# 파이프라인 데이터패스 합성을 위한 점진적 배정가능범위 축소를 이용한 스케줄링 방법

유 희 진[†] · 오 주 영[††] · 이 준 용[†††] · 박 도 순[††††]


## 요 약

본 논문은 자원제약 조건에서 파이프라인 데이터패스 합성을 위한 스케줄링 방법이며, 우선순위 함수를 사용하여 스케줄할 연산을 선택하는 방법들과는 달리 연산들의 배정가능범위를 점진적으로 축소하여 스케줄한다. 제안방법은 스케줄링 알고리즘과 자원제약 위반을 검출하는 판단 알고리즘으로 구성되며, 연산의 배정 가능한 제어단계의 처음 또는 마지막 단계에 임시로 연산을 배정하여 스케줄링 해가 존재하는지를 평가한다. 만약 해를 발견할 수 없다면 이는 자원제약 위반에 의해 연산을 그 제어단계에 배정하는 것이 불가능함을 의미하기 때문에 그 제어단계를 제거하며, 모든 연산에 대하여 배정가능범위 축소가 없을 때까지 이 과정을 반복한다. 벤치마크에 대한 실험결과는 다른 방법들과 비교해서 개선된 스케줄링 결과를 보인다.


# A Scheduling Approach using Gradual Mobility Reduction for Synthesizing Pipelined Datapaths

Heejin Yoo[†] · Juyoung Oh[††] · Junyong Lee[†††] · Dosoon Park[††††]


## ABSTRACT

This paper presents a scheduling approach for synthesizing pipelined datapaths under resource constraints. Our approach builds up a schedule based on gradual mobility reduction in contrast to other algorithms of previous researches, where an operation being scheduled is selected by using a priority function. The proposed method consists of a scheduling algorithm and a decision algorithm for detecting any violation against resource constraints. Our approach evaluates whether or not a scheduling solution can exist in case an operation temporarily is assigned to the earliest or latest control step among the assignable steps for the operation. If a solution cannot be found, it is impossible to assign the operation to that step due to a violation against resource constraints, and so we can eliminate that control step. This process is iterated until a reduction of mobility for all operations can not be obtained. Experiments on benchmarks show that this approach gains a considerable improvement over those by previous approaches.


**키워드 : 상위수준합성(high level synthesis), 스케줄링(scheduling), 파이프라인 데이터패스(pipelined datapaths)**

## 1. 서 론

상위 수준 합성(HLS : High Level Synthesis)은 설계하려는 하드웨어의 동작기술을 입력하여 스케줄링과 할당 과정에 의해 레지스터 전송 수준(RTL : Register Transfer Level)의 데이터패스와 컨트롤러를 생성하는 과정이다. 스케줄링은 중간그래프 형태인 CDFG(Control/Data Flow Graph)로 표현되어 있는 연산들을 주어진 제약조건을 만족하면서 최적의 제어단계에 배정하는 과정이며, 제약조건은 시간과 자원 등이 된다. 파이프라인 데이터패스의 스케줄링은 NP-complete[1, 2]문제이며, 최적의 해를 찾기 위해 리스트 스케줄링, 확률 기반 스케줄링, 변환 기반 스케줄링 등의 휴리스틱 알고리즘이 연구되어 왔다[1, 3].

기존의 휴리스틱 알고리즘들은 우선순위 함수를 정의하고, 우선순위 함수 값에 따라 최적의 해에 단계적으로 접근하는 방법을 사용한다. 즉, 각 단계마다 모든 연산을 그 연산의 배정 가능한 제어단계들에 배정하는 경우의 상황을 함수로 표현하여 가장 최적의 해가 될수 있는 제어단계에 그 연산을 배정하여 스케줄한다. 그 결과로 하나의 연산이 선택되어 제어단계에 배정될때 이 제어단계로의 배정이 최적의 해가 아닐 수 있으므로 항상 최적의 해를 찾는다는 보장이 없다.

본 논문에서는 자원제약하에서 합성되는 파이프라인 데이

† 정 회 원 : 순천제일대학 컴퓨터과학부 교수
†† 준 회 원 : 경인여자대학 전임강사
††† 정 회 원 : 홍익대학교 컴퓨터공학과 교수
†††† 종신회원 : 홍익대학교 컴퓨터공학과 교수
논문접수 : 2001년 9월 14일, 심사완료 : 2002년 6월 18일

터패스의 실행시간을 최소화하는 것을 목적으로 하여, 연산의 제어단계 배정을 위해 우선순위 함수를 사용하지 않고 최종 해에 접근할 수 있는 방법을 제안한다. 제안 방법은 연산의 배정 가능한 제어단계들중에서 하나의 제어단계에 그 연산을 배정하는 경우에 다른 연산들이 주어진 제어단계들 내에서 자원충돌을 해결할 수 있는지를 판단하여 스케줄한다. 이 경우에 주어진 제어단계들내에서 자원충돌을 해결할 수 없으면 그 제어단계로의 배정은 최종 해를 찾을 수 없으므로 그 제어단계를 제거 즉 배정가능범위를 축소하는 방법으로 최종 해에 접근한다. 제안 방법은 스케줄링 알고리즘과 자원 배정 가능 판단 알고리즘으로 구성되며, 기능적 파이프라인을 지원할 뿐만 아니라 실질적인 하드웨어 합성을 위해 멀티사이클링, 체이닝, 그리고 연산장치의 구조적 파이프라인을 지원한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 살펴보고, 3장에서는 제안한 파이프라인 스케줄링 알고리즘과 자원 배정 가능 판단 알고리즘을 설명한다. 4장에서는 벤치마크를 사용하여 제안 알고리즘의 성능을 평가하고, 5장에서 결론을 기술한다.

