

論文91-28A-12-11

하한 비용 추정에 바탕을 둔 최적 스케줄링 기법

(An Optimal Scheduling Method based upon the Lower Bound Cost Estimation)

嚴聖鉢*, 全洲植*

(Seong Yong Ohm and Chu Shik Jhon)

要 約

본 논문에서는 상위 단계 합성에서 발생하는 스케줄링 문제를 해결하기 위한 새로운 알고리듬에 대해 설명한다. 이 알고리듬은 ASAP 스케줄링 결과로부터 시작하여 선택된 노드를 나중 세이스텝으로 미루는 일련의 재스케줄링 과정을 통해, 주어진 시간 제약 하에서 최소의 면적 비용을 갖는 스케줄링 결과를 branch-and-bound 방식으로 찾는다. 이 방법에서는 면적 비용의 결정에 영향을 미치는 노드들에 대해서만 재스케줄링을 고려하는 branch 기법과 각 스케줄링 결과에 대해 추정한 하한 비용(lower bound cost)을 이용하여 불필요한 탐색을 줄이는 bound 기법을 사용한다. 이러한 branch-and-bound 기법은 많은 경우 탐색 공간을 효과적으로 줄임으로써 결과적으로 알고리듬의 수행 속도를 대폭 향상시킨다.

Abstract

This paper presents a new approach to the scheduling problem in the high level synthesis. In this approach, iterative rescheduling processes starting with ASAP (As Soon As Possible) scheduling result are performed in a branch-and-bound manner so as to arrive at the scheduling result of the lowest hardware cost under the given timing constraint. At each iteration step, only the selected nodes are considered for rescheduling, and the lower bound cost estimation is performed to avoid the unnecessary attempts to search for an optimal result. This branch-and-bound method turns out to be effective in pruning the search space, and thus reducing run time considerably in many cases.

I. 서 론

설계하고자 하는 하드웨어 시스템의 기능이 다양해지고 복잡해짐에 따라, 하드웨어 시스템의 기능을 상위 단계의 하드웨어 설계 언어(hardware description language)로 기술하고 이로부터 기술된 기능을 효율

적으로 수행하는 논리 회로를 자동 합성하는 상위 단계 합성(high level synthesis)^[1,2]에 대한 필요성이 더욱 높아지고 있다. 상위 단계 합성의 주요 과정 중의 하나는 주어진 상위 단계 설계 정보로부터 ALU나 곱셈기 등과 같은 기능 단위(function unit)들과 입력 및 중간 연산 결과를 저장하는 레지스터들, 그리고 기능 단위나 레지스터를 공유하기 위해 필요한 MUX(multiplexor)들과 이들간의 데이터 전달을 위한 연결 라인 등의 하드웨어 원소들로 구성되는 데이터부(data path)를 합성하는 과정이다. 대부분

*正會員, 서울大學校 컴퓨터工學科
(Dept. of Comp. Eng., Seoul Nat'l Univ.)
接受日字: 1991年 11月 5日

의 데이터부 합성 방법에서는 처리의 효율을 높이기 위해 상위 단계 설계 정보로부터 그래프 형태의 정보^[1,2]를 형성하고, 이로부터 데이터부 합성을 수행해 나간다. 그래프 형태로는 수행할 연산을 의미하는 노드들과 노드(연산) 간의 데이터 의존 관계(또는 수행 우선 관계)를 나타내는 방향성 아크들로 구성되는 데이터 의존 그래프(DDG : Data Dependency Graph)가 주로 사용된다. 데이터 의존 그래프에서 노드 i로부터 노드 j로 연结되는 아크, 즉 데이터 의존 관계는 노드 i가 의미하는 연산의 출력으로 사용됨을 의미한다. 따라서 기술된 기능이 정확히 수행되기 위해서는 임의의 한 노드 i에 대하여 의존하는 노드 j는 반드시 노드 i가 수행을 마친 후에 수행을 시작해야 한다. 그림 1은 2차 미분 방정식 $y'' + 3xy' + 3y = 0$ 을 계산하는 하드웨어 시스템의 기능을 기술하는 상위 단계의 설계 정보와 이에 대한 데이터 의존 그래프의 예를 보여준다. 예로서, 노드 11의 빼셈 연산은 노드 6의 곱셈 연산이 수행된 후에 수행을 시작해야 하며, 노드 9의 비교 연산의 수행은 반드시 노드 5의 덧셈 연산의 수행 후에 이루어져야 한다.

그래프가 형성되면 이로부터 데이터 의존 관계가 모두 유지되는 한도 내에서 요구되는 전체 세이스텝(control step)의 수와 하드웨어 원소의 비용이 최소인 논리 회로를 합성하는 작업이 이루어진다. 여기서 세이스텝이란 하나의 클럭 주기(clock cycle)를 의미하며, 하드웨어 원소의 비용이란 합성에 필요한 가능한 단위와 레지스터, 그리고 MUX 등과 같은 논리 블럭들과 이들 간의 정보 전달을 담당하는 연결 라인들이 차지할 면적 비용의 합을 의미한다.

데이터부의 최소화는 대부분의 경우^[1,2] 각 연산을 어느 세이스텝에서 수행할지를 결정하는 스케줄링(scheduling) 과정과 각 연산이 수행될 기능 단위의 종류를 결정하고, 또한 같은 종류의 기능 단위가 여러 개 사용될 경우 구체적으로 어느 기능 단위, 즉 어느 인스턴스(instance)에 의해 수행될지를 결정하는 하드웨어 할당(hardware allocation) 과정에 의해 이루어진다. 이 두 과정은 서로 상충되는 목표를 갖기 때문에 궁극적으로 가장 좋은 결과를 얻기 위해서는 두 과정이 동시에 수행되어야 한다^[1,4]. 하지만 두 과정을 동시에 수행할 경우 야기되는 높은 시간 복잡도를 줄이기 위해, 대부분의 시스템들^[5,6,7,8]은 두 과정을 분리 수행하거나, 각 과정을 순차적으로 반복 적용하는 방식을 채택하고 있다. 스케줄링 과정에서는 주어진 제약 조건 범위 내에서 노드(연산)들의 순차

$$y'' + 3xy' + 3y = 0$$

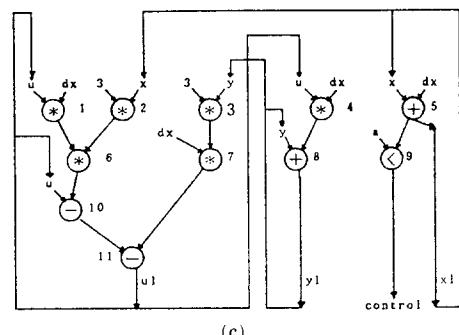
(a)

while ($x < a$)

```

    x1 = x + dx;
    u1 = u - (3*x*u*dx) - (3*y*dx);
    y1 = y + (u*dx);
    x = x1; u = u1; y = y1;
  
```

(b)



(c)

그림 1. 2차 미분 방정식 예에 대한 상위 단계의 설계 정보 및 이에 대한 데이터 의존 그래프의 예

(a) 2차 미분 방정식의 예

(b) 그림 (a)에 대한 상위 단계 설계 정보
(c) 데이터 의존 그래프

Fig. 1. A high level description for “2nd order differential equation” example and its corresponding data dependency graph.

(a) A 2nd order differential equation,
(b) High level description for the example in Fig. (a),
(c) A data dependency graph.

또는 병렬 수행을 결정하는데, 이는 면적 비용과 수행 시간 간의 결충점(trade off)을 구하는 것으로 데이터 합성 결과의 효율 결정에 있어 매우 중요한 역할을 차지한다.

가장 간단한 스케줄링 기법 중의 하나는 ASAP(as soon as possible) 스케줄링 기법^[1,9]으로서 데이터 의존 그래프의 각 노드를 처음 세이스텝부터 차례로 할당하는 방법이다. 여기서 각 노드는 자신에 입력되는 아크가 하나도 없거나, 또는 그러한 아크에 연결된 노드들(즉, predecessor)이 모두 이미 이전의 세이스텝에 스케줄링된 경우에만 스케줄링될 수 있다. 또 하나의 간단한 방법은 ALAP(as late as possible) 스케줄링^[10]으로서, 이 방법에서는 ASAP 스케줄링과

는 반대로 각 노드를 마지막 세이스텝부터 차례로 할당해 나간다. 이 경우, 각 노드는 자신으로부터 나가는 아크가 없거나, 또는 그러한 아크들에 연결된 노드들(즉 successors)이 이미 다른 세이스텝에 모두 할당된 경우에만 스케줄링될 수 있다. 이 두 방법은 매우 간단하고 또한 회로의 가장 빠른 수행 속도를 보장하지만, 많은 경우 불필요하게 많은 기능 단위를 요구하게 된다. 그림 2의 (a)와 (b)는 그림 1의 그래프에 대해 ASAP 및 ALAP 스케줄링을 수행한 결과를 각각 보여준다. 이 예에서 곱셈 연산은 곱셈기에 의해, 그리고 덧셈, 뺄셈, 및 비교 연산은 ALU에 의해 수행된다. ASAP 스케줄링 결과의 합성에 필요한 기능 단위는 곱셈기 4개와 ALU 2개인 반면, ALAP 스케줄링 결과의 합성을 위해서는 곱셈기 2개와 ALU 3개가 필요하다. 그런데 이러한 결과는 동일한 세이스텝 수에 대해 수행한 최적 스케줄링 결과보다 많은 수의 기능 단위를 필요로 한다. 그림 2의 (c)는 최적 스케줄링 결과의 한 예로서 곱셈기 2개와 ALU 2개만을 필요로 한다.

