의 노드-프로세서 쌍에 대해 2개 이상의 구현이 풀려 있다면, 스케줄러는 가장 빠른 구현을 노드-프로세서 쌍의 구현으로 선택한다. 왜냐하면, 스케줄러의 목적은 가장 빠른 수행시간을 만들어 내는 것이기 때문이다. 참고로 풀려진 쌍이라고 해서 그 쌍의 노드가 프로세서-구현 쌍에 스케줄되는 것은 아니다. 이것은 전적으로 스케줄러의 일이다. 즉 스케줄러는 통신 시간을 모두 고려해서 전체 스케줄 길이가 짧게 되도록 노드를 스케줄한다.

앞의 설명을 정리하면 다음과 같은 과정을 주어진 시간 제약 조건이 만족될 때까지 수행한다.

```
1: schedule()
2: if (제약 조건을 만족하지 않으면) goto 3:
   else exit()
3: slack = computeSlack()
4: pair_i_j_k = selectMax(lockedPairList)
   /* selectMax()는 min(EMD, Slack)/ECI 값이 가장
  큰 pair를 선택하는 함수 */
5: unlock(pair_i_j_k)
6: goto 1:
```

자세한 알고리즘은 그림 4 (가)에 나타나 있다.

그림 2의 예를 생각해 보자. 그림 2의 제약 시간을 32라고 가정하자. 먼저 최소 비용을 가지는 쌍을 풀게 되면 표 1의 (가)와 같은 수행 시간을 가지는 입력에 대해서 스케줄을 수행하게 된다. 스케줄된 길이는 37이 되고 이값은 제약 시간을 만족시키지 않는다. 표 1의 (다)는 EMD/ECI값을 계산해 놓은 표이다. 여기서 여유분은 5(=37-32)가 된다. 가장 큰 $\frac{\min(EMD, Slack)}{ECI}$ 값은 C-P0-B0쌍이다. 이 쌍을 풀게 되면 표 1의 (나)와 같은 수행시간표를 가지게 되며 이를 가지고 스케줄하면 32가 되고 이 값은 제약 시간을 만족하게 된다.

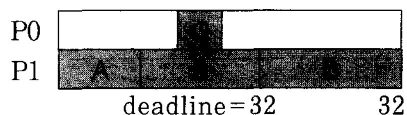


그림 3. 스케줄 결과
Fig. 3. Schedule result.

표 1. 그림 1 예의 분할 과정 (가) 초기 수행시간표 (나) C-P0-B0를 푼 다음의 고쳐진 수행시간표 (다) EMD/ECI 표
Table 1. Partition procedure of fig. 1. (가) Initial time table (나) Modified time table after unlocking C-P0-B0 (다) EMD/ECI table.

(가)

	P0	P1
A	∞	7
B	∞	10
C	∞	5
D	∞	15

(나)

	P0	P1
A	∞	7
B	∞	10
C	3	5
D	∞	15

(다)

	P0(HW)			P1
	B0	B1	B2	
A	4/3	5/5	6/9	-
B	6/4	8/7	9/9	-
C	3/2	4/4		-
D	10/9	12/14		-

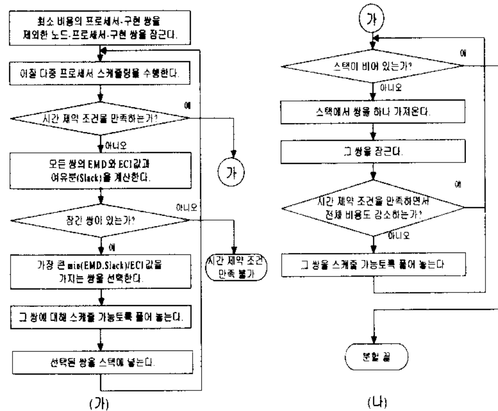


그림 4. BILI 분할 알고리즘
Fig. 4. BILI partition algorithm.

