하드웨어/소프트웨어 통합설계를 위한 FDS 분할 알고리즘의 성능개선

오 주 영[†] · 이 면 재[††] · 이 준 용[†††] · 박 도 순[††††]

요 약

하드웨어/소프트웨어 통합설계를 위한 대부분의 분할 알고리즘들은 스케줄링을 고려하지 않기 때문에 분할결과를 스케줄 하였을 때에 시간제약을 만족하지 못한다면 재분할 해야하는 오버헤드가 있다. 분할 단계에서 스케줄링을 함께 고려하는 FDS를 응용하는 기존의 방법들은 분할될 노드를 선택하면서 그 노드가 스케줄 되어야 하는 제어구간을 결정한다. 분할될 노드의 선택은 한 노드를 분할함으로써 상승하는 비용 또는 시간과 그 노드의 스케줄로 인해 다른 노드들의 스케줄을 방해하는 정도를 함께 고려한다. 이때, 다른 노드들의 스케줄을 방해하는 정도를 의미하는 유도 힘은 자신과 종속성을 갖는 노드들의 모든 제어구간에서 계산된다. 본 논문은 FDS를 응용하는 분할 방법으로서 노드의 스케줄 긴박도와 상대적 스케줄 긴박도를 정의하여 분할하는데, 노드들의 모빌리티 중에서 처음 제어단계와 마지막 제어단계에서의 상대적 스케줄 긴박도 계산만으로 분할을 결정하기 때문에 기존의 FDS 응용 방법에서의 유도힘 계산에 소요되는 시간복잡도를 개선한다. 벤치마크들에 대한 실험결과는 기존의 FDS 응용 방법과 비교해서 개선된 알고리즘 실행시간을 보인다.

Performance Improvement of Force-directed Partitioning Algorithm for HW/SW Codesign

Juyoung Oh[†] · Myounjae Lee[††] · Junyong Lee[†††] · Dosoon Park[††††]

ABSTRACT

Most partitioning algorithms for hardware-software codesign do not consider scheduling. Therefore, partitioning should be performed again if time constraints are not satisfied in scheduling the partitioned results. Existing FDS-applied methods which consider scheduling in partitioning decide the control step of the node to schedule while selecting nodes for partitioning. In selecting nodes for partitioning, several aspects should be considered together such as added cost or time due to the partition of the node, or the degree of interference due to the scheduling of the node. At this time, the induced force, which means the degree of intereference of scheduling other nodes, is computed all over the control step of the corresponding node and other depending nodes. In this paper, a new FDS-applied partitioning algorithm is proposed, where partitioning is performed using the defined scheduling urgency and relative scheduling urgency of the nodes. Since the nodes are partitioned by the computation of relative scheduling urgencies only at the earliest control step and the latest control step among the assignable steps, the time complexity for the computation of induced force could be improved. Experimental result on the benchmarks show the improvement of execution time of the proposed algorithm compared to the existing FDS-applied methods.

키워드 : 통합설계(Codesign), 하드웨어/소프트웨어 분할(Hardware-software Partition), 스케줄링(Scheduling), 상대적 스케줄 긴박도 (Relative Scheduling Urgency)

1. 서 론

내장형 시스템은 ASIC 또는 FPGA 등의 하드웨어와 소프트웨어 실행을 위한 프로세서로 구성된다[1]. 하드웨어/소프트웨어 통합설계는 최소의 비용으로 최적의 성능을 갖는 시스템을 얻어야 하는데, 이때 가장 큰 영향을 미치는 작업단계가 분할이며 이 문제는 NP-hard[2]이다. 이를 해결하기 위하여 그리디(greedy), 클러스터링, 반복 개선 방법뿐만 아니라 설계 대상 시스템 환경과 목적함수에 따른 다양한 휴리스틱 알고리즘들[3]이 개발되어왔다.

대부분의 휴리스틱 알고리즘들은 분할과 스케줄링 작업을 독립적으로 진행함으로써 분할 이후의 스케줄링과 자원할당으로 인하여 시스템 시간제약을 위배할 수 있다. 시간제약 위배는 시스템을 재분할하는 비용을 유발하게 되므로 통합설계에서는 분할과 스케줄링을 함께 고려하는 것이 효율적이다. 분할과 스케줄링을 함께 고려하는 FDS(Force Directed Scheduling)를 응용하는 방법[6, 7]들은 힘값을 갖고 분할단계에서 노드들중의 하나를 스케줄하며 분할한다. 이를 위해 노드를 분할할 때 증가되는 비용 또는 시간에 대한 힘과 분할로 인해 다른 노드들의 스케줄을 방해하는 힘인 유도

† 정 회 원 : 경인여자대학 교수
†† 준 회 원 : 홍익대학교 대학원 전자계산학과
††† 정 회 원 : 홍익대학교 컴퓨터공학과 교수
†††† 종신회원 : 홍익대학교 컴퓨터공학과 교수
논문접수 : 2002년 9월 27일, 심사완료 : 2002년 11월 8일

힘의 합을 하드웨어 부분과 소프트웨어 부분에 대해 구하여 이 값이 가장 작은 노드를 선택하여 분할한다. 이러한 FDS 응용방법에서는 종속성을 갖는 노드들에 대해 그 노드들의 모든 제어구간에서 유도힘을 계산한다.

