조건부 분기를 가진 데이타-흐름 그래프 스케쥴링 알고리즘

(A Scheduling Algorithm for Dataflow Graphs with Conditional Branches)

김 태 환 †

(Taewhan Kim)

요 약 이 논문에서는 중첩된 조건부 분기를 가진 데이타-흐름 그래프에 대한 효과적인 스케쥴링 알고리즘을 제안한다. 이러한 그래프의 스케쥴링은 조건부 자원 공유 문제를 추가적으로 고려해야 하기 때문에 상당히 복잡하게 된다. 이 논문은 이를 적절히 해결하기 위한 방법을 제시하고 있는데 그 핵심은 조건부 분기가 있는 데이타-흐름 그래프를 조건부 분기가 없는 동일한 기능의 그래프로 변형시키는데 있다. 이렇게 함으로서, 변형된 그래프에 설계자의 관심에 맞는 기존의 스케쥴링 알고리즘을 선택 적용하여 스케쥴을 얻을 수 있고, 이것에서부터 원래 그래프의 스케쥴을 생성 할 수 있다. 실험 결과로부터 우리는 이러한 접근 방식이 매우 효과적임을 입증한다.

Abstract An effective scheduling algorithm for dataflow graphs with conditional branches is presented. Since such data-flow graphs entail the problem of conditional resource sharing additionally, the scheduling problem becomes more complicated. To resolve this problem, we propose a new algorithm whose main role is to transform a data-flow graph with conditional branches into a functionally equivalent one that has no conditional branches. A schedule is then obtained for the transformed one by using an existing scheduling algorithm, from which a schedule for the original data-flow graph is obtained. Experimental results show that the proposed approach is quite effective.

1. 서 론

상위단계 합성(High-level synthesis or Behavioral synthesis)은 VLSI 시스템의 주어진 Behavioral Description으로부터 제약된 일련의 하드웨어 및 수행시간 조건을 만족하는 RT (Register-Transfer) 단계 설계를 생성해 내는 과정이다. 상위단계 합성에서의 두 가지 주요 작업은, (1) 연산들의 수행 과정에 대한 스케쥴링(scheduling)과 (2) 연산, 변수, 연결선에 대한 하드웨어 할당(hardware allocation) 작업이다. 이 두 작업들은 서로 밀접히 연관되어 있으며, 각각의 결과는 서로에게 상당한 영향을 끼친다[1]. 스케쥴링 알고리즘은 clock 스텝의 개수와 하드웨어 비용을 최소화하기 위해, 연산들 사이의 잠재적인 하드웨어 자원공유(resource sharing)를 잘 이용해야 한다. 자원 공유는 두 가지 종류로 나눌 수 있다 [5]: 무조건적 자원 공유(unconditional resource sharing)와 조건부 자원 공유 (conditional resource sharing)이다. 무조건적 자원 공유는 각기 다른 clock 스텝에 스케쥴 된 두 연산들이 한 자원을 각기 다른 시간대에 공유하는 형태를 말하며, 조건부 자원 공유는 조건부 분기(conditional branch)의 상호 배제적(mutual exclusiveness)인 부분에 놓여진 연산들 간에 한 자원을 공유하는 형태를 말한다. 그림 1은 두 종류의 자원 공유에 대한 예를 보여준다. 연산 *op1*과 *op2*는 다른 조건부 분기에 있음을 나타낸다. 따라서, 두 연산이 그림에서 보듯 같은 clock 스텝에서 스케쥴되어 있어도 한 자원을 공유할 수 있는 조건부 자원 공유를 나타내고, 반면 연산 *op1* 과 *op3* (또는, *op2* 과 *op3*)는 다른 clock 스텝에서 한 자원을 공유하는 무조건적 자원 공유를 나타낸다.