일단 스케줄된 결과가 몇 번의 푸는 과정을 통해서 주어진 시간 제약 조건을 만족하고 나면, 이제는 푸는 과정의 반대의 순서로 풀려진 쌍을 다시 잠금으로써 시스템 구현의 비용을 줄일 수 있는 지 알아본다. 이 과정이 비용을 줄이는 과정이다. 스택에 쌓여진 풀려진 쌍을 하나씩 가져와서 다시 한번 잠근 후 스케줄해 본다. 스케줄된 길이가 시간제약 조건을 어긴다거나 오히려 전체 비용이 증가한다면, 그 쌍은 원래의 풀려진 상태로 돌아가야 한다. 하지만, 만일 시간제약 조건을 만족하고 전체 비용도 감소한다면 그 노드는 그 잠긴 상태로 남아 있을 것이다. 이러한 비용 줄이기 과정을 모든 스택에 쌓인 쌍에 대해서 수행한다. 비용을 줄이는 과정을 통해서 앞에서 수행한 스케줄 길이를 줄이는 과정에서의 지나친 비용 증가와 그리디(greedy) 알고리즘의 한계를 극복할 수 있다. 그림 4에 지금까지 설명한 알고리즘을 요약하였다.

이종 다중 프로세서 스케줄의 시간 복잡도가 $O(S)$라 하면, 제안된 분할 알고리즘의 시간 복잡도는 $O(npb*S)$이다. 스케줄 길이를 줄이는 과정에서 최대 npb번만큼의 스케줄이 있을 수 있고, 비용을 줄이는 과정에서 또 npb번만큼의 스케줄이 있을 수 있기 때문이다. 단 n은 노드의 수를, p는 프로세서의 수를, b는 구현 수를 각각 나타낸다.

2. 자원 공유를 고려한 분할 알고리즘

앞에서 설명한 분할 알고리즘을 자원 공유를 고려한 분할 알고리즘으로 확장하기 위해서 노드-프로세서-구현의 EMD를 확장해서 정의한다. 즉 n개의 노드가 한 개의 자원을 공유할 수 있다면 그 자원을 처음으로 할당할 때의 EMD값은 n*(기존의 EMD)로 정의한다. 왜냐하면 n개의 노드가 한 개의 자원에 스케줄되면 n배로 스케줄이 짧아질 걸로 예상할 수 있기 때문이다. 그림 5와 같은 예를 생각해 보자. C1, C2, C3는 같은 자원을 공유할 수 있는 노드들이라 가정하자. 이 노드를 위한 자원을 할당하면 세 개의 노드는 같은 자원에 동시에 스케줄될 수 있다. 그렇기 때문에 3*(기존 EMD)로 C종류 자원의 EMD를 정의한다. C종류의 자원을 또 할당하면 세개의 노드가 병렬적으로 스케줄되어 스케줄 길이 짧아질 수 있다. 이것을 고려하기 위해서는 두번째 이후로 할당되는 같은 종류의 자원의 EMD값은 그 자원을 할당하기 전의 스케줄 길이와 할당한 후의 스케줄 길이의 차로 정의한다.

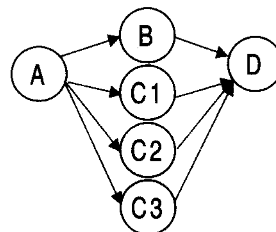


그림 5. 자원 공유를 고려한 예
Fig. 5. The example with resource sharing.

이렇게 자원의 EMD값을 정의하고, 노드-프로세서-구현의 EMD대신 자원의 EMD를 사용하여 앞 절에서 설명한 분할 알고리즘과 동일한 과정을 거쳐 분할

을 수행한다. 이때 이 자원 공유를 고려한 분할 알고리즘의 시간 복잡도는 $O((npb)^2 * S)$이다. 왜냐하면 npb개의 자원에 대해서 풀(unlock) 수 있는데, 자원을 하나 풀기 위해서는 O(npbS)의 시간 복잡도로 EMD값을 매번 계산해야 하기 때문이다.

V. BIL 이종 다중 프로세서 스케줄링 알고리즘

이 논문에서 제안하는 분할 알고리즘은 이종 다중 프로세서 스케줄링 알고리즘을 분할 알고리즘의 핵심으로 사용한다. 특히 이 논문에서 제안한 분할 알고리즘은 BIL 스케줄링 알고리즘[2]을 사용하므로, BILI (BIL-iterative)알고리즘이라 명명했다. 이 장에서는 BIL 스케줄러에 대해 설명하도록 한다.