본 논문에서는 하나의 프로세서와 하나의 ASIC 구조에 대해 시간제약 조건을 만족하면서 하드웨어 비용의 최소화를 목적함수로 하고, 기존의 FDS를 응용하는 방법보다 낮은 시간복잡도의 FDS 응용 분할 알고리즘을 제안한다. 이를 위해 하나의 노드가 하나의 제어단계에 스케줄되어야 하는 우선순위를 의미하는 스케줄 긴박도와 다른 노드들과의 경쟁을 고려한 상대적 스케줄 긴박도를 정의하여 사용한다. 이때 분할을 결정하는 힘인 상대적 스케줄 긴박도는 다른 노드와 모빌리티 구간의 중첩이 적은 처음 배정가능 제어단계와 마지막 배정가능 제어단계에 대해서만 계산함으로써 모든 배정가능 제어단계에서의 유도힘 계산을 하는 기존 알고리즘의 시간복잡도를 개선한다.

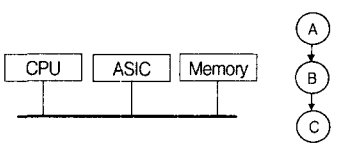
본 논문의 구성은 다음과 같다. 2장에서는 관련연구를 살펴보고, 3장에서는 제안분할 알고리즘을 설명한다. 4장에서는 실험 및 결과를 설명하고, 5장에서 결론을 기술하였다.

2. 관련 연구

Gupta[4]등은 데이터 종속지연연산(data-dependent delay operation)을 제외한 모든 연산을 초기에는 하드웨어로 배정한 후에 주어진 시간제약조건을 만족하는 범위에서 greedy 알고리즘을 사용하여 한 연산씩 소프트웨어로 보내는 방법을 제안하였는데, 현재의 해로부터 하드웨어 비용이 감소되는 쪽으로만 단방향 탐색(serial-traversal)을 하므로 국부최적치에 빠질 수 있다. 이러한 단점을 극복하기 위해 Kalavade[2]등은 하드웨어 자원과 실행시간을 최적화하는 두 가지 목적함수를 정의하였는데, 목적함수의 변경은 분할된 노드들을 스케줄하여 시간제약과의 비교에 의해 얻어지는 전역시간에 의한다. 즉 전역시간이 임계점에 도달하면 목적함수는 실행시간의 감소가 되며, 그렇지 않은 경우에는 하드웨어 자원의 감소가 목적함수가 되도록 하여 분할을 한다. 이러한 대부분의 알고리즘들은 분할과 스케줄링 작업단계를 독립적으로 진행함으로써 분할 이후의 스케줄링과 자원할당으로 인하여 시스템 시간제약을 위배할 수 있다. 시간제약 위배는 시스템을 재분할하는 비용을 유발하게 되므로 통합 설계에서는 분할과 스케줄링을 함께 고려하는 것이 효율적이다[5]. FDS를 응용하는 방법[6, 7]은 분할과 스케줄링을 함께 고려하는데, FDS[8]는 주어진 시간제약내에서 스케줄 대상 노드를 각 제어구간에 균등하게 분포시킴으로써 하드웨어 비용을 최소화하는 스케줄 방법이다. Rousseau[6]등은 설계 대상 구조를 하나의 프로세서와 하나의 ASIC 구조로 하여 FDS를 응용할 때에 비용함수(실행시간 또는 설계비용)에 따라 계산되는 힘값을 갖고 분할단계에서 노드들중의 하나를 스케줄하며 분할한다. 노드를 분할할 때 증가되는 설계비용을 자신의 힘으로 정의하고, 분할 이후 그 노드의 스케줄로 인해서 삭제되는 다른 노드들의 모빌리티를 유도힘으로 정의하여, 두 개의 힘의 합을 하드웨어 부분과 소프트웨어 부분에 대해 구하여 이 값이 가장 작은 노드를 선택하여 분할한다. 이 방법의 시간복잡도는 제어구간의 수가 c이고 노드 개수가 n일 경우, 알고리즘이 한번 반복될 때마다 하나의 노드만이 분할되고 힘 계산은 각 제어구간에 배정될 수 있는 모든 노드들에 대해 계산하므로 $O(cn^2)$이며, 각 제어구간에서의 힘 계산에서는 다른 모든 노드들에 대해 그 노드들의 배정 가능한 제어단계에서 유도 힘을 계산하므로 $O(c^2 n^3)$이다. Choi[7]등은 시스템 실행시간과 구현 비용을 최소화하기 위하여, 처음에 모든 노드를 소프트웨어로 배정하고 시간제약을 만족할 때까지 한번에 하나의 노드를 선택하여 하드웨어로 보내는 것을 반복한다. 하드웨어로 분할될 노드의 선택을 위한 힘은 그 노드의 모빌리티 구간에 걸쳐서 소프트웨어에서 병행해서 실행될 수 있는 노드들과 하드웨어로 선택할 때 상승하는 비용을 반영한다. 대부분의 분할 알고리즘들은 하나의 프로세서와 하나의 ASIC 구조에 대한 것이지만, Liu[5]는 두 개의 프로세서와 k개의 하드웨어 자원을 갖는 구조에 대해 처음에 모든 태스크를 소프트웨어로 분할하고 이에 대한 최적의 스케줄 결과를 구한다. 이후에 두 개의 프로세서에 스케줄된 각 태스크들에 대해 하드웨어로 배정했을 때의 이동성(이전 제어단계로 이동가능, 이후 제어단계로 이동가능, 이동불가능)을 계산한다. 제어단계별로 이동성이 있는 태스크들의 하드웨어 비용을 계산하여 최소비용을 갖는 태스크들을 하드웨어로 분할하며, k개의 태스크들을 하드웨어로 이동시킬 때까지 반복한다.