## 2. 관련 연구

정수 선형 프로그래밍[4]은 제약조건과 목적함수를 수식으로 모델링하여 제약조건을 만족하는 최적의 스케줄링 결과를 생성한다. 그러나 스케줄할 연산의 수와 제약조건이 증가하면 변수의 수가 증가하여 해를 구하는 시간이 많이 걸리는 문제점이 있다. 보통은 이러한 NP-complete문제를 해결하기 위하여 우선순위 함수를 정의하여 한 번에 하나의 연산을 스케줄하며, 모든 연산이 스케줄될때까지 반복하는 휴리스틱 방법을 사용한다. Sehwa[5]는 스케줄할 연산들을 우선순위 함수에 따라 리스트 스케줄링한 후에 자원제약을 위반하지 않는 범위 내에서 각 제어단계에 연산을 배정한다. 우선순위 함수는 순방향 긴급도 또는 역방향 긴급도를 사용하며, 순방향 긴급도는 노드로부터 출력까지의 가장 긴 패스의 길이로 정의하고 역방향 긴급도는 입력으로부터 노드까지의 가장 긴 패스의 길이로 정의한다. HAL[6] 시스템은 서로 다른 제어단계에 사용되는 자원은 공유될 수 있다는 개념하에 연산장치, 레지스터, 연결구조의 사용을 제어단계에 균등히 배분하는것을 목표로 한다. 한 연산을 배정 가능 범위의 한 제어단계에 배정하면 전체적인 병렬성이 변하는데, 이 병렬성의 반발정도를 힘(force)으로 표현하여 시간제약조건에서는 FDS(Force Directed Scheduling), 자원제약조건에서는 FDLS(Force Directed List Scheduling)을 사용한다. IFDS[7]는 연산의 전체 배정가능범위에 대한 힘 계산을 연산의 처음 제어단계와 마지막 제어단계에서의 두 번의 힘 계산으로 축소하여 FDS 알고리즘의 시간복잡도를 감소하였다. Choi[8]는 임계경로를 조사하여 자원제약에 의한 자원충돌이 발생하지 않도록 지연 연산자를 삽입하고, 연산의 분포가 균등하고 연산의 활용도가 높은 연산의 종류를 선택한 후에, 연산의 밀집도가 최소인 파티션에 힘 값이 최소인 연산을 배정한다. PLS[9, 10]는 LCD(loop carried dependency)를 고려하면서 전진 스케줄링과 후진 스케줄링을 사용하여 이미 배정된 연산을 재배정하는 반복적인 방법으로 루프의 반환시간을 줄였다. Folding 스케줄링[11]은 자원제약조건을 위반하는 제어단계에 존재하는 연산들중에서 임계경로에 있지 않은 연산을 재스케줄할 후보 연산으로 선택하여 각 후보 연산들을 각각 미리 보기(look ahead) 스케줄링을 한 후에, 우선순위 함수인 변화도(variability)를 사용하여 평균 하드웨어 비용을 계산하고 비용이 가장 적은 후보 연산을 선택하여 스케줄한다. Hwang[12]은 하드웨어 비용을 최소화할 목적으로 각 파티션에 연산을 균등히 분배하여 자원공유가 최대화되도록 각 연산의 종류별로 연산 밀집도가 가장 적은 파티션에 배정가능범위가 가장 작은 연산을 배정한다. 이러한 휴리스틱 알고리즘들은 정의된 우선순위 함수에 따라 한 번에 하나의 연산을 스케줄링하여 모든 연산이 스케줄될때까지 반복한다.

본 논문에서는 하드웨어 동작을 표현하는 DFG(data flow graph)와 사용 가능한 자원의 개수를 입력으로 하여 최소의 파이프라인 데이터 입력 간격을 구하고, 이 최소 데이터 입력 간격에서 우선순위 함수를 사용하지 않고 합성되는 파이프라인 데이터패스의 실행시간 최적화를 목적으로 하는 자원 제약 휴리스틱 스케줄링 알고리즘을 제안한다.

