멀티모드 멀티태스크 임베디드 어플리케이션을 위한 HW/SW 분할 기법

(HW/SW Partitioning Techniques for Multi-Mode Multi-Task Embedded Applications)

김 영 준 † 김 태 환 ††
(Youngjun Kim) (Taewhan Kim)

요 약 시스템의 기능을 바꾸어 가면서 여러 개의 어플리케이션을 작동시키는 임베디드 시스템을 멀티모드(multi-mode) 임베디드 시스템이라 부른다. 더 나아가서 하나의 모드가 여러 개의 태스크로 구성된 임베디드 시스템을 멀티모드 멀티태스크(multi-task) 임베디드 시스템이라 부른다. 본 논문에서는 시간제한 조건을 가지고 있는 멀티모드 멀티태스크 임베디드 어플리케이션을 대상으로 하는 HW/SW 분할 방법에 대한 연구이다. 시간제한 조건을 만족하는 스케줄과 함께 태스크의 기능모듈(functional module)을 동작시킬 효율적인 처리 자원(processing resource)을 할당/매핑하여 시스템의 비용(가격)을 최대한 낮추는 것이 목적이다. 이 문제를 잘 풀기 위해 중요한 것은 모듈사이의 병렬성을 최대한 이용하여 실행시키는 것이다. 그러나 이전의 HW/SW 분할 방법은 모듈의 병렬 실행 가능성을 최대한 이용하지 않았는데, 병렬성 이용을 위한 탐색 계산이 복잡할 뿐 아니라 스케줄 가능성(schedulability) 검사를 단순하게 하려고 하였기 때문이다. 기존 방법의 한계를 극복하기 위해서 우리는 다음의 세 개의 세부문제를 동시에 고려하는 HW/SW 분할 기법을 제안 한다: (1) 처리 자원의 할당 (2) 태스크 모듈에 대한 처리 자원 매핑 (3) 모듈 실행 스케줄의 결정. 특별히 모듈의 병렬 실행과 실행가능성을 간결하게 측정하는데 바탕을 둔 단순모드(single-mode) 멀티태스크 어플리케이션에 대한 반복 개선 방식을 갖는 분할 기법을 만들었다. 다시 이 기법을 확장하여 멀티모드 멀티태스크 어플리케이션의 분할 기법을 만들었다. 실제 사용되는 어플리케이션을 대상으로 한 실험에서 제안된 우리의 기법이 기존의 방법에 비해서 단순모드와 멀티모드 멀티태스크 어플리케이션에 대해서 각각 17.0%와 19.0%의 가격을 낮추는 것이 확인되었다.

키워드 : 임베디드 시스템, 통합설계, 자원공유, 병렬성, 스케줄링

Abstract An embedded system is called a multi-mode embedded system if it performs multiple applications by dynamically reconfiguring the system functionality. Further, the embedded system is called a multi-mode multi-task embedded system if it additionally supports multiple tasks to be executed in a mode. In this paper, we address a HW/SW partitioning problem, that is, HW/SW partitioning of multi-mode multi-task embedded applications with timing constraints of tasks. The objective of the optimization problem is to find a minimal total system cost of allocation/mapping of processing resources to functional modules in tasks together with a schedule that satisfies the timing constraints. The key success of solving the problem is closely related to the degree of the amount of utilization of the potential parallelism among the executions of modules. However, due to an inherently excessively large search space of the parallelism, and to make the task of schedulabilty analysis easy, the prior HW/SW partitioning methods have not been able to fully exploit the potential parallel execution of modules. To overcome the limitation, we propose a set of comprehensive HW/SW partitioning techniques which solve the three subproblems of the partitioning problem simultaneously: (1) allocation of processing resources, (2) mapping the processing resources to the modules in tasks, and (3) determining an execution schedule of modules. Specifically, based on a precise measurement on the parallel execution and schedulability of modules, we develop a stepwise refinement partitioning

· 이 논문은 2005년도 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2005-041-D00589)
† 비 회 원 : 삼성전자 System LSI 연구원
yjkim@ssl.snu.ac.kr
†† 종신회원 : 서울대학교 전기공학부 교수
tkim@ssl.snu.ac.kr
논문접수 : 2006년 7월 5일
심사완료 : 2007년 5월 23일

technique for single-mode multi-task applications. The proposed techniques is then extended to solve the HW/SW partitioning problem of multi-mode multi-task applications. From experiments with a set of real-life applications, it is shown that the proposed techniques are able to reduce the implementation cost by 19.0% and 17.0% for single- and multi-mode multi-task applications over that by the conventional method, respectively.

Key words : Embedded system, codesign, resource sharing, parallelism, scheduling

1. 서 론

여러 개의 태스크로 구성된 여러 개의 어플리케이션을 시스템의 기능(functionality)을 바꾸어 가면서 동작시킬 수 있는 임베디드 시스템을 멀티모드 멀티태스크 임베디드 시스템이라고 한다. 모드를 변경 시키는 능력은(즉, 동적 재구성능력은) 다양하고 빠르게 변화하는 멀티미디어 알고리즘 다루기에 알맞다. 예를 들면 멀티모드 모바일 전자기기는 모드 변경을 통해서 PCS 전화기, MP3 플레이어 그리고 VOD 터미널로 사용될 수 있다. 이러한 임베디드 시스템을 설계할 때는 효율적인 자원 공유를 통한 가격 절감이 중요하다.

대부분의 임베디드 시스템은 성능 제약 조건을 가지고 있다. 어플리케이션은(즉, 모드) 수행되어야 할 태스크 집합으로 구성되어 있으며 태스크는 정해진 시간 내에 실행되어야 한다. 하나의 처리 자원을 가지고 있는 임베디드 시스템에서는 기존의 스케줄 가능성 테스트 기법(예, [1,2])을 이용하여 각 태스크에 대한 스케줄 가능성을 정확히 측정할 수 있다. 하지만 다수의 처리 자원을 가지고 있는 임베디드 시스템을 설계할 때는 정확하게 스케줄 가능성을 판단하는 것이 매우 복잡하기 때문에 대부분의 HW/SW 분할 방법에서는 기존에 잘 알려진 스케줄 가능성 검사 방법을 이용하며 이 때문에 느슨하게 스케줄 되게 된다. 단순모드 단순태스크 어플리케이션을 위해서는 성능과 동작 시간 측면에서 잘 만들어진 HW/SW 분할 기법들이(예, [3,4]) 알려져 있다. 그러나 이러한 방법들이 멀티태스크 어플리케이션에는 그대로 잘 적용되지는 않고 있다.

멀티모드 멀티태스크 어플리케이션의 HW/SW 분할 문제는 최소 비용의 자원으로 구성된 시스템 구조와 시간 제약 조건을 만족하는 스케줄을 찾는 것이다. 어플리케이션에 속한 각 태스크는 모듈이라고 불리는 기능 블록으로 구성된다. 예를 들어 DCT(Discrete Cosine Transformation) 블록, MC(Motion Compensation) 블록 등으로 태스크가 구성된다. 태스크는 같은 기능을 담당하는 모듈을 중복해서 가질 수 있으며 서로 다른 태스크도 같은 기능을 하는 모듈을 가질 수 있다. (우리는 설계자가 처리 자원 라이브러리를 가지고 있다고 가정한다. 처리 자원 라이브러리에는 하드웨어 IP 블록 또는 프로세서와 같은 처리 자원이 해당 기능 모듈을 수행시키는데 필요한 시간 정보가 명시되어 있다.)