† 종신회원 : 한국과학기술원 전산학과 및 첨단정보기술연구센터 교수
tkim@cs.kaist.ac.kr
논문접수 : 1999년 12월 20일
심사완료 : 2000년 11월 23일

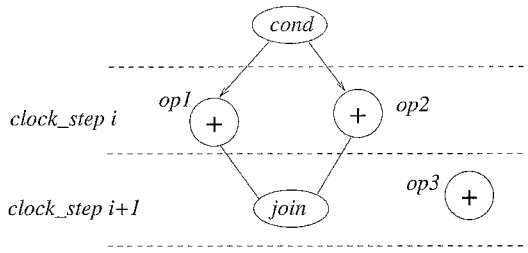

그림 1 스케줄에서의 자원 공유의 예

이 논문에서는 조건부 자원 공유를 좀더 효과적으로 이끌어내기 위한 스케줄링 알고리즘을 제안한다. 조건부 분기들 사이에서의 자원 공유를 다룬 기존의 몇몇 스케줄링 알고리즘들이 있었다[2,3,4]. Tseng *et al.* [2]은 이진(boolean) 조건 식을 사용하여 같은 clock 스텝에 스케줄될 수 있는 상호 배제적인 연산들을 찾아내었다. Wakabayashi와 Yoshimura [3]는 소위 조건 벡터(conditional vector)라는 것을 정의하여 상호 배제적인 연산들을 효율적으로 찾아내는 방법을 제안하고, 이를 이용한 간단한 리스터(list) 스케줄링 알고리즘을 고안하였다. Tseng의 접근방식에는 각각의 연산을 고려하는 대신 연산들의 집합들을, 즉, 한 집합에 있는 모든 연산들을 같은 clock 스텝에 스케줄 되어야 한다는 제한 조건하에서 스케줄이 이루어지는 단점이 있었다. Wakabayashi와 Yoshimura는 한 연산을 다른 실행 실현(execution instance)에서 동일하지 않은 clock 스텝에 스케줄할 수도 있음을 지적했다. 그러나, 그들은 실제로 이를 효과적으로 이용한 알고리즘은 제안하지는 않았다. Camposano [4]은 소위 path-based 스케줄링이라 불리는 알고리즘을 제안하였는데, 그 방법에서는 실행 실현들 사이에 일어날 수 있는 연산들간의 자원 공유에 대한 충돌(conflict)을 해결함으로써 각각의 실행 실현의 clock 스텝 수를 최소화하고, 이로 인해, 전체 clock 스텝 수의 최소화를 시도하였다. 그러나, path-based 스케줄링 알고리즘의 계산 복잡도는 실행 실현의 개수에 좌우되는데, 이 개수는 조건부 분기의 수에 따라 기하 급수적(exponential)으로 늘어 나게된다. 또한, 이 알고리즘은 각각의 실행 실현에 있어서, 연산들이 수행되는 순서는 입력 설계의 표현 그대로 미리 결정되어 있다는 가정 아래서 이루어지기 때문에 엄밀한 의미에서 상당한 제약을 이미 내포한 스케줄링을 하고 있다.

이 논문에서 제시된 접근 방법은 위에서 논의된 기존의 접근 방법들의 단점을 극복하고자 하는데 있다. 제안한 알고리즘은 세 단계로 구성되어 있다: (단계 1) 조건부 분기가 있는 데이타-흐름 그래프를 조건부 분기가 없는 데이타-흐름 그래프로 변형한다; (단계 2) 기존의 알려진 스케줄링 알고리즘을 사용하여 조건부 분기가 없는 데이타-흐름 그래프에 대한 스케줄을 얻는다; (단계 3) 단계 2에서 얻어진 스케줄로 부터 원래의 데이타-흐름 그래프에 대한 스케줄로 구한다.

단계 1에서는 조건부 분기가 있는 데이타-흐름 그래프를 조건부 분기가 없는 데이타-흐름 그래프로 바꾸기 위해서, 조건부 자원 공유을 어떻게 하면 효과적으로 탐색하여 이용할 것인가에 대한 알고리즘이 필요하며, 이것이 이 논문의 핵심 사항이 된다. 제안한 접근 방식은, 중첩된 조건부 분기에 대해 제일 안에 있는 조건부부터 바깥으로 가면서 처리하는 접근 방식을 취한다. 다시 말하자면, 가장 안쪽에 있는 조건부 블록을 비조건부 블록으로 먼저 변형하고, 가장 바깥쪽의 조건부 블록은 맨 마지막에 비조건부 블록으로 변형한다. 조건부 자원 공유로 인해 데이타-흐름 그래프 전체나 혹은 일부분에서 최장 실행 경로(critical execution path)의 길이를 증가시킬 수도 있음에 주목할 필요가 있다. 우리의 알고리즘에서는 제어 매개변수를 사용하여 그러한 증가를 일정 범위 안으로 유지한다.

