論文94-31A-1-13

구간 그래프를 이용한 스케쥴링 알고리듬

(A Scheduling Algorithm Using the Interval Graph)

金 基 鉉*, 鄭 正 和*

(Ki Hyun Kim and Jong Wha Chong)

要 約

본 논문에서는 가중치를 갖는 구간 그래프(weighted interval graph)를 이용한 스케쥴링 알고리듬을 제안한다. ASAP와 ALAP를 이용하여 계산된 연산의 time frame에 대해 구간 그래프를 구성한다. 각 연산형별로 구성된 구간 그래프에서 최대 클릭크(maximum clique)를 구한다. 최대 클릭크의 노드 수는 동시에 수행되는 연산의 수를 의미한다. 최소의 자원 비용(resource cost)을 위해 최대 클릭크의 노드 수를 감소시킬 연산형을 선택한다. 선택된 연산 형에 대해 최대 클릭크의 노드 수를 감소시키도록 연산을 이동하여 각 제어 스텝에 최소의 연산이 할당되도록 한다. 이때 연산을 제어 스텝에 할당하는 것은 반복적인 연산의 이동에 의해 이루어 진다. 제안된 스케쥴링 알고리듬은 파이프라인형 데이타패스(pipelined datapath)와 비파이프라인형 데이타패스(nonpipelined datapath) 합성에 적용되며 실험을 통하여 효용성을 보인다.

Abstract

In this paper, we present a novel scheduling algorithm using the weighted interval graph. An interval graph is constructed, where an interval is a time frame of each operation. And for each operation type, we look for the maximum clique of the interval graph: the number of nodes of the maximum clique represents the number of operations that are executed concurrently. In order to minimize resource cost, we select the operation type to reduce the number of nodes of a maximum clique. For the selected operation type, an operation selected by selection rule is moved to decrease the number of nodes of a maximum clique. A selected operation among unscheduled operations is moved repeatly and assigned to a control step consequently. The proposed algorithm is applied to the pipeline and the nonpipeline data path synthesis. The experiment for examples shows the efficiency of the proposed scheduling algorithm.

* 正會員, 漢陽大學校 CAD 및 通信 回路 硏究室
(CAD & Communication Circuit Lab., Hanyang Univ.)

接受日字 : 1993年 1月 21日

I. 서 론

1980년대 중반 이후부터 상위 수준 합성(high-level synthesis)은 CAD분야에서 많은 주목을 받았으며, 이에 대한 연구도 활발히 진행되어 많은 발전

을 하고 있다. 상위 수준 합성은 동작 기술로 부터 목적 함수와 제한 조건들을 만족하는 RT 레벨의 구조적인 설계를 자동적으로 생성하는 것이다.

상위 수준 합성은 크게 스케쥴링과 allocation으로 구분할 수 있다. 스케쥴링은 제어 스텝에 연산을 할당하는 것이며 allocation은 연산과 변수를 연산자와 메모리에 할당하고 연산자와 메모리간에 연결 구조(interconnection)를 할당하는 것이다.

스케쥴링 알고리듬은 ASAP[1]가 제안된 후 list 스케쥴링[2], FDS(Force Directed Scheduling)[3], ILP(Integer Linear Programming)[4] 등의 다양한 알고리듬이 제안되었으며, 특히 ILP의 경우 최적의 해를 구하지만 문제의 복잡도가 증가함에 따라 실행시간이 급속히 증가한다는 단점을 가지고 있다.

스케쥴링은 제한 조건에 따라 설계된 칩의 속도와 면적을 결정한다. 만일 동작 속도가 제한 조건이면 스케쥴링 알고리듬은 시간 제한 조건을 만족하면서 최소의 연산들이 동시에 수행되도록 할 것이다. 만일 면적이 제한조건이라면 면적 제한 조건을 만족하면서 최소한의 시간에 동작될 수 있도록 연산들이 연속적으로 수행되도록 할 것이다.

본 논문은 시간 제한 조건에서 면적의 최소화를 목적 함수로 하여 각 연산의 time frame에 대한 구간 그래프를 이용한 스케쥴링 알고리듬을 제안한다. 본 논문에서 제안한 알고리듬의 특징은 연산을 한번에 제어스텝에 할당하지 않고 반복적으로 연산의 time frame을 조절하여 하드웨어 자원의 사용이 최소가 되도록 하였다. 또한 다구간(multi-cycle), 체이닝(chaining) 및 파이프라인형 연산이 하나의 모델에 의해서 쉽게 처리되며, 파이프라인형 및 비파이프라인형 데이타 패스에 대한 스케쥴링이 가능하다. 본 논문의 구성은 2장에서는 본 논문에서 사용하는 함수에 대해 기술하고 3장에서 스케쥴링 알고리듬에 대해 기술한다. 실험 및 결과는 4장에서 기술한다.