## 3. 파이프라인 스케줄링

스케줄링은 행위 모델의 모든 연산들을 서로의 종속성을 유지하면서 주어진 제어단계 내에 배열하는 것이다. 스케줄링을 할때에는 제어단계의 수, 지연시간, 전력 소모, 하드웨어 자원 등의 여러 가지 제약조건들이 요구되지만, 보통은 시간 또는 자원중에 하나를 제약조건으로 하여 다른 하나를 최소화한다. 본 논문은 하드웨어 자원을 제약조건으로 제어단계의 수를 최소화하는 것을 목적으로 한다. 연산들의 처음 배정 가능한 제어단계들의 범위는 DFG로부터 ASAP(As Soon As Possible) 스케줄링과 ALAP(As Late As Possi-

ble) 스케줄링 결과로부터 구할 수 있다. 스케줄링 과정중에 연산들의 배정가능범위가 수정되면 그 연산들과 종속관계에 있는 다른 연산들의 배정가능범위가 수정되어 연산들의 종속성이 항상 유지되어야 한다.

자원제약조건의 비파이프라인 스케줄링에서는 각 제어단계에 배정되는 연산의 수만을 고려하지만, 파이프라인 스케줄링에서는 동시에 실행될 수 있는 제어단계들의 집합인 파티션(partition)을 고려하여야 한다. 파이프라인에서의 데이터 입력 간격은 스케줄링에 중요한 변수로서 DFG내의 연산의 개수, 사용 가능한 하드웨어 자원의 개수, 그리고 하드웨어 자원들의 지연시간에 의해서 결정된다. DFG에서 $k$유형의 연산의 수를 $N_k$, $k$유형의 사용 가능한 자원의 개수를 $M_k$, 그리고 $k$유형자원의 지연시간을 $D_k$라 가정할 때에 $k$유형과 관련된 최소 데이터 입력간격은 식 (1)과 같다.

$$\lceil \frac{N_k}{M_k} \rceil * D_k \qquad (1)$$

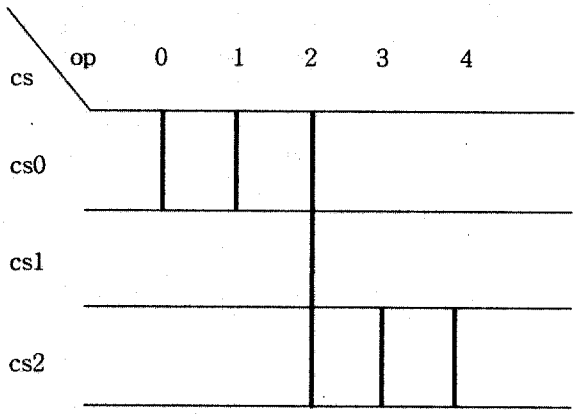
한편 $m$개 유형의 연산들이 존재하는 경우에는 최소 데이터 입력간격은 식 (2)와 같다[8, 10].

$$\max_{1 \le k \le m} \lceil \frac{N_k}{M_k} \rceil * D_k \qquad (2)$$

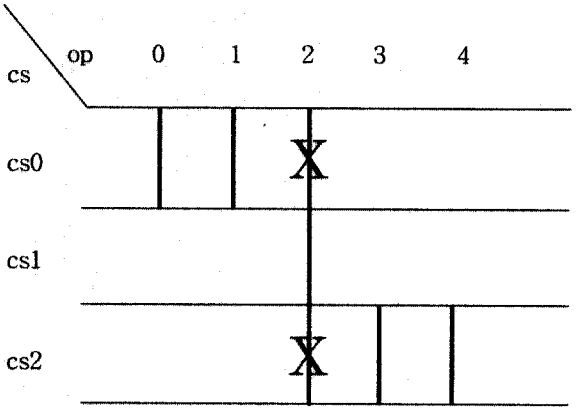
### 3.1 스케줄링 알고리즘

연산의 배정가능 범위중에서 처음 제어단계와 마지막 제어단계는 연산들 상호간의 종속성에 의한 연산의 배정 가능 제어단계 범위의 상한과 하한을 나타낸다. 스케줄링할 때에 연산들의 종속성 유지와 자원제약은 항상 만족되어야 하며, 하나의 제어단계에서 연산에 대한 자원 배정이 자원제약조건을 위반하는 경우에는 그 제어단계에 그 연산의 배정은 될 수 없기 때문에 그 제어 단계를 제거하는데, 이때 이 연산과 종속 관계에 있는 연산들의 배정 가능 범위도 함께 연쇄적으로 축소되어 종속성이 유지된다.

(그림 1)은 제안 알고리즘의 스케줄링 원리이다. 5개의 연산은 모두 같은 유형의 연산이고, 사용 가능한 자원의 개수는 2개로 가정한다. (그림 1)(a)에서 연산 2의 배정가능범위는 제어단계 0에서 제어단계 2인데, 연산 2의 처음 제어단계인 제어단계 0에 이 연산을 배정한다면 자원제약 때문에 스케줄링이 불가능하므로 (그림 1)(b)에 표현된 것과 같이 배정가능범위를 축소한다. 또한 연산 2의 마지막 제어단계인 제어단계 2에 이 연산을 배정하는 경우에도 자원제약 때문에 스케줄링이 불가능하므로 점진적으로 배정가능범위를 축소하면 스케줄이 가능하다.