기존의 많은 논문에서는 단순모드 멀티태스크 어플리케이션을 위한 HW/SW 분할 기법(예, [5,6,8])을 연구하여 왔지만, 몇 개의 논문에서만 멀티모드 멀티태스크 어플리케이션을 고려하였다. 논문 [7]에서는 태스크 수준에서 멀티모드 어플리케이션의 스케줄 가능성 검사 방법을 제안하였으며, 논문 [9]에서는 서로 다른 태스크에 있는 모듈 사이의 자원 공유를 고려하는 모듈 수준 분할 기법을 제안하였다. 이 분할 기법은 두 단계(phase)를 갖고 이를 반복하여 수행하고 있다. 첫 번째 단계에서는 모듈 사이의 자원 공유를 고려한 처리 자원의 할당/매핑이 이루어진다. 두 번째 단계에서는 다른 태스크의 모듈은 병렬적으로 계산 할 수 없다는 가정아래 HMS(heterogeneous multiple scheduling) 방법을 이용해서 스케줄 가능성 테스트가 이루어진다. 결과적으로 실행 가능한 스케줄이 존재하는 처리 자원 할당/매핑의 경우에도 HMS 방법을 이용해서 스케줄 가능성 검사를 하면 스케줄을 찾지 못할 수 있다. 이는 두 번째 단계의 스케줄 가능성 검사가 첫 단계에서 이루어진 처리 자원 할당/매핑 결과를 최대한 활용하지 못하기 때문이다. 논문 [14]는 논문 [9]의 방법을 MPSoC(Multiprocessor SOC) 설계의 실용적인 측면에서 자세히 다루고 있다. 논문 [10]에서는 에너지 소비를 최소화시키는 것을 주된 목표로 하여 멀티모드 어플리케이션 분할 기법을 연구했다. 이 연구에서는 시간과 공간(area) 제한 조건을 고려하면서 자주 사용되는 모드의 태스크는 에너지 소비가 적은 처리 자원에 할당을 한다. 반면에, 논문 [11]에서는 시스템 비용의 최소화를 목표로 멀티모드 어플리케이션에 대한 분할 기법을 연구하였다. 다른 모드에서 공통으로 자주 사용되는 태스크 또는 모듈은 하드웨어로 실행될 가능성을 높이는 방법으로 분할을 한다. 논문 [6]에서는 여러 개의 목적을 가진 분할 방법을 연구하였다. 이 논문에서는 공간과 전력 효율성 측면에서의 높은 구조에 대한 파레토(pareto) 최적화 집합을 찾는다. 연구 [11]과 연구 [6]은 서로 다른 모드에 속한 모듈/태스크 또는 같은 모드 안의 모듈/태스크 사이의 자원 공유를 충분히 활용하지 못하고 있다. 왜냐하면 연구 [10]의 각 모드는 하나의 태스크만을 가질 수 있으며 연구 [6]

은 단순모드 어플리케이션에만 적용할 수 있기 때문이다.

최근에는 기존과 다른 HW/SW 분할 연구가 진행 중에 있다. 논문 [13]에서는 모의 담금질(simulated annealing) 방법과 같은 상태 이동(state movement)을 바탕으로 하는 방법을 사용하여 분할 결과를 평가하는 기법을 제안하였다. 이 기법의 한 가지 결정적인 단점은 순차적인 프로그램만을 HW/SW 분할 대상으로 한다는 것이다. 논문 [12]에서는 제한된 HW 자원 제약 조건 안에서 시스템의 가격이 아닌 실행 시간을 최소화하는 기법을 제안하였는데, 이는 동적 재구성을 할 수 있는 재구성형 구조에 적용될 수 있도록 목표를 두었다.

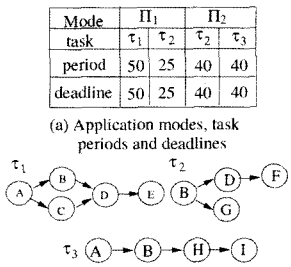
기존의 연구와 비교하여 멀티모드 멀티태스크 어플리케이션에 대해서, 다음과 같은 두 가지 차이점 때문에 우리가 제안하는 HW/SW 분할 방법이 더 효율적인 면을 가진다.

(1) 처리 자원의 할당, 처리 자원의 모듈 매핑 및 수행 스케줄의 세 가지의 작업을 하나의 프레임워크 안에서 해결하고자 하며, 세 문제를 동시에 풀어서 전체적으로 최적화된 해를 찾는다.

(2) 태스크 수준의 자원 공유 대신 더 세밀한 단계의 모듈 수준의 자원 공유와 수행 가능한 병렬성을 찾고자 한다. 이것은 모듈이 스케줄의 최소 단위가 됨을 말하며 태스크에 속한 모듈의 자원 공유 상태에 따라서 태스크가 부분적으로 또는 완전히 병렬적으로, 또는 완전히 직렬적으로 실행된다. (본 연구에서는 태스크 단위 보다 더 작은, 모듈 수준에서의 병렬적 수행을 찾고자 함으로써 [9,14]에서 태스크 단위의 병렬적 수행의 단점을 극복하고자 하는 것이다.)

2. 연구 동기

처리 자원의 할당/매핑과 모듈 스케줄 결과가 전체 시스템 비용과의 관계를 보여주는 몇 가지 예를 먼저 들고자 한다. 두 개의 어플리케이션 모드 Π_1과 Π_2를 가지고 있는 임베디드 시스템 설계를 생각해 보자. 그림 1(a)에 있는 것처럼 Π_1은 태스크 τ_1과 τ_2를 동작시키며 Π_2는 τ_2와 τ_3를 동작시킨다. (예를 들어 비디오폰 (Π_1 모드)과 MP3 플레이어(Π_2 모드) 두 개의 동작 모드를 지원하는 시스템을 생각할 수 있다.) τ_1,τ_2과 τ_3의 태스크 그래프는 그림 1(b)에 있으며 태스크의 주기(period)와 수행 데드라인(deadline)은 그림 1(a)에 명시되어 있다. (주의: 본 연구 내용의 이해를 돕기 위해 사용한 예이기 때문에 특정 모듈에 입출력 포트 수의 일치 등은 고려하지 않았다.) 예를 살펴보면 특정 모듈은 여러 개의 태스크에서 공통으로 사용되고 있음을 알 수 있다. (예를 들어, H.264 부호기와 H.264 복호기는 모두 T^{-1}(역변환 모듈)과 Q^{-1}(역양자화 모듈)을 실행시킨다. T^{-1}과 Q^{-1}의 역할에 대한 자세한 내용은 참고문헌[15]의 그림 6.1과 6.2에 자세히 나와 있다.)) 여기에 그림 1(c)에 요약된 것처럼 선택 가능한 처리 자원 라이브러리가 있다. 라이브러리에는 SW[1]로 표시되어 있는 processor와 HW 열의 A_{hw1}, B_{hw2}, ..., I_{hw1} 하드웨어 구현 정보를 가지고 있다. 각 칸의 숫자(괄호 밖)는 처리 자원에서 해당 모듈 수행 처리에 대한 자원의 수행 시간을 나타낸다. 괄호 안의 숫자는 해당 처리 자원을 구현하였을 때의 비용(또는 면적)을 나타낸다. 예를 들어 모듈 A는 SW (processor)에서 10의 시간동안 실행되거나, 하드웨어 A_{hw1}에서 4의 시간동안에 실행됨을 나타낸다.



(a) Application modes, task periods and deadlines

(b) Task graphs

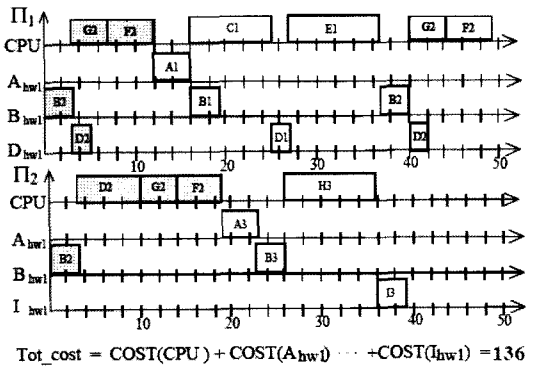
	SW time	HW time (cost)
A	10	A_{hw1} :4 (7)
B	7	B_{hw1} :3 (12)
C	9	C_{hw1} :5 (8)
D	7	D_{hw1} :2 (10)
E	10	E_{hw1} :3 (15) E_{hw2} :5 (8)
F	5	F_{hw1} :3 (8)
G	4	G_{hw1} :4 (8)
H	10	H_{hw1} :5 (10)
I	7	I_{hw1} :3 (7)

(c) Module-resource profile table

그림 1 멀티모드 멀티태스크 어플리케이션의 모듈 수준 스케줄과 병렬적 실행의 중요성을 설명하는 예; (a) 응용 모드, 태스크 주기와 데드라인; (b) 태스크 그래프; (c) 모듈-자원 프로파일 표