Ⅱ. 함수 정의

스케쥴링 알고리듬을 설명하기 전에 알고리듬에서 사용하는 함수들에 대해 정의한다. 구간 그래프의 에지에 부여되는 가중치는 에지에 인접한 두 연산(노드)의 time frame이 중첩되어 두 연산이 중첩된 time frame내에서 동시에 수행될 수 있는 가능성을 의미한다. 따라서 에지가 존재하지 않는 연산은 같은 제어 스텝에서 동작할 수 없거나, 동작하지 않는다는 것을 의미한다. 한편 가중치가 0인 것은 두 연산이 동시에 수행될 가능성이 없는 것을 의미하므로 두 연산에 인접한 에지가 존재하지 않는 것으로 본다. 가중치의 계산은 다음과 같이 한다.

1. 가중치의 계산

같은 형의 두 연산이 한 제어 스텝에서 동시에 수행될 수 있는 확률로서 가중치를 계산한다. 이 때 두 연산 간에는 데이타 종속 관계가 존재하지 않아야 하며 time frame이 중복되어야 한다. 같은 형의 연산 O_i와 O_j에 대한 가중치는 다음과 같이 계산한다.[15]

$$\text{Weight}(i,j) = \frac{\text{OVER}(i,j)}{T_i \times T_j} \quad (1)$$

OVER(i, j) : 연산 O_i와 O_j의 time frame이 중복되는 시간

T_i : 연산 O_i의 time frame, T_j : 연산 O_j의 time frame

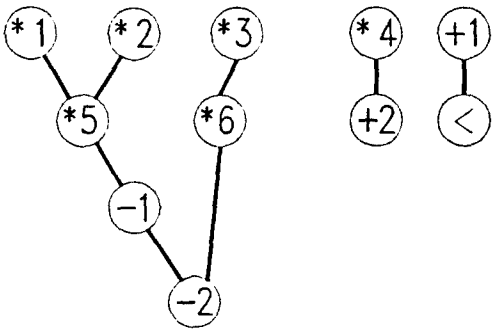


그림 1. 2차 미분 방정식의 CDFG
Fig. 1. CDFG for a 2nd differential equation.

그림 1의 2차 미분 방정식의 CDFG(Control Data Flow Graph)에 대한 time frame이 그림 2와 같이 구해진 경우 연산 *1과 *4간의 가중치는 1 / (1 X 3) = 0.33이 된다.

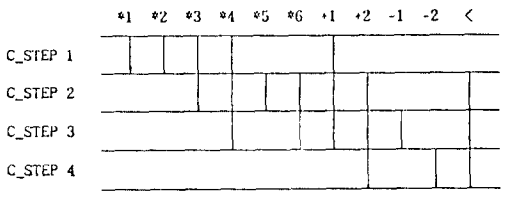
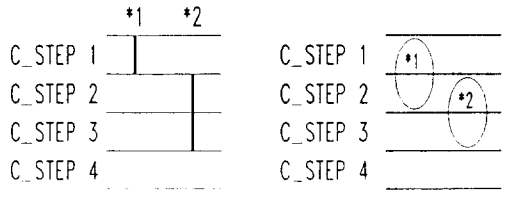


그림 2. 그림 1의 연산에 대한 Time frame
Fig. 2. Time frames of operations in fig. 1.

체이닝 연산의 경우 가중치는 식 (1)에 의해 계산되지만 연산의 동작 시간이 1개의 제어 스텝을 넘어서 동작하는 다주기 연산의 경우 두 연산의 time frame이 중복되지 않으면 식(1)의 계산에 의해 가중치는 0이 되어 3장에서 설명할 구간 그래프 형성시 두 연산 간에는 에지가 존재하지 않게 된다. 따라서 두 연산은 동시에 수행되지 않아야 한다. 그러나 그림 3과 같이 식(1)의 적용으로 두 연산이 동시에 수행되지 않는것처럼 보이지만 실제적으로 두 연산이 동시에 수행되는 경우가 발생할 수 있다. 예를 들면 곱셈연산을 수행하는 데 2개의 제어 스텝이 필요하다고 하자. 식(1)에 의한 가중치에 의해 본 논문에서 제안한 스케줄링 알고리듬을 수행하면 그림 3의 (a)와 같이 연산 *1이 제어 스텝 1에 할당되고 연산 *2는 제어스텝 2또는 3에 할당될 수 있다. 이때 연산 *2는 제어 스텝 2에 할당되고 두 연산간의 가중치는 0이 되어 구간 그래프는 에지가 없는 2개의 노드만 존재하는 그래프가 된다. 이 그래프의 최대 클릭크의 노드 수는 1이 되어 1개의 *연산만이 필요하다고 계산되지만 실제로는 연산의 수행시간이 중복되어 그림 3의 (b)와 같이 2개의 *연산이 제어 스텝 2에서 동시에 수행된다. 따라서 본 논문에서 제안하는 스케줄링 알고리듬에서 다주기 연산을 처리하고자 식(1)을 식(2)와 같이 수정한다. 식(2)에서 는 그림 3과 같은 경우 1의 값을 갖는다. 따라서 그림 3(a)의 경우 *1과 *2간에 구간 그래프는 가중치가 1/(2X3) = 0.16을 갖는 에지가 존재하게 된다.