BIL 스케줄링 기법에서는 시스템이 동종인 경우와 마찬가지로 이종 시스템에서 이상적인 경우에 실제시간과 일치할 수 있도록 노드의 정적 레벨을 정의한다. 이러한 정적 레벨은 BIL (Best Imaginary Level)이라 부르며, 동종 시스템에서 노드의 정적 레벨은 BIL의 특별한 경우로 여길 수 있다. 노드의 BIL은 다음과 같은 가정에 기초한다.

가정 : 모든 자손 노드들은 가장 이른 순간에 스케줄 된다.

물론 이 가정은 항상 실현 가능한 것은 아니다. 즉 둘 이상의 노드들이 동시에 같은 프로세서에 스케줄될 수가 있다. 그래서 최적 가상(Best Imaginary)이라는 말은 이러한 가정이 항상 실제적이지는 않다는 점을 강조하는 것이다.

위의 가정에 따라 노드의 BIL은 다음과 같은 재귀적인 형태로 정의한다.

정의 2

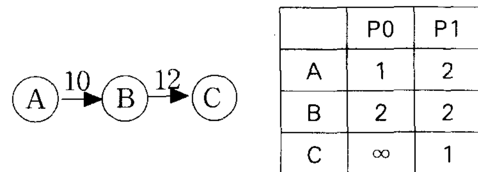
$$BIL(N_i, P_j) = E(N_i, P_j) + \max_{d \in D(N_i)} [\min(BIL(d_l, P_j), \min_{k \neq j}(BIL(d_l, P_k) + d(N_i, d_l))]$$

단, $D(N_i)$는 N_i의 자식 노드 집합을, $d(N_i, d_l)$은 노드 N_i와 노드 d_l 사이의 통신시간을 나타낸다.

그림 6의 예에 대해서 BIL값을 계산해 보도록 하자. 먼저 재귀적인 정의에 따라서 가장 마지막 노드인 C의 BIL값을 구한다. 노드 C는 자손 노드를 가지지 않은 가장 마지막 노드이므로 BIL값은 수행 시간값과 같다. 노드 B와 A의 BIL값은 다음과 같다.

$$\begin{aligned} BIL(B,P0) &= E(B,P0) + \max[\min(BIL(C,P0), BIL(C,P1)+d(B,C))] \\ &= 2+\max[\min(\infty, 1+12)] = 15. \\ BIL(B,P1) &= E(B,P1) + \max[\min(BIL(C,P1), BIL(C,P0)+d(B,C))] \\ &= 2+\max[\min(\infty+12)] = 3. \\ BIL(A,P0) &= E(A,P0) + \max[\min(BIL(B,P0), BIL(B,P1)+d(A,B))] \\ &= 1+\max[\min(15, 3+10)] = 14. \\ BIL(A,P1) &= E(A,P1) + \max[\min(BIL(B,P1), BIL(B,P0)+d(A,B))] \\ &= 2+\max[\min(3, 15+10)] = 5. \end{aligned}$$

노드 N_i가 프로세서 P_j에 스케줄된다고 가정할 때의 $BIL(N_i, P_j)$는 통신시간을 포함한 가장 긴 수행 경로 길이의 최소 값을 나타낸다. BIL은 계산하는 중에 통신시간을 고려하기 때문에 노드의 BIL은 낙관적인 가정 하에서 모든 자손 노드들을 고려한 전역적인 정보로 생각할 수 있다. 그래서 BIL 스케줄러는 전역적인 정보를 가지고 있지 않은 GDL[14] 스케줄러에 비해서 그림 6의 예에서 매우 뛰어난 성능을 나타낸다. 그 결과가 그림 8에 나타나 있다. 스케줄러는 손자 노드인 C를 노드 A의 BIL 계산에서 고려했기 때문에, 노드 A는 프로세서 P1에 스케줄된다. 실제로 BIL 스케줄러는 입력 그래프가 선형일 경우에 항상 최적의 결과를 생성한다.