(a) 연산의 배정가능범위




(b) 스케줄링 결과

(그림 1) 스케줄링 원리

(그림 2)는 제안한 스케줄링 알고리즘이며, 연산들의 배정가능범위들을 가지고 연산들의 종속성을 유지하면서 자원의 배정 가능을 다음절에서 설명되는 자원 배정 가능 판단 알고리즘에 의해 조사하여 자원 배정이 불가능하면 그 연산의 배정가능범위를 축소하는 방법으로 스케줄한다. 즉 단계 4에서 한 연산의 처음 제어단계에 대해 그 연산에 대한 자원 배정이 자원제약을 만족할 수 없다면 그 제어단계로 연산을 배정할 수 없으므로 그 제어단계를 제거한다. 물론 이 연산과 종속되는 연산들의 배정 가능 제어 단계들도 함께 축소된다. 이때 연산의 처음 제어단계가 마지막 제어단계보다 커지면 연산의 현재 배정 가능 제어단계 범위로는 스케줄링이 불가능하므로 제어단계 수를 증가한 후에 단계 4를 반복한다. 그리고 그 연산의 마지막 제어단계에 대해서도 같은 작업을 반복하여 제어단계 축소여부를 결정한다. 알고리즘은 모든 연산에 대하여 각 연산의 처음 제어단계와 마지막 제어단계가 더 이상 축소되지 않을 때까지 반복한다. 축소된 처음 제어단계와 마지막 제어단계가 같으면 스케줄링을 종료하고, 그렇지 않으면 축소된 배정가능범위 내에서 연산의

자원배정이 이루어져 스케줄이 될수 있으므로 각 연산의 처음 제어단계가 스케줄링 결과가 된다.

```
단계 1 최소 데이터 입력 간격 계산
단계 2 ASAP,ALAP 스케줄링으로 각 연산들의 초기 배정 가능 제어 단계의 범위 계산
단계 3 DFG에서 임계경로를 찾음
    if ( 임계경로상에서 자원 충돌 해결 불가능 )
    then 제어단계의 수를 1증가하여 초기 배정 가능 제어 단계 범위 재계산
단계 4 모든 연산 op_i에 대해 다음을 반복
  단계 4.1 연산의 처음 배정 가능 제어 단계에 자원을 임시 배정
  단계 4.2 if ( 자원 배정 가능 )
        then 연산의 처음 배정 가능 제어 단계에 자원 배정
        else 연산의 처음 배정 가능 제어 단계를 제거
            if ( 연산의 처음 배정 가능 제어 단계 > 마지막 배정 가능 제어 단계 )
            then 제어 단계의 수를 1 증가하고 연산들의 배정 가능 제어 단계 범위를 재계산한 후에 단계 4로 이동
  단계 4.3 연산의 마지막 배정 가능 제어 단계에 자원을 임시 배정
  단계 4.4 if ( 자원 배정 가능 )
        then 연산의 마지막 배정 가능 제어 단계에 자원 배정
        else 연산의 마지막 배정 가능 제어 단계를 제거
            if ( 연산의 처음 배정 가능 제어 단계 > 마지막 배정 가능 제어 단계 )
            then 제어 단계의 수를 1 증가하고 연산들의 배정 가능 제어 단계 범위를 재계산한 후에 단계 4로 이동
단계 5 if ( 모든 연산의 처음 배정 가능 제어 단계 = 마지막 배정 가능 제어 단계 )
    then 스케줄링 종료
    else 각 연산의 처음 제어 단계가 스케줄 결과
```

(그림 2) 스케줄링 알고리즘

### 3.2 자원 배정 가능 판단 알고리즘

연산에 대한 자원 배정 가능 판단 알고리즘은 파티션에 이미 자원이 배정된 연산들과 그 연산들의 배정 가능한 제어단계들 그리고 파티션에 배정할 연산을 입력으로 하여 연산들의 종속성을 만족하며 배정가능범위내에서 자원제약조건을 유지할 수 있는지를 판단한다. 한 파티션에서 연산이 자원 배정을 요구할때에 파티션에 아직 배정되지 않은 자원이 존재하면 그 연산에 자원을 배정한다. 그러나 이미 모든 자원을 배정한 파티션에 새로운 연산이 자원 배정을 요구하면, 해당 파티션에는 사용 가능한 자원이 없기 때문에 자원충돌이 발생하는데, 다음의 방법으로 연산의 자원 배정 가능 여부를 판단한다. 여기에서 배정 가능한 확장된 파티션 집합 P는 자원 충돌이 있는 파티션내의 연산들의 다음 파티션으로 이동, 그리고 이 이동에 의한 연산들의 연쇄적인 이동에 의해 자원 배정을 할수 있는 파티션들로 정의한다.