단계 2에서는 변환한 데이타-흐름 그래프를 기존의 스케줄링 알고리즘을 적용하여 스케줄을 얻는다. 모든 조건부 자원 공유들은 단계 1에서 처리되었으므로, 변형된 데이타-흐름 그래프는 조건부 분기를 전혀 가지고 있지 않게 된다. 그 결과로, 여기에는 어떤 종류의 다양한 기존의 스케줄링 알고리즘의 적용을 고려할 수 있다. 적용할 스케줄링 알고리즘의 선택은 설계 목표와 제약조건들, 이를테면 실행 시간 제약조건, 하드웨어 제약조건, 연산 연결(chaining), 다중-사이클(multi-cycling) 연산, 파이프라인 등과 같은 것들에 바탕을 두고내려질 수 있다.

단계 3은 단계 2에서 얻어진 조건부 분기가 없는 데이타-흐름 그래프에 대한 스케줄로부터 원래의 데이타-흐름 그래프에 대한 스케줄을 유도 해낸다. 이것은 단순히 그냥 단계 1에서의 변형 과정을 역으로 하는 것만으로 얻어 질 수 있다. 그러므로 결국, 역 변형은 하향식 접근방식이 되며, 제일 바깥의 조건부 블록에 대한 스케줄을 구하고 제일 안쪽의 조건부 블록에 대한 스케줄을 구하게 된다.

그림 2는 제안한 알고리즘의 전체 흐름을 잘 나타내주는 예이다. 두 개의 조건부 분기블록이 겹쳐 있을 때 변형의 단계와 이것으로부터의 스케줄을 구하는 단계를 잘 보여주고 있다. 조건부 분기를 가진 데이타-흐름 그

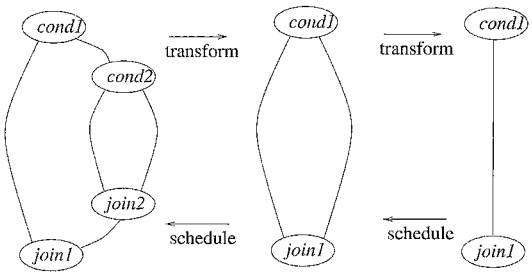

그림 2 제안한 스케쥴링 알고리즘의 흐름 예

래프의 스케쥴링 문제를 복잡도 면에서 살펴 보면, 가령 n 개의 조건부 블록이 중첩된 형태의 데이타-흐름 그래프 스케쥴링을 생각했을 경우, 서로 다른 실행 경로의 개수는 $2^{**}n$ 이다. 따라서, 이들 수행 경로 상에 있는 연산들 간의 자원 공유 최적화 문제는 n이 증가함에 따라 기하급수적으로 커지게 된다. 하지만 제안한 조건부 블록 변형 알고리즘을 적용하면 서로 다른 실행 경로 수를 한개로 줄일 수 있다. 따라서, 오직 무조건적 자원 공유 문제만 고려하게 됨으로 복잡도 면에서 매우 절약적이다. 하지만 명시하고자하는 것은 복잡도를 줄인 반면 데이타-흐름 그래프의 변형으로 인한 공간 탐색 범위는 줄어들게 될 수 있다. 따라서 제안한 변형 알고리즘의 핵심은 변형 과정에서 조건부 자원 공유를 최대한 활용하고자 하는 것이다.