	P0	P1
A	1	2
B	2	2
C	∞	1

그림 6. 스케줄링 입력 예
Fig. 6. Scheduling input example.

스케줄이 진행됨에 따라 GDL과 마찬가지로 스케줄 가능한 노드들의 우선 순위를 상황에 따라 바뀌는 동적인 레벨을 정의하도록 한다. 따라서 프로세서 P_j에 대한 노드 N_i의 동적인 레벨 BIM(Best Imaginary Makespan)을 다음과 같이 정의한다.

정의

$$BIM(N_i, P_j) = T(N_i, P_j) + BIL(N_i, P_j)$$

$T(N_i, P_j)$는 노드 N_i가 프로세서 P_j에 스케줄 가능한 가장 이른 시각이다.

전체 프로세서 개수가 N개이면 현 시점에서 스케줄 가능한 노드들은 각각의 프로세서마다 하나씩의 총 N개의 BIM값들을 가진다. 현 시점에서 스케줄 가능한 노드들 중에서 스케줄이 최악으로 되는 경우에 성능이 나빠지는 것을 최대한 줄이기 위해서 가장 비관적인 가정 하에서 스케줄될 노드와 프로세서를 선택한다. 즉 BIM값이 큰 노드를 먼저 선택해서 BIM값이 작아지는 프로세서에 스케줄하는 것이다.

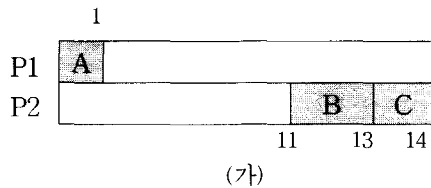
요약하면 다음과 같은 스케줄링 알고리즘이다.

BIL 스케줄링 알고리즘

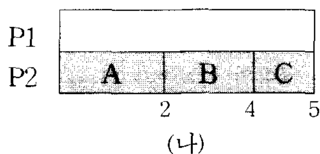
1) 모든 노드의 BIL값을 계산한다.
2) 모든 노드가 스케줄 되었다면 스케줄 끝.
3) 스케줄 가능한 노드의 BIM값을 계산한다.
4) 노드와 프로세서를 선택한다.
5) 선택한 노드를 선택한 프로세서로 스케줄하고, 2단계로 간다.

	P0	P1
A	14	5
B	15	3
C	∞	1

그림 7. BIL값
Fig. 7. BIL.



(가)



(나)

그림 8. 현재의 스케줄링 결정이 노드의 영향을 고려하지 못한 (가) GDL 스케줄러와 이를 고려한 (나) BIL 스케줄러의 성능 비교
Fig. 8. The comparison of the scheduler without considering nodes (가) GDL Scheduler and with cosidering nodes (나) BIL scheduler.

BIL 스케줄러의 시간 복잡도는 $O(n^2 p \log p)$이므로, BIL스케줄러를 사용하는 BILI 분할 알고리즘의 시간 복잡도는 $O(n^3 p^2 b \log p)$이고, 자원 공유를 고려한 경우에는 $O(n^4 p^3 b^2 \log p)$이다. 단 n은 노드 수를, p는 프로세서 수를, b는 구현 수를 나타낸다.

VI. 실 험

1. BILI 대 GCLP

GCLP[11] 알고리즘은 매우 빠른 휴리스틱으로 BILI 알고리즘과 성능을 비교해 보았다. GCLP는 실험적으로 랜덤한 그래프에 대해서 약 30%이내의 비용 증가를 수반한다고 알려져 있다. 그래서 랜덤하게 각각 10, 20, 30, 40, 50, 70, 100, 200, 300, 400개의 노드로 생성된 그래프를 대상으로 성능을 측정하였다.