리스트 스케줄링(list scheduling)^[5,7,8,11]은 ASAP 및 ALAP 스케줄링의 단점을 극복하기 위해 사용 가능한 기능 단위의 수를 각 연산 종류별로 제한한다. 이 방법에서는 ASAP 스케줄링에서와 같이 처음 세이스텝부터 시작하여 스케줄링 가능한 노드들을 하나씩

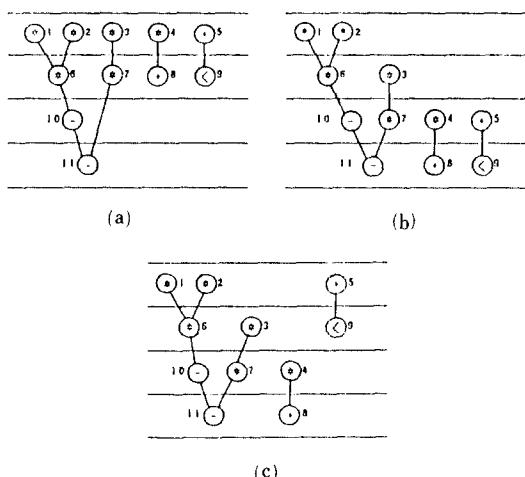


그림 2. 그림 1의 예에 대한 ASAP 및 ALAP, 그리고 최적 스케줄링 결과의 예

- (a) ASAP 스케줄링
- (b) ALAP 스케줄링 결과
- (c) 최적 스케줄링 결과의 예

Fig. 2. (a)ASAP, (b) ALAP, and (c) an optimal scheduling result for the example in Fig. 1.

스케줄링해 나가는데, 만일 각 연산 종류별로 한 세이스텝에 스케줄링 가능한 노드들이 허용된 기능 단위의 개수보다 많은 경우, 이를 가운데 먼저(또는 나중에) 스케줄링할 노드들을 우선 순위(priority)에 따라 선택한다. 기존의 리스트 스케줄링 기법에서 사용된 우선 함수의 예로는 MAHA^[5]에서 사용한 freedom, SLICER^[7]에서 사용한 mobility, Elf^[8]에서 사용된 urgency, 그리고 FDLS^[11]에서 사용된 force^[6,11] 등이 있다.

HAL^[6]에서 사용된 FDS(force directed scheduling) 방식^[6,12]은 이전의 스케줄링 기법과는 달리, 전체 세이스텝 수에 대한 제약이 주어지면, 노드들을 각 연산 종류별로 전체 세이스텝에 걸고부분산시킴으로써 합성에 필요한 기능 단위의 수가 최소화되도록 한다. 이 방법에서는 가장 먼저 스케줄링할 노드와 그 노드가 할당될 세이스텝을 결정하기 위해 force라고 불리우는 우선 함수를 사용한다. Force값은 그 값이 낮을수록 동일 종류의 기능 단위를 사용할 노드들이 고르게 분포되었음을 나타내므로, 선택 가능한 노드-세이스텝쌍 가운데 가장 작은 force 값을 갖는 쌍을 우선적으로 선택하여 스케줄링 한다.

앞에서 설명한 스케줄링 방법들은 비교적 빠르게 수행되지만 최적의 스케줄링 결과를 보장하지는 못한다. ALPS^[13]는 스케줄링 문제를 최소화할 면적 비용 함수와 시간 제약 조건들을 각각 목적 함수(objective function)와 제약 수식(constraint expression)들로 하는 ILP(integer linear programming) 문제로 바꾸고 이를 ILP용 패키지를 이용하여 풀다. 이 방법은 항상 최적의 결과를 산출하지만, 근본적으로 문제의 크기의 증가에 따라 수행 시간이 기하급수적으로 증가하는 문제점이 있다.

따라서 본 논문에서는 항상 최적의 스케줄링 결과를 보장하면서도, 대부분의 경우 빠른 시간 내에 결과를 산출하는 새로운 최적 스케줄링 기법을 제안한다. 그럼 먼저, 본 논문에서 고려할 문제의 범위 및 용어에 대해 설명한 다음, 새로운 최적 스케줄링 기법에 대해 설명한다. 그런 다음, 새로운 스케줄링 기법의 효율성을 보이기 위해 기존의 연구에서 사용한 대표적인 세가지 예에 대해 수행한 실험 결과를 기준 방법의 실현 결과와 비교한다. 그리고 마지막 절에서는 이 논문에 대한 결론 및 앞으로의 연구 방향을 제시한다.

II. 문제 범위 및 용어 정의

본 논문에서는 시간 비용에 대한 세약 조건으로서 전

체 세이스텝의 수와 각 종류별 가능 단위에 대한 수행 주기가 주어질 경우, 요구되는 가능 단위들이 차지하는 전체 면적 비용이 최소화되도록, 입력된 데이터의 존 그래프의 각 노드를 특정 세이스텝에 할당하는 스케줄링 문제를 다룬다. 여기서 전체 면적 비용 F 는 연산 종류 t 를 수행할 가능 단위의 면적을 A_t 라고 하고, 세이스텝 j 에 할당되어 있는 노드들 가운데 연산 종류가 t 인 노드들의 갯수를 N_{tj} 라고 표기할 때 다음의 식 (1)과 같이 정의된다.

$$F = \sum_t (A_t * M_t) \text{ where } M_t = \max_j (N_{tj}) \quad (1)$$

전체 세이스텝의 수와 각 종류별 가능 단위의 수행 주기가 주어지면, 데이터의 존 관계를 유지하는 가운데 각 노드가 스케줄링될 수 있는 세이스텝의 범위가 설정되는데, 이를 각 노드에 대한 할당 가능 구간(time frame)이라고 하고, 노드 n 의 할당 가능 구간을 $TF(n) = [S_n, L_n]$ 로 표기한다. 여기서 S_n 과 L_n 은 노드 n 이 스케줄링될 수 있는 최초 및 최후 세이스텝으로서 ASAP 및 ALAP 스케줄링에 의해 각각 구할 수 있다. 예를 들어, 그림 1의 예에서 전체 세이스텝의 수가 4이고, 모든 연산 종류의 수행 주기가 1이라고 가정하면, 그림 2의 ASAP 및 ALAP 스케줄링 결과로부터 그림 3과 같은 각 노드의 할당 가능 구간을 구할 수 있다. 그림 3에서 노드 1, 2, 6, 10, 11의 할당 가능 구간은 각각 $[1, 1], [1, 1], [2, 2], [3, 3], [4, 4]$ 로서 이를 노드들을 세이스텝을 선택할 여지가 없는 반면, 노드 3, 4, 5, 7, 8, 9의 할당 가능 구간은 각각 $[1, 2], [1, 3], [1, 3], [2, 3], [2, 4], [2, 4]$ 로서 두 가지 이상의 스케줄링 가능성을 갖는다.

| | | | | | | | |
|---|----|---|----|---|---|---|---|
| 1 | 1 | | 2 | 3 | 4 | | 5 |
| 2 | | 6 | 7 | | | 8 | 9 |
| 3 | 10 | | | | | | |
| 4 | | | 11 | | | | |

$$\begin{aligned} TF(1) &= [1, 1], \quad TF(2) = [1, 1], \quad TF(3) = [1, 2], \quad TF(4) = [1, 3] \\ TF(5) &= [1, 3], \quad TF(6) = [2, 2], \quad TF(7) = [2, 3], \quad TF(8) = [2, 4] \\ TF(9) &= [2, 4], \quad TF(10) = [3, 3], \quad TF(11) = [4, 4] \end{aligned}$$

그림 3. 그림 1의 예에 대한 각 노드의 할당 가능 구간들(전체 세이스텝의 수가 4인 경우)

Fig. 3. Time frames for the example in Fig. 1.
(when the total estep # is 4)

한편, 데이터의 존 그래프의 노드 집합을 V 라고 할 때, 임의의 스케줄링 결과는 다음과 같이 그래프의 각 노드와 그 노드가 할당된 세이스텝의 쌍으로 구성된 집합으로 표기될 수 있다.

$$S = \{(n, c) \mid n \in V \text{ and } S_n \leq c \leq L_n\}$$

본 논문에서는 이러한 집합을 해당 스케줄링 결과에 대한 상태(state)라고 정의한다. 연산 종류 t 를 수행할 가능 단위의 면적을 A_t 라고 하고, 상태 S 에서 세이스텝 j 에 할당되어 있는 노드들 가운데 연산 종류가 t 인 노드의 갯수를 N_{tj} 라고 표기할 때, 상태 S 의 면적 비용 $COST(S)$ 는 다음의 수식 (1')와 같이 표현된다.

$$\begin{aligned} COST(S) &= \sum_t (A_t * M_{st}) \text{ where } M_{st} = \\ &\quad \max_j (N_{tj}) \end{aligned} \quad (1')$$

본 논문에서는 최소의 면적 비용을 갖는 상태를 최적 상태(optimal state)라고 한다. 상태 S 에서 연산 종류 t 의 노드들이 가장 많이 할당된 세이스텝을 상태 S 에서의 연산 종류 t 에 대한 최대 세이스텝이라고 하고 MAX-CST_{st} 로 표기한다. 그리고 세이스텝 MAX-CST_{st} 에 할당되어 있는 노드들 가운데 연산 종류가 t 인 노드들의 집합을 상태 S 에서의 연산 종류 t 에 대한 BCS(branch candidate set)라고 정의하고, BCS_{st} 로 표기한다. BCS_{st} 의 크기, 즉 BCS_{st} 에 소속된 원소의 갯수는 수식 (1')의 M_{st} 를 결정하는 요소이다.

모든 노드 n 에 대해, $(n, C_s) \in S$ 이고 $(n, C_t) \in T$ 이면 반드시 $C_s \leq C_t$ 가 성립하는 경우, 상태 S 는 상태 T 로 전이 가능하다고 하고, “ $S \Rightarrow T$ ”로 표기한다. 특히 상태 S 가 임의의 최적 상태로 전이 가능한 경우, 상태 S 는 “admissible”하다고 한다.

