2001년 한국정보처리학회 추계 학술발표논문집 제8권 제2호

하드웨어 소프트웨어 통합설계에서의 개선된 분할 알고리즘

오 주 영 *
* 홍익대학교 컴퓨터 공학과
email : jyoh@cs.hongik.ac.kr

An Improved Partitioning Algorithm in Hardware Software Codeisgn

Ju-Young Oh *
* Dept. of Computer Engineering, Hongik University

요 약

본 논문에서는 주어진 제약조건을 만족하며 저비용 고효율의 목적물 합성을 위하여 어느 부분을 하드웨어로 또는 소프트웨어로 구현할 것인지를 결정하는 분할 알고리즘을 제안한다. 논문[6]에서 제시한 시뮬레이티드 어닐링의 후보자 선택은 랜덤한 방식에 의해 노드의 이동이 이루어지기 때문에 중복된 후보자의 선택으로 인하여 시간이 오래 걸리는 단점이 있다. 이러한 단점을 극복하기 위해, 본 논문에서는 비용 함수를 구성하는 변수들 중에서 시스템 실행시간과 구현 비용에 영향을 미칠 수 있는 부분들을 고려해 후보자를 선택하도록 하여 최적해 탐색을 위한 분할 알고리즘의 실행시간을 단축시켰다. 실험 결과는 대상 노드가 많아질수록 기존의 방법보다 빠른 시간에 최적의 해를 탐색한다.

1. 서론

시스템 수준의 합성에서 설계 비용, 효율성 및 시스템 성능을 가늠하는 분할 알고리즘은 분할 과정의 자동화 정도, 시스템 환경과 목적함수, 분할 대상 노드의 크기에 따라 다양한 알고리즘들[1]이 개발되었다. Gupta와 De Michieli[2]는 처음에 모든 기능을 하드웨어에 매핑해 놓고 전체 시스템의 성능을 만족하는 한 소프트웨어로 기능들을 옮김으로써 비용을 개선해 나간다. 반면, Cosyma시스템[3]은 처음에 모든 기능을 소프트웨어로 매핑해 놓고 비용 제약 조건만 만족하면 가능한 한 하드웨어로 기능들을 옮김으로써 시스템의 성능을 최대화한다. Kalavade[4]는 두 가지 목적함수를 정의하여 하드웨어 자원과 실행시간의 최적화를 행하는데, 목적함수의 변경은 분할되는 노드들을 스케줄 하여 제약시간과의 비교에 의해 얻어지는 전역시간에 의한다. 즉 전역시간이 임계점에 도달하면 목적함수는 실행시간의 감소가 되며, 그렇지 않은 경우에는 하드웨어 자원의 감소가 목적함수가 되도록 하여 분할을 한다. Choi[5]는 힘값을 응용하여 하드웨어로 매핑할 노드를 선택할 경우 소프트웨어 노드와의 병렬실행 가능성을 힘값으로 하여 분할을 수행하였다. Peng[6]은 local minima를 극복하기 위해 시뮬레이티드어닐링에 기반을 둔 휴리스틱 알고리즘을 구현하였다. 그러나, 분할 노드를 랜덤하게 생성하므로 중복된 선택이 많아짐으로 인해 해를 찾기까지 많은 실행시간이 소요된다. 본 논문에서는 노드 선택을 위한 알고리즘의 실행 시간을 단축하기 위해 비용 함수를 구성하는 변수의 특성을 고려해서 이웃해의 후보자를 선택하는 방법을 제안한다. 2절에서 알고리즘 실행을 위한 비용함수를 3절에서 비용함수를 고려한 이웃해 선택 방법을 4절에서 실험 및 결과를 5절에서 결론을 각각 기술하였다.

2. 비용함수

분할을 위한 환경변수는 논문[6]과 동일하며, 입력 그래프는 프로세서와 프로세서간의 상호작용을 표현한 노드와 에지로 구성되는 프로세스 그래프이다. 각 노드는 연산의 수, 연산 종류의 수, 연산의 종류별 개수 등의 정보를 가지고 실질적으로 연산을 수행하는 부분이며 에지는 노드사이의 직접적인 통신을 나타낸다. 프로세서 그래프에서 각 노드 i에 할당되는 변수는 식 (1)과 (2)이다. 식 (1)은 전체 노드들 중에서 시스템에 부과하는 연산의 비중이다. 식 (2)를 구성하는 각각의 M은 설계자에 의해 부여되는 가중치이고, 변수 K는 다른 노드에 대한 해당 노드의 상대적 연산의 비중, 노드 i내에서의 연산의

균일성 정도, 노드 i내의 연산의 특징을 고려한 소프트웨어로의 구현의 적합성이다.