2. 조건부 분기가 있는 데이타-흐름 그래프 스케쥴링 알고리즘

본론의 알고리즘 제안에 앞서, 먼저 독자의 이해를 돕기 위해 조건부 분기가 있는 데이타-흐름 그래프에 대해 조건부 분기가 있는 데이타-흐름 그래프는 세 가지 종류의 노드로 구성되어 있다:

포크(fork) 노드, 조인(join) 노드, 연산(operation) 노드이다. 포크 노드에서는 두 개 이상의 분기가 뻗어나가고(조건부 분기), 하나의 실행 실현에서는 그 중의 하나의 분기에 있는 연산들만 수행된다. 포크 노드에서 나가는 분기들은 다른 하나의 노드에서 다시 합쳐지는데, 이 노드를 조인 노드라고 한다. 그러므로 각각의 포크 노드들은 대응하는 조인 노드를 가지게 된다. 포크 노드와 그에 대한 조인 노드, 그리고 그 사이에 일련의 분기들을 조건부 블록이라고 한다. 이와 유사하게, 비조건부 블록이라는 말은 데이타-흐름 그래프에서 조건부 분기가 없는 부분을 의미한다. 데이타-흐름 그래프에서 포크와 조인 노드를 제외한 나머지 노드들은 연산 노드이다.

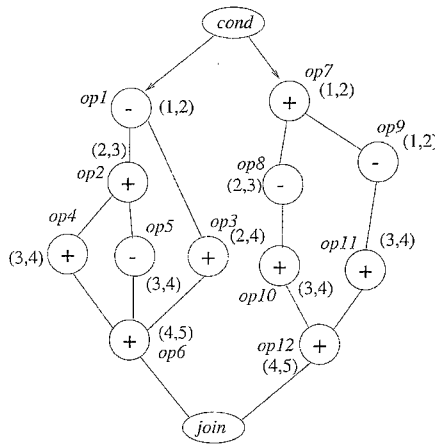

그림 3 조건부 분기를 가진 데이타-흐름 그래프의 예

예로서 그림 3을 보자. 노드 *cond*는 포크 노드이며, 노드 *join*은 조인 노드이고, 나머지들은 연산 노드이다. 두 노드 사이에 방향성을 주지 않은 에지는 위쪽의 노드에서 아래쪽의 노드로의 데이타 흐름을 뜻한다. 두 노드 사이에 방향성을 가한 에지는 화살표 방향으로의 컨트롤 흐름을 뜻한다. 포크 노드의 각 child는 분기를 의미하고, 연산 노드에서의 각 child는 데이타의 연결을 의미한다. 다시 말해 포크 노드와 연산 노드 모두 여러개의 child를 가질수 있지만 의미는 다름을 주목할 수 있다. 여기서 본 논문의 제안한 방법의 이해를 돕기위해, 포크 노드에서의 계산이나 연산 노드에서의 계산에는 한 clock 스텝의 계산 시간이 소요되고, 조인 노드에서의 계산 시간은 무시할만 한다고 가정한다. 간략히 설명을 한다.

이제 조건부 분기가 있는 데이타-흐름 그래프의 스케쥴링에 대한 새로운 방법을 제시하며 아래의 3 단계로 나누어 수행 된다.

단계 1 에서는 조건부 분기가 있는 데이타-흐름 그래프의 변형하는 것이다. 변형 단계의 목표는 조건부 블록을 원래의 기능성이 보존된 한 비조건부 블록으로 대체하는 것이다. 이러한 비조건부 블록은 어떤 블록에서의 조건부 분기들을 대표해서 나타내는 형태라고 볼 수 있으며, 조건부 분기들의 연산들 사이의 조건부 자원 공유들을 탐색해 봄으로서 이루어진다. 이런 식으로 결국, 비조건부 블록 스케쥴로부터 조건부 분기에 대한 스케쥴을 얻어낼 수 있게 된다. 비조건부 블록에 대해 구한 스케쥴의 총 clock 스텝 수는 조건부 분기들에 대한 스케쥴의 clock 수의 상위 한계치(upper bound)라 할 수

있다. 여기서 한가지 명시하고자하는 것은 조건부 분기가 있는 데이타-흐름 그래프는 그렇지 않은 그래프로 변형하는 것이 항상 가능하다. 하지만 조건부 분기가 비교적 적은 데이타-흐름 그래프는 조건부와 비조건부 자원 공유를 동시에 고려한 스케줄링이 더 효과적일 수도 있음을 배제할 수 없다. 이제 예를 통해 제안한 조건부 자원 공유 방법을 이용한 변형 알고리즘에 대해 자세히 논한다.