예1. (태스크 수준 스케줄링과 모듈 수준 스케줄링의 비교): 그림 1(a), (b), (c)에 대해 그림 1(d)은 기존의 연구인 논문 [8,9]에서 제안한 기법에 따른 처리 자원의 할당과 모듈에 대한 자원 매핑 결과 및 태스크 수준 스케줄 결과를 보여주고 있다. 여기에서 할당된 자원의 가격은 136이고 모든 태스크의 실행이 데드라인을 만족하는 것을 볼 수 있다. 논문 [9]에서는, 태스크 수준의 스케줄링을 할 때 기존의 멀티프로그래밍 스케줄링 기법([1])을 사용하여 스케줄 가능성 테스트를 하였다. 이 기법을 이용하여 스케줄을 할 경우 단점은 서로 다른 태스크에서의 모듈들에 대한 병렬적 수행을 고려할 수 없다는 것이다. 반면에 그림 1(e)은 태스크 수준이 아닌, 모듈 수준의 스케줄링을 이용하여 스케줄하고 자원 할당과 모듈 매핑 된 결과를 보여주고 있다. 결과적으로 태스크 수준 스케줄링에서는 가능하지 않는 태스크의 중첩된 실행이 가능하게 된다. 예를 들어 그림 1(e)에서 점선의 원으로 표시된 부분에서 보듯이, 모드

1) 설명을 간단하고 쉽게 하기 위해서 임시적으로 라이브러리에는 오직 하나의 소프트웨어 프로세서가 있다고 가정을 하고 소프트웨어 프로세서는 시스템에 항상 할당되어 있다고 하자.



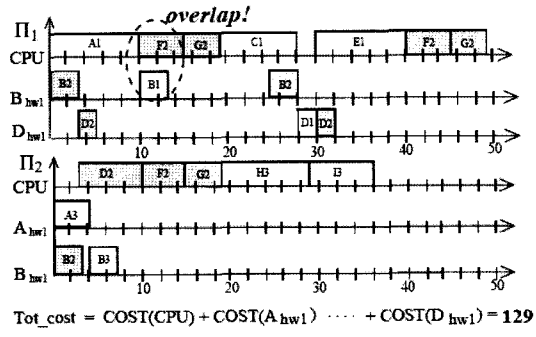
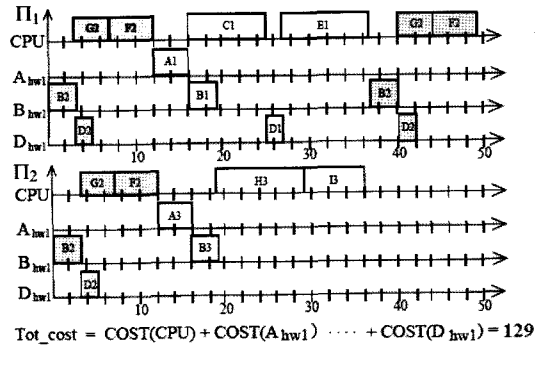


그림 1 앞의 그림과 연속됨; (d) [9]의 태스크-수준 스케줄링에 의한 결과; (e) 본 제안한 모듈-수준 스케줄링에 의한 결과

Π_1에서 태스크 τ_2의 모듈 F와 태스크 τ_1의 모듈 B가 중첩되어 실행되는 것을 볼 수 있다. 결과적으로 전체 시스템 가격이 136에서 129로 줄어들게 되는데, 이는 모듈 수준에서의 스케줄링이 시스템 전체 비용을 낮추는데 중요한 역할을 함을 입증 한다.

예2. (모드 간의 자원 공유): 멀티모드 어플리케이션의 경우에는 시스템 비용을 더 낮추기 위해서는 같은 모드 내에서의 모듈 뿐 만 아니라 다른 모드 사이에서의 모듈들이 자원을 공유하도록 하는 것이 필요하다. 그림 1(f)은 모드 사이의 자원 공유를 고려하여 그림 1(d)의 상태를 최적화한 결과를 보여주는데, 자원 I_{hw1}가 이미 할당된 자원 D_{hw1}로 대체됨에 따라 시스템 전체 비용이 136에서 129로 줄어든 것을 볼 수 있다. 또, 그림 1(g)은 그림 1(e)의 상태에 모드 사이의 자원 공유를 고려하여 129에서 122로 시스템 가격을 낮춘 모습을 보여준다. 이러한 가격 감소는 멀티 모드 자원 공유가 시스템 가격을 낮추는데 꼭 필요함을 나타내고 멀티 모드 자원 공유와 모듈 수준 스케줄링이 동시에 적용되면 더 시스템 비용을 줄일 수 있음을 보여준다 하겠다.



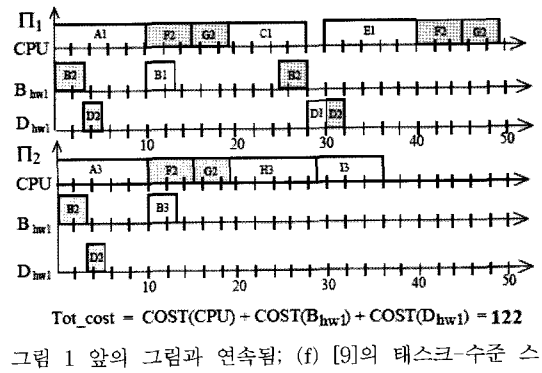


그림 1 앞의 그림과 연속됨; (f) [9]의 태스크-수준 스케줄링에 본 제안한 다중모드의 자원공유 방법을 결합했을 때의 생성된 결과; (g) 본 제안한 모듈-수준 스케줄링과 다중모드의 자원공유 방법을 결합했을 때의 생성된 결과

3. 문제 정의

멀티모드 어플리케이션에 대한 주기와 데드라인 정보와 함께 태스크 그래프가 주어지고 $r_1, r_2, \cdots, r_L$ 자원의 라이브러리 정보가 주어진다. $\tau_1^k, \tau_2^k, \cdots \tau_{N_k}^k$ 을 모드 Π_k에 속한 태스크들이라 하고 각 태스크는 데드라인 제한 조건 $t_1^k, t_2^k, \cdots t_{N_k}^k$를 갖는다. 그리고 $m_{k,1}^j, m_{k,2}^j, \cdots$ 를 태스크 τ_j^k 에 속한 모듈들이라고 하자. (τ_j^k 란 모드 Π_k에 속한 태스크들 중에 j 번째 태스크를 말한다.) 여기서 주목할 점은 태스크들에 있는 모듈들 간에는 같은 기능을 가진 두 개의 모듈이 같은 태스크 안에 있을 수 있고 또는 서로 다른 태스크에 있을 수 있으며 서로 다른 모드의 서로 다른 태스크 안에도 있을 수도 있다.

$S(\tau_j^k)$는 τ_j^k에 속한 모듈의 스케줄을 나타내고, $B(S(\tau_j^k))$는 τ_j^k에 속한 모듈이 $S(\tau_j^k)$대로 스케줄 되었을 때의 자원 배정을 표시한다고 하자. 여기에 더해서 $R(B(\cdot))$을 배정 $B(\cdot)$에 사용된 자원의 집합을 나타내는 표기법으로 사용한다.

문제-1. (분할 문제): 어플리케이션 모드 Π_k의 태스크 $\tau_j^{k(=1,2,\cdots,K)}$ 각각에 대해서 태스크의 데드라인 제한 조건을 만족 시키면서 아래의 식을 최소화 시키는 스케줄 S와 바인딩 B를 찾고자 한다.

$$COST_{total}^{m-\mathrm{mode}}(\Pi_1,\Pi_2,\cdots,\Pi_k)= \quad (1)$$
$$COST(\bigcup_{k=1}^{K}\bigcup_{j=1}^{j_k} R(B(S(\tau_j^k))))$$

위 식에서 $COST(r_i)$는 자원 라이브러리 L에 있는 자원 r_i의 비용을 의미하며 $COST(r_{i_1},r_{i_2},\cdots)=COST(r_{i_1})+COST(r_{i_2})\cdots$ 로 나타낼 수 있다.