(a) Time frame (b) 연산의 수행

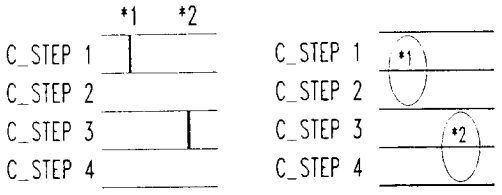
그림 3. 식(1)에 의한 가중치를 적용한 스케줄링
Fig. 3. A scheduling result using weights of equation (1).
(a) Time frames.
(b) Operation execution.

즉 최대 클릭크의 노드 수는 2가 되어 2개의 *연산이 동시에 수행된다고 계산된다. 따라서 두 연산의 에지를 제거하고자 그림 4와 같이 *2 연산을 제어 스텝 3에 할당하여 1개의 *연산이 수행되도록 본 논문에서 제안한 스케줄링 알고리듬이 적용된다. 다주기가 아닌 경우, α는 0이 되지만 다주기의 경우는 (다주기 - 1)의 값을 갖게 된다.

$$\text{Weight}(i,j) = \frac{\text{OVER\_MUL}(i,j)}{(T_i+\alpha)\ X\ (T_j+\alpha)} \qquad (2)$$

OVER_MUL(i, j) : 다주기를 고려하여 연산 O_i와 O_j의 time frame이 중복되는 시간

α : 다주기 연산의 (수행 시간 -1) 또는 파이프라인형 연산의 (latency -1)



(a) Time frame (b) 연산의 수행

그림 4. 식(2)의 가중치를 적용한 스케줄링
Fig. 4. A scheduling result using weights of equation (2).
(a) Time frames.
(b) Operation execution.

2. 연산 형의 선택

스케줄링 시에 면적을 감소 시키기 위하여 이동할 한 쌍의 연산을 선택하게 된다. 예를 들면 그림 1에서 *, +, -, < 의 연산 형 중에서 하나를 선택하여 이 형의 최대 클릭크의 노드 수를 감소시킴으로서 전체 면적을 효율적으로 감소시킬 수 있는가 하는 문제이다. 따라서 연산 형의 선택은 최종 스케줄링 결과에 영향을 주게 된다. 본 논문에서는 다음과 같은 연산 형의 선택 함수를 정의한다. 식(3)에서 α, β는 각 연산 형의 면적 및 수에 따라 사용자가 결정할 수 있다. 예를 들면 +의 최대 클릭크의 노드 수가 9이고 면적이 10, *의 최대 클릭크의 노드 수가 2이고 면적이 40 인 경우에 대해 어느 연산(+, 또는 *)를 선택하여 최대 클릭크의 노드 수를 줄이느냐 하는 것을 결정하기 위해 식 (3)을 적용하면 TA(+, *) = 90 - 80 = 10, DA(+, *) = -30이고 TA(*, +) = 80 - 90 = -10, DA(*, +) = 40 - 10 = 30 이므로 $\alpha = \beta = 1$ 이면 TA + DA 가 최대인 *가 선택된다. 만일 α = 1, β = 0 이면 최대 클릭크의 노드 수를 갖는 연산형 +가 결정되며, α = 0, β = 1이면 면적이 가장 큰 연

산형인 *가 선택된다. 따라서 α 와 β 는 면적과 클릭크의 노드 수에 대한 상대적인 중요성을 의미한다. 면적의 차가 큰 경우는 클릭크의 노드 수를 고려하는 것이 의미가 없으나 면적 차가 크지 않은 경우는 클릭크의 노드 수를 고려한 연산형의 선택을 하기 위해 식 (3)을 제안한다.