본 논문에서는 주어진 상태 S 는 물론 그 상태로부터 전이 가능한 임의의 다른 상태 S' 가 가질 수 있는 면적 비용의 최소값보다 작거나 같은 값을 상태 S 에 대한 하한 비용(lower bound cost)이라고 정의한다. 즉 상태 S 에 대한 하한 비용을 $LB(S)$ 라고 표기할 때 다음의 두 수식이 성립한다.

$$LB(S) \leq COST(S) \text{ for each } S \quad (2)$$

$$LB(S) \leq COST(S') \text{ if } S \Rightarrow S' \quad (3)$$

III. 새로운 스케줄링 기법

1. 이론 배경

Admissible한 상태가 주어질 경우, 그 상태가 바로

최적 상태이거나, 아니면 그 상태에서 선택된 노드를 현재 그 노드가 할당된 제어스텝 이후의 제어스텝으로 재스케줄링 하는 과정을 반복함으로써 최적 상태로 전이 될 수 있다. 따라서 LBS 알고리듬에서는 다음의 정리에 의해 항상 admissible한 상태로 표현되는 ASAP 스케줄링 결과를 시작점으로 하여, 주어진 상태에서 선택된 노드를 나중 제어스텝으로 미루는 재스케줄링 과정을 통해 최종적으로 최적 상태를 찾는다.

[정리 1] ASAP 스케줄링 결과를 나타내는 상태 I 는 최적의 상태를 포함하는 임의의 상태로 전이 가능하다.

[증명] ASAP 스케줄링 결과를 나타내는 상태 I 와 최적 상태를 포함한 임의의 상태 T는 정의에 의해 각각 다음과 같이 표현된다.

$$I = \{(n, C_i) \mid n \in V \text{ and } C_i = S_n\}$$

$$T = \{(n, C_t) \mid n \in V \text{ and } S_n \leq C_t \leq L_n\}$$

여기서 C_i 는 S_n 과 같고 C_t 는 S_n 즉 C_i 보다 는 반드시 크거나 같다. 따라서 $(n, C_i) \in I$ 이고 $(n, C_t) \in T$ 인 임의의 노드 n에 대해 $C_i \leq C_t$ 가 성립한다. 그러므로 상태 I는 임의의 상태 T로 전이 가능하다.

하지만, 주어진 상태에 대해 어떤 노드를 어느 제어스텝으로 재스케줄링 하느냐에 따라 다양한 상태로의 전이가 가능하다. 따라서 주어진 상태로부터 전이 가능한 최적 상태를 찾기 위해서는 기본적으로는 모든 상태를 탐색해야 한다. 그러나, 많은 경우에 있어 최적 상태로의 전이가 불가능한 상태들을 미리 판별할 수 있기 때문에, 이러한 상태의 탐색은 미연에 막을 수 있다. 이에 본 논문에서는 주어진 상태로부터 전이 가능한 모든 상태를 탐색함에 있어, 불필요한 상태의 구성을 미연에 방지하는 방법으로서 branch-and-bound 기법을 활용한다.

본 논문에서는 구성 가능한 최적 상태를 모두 찾는 것이 아니라, 임의의 한 최적 상태를 찾는 것을 목표로 하기 때문에, 주어진 상태에 대해 궁극적으로 면적 비용의 감소에 영향을 주지 않는 노드의 재스케줄링은 고려할 필요가 없다. 따라서 임의의 상태가 주어질 경우, 다음의 정리에 의해, 면적 비용 감소에 영향을 미치는 노드들의 집합인 BCS를 각 연산 종류별로 구성하고 각 BCS에 속한 노드들에 대해서만 재스케줄링을 고려한다.

[정리 2] 주어진 상태 S가 실제 admissible하지만 최적 상태가 아닌 경우, 주어진 상태 S가 최적 상태로 전이되기 위해서는 상태 S에서 구성한 각 연산 종류별 BCS에 속한 노드 가운데 적어도 한 노드를 반드시 재스케줄링을 고려한다.

링시켜야 한다.

[증명] 만일 주어진 상태 S가 실제 admissible하지만 최적 상태가 아니라면, S는 보다 작은 면적 비용을 갖는 상태 T로 전이 가능하다. 즉 " $S \Rightarrow T$ "이고 " $\text{COST}(S) > \text{COST}(T)$ "인 최적 상태 T가 반드시 존재한다. 이는 수식 (1')에서 " $M_{st} > M_{Tt}$ "라는 수식이 성립하는 연산 종류 t가 적어도 하나 이상 존재한다는 뜻이다. 그런데 수식 (1')에서 M_{st} 를 결정짓는 것은 상태 S에서 제어스텝 MAX-CST_{st}에 대해 구성한 BCS_{st}에 소속된 노드들의 갯수이다. 따라서 수식 (1')의 BCS_{st}에 속한 노드를 증 적어도 하나 이상의 노드를 다른 제어스텝에 재스케줄링하지 않고서는 M_{st} 를 줄일 수 없다.

반면에, 주어진 상태가 최적 상태이거나 또는 admissible하지 않다고 판단될 경우, 그 상태로부터 전이 가능한 모든 상태는 구성해 볼 필요가 없다. 그러나, 일반적인 경우 최적 상태를 미리 알 수 없으므로 주어진 상태가 admissible한 지의 여부를 쉽게 판별할 수 없다. 따라서 본 논문에서는 어떤 상태가 주어질 경우, 그 상태가 결코 최적 상태로 전이되지 않은 상태, 즉 admissible하지 않은 상태인지를 판별하는 충분 조건과 주어진 상태가 최적 상태인지를 판별하는 충분 조건을 각각 설정한다. 이러한 조건의 설정은 주어진 상태 및 그 상태에서 전이 가능한 임의의 상태가 갖는 면적 비용에 대한 하한, 즉 주어진 상태에 대한 하한 비용을 추정함으로써 이루어진다. 이에 본 알고리듬을 하한 비용 추정에 의거한 스케줄링 알고리듬 또는 간략하게 LBS(lower bound directed scheduling) 알고리듬이라고 부른다. 주어진 상태 S에 대한 하한 비용 LB(S)는 수식 (2)와 (3)을 항상 만족하기 때문에 다음의 두 정리가 성립한다.

[정리 3] 주어진 상태 S의 면적 비용과 ASAP 스케줄링 결과를 나타내는 초기 상태 I에 대한 하한 비용이 동일한 경우, 즉 " $\text{COST}(S) = LB(I)$ "라는 등식이 성립하는 경우, 상태 S는 반드시 최적 상태이다.

[증명] 임의의 상태 K는 초기 상태 I로부터 전이 가능하다(즉 " $I \Rightarrow K$ "). 따라서 $\text{COST}(S) \geq LB(I)$ 와 같은 경우, 수식 (3)에 의해 " $\text{COST}(S) = LB(I) \leq \text{COST}(K)$ "라는 수식이 임의의 상태 K에 대해 성립한다. 그런데 이 수식은 상태 S의 면적 비용은 임의의 상태 K의 비용보다 작거나 같음을 의미한다. 즉 상태 S의 면적 비용

은 최소 비용이다. 따라서 “ $COST(S) = LB(1)$ ”이라는 조건이 성립하면 상태 S 는 최적 상태이다.

| 정리 4 | 현재까지 탐색된 상태 중 최소의 면적 비용을 갖는 상태를 Min-state라고 할 때, Min-state의 면적 비용이 주어진 상태 S 에 대한 하한 비용보다 작은 경우, 즉 “ $COST(\text{Min-state}) < LB(S)$ ”라는 조건이 성립할 경우, 상태 S 는 admissible 하지 않다.

[증명] “ $COST(\text{Min-state}) < LB(S)$ ”라는 조건이 성립할 경우, 수식 (2), (3)에 의해 상태 S 와 상태 S' 에서 전이 가능한 임의의 상태 S' 에 대해 다음의 두 수식이 항상 성립한다.

$$\begin{aligned} "COST(\text{Min-state}) < LB(S) \leq COST(S)" \\ "COST(\text{Min-state}) < LB(S) \leq COST(S')" \end{aligned}$$

즉 상태 S 및 S' 의 면적 비용은 이미 발견된 상태 Min-state의 비용보다 항상 크다. 이는 상태 S 는 상태 Min-state보다 작거나 같은 면적 비용을 갖는 임의의 상태로 전이될 수 없음을 뜻한다. 따라서 “ $COST(\text{Min-state}) < LB(S)$ ”라는 조건이 성립하면 상태 S 는 admissible 하지 않다.

그럼 먼저, 각 상태에 대한 하한 비용을 어떻게 추정하는지에 대해 설명한 다음, LBS 알고리듬의 개략적인 구조에 대해 설명한다.

2. 하한 비용 추정 알고리듬

하한 비용 추정 알고리듬의 목표는 주어진 상태 및 그 상태로부터 전이 가능한 상태들이 갖는 면적 비용 가운데 가장 작은 값과 동일한 값을 그 상태에 대한 하한 비용으로 추정하는 것이다. 즉 임의의 주어진 상태 S 에 대해 수식 (2)와 (3)을 만족하는 가장 큰 하한 비용 $LB(S)$ 를 추정하는 것을 궁극적인 목표로 한다. 주어진 상태를 비롯하여 그 상태로부터 전이 가능한 상태가 갖는 최소 비용을 알기 위해서는 기본적으로 해당하는 모든 상태를 구성해 보아야 한다. 하지만 이러한 작업은 대부분의 경우 많은 수행 시간을 요구한다. 따라서 본 논문에서는 항상 정화하지는 않더라도 많은 경우에 있어 최소 비용과 동일한 값을 하한 비용으로 추정해 내는 비교적 빠른 하한 비용 추정 기법을 세운다. 하한 비용의 추정 과정은 주어진 상태에서 각 연산 종류별 기능 단위의 사용 갯수에 대한 하한(이를 본 논문에서는 연산 종류별 하한 갯수라고 한다)을 추정하는 과정과 각 연산 종류별 하한 갯수를 바탕으로 주어진 상태는 물론 그 상태로부터

전이 가능한 임의의 다른 상태가 가질 수 있는 면적 비용에 대한 하한(즉 주어진 상태에 대한 하한 비용)을 추정하는 과정으로 나뉜다.