조건부 자원 공유를 통한 변형 알고리즘: 그림 3의 데이타-흐름 그래프에 나타난 조건부 분기를 살펴보자. 서로 다른 종류의 연산들은 각각의 대응되는 종류의 기능 모듈(functional module)에서 수행된다고 가정한다. 변형 단계의 기본적인 아이디어는, 같은 종류의 두 개의 연산을 각 조건부 분기에서 하나씩 골라서, 같은 clock 스텝에 수행되도록 같은 기능 모듈에 할당하는 것이다. 일단 하나의 쌍을 고르고 나서, 다음의 추가적인 쌍들을 차례로 결정하도록 한다.

조건부 블록의 임계 실행 경로(critical path)는 블록의 포크 노드에서 조인 노드까지의 가장 긴 실행 경로로 정의된다. 어떤 두 개의 조건부 분기들이 짝지어 졌을 때 (즉, 조건부로 같은 기능 모듈을 공유하도록 할당되어 졌을 때), 블록의 임계 경로의 길이는 증가할 수 있다. 예를 들어, 그림 3에서, 만약 연산 *op2*와 *op12*가 같은 기능 모듈을 공유하도록 되었다면, 조건부 블록의 임계 실행 경로의 길이는 4에서 6로 늘어날 것이다. 즉, *op7, op8, op10, (op2, op12), op4, op6*으로 이어지는 임계 경로가 된다. 임계 실행 경로 길이는 전체 실행 시간과 밀접한 관계가 있으므로, 임계 실행 경로 길이가 어떤 한계선을 넘지않게 하도록 자원의 공유를 잘 조절하려고 한다. T_{init} 가 원래의 조건부 블록의 임계 실행 경로 길이라고 하자. 사용자 정의 상수 u 가 주어졌다고 할 때, 다음의 u 개의 경우들을 검사해 보도록 한다. 조건부 분기에서의 연산들을 짝지어 냈을 때 결과로 임계 경로 길이가 $T_{init}+T_{inc}$ 을 넘지 않도록 하는 $T_{inc} = 0,1,2,\ldots u-1$까지의 u 개의 경우들 만을 검사해 본다. 각각의 경우에서 조건부 자원 공유를 최대로 만드는 변형을 수행한다. 변형 과정에 대해서는 3.2 절에서 자세히 설명한다. 구한 u 개의 경우들 중에서 가장 좋은 변형을 선택하는데 기준이 되는 비용 함수를 정의한다. T_{inc} 의 값이 정해진 상태에서, 3.2 절의 과정을 사용하여 조건부 블록의 변형을 결정한다. $S_i,\ i=1,2,\ldots k$ 를 타입 i의 연산(예: 가산, 감산, 곱셈 연산)에 대해 변형에서 조건부 자원 공유로 짝 지워진 쌍의 개수라고 하자. (여기서 k 는 서로 다른 연산 타입의 총 개수이다.) $w_i,\ i=1,2,\ldots k$ 를 사용자로부터 주어진 일련의 가중치라고 하면, 우리는

$$C = w_1S_1 + w_2S_2 + \ldots\ldots + w_kS_k \qquad (1)$$

라는 값을 계산하여 변형에서의 조건부 자원공유의 정량적인 예상 값으로 사용한다. u 개 각각의 경우에 대해서, 다음의 비율

$$C/(T_{init}+T_{inc}) \qquad (2)$$

을 계산하는데, 이 비율은 임계 실행 경로 길이를 증가시키는 것에 비례하여 조건부 자원 공유 증가의 효과를 나타낸다. 우리는 u 개의 경우 중에서 식 (2)의 값이 가장 크게 되는 해당 변형을 고르도록 한다.