그림 9는 성능을 비교한 결과이다. 그림에서 x 축은 제약 시간을 나타내며, 오른쪽으로 갈수록 제약 시간이 짧아지는 것을 나타낸다. 즉 시간 제약 조건을 만족하기 위해서 많은 노드가 하드웨어로 가야할 것이다. 예를 들어 100% 제약 시간의 경우는 모든 노드가 최소 비용의 프로세서에 스케줄될 수 있는 것을 나타내는 것이다. 그리고 그러한 경우에는 BILI알고리즘은 GCLP에 비해서 비용 감소가 없음을 알 수 있다. y 축은 GCLP에 대한 BILI알고리즘의 비용 감소 비율을 나타낸 것으로, $\frac{GCLP의\ 비용 - BILI의\ 비용}{GCLP의\ 비용} * 100(\%)$로 계산한 값이다. 시간 제약 조건이 짧아지다 보면-오른쪽으로 가면-시간 제약 조건을 만족하는 분할이 없는 경우도 있는데, 그러한 경우에는 그래프를 그리지 않았다. 그래프를 살펴보면, 시간 제약 조건이 짧아질 수

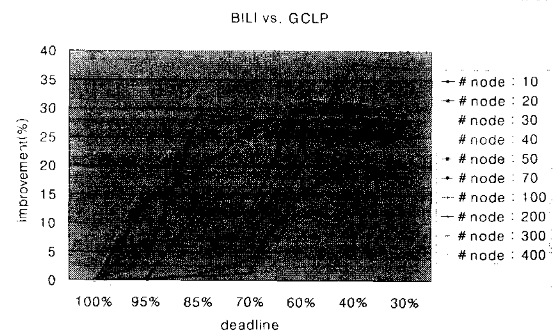


그림 9. BILI와 GCLP와 성능 비교
Fig. 9. The performance comparison of BILI and GCLP.

표 2. 그림 2예에서 BILI알고리즘과 ILP알고리즘의 결과
Table 2. The results of BILI and ILP with figure 2.

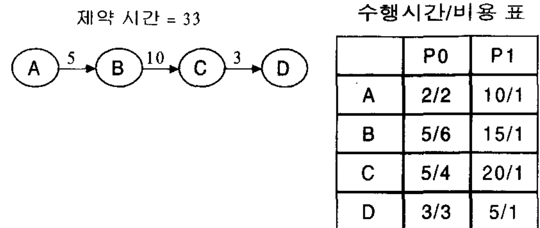
제약 시간	5	8	11	15	18	21	24	27	31	34	37	평균
1. BILI	40	31	25	22	20	18	13	11	8	6	4	
2. 여유분이 없는 경우	40	31	25	25	23	20	13	11	8	8	4	
3. 비용 줄이기가 없는 경우	40	31	25	29	29	20	13	13	8	6	4	
4. ILP	40	30	24	22	20	17	13	11	8	6	4	
1과 2의 비교	1	1	1	1.14	1.15	1.11	1	1	1	1.33	1	1.066
1과 3의 비교	1	1	1	1.31	1.45	1.11	1	1.18	1	1	1	1.096
1과 4의 비교	1	1.03	1.04	1	1	1.06	1	1	1	1	1	1.012

록 BILI알고리즘의 성능이 증가함을 알 수 있다. 평균적으로 보면, BILI알고리즘은 GCLP에 비하여 약 15%정도의 비용감소를 가진다.

2. BILI 대 ILP

본 논문에서 제안한 BILI 알고리즘을 최적의 해를 찾는 ILP(integer linear programming)와 비교 실험하였다. ILP를 푸는 프로그램인 CPLEX를 사용해서, 그림 2의 예제에 대한 ILP 공식을 푼 것이 표2에 나타나 있다. 표 2는 BILI 분할 알고리즘과 ILP의 해를 가장 짧은 시간 제약 조건에서부터 가장 긴 시간 제약 조건까지의 결과를 나타내고 있다. 이 표에서 비교란은 $\frac{BILI\text{의 비용}}{ILP\text{의 비용}}$으로 계산된 값이다. 이 예에서는 여유분을 도입함으로써 평균 약 7%의 비용감소를 가진다. 그러나 여유분은 대개 마지막으로 풀리는 쌍의 선택에 영향을 미치므로 노드 개수가 많은 경우에는 그리 큰 효과를 나타내지 않는다.