1) 각 연산 종류별 하한 갯수의 추정

주어진 상태에서 각 연산 종류별 기능 단위의 사용 갯수에 대한 하한을 추정하는 방법은 해당 연산 종류의 노드들이 주어진 세이스텝 범위 R 에 걸쳐 균등하게 분포될 경우 기능 단위의 사용이 최소화되며, 이 때 요구되는 기능 단위의 갯수는 범위 R 내에 소속된 해당 연산 종류의 노드 갯수를 N 이라고 할 때, $\lceil (N/R\text{의 크기}) \rceil$ 가 된다는 점에 착안을 두고 있다. 여기서 세이스텝 범위 R 에 소속된 노드란, 범위 R 내에 포함된 임의의 한 세이스텝에 현재 스케줄링되어 있는 노드 가운데 자신이 할당될 수 있는 마지막 세이스텝 역시 범위 R 내에 포함되는 노드로서, 상태 전이가 이루어지더라도 세이스텝 범위 R 내에 반드시 스케줄링될 노드를 의미하며, 기호 $\lceil \cdot \rceil$ 는 계산된 값보다 크거나 같은 최소의 정수를 뜻한다. 다시 말해, 세이스텝 범위 R 에 반드시 스케줄링되어야 할 해당 연산 종류의 노드들을 구현하기 위해서는 적어도 $\lceil (N/R\text{의 크기}) \rceil$ 개 이상의 해당 기능 단위가 필요하다. 이러한 측면에서 볼 때, 가장 간단한 하한 갯수 추정 방법은 입력 그래프에 있는 각 연산 종류별 노드의 갯수를 허용된 전체 세이스텝의 수로 나눈 값보다 크거나 같은 정수 가운데 가장 작은 정수를 해당 연산 종류에 대한 하한 갯수로 추정하는 것이다.

그러나 각 노드간에 존재하는 데이터 의존 관계 때문에, 대부분의 상태에서는 각 연산 종류별 노드들이 스케줄링될 가능성이 각 세이스텝에 걸쳐 균등하게 분포되지 못하는 경우가 매우 많다. 따라서 본 논문에서는 이러한 특성을 이용하여, 보다 큰 하한 갯수를 추정한다. 이를 위해 전체 세이스텝 (TC) 내에서 구성 가능한 임의의 세이스텝 범위 $R' = [s, d]$ ($s \geq 1, d \leq TC$)에 대해 $\lceil (범위 R'에 소속된 해당 연산 종류별 노드의 갯수) / (d - s + 1) \rceil$ 값을 계산하여 그 범위에 대한 각 연산 종류별 하한 갯수로 둔다. 주어진 상태 및 그 상태에서 전이 가능한 임의의 상태를 구현하는데 필요한 기능 단위의 최소 갯수는 각 범위에 대해 계산한 하한 갯수보다 적어도 같거나 커야 한다. 따라서 각 범위별로 추정된 하한 갯수 가운데 최대의 값을 주어진 상태에서의 해당 연산 종류에 대한 하한 갯수로 추정한다.

한편, 주어진 상태에서 각 연산 종류별로 볼 때 해당 종류의 노드들이 할당될 가능성이 전혀 없는 세이스텝이 존재할 수 있다. 이러한 세이스텝을 포함하는 세이스텝 범위에 대해서는 다음과 같은 정리가 성립

된다.

[정리5] 한 연산 종류 t 에 대해, 해당 연산 종류의 노드들이 할당될 가능성이 없는 세이스텝을 포함하는 임의의 범위 R 에 대해 추정한 하한 갯수는 그런 세이스텝을 제외하고 구성된 범위 R' 에 대해 추정한 하한 갯수보다 항상 작거나 같다.

[증명] 범위 R 및 R' 에 반드시 스케줄링되어야 할 연산 종류 t 의 노드 갯수를 각각 N, N' 라고 하자. 그런데 R 및 R' 의 정의에 의해 R 은 R' 에 비해 해당 연산 종류의 노드가 없는 세이스텝만을 더 포함하고 있다. 따라서 $N=N'$ 이 되며 동시에 “범위 R 의 크기 > 범위 R' 의 크기”가 성립한다. 따라서 $[(N/\text{범위 } R\text{의 크기})] \leq [(N'/\text{범위 } R'\text{의 크기})]$ ”가 성립한다. 즉 범위 R 에 추정한 하한 갯수는 그런 세이스텝을 제외하고 구성된 범위 R' 에 대해 추정한 하한 갯수보다 항상 작거나 같다.

따라서 본 논문에서는 알고리듬의 속도 향상을 위해 해당 연산 종류의 노드가 할당될 가능성이 없다고 판단되는 세이스텝을 포함하는 임의의 세이스텝 범위는 고려하지 않는다. 이를 위해 각 연산 종류에 대해, 해당 연산 종류의 노드가 할당될 수 있는 세이스텝들을 모두 추출하고, 추출된 세이스텝 가운데 연속된 세이스텝의 범위를 각각 구성한다. 이렇게 구성된 각 범위는 서로 겹치지 않으며, 각 노드는 반드시 한 범위에만 속한다. 연속된 세이스텝 범위가 구성되면, 연속된 각 세이스텝 범위에 대해 그 범위내에서 구성 가능한 임의의 세이스텝 범위에 대해 각 연산 종류별 하한 갯수를 추정한 뒤, 가장 큰 값을 주어진 상태에서의 해당 연산 종류에 대한 하한 갯수라고 결정한다.

2) 주어진 상태에 대한 하한 비용의 계산

주어진 상태 S 에 대한 각 연산 종류별 하한 갯수가 추정되면, 이를 해당 연산 종류(t)의 기능 단위 사용 갯수(즉, M_{st} 로) 할 때의 비용을 수식 (1')에 의거해 계산한다. 주어진 상태 S 에서 임의의 연산 종류 t 에 대해 추정된 하한 갯수는 M_{st} 가 가질 수 있는 최소 값보다 항상 작거나 같으므로 수식 (1')에 의거해 계산한 비용은 반드시 상태 S 의 면적 비용이나 상태 S 로부터 전이 가능한 임의의 상태 S' 의 면적 비용보다 작거나 같다. 따라서 이 비용을 주어진 상태 및 그 상태로부터 전이 가능한 상태가 가질 수 있는 면적 비용에 대한 하한, 즉 주어진 상태 S 에 대한 하한 비용으로 추정한다.

그림 4는 주어진 상태에 대한 하한 비용을 추정하는 알고리듬의 개관을 보여준다. 그림 4에서 LOCAL-LB()는 주어진 상태 S 에 대한 각 연산 종류별 하한 갯수를 추정하는 함수로서, 먼저 해당 연산 종류의 노드가 할당될 수 있는 세이스텝들 가운데 연속된 세이스텝 범위를 모두 찾는다. 그런 다음, 연속된 각 범위 내에서 구성 가능한 임의의 세이스텝 범위 R 에 대해, $[(\text{범위 } R\text{내에 반드시 할당되는 해당 연산 종류의 노드 갯수}) / (\text{범위 } R\text{의 크기})]$ 를 각각 계산하여, 그 최대값을 주어진 상태에 대한 해당 연산 종류별 하한 갯수라고 결정한다. 주어진 상태에 대한 각 연산 종류별 하한 갯수에 대한 추정이 완료되면, 함수 LB()는 수식 (1')을 이용하여 주어진 상태에 대한 하한 비용, 즉 주어진 상태로부터 전이 가능한 임의의 상태가 가질 수 있는 면적 비용에 대한 하한을 추정한다.

```

LB(S) /* 주어진 상태 S에 대한 하한 비용을 추정 */ *
| 모든 연산 종류 t에 대해,
|   L-LBt ← LOCAL-LB(S, t);
|   Global-LB ← Σt(At*L-LBt);
|   return(Global-LB);
|
LOCAL-LB(S, t) /* 주어진 상태 S에 대해 연산 종류 t에 대한 하한
|   갯수를 추정 */;
| L-LB ← 0;
| 연산 종류가 t인 노드가 할당될 가능성이 있는 세이스텝을
| 모두 찾는다;
| 이를로부터 연속된 세이스텝 범위들을 각각 구성한다;
| 연속된 각 세이스텝 범위에 속하는 임의의 세이스텝 범위
| R=[from, to]에 대해,
|   N ← 상태 S 및 상태 S에서 전이 가능한 임의의
|   상태에서 세이스텝 범위 R내에 반드시 할당되는
|   노드 가운데 연산 종류가 t인 노드들의 갯수;
|   M-LB ← |N/(to - from + 1)|;
|   If(M-LB > L-LB) L-LB ← M-LB;
|
| return(L-LB);
|

```

그림 4. 하한 비용 추정 알고리듬의 개관

Fig. 4. An overview of the lower bound cost estimation algorithm.

이렇게 추정된 하한 비용은 경우에 따라 실제의 최소 면적 비용과 차이가 날 수 있다. 하지만 이러한 오차는 다음의 정리와 같이 각 노드를 현재의 세이스텝이나 그 이후의 세이스텝에 채스케줄링하는 방향으

로만 이루어지는 상태의 전이가 계속될수록 줄어드는 경향이 있다.

| 정리 6 | $S \Rightarrow S'$ 인 두 상태 S, S' 에 대해, 동일한 세어스텝 범위 R 에서 연산 종류 t 에 대해 추정한 하한 갯수를 각각 L, L' 라고 할 때, $L \leq L'$ 가 성립한다.