3. 제안 분할 알고리즘

분할을 위한 대상 아키텍처는 (그림 1)(a)와 같이 소프트웨어 실행을 위한 하나의 일반적인 프로세서와 확장 가능한 하나의 하드웨어 모듈이다. 또한 입력은 (그림 1)(b)와 같이 태스크를 명시하는 노드와 노드간의 종속성을 명시하는 간선으로 구성되는 방향성 비순환 그래프(DAG)와 (그림 1)(c)와 같은 노드 정보인 HT_i, ST_i, HC_i, SC_i이며 각각은 노드 i의 하드웨어에서의 실행 시간, 소프트웨어에서의 실행시간, 하드웨어 구현비용, 그리고 소프트웨어 구현 비용을 의미한다. 그러면 주어지는 시간제약과 노드의 실행 시간을 이용하



	HT_i	ST_i	HC_i	SC_i
A	1	3	2	1
B	3	6	3	1
C	1	2	1	1

(a) 대상 아키텍처 (b) 입력 예 (c) 노드의 정보

(그림 1) 분할 환경

여 입력 DAG로부터 ASAP(As Soon As Possible) 스케줄링과 ALAP(As Late As Possible) 스케줄링에 의해 하드웨어/소프트웨어 분포그래프를 생성할 수 있다.

3.1 노드의 스케줄 긴박도

하나의 노드가 스케줄될 수 있는 제어단계 집합중에서 하나의 제어단계에 스케줄되어야 하는 필요성의 정도를 스케줄 긴박도로 정의한다. 즉, 모빌리티가 작은 노드가 우선 스케줄되고, 같은 크기의 모빌리티를 갖는 경우에는 비용이나 실행시간이 작은 노드를 우선 스케줄하기 위한 힘이다.