이제 선택된 T_{inc} 값에 대해서, 조건부 분기에서 연산들을 차례로 짝지어 내되, 임계 실행 경로 길이가 안으로 유지하게 하는 알고리즘을 기술하도록 하겠다. $T_{init}+T_{inc}$ 의 값이 주어졌을 때, 어떤 연산이 스케줄될 수 있는 가장 빠른 시간과 가장 늦은 시간 사이의 간격을 시간 프레임이라고 부르는데, 이것은 ASAP(as soon as possible)와 ALAP(as late as possible) 스케줄링 알고리즘[7]을, 하드웨어나 다른 제약조건이 없다는 가정 아래에서 사용하여 쉽게 구할 수 있다. 예를 들어, 그림 3의 조건부 블록에서 $T_{init}+T_{inc}=5$ 인 경우에 대해 각각의 연산들의 시간 프레임은 괄호 안의 숫자들로 나타나 있다. 우리는 다음과 같이 가능한 모든 쌍을 짝지어 (즉, 조건부 자원 공유가 이루어 짐) 조사해본 다음, 아래의 절차에 따라 가장 효과가 있을 쌍을 선택한다. 즉, 각각의 가능한 연산의 쌍들 중에서, 만약 실제로 이 쌍이 선택되었을 때의 식 (1)의 C 값의 추정치를 계산하여 본다. 두 연산 op_i 와 op_j 가 짝 지워졌다고 해보자. $C(op_i, op_j)$ 는 이때의 C 값에 해당하는 추정치를 나타낸다고 하자. $C(op_i, op_j)$ 를 계산하기 위해서, 어떤 연산이 그 시간 프레임 안의 어떤 특정한 clock 스텝에 스케줄 될 확률 측도가 이용되는데, 이는 force-directed scheduling(FDS[7])에서 사용했던 것과 같은 방식을 적용하여 구할 수 있다. 이를테면, 그림 3의 조건부 블록에서, 연산 *op2* 와 *op10* 이 같은 기능 모듈을 공유한다고 해보자. 그러면, *op2* 와 *op10* 의 공통된 clock 시간대인 (3,3)으로 두 연산의 시간 프레임이 변경되고, 이 영향으로, 다른 선/후 관계된 연산들의 시간 프레임도 다시 조정된다. 각 분기의 연산들에 대해, 이제 그들의 시간 프레임들로부터 연산 타입 i 에 대한 연산의 밀도(density)를 얻을 수 있다. (여기서 연산 밀도에 대한 설명을 부연하면, 어떤 연산의 수행 가

능 범위란 총 clock 스텝수의 제약이 주어진 스케쥴링 문제에서 그 연산이 제일 이른 clock 스텝에서 수행 가능한 시간과 제일 늦은 clock 스텝에서의 수행 가능한 시간을 경계로하는 범위를 말하며 이것을 이용하여 그 범위의 한 특정 시간에 스케쥴링 되는 확률이 같다고 가정하면 같은 타입의 연산들에 대해 그 시간에서 얼마나 많은 연산들이 수행될 것인지를 예측할 수 있다. 이 예측치를 그 연산 타입에 대한 스케쥴링 밀도(density)라 부른다. force-directed scheduling은 이러한 연산 밀도를 각 시간대(즉, ckock 스텝별)로 최대한 균등하도록 하기 위해, 반복 수행과정을 통해 greedy 한 방법으로 연산들을 하나씩 스케쥴하는 방법이다.)

$X_i(t)$ 와 $Y_i(t)$$ 는 각각 왼쪽과 오른쪽 조건부 분기의 t 스텝에서 ($0 \le t \le n, n = T_{init} + T_{inc} - 1$) 연산 타입 i 의 연산 밀도를 표시한다. 그러면, 서로 다른 조건부 분기에 놓인 연산들 간의 수행에서의 상호 배제적(mutual exclusiveness) 성질에 의해 연산 타입 i 에 대한 clock 스텝 t 에서의 조건부 자원 공유양의 추정치로 $\min(X_{i(t)}, Y_{i(t)})$ 를 이용한다. 이때 $C(op_i, op_j)$는

$$C(op_i, op_j) =$$
$$w_1 \sum_{t=0}^{n} \min(X_1(t), Y_1(t)) + w_2 \sum_{t=0}^{n} \min(X_2(t), Y_2(t)) + \ldots$$
$$+ w_k \sum_{t=0}^{n} \min(X_i(t), Y_i(t)) \quad (3)$$

과 같이 표시된다.