그림 10의 예를 살펴보자. 이 경우에 A,C,B,D의 순서로 노드가 풀린다. 그런데 B노드가 풀린 후 스케쥴된 결과는 그림 11과 같이 20이 된다. 그러면, 비용감소 과정을 통해서 C-P0쌍을 잠그고(마지막으로 풀린 쌍은 다시 잠글 필요가 없다) 스케쥴하면 그 결과는 주어진 시간 제약 조건을 만족하지 않게 된다. 그래서 C-P0쌍은 다시 풀린 상태로 돌아간다. A-P0를 잠근 후에 스케쥴하게 되면 그림과 같이 시간 제약 조건 33을 만족하는 작은 비용의 분할이 이루어지게 된다. 비용 감소 과정을 통해서 평균 약 10% 정도의 비용을 감소시킬 수 있다.



	P0	P1
A	2/2	10/1
B	5/6	15/1
C	5/4	20/1
D	3/3	5/1

그림 10. 비용 감소 과정의 효과의 예
Fig. 10. The example to show the effect of cost reduction step.

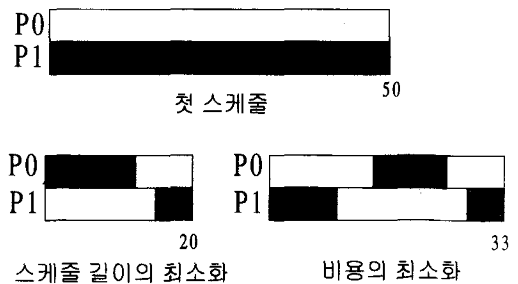


그림 11. 스케쥴된 결과
Fig. 11. The schedule result.

표 3은 다른 종류의 그래프와 더 많은 노드의 수를 가지는 트리 구조에 대한 비교 실험의 결과를 나타낸다. 이 결과에 의하면 BILI 분할은 ILP에 비해서 매우 빠르면서도 비용은 약 6%이하로 증가한다. 실제로 ILP의 경우에는 노드의 수가 30개를 넘어가면 수일동안 계산하더라도 거의 결과를 낼 수 없다. 평균적으로 BILI분할 방법은 ILP에 비해서 약 5%정도의 비용 증가만을 가져온다.

표 3. BILI와 ILP와 CPU 시간 비교
Table 3. The CPU time comparison of BILI and ILP.

	chaos	butterfly	tree	tree
노드 수	11	17	23	31
BILI 수행 시간(초)	0.01~0.04	0.03~0.2	0.07~0.93	0.1~2.25
ILP 수행 시간(초)	0.19~9.57	0.18~364	0.39~ ∞	0.5~ ∞

표 4는 실제 예인 JPEG에 대한 분할 결과이다. 여기서 사용한 JPEG 예제는 모두 8개의 노드로 이루어져 있고, 각 노드마다 4~5개 정도의 구현을 가지고 있다. 그리고 한개의 하드웨어 모듈과 소프트웨어 모듈을 가지고 있는 구조로 가정했다. 이 경우 비용은 ILP에 비해서 최대 15%, 평균적으로는 4.4% 증가했다.

표 4. JPEG 분할 결과
Table 4. JPEG partition result.

제약 시간	5828	8034	9505	10240	평균
ILP 비용	73263443	13900850	4403364	864804	
BILI 비용	76054204	14553724	5056238	864804	
비용 증가	1.038	1.047	1.148	1	1.044

16-QAM(Quadrature Amplitude Modulation)의 예는 173개의 노드를 가지는 예로 ILP로 풀기에는 너무 많은 노드를 가지고 있다. 이를 자원 공유를 고려한 분할 알고리즘으로 분할을 했을 경우 자원 공유를 고려하지 않은 경우에 비해서 1/15의 비용으로 주어진 시간 제약을 만족할 수 있다. 이것은 QAM의 한번에 16개의 점에 대해서 처리하기 때문에 한 블럭이 16개의 노드로 생성되어 서로 같은 자원을 공유할 수 있는 노드들이 많기 때문이다.

자원 공유를 고려한 분할 알고리즘도 ILP에 비해서 평균 약 5%정도의 비용 증가를 가진다.