T : 연산 형의 집합

예) 그림 1의 경우 T = { *, +, −, 〈 }

T_i : i 형의 연산 예) T_1 = *

$$T_i = \underset{T_j \in T}{MAX}(a \times TA(i,j) + b \times DA(i,j)), T_j \in T \text{ and } T_j \neq T_i \qquad (3)$$

$$0 \le a, b \le 1$$

$$TA(i,j) = (\#\text{of max. clique X Area}) \text{ of } T_i - (\#\text{of max. clique X Area}) \text{ of } T_j$$

$$DA(i,j) = \text{Area of } T_i - \text{Area of } T_j$$

Ⅲ. 스케쥴링 알고리듬

1. 구간 그래프구성

ASAP와 ALAP를 이용하여 각 연산이 스케쥴링될 수 있는 time frame을 구한다. 데이타 종속 관계와 연산 형을 고려하여 연산들의 time frame에 대한 구간 그래프를 구성한다. 그림 2의 time frame에 대한 *의 구간 그래프는 그림 5와 같다. 두 연산이 같은 연산형이고 데이타 종속관계가 없고 time frame이 겹치면 구간 그래프의 대응하는 노드 사이에 에지가 존재한다. 예를 들면 그림 5에서 *3과 *4에는 에지가 존재하나 *3과 *6은 데이타 종속 관계가 있기 때문에 에지가 존재하지 않는다. 그림 5의 구간 그래프에서 에지에 부여된 가중치는 식(1)에 의해 계산된 결과이다.

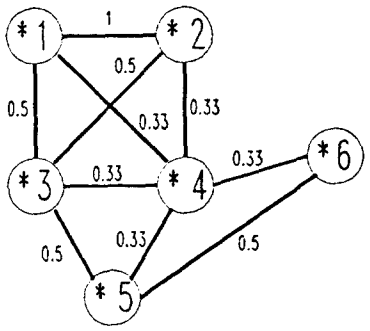


그림 5. "*"에 대한 구간 그래프 (그림 2)

Fig. 5. The interval graph for * in fig. 2.

2. 최대 클릭크를 구하는 알고리듬

3.1절에서 구성한 구간 그래프에서 각 연산 형별로 그림 5와 같은 알고리듬을 적용하여 최대 클릭크를 찾는다.

```
for(각 연산형){
 for(step=first step;step<=last_step ; step++){
  While(선택된 연산형의 모든 연산){
   if(연산의 time frame 범위에 step이 속한다.)
    if(clique를 구성하는 노드와 종속 관계가 없다.)
     현재의 연산을 clique에 포함.
      clique의 크기 증가.
   }
   clique의 크기 비교에 의한 최대 클릭크 설정
  }
}
```

그림 6. 최대 클릭크를 찾는 알고리듬

Fig. 6. An algorithm for finding the maximum clique.

최대 클릭크의 노드 수가 같은 경우 다음 사항을 적용하여 최대 클릭크를 결정한다.

1) 클릭크를 구성하는 연산(노드)중 time frame이 1인 연산의 수가 많은 클릭크

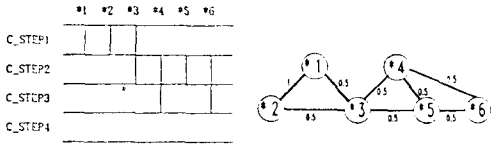
2) 최소 가중치를 갖는 에지가 존재하는 클릭크

3. 최대 클릭크의 노드 수 최소화

구간 그래프의 최대 클릭크의 노드 수는 동시에 수행되는 연산의 수를 의미하므로 최대 클릭크의 노드 수를 작게 하는 것이 면적을 줄이는 것과 같다. 최대 클릭크의 노드 수를 작게 하기 위하여 최대 클릭크에서 최소 가중치를 갖는 에지를 선택하여 이를 제거한다. 이때 최소 가중치를 갖는 에지를 선택하는 것은 가중치 함수의 정의에서 알 수 있듯이 이 에지의 인접(adjacent) 노드가 동시에 수행될 가능성이 작기 때문이다. 그림 5에서 최대 클릭크의 연산 쌍 (1,4), (2,4)와 (3,4)가 같은 가중치인 0.33을 가지게 된다. 이때 연산 쌍들의 가중치가 같기 때문에 두 연산 간의 중복되는 time frame이 작은 쌍인 (1,4)나 (2,4)를 선택한다. 가중치가 같을 때, 중복되는 time frame이 작은 것은 결국 두 연산이 작은 시간 영역(경우의 수)을 이동하여도 두 연산의 time frame이 중복을 피할 수 있다는 것을 의미한다. 따라서 이동할 수 있는 시간 영역이 작은 연산 쌍을 선택하는 것은 다른 연산이 이동할 수 있는 영역을 가능한 크게하여 최대 클릭크의 노드 수를 최소화할 수 있는 가능성을 높이기 위해서이다.