연산 $n$에 대한 배정 가능한 제어단계들의 파티션 집합을 $M_n$이라 하고, 연산 $n$의 배정 가능한 확장된 파티션 집합을 $P_n$이라 하면, 연산 $n$에 대한 초기 $P_n$은 $P_n = M_n$이다. $M_n$내의 파티션들을 $i, j, k, \cdots$라 할때에 파티션 $i$에 배정된 연산들을 $op_{i1}, op_{i2}, \cdots$, 파티션 $j$에 배정된 연산들을 $op_{j1}, op_{j2}, \cdots$, 파티션 $k$에 배정된 연산들을 $op_{k1}, op_{k2}, \cdots$라 가정하자. 그러면 이 연산들의 배정 가능한 파티션 집합은 $M_{i1}, M_{i2}, \cdots, M_{j1}, M_{j2}, \cdots, M_{k1}, M_{k2}, \cdots$이 된다. 파티션 집합 $M_{i1}, M_{i2}, \cdots, M_{j1}, M_{j2}, \cdots, M_{k1}, M_{k2}, \cdots$은 자원충돌이 발생한 파티션에서 다음 파티션으로 자원 배정을 양보할 수 있는 연산들과 그 연산들의 배정 가능한 제어단계를 의미한다. 그래서 연산 $n$의 배정 가능한 확장된 파티션 집합은 $P_n = M_n \vee M_{i1} \vee M_{i2} \vee \cdots \vee M_{j1} \vee M_{j2} \vee \cdots \vee M_{k1} \vee M_{k2} \cdots$이 되는데, $P_n$은 더이상 확장되지 않을때까지 반복하여 생성된다. 한 파티션에 대한 연산의 자원 배정 요구에 대해, $P_n$내의 파티션에 할당된 자원 중에 아직 배정되지 않은 자원이 존재한다면 자원 배정 양보에 의해 자원충돌을 해결할 수 있다. 그러나 $P_n$내의 파티션에 할당된 자원이 모두 이미 배정되어 있는 경우라면 자원충돌은 해결할 수 없다.

<표 1>은 연산의 자원 배정 가능 판단의 예를 보여준다. <표 1>(a)는 같은 유형의 자원을 사용하는 5개 연산의 배정가능범위를 나타낸다. <표 1>(b)는 사용 가능한 자원이 2개로 제약된 경우에 각 파티션으로 4개의 연산에 대한 자원 배정 상태이다. 이때 연산 $op_4$가 파티션 $P_0$에 자원 배정을 요구하면 자원 충돌이 발생하는데 연산 $op_4$의 배정 가능한 제어단계들의 파티션 집합 $M_4$은 $\{P_0\}$이고, 연산 $op_4$의

〈표 1〉 연산의 자원 배정 가능 판단 예

| 연산 / 배정가능 제어단계 | $OP_0$ | $OP_1$ | $OP_2$ | $OP_3$ | $OP_4$ |
|---|---|---|---|---|---|
| 처 음 | 0 | 0 | 1 | 1 | 0 |
| 마지막 | 1 | 0 | 2 | 1 | 0 |

(a) 연산에 대한 자원 배정 가능 제어단계 범위

| 자원 / 파티션 | $R_1$ | $R_2$ |
|---|---|---|
| $P_0$ | $OP_0$ | $OP_1$ |
| $P_1$ | $OP_2$ | $OP_3$ |
| $P_2$ | | |

(b) 연산의 자원 배정

션 $P_0$에 이미 배정된 연산들이 $\{op_0, op_1\}$이므로 파티션 집합 배정 가능한 확장된 초기 파티션 집합 $P_4$은 $\{P_0\}$이다. 파티션 집합 $P_4$은 $\{P_0, P_1\}$으로 확장되며, 이 과정을 반복하면 $P_4$은 $\{P_0, P_1, P_2\}$로 확장된다. 그러므로 연산 $op_4$는 연산 $op_2$를 파티션 $P_2$로, 연산 $op_0$을 파티션 $P_1$으로 이동한 후에 파티션 $P_0$에 자원 배정을 할 수 있다.

제안한 자원 배정 가능 판단 알고리즘은 (그림 3)과 같다. 알고리즘은 모든 연산들의 처음 제어단계에 해당하는 파티션에 자원을 배정한다. 만약 자원제약 때문에 파티션에서 자원충돌이 발생하면, 그 파티션내의 연산들중에서 자원 배정의 여유가 가장 클 가능성이 있는 연산을 다음 파티션으로 이동하여 재배정한다. 이는 다음 자원충돌시에도 자원충돌의 해결 가능성을 높이기 위한 것이다. 자원 배정의 여유는 식 (3)과 같이 정의한다.

$$M(op_i) - Average(M(successors(op_i))) \qquad (3)$$

식 (3)에서 $M(op_i)$는 연산 $op_i$의 배정가능범위를 나타내고, $M(successors(op_i))$는 연산 $op_i$와 같은 유형의 후행연산들의 배정가능범위를 나타낸다. 자원 배정의 여유는 배정가능 범위가 큰 것이 여유도 크지만, 자원충돌시에 다음 파티션으로 이동되는 연산은 연산의 종속성 유지를 위하여 후행연산을 연속적으로 이동해야할 가능성이 있다. 후행연산들의 연속적인 이동은 연산이 자원을 배정 받을 수 있는 확률에 반비례하기 때문에 후행연산들의 배정가능범위의 평균을 감하여 계산한다. 연산의 자원충돌은 이동되는 연산과 같은 유형의 연산들간에 발생하기 때문에 같은 유형의 후행연산들만 고려한다.

```
자원배정가능판단( )
 {
 for (모든 연산 op_i )
  {
  if (연산 op_i의 처음 제어 단계에 해당하는 파티션에 가용 자원
     존재)
    then return 가능 ;
    else if ( yield(연산 op_i의 처음 제어 단계에 해당하는 파티션
                내에서 자원 배정 여유가 가장 큰 연산) )
          then return 가능 ;
          else return 불가능 ;
   }
 }

yield( op_i )
{
  yield(같은 유형의 모든 후행 연산들) ;
  if ( op_i의 처음 제어 단계 = op_i의 마지막 제어 단계 ) return
     불가능 ;
  op_i의 처음 제어 단계를 1증가 ;
  if (연산 op_i의 처음 제어 단계에 해당하는 파티션에 가용 자원
     존재)
   then return 가능 ;
   else if ( yield (연산 op_i의 처음 제어 단계에 해당하는 파티션
                내에서 자원 배정 여유가 가장 큰 연산) )
         then return 가능 ;
         else return 불가능 ;
  }
}
```