Ⅷ. 결 론

본 논문은 시스템의 설계 자동화를 위한 중요한 부분인 하드웨어/소프트웨어 분할의 새로운 알고리즘을 제시하고, 기존에 개발된 알고리즘과 비교하였다. 분할 알고리즘은 시스템 수준에서 기술한 표현으로부터 사용자가 원하는 성능을 만족하면서 최소 비용의 시스템을 구현하는 것을 도와준다. 특히 이 논문에서 제시한 분할 알고리즘은 하드웨어/소프트웨어 분할뿐만 아니라 분할 후의 적절한 구현을 선택하는 것과 자원 공유를 고려할 수 있다.

기존에 개발된 GCLP[11] 분할 휴리스틱과 비교해 보았을 때 약 15%정도의 더 나은 성능(비용감소)을 나타내는 것을 알 수 있으며, 항상 최적의 해를 찾는 ILP 알고리즘과 비교해 보았을 때 5%정도의 비용 증가만을 가져온다. 이 같은 적은 비용 증가는 BILI분할 알고리즘에서 사용하는 BIL 이종 다중 프로세서 스케줄러와 여유분, 비용 줄이기 과정에 기인한 것으로 생각된다. 즉 분할 알고리즘이 지역적인 최소값에 빠지는 것을 스케줄러가 보완하고, 여유분, 비용 줄이기 과정이 더 적은 비용을 가지는 분할을 찾도록 도와준다. BILI 분할 알고리즘은 우선 모든 노드들을 최소 비용이 되는 프로세서-구현으로 스케줄되도록 다른 프로세서에서의 실행 시간을 무한대로 만든다. 그 다음 스케줄링을 통해서 시간 제약 조건을 만족하는지 검사해서 시간 제약 조건을 만족하지 않는다면, 다른 곳에 스케줄되었을 때 가장 작은 비용 증가로 가장 스케줄되는 길이가 짧아질 수 있는 노드를 선택해서 그 노드가 그 곳으로 스케줄될 수 있도록 원래의 실행시간으로 회복시켜준다. 이러한 과정은 실제 노드가 그 프로세서로 매핑되는 것이 아니라 가능성만을 열어 두기 때문에, 이전의 분할 연구에서의 지역적인 최소값을 피해갈 수 있는 것이다. 이러한 방식은 여러 하드웨어와 소프트웨어로 이루어진 시스템에 대한 분할뿐만 아니라, 여러 가지 구현중 가장 적은 비용의 구현을 선택하는 분할 문제로도 적용이 가능했다. 그리고 쉽게 자원 공유를 고려하는 분할 알고리즘으로 확장이 가능했다.

앞으로는 데이타 병렬성을 고려한 분할 알고리즘을 개발하는데 주력할 것이다. 즉 실용적인 예들은 문제 자체에 큰 데이터 병렬성을 가지고 있기 때문에 이것을 잘 처리해 주는 분할 알고리즘의 개발이 요구되기 때문이다. 데이타 병렬성의 고려는 지금까지의 스케줄을 사용해서는 어렵고, 매크로 노드의 스케줄을 지원하는 스케줄러가 필요하다. 따라서 분할 수행 시간과 합성의 편의성을 도모하기 위한 병렬성을 고려한 분할 알고리즘은 먼저 스케줄러의 개발과 함께 진행할 것이다.

현재 이 알고리즘은 시스템 통합설계 도구인 PeaCE (Ptolemy extension as Codesign Environment)[15]

에 구현되어 있다.

참 고 문 헌

[1] S. Edwards, L. Lavagno, E. A. Lee and A. Sangiovanni-Vincentelli, "Design of Embedded Systems: Formal Models, Validation, and Synthesis." *in Proceedings of IEEE*. vol. 85. no. 3. pp 366-390. Mar. 1997.

[2] 오현옥, 하순회, "이종 프로세서를 위한 정적인 스케줄링 휴리스틱." 정보과학회논문지(A) 제25권 제12호 1339-1347. 1998년 12월 Hyunok Oh and Soonhoi Ha , "A Static Scheduling Heuristic for Heterogeneous Processors." *Second International Euro-Par Conference Proceedings, Volume* II, Lyon, France, August 1996.