최소 가중치를 갖는 에지의 제거는 선택된 연산의 이동에 의해 동시에 수행될 수 있는 연산의 수를 줄

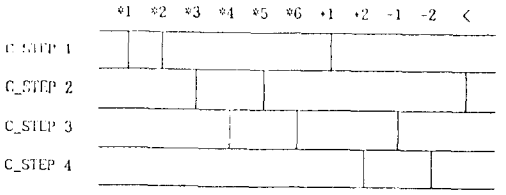
이는 데 있다. 그림 5에서 연산 *4의 time frame을 그림 7의 (a)와 같이 이동하게 되면 그림 7의 (b)와 같은 구간 그래프로 바꾸게 된다.



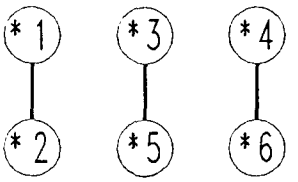
(a) 수정된 time frame (b) 수정된 구간 그래프

그림 7. 수정된 time frame 과 구간 그래프
Fig. 7. The modified time frames and interval graph.
(a) The modified time frames.
(b) The modified interval graph.

에지의 제거는 연산 쌍 중에서 하나의 연산을 선택하여 선택된 연산의 time frame을 조정하는 것으로서 이동할 연산의 선택 및 이동 방향의 선택은 1) 이동할 곳에 있는 같은 형의 연산 수, 2) 연산의 이동에 의한 선행 또는 후행 노드의 time frame 감소를 고려하여 결정한다. 선행 노드 또는 후행 노드의 time frame 감소에 대한 고려는 [16] 에서 정의한 SD (scheduling difficulty)의 개념을 확장 적용하였다. 선택된 연산쌍은 최대 클릭크의 노드 수를 감소 시킬 수 있을 만큼의 시간 영역을 이동한다. 즉 그림 4에서 그림 7의 (a)와 같이 연산 *4가 이동된 것은 *1은 이동할 수 없기 때문이다. 또한 *4 연산의 time frame이 1만큼 감소한 것은 그림 7의 (b)에서 보듯이 이정도의 time frame을 이동하여도 최대 클릭크의 노드 수가 4에서 3으로 감소하기 때문이다. 여기서 선택된 연산을 한번에 고정된 제어 스텝에 할당하지 않고 반복적인 time frame의 조정에 의해 제어 스텝에 할당되도록 한 것은 이미 앞에서 선택된 연산이 고정된 제어 스텝에 할당됨으로서 뒤에 선택된 연산의 제어 스텝 할당에 주는 영향을 줄이기 위해서이다. 그림 7의 (b)에서 최대 클릭크의 노드 수가 같은 완전 그래프가 3개 발생한다. 이 때 최대 클릭크의 선택은 (*1, *2, *3)의 경우와 같이 time frame이 1인 연산을 가장 많이 포함하고 있는 최대 클릭크를 선택한다. 이와 같은 과정을 반복하면 그림 8과 같은 최종 스케쥴링 결과와 구간 그래프를 구할 수 있다. 스케쥴링은 모든 연산이 스케쥴링되거나 각 연산 형 별로 에지 가중치가 1인 최대 클릭크가 생성되면 종료된다. (*를 제외한 다른 연산은 구간 그래프의 클릭크의 노드 수가 1이므로 그림에서 제외하였음)



(a) 최종 time frame



(b) 최종 구간 그래프(*)

그림 8. 최종 time frame과 구간 그래프
Fig. 8. The final time frames and interval graph.
(a) The final time frames.
(b) The final interval graph(*).

4 비파이프라인형 데이타 패스의 스케쥴링 알고리듬

이상에서 설명한 과정을 비파이프라인형 데이타 패스 합성을 위한 스케쥴링 알고리듬에 대해 기술하면 그림 9와 같다.

ASAP와 ALAP로 각 연산의 time frame을 설정
While(unscheduled operation){
- 각 연산 형별로 구간 그래프를 구성
- 식(1)과 식(2)를 이용하여 에지의 가중치를 계산
- 각 연산의 형(type)별로 최대 클릭크의 노드 수를 계산
- 식(3)를 이용하여 연산의 형을 결정
- 결정된 연산 형의 최대 클릭크에서 에지의 가중치가 최소인 노드쌍을 선택
- 선택된 에지를 제거
- Time frame을 수정
}

그림 9. 비파이프라인형 데이타 패스의 스케쥴링 알고리듬
Fig. 9. The proposed scheduling algorithm for the nonpipelined datapath.