그림 3의 예에서 각 연산 쌍에 대한 식(3)의 값들을 구했을 때 $C(op_3, op_{10})$가 가장 큰 값을 가지고 있다. 그러므로 연산 *op3* 과 *op10* 이 짝 지워지게 된다. 연산 *op3* 과 *op10* 이 짝지어 졌을 때, 이들은 하나의 연산 노드 (*op3, op10*)으로 합쳐져서 두 연산 노드 *op3* 과 *op10* 의 모든 데이타 의존성을 그대로 물려받게 된다는 데 주목할 필요가 있다. 새 노드 (*op3, op10*)의 시간 프레임은 (2,3)이 되는데 이것은 두 노드 *op3* 과 *op10* 의 시간 프레임의 교집합이다. 알고리즘의 두 번째 반복에서 연산 *op4* 와 *op11* 이 선택되어 짝지어진다. 다시금 각 연산들의 시간 프레임들이 새로운 값을 가지게 된다는 점에 주목하라. 이러한 반복은 시간 프레임이 겹치는 연산의 쌍들이 더 이상 없을 때까지 계속 된다. 그림 4는 마지막 반복이 끝난 다음에의 변형이 완료된 데이타-흐름 그래프를 나타내고 있다. 그림 4의 데이타-흐름 그래프로부터, 네 쌍의 덧셈 연산들과 한 쌍의 뺄셈 연산이 조건부 자원 공유하게 됨을 알 수 있다.

단계 2 에서는 기존의 스케쥴링 알고리즘의 적용하는 것이다. 즉, 단계 1에서 얻어진 데이타-흐름 그래프에대

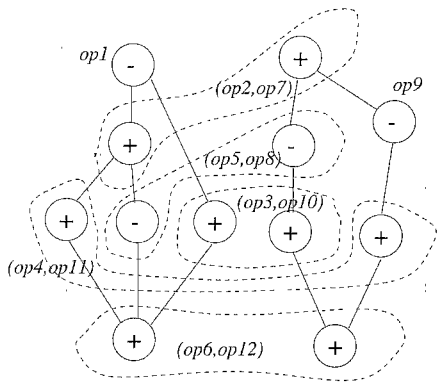

그림 4 그림 3의 그래프를 조건부 분기를 가지지 않은 그래프로의 변형 결과

한 스케쥴을 구하는 일이다. 그런데, 이 데이타-흐름 그래프에는 조건부 분기가 없으므로, 이미 존재하는 많은 스케쥴링 알고리즘들이 다 적용 될 수 있다. 본 실험에서는 force-directed list scheduling (FDLS)[7]을 사용하였지만, 설계의 대상이 되고 있는 특정한 데이타-흐름 그래프의 종류에 따라 더 적합한 다른 어떠한 스케쥴링 알고리즘으로도 쉽게 대체할 수 있는 장점이 있다.

마지막으로 단계 3에서는 원래 데이타-흐름 그래프에 대한 스케쥴 유도를 유도한다. 즉, 단계 1에서, 조건부 블록을 상향식(bottom-up) 방식으로 비조건부 블록으로 변형하였다. 이제는 비조건부 블록 스케쥴에서부터 조건부 분기들에 대한 스케쥴을 하향식(top-down) 방식으로 유도해 낸다 (그림 2 참조). 비조건부 블록 스케쥴에는 각각의 연산들이 하나의 clock 스텝에 배정되어 있다. 두 조건부 분기에 대해서 두 개의 스케쥴을 만들어 내는데, 단순히 모든 연산들을 비조건부 블록 스케쥴에서 배정된 clock 스텝에 그대로 배정하는 식으로 한다. 단계 1에서 수행된 변형에 의하면, 이렇게 해서 얻어진 스케쥴들은 조건부 분기들에 대한 타당한(valid) 스케쥴이 되게 된다. 게다가 단계 1에서 비조건부 블록을 형성할 때 하나의 노드로 합쳐졌던 연산들은, 조건부로 하나의 기능 모듈을 공유하게 된다.