4. 제안 방법

우선 문제-1의 특별한 경우에 대해서 생각해보기로 한다.

문제-2. (단순모드 분할 문제): τ_j^k, $j=1,2,\cdots,j_1$인 각 태스크에 대해서 아래 식을 최소화 시키는 스케줄 S와 바인딩 $B(S)$를 찾는다.

$$COST_{total}^{m-\mathrm{mode}}(\Pi_1)=COST_{total}^{s-\mathrm{mode}} \quad (2)$$
$$=COST(\bigcup_{j=1}^{j_1} R(B(S(\tau_j^1))))$$

서로 다른 태스크에 속한 모듈 사이에 존재하는 병렬성을 최대한 이용해서 태스크가 중첩되어 실행시키는 것이 문제-2를 푸는데 가장 중요한 사항이다. 각 어플리케이션에 대해서 식 (2)의 $COST_{total}^{s-\mathrm{mode}}$를 최소화하는 스케줄링과 자원 배정을 반복적인 방법을 통해서 구한다. 그 후에 식 (1)의 $COST_{total}^{m-\mathrm{mode}}$를 낮추기 위해서 모드 사이의 자원을 상향식(bottom-up) 방식으로 공유시킨다.

A. 단순모드 HW/SW 분할

태스크에 속한 각 모듈을 라이브러리 L에 있는 자원 중에서 가장 빠른(즉, 가장 비싼) 하드웨어에 바인딩(배정)시키고 하나의 하드웨어는 하나의 모듈만을 동작시키도록 할당한다. 우리가 제안하는 HW/SW 분할 알고리즘에서는 이 상태를 초기 분할 상태로 한다. 결과적으로 태스크의 수행시간은 가장 짧지만 높은 비용(즉 $COST_{total}^{s-\mathrm{mode}}$의 값)을 가진다. 그 후에 태스크의 시간제한 조건을 만족하면서 $COST_{total}^{s-\mathrm{mode}}$의 값이 최대한 작아지게 되도록 모듈을 다시 바인딩 시키는 과정을 반복적으로 수행한다. 우리가 제안하는 HW/SW 분할 방법의 프레임워크는 K-L 그래프 분할 방법[13]의 변형이며 $COST_{total}^{s-\mathrm{mode}}$의 값이 줄어들지 않을 때까지 반복적인 개선 작업을 수행한다. 예를 들어 그림 1(b)에 있는 태스크 τ_1과 τ_2의 각 모듈마다 하드웨어 자원이 하나씩 바인딩 된다. 그림 2(a)는 이런 초기 바인딩 상태를 보여준다. 모드 Π_1의 하이퍼주기[2](hyper period)는 50이며 τ_1과 τ_2는 이 시간동안 스케줄이 된다. 점선 화살표는 자원과 모듈의 바인딩을 나타내며 각 모듈의 괄호 속의 숫자는 바인딩 된 자원에서의 실행 시간을 나타낸다. 이 때의 시스템 비용은 228이며 실행 스케줄은 모든 시간 제한 조건을 만족하고 있다. 예를 들어 τ_1에서 가장 긴 실행 경로(그림 2(a)에 굵은 선으로 표시)는 $A_1 \rightarrow C_1 \rightarrow D_1 \rightarrow E_1$이며, 이 경로는 0에서 시작하여 14시간까지 실행된다.

PARTITION-s라고 부르는 이 분할 방법은 두 개의 순환-루프로 구성되어 있는데 하나의 루프는 다른 하나에 중첩되어 있다. 안쪽 루프에서는 초기의 바인딩과 스케줄을 이용하여 PARTITION-s가 재바인딩(rebinding)과 재스케쥴링(rescheduling)을 반복하면서 HW/SW 분할 문제를 푼다. 재바인딩 과정에서는 PARTITION-s가 모듈을 다른 자원에 할당시켜보고 $COST_{total}^{s-\mathrm{mode}}$의 값이 얼마나 줄어들었는지 검사한다. 재스케쥴링 과정에서는 PARTITION-s가 변화된 바인딩 결과를 반영하여 모듈의 실행이 재스케쥴링 된다. PARTITION-s는 모듈을 자원에 재바인딩한 결과 중에서 시간제한 조건을 만족시키면서 $COST_{total}^{s-\mathrm{mode}}$의 값을 가장 많이 감소시키는 재스케쥴링을 갖는 재바인딩을 선택한다. 선택된 재바인딩 대로 모듈이 자원에 바인딩 되며 모듈은 고정(lock) 된다. 이런 과정은 아직 고정이 안 된 모듈을 대상으로 반복이 되며 모든 모듈이 고정되거나 시간제한 조건을 만족하는 재스케줄 결과가 없으면 안쪽 루프의 반복을 멈춘다. 바깥쪽 루프에서는 모든 모듈의 고정을 풀어주며 안쪽 루프의 마지막 반복에서 생성된 재바인딩 중에서 $COST_{total}^{s-\mathrm{mode}}$의 값이 가장 작은 재바인딩 상태를 새로운 초기값으로 설정한다. 이렇게 새롭게 초기화된 상태에서 PARTITION-s는 안쪽 루프를 다시 반복한다. 바깥쪽 루프는 더 이상의 시스템 비용 감소가 없는 경우에 반복을 마친다. 이 방법에서 우리의 핵심적인 고려사항은 다음과 같다.

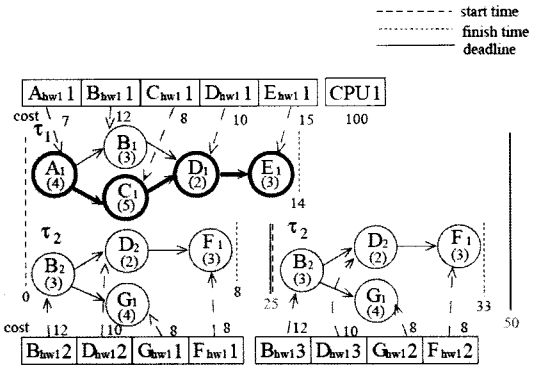
고려사항 1: 시간제한 조건을 만족하면서 서로 다른 태스크에 속한 모듈이 병렬적으로 실행 되도록 하는 스케줄을 생성한다.

고려사항 2: 재스케쥴과 재바인딩을 효율적으로 평가할 수 있는 방법을 고려한다.

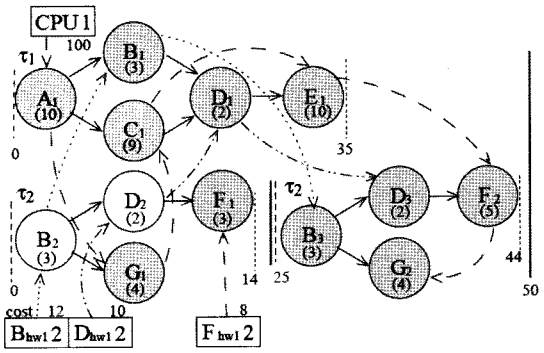
이제부터 그림 2의 예를 이용하여 두 가지 고려사항을 중심으로 PARTITION-s 과정의 세부 사항을 설명한다.

그림 2(b)의 표는 각 재바인딩 시도에 대한 시스템

2) 모드에 속한 태스크 주기의 최소공배수.