5. 파이프라인형 데이타패스의 스케줄링

파이프라인형 스케줄링은 수행 시간과 latency에 의해 Instance의 수가 결정된다. Instance에 대한 정의는 [3] 의 개념을 적용하며 그림 10에서 보여준다. 비파이프라인형 데이타패스 스케줄링 알고리듬을 파이프라인형 데이타패스 스케줄링 알고리듬에 적용하기 위해서는 증가되는 instance에 대한 고려가 있어야 한다. 즉 instance를 별도의 CDFG로 생각하여 각 CDFG로 구성된 forest에 대한 구간 그래프를 구성하여야 한다. 예를 들면 그림 10에서 instance 1에 있는 *6이 비파이프라인형 데이타패스 합성인 경우에는 그림 4와 같이 노드 *1, *2 *3 및 *4로 구성된 최대 클릭크에 포함되지 않으나 파이프라인형 데이타패스 합성인 경우에는 instance 2에 있는 *3과 instance 1에 있는 *6간에는 데이타 종속이 존재하지 않는다고 처리한다. 따라서 instance 2에 있는 *1, *2, *3, *4 와 instance 1에 있는 *4와 *6으로 구성된 최대 클릭크가 형성된다. 즉 같은 instance내에서는 데이타 종속 관계가 유지되지만 그림 10과 같이 instance 1에 있는 *6과 instance 2에 있는 *3과의 사이에는 데이타 종속 관계가 존재하지 않는 것으로 처리하여 클릭크를 구성한다. 따라서 최대 클릭크의 노드 수는 비파이프 라인인 경우에 비해 latency에 따라 증가한다. 그림 10과 위에서 설명한 데이타 종속 관계를 수정하면 3.4 절에서 제안한 비파이프라인형 데이타패스의 스케줄링 알고리듬에 의해 파이프라인형 데이타패스 스케줄링이가능하다.

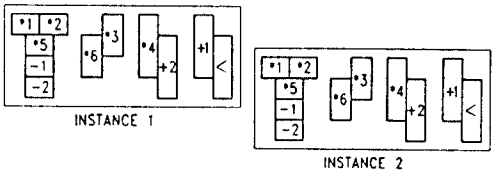


그림 10. 파이프라인형 데이타패스 합성시 instance의 예

Fig. 10. The example of instances for the pipelined datapath synthesis.

Ⅳ. 실험 결과

본 논문에서 제안한 알고리듬은 SUN 4/60에서 C 언어로 구현되었다. 본 절에서는 여러 예를 통하여 알고리듬의 효용성을 보이고자 한다. 각 예제는 참고문헌 [4] 및 [5] 에서 인용하였다. [5] 에서 제시한 예만 수행 시간이 표시되어 있기 때문에 표 3에만 시간 비교를 하였다.

첫번째 예의 경우 1988년 high-level synthesis workshop에서 benchmark로 채택된 4개의 예중 가장 많이 인용되고 있는 fifth-order elliptic filter이다. 표 1의 경우 비파이프라인형 곱셈기 및 덧셈기를 이용하여 스케줄링한 결과를 보여준다. 제어 스텝 18의 경우 FDS는 2개의 곱셈기와 3개의 덧셈기가 필요하지만 본 논문의 결과는 2개의 곱셈기와 2개의 덧셈기가 필요함을 보여주었다. 표 2는 파이프라인형 곱셈기를 이용한 경우의 결과를 보여준다. 표 3은 파이프라인형 데이타패스의 전용 합성 툴인 PISYN과 비교한 결과를 보여주고 있다. 표 3에서는 기존의 툴과 같은 결과를 보여주면서 수행 시간이 짧음을 보여주고 있다. 표 4는 파이프라인형 데이타 패스 합성에 대해 본 논문에서 제안한 알고리듬과 ILP를 이용한 ALPS 및 [7] 에 의한 스케줄링 결과를 비교하였다. 곱셈기 및 덧셈기의 수행 시간은 [3] 에서 제시한 시간인 80ns와 40ns를 적용하였으며 파이프라인형 곱셈기의 경우는 latency를 40ns로 하였다. 본 논문과 ALPS는 클럭 주기를 50ns로 하였으며 [7] 의 경우는 100ns로 하여 스케줄링한 결과이다. 파이프라인형 곱셈기를 사용한 경우는 ILP에 의해 구해진 optimal해와 대부분 일치하는 결과를 얻었다. 그러나 비파이프라인형 곱셈기를 사용한 경우는 delay 및 latency가 작을 때는 최적해를 얻었으나 delay가 커지면서 ILP에 의해 구해진 최적해에 비해 덧셈기를 1개씩 더 사용하고 있다. 이러한 단점은 동시에 연산의 위치를 고려하는 것이 아니고 곱셈기나 덧셈기를 각각 스케줄링하기 때문에 곱셈기는 optimal한 갯수를 구하였지만 이로 인하여 덧셈기의