$$W1_i^N \quad (1)$$

$$W2_i^N = M^{CL} \times K_i^{CL} + M^U \times K_i^U + M^P \times K_i^P - M^{SO} \times K_i^{SO} \quad (2)$$

노드 i와 j를 연결하는 에지에 할당되는 값은 프로세스간의 통신량에 따라 할당되며 식(3)과 (4)의 변수로 정의된다. Ch_{ij}는 노드 i와 j간의 통신 채널의 집합이고, wd_{ck}는 채널을 통해 전송되는 비트수이며, CI_{ck}는 채널 ck에서의 통신량을 나타낸다.

$$W1_{ij}^E = \sum_{c_k \in Ch_{ij}} wd_{ck} \times CI_{ck} \quad (3)$$

$$W2_{ij}^E = \sum_{c_k \in Ch_{ij}} CI_{ck} \quad (4)$$

각각의 변수에 의해 정의되는 비용함수는 식(5)와 같다.

$$C(HW,SW) = Q1 \times \sum_{(ij)\in cut} W1_{ij}^E + Q2 \times \frac{\sum_{(i)\in HW} \frac{\sum_{\exists(ij)} W2_{ij}^E}{W1_i^N}}{N_H} - Q3 \times \left[\frac{\sum_{i\in HW} W2_i^N}{N_H} - \frac{\sum_{i\in SW} W2_i^N}{N_S} \right] \quad (5)$$

비용함수의 첫 항은 하드웨어와 소프트웨어간의 통신 양을 나타내는데, 이 값의 감소는 두 개의 분할 사이의 통신량의 감소와 하드웨어와 소프트웨어로 분할된 노드간의 병렬성이 증가될 수 있음을 의미한다. 두 번째 항은 노드의 계산량이 다른 노드에 비해 많지만 해당 노드와 연결된 통신량이 상대적으로 적은 경우에 값이 작아지므로, 값이 작은 노드를 하드웨어로 분할함으로써 하드웨어에 분할된 프로세스들 사이의 병렬성을 향상시킬 수 있도록 한다. 세 번째 항은 계산량이 많은 노드는 하드웨어에 매핑시키고, 계산량이 적은 노드는 소프트웨어로 매핑시켜서 두 개의 분할이 가지는 노드의 평균 계산량의 차를 크게 하는 것으로써 실행 시간비중이 큰 노드가 하드웨어에 매핑될 수 있도록 한다. 비용 함수에서 Q는 타겟 아키텍처의 특징에 따라 조정될 수 있다. 분할의 목적은 식(5)의 비용 함수의 값이 최소화 되도록 하는 것이다.

3. 분할 알고리즘

본 논문에서는 하드웨어와 소프트웨어 분할의 해를 찾기 위해 논문[6]의 시뮬레이티드어닐링 방법을 사용하였으며 알고리즘은 그림[1]과 같다.

```
step 1. Construct initial configuration x^now := (HW_0, SW_0)
step 2. Initial Temperature T := TI
step 3.
 3.1. for i := 1 to TL do
  Generate randomly a neighboring solution x' ∈ N(x^now)
  Compute change of cost function ΔC := C(x') − C(x^now)
  if ΔC ≤ 0 then x^now := x'
   else
    Generate q := random(0,1)
    if q < e^{−ΔC/T} then x^now := x'
 3.2. Set new temperature T := a * T
step 4. if stopping criterion not met then goto step3
  return solution corresponding to the minimum cost function
```

그림 [1] 분할 알고리즘

알고리즘에서 x는 HW와 SW집합으로 구성되는 하나의 해를, x^{now}는 현재의 해, 그리고, $N(x^{now})$는 현재 해의 영역에서 x^{now}의 이웃해를 의미한다. TI는 초기 온도를, TL은 온도의 길이를 의미한다. a는 냉각률이며, 알고리즘의 종료조건은 3번의 연속적인 온도동안 새로운 해를 찾을 수 없는 경우로 한다. 논문[6]의 방법은 알고리즘의 단계 3.1과 같이 현재의 해인 x^{now}의 이웃 해인 $N(x^{now})$에서 새로운 해인 x'을 찾을 때 랜덤한 방식으로 찾게 되므로 탐색했던 해를 중복 방문함으로써 해를 찾는데 많은 시간이 소요된다.

```
function new_solution( x^now)
진의 x' 구성이 sw에 있던 hw로 옮겼었다면 goto 단계 2
단계1. if(node in SW)
 if(HW area 만족)
  if( W2_i^N의 값이 같다면) W'1_i^N(CL) 큰 것을 HW로 이동
   else if( W2_i^N의 값이 비슷할 때)
     노드의 에지의 개수가 작은 것을 이동
   else
     전에 이동된 노드를 제외한 노드중에서
     W2_i^N가 큰 것부터 HW로 이동
     return x', break ;
단계2. else if(node in HW)
 if( W2_i^N의 값이 같다면) W'1_i^N(CL)작은 것을 SW로 이동
  else if( W2_i^N의 값이 비슷할 때)
    노드의 에지의 개수가 작은 것을 이동
  else
    전에 이동된 노드를 제외한 노드중에서
    W2_i^N가 작은 것부터 SW로 이동
    return x' ;
end function ;
```