$$distr_{hard}(i)=1/n_{hard}, \quad distr_{soft}(i)=1/n_{soft} \qquad (1)$$

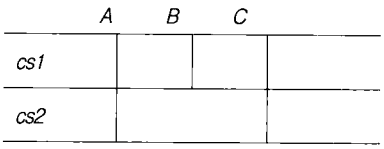
식 (1)은 하드웨어와 소프트웨어 분포그래프에서 모빌리티 n_{hard}과 n_{soft}를 갖는 노드 i가 하나의 제어단계에 스케줄될 확률[6, 7]이다. 확률값이 큰 노드는 상대적으로 모빌리티가 작기 때문에 이러한 노드에 대한 분할이 지연된다면 시간제약을 만족시키기 위해 실행시간이 짧은(일반적으로 구현 비용이 높음)쪽으로 분할될 수 있으므로 비용을 줄이기 위해서는 확률값이 큰 노드를 우선 선택하여 분할해야한다. 소프트웨어로의 분할을 고려하는 경우에는 그 노드의 실행시간이 길수록 다른 노드의 소프트웨어로의 스케줄을 방해하는 힘 또한 커진다. 따라서 소프트웨어의 경우에는 스케줄될 확률이 크고 실행시간이 짧은 노드가 우선 선택될 수 있도록 스케줄 긴박도를 정의한다. 이때, 확률과 실행시간을 반영한 곱이 같은 경우에는 이 노드들중에서 하드웨어 구현 비용이 상대적으로 큰 노드가 우선 선택될 수 있도록 하드웨어 구현비용 HC_i를 반영함으로써 시스템 설계비용을 줄일 수 있다. 하드웨어로의 분할을 고려하는 경우에는 노드의 하드웨어 구현비용만큼 시스템 설계 비용이 증가되므로 스케줄될 확률이 크고 구현 비용이 작은 노드가 우선 선택될 수 있도록 스케줄 긴박도를 정의한다. 이때 확률과 구현비용을 반영한 곱이 같은 경우에는 이 노드들중에서 소프트웨어 실행시간이 상대적으로 긴 노드가 우선 선택될 수 있도록 소프트웨어 실행시간 ST_i를 반영한다. 이러한 노드를 하드웨어로 우선 분할하면 후에 다른 노드들의 소프트웨어 분할 가능성을 높일 수 있다. 식 (2)와 식 (3)이 정의한 힘 즉 노드의 스케줄 긴박도이며, $Cost-function$은 하드웨어 또는 소프트웨어로 분할할 때의 구현비용과 실행시간을 의미한다.

$$Urgency^{i}_{soft}(i) = \left(distr_{soft} \times \frac{1}{Cost-function(i, sw-implementation)}\right) \times \beta \times HC_i \qquad (2)$$

$$Urgency^{i}_{hard}(i) = \left(distr_{hard} \times \frac{1}{Cost-function(i, hw-implementation)}\right) \times \alpha \times ST_i \qquad (3)$$

3.2 상대적 스케줄 긴박도

노드의 한 제어단계에서의 스케줄 긴박도에서 그 제어단계에 모빌리티를 갖는 다른 노드들의 스케줄 긴박도를 감한 값을 상대적 스케줄 긴박도로 정의한다. 일반적으로 하드웨어 분포그래프의 경우에 같은 제어구간에 스케줄되는 노드가 많을수록 설계비용은 증가하고, 소프트웨어 분포그래프의 경우에는 한 노드의 스케줄로 인해 그 제어구간에 모빌리티를 갖던 다른 노드들의 스케줄 긴박도가 증가하게 되므로 같은 제어구간에 모빌리티를 갖는 다른 노드들의 상대적인 힘을 반영해서 스케줄을 결정해야한다. 예를들어 (그림 2)의 분포그래프에서 노드 A는 두 개의 제어구간 $cs1$과 $cs2$에서 스케줄 긴박도는 같지만, $cs1$에서의 경쟁 노드수가 $cs2$에서의 경쟁 노드 수보다 많기 때문에 상대적 스케줄 긴박도는 $cs2$에서 커야하며, 노드 A를 분할할 경우에 $cs2$에 스케줄함으로써 다른 노드들의 스케줄에 여유를 줄 수 있다.



(그림 2) 상대적 스케줄 긴박도

상대적 스케줄 긴박도가 크다는 것은 스케줄 긴박도가 같을지라도 그 제어단계에서 경쟁하는 다른 노드들에 영향을 주는 정도가 적음을 의미한다. 노드의 상대적 스케줄 긴박도는 식 (4)와 식 (5)와 같은데, 노드 i가 제어구간 j에 스케줄되어야 하는 스케줄 긴박도로부터 제어구간 j에 스케줄될 수 있는 즉 모빌리티를 갖는 다른 노드들의 스케줄 긴박도의 합을 감하여 계산한다.

$$Rel-Urgency^{i}_{soft}(i) = Urgency^{i}_{soft}(i) - \sum_{k=1(k\neq i)}^{n} Urgency^{i}_{soft}(k) \qquad (4)$$

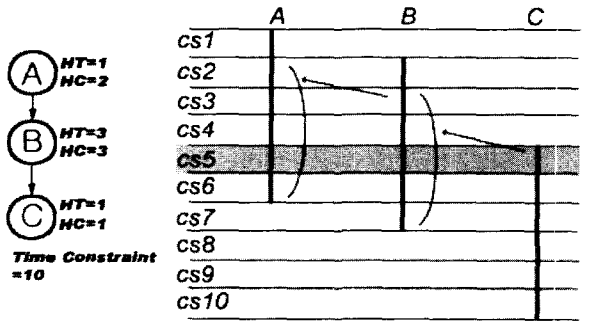
$$Rel-Urgency^{i}_{hard}(i) = Urgency^{i}_{hard}(i) - \sum_{k=1(k\neq i)}^{n} Urgency^{i}_{hard}(k) \qquad (5)$$