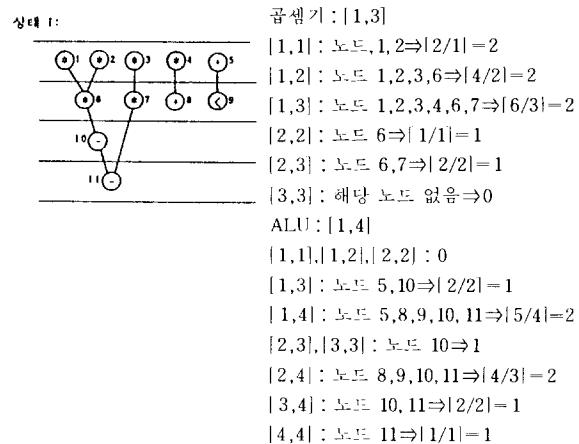
| 증명 | 상태 S 에서 범위 $R=[s, d]$ 내에 소속된 노드 가운데 연산 종류가 t 이고 할당 가능 구간이 $[S_n, L_n]$ 인 임의의 노드 n 이 상태 S 에서 세어스텝 c 에 스케줄링되어 있다고 하자. 그러면 노드 n 은 범위 R 내에 소속되므로 “ $s \leq c \leq L_n \leq d$ ”가 성립한다. 이러한 노드 n 이 만일 상태 S' 에서 세어스텝 c' 에 스케줄링된다고 가정하면, 상태 전이의 특성상 $c \leq c'$ 가 성립한다. 또한 상태 전이에 있어 한 노드에 대한 할당 가능 구간의 최후 세어스텝은 변동이 없으므로 부등식 “ $s \leq c' \leq L_n \leq d$ ”가 성립한다. 따라서 상태 S 에서 범위 R 에 소속된 임의의 노드는 반드시 상태 S' 에서도 동일 범위에 소속된다. 즉 범위 R 에 소속되는 노드 가운데 연산 종류가 t 인 노드의 갯수를 각각 N, N' 라고 할 때, $N \leq N'$ 가 항상 성립한다. 상태 S 에서 범위 R 에 대해 추정한 연산 종류 t 에 대한 하한 갯수 L 은 $\lceil N / (d - s + 1) \rceil$ 으로, 그리고 상태 S' 에서 동일 범위에 대해 추정한 연산 종류 t 에 대한 하한 갯수 L' 는 $\lceil N' / (d - s + 1) \rceil$ 으로 각각 표현되므로 $L \leq L'$ 가 성립한다.

3) 하한 비용 추정의 시간 복잡도

전체 노드의 갯수를 m 이라고 하고 전체 세어스텝의 수를 e , 그리고 연산 종류의 갯수를 t 라고 할 때, 각 연산 종류별로 할당 가능 세어스텝을 모두 찾는데 $O(m+e)$, 그리고 각 연산별 하한 갯수를 구하는데 $O(m * e^2)$ 의 시간 복잡도가 발생한다. 따라서 연산 종류별 하한 갯수를 모두 구하여 주어진 상태에 대한 하한 비용을 계산하기 위한 시간 복잡도는 $O(t * m * e^2)$ 이라고 할 수 있다.

그림 5는 그림 2의 ASAP 스케줄링 결과를 나타내는 상태 I에 대해 각 연산 종류별 하한 갯수를 추정하는 과정을 보여준다. 먼저 곱셈기의 경우를 살펴보면, 곱셈 연산들이 할당 가능한 연속된 세어스텝의 범위는 단 하나로서 $[1,3]$ 이다. 범위 $[1,3]$ 에 대해 구성 가능한 각 범위($[1,1], [1,2], [1,3], [2,2], [2,3], [3,3]$)에 대해 곱셈기에 대한 하한 갯수를 각각 추정한다. 그림 5의 상태 I의 경우, 그 값으로 각각 2, 2, 2, 1, 0이 나오는데, 그 중 최대값인 2를 곱셈기에 대한 하한 갯수로 설정한다. 마찬가지로 ALU의 경우, 연속된

범위는 $[1,4]$ 단 하나로서 구성 가능한 각 범위들에 대해 추정한 하한 갯수의 최대값은 2가 된다. 따라서 최대값인 2를 상태 I에서의 ALU에 대한 하한 갯수로 결정한다. 상태 I에 대한 하한 비용, LB(I)는 수식 $(1')$ 에 의해 “ $2 \times (\text{곱셈기의 면적}) + 2 \times (\text{ALU의 면적})$ ”이며, 이 예의 경우 그 값은 최적의 스케줄링 결과가 갖는 면적 비용과 동일함을 알 수 있다.



상태 I에서 곱셈기에 대한 하한 갯수 = $\text{MAX}(2, 2, 2, 1, 0) = 2$
상태 I에서의 ALU에 대한 하한 갯수 = $\text{MAX}(0, 0, 0, 1, 2, 1, 1, 2, 1, 1) = 2$

그림 5. 각 연산 종류별 하한 갯수의 추정 과정

Fig. 5. Process of lower bound estimation for FU requirement.

3. 하한 비용 추정에 의거한 스케줄링 알고리듬

(LBS 알고리듬)

LBS 알고리듬은 admissible한 초기 상태에서 시작하여 최종적으로 최적 상태를 발견함에 있어, 최적 상태로 전이될 수 없는 상태를 부분적이나마 빨리 판별하여 미리 제거할 뿐 아니라, 최적 상태의 도달 여부를 부분적으로 판별함으로써 궁극적으로 최적 상태에 도달하는 속도를 향상시킨다. 그림 6은 LBS 알고리듬의 개략적인 구조를 보여준다.

그림 6에서 변수 “I_state”는 초기 상태를, “Min_state”는 현재까지 구성된 상태중 최소의 면적 비용을 갖는 상태, 그리고 “Final_state”는 최종적으로 선택된 최적 상태를 각각 의미한다.

그림 6에서 함수 FIND_BEST()는 branch-and-bound 방식을 통해 주어진 상태로부터 전이 가능한

```

DDG : 입력된 데이터 의존 그래프
TC : 활용된 전체 세이스텝의 수
LBS(DDG, TC)
{
    TC 범위 내에서 DDG의 각 노드 n에 대해,
        할당 가능 구간 TF(n) = [Sn, Sn] 을 설정한다;
        I-state ← ASAP_SCHEDULING(할당 가능 구간들);
        Min-state ← I-state;
        FIND-BEST(I-state);
        Final-state ← Min-state;
        exit;
    }

    FIND-BEST(S)
    {
        If(COST(S) == LB(I-state))
        {
            Final-state ← S;
            exit;
        }
        If(COST(S) < COST(Min-state))
            Min-state ← S;
        If(LB(S) > COST(Min-state) 또는
            S와 동일한 상태가 이미 고려된 바 있으면)
            return;
        모든 연산 종류 t에 대해 (정렬된 순서에 따라),
        |   연산 종류가 t인 노드가 가장 많이 할당되어 있는 최대
            세이스텝이란한 세이스텝이 여러 개가 있을 경우, 임계
            경로에 있는 노드가 가장 많은 세이스텝)을 선택하여
            MAX-CSTst로 둔다;
        BCSst ← MAX-CSTst에 할당되어 있는 연산 종류
        |의 노드들];
        If(BCSst의 원소 개수 == LOCAL.LB(S, t))
            BCSst ← Φ;
        BCSst내의 모든 노드 n에 대해,
        |   later_cstep ← MAX-CSTst+1;
        |   If(later_cstep ≤ Ln)
        |       New-state ← RESCHEDULING(S, n,
        |           later_cstep);
        |       FIND-BEST(New-state);
        |
    }
    return;
}

```

그림 6. LBS 알고리듬의 개략적인 구조
Fig. 6. An overview of LBS algorithm.

상태 중 최소의 면적 비용을 갖는 상태를 발견하는 기능을 수행한다. 따라서 주어진 상태 S가 admissible하면, FIND-BEST()는 반드시 최적 상태를 발견한다. 함수 LB()는 앞에서 설명한 바와 같이 주

어진 상태에 대한 하한 비용을 추정하며, COST()는 수식 (1')를 이용해 주어진 상태의 면적 비용을 계산한다. 그리고 함수 ASAP_SCHEDULING()은 데이터 의존 그래프의 각 노드를 자신의 할당 가능 구간의 첫번째 세이스텝에 스케줄링하며, RESCHEDULING()은 현재의 상태 (S)로부터 선택된 한 노드(n)를 현재 스케줄링된 세이스텝 이후의 다른 세이스텝 (later_cstep)에 재스케줄링한다. RESCHEDULING()에서 한 노드를 재스케줄링시킬 때, 이로 인해 데이터 의존 관계가 유지될 수 없으면, 재스케줄링된 노드에 데이터 의존하는 다른 노드들을 반복적으로 재스케줄링시켜 데이터 의존 관계가 계속 유지되도록 한다.

1) 초기 상태의 선택

ASAP 스케줄링 결과를 나타내는 상태는 (정리1)에 의해 항상 admissible하다. 따라서 이를 초기 상태 I-state로 선택한다. 초기 상태 I-state는 admissible하기 때문에 함수 FIND-BEST()에 의해 반드시 임의의 최적 상태로 전이된다.

2) bound 기법

LBS 알고리듬에서는 초기 상태를 비롯한 각 상태에 대해 추정한 하한 비용을 이용하여, 주어진 상태가 최적 상태인지의 여부와 admissible하지 않은지의 여부를 부분적이나마 미리 판별하여 bound 함으로써 불필요한 상태의 탐색을 가급적 줄인다.

(1) 최적 상태의 부분적인 발견

주어진 상태 S의 면적 비용 COST(S)가 초기 상태 I-state에 대한 하한 비용 LB(I-state)와 같은 경우, (정리3)에 의해 상태 S는 최적 상태이다. 따라서 이러한 관계가 성립하면 LBS 알고리듬은 그 상태 S를 최종 스케줄링 결과로 출력한 뒤 더 이상의 탐색을 수행하지 않는다.

(2) admissible하지 않은 상태의 부분적인 판별

만일 주어진 상태 S에 대한 하한 비용 LB(S)가 현재까지 탐색된 상태 중 최소의 면적 비용을 갖는 상태인 Min_state의 면적 비용 COST(Min_state)보다 큰 경우, (정리4)에 의해 상태 S는 결코 admissible하지 않다. 상태 S가 admissible하지 않다고 판명된 경우, 그 상태로부터 전이 가능한 최적 상태는 존재하지 않으므로 주어진 상태 S는 물론 S로부터 전이 가능한 모든 상태는 더 이상 고려될 필요가 없다. 따라서 상태 S를 bound시킨다.