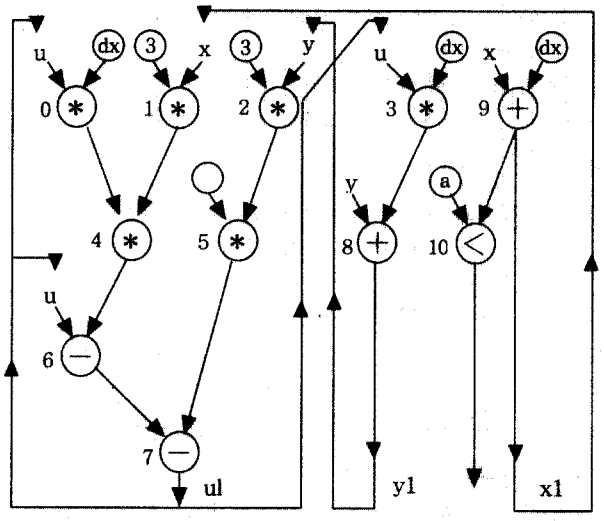
(그림 3) 자원 배정 가능 판단 알고리즘

### 3.3 실행시간

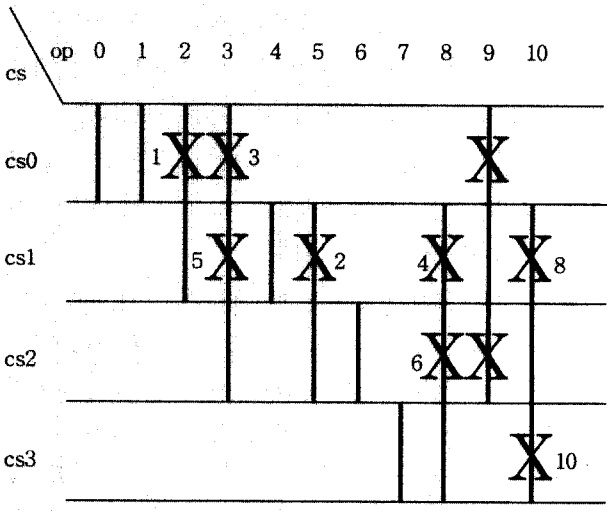
스케줄링 알고리즘은 모든 연산에 대해서 연산의 처음 제어단계와 마지막 제어단계에 자원을 임시로 배정한 후에 자원 배정 가능 판단 알고리즘을 호출한다. 제안 알고리즘의 실행시간 계산을 위하여 연산의 수를 $n$, 연산의 평균 배정가능범위를 $m$이라고 가정한다. 자원 배정 가능 판단 알고리즘은 최악의 경우에 모든 연산의 배정 가능한 제어단계들의 범위만큼 수행하기 때문에 실행시간은 $O(nm)$이다. 그리고 스케줄링 알고리즘은 모든 연산들이 하나의 제어단계에 배정될때까지 자원 배정 가능 판단 알고리즘을 반복 호출하기 때문에 $O(n^2m^2)$의 실행시간을 필요로 한다.

## 4. 실험 결과

제안한 스케줄링 알고리즘은 PC환경에서 C++ 언어로 구현하였으며, 성능 평가를 위해 미분방정식, 5차 엘립틱 웨이브 필터, 그리고 16 포인트 FIR 필터 벤치마크를 사용하였다. (그림 4)는 미분방정식의 스케줄링 과정을 보여주는데, (그림 4)(a)는 미분 방정식 $y'' + 3xy' + 3y = 0$의 해를 수치해석적으로 구하는 DFG 표현이다. 사용 가능한 자원은 곱셈 연산을 수행하는 곱셈기 2개, 비교, 덧셈, 뺄셈 연산을 수행하는 덧셈기 2개로 제약한다. 그리고 곱셈기와 덧셈기의 지연시간을 1 클럭사이클로 가정한다. 그러면 파이프라인의 최소 데이터 입력 간격은 식 (2)에 의해 3클럭사이클이 된다. (그림 4)(b)는 ASAP과 ALAP 스케줄링에 의한 DFG내의 연산들의 초기 배정가능범위를 나타낸다. (그림 4)(c)는 제안 방법에 의한 스케줄링 과정을 보여주는데, X 표시와 숫자는 연산의 자원 배정 불가능에 따른 배정가능범위 축소와 연산들의 종속성에의해 제거되는 제어 단계들의 순서를 나타낸다. 이중에서 자원 배정 불가능에의한 축소는 번호 1, 3, 5, 7, 9들이고, 이 축소에 의해 연산들의 종속성에 기인하여 제