그러면 하드웨어 및 소프트웨어 분포그래프의 모든 제어단계에서 구한 상대적 스케줄 긴박도중에서 식 (6)과 같이 최대의 값을 갖는 노드가 우선적으로 분할 및 스케줄 대상이 된다.

$$Rel-Urgency_i(i) = \max\{Rel-Urgency^{i}_{soft}(i), Rel-Urgency^{i}_{hard}(i)\} \qquad (6)$$

일반적으로 상대적 스케줄 긴박도는 한 노드를 특정 제어단계에 스케줄할 때에 그 제어단계에 모빌리티를 갖는 다른 노드의 개수가 적을수록 커진다. 예를들어 (그림 3)의

하드웨어 분포그래프에서 3개 노드들의 스케줄될 확률은 동일하므로, 노드 A, B, C중에서 자신의 힘인 스케쥴 긴박도는 구현 비용이 가장 작은 노드 C가 가장 크다. 그런데 노드 C의 각 제어단계에서의 상대적 스케줄 긴박도는 처음 제어단계 *cs5*에서 작고 마지막 제어단계 *cs10*에서 큰데 이는 처음 제어단계에서 스케줄 경쟁을 하는 다른 노드들의 힘의 합이 크기 때문이다. 이와 같이 노드들의 종속성으로 인해 다른 노드들의 스케줄 방해의 힘은 일반적으로 입력의 소스 노드에 가까울수록 처음 제어단계에서의 값이 크며, 터미널 노드에 가까울수록 마지막 제어단계에서의 값이 크다. 따라서, 상대적 스케줄 긴박도의 계산은 모빌리티의 처음 제어단계와 마지막 제어단계로 한정할 수 있어서, 분할 대상 노드를 선택하기 위한 힘 계산시에 제어단계 수에 의해 부과되던 시간복잡도를 줄일 수 있다.



(그림 3) 각 제어단계에서 스케줄 방해의 비교

3.3 제안 알고리즘

상대적 스케줄 긴박도를 사용하는 제안 분할 알고리즘은 <표 1>과 같다. 단계 2에서는 입력받은 시간제약 내에서 형성될 수 있는 분포그래프를 하드웨어와 소프트웨어 각각에 대해 생성한다. 단계 3에서 각 분포그래프내의 노드들에 대해 상대적 스케줄 긴박도를 계산하여 이 값이 최대인 노드를 선택하여 분할하고, 분할에 따른 분포그래프를 수정하며 모든 노드가 분할될 때까지 이 단계를 반복한다.

〈표 1〉 제안 알고리즘

단계 1 *DAG* 및 노드특성, 시간제약 입력
단계 2 *ASAP*, *ALAP* 스케줄링으로 분포그래프(*DG*) 생성
단계 3 ***repeat until*** (모든 노드가 분할될 때까지)
 단계 3.1 ***for*** (분할 대상 노드)
  모빌리티의 처음 제어단계와 마지막 제어단계에서 상대적 스케줄 긴박도 계산
  end for
 단계 3.2 최대의 상대적 스케줄 긴박도를 갖는 노드를 선택하여 분할
 단계 3.3 ***for*** (선택된 노드를 제외한 분할 대상 노드)
  *HW DG*와 *SW DG*의 모빌리티 수정
  end for
 단계 3.4 선택 노드를 분할 대상 노드 집합에서 삭제
단계 4 ***end repeat***

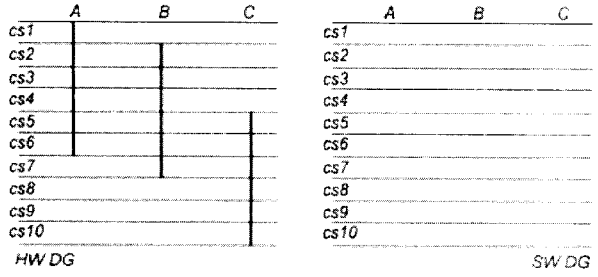
3.3.1 시간복잡도

제안 알고리즘은 각 노드의 상대적 스케쥴 긴박도에 의해 분할을 수행하는데, 스케줄을 경쟁하는 노드 수가 적어서 상대적 스케줄 긴박도가 크게 되는 처음 배정가능 제어단계와 마지막 배정가능 제어단계에서의 계산만으로 분할을 결정할 수 있으므로 모든 제어단계에서 유도힘을 계산하는 기존 방법의 시간복잡도를 개선할 수 있다. 제안 알고리즘의 시간복잡도는 노드의 개수가 n인 경우에, 한번에 하나의 노드가 선택되어 분할되므로 $O(n)$이며, 이때 n개의 각 노드의 처음 배정가능 제어단계와 마지막 배정가능 제어단계에서 상대적 스케줄 긴박도를 계산하므로 $O(2n)$이 되어 전체적인 시간복잡도는 $O(2n^2)$즉 $O(n^2)$이 된다.