(3) 불필요한 상태의 판별

주어진 상태 S에 대해 조건 “LB(S) = COST(Min_state)”가 성립하는 경우, 수식 (2), (3)에 의해 수식 “COST(Min_state) = LB(S) ≤ COST(S)”와 “COST”

$(\text{Min_state}) \leftarrow \text{LB}(S) \leq \text{COST}(S')$ "가 각각 성립한다. 이 두 수식은 상태 S 로부터 전이 가능한 상태들 가운데 최소의 면적 비용을 갖는 상태를 발견하더라도 그 상태의 면적 비용은 이미 발견된 상태 Min state 의 면적 비용보다 결코 작지 않음을 의미한다. LBS 알고리듬에서는 최적 상태를 모두 찾기 보다는 임의의 한 최적 상태를 찾는 것이 목표이므로, 알고리듬의 속도 향상을 위해 이러한 조건이 만족되는 경우에도 마찬가지로 상태 S 를 bound시킨다. 한편, 새스케줄링이 일어나는 순서에 따라 동일한 상태가 여러 번 구성될 수 있다. 동일한 상태가 이미 구성된 적이 있다면 그 상태 및 그 상태에서 전이 가능한 임의의 상태가 이미 탐색되었다는 뜻이다. 따라서 이미 구성된 적이 있는 상태가 탐색되는 경우 그 상태를 bound시킨다.

3) branch 기법

주어진 상태 S 가 bound되지 않을 경우, 주어진 상태로부터 새스케줄링 할 노드의 집합을 구성하고, 그 집합의 각 노드가 선택될 가능성에 대해 각각 branch를 형성한다.

(1) branch의 구성

LBS 알고리듬에서는 (정리 2)에 의해 주어진 상태 S 에서 구성된 각 연산 종류별 BCS에 소속된 노드들에 대해서만 새스케줄링을 고려한다. 따라서 BCS에 소속된 각 노드에 대해 그 노드가 선택될 가능성을 각각 하나의 branch로 구성한다. 그런데 수식 (1')에서 M_{st} 가 더 이상 감소될 수 없다고 판명되는 경우, 주 BCS_{st}에 속한 원소의 갯수가 연산 종류 t 의 가능 단위의 갯수에 대한 하한(즉 LOCAL_LB(S, t))과 같을 경우, BCS_{st}의 원소들을 반드시 새스케줄링 할 필요가 없다. 따라서 이런 경우에는 알고리듬의 효율을 위해 BCS_{st}를 공집합으로 둠으로써 이러한 노드들에 대한 새스케줄링 가능성을 고려하지 않는다. 한편, 한 연산의 종류에 대해 BCS가 여러 개 존재할 경우가 있는데, 이 경우 임계 경로에 있는 노드가 가장 많은 BCS를 하나 선택함으로써 새스케줄링 할 노드 선택시의 가지수를 되도록 줄인다.

(2) 노드의 선택 (branch의 선택)

BCS에 속한 노드 가운데 어느 노드에 대한 새스케줄링을 먼저 고려하느냐에 따라 최적 상태에 도달하기 위해 탐색해야 할 상태의 수가 달라진다. 따라서 각 연산 종류별로 BCS가 구성되면, BCS에 속한 노드들 가운데 어떤 노드를 먼저 선택할지를 결정하는 것은 알고리듬의 효율 결정에 매우 중요하다. 따라서 LBS 알고리듬에서는 알고리듬의 효율을 높이기 위해 BCS에 속한 노드들 가운데 할당 가능

구간의 최후 제어스텝이 큰 노드를 먼저 선택한다. 이러한 선택 기준은 마치 리스트 스케줄링 기법에서 할당할 노드의 수가 허용된 기능 단위의 수보다 많을 경우, 나중에 스케줄링 할 노드를 선택하는 우선 순위와 유사하다. 하지만 이러한 우선 순위는 단지 탐색 속도를 향상하기 위한 것일 뿐, 리스트 스케줄링에서와 같이 최종 선택된 스케줄링 결과의 면적 비용의 결정에 영향을 미치지는 않는다.

(3) 새스케줄링 할 제어스텝의 결정

새스케줄링 할 노드 n 이 선택되면, 그 노드를 BCS가 구성된 해당 최대 제어스텝의 바로 다음 제어스텝 later_cstep에 새스케줄링 시켜 새로운 상태 New_state를 구성한다. 여기서 각 노드에 대해 단지 하나의 제어스텝에 대한 새스케줄링만을 고려하는 이유는, 만일 새로이 구성된 상태가 admissible 하다면 같은 노드 n 을 다른 제어스텝에 새스케줄링함으로써 구성되는 임의의 다른 상태는 고려할 필요가 없기 때문이다. 반대로 새로운 상태가 admissible 하지 않다면 선택된 제어스텝 later_cstep 보다 늦은 제어스텝에 새스케줄링 함으로써 구성되는 임의의 상태 역시 admissible하지 않기 때문이다. 새로운 상태가 구성되면 그 상태에 대해 함수 FIND-BEST()를 적용하여 그 상태로부터 전이 가능한 상태 가운데 최소의 면적 비용을 갖는 상태를 찾는다. 만일 새스케줄링될 제어스텝 later_cstep이 선택된 노드 n 의 할당 가능 구간을 벗어나서 새스케줄링될 수 없거나 (즉 later_cstep > L_n) 선택된 노드에 대한 새스케줄링에 의해 구성된 새로운 상태로부터 전이 가능한 상태 가운데 최적 상태가 없다고 판단되는 경우, 그 다음 노드에 대한 새스케줄링을 고려한다.

(4) 연산 종류의 선택

각 연산 종류별로 구성된 BCS 가운데 어느 연산 종류에 대한 BCS를 먼저 고려하느냐에 따라 알고리듬의 효율이 달라질 수 있다. LBS 알고리듬에서는 주어진 상태 S 에서의 연산 종류 t 에 대한 하한 갯수, 즉 LOCAL_LB(S, t)와 그 상태에서의 해당 연산 종류에 대한 BCS를 우선적으로 고려한다. 만일 그 차이가 동일한 경우, 해당 기능 단위의 면적 (At)이 큰 연산 종류를 먼저 선택한다.

그림7은 그림1의 예에 대해 탐색이 일어나는 과정을 보여준다. 그림 7의 (a)는 그림 1의 예에 대해 구성된 탐색 공간을 나타내는 나무구조로서 각 노드는 하나의 상태를, 그리고 각 액션은 새스케줄링 할 노드의 선택에 따른 상태의 전이 가능성을 의미한다. 이 그림에서는 ●는 탐색된 상태를, ○는 탐색되지는 않았지만 다른 상태의 탐색에서 최적 상태 발견에 실패할 경우에 탐색될 가

능성이 있는 상태를 각각 의미한다. 이 예의 경우, ASAP 스케줄링 결과를 표현하는 초기 상태로부터 최적 상태에 이르기까지 3번의 상태 전이가 수행되며, 이 경우 탐색된 상태는 결과적으로 모두 admissible한 상태이다. 그림7의 (b)에 제시된 상태 S1은 그림1의 예에 대한 초기 상태로서 이 상태에서의 곱셈 연산 종류에 대한 BCS는 {1, 2, 3, 4}가 된다. 한편 상태 S1에서의 ALU 연산 종류에 대한 BCS는 {8, 9}가 되지만 ALU에 대

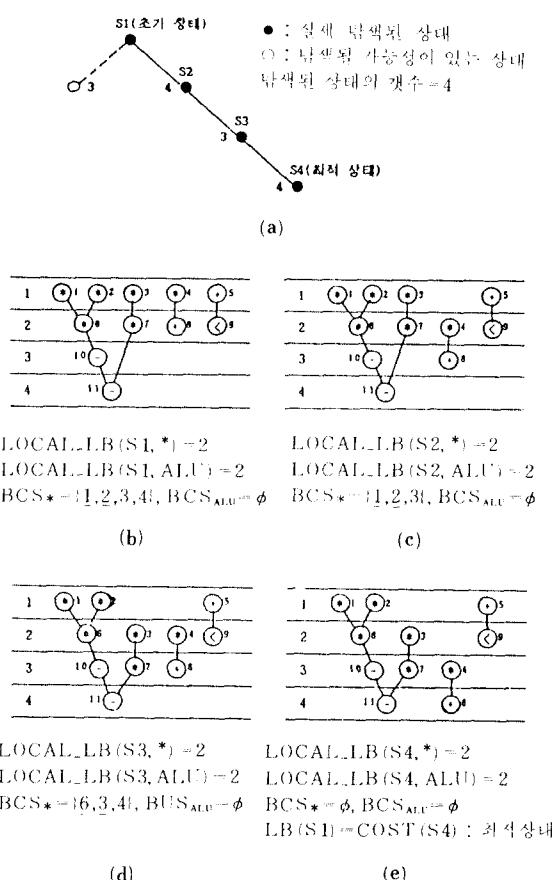


그림 7. 그림 1의 예에 대한 LBS 알고리듬의 탐색 공간

- (a) 탐색 나무 구조
- (b) 초기 상태 S1
- (c) 노드 4를 세어스텝 2로 옮긴 후 (S2)
- (d) 노드 3을 세어스텝 2로 옮긴 후 (S3)
- (e) 노드 4를 세어스텝 3으로 옮긴 후 (S4)

Fig. 7. Search space of LBS algorithm for the example in Fig. 1.

- (a) Search tree,
- (b) Initial state S1,
- (c) After rescheduling node 4(S2),
- (d) After rescheduling node 3(S3),
- (e) After rescheduling node 4 again(S4).