(a) Initial resource mapping (cost=228, slack=70)



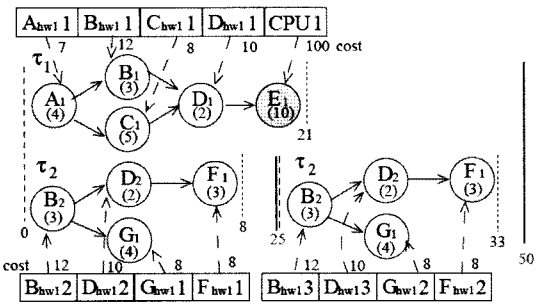
(d) Resource mapping result of the 11-th iteration (cost=130, slack=32)

module	assigned resource	reassigned resource	resource_link	schedule feasiblity	$\Delta slack$	ΔC^{s-mode}
A_1	$A_{hw1}1$	CPU	1	O	−6	7
B_1	$B_{hw1}1$	CPU	1	O	−2	12
		$B_{hw1}2$	1	O	−7	12
			2	O	0	12
		$B_{hw1}3$	1	O	0	12
			2	O	−22	12
E_1	$E_{hw1}1$	CPU	1	O	−4	**15**
⋮	⋮	⋮	⋮	⋮	⋮	⋮
F_2	$F_{hw1}2$	CPU	1	O	−2	8
		$F_{hw1}1$	1	—	—	—
			2	O	0	8

(b) $\Delta slack$, ΔC^{s-mode} computations during the first iteration

module	assigned resource	reassigned resource	resource_link	schedule feasiblity	$\Delta slack$	ΔC^{s-mode}
B_2	$B_{hw1}2$	CPU	1	X	—	—
			⋮	⋮	⋮	⋮
			5	X	—	—
D_1	$D_{hw1}2$	CPU	1	X	—	—
			⋮	⋮	⋮	⋮
			5	X	—	—

(e) $\Delta slack$, ΔC^{s-mode} computations during the 12-th iteration



(c) Resource mapping result of the first iteration (cost=213, slack=63)

module	assigned resource	reassigned resource	resource_link	schedule feasiblity	$\Delta slack$	ΔC^{s-mode}
A_1	CPU	$A_{hw1}1$	1	O	18	−7
B_1	$B_{hw1}2$	CPU	1	X	—	—
			⋮	⋮	⋮	⋮
			5	X	—	—
C_1	CPU	$C_{hw1}1$	1	O	13	−8
⋮	⋮	⋮	⋮	⋮	⋮	⋮
F_1	$F_{hw1}2$	CPU	1	X	—	—
			2	O	−15	**8**
			3	O	−15	8
⋮	⋮	⋮	⋮	⋮	⋮	⋮

(f) $\Delta slack$, ΔC^{s-mode} computations during 13-th iteration

그림 2 제안한 분할 알고리즘 PARTITION-s의 흐름을 설명하기 위한 예; (a) 초기 자원 배정; (b) 첫 반복 수행에서의 비용 계산; (c) 첫 반복 후의 자원 배정 결과; (d) 11째 반복 수행 후의 자원 배정 결과; (e) 12번째 수행에서의 비용 계산; (f) 13번째 수행에서의 비용 계산

비용과 스케줄 가능성 여부(즉, 시간제한 조건의 만족 여부)를 정리한 내용을 담고 있다. 예를 들어 $B_{hw1}1$에 바인딩(assigned resource 열)된 모듈 B_1은 프로세서(reassigned resource 열의 CPU라고 표시), $B_{hw1}2$, $B_{hw1}3$에 재바인딩 된다. resource_link 열의 각 칸에 있는 숫자(i 라고 하면)는 할당된 자원이 i-번째로 해당 모듈을 실행시킨다는 의미이다. 그리고 schedule feasibility 열은 해당하는 재바인딩의 결과가 시간제한 조건을 만족하는지 아닌지를 표시한다. 재바인딩의 평가에 대한 정보는 △slack열과 $\Delta COST^{s-mode}$열이 가지고 있

다. △slack열에서는 재바인딩의 결과로 태스크의 실행 시간이 변화한 양을 나타내고 $\Delta COST^{s-\mathrm{mode}}$열은 재바인딩의 결과로 모든 자원 가격을 합한 결과가 변화한 양을 나타낸다.

$$\Delta COST^{s-\mathrm{mode}} = COST^{s-\mathrm{mode}}_{tot\_old} - COST^{s-\mathrm{mode}}_{tot\_\neq w} \quad (3)$$

위의 식에서 $COST^{s-\mathrm{mode}}_{tot_\neq w}$과 $COST^{s-\mathrm{mode}}_{tot_old}$는 각각 재바인딩 전과 후의 식 (2)의 $COST^{s-\mathrm{mode}}_{total}$을 나타낸다.

예를 들어 τ_1의 모듈 B_1은 B_{hw1}1에서 B_{hw1}2로 할당이 바뀌었다. B_{hw1}2는 모듈 B(그림 2(a)에는 B2로 표시)가 이미 사용하고 있으며 자원 B_{hw1}2에 의한 모듈 실행 순서는 $B_1 \rightarrow B_2$가 된다. (resource link 열에 있는 숫자가 모듈의 실행 스케줄을 결정한다. 예를 들어 자원 r_i에 모듈 m_1, m_2, m_3가 바인딩 되어있는 상태이고 r_i가 모듈을 $m_1 \rightarrow m_2 \rightarrow m_3$ 스케줄 순서를 가진다고 가정하자. 이제, 모듈 m_4을 r_i에 할당 시키면 4개의 resource link 값: 1, 2, 3, 4를 갖게 된다. resource link가 1인 경우에는 $m_4 \rightarrow m_1 \rightarrow m_2 \rightarrow m_3$ 스케줄이 생성된다. resource link가 2인 경우에는 $m_1 \rightarrow m_4 \rightarrow m_2 \rightarrow m_3$ 스케줄이 생성된다. resource link가 3인 경우에는 $m_1 \rightarrow m_2 \rightarrow m_4 \rightarrow m_3$ 스케줄이 생성된다. resource link가 4인 경우에는 $m_1 \rightarrow m_2 \rightarrow m_3 \rightarrow m_4$ 스케줄이 생성된다.) 결과적으로 B_2의 시작 시간은 0에서 7로 바뀌게 되며 종료 시간은 7이 증가하게 된다. (즉, 표에서 △slack = -7이 된다.) 반면에 모듈 B_1을 실행시키는 자원이 B_{hw1}1에서 B_{hw1}2로 변화하여 B_{hw1}1은 어떤 모듈도 실행하지 않는다. 시스템 비용은 B_{hw1}1의 가격만큼 낮아지게 되고 $\triangle COST^{s-\mathrm{mode}}$ = 12가 된다. 그림 2(b)에 있는 표는 모든 가능한 모듈의 재바인딩을 나타낸다. 그 중에서 시간제한 조건을 만족하면서 시스템 비용을 가장 많이 낮추는 재바인딩을 선택하게 되며 표에서는 시스템 가격을 15를 낮추는 재바인딩이 선택된다. 그러므로 그림 2(c)가 보여주는 것처럼 모듈 E_1은 E_{hw1}에서 CPU로 할당이 변화하고 스케줄이 된다. 이제 E_1은 고정되며 그림 2(c)에서는 회색으로 표시된다. 고정이 되지 않은 모듈에 대해서 다시 재바인딩 과정을 반복한다. 그림 2(d)는 우리의 재바인딩 과정이 11번 반복된 후의 상태를 보여준다. 이 분할 상태에서 고정이 되지 않은 모듈은 B_2와 D_2이다. 그림 2(e)에 있는 표는 고정이 되지 않은 모듈을 재바인딩한 결과를 보여준다. 표에서 어떤 재바인딩도 시간제한 조건을 만족시키는 스케줄을 만들지 못하는 것을 볼 수 있다. 결과적으로, 여기에서 반복이 멈추게 된다. 지금까지 생성된 11개의 상태 중에서 시스템 비용이 가장 낮은 상태를 선택 한다. 그림 2(d)에 있는 분할 상태가 시스템 비용이 가장 낮기 때문에 선택된다. 선택된 상태의 모든 고정된 모듈을 풀어주고 더 이상의 시스템 비용의 감소가 없을 때까지 재바인딩 과정을 반복한다. 마지막으로 그림 2(f)는 그림 2(d) 상태에 대한 재바인딩 비용 계산 결과를 보여준다. 표로부터 몇몇 재바인딩은 시스템 비용을 높이는 것을 볼 수 있다. 어떤 재바인딩도 시스템 비용을 낮추는 것이 없는 경우에는 시스템 비용을 가장 적게 높이는 재바인딩을 선택하게 된다. 이러한 절차는 K-L 분할 방법 알고리즘의 복잡도를 그대로 따르며 단지 추가 되는 시간 복잡도는, 재바인딩이 시도 될 때 재스케줄링을 행하는 시간이 O(n) 만큼 추가로 들게 된다 (n은 module의 개수).