그림 [2] 이웃해 선택방법

다음의 해를 효율적으로 찾기 위해 본 논문에서 제안하는 비용 함수의 특성을 고려한 이웃해 선택방법은 그림 [2]와 같다. 단계1에서는 소프트웨어에 있는 노드를 선택해 하드웨어로 이동한다. 하드웨어 면적

제약조건을 만족시킬 때 노드의 $W2_i^N$값이 같다면 $W1_i^N$의 값이 큰 노드를 선택해 하드웨어로 이동시키고, $W2_i^N$의 값이 비슷하면 하드웨어와 소프트웨어간의 통신량을 줄이기 위해서 노드에 연결된 에지의 수가 작은 것을 선택해 이동시킨다. 위의 두 조건에 해당되지 않을 경우 반복을 피하기 위해서 전 단계에서 이동된 노드를 제외한 노드중에서 $W2_i^N$가 큰 것부터 선택해 하드웨어로 옮김으로써 실행시간을 줄일수 있도록 한다. 단계 1이 끝나면 현재의 해와 새로 생성된 이웃 해의 비용 함수를 비교하고, 좋은 해를 찾지 못하였을 경우에 단계 2를 수행하게 된다. 단계 2에서는 하드웨어에 있는 노드를 선택해 소프트웨어로 이동시키는데, 노드의 $W2_i^N$값이 같다면 $W1_i^N$의 값이 작은 노드를 선택해 소프트웨어로 이동시키고, $W2_i^N$의 값이 비슷하면 하드웨어와 소프트웨어간의 통신량을 줄이기 위해서 노드에 연결된 에지의 수가 작은 것을 선택해 이동시킨다. 위의 두 조건에 해당되지 않을 경우에는 반복을 피하기 위해서 전 단계에서 이동된 노드중에서 $W2_i^N$가 작은 것부터 선택해 소프트웨어로 옮김으로써 하드웨어 비용을 줄일수 있도록 한다. 그림[3]은 시뮬레이티드어닐링 알고리즘에서 어떤 동일한 온도의 시점에서 비용 함수를 고려해서 이웃해를 찾는 과정이다. 입력 노드 $N1, N2, N3, N4, N5, N6$의 $W2_i^N$의 값의 크기가 $N1 > N2 > N3 > N4 > N5 > N6$라고 가정하고, 하드웨어 면적제약은 4로 가정한다. 하나의 노드가 하드웨어로 이동을 할 때는 하드웨어 면적 1을 사용한다고 가정하고 소프트웨어는 면적제약 조건이 없다고 가정한다.

		HW	SW
X^{now}		1 2 3	4 5 6
X'	A	1 2 3 ④	5 6
	B	1 2 4	③ 5 6
	B'	1 3 4	② 5 6
	B''	2 3 4	① 5 6

그림[3] 이웃해의 생성과정

현재의 해에서 이웃 해를 찾을 때 그림[3]의 A는 소프트웨어에서 $W2_i^N$의 값이 가장 큰 노드를 선택해서 하드웨어로 이동한 후 생성된 이웃해를 현재해의 비용과 비교하여 우위인 경우 이웃해를 현재해로 바꾼다. 다음 이웃해를 찾을 때에는 생성된 새로운 현재의 해로부터 이웃해를 찾게 된다. B는 현재의 해인 A를 기준으로 하드웨어로부터 이동된 이전 노드를 제외한 노드 중에서 $W2_i^N$의 값이 가장 작은 노드를 선택해서 소프트웨어로 이동한 상태이다. A와 B의 비용을 비교해서 좋은 해를 찾지 못할 경우에는 현재의 해를 기준으로 위와 같은 방법으로 B', B''을 반복한다.

4. 실험 및 결과

실험을 위한 노드의 개수, 입력 노드의 값 $W1_i^N$, $W2_i^N$와 에지의 개수는 랜덤하게 설정되고 하나의 노드의 하드웨어 면적은 1이라고 가정한다. 그림 [4]와 같은 입력그래프에 대해 랜덤한 방식의 시뮬레이티드어닐링과 본 논문에서 제안한 후보 선택 방법에 의한 알고리즘의 실험 결과는 그림 [5]와 같다. 실험을 위해 $W2_i^N$의 값은 1에서 100까지의 수중에서 랜덤하게 선택하였다. 에지의 개수는 전체 노드를 n이라고 하였을 때 최대 n(n-1)개 중에서 최대 30%를 랜덤하게 생성하였다.