[3] J. Henkel, R. Ernst, U. Holtmann and T. Benner, "Adaptation of partitioning and high-level synthesis in hardware/software cosynthesis." in *Proc. Int. Conf. on Computer-Aided Design*. Nov. 1994.

[4] S. Kumar, J. H. Aylor, B. W. Johnson and W. A. Wulf, " Exploring hardware/software abstractions and alternatives for codesign." in *Proc. Int. Workshop on Hardware-Software Codesign*. Oct. 1993.

[5] X. Hu, J. G. D'Ambrosio, B. T. Murray, and D.-L. Tang, "Codesign of architectures for powertrain modules." *IEEE Micro*, vol. 14, no. 4, pp. 48-58, Aug. 1994.

[6] R Nieman and P. Marwedel, "An Algorithm for Hardware/Software Partitioning Using Mixed Integer Linear Programming." *Design Automation for Embedded Systems, special Issue : Partitioning Methods for Embedded Systems*. Vol. 2, No. 2, 165-193, Kluwer Academic Publishers, March 1997.

[7] R.K. Gupta, C. Coelho, and G. De Micheli, "Synthesis and Simulation of Digital Systems Containing Interacting Hardware and Software Components." *29th ACM, IEEE Design Automation Conference*, pages 225-230, 1992.

[8] R.K. Gupta, C. Coelho, and G. De Micheli, "Program implementation schemes for hardware-software systems." *IEEE Computer*, pp.48-55, Jan. 1994.

[9] K. Olokuntun, R. Helaihel, J. Levitt and R. Ramirez, "A software-hardware cosynthesis approach to digital system simulation." *IEEE Micro*, vol. 14, no. 4, pp. 48-58, Aug. 1994.

[10] F. Vahid and T. D. Le, "Extending the Kernghan/Lin Heuristic for Hardware and Software Functional Partitioning." *Design Automation for Embedded Systems, special Issue : Partitioning Methods for Embedded Systems. Vol. 2, No. 2*, 237-261, Kluwer Academic Publishers, March 1997.

[11] A. Kalavade and E.A. Lee, "A Global Critically/Local Phase Driven Algorithm for the Constrained Hardware/Software Partitioning Problem." *Third International Workshop on Hardware/Software Codesign, Grenoble*, pages 42-48, 1994.

[12] A. Kalavade and E.A. Lee, "The Extended Partitioning Problem: Hardware/Software Mapping and Implementation-Bin Selection." *Proceedings of the 6th International Workshop on Rapid Systems Prototyping*, 1995.

[13] Hamada, T.Banerjee, S.Chau, P.M. Fellman, R.D, "Macropipelining based heterogeneous multiprocessor scheduling", *Proceedings of ICASSP-92* : 1992 IEEE International Conference on Acoustics, Speech and Signal Processing, San Francisco, CA, USA, 23-26 March 1992, New York, NY, USA: IEEE, 1992, p.597- 600 vol. 5.

[14] G.C Sih and E. A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures", *IEEE Trans. parallel and distributed systems*, vol 4, no.2, pp. 175-187, Feb. 1993.

[15] http://peace.snu.ac.kr/

저 자 소 개

吳 鉉 玉(正會員)

1996년 서울대학교 컴퓨터공학 학사. 1998년 서울대학교 컴퓨터공학 석사. 1998년~현재 서울대학교 전기및컴퓨터공학 박사 과정 재학. 관심분야는 하드웨어/소프트웨어 Codesign, 프로세서 스케줄링, 소프트웨어/하드웨어 분할, 시스템 설계, 멀티미디어 시스템

河 舜 會(正會員)

1985년 서울대학교 전자공학 학사. 1987년 서울대학교 전자공학 석사. 1992년 캘리포니아(Berkeley)대학교 전기및 컴퓨터 박사. 1993년~1994년 현대 전자에서 근무. 1994년~서울대학교 전기 및 컴퓨터 공학부, 현재 부교수. 관심분야는 하드웨어/소프트웨어 codesign, 신호처리용 설계 방법론, PC Clustering