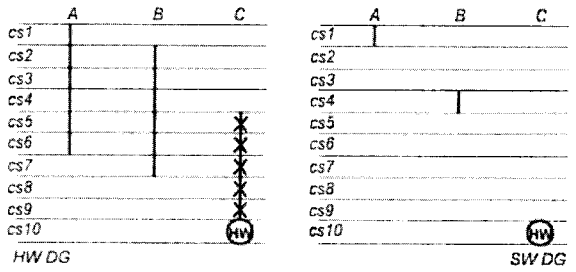
4. 실험 및 결과

(그림 1)의 분할 환경과 시간제약 10에 대한 제안 알고리즘의 진행과정은 (그림 4)와 같다. 처음에는 시간제약으로 인하여 소프트웨어 분포그래프는 생성되지 않으며, 노드 C가 하드웨어의 *cs10*에 분할되고, (그림 4)(b)와 같이 노드 A, B의 소프트웨어 분포그래프가 생성된다. 그러면 (그림 4)(c)와 같이 소프트웨어의 실행시간이 짧은 노드 A가 소프트웨어의 *cs1*에 분할되고, 마지막으로 노드 B는 소프트웨어의 *cs4*에 분할되는데, 이 분할 결과는 ILP[9]에 의한 결과와 동일하다. (그림 4)(c)에서 노드 B의 모빌리티중에서 X표시가 된 *cs2*와 *cs3*은 노드 A의 분할에 따라 노드 A와 종속 관계에 있던 노드 B의 모빌리티 축소 즉 수정을 의미한다.

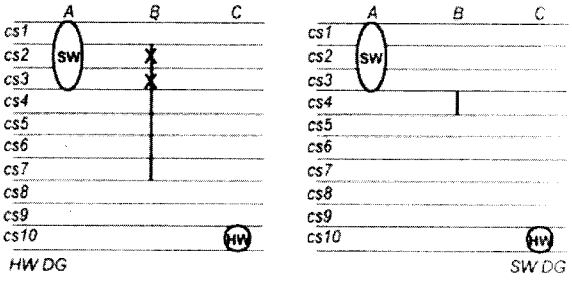
제안 알고리즘은 Pentium 컴퓨터에서 C++로 구현하였으며, 성능평가는 시스템 구현비용과 알고리즘 실행시간으로 최적의 결과를 보이는 ILP[9] 및 FDS응용에 의한 분할 방법[6]과 비교하였다. 상위수준 합성에서의 스케줄링을 위한 벤치마크들[11, 13, 14]과 통합설계에서의 분할을 위한 벤치마크들[6, 10, 12, 13, 15]에 대한 시간제약은 전체노드의 하드웨어 실행시의 임계경로 시간을 시간제약 1로 부여하고, 전체 노드의 소프트웨어 실행시의 임계경로 시간과 시간제약 1의 평균 시간을 시간제약 2로 부여하여 실험하였으며, 면적으로 표현되는 시스템 합성 비용과 알고리즘 실행에 소요되는 CPU 시간의 실험 결과는 (그림 5)와 같다. 제안 알고리즘의 실행시간은 (그림 5)(a)와 같이 모든 벤치마크에 대해 기존의 방법보다 향상되었으며 제어단계의 개수가 많아지는 시간제약 2의 경우에 CPU 시간은 크게 감소된다. 반면, 시스템 합성 비용은 (그림 5)(b)와 같이 대부분의 경우에 기존 방법의 결과와 동일하지만 비용이 크게되는 경우도 있다. 이는 상대적 스케쥴 긴박도를 노드의 모빌리티의 처음 제어단계와 마지막 제어단계에 대해서만 계산하는데, 여러 개의 그래프로 구성된 입력의 경우에는 모빌리티의 중간 제어단계에서의 상대적 스케쥴 긴박도가 크게 나타나기 때문이다.