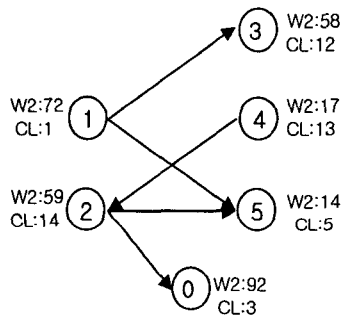

그림 [4] 입력 그래프

최적 값	논문[18]	제안알고리즘
1 loop : 2 loop : 3 loop :	1th iteration current cost= -13.678571 2th iteration current cost= -5.257144 : 16th iteration current cost= -51.214286 : 3834th iteration current cost= -51.214286 3095th iteration current cost= -47.000000 4051th iteration current cost= -33.178571 4218th iteration current cost= 52.000000	1th iteration current cost= -51.214286 2th iteration current cost= -41.333333 3th iteration current cost= -19.730769 : 64th iteration current cost= -52.214286 : 98th iteration current cost= 46.073260 132th iteration current cost= -13.678571 165th iteration current cost= -5.257144
time(sec)	2.970000 sec	0.490000 sec
cost=-51.214286	cost=-51.214286	cost=-51.214286

그림 [5] 알고리즘의 실행결과

그림 [5]에서 시뮬레이티드어닐링 알고리즘은 16번째에서 최적의 해를 찾을 수는 있지만 3834번째에서의 값보다 좋은 값을 1번째, 2번째, 3번째의 연속되는 에퍽 구간 동안에도 찾을 수 없을 때 알고리즘이 종료되어 4218번째에서 끝나게 된다. 제안 알고리즘에서는 후보자를 선택할 때 비용 함수를 고려해 노

드를 이동함으로써 1번째에서 최적의 해를 찾지만 64번째에서 다시 최적의 해를 찾은 후에 1번째, 2번째, 3번째 연속되는 에퍽 구간에도 이보다 더 좋은 값을 찾을 수 없어서 알고리즘이 종료되어 165번째에서 끝나게 된다. 제안방법에 의한 시간 측정 결과는 기존의 방법보다 효과적이며, 그림 [6]과 같이 노드의 개수가 많아져도 최적의 해를 찾으면서 실행시간을 단축시킨다.

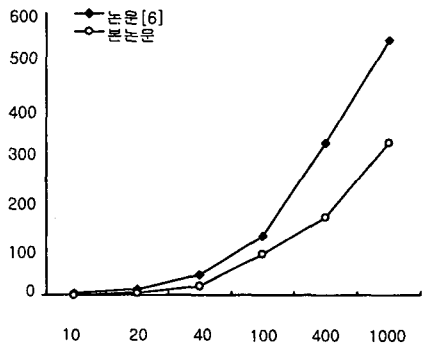

그림 [6] 노드의 개수에 따른 실험결과

5. 결론

본 논문에서는 하드웨어와 소프트웨어 분할을 위해 제안된 논문[6]의 시뮬레이티드어닐링 방법에서 이웃해를 랜덤하게 탐색함으로 인해 소요되던 알고리즘의 지연시간을, 비용 함수를 구성하는 변수에 기반하여 이웃해를 생성함으로써 반복 탐색 횟수를 줄이고 최적의 가능성이 높은 해를 우선 탐색함으로써 분할을 수행하는 시간을 단축시켰다. 향후 연구과제는 분할단계에서 스케줄을 함께 고려할 수 있도록 하며 다양한 목적 시스템에 적용될 수 있는 알고리즘 개발을 연구과제로 한다.

참고문헌

[1] Giovanni De Micheli, Mariagiovanna Sami, ' Hardware/Software Co-Design.' Kluwer Academic Publishers, 1997.

[2] R. K., Gupta and G. D Micheli, "Specification and analysis of timing constraints for embedded systems," IEEE Trans. on CAD of IC and Systems, Vol, 16, pp.240-256, March. 1997.

[3] J. Henkel, R. Ernst, U. Holtmann, and T. Benner, "Adaptation of partitioning and high-level synthesis in hardware/software co-synthesis," Proc. of ICCAD, pp.96-100, IEEE Computer Society Press, 1994.

[4] 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.

[5] Jinhwan Jeon, Kiyoung Choi, "An Effective Force-Directed Partitioning Algorithm for Hardware-Software Codesign," on TR report, School of Electrical Engineering, Seoul National Univ. May 30, 1997

[6] J.A. Maestro, "New methodologies for Hardware-Software Codesign Partitioning to Avoid High Communication Overhead", Departmento de informatica y Automatica Universidad complutense de Madrid, 1994.