한 하한 갯수가 2이므로 ALU에 대한 BCS는 공집합이 된다. 따라서 노드 1, 2, 3, 4에 대해서만 재스케줄링을 고려한다. 그런데 노드 1, 2는 임계 경로 (critical path)에 있어, 재스케줄링 할 여지가 없으므로 노드 3 또는 노드 4가 선택될 수 있다. (그림 7에서 임계 경로에 있는 노드들은 각각 밑줄을 그어 표시하였다) 그런데 노드 4의 최후 세어스텝 ($L_4=3$)이 노드 3의 최후 세어스텝 ($L_3=2$) 보다 크므로 노드 4를 먼저 선택하여, 해당 BCS가 구성된 세어스텝 1의 바로 다음 세어스텝인 세어스텝 2에 재스케줄링 한다. 이 때, 노드 4에 데이터 의존하는 노드 8 역시 데이터 의존관계를 유지하기 위해 세어스텝 3에 재스케줄링 된다. 결과적으로 상태 S1은 재스케줄링에 의해 그림 7의 (c)에 나타난 상태 S2로 전이된다. 새로이 구성된 상태 S2 역시 최적 상태를 위한 충분 조건을 만족하지 않을 뿐더러, admissible하지 않은 충분조건도 만족하지 않는다. 따라서 상태 S2에 대해 BCS를 다시 구성하는데, 이 경우 곱셈기에 대한 BCS는 {1, 2, 3}이 되며, ALU에 대한 BCS는 역시 공집합이 된다. 곱셈기에 대한 BCS에서 노드 1, 2는 임계 경로에 있으므로 노드 3이 재스케줄링을 위해 선택된다. 그림7의 (d)에 있는 상태 S3는 상태 S2에서 노드 3을 재스케줄링할 때 구성되는 상태를 보여준다. 이러한 재스케줄링 과정은 최적 상태를 위한 충분 조건이 만족되거나 더 이상 고려할 상태가 없을 때까지 계속 반복된다. 이 예의 경우, 최종적으로 그림7의 (e)에 있는 상태 S4가 구성되는데, 상태 S4는 (정리 3)에 의해 최적 상태임을 알 수 있다. 따라서 LBS 알고리듬은 상태 S4를 최종 스케줄링 결과로 출력한다.

IV. 알고리듬의 확장

앞에서는 설명의 편의를 위해 모든 연산이 하나의 세어스텝, 즉 클록 주기내에 수행되는 것으로 가정하였다. 하지만 실제의 경우에 있어서는 연산의 수행 시간이 세각기 다른 점을 이용해 더욱 짧은 수행 시간을 갖는 결과를 얻을 수 있다. 전체 수행 시간을 줄이는 방법으로는 여러 개의 순차적인 연산을 동일한 세어스텝에 수행하는 체이닝(chaining) 기법과 수행 시간이 클록 주기보다 길어 여러 세어스텝에 걸쳐 수행되는 다주기(multi-cycling) 연산을 지원하는 방법, 그리고 파이프라인(pipeline) 데이터부를 합성하는 방법 등을 들 수 있다. 본 논문에서는 연산의 체이닝과 다주기 연산, 그리고 파이프라인 데이터부의 합성을 지원하기 위해 LBS 알고리듬을 확장하였다.

1. 체이닝 연산의 지원

한 연산의 수행 주기가 클록 주기에 비해 작아서 여러 개의 연산을 순차적으로 수행하더라도 그 수행 시간이 하나의 클록 주기보다 작은 경우가 있다. 이 경우, 그러한 연산들을 동일한 제어스텝에서 수행시킬 수 있는데, 이러한 기법을 체이닝 기법이라고 하고, 체이닝되는 연산을 체이닝연산이라고 한다. 체이닝 기법을 이용할 경우 데이터 의존하는 두개 이상의 연산들이 하나의 제어스텝에 스케줄링될 수 있기 때문에 LBS 알고리듬이 연산의 체이닝을 지원하기 위해서는 데이터 의존 관계를 유지하는 방법을 확장해야 한다. LBS 알고리듬에서 이와 관련된 부분은 각 노드에 대해 할당 가능 구간을 설정하는 과정과 노드를 나중 제어스텝으로 재스케줄링 과정인데, 이 두 부분을 연산의 체이닝을 허용하도록 확장하였다.

2. 다주기 연산의 지원

수행 시간이 긴 연산이 있을 경우, 모든 연산을 한 클록 주기내에 수행하기 위해서는 매우 큰 클록 주기를 사용해야 한다. 하지만 클록 주기의 작게 하여 수행 시간이 긴 연산은 여러 제어스텝에 걸쳐 수행되도록 함으로써 오히려 전체 수행 시간을 줄일 수 있다. 이렇게 여러 제어스텝에 걸쳐 수행되어야 하는 연산을 다주기 연산이라고 하고, 다주기 연산이 수행되는 제어스텝의 수를 그 연산의 수행 주기라고 한다. 수행 주기가 s 인 노드 n 이 제어스텝 j 에 스케줄링될 경우, 노드 n 은 제어스텝 j 에서부터 제어스텝 $(j+s-1)$ 에 걸쳐 수행된다. 따라서 입력 그래프에 정의된 데이터 의존 관계를 유지하기 위해서는 노드 n 에 데이터 의존하는 노드 q 는 적어도 제어스텝 $(j+s)$ 이후에 스케줄링되어야 하며, 반대로 노드 n 을 차식 노드로 하는 노드 p 는 노드 p 의 수행 주기를 r 이라고 할 때, 늦어도 제어스텝 $(j-r)$ 이전에 스케줄링 되어야 한다. 따라서 LBS 알고리듬에서 다주기 연산을 지원하기 위해서는 데이터 의존 관계를 유지하는 방법을 확장해야 한다. LBS 알고리듬에서 이와 관련된 부분은 각 노드에 대해 할당 가능 구간을 설정하는 방법과 재스케줄링 방법 등인데, 이를 다주기 연산을 허용하도록 확장하였다.

한편, 수행 주기가 s 인 노드 n 이 제어스텝 j 에 스케줄링되는 경우, 노드 n 의 실행을 위해 할당될 기능 단위는 제어스텝 j 에서부터 제어스텝 $(j+s-1)$ 에 걸쳐 연속적으로 사용되므로 이 제어스텝 범위 동안은 다른 노드가 이 기능 단위를 공유할 수 없다. 따라서 비록 노드 n 이 제어스텝 j 에 스케줄링되더라도, 마치 노드 n 이 제어스텝 j 부터 제어스텝 $(j+s-1)$ 까-

지의 모든 제어스텝에 각각 할당된 것으로 취급해야 한다. 이는 LBS 알고리듬에서 최대 제어스텝을 선정하는 과정과 각 상태의 면적 비용을 계산하는 과정, 그리고 하한 갯수를 추정하는 과정과 관련있다. 따라서 이 부분을 수정하여 다주기 연산을 지원하도록 확장하였다.

3. 파이프라인 데이터부의 합성

데이터부 전체가 파이프라인 방식으로 동작하도록 함으로써 회로의 전체적인 실행 효율을 높일 수 있는데, 이를 위해 파이프라인 데이터부의 합성을 지원하는 스케줄링 기법이 필요하다. 파이프라인 방식에서는 동일한 연산이 일정한 간격을 두고 반복되는 특징이 있는데, 이 간격을 지연 주기 (latency) 라고 한다. 지연 주기가 ℓ 인 파이프라인 데이터부에서는 $j+\rho * \ell (\rho=0, 1, 2, \dots)$ 제어스텝에 있는 모든 연산은 일정한 시간이 지난 후에 동시에 수행된다. 따라서 이 연산들은 같은 기능 단위를 공유할 수 없다.

LBS 알고리듬에서는 이와 같은 파이프라인 데이터부의 합성을 지원하기 위해 동시에 수행되는 제어스텝들에 대해 하나의 의사 제어스텝 (pseudo control step)을 구성하고, 실질적으로 동시에 수행되는 연산들을 하나의 의사 제어스텝에 모두 모은다. 이 경우, 의사 제어스텝의 수는 지연 주기와 같은 ℓ 이 된다. LBS 알고리듬에서 이와 관련된 부분은 각 연산 종류별 최대 제어스텝을 선정하는 과정과 각 상태의 면적 비용을 계산하는 과정, 그리고 하한 갯수를 추정하는 과정 등이다. 따라서 이 부분을 파이프라인 합성을 지원하도록 확장하였다.

V. 실험결과

LBS 알고리듬은 UNIX 환경 하에서 C 언어로 구현되었다. 본 절에서는 LBS 알고리듬의 성능을 증명하기 위해 실험된 여러가지 예제들 가운데 기존의 연구에서 사용한 세가지 예에 대한 실험 결과를 제시한다. 표1, 2, 3은 LBS 알고리듬과 기존의 대표적인 스케줄링 알고리듬인 FDS, ALPS, SEHWA^[15]의 실험 결과를 보여준다. 표에서 기존 연구의 실험 결과는 각각 참고문헌 [12], [13] 및 [15]에서 인용하였으며, LBS 알고리듬에 대한 실험은 SUN 4/110 시스템에서 수행하였다.

첫번째 예는 그림1의 “2차 미분 방정식”예로서, 이 실험에서는 덧셈, 뺄셈, 및 비교 연산 모두 ALU에 의해 수행되며 곱셈 연산은 곱셈기애 의해 수행됨을 가정하였다. 또한 다른 실험 결과와의 비교를 위해, 곱셈의 수행 주기는 2이며, 나머지 연산의 수행 주기

표 1. 첫번째 예에 대한 실험 결과

Table 1. Experimental result for the first example.

| 2 차 미분 방정식 예 | | | | |
|--------------|------|-----|--------|-------|
| 알고리듬 | 세어스텝 | (*) | (ALUs) | CPU시간 |
| FDS* | 6 | 3 | 2 | 15초 |
| | 7 | 2 | 2 | 35초 |
| ALPS** | 6 | 3 | 2 | 0.17초 |
| | 7 | 2 | 2 | 0.28초 |
| LBS*** | 6 | 3 | 2 | 0.01초 |
| | 7 | 2 | 2 | 0.03초 |

*: Xerox 1108 Lisp machine

**: VAX 11/8800

***: SUN 4/110

표 2. 두번째 예에 대한 실험 결과

Table 2. Experimental result for the second example.

| Fifth Order Elliptic Wave Filter | | | | |
|----------------------------------|------|-----|-----|-------|
| 알고리듬 | 세어스텝 | (*) | (+) | CPU시간 |
| FDS * | 17 | 3 | 3 | 1분 |
| | 18 | 2 | 3 | 3분 |
| | 19 | 2 | 2 | 7분 |
| | 20 | 2 | 2 | 10분 |
| | 21 | 1 | 2 | 13분 |
| ALPS** | 17 | 3 | 3 | 0.26초 |
| | 18 | 2 | 2 | 3.1초 |
| | 19 | 2 | 2 | 10.5초 |
| | 20 | 2 | 2 | 19.4초 |
| | 21 | 1 | 2 | 34.5초 |
| LBS*** | 17 | 3 | 3 | 0.02초 |
| | 18 | 2 | 2 | 0.05초 |
| | 19 | 2 | 2 | 0.04초 |
| | 20 | 2 | 2 | 0.04초 |
| | 21 | 1 | 2 | 0.05초 |

표 3. 세번째 예에 대한 실험 결과

Table 3. Experimental result for the third example.

| 16-point Digital FIR Filter | | | | |
|-----------------------------|------|-----|-----|-------|
| 알고리듬 | 세어스텝 | (*) | (+) | CPU시간 |
| SEHWA(전체답색)+ | 6 | 3 | 5 | 1시간내 |
| SEHWA(부분답색) | 7 | 3 | 6 | N/A |
| FDS | 6 | 3 | 5 | 30초 |
| ALPS | 6 | 3 | 5 | 0.32초 |
| LBS | 6 | 3 | 5 | 0.02초 |
| | 7 | 3 | 5 | 0.02초 |

+: VAX 11-750

는 1이라고 가정하였다. 이 경우 임계 경로의 길이는 6이 된다. 표1에 나타난 바와 같이 이 예에 대해서는 전체 제어스텝의 수에 관계없이 LBS 알고리듬을 포함한 모든 알고리듬이 최적의 결과를 산출한다.