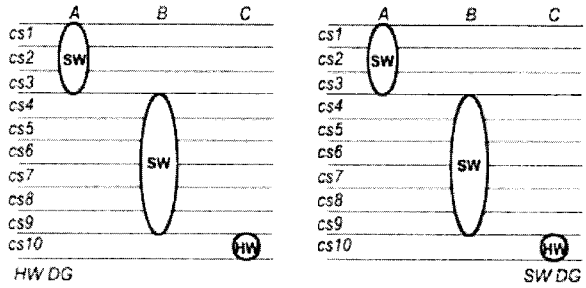
(a) 초기 분포그래프



(b) 노드 C를 분할한 이후의 분포그래프

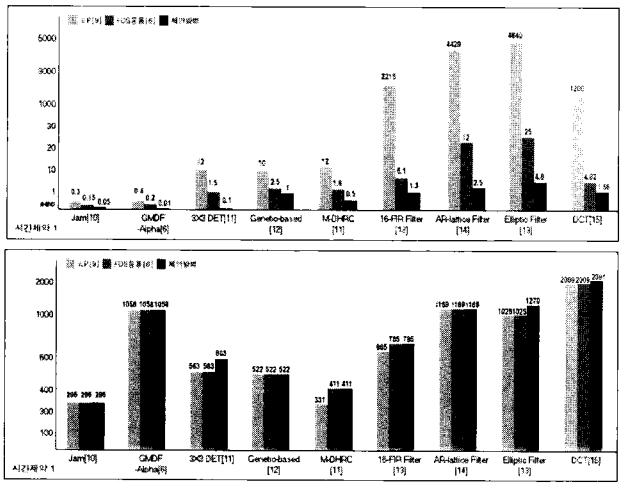


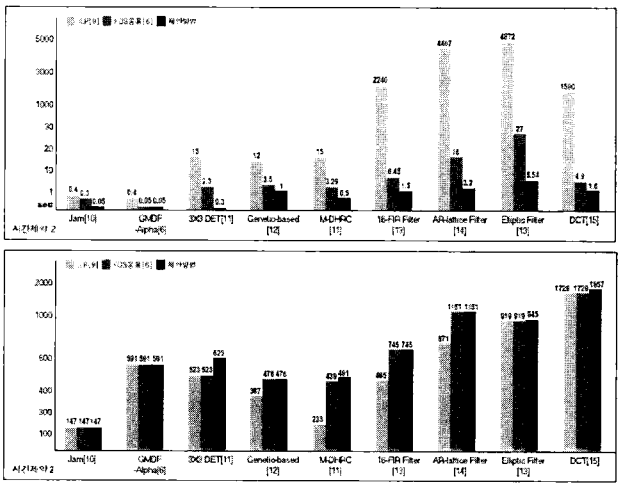
(c) 노드 A를 분할한 이후의 분포그래프



(d) 분할 결과

(그림 4) 분할과정





(a) 알고리즘 실행시간

(b) 시스템 설계비용

(그림 5) 실험 결과

5. 결 론

본 논문은 내장형 시스템을 위한 통합설계의 분할 단계에서 스케줄링을 함께 고려하는 FDS를 응용하는 분할 방법으로, 기존의 방법보다 낮은 시간복잡도의 알고리즘을 제안하였다. 이를 위해 분할하여 구현할 때에 소요되는 비용 및 시간에 의해 노드의 스케줄 긴박도와 상대적 스케줄 긴박도를 정의하고, 노드들의 모빌리티중에서 처음 제어단계와 마지막 제어단계에서의 상대적 스케줄 긴박도를 계산하여 분할을 수행하였다. 제안 알고리즘의 시간복잡도는 $O(n^2)$이어서, 기존의 FDS 응용 방법[6]에 비해 감소된 알고리즘 실행시간을 얻을 수 있는데, 여러 개의 그래프로 구성된 입력의 경우에는 모빌리티의 중간 제어단계에서의 상대적 스케줄 긴박도가 커질 수 있어서 설계비용이 상승하기도 한다.

향후 연구 과제는 여러 개의 그래프로 구성된 입력의 경우에 모빌리티의 중간 제어단계에서의 상대적 스케줄 긴박도가 크게되어 설계 비용이 상승하는 경우를 반영하기 위해 주어진 시간제약 내에서 형성되는 모빌리티를 충분히 사용할 수 있는 방법을 찾는 것이다. 또한, 현실적인 시스템 합성에 적용할 수 있도록 통신의 지연 시간을 고려하고 2개 이상의 CPU와 2개 이상의 하드웨어 모듈 등의 다양한 목적 시스템으로 확장할 것이다.

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] A. Kalavade and E. A. Lee, "A Global Critically/Local Phase Driven Algorithm for the Constrained Hardware/Software Partitioning Problem," 3th *International Workshop on Hardware/Software Codesign*, Grenoble, pp.42-48, 1994.

[3] X. Hu, B. T. Murray, and D.-L. Tang, "Codesign of architectures for powertrain modules," *IEEE Micro*, Vol.14, No. 4, pp.48-58, Aug., 1994.

[4] R. K. Gupta, C. Coehlo, and G. De Micheli, "Synthesis and Simulation of Digital Systems Containing Interacting Hardware and Software Component," 29th *ACM, IEEE Design Automation Conference*, pp.225 230, 1992.