B. 멀티모드 HW/SW 분할

멀티모드 시스템을 분할하기 위해서 단순모드 분할 방법인 PARTITION-s를 활용한다. 우선 두개의 어플리케이션 모드에 대한 HW/SW 분할 문제를 어떻게 풀었는지 설명하고 일반적인 경우의 풀이로 확장하여 설명하겠다.

• 2개의 어플리케이션 모드 문제 Π_1, Π_2: 각 모드에 대해서 PARTITION-s를 적용하여 분할 결과를 생성한다. 즉, 각 모드에 대한 바인딩 B, 스케줄 S, 할당된 자원 집합 R을 찾는다. (B_i, S_i, R_i)가 모드 $\Pi_i, i=1,2$의 분할 결과라고 하자. (B_1, S_1, R_1)과 (B_2, S_2, R_2)의 자원 가격의 합보다 자원의 가격이 작아지도록 모드 Π_1과 Π_2을 위한 새로운 분할 결과 $(B_{1,2}, S_{1,2}, R_{1,2})$을 찾아야 한다. 그러므로 $(COST^{m-\mathrm{mode}}_{total}(\Pi_1) + COST^{m-\mathrm{mode}}_{total}(\Pi_1)) - COST^{m-\mathrm{mode}}_{total}(\Pi_1, \Pi_2)$의 값이 가장 커지도록 하는 $(B_{1,2}, S_{1,2}, R_{1,2})$을 찾는다. 모드$\Pi_1$과 Π_2을 위한 분할 결과는 국지적으로 최적화된 해이므로 우리가 제안하는 멀티모드 분할 방법 PARTITION-m은 자원 라이브러리 L에서 새로운 자원을 할당하기 보다는 R_1 과 R_2 에 존재하는 자원을 최대한 활용하여 전체적으로 최적화된 해를 찾는다.

Π_1과 Π_2에 대한 분할을 찾기 위해 PARTITION-m은 세 가지 과정을 거친다.

과정 1 • PARTITION-s를 각각 Π_1과 Π_2에 적용시켜서 (B_1, S_1, R_1)과 (B_2, S_2, R_2)을 생성한다. $R_1 - R_2$ 과 $R_2 - R_1$ 은 각각 Π_1과 Π_2에 있는 모듈만 사용하는 자원의 집합을 나타낸다.

과정 2 • $R_{1,2} = R_1 \bigcap R_2$ 을 계산한다.

과정 3 • 다음 두 가지 조건을 만족하는 (i) $COST(R_{unused})$가 가장 큰 값을 갖고 (ii) PARTITION-s를 자원 라이브러리 $L = R_{1,2} \cup ((R_1 - R_2 \cap R_2 - R_1) - R_{unused})$을 가

지고 각각 Π_1과 Π_2에 적용시켜서 시간제한 조건을 만족하는 스케줄과 바인딩을 만들 수 있도록 부분집합 $R_{unused} \sqsubseteq R_1 - R_2 \cap R_2 - R_1$ 을 찾는다.

그림 3은 R_{unused}을 결정하는 과정을 보여준다. while-loop의 마지막 라인 바로 전에서 현재 반복에서 얻어진 시스템비용 감소량을 의미하는 $COST(R_1 \cup R_2) - COST(R'_1 \cup R'_2) = COST(r_i)$가 저장된다. 한 번의 반복에서 R_{unused}는 하나의 자원만을 추가하기 때문에 반복은 $O(|R_1 - R_2 \cap R_2 - R_1|)$ 횟수 안에 끝나게 된다.

PARTITION-s를 이용한 분할 결과를 나타내는 그림 1(e)을 예로 들어 설명하겠다. 그림 1에서 $R_1 = CPU, B_{hw1}, D_{hw1}$이고 $R_2 = CPU, A_{hw1}, B_{hw1}$이므로 $R_1 - R_2 = D_{hw1}$이고 $R_2 - R_1 = A_{hw1}$이 된다. PARTITION-m은 모드 Π_1에 라이브러리를 $L = CPU, B_{hw1}, D_{hw1} \cup A_{hw1}$이고 $COST(A_{hw1}) = 0$ 이 되도록 설정하고 PARTITION-s 를 적용시킨다. (우리는 이 과정을 재분할-1이라 부른다.) 같은 방법으로 PARTITION-m은 모드 Π_2에 라이브러리를 $L = CPU, A_{hw1}, B_{hw1} \cup D_{hw1}$이고 $COST(D_{hw1}) = 0$이 되도록 설정하고 PARTITION-s를 적용시킨다. (우리는 이 과정을 재분할-2이라 부른다.) 위의 예에서 재분할-1 과정에서는 모듈이 바인딩된 자원이 변하지 않지만 재분할-2 과정에서는 시간제한 조건을 만족하면서 A_{hw1}이 가격이 영(0)으로 설정된 D_{hw1} 자원으로 대치되게 된다. (즉, $R_{unused} = A_{hw1}$) 재분할-2의 재스케쥴링과 재바인딩이 된 결과는 그림 1(g)에 있으며 모드 Π_1과 Π_2은 $R_1 \cup R_2 = CPU, B_{hw1}, D_{hw1}$ 자원을 사용하게 되어 전체 가격이 129에서 122로 줄어들게 된다.

• 2개 이상의 어플리케이션 모드 문제 $\Pi_1, \Pi_2 \cdots$, Π_K ($K>2$) : 두 개의 어플리케이션 모드에 적용했던 PARTITION-m 경우로부터 추론해보면 $R_1 \cup R_2 \cup \cdots \cup R_K$으로부터 자원비용의 합이 가장 크게 되는 집합 R_{unused}을 찾는 것이 목표가 된다. 이런 목적을 달성하기 위하여 두 개의 어플리케이션 모드에 적용했던 분할 방법을 반복적으로 적용한다. 문제를 풀기위해서 상향식으로 두 개의 모드를 위한 분할 방법을 적용하여 자원 가격을 줄인다. 처음에 각 모드 $\Pi_i, i = 1, 2, \cdots, K$ 에 PARTITION-s를 적용하여 R_i을 구한다. 그 후에 각 쌍의 자원에 대해서 PARTITION-m 그림 3 부분을 적용한다. 각 쌍의 어플리케이션 모드 Π_i과 Π_j에 대해서 자원 집합 $R_{i,j}$을 구하고 가장 가격 감소가 큰 쌍을 선택한다. (즉, $COST(R_i) + COST(R_j) - COST(R_{i,j})$가 가장 큰 모드 쌍) 그 후에 PARTITION-m을 선택된 2개의 어플리케이션 모드에 적용한다. 그리고, 이전에 선택된 두개의 모드는 한 개의 모드로 취급해서 K-1개의 모드에 대해서 재분할 과정을 적용한다. 모든 어플리케이션 모드가 한 개가 될 때까지 이 과정을 반복한다.

그림 4는 Π_1에서 Π_5까지 다섯 개의 모드에 대한 상향식 재분할 과정을 보여준다. 절약된 자원 가격 (= $COST(R_{unused})$)은 트리의 병합되는 지점에 표시되어 있다. 첫 번째, 두 번째, 세 번째 반복에서 (Π_1, Π_2), (Π_1, Π_2, Π_3), (Π_4, Π_5)가 가격을 10, 7, 4 만큼 줄이면서 재분할 되었다. 그리고 마지막 재분할 결과 전제 비용은 처음 단계보다 21 (=10+7+4)이나 줄었다.