표 1. 비파이프라인형 곱셈기를 이용한 fifth-order elliptic filter에 대한 실험 결과

Table 1. The experimental result for fifth-order elliptic filter using non-pipelined multiplier.

제어 스텝	17	18	19	20	21
ALPS	+3 *3	+2 *2		+2 *2	+2 *1
FDS	+3 *3	+3 *2	+2 *2		+2 *1
본 논문	+3 *3	+2 *2	+2 *2	+2 *2	+2 *1

표 2. 파이프라인형 multiplier를 이용한 fifth-order elliptic filter에 대한 실험 결과

Table 2. The experimental result for fifth-order elliptic filter using pipelined multiplier.

제어 스텝	17	18	19
ALPS	+3 *2	+3 *1	+2 *1
HAL	+3 *2	+3 *1	+2 *1
본 논문	+3 *2	+3 *1	+2 *1

표 3. FIR filter에 대한 실험 결과

Table 3. The experimental result for FIR filter..

DII	1	2	3	4	5	6
PISYN*	*8	*4	*3	*2	*2	*2
	+15	+8	+5	+4	+4	+3
CPU 시간 (Sec)	9	139	144	142	143	147
본논문**	*8	*4	*3	*2	*2	*2
	+15	+8	+5	+4	+4	+3
CPU 시간 (Sec)	1.8	1.3	1.1	1.3	1.3	1.0

* SUN based system (CPU 시간:스케줄링과 모듈할당을 수행한 시간)

* SUN4/60 (CPU 시간:스케줄링만 수행한 시간)

표 4. fifth-order elliptic filter의 파이프라인 데이타 패스 스케줄링 결과

Table 4. The experimental result for fifth-order elliptic filter with pipelined data path.

(a) Using nonpipelined multiplier

제어 스텝	17	17	17	18	19	19	18	20	21	23	21	33
Latency	1	2	3	4	5	6	7	8	9	13	16	26
ALPS	+26 *16	+13 *8	+9 *6	+7 *4	+6 *4	+5 *3	+4 *3	+4 *2	+3 *2	+2 *2	+2 *1	+1 *1
본 논문	+26 *16	+13 *8	+9 *6	+7 *4	+6 *4	+5 *3	+4 *3	+4 *3	+4 *2	+3 *2	+3 *1	+2 *1

(b) Using pipelined multiplier

제어 스텝	17	17	18	19	19	17	18	20	22	23	33
Latency	1	2	3	4	5	6	7	8	9	13	26
ALPS	+26 *8	+13 *4	+9 *3	+7 *2	+6 *2	+5 *2	+4 *2	+4 *1	+3 *1	+2 *1	+1 *1
본 논문	+26 *8	+13 *4	+9 *3	+7 *2	+6 *2	+5 *2	+4 *2	+4 *2	+3 *1	+2 *1	+2 *1

표 5. MAHA 예에 대한 실험 결과

Table 5. The experimental result for the example of MAHA.

	Control step	opers/c_step	(+)	(-)	CPU time(s)
MAHA*	8	1	1	1	160
	4	3	2	3	160
FDS**	8	1	1	1	50
	4	2	2	2	25
ALPS***	8	1	1	1	0.26
	4	2	2	2	0.08
본논문****	8	1	1	1	0.2
	4	2	2	2	0.2

* VAX 11/750 ** Xerox 1108 LISP machine

*** VAX 11/8800 **** SUN 4/60

스케줄링에 영향을 받기 때문이다. 또한 스케줄링시에 이동할 연산의 선택 및 이동 방향이 최종 결과에 대한 정확한 예측을 할 수 없기 때문에 국부 최소화에 빠지는 문제점을 가지고 있다.