[5] Huiqun Liu. D. F., "Integrated Partitioning and Scheduling for Hardware/Software Co-design," *IEEE Proc. of Int'l Conference on Computer Design : VLSI in Computers and processors*, pp.609-614, 1998.

[6] F. Rousseau, J. Benzakki, J-M. Berge, M. Israel, "Adaptation of Force-Directed Scheduling Algorithm for Hardware/Software Partitioning," Rapid System Prototyping, 1995.

[7] Jinhwan Jeon, Kiyoung Choi, "An Effective Force-Directed Partitioning Algorithm for Hardware-Software Codesign," *on TR report, SNU*, May, 1997.

[8] P. G. PAULIN, JOHN P. KNIGHT, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's," *IEEE Transactions on CAD/ICAS*, Vol.CAD-8, No.6, pp.661-679, July, 1989.

[9] 오주영, 한갑수, 박도순, "하드웨어 소프트웨어 분할을 위한 ILP 구현", *한국정보과학회 추계 학술발표논문집, 제27-권 제2호*, pp.21-23, Oct., 2000.

[10] Hidalgo, J. L. Lanchares, J., "Functional partitioning of hardware-software codesign using genetic algorithms," *Proceedings of the 23rd EUROMICRO conference*, pp.631-638, 1997.

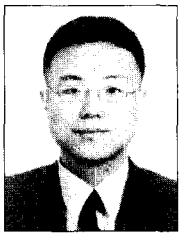
[11] Chuck Monahan, Forrest Brewer, "Scheduling and binding bounds for RT-level symbolic execution," *International Conference on Computer-Aided Design*, pp.230-235, Nov., 1997.

[12] J. A. Maestro, "New methodologies for Hardware-Software Codesign Partitioning to Avoid High Communication Overhead," Departmento de informatica y Automatica Universidad complutense de Madrid, 1997.

[13] K. S. Hwang, Albert E. casavant, Ching-Teng Chang, "Scheduling and hardware sharing in pipelined data paths," *In Proceedings of the IEEE International Conference on CAD*, pp.24-27, Nov., 1989.

[14] Samit Chaudhuri, Stephen A. Blythe, R. A. Walker, "A solution methodology for exact design space exploration in a 3D Design Space," *In IEEE Trans. on VLSI systems*, Vol.5, No.1, pp.1-13, March, 1997.

[15] Ganesh Krishnamoorthy, John A. Nestor, "Data Path Allocation using an Extended Binding Model," *29th ACM/IEEE Design Automation Conference*, pp.279-284, 1992.



오 주 영

e-mail : odid080@kic.ac.kr

1996년 서울산업대학교 전자계산학과(학사)

1998년 홍익대학교 대학원 전자계산학과 (석사)

2002년 현재 홍익대학교 대학원 전자계산학과 박사과정수료

1998년 ETRI 위촉연구원

2000년 (주)eKalos 선임연구원

2001년 (주)참좋은 인터넷 책임연구원

2002년~현재 경인여자대학 전임교수

관심분야 : 통합설계, 상위수준합성

이 면 재

e-mail : mjlee@cs.hongik.ac.kr

1992년 홍익대학교 전자계산학과 졸업 (학사)

1994년 홍익대학교 대학원 전자계산학과 졸업(석사)

1994년~1999년 정원엔 시스템 연구소

2000년~현재 용인 송담대학 겸임교수

2000년~2001년 코리아 홈넷 수석연구원

2002년~현재 아라마루 수석연구원, 홍익대학교 대학원 전자계산학과 박사과정수료

관심분야 : 멀티미디어 통신, 설계자동화

이 준 용

e-mail : jlee@cs.hongik.ac.kr

1986년 서울대학교 공과대학 컴퓨터공학과 졸업(학사)

1988년 미국 University of Minnesota 대학원, Dept. of Computer Sci. & Eng.(석사)

1996년 미국 University of Minnesota 대학원, Dept. of Computer Sci. & Eng.(공학박사)

1996년~1997년 미국 IBM Staff 연구원

1997년~현재 홍익대학교 컴퓨터공학과 교수

관심분야 : VLSI Design, CAD, 컴퓨터구조

박 도 순

e-mail : dspark@cs.hongik.ac.kr

1978년 서울대학교 공과대학 전자공학과 졸업(학사)

1980년 한국과학기술원 전산학과 졸업 (석사)

1988년 고려대학교 수학과 졸업(박사)

1980년~1983년 국방과학연구소 연구원

1983년~현재 홍익대학교 컴퓨터공학과 교수

관심분야 : 컴퓨터구조, 설계자동화