5. 실험 결과

본 연구에서 제안한 HW/SW 분할 기법인 PARTITION-s와 PARTITION-m을 C++로 구현하였으며 3.06Ghz XEON 프로세서를 장착한 컴퓨터에서 논문 [9], [5]의

```
PARTITION-m (Π1, Π2, R1, R2) /* Repartition algorithm for Π1,Π2 */
    /* Π1,Π2: two application modes */
    /* R1,R2: sets of bound resources in Π1,Π2 */
• Runused = 0;
while (1)
    foreach (ri in R2 − R1)
        • Apply PARTITION-s to Π1 with L = R1 ∪ {ri}, COST(ri) = 0;
    foreach (ri in R1 − R2)
        • Apply PARTITION-s to Π2 with L = R2 ∪ {ri}, COST(ri) = 0;
    • Select ri whose corresponding application produces the largest
        value of ΔCOST^s−mode in Eq. (3);
    if (ΔCOST^s−mode ≤ 0) break;
    • Perform the selected application (i.e., repartioning), and
        generate new R'1 and R'2;
    • Runused = Runused ∪ (R2 − R1 ∪ R1 − R2) − (R'1 ∪ R'2));
    • Set R1 = R'1 or R2 = R'2;
endwhile
• return (Repartition result of Π1 and Π2, Runused);
```

그림 3 두개의 어플리케이션 모드 Π_1과 Π_2에 대한 PARTITION-m (Step 3)

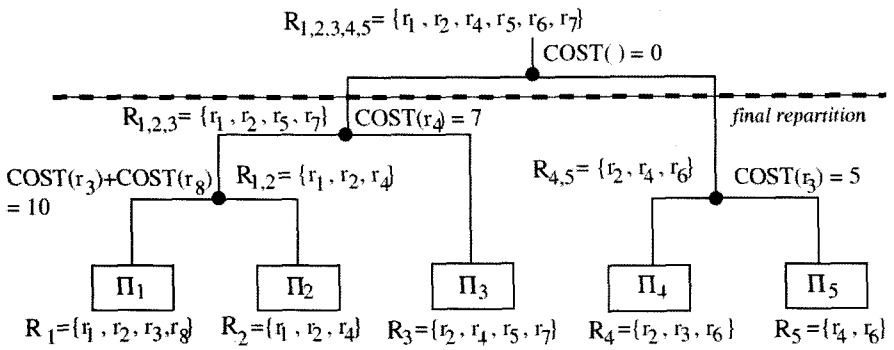


그림 4 5개의 어플리케이션 모드에 $PARTITION^{m-mode}$을 상향식으로 적용하여 얻은 HW/SW 재분할 과정의 결과를 보여주는 예

사용된 어플리케이션과 본 연구에서 만든 다양한 어플리케이션에 대해서 제안한 기법의 효율성을 평가하기위해 실험하였다. 본 제안된 기법은 세 가지 측면에서 평가 되었다. 즉, (1) PARTITION-m이 단순모드에 적용된 경우에 (즉, PARTITION-s) 얼마나 효율적으로 하나의 모드내의 모듈들 사이의 병렬성을 이용하여 사용할 자원의 총 비용을 낮출 수 있는지 평가하는 것과, (2) PARTITION-m이 멀티모드에 적용된 경우 서로 다른 모드에 속한 모듈들 사이의 병렬성을 얼마나 잘 이용하여 자원의 총 사용 비용을 낮추는지에 대한 평가, 그리고 (3) PARTITION-s와 PARTITION-m 자체에 대한 수행 시간 등 평가로 이루어졌다.

• **기존 방법과의 비교를 통한 PARTITION-s의 효율성 평가**: 표 1은 논문 [9]의 HW/SW 분할 기법과 PARTITION-s에 의한 시스템 비용을 정리하여 보여주고 있다. 논문 [9]의 MP3, VP, 그리고 VPL은 실생활에 쓰이는 예이며 각각 MP3 player, video phone, video player 이렇게 세 가지 어플리케이션 모드를 나타낸다. MP3+VP+VPL은 MP3, VP와 VPL 모드에 속한 모든 태스크가 모인 하나의 모드를 나타낸다. (MP3+VP+VPL)*0.8은 각 태스크의 시간제한 조건을 20% 줄인 MP3+VP+VPL 모드를 나타낸다. 논문 [9]의 라이브러리를 사용하여 MP3+VP+VPL과 (MP3+VP+VPL)*0.8 모드를 실험하였고 논문 [5]의 라이브러리를 사용하여 모드 ex1, ex2 와 ex3의 태스크를 하나로 모은 ex1+ex2+ex3 모드를 실험하였다. 표의 각 요소에는 전체 시스템 비용과 함께 할당된 자원이 표시되어 있다. 기존의 방법으로는 ex1+ex2+ex3과 (MP3+VP+VPL)*0.8 모드를 분할하지 못하는 것을 볼 수 있다. 반면에 PARTITION-s는 시간 제약 조건을 만족하는 분할 결과를 찾아내며, MP3+VP+VPL 모드의 경우에는 기존의 방법의 결과보다 14.4%의 비용 감소를 이루었다. 이로써 기존의 방법에서 활용하지 않은 모듈 수준의 병렬성(자원 공유)을 활용한 PARTITION-s가 더 좋은 성능을 보인다는 것을 확인하였다.

• **기존 방법과의 비교를 통한 PARTITION-m의 효과성 평가**: 표 2는 논문 [9]의 멀티모드 분할 방법과 PARTITION-m의 분할 결과를 비교하여 보여준다.

표 1 논문 [8]의 단순모드 분할 방법과 PARTITION-s가 분할한 설계의 시스템 비용 비교

어플리케이션	전체 시스템 비용		감소율
	기존 방법[9]	PARTITION-s	
MP3+VP+VPL [8]	309 (P1, ME_{hw}, VLC_{hw}, deQ_{hw}, MC_{hw}, FB_{hw}, Q_{hw}, DCT_{hw}, $IDCT_{hw}$)	250 (P1, ME_{hw}, DCT_{hw}, $IDCT_{hw}$, MC_{hw})	19.0%
(MP3+VP+VPL)*0.8 [8]	no feasible partition	270 (P1, ME_{hw}, DCT_{hw}, deQ_{hw}, $IDCT_{hw}$, VLC_{hw})	–
ex1+ex2+ex3 [5]	no feasible partition	320 (X, X, Y, Y, Link)	–

표 2 논문 [8]의 멀티모드 분할 방법과 PARTITION-m이 분할한 설계의 시스템 비용 비교

어플리케이션	전체 시스템 비용		감소율
	기존 방법[9]	PARTITION-s	
MP3,VP,VPL [8]	220 (P1, ME_{hw}, $IDCT_{hw}$)	220 (P1, ME_{hw}, $IDCT_{hw}$)	0.0%
(MP3,VP,VPL)*0.8 [8]	284 (P1, ME_{hw}, deQ_{hw}, Q_{hw}, DCT_{hw}, $IDCT_{hw}$)	240 (P1, ME_{hw}, DCT_{hw}, $IDCT_{hw}$)	15.7%
ex1,ex2,ex3 [5]	220 (x, x, Link)	170 (X, Y, Link)	22.7%

MP3, VP, VPL 모드에 대해서 논문 [9]의 분할 방법과 PARTITION-m의 분할 결과가 같다. 그러나 시간 제약 조건을 줄이게 되면 PARTITION-m은 기존의 방법보다 15.7%까지 시스템 비용이 적은 분할 결과를 만든다. 이는 논문 [9]의 방법에서는 태스크 수준의 자원 공유를 하지만 PARTITION-m에서는 세밀한 모듈 수준의 자원 공유를 하기 때문이다.

• PARTITION-m과 PARTITION-s 알고리즘의 효율성 평가: 표 3은 본 실험 과정에서 생성된 어플리케이션에 대해서 PARTITION-s와 PARTITION-m을 적용한 결과를 보여준다. 각 어플리케이션은 두 개의 모드로 구성이 되어있다. 두 번째와 세 번째 열은 각 모드에 속한 모듈의 개수를 나타낸다. 네 번째 열의 숫자는 중복되지 않은 고유한 기능을 가진 모듈의 개수를 나타낸다. 5번째, 6번째, 8번째 열은 각각 PARTITION-s, PARTITION-m에 의해 분할된 시스템 비용과 프로그램 실행시간을 나타낸다. 실행 시간 정보로부터 1000개가 넘는 모듈의 분할 문제를 푸는데 3시간 정도 걸리는 것을 볼 수 있으나 대부분의 실용적인 어플리케이션의 경우에는 몇 분 안에 풀 수 있었다. 논문 [6,10]의 유전 알고리즘을 이용한 파티션 방법에 사용한 리스트 스케줄 방법을 구현하여 우리의 스케줄과 비교해 보았다. 구현한 리스트 스케줄은 모듈의 가장 빠른 시작 시간과 가장 늦은 시작 시간의 차이를 이용해서 모듈에 우선순위를 부여하였다. .#fails-by-list-scheduler 열의 'n/m'은 PARTITION-m을 실행시키는 동안에 m번의 list scheduling을 적용시키는데 n번의 경우 시간 제약 조건을 만족 시키지 못했음을 보여준다.