V. 결 론

본 논문에서는 데이타 패스 합성을 위한 새로운 휴리스틱 알고리듬을 제안하였다. ASAP와 ALAP를 이용하여 구한 연산의 life time에 대한 가중치를 갖는 구간 그래프를 구성한다. 구성된 구간 그래프의 최대 클릭크의 노드 수는 동시에 수행될 수 있는 연산의 수와 같으므로 구간 그래프의 최대 클릭크의 노드 수를 감소 시키는 것은 사용되는 하드웨어 자원을 감소 시키는 것과 동일하다. 따라서 구간 그래프의 최대 클릭크 수를 감소시키기 위하여 최소의 가중치를 갖는 두 연산을 선택하여 life time을 조정한다. 반복적인 연산의 이동으로 최소의 자원을 사용하는 스케줄링 결과를 얻는다.

본 논문에서 제안한 알고리듬은 체이닝 연산, 다주기 연산 및 파이프 라인 연산에 적용할 수 있으며 또한 비파이프라인 및 파이프라인형 데이타 패스 합성에 적용할 수 있는 특징을 가지고 있다.

연산의 제어 스텝 할당 시 ILP에 의한 스케줄링과

같이 모든 연산이 할당될 제어 스텝을 동시에 고려하지 못하고 순차적으로 연산의 제어 스텝이 결정되기 때문에 실험 결과 부분적으로 ILP에 의해 구한 최적의 해보다 좋지 않은 결과를 생성하였지만 대부분의 예에서 최적의 스케쥴링 결과를 생성하였다.

參 考 文 獻

[1] M.C. McFarland, A.C. Parker and Raul Camposano, " Tutorial on High-Level Synthesis," *Proc.* 25th DAC, 1988, pp. 330-336

[2] Barry M. Pangrle and Daniel D. Gajski, " Slicer : A State Synthesis for Intelligent Silicon Compilation," ICCD, 1987, pp. 42-45

[3] P.G. Paulin, J.P. Knight and E.F. Girczyn, " HAL : A MULTIPLE-PARADIGM APPROACH TO AUTOMATIC DATA PATH SYNTHESIS," Proc. 23rd DAC, 1986, pp. 263-270

[4] Cheng-Tsung Hwang, Yu-Chin Hsu and Yong-Long Lin, " Optimum and Heuristic Data Path Scheduling Under Resource Constraints," *Proc.* 27th DAC, 1990, pp. 65-70

[5] Raul Camposano and Wayne Wolf, High-Level VLSI Synthesis, Kluwer Academic Pub. 1990, pp. 55-78

[6] Martin Charles Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, 1980

[7] K.S. Hwang , A.E. Casvant, C.T. Chang and M.A. d'Abreu, " SCHEDULING AND HARDWARE SHARING IN PIPELINED DATA PATHS," ICCAD-89, 1989, PP.24-27

[8] Raul Camposano, Reinaldo A. Bergamaschi, "SYNTHESIS USING PATH-BASED SCHEDULING : ALGORITHMS AND EXERCISES, " *Proc.* 27th DAC, 1990, pp. 450-455

[9] Miodrag M. Potkonjak, ALGORITHM FOR HIGH LEVEL SYNTHESIS : RESOURCE UTILZATION BASED APPROACH, PhD thesis, U.C. Berkeley, 1992

[10] N. Park and A.C. Parker, " SEHWA : A PROGRAM FOR SYNTHESIS OF PIPELINES," *Proc.* 23rd DAC, 1986, pp. 454-460

[11] G. Goossens, J. Rabaey, J. Vandewalle and H.D. Man, " An Efficient Microcode Compiler for Applications Specific DSP Processors," *IEEE Trans.* CAD, vol.9, no. 9, Sep. 1990, pp. 925-937

[12] A.C. Parker, T. "T" Pizzaro, M. Mliner, "MAHA : A Program for Datapath Synthesis, " *Proc.* 23rd DAC, 1986, pp. 461-465

[13] D.E. Thomas et al., "The System Architect's Workbench," *Proc.* 25th DAC, 1988, pp. 337-343

[14] F. Brewer and D. Gajski, " Chippe : A System for Constraints Driven Behavioral Synthesis," *IEEE Trans.* CAD, vol. 9, no. 7, Jul. 1990, pp. 681-695

[15] M. Potkonjak and J. Rabaey, " Optimizing resource utilization using transformations," *Proc.* ICCAD, 1991, pp. 88 - 91

[16] M. Potkonjak, "ALGORITHMS FOR HIGH LEVEL SYNTHESIS:RESOURCE UTILIZATION BASED APPROACH," Memorandum no. UCB/ERL M92/10

著 者 紹 介

金 基 鉉(正會員) 會誌 第 20卷 第 11號 參照
현재 한양대학교 전자공학과 박사과정 재학중

鄭 正 和(正會員) 會誌 第 20卷 第 11號 參照
현재 한양대학교 전자공학과 교수