(a) DFG




(c) 스케줄링 과정

| 배정가능 제어단계 \ 연산 | $OP_0$ | $OP_1$ | $OP_2$ | $OP_3$ | $OP_4$ | $OP_5$ | $OP_6$ | $OP_7$ | $OP_8$ | $OP_9$ | $OP_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 처 음 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 3 | 1 | 0 | 1 |
| 마지막 | 0 | 0 | 1 | 2 | 1 | 2 | 2 | 3 | 3 | 2 | 3 |

(b) 배정가능범위

(그림 4) 미분방정식의 스케줄링 과정

거되는 것들은 번호 2, 4, 6, 8, 10들이다.

<표 2>는 임의의 지연시간을 갖는 자원과 자원의 구조적 파이프라인 지원 여부에 따른 미분방정식의 스케줄링 결과를 나타낸다. 여기에서는 덧셈기의 지연시간은 1 클럭사이클이고, 곱셈기의 지연시간은 2 클럭사이클로 가정한다. 파이프라인의 경우는 곱셈기가 매 클럭사이클마다 데이터 입력을 받을 수 있도록 구조적 파이프라인으로 구성된 경우이다. 자원제약에 따른 파이프라인의 데이터 입력 간격은 식 (2)에 의해서 계산하며, 스케줄링 결과는 미분방정식 실행에 필요한 제어단계의 수이다. 이 결과는 모든 자원제약에 대하여 최적의 해가 된다.

〈표 2〉 미분 방정식의 스케줄링 결과

| # of multipliers | | 6 | 4 | 4 | 3 | 3 | 2 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| # of adders | | 5 | 3 | 2 | 2 | 1 | 2 | 1 | 1 |
| latency | | 1 | 2 | 3 | 3 | 5 | 3 | 5 | 6 |
| # of control step | non-pipelined unit | 6 | 6 | 6 | 6 | 7 | 7 | 8 | 13 |
| | pipelined unit | 6 | 6 | 6 | 6 | 6 | 7 | 6 | 8 |

<표 3>은 자원제약에 따른 5차 디지털 엘립틱 웨이브 필터의 스케줄링 결과를 나타낸다. 곱셈연산은 2 클럭사이클에 걸쳐 수행되는 멀티사이클 연산파이프라인 구조로 구조적 파이프라인을 가정한다. 그리고 덧셈연산의 지연시간은 1 클럭사이클로 가정한다. 제안 알고리즘은 자원제약이 {(*, +) : (1, 1)}인 경우를 제외하면 정수 선형 프로그램과 같은 최적의 스케줄링 결과를 보인다. 이는 연산의 자원충돌을 해결하기 위하여 다음 파티션으로 재배정할 연산 선택을 위한 식 (3)의 계산에서 후행연산들의 서로 중복되는 배정가능범위에 대한 고려가 부족하기 때문인 것으로 판단된다.

〈표 3〉 5차 엘립틱 웨이브 필터 스케줄링 결과

| # of multipliers | | 8 | 4 | 3 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # of adders | | 26 | 13 | 9 | 7 | 6 | 5 | 4 | 4 | 3 | 2 | 1 |
| latency | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 13 | 26 |
| # of control steps | ILP[5] | 17 | 17 | 18 | 19 | 19 | 17 | 18 | 20 | 22 | 23 | 33 |
| | PLS[5] | 17 | 17 | 18 | 19 | 19 | 17 | 18 | 20 | 22 | 23 | 33 |
| | Sehwa[5] | 17 | 17 | 18 | 19 | 20 | 21 | 20 | 22 | 23 | 24 | 33 |
| | 제안 방법 | 17 | 17 | 18 | 19 | 19 | 17 | 18 | 20 | 22 | 23 | 34 |

<표 4>는 자원제약에 따른 16 포인트 디지털 FIR 필터의 스케줄링 결과를 나타낸다. 곱셈연산의 지연시간은 1 클럭사이클로 가정한다. 그리고 지연시간이 짧은 덧셈연산은 하나의 제어단계에서 두 개의 연산을 실행하는 체이닝을 지원한다. 제안 알고리즘은 다른 연구들과 비교하여 최적의 스케줄링 결과를 보이고 있다.