두번째 예는 "fifth order elliptic filter" 예로서 이 예는 1988년에 개최된 high level synthesis workshop^[14]에서 공식 예제로 채택된 것이다. 이 예는 26개의 덧셈 연산과 8개의 곱셈 연산으로 구성되어 있다. 곱셈의 수행 주기는 2이며 덧셈의 수행 주기는 1이다. 따라서 이 경우 임계 경로의 길이는 17이 된다. 이 예에 대해서는 표2에 제시한 바와 같이 허용된 제어스텝의 수를 17에서 21까지 바꾸어 가며 실험하였다. 실험에서 FDS는 허용 제어스텝의 수가 18인 경우 최적의 결과를 산출하지 못하는 반면, ALPS와 LBS 알고리듬은 모두 경우에서 최적의 결과를 산출한다.

세번째 예는 SEHWA^[15]에서 사용한 "pipelined 16-point digital FIR filter" 예로서, 수행 주기가 1인 곱셈 연산과 수행 주기가 0.5인 덧셈 연산으로 구성된다. 이 예에 대한 실험에서는 지연 주기가 3이라고 가정하였다. 표3은 이 예에 대한 실험 결과를 보여준다. 이 실험의 경우, SEHWA는 제어스텝 7에서 최적 결과를 얻지 못하는 반면, LBS 알고리듬을 포함한 나머지 알고리듬들은 모두 최적 결과를 산출한다.

실험 결과, LBS 알고리듬은 예상한 바와 같이 모든 경우에 대해 최적의 결과를 산출하였다. 흥미로운 사실은 LBS 알고리듬에서는 제어스텝의 증가가 반드시 많은 CPU 시간을 요구하는 것은 아니라는 점이다. 이는 제어스텝의 증가가 CPU 시간에 많은 영향을 주는 다른 알고리듬과는 대조적이다. LBS 알고리듬이 최적의 결과를 빠른 시간 내에 낼 수 있었던 원인은, 하한 비용이 비교적 정화하게 추정되어 최적 상태의 도달 여부에 대한 판별 및 탐색이 불필요한 상태에 대한 선별이 잘 이루어졌기 때문이다. 고려함으로써 불필요한 상태의 탐색을 효과적으로 방지했기 때문이다.

VI. 결론 및 앞으로의 연구 방향

본 논문에서는 주어진 제어스텝 세팅하에서 최소의 면적 비용을 갖는 스케줄링 결과를 산출하는 LBS 알고리듬에 대해 설명하였다. LBS 알고리듬은 ASAP 스케줄링 결과로부터 시작하여 선택된 노드를 나중 세어스텝으로 미루는 일련의 채스 케줄링 과정을 통

해 주어진 세이스텝 제약하에서 최소의 면적 비용을 갖는 최적의 스케줄링 결과를 찾는데, 이를 위해 branch-and-bound 기법을 활용한다. 이 방법에서는 전체 면적 비용의 설정에 영향을 미치는 노드들에 대해서만 재스케줄링을 고려하는 branch 기법과 주어진 스케줄링 결과는 물론 그 결과에 대한 재스케줄링에 의해 구성될 수 있는 임의의 스케줄링 결과가 갖는 면적 비용에 대한 하한 추정을 통하여 불필요한 탐색을 미연에 방지하는 bound 기법을 사용한다. 이러한 branch-and-bound 기법은 결과적으로 탐색 공간의 크기를 대폭 줄이며, 따라서 많은 경우 최적의 결과를 빠른 시간내에 찾도록 돋는다.

한편, 본 알고리듬을 체이닝 연산과 다주기 연산, 그리고 데이터부의 합성 등과 같이 보다 일반적인 합성 환경을 지원하도록 확장함으로써 알고리듬의 실용성을 높였다.

실험 결과, 본 알고리듬은 예상한 바와 같이 실험된 모든 경우에서 최적의 스케줄링 결과를 빠른 시간내에 찾았다. 최적의 결과를 빠른 시간 내에 찾을 수 있었던 원인은 본 논문에서 개발한 하한 추정 알고리즘이 대부분의 경우 최소의 면적 비용과 동일한 값을 추정함으로써 최적 스케줄링 결과와 bound 할 스케줄링 결과를 거의 완벽하게 판별했기 때문이다. 또한 면적 비용의 설정에 영향을 미치는 노드에 대해서만 재스케줄링 가능성을 고려함으로써 불필요한 탐색이 효과적으로 방지되었기 때문이다.

하지만 본 알고리듬은 추정된 하한 비용을 토대로 탐색이 불필요한 경우를 판별하는 bound 기법을 사용하기 때문에, 추정된 하한 비용이 실제의 최소 면적 비용과 차이가 날 경우에는 불필요한 상태의 탐색 수가 증가할 가능성이 있다. 따라서 본 알고리듬의 속도를 더욱 향상시키기 위해서는, 하한 추정 기법의 성능을 보다 향상시켜 더욱 많은 경우에서 최소 면적 비용과 동일한 하한 비용을 추정하도록 하는 한편, 반드시 최적의 결과는 보장하지 않더라도 알고리듬의 속도를 상당히 향상시킬 수 있는 heuristic의 개발이 필요하다.

参考文献

- [1] M.C. McFarland, A.C. Parker, and R. Composano, "Tutorial on high level synthesis," *Proc. the 25th Design Automation Conference (DAC)*, pp. 330-336, June 1988.
- [2] D.D. Gajski, N.D. Dutt and B.M. Pangrle, "Silicon compilation (Tutorial)," *Proceedings of the IEEE 1986 Custom In-*
- tegrated Circuits Conference (CICC)*, pp. 102-110, May 1986.
- [3] A. Orailglu and D.D. Gajski, "Flow graph representations," *Proc. 23rd DAC*, pp. 503-509, June 1986.
- [4] R.J. Cloutier and D.E. Thomas, "The combination of scheduling, allocation ,and mapping in a single algorithm," *Proc. 27th DAC*, pp. 71-76, June 1990.
- [5] R.J. Jain, et al., "Experience with the ADAM synthesis system," *Proc. 26th DAC*, pp. 56-61, June 1989.
- [6] P.G. Paulin, J.P. Knight and Girczyc, "HAL: a multi-paradigm approach to automatic data path synthesis," *Proc. 23rd DAC*, pp. 263-270, July 1986.
- [7] B.M. Pangrle and D.D. Gajski, "State synthesis and connectivity binding for microarchitecture compilation" *Proc. ICCAD-86*, pp. 210-213, November 1986.
- [8] E.F. Girczyc and J.P. Knight, "An ADA to standard cell hardware compiler based on graph grammar and scheduling," *Proc. ICCTD-84*, pp. 726-731, October 1984.
- [9] C. Tseng and D.P. Siewiorek, "Automated synthesis of data paths in digital systems," *IEEE Trans. CAD*, vol. 8, pp. 661-679, June 1989.
- [10] S.Y. Kung, H.J. Whitehouse, and T. Kailath, "VLSI and modern signal processing," Prentice Hall, pp. 258-264, June 1985.
- [11] P.G. Paulin and J.P. Knight, "Scheduling and binding algorithms for high level synthesis," *Proc. 26th DAC*, pp. 1-6, June 1989.
- [12] P.G. Paulin and J.P. Knight, "Force-directed scheduling in automatic data path synthesis," *Proc. 24th DAC*, pp. 195-202, June 1987.
- [13] J.H. Lee, Y.C. Chu, and Y.L. Lin, "A new integer linear programming formulation for the scheduling problem in data path synthesis," *Proc. ICCAD-89*, pp. 20-23, November 1989.
- [14] G. Borriello and E. Detjens, "High level synthesis: current status and future directions," *Proc. 25th DAC*, pp. 477-482, June 1988.
- [15] N. Park and A.C. Parker, "SEHWA: a program for synthesis of pipelines," *Proc. 23rd DAC*, pp. 454-460, July 1986.

著者紹介

嚴聖鏞(正會員)



1962年 11月 13日生, 1985年 2月
서울대학교 컴퓨터공학과 졸업
(학사). 1987年 2月 동대학원 석사
학위 취득. 1989年 8월 동대학원
박사과정수료. 주 관심분야는
상위단계합성, VLSI/CAD 등임.

全洲植(正會員)



1952年 3月 18日生, 1975年 2月 서
울대학교 응용수학과 졸업(학사)
1977年 2月 한국과학기술원 석사
학위 취득. 1983年 2月 Univ. of
Utah 박사학위 취득. 1983~1985
年 Univ. of Iowa 조교수. 1985年
현재~서울대학교 컴퓨터공학과
부교수 주 관심분야는 컴퓨터구조, VLSI/CAD 등임.