결론적으로 그림 5의 곡선은 PARTITION-s의 반복 횟수에 따른 분할 결과의 개선 정도를 보여준다. MP3+VP+VPL에 대해서 PARTITION-s는 9번의 반복을 해서 시스템 비용을 1904에서 250까지(표 1에 표시된 것처럼) 낮추었다. 그리고 VP와 VP*0.8의 경우에는 6번의 반복 후에 종료되었다.

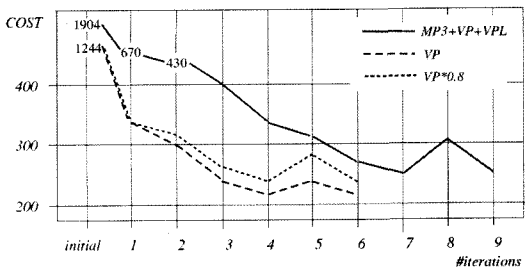


그림 5 PARTITION-s의 바깥 루프의 반복에 따른 시스템 비용 감소 변화를 보여주는 그래프

6. 결 론

본 연구에서는 멀티모드 멀티태스크 HW/SW 분할 문제를 포괄적으로 푸는 기법을 제안했다. 우리가 제안하는 기법은 기존의 연구에서는 고려하지 않았던 모듈 수준 병렬성과 연관된 두 가지 중요한 문제를 풀었다. 두 가지 문제는 (1) 모드 내에서의 자원 공유하는 문제와 (2) 다른 모드 사이의 자원을 공유하는 문제이다. 반복적인 개선 방법을 사용하여 자원 바인딩과 스케줄 가능성 검사를 동시에 하는 PARTITION-s를 제안하여 단순모드 문제를 풀었다. 그리고 PARTITION-s에 의해서 생성된 모드 마다 할당된 자원 집합을 전체적으로 최적화하는 PARTITION-m을 제안하여 멀티모드 문제를 풀었다. 그리고 실험 데이타를 통해서 PARTITION-s와 PARTITION-m가 단순모드 및 멀티모드 멀티태스크 어플리케이션을 분할하는데 시스템 비용을 각각 19%와 17% 줄일 수 있는 것을 확인하였다. 다양한 실험 결과에 대한 분석으로, 우리는 특히, 태스크 내의 모듈들 간에 사용될 자원이 공유될 여지를 많이 내포하는 어플리케이션에는 PARTITION-m 방법이 HW/SW 분할을 하는데 있어서 아주 효과적으로 쓰일 수 있음을 확신할 수 있었다.

본 기법의 단점으로는 첫째, 본 알고리즘은 K-L[13]의 그래프 분할 알고리즘에 기반되었기 때문에 많은 반

표 3 PARTITION-s의 결과와 PARTITION-m에 의해서 생성된 결과의 시스템 비용 감소 비교

어플리케이션	모듈 총 수		중복제거 한 모듈 총 수	시스템 비용		#fails-by list-scheduler	run-time (초)
	Π_1	Π_2		PAR-s	PAR-m		
test1	44	431	66	286	242	6/22	3698
test2	504	47	72	381	370	1/23	3258
test3	264	504	78	460	424	7/32	3158
test4	48	253	76	401	389	2/13	219
test5	47	446	75	541	482	3/21	1527
test6	51	600	83	280	270	7/25	6367
test7	267	54	88	315	313	1/46	10671
test8	633	632	87	494	467	5/25	13063

복 수행에 따른 수행 시간이 많이 들게 된다. 따라서 알고리즘 수행 시간을 줄이면서 결과의 질을 최대한 유지할 수 있는 추가적인 연구가 앞으로 있어야 할 것으로 보이며, 둘째, 본 연구에서는 HW/SW 분할 문제의 가정에서 모드(mode)들 간에는 병렬적인 수행을 고려하지 않았는데 실제 응용에서는 하나 이상의 모드가 중복해서 특정 작업을 수행할 수 있을 것이다. 이러한 경우에 대한 HW/SW 분할 연구에 대해서 심도 있는 연구가 있어야 할 것이다.

참 고 문 헌

[1] C. L. Liu and J. W. Layland, "Scheduling Algorithm for Multiprogramming in a Hard Real-time Environment," *Journal of ACM*, Vol. 20, pp. 46-61, Jan. 1973.

[2] N. Audsley, A. Burns, M. Richardsons, and W. Wellings, "Hard Realtime Scheduling: The Deadline-monotonic Approach," *Proc. IEEE Workshop on Real-time Operating Systems and Software*, pp. 133-137, 1991.

[3] S. Banerjee and N. Dutt, "Efficient Search Space Exploration for HWSW Partitioning," *Proc. International Conference on Hardware/Software Codesign and System Synthesis*, pp. 122-127, 2004.

[4] F. Vahid and T. D. Le, "Extending the Kernighan-Lin Heuristic for Hardware and Software Functional Partitioning," *Journal of Design Automation for Embedded Systems*, Vol. 2, 1997.

[5] J. Hou and W. Wolf, "Process Partitioning for Distributed Embedded Systems," *Proc. International Workshop on Hardware/Software Codesign*, pp. 70-76, 1996.

[6] R. P. Dick and N. K. Jha, "MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Cosynthesis of Distributed Embedded Systems," *IEEE Transactions on Computer-Aided Design of Integrated Systems*, Vol. 17, No. 10, pp. 920-935, Oct. 1998.

[7] Y. Shin, D. Kim, and K. Choi, "Schedulability-Driven Performance Analysis of Multiple Mode Embedded Real-time Systems," *Proc. Design Automation Conference*, pp. 495-500, 2000.

[8] H. Ok and S. Ha, "A Hardware-Software Cosynthesis Technique Based on Heterogeneous Multiprocessing Scheduling," *Proc. International Workshop on Hardware/Software Codesign*, pp.183-187, 1999.

[9] H. Ok and S. Ha, "Hardware-Software Cosynthesis of Multi-Mode Multi-Task Embedded Systems with Real-time Constraints," *Proc. International Workshop on Hardware/Software Codesign*, pp. 133-138, 2002.

[10] M. T. Schmitz, B. M. Al-Hashimi and P. Eles, "Cosynthesis of Energy-Efficient Multimode Embedded Systems With Consideration of Mode-Execution Probabilities," *IEEE Transactions on Computer-Aided Design of Integrated Systems*, Vol. 24, No. 2, pp. 153-169, Feb. 2005.

[11] A. Kalavade and P.A. Subrahmanyam, "Hardware/Software Partitioning for Multifunction Systems," *IEEE Transactions on Computer-Aided Design of Integrated Systems*, Vol. 9, No. 9, pp. 819-837, Sep. 1998.

[12] S. Banerjee, E. Bozorgzadeh, and N. Dutt, "Physically-Aware HW/SW Partitioning for Reconfigurable Architecture with Partial Dynamic Reconfiguration," *Proc. Design Automation Conference*, pp. 335-340, 2005.

[13] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell System Tech. Journal*, 1970.

[14] C. Lee and S. Ha, "Hardware-Software Cosynthesis of Multi-Task MPSoCs with Real-time Constraints," *Proc. International Conference on ASIC*, pp. 24-27, 2005.

[15] 조중휘, 손요한, *H.264 and MPEG-4: 차세대 영상압축기술*, 홍릉과학출판사, 2004.

김 영 준

2004년 한양대학교 전자과 학사. 2006년 서울대학교 전기컴퓨터공학부 석사. 현재, 삼성전자 System LSI 연구원. 관심분야는 Hardware/Software codesign, Embedded system design

김 태 환

1985년 서울대학교 계산통계학과 학사. 1987년 서울대학교 계산통계학과 석사. 1993년 미국 일리노이주립대 전산학과 박사. 현재, 서울대학교 전기컴퓨터공학부 교수. 관심분야는 Embedded system design