〈표 4〉 16 포인트 FIR 필터 스케줄링 결과

| # of multipliers | | 8 | 4 | 3 | 3 | 2 | 2 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| # of adders | | 15 | 8 | 6 | 5 | 4 | 3 | 2 | 1 |
| latency | | 1 | 2 | 3 | 3 | 4 | 5 | 8 | 15 |
| # of control steps | Sehwa[5] | | | 7 | 6 | | | | |
| | FDS[6] | | | | 6 | | | | |
| | Choi[8] | | 6 | | 6 | 6 | 7 | | |
| | Lee[11] | 6 | 6 | 6 | 6 | 6 | 7 | 10 | 16 |
| | Proposed method | 6 | 6 | 6 | 6 | 6 | 6 | 10 | 15 |

## 5. 결 론

본 논문에서는 자원제약조건을 갖는 파이프라인 데이터패스 합성을 위한 스케줄링 알고리즘을 제안하였다. 제안 알고리즘은 해에 접근할 수 없는 제어단계를 찾아 배정가능범위를 단계적으로 축소하여 최적의 해에 접근하였다. 해로의 접근은 연산의 종속성과 자원제약조건을 만족하며 연산의 배정가능범위내에서 자원 배정이 가능한지에 대한 판단에 의하여 결정하였으며, 다른 연구들과 비교하여 개선된 스케줄링 결과를 얻을 수 있었다.

향후 알고리즘의 성능 개선을 위하여 첫째, 파티션내 자원충돌을 해결하기 위하여 다음 파티션으로 재배정할 연산의 선택에서 후행연산들의 서로 중복되는 배정가능범위 계산에 대한 개선 방법과 둘째, 행위 수준 모델에서 사용 가능한 문장 구조를 지원하도록 확장하는 것을 연구과제로 한다.

## 참 고 문 헌


[1] Y. C. Hsu and Y. L. Jeang, "Pipeline Scheduling Techniques in High-Level Synthesis," in Proc. Int. Conf. Computer-Aided Design, pp.396-403, 1993.

[2] G. D. Micheli, "Synthesis and Optimization of Digital Circuits," Mcgraw-Hill, 1994.

[3] D. D. Gajski, N. Dutt, A. Wu, and S. Lin, "High-Level Synthesis : Introduction to Chip and System Design," Kluwer Academic Publishers, Boston, 1992.

[4] J. Lee, Y. Hsu, and Y. Lin, "A New Integer Linear Programming Formulation for the Scheduling Problem in Data-Path Synthesis," in Proc. Int. Conf. Computer-Aided Design, pp. 20-23, 1989.

[5] N. Park and A. C. Parker, "Shewa : A software package for synthesis of pipelines from behavioral specification," IEEE Trans. Computer-Aided Design, Vol.7, pp.356-370, March, 1988.

[6] P. G. Paulin and J. P. Knight, "Force-directed scheduling for behavioral synthesis of ASIC's," IEEE Trans. Computer-Aided Design, Vol.8, pp.661-679, March, 1989.

[7] W. F. J. Verhaegh, P. E. R. Lippens, E. H. L. Aarts, J. H. M. Korst, A. van der Werf and J. L. van Meerbergen, "Efficiency Improvements for Force-Directed Scheduling," in Proc. Int. Conf. Computer-Aided Design, pp.286-291, 1992.

[8] Y. H. Choi, "Synthesis of pipelined datapaths," in Proc. Int. Conf. Computer-Aided Design, pp.36-40, Jan., 1992.

[9] C. T. Hwang, Y. C. Hsu and Y. L. Lin, "Scheduling for functional Pipelining and Loop Winding," in Proc. 28th Design Automation Conf., pp.764-769, 1991.

[10] C. T. Hwang, Y. C. Hsu and Y. L. Lin, "PLS : A scheduler for pipeline synthesis," IEEE Trans. on CAD/ICAS, Vol.12, No.9, pp.1279-1286, Sep., 1993.

[11] T. F. Lee, A. C. Wu, D. D. Gajski and Y. L. Lin, "An effective methodology for functional pipelining," in Proc. Int. Conf. Computer-Aided Design, pp.230-233, 1992.

[12] K. S. Hwang, A. E. Casavant, C. T. Chang and A. d'Abreu Manuel, "Scheduling and Hardware Sharing in Pipelined Data Path," in Proc. Int. Conf. Computer-Aided Design, pp.24-27, 1989.



**유 희 진**

e-mail : hjyoo@suncheon.ac.kr

1990년 원광대학교 전자계산공학과 졸업 (공학사)

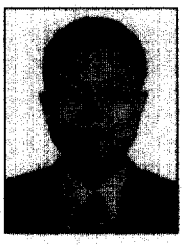
1992년 홍익대학교 대학원 전자계산학과 졸업(이학석사)

2000년 홍익대학교 대학원 전자계산학과 졸업(이학박사)

1998년~2002년 우석대학교 컴퓨터교육과 시간강사

2002년~ 현재 순천제일대학 컴퓨터과학부 교수

관심분야 : 설계자동화, 상위수준합성


**오 주 영**

e-mail : odid080@kic.ac.kr

1996년 서울산업대학교 전자계산학과

1998년 홍익대학교 대학원 전자계산학과 (이학석사)

2002년~현재 홍익대학교 대학원 전자계산학과 박사과정수료

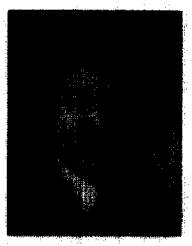
1998년 ETRI 위촉연구원

2000년 (주)eKalos 선임연구원

2001년 (주)참좋은 인터넷 책임연구원

2002년~현재 경인여자대학 전임강사

관심분야 : 통합설계, 상위수준합성




**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