3. 실험 결과

본 논문에서 제안한 할고리즘의 결과를 path-based 스케쥴링 알고리즘[4], MAHA(critical-path-first scheduling)[6], Wakabayashi의 스케쥴링 알고리즘[3]의 결과들과 비교하였다. 실험에서 관심을 두는 것은 조건분기를 가진 데이타-흐름 그래프를 주어진 사용할 수

있는 자원 제약 조건하에서 얼마나 수행 시간 (즉, clock 스텝 수)를 줄이는 가를 비교하는 것이다. 실험 절차는 먼저 입력 데이타-흐름 그래프를 받아 조건부 분기를 비조건부로 만들고, 이를 force-directed 스케쥴링 알고리즘을 적용하여 결과를 얻고, 그 결과를 각 실행 경로에 따라 얼마나 많은 clock 스텝을 사용하는지를 점검하였다. 비교하고자하는 기존의 결과는 논문으로부터 인용하였다.

표 1은 [6]에서 보여진 데이타-흐름 그래프 예에 대한 결과를 보여준다. 여기에서 모든 분기들은 조건부 분기라고 가정하였으며, 표의 가산기 열은 사용한 가산기의 개수, 감산기 열은 사용한 감산기의 개수, 체인 열은 한 clock 스텝 안에 연결 수행 될 수 있는 연산의 개수이다. clock 스텝수 열은 FSM(finite state machine)의 clock 스텝의 총 개수(즉, 총 state 수), 가장 긴 실행 실현에서의 clock 스텝의 총 개수, 가장 짧은 실행 실현에서의 총 개수를 나타낸다. 우리의 결과는 path-based 스케쥴링과 MAHA 알고리즘의 결과와 비교하였다. MAHA 스케쥴링은 체인 값이 2일 때 8 clock 스텝 짜리 스케쥴을 생성해 낸 것과 달리, 제안한 알고리즘의 스케쥴은 체인 값이 1일 때 역시 8 clock 스텝을 만들어 내었다. 체인이 2이고, 가산기와 감산기가 하나씩 사용되었을 때, 제안한 알고리즘과 path-based 스케쥴링은 최장/최단 실행 실현에 있어서 같은 수의 clock 스텝을 사용하였다. 그렇지만, 전체 clock 스텝의 수는 우리 결과의 경우 크게 줄어들었다. 두 개의 가산기와 세 개의 감산기가 사용된 경우에는, 두 알고리즘은 유사한 결과를 내었지만, path-based 스케쥴링에서는 체인이 5인데 비해 제안한 알고리즘의 경우에는 체인이 단지 3으로 구해졌다.

표 2는 [3]에서의 사용한 데이타-흐름 그래프로부터 얻어진 결과이다. #ALU는 덧셈과 뺄셈을 할 수 있는 ALU의 개수를 나타내고 비교연산기 열은 사용한 비교 연산기의 개수이며, 평균 열은 모든 조건부 분기가 일어날 확률이 같다는 가정아래에서 평균 clock 스텝의 개수를 나타낸다. 우리는 결과를 Wakabayashi의 알고리즘과 path-based 스케쥴링의 결과와 비교하였다. 하나의 가산기, 감산기, 비교 연산기를 사용하고, 체인을 1로 놓았을 때 우리의 알고리즘은 Wakabayashi의 알고리즘에 비해 최단 실행 실현에 있어서는 하나 적은 clock 스텝을 사용했고, 전체 실행 실현에 대한 평균에서도 더 적은 수를 나타내었다. 체인의 값을 2로 놓았을 때는, 우리의 알고리즘은 path-based 스케쥴링 알고리즘의 결과보다 역시 하나 적은 clock 스텝을 사용하였다. 두 개의 ALU가 사용되었을 때는 우리의 알고리즘은 path-based 스케쥴링 알고리즘과 같은 수의 clock 스텝을 사용하였다.

표 1 MAHA[6]의 데이타-흐름 그래프에 대한 결과 비교

	제약조건 +/-/체인	clock 스텝수		
		총개수	가장긴것	가장짧은것
ours	1/1/1	8	8	3
	1/1/2	6	5	2
	2/3/3	3	3	2
PATH-BASED	1/1/2	9	5	2
	2/3/5	4	3	1
MAHA	1/1/2	8	8	-
	2/3/3	4	4	-

표 2 [3]의 데이타-흐름 그래프에 대한 결과 비교

	제약조건 #ALU/+/-/체인	clock 스텝수 총개수/긴것/짧은것/평균
ours	0/1/1/1/1	7/7/4/4.75
	0/1/1/1/2	7/7/3/4.25
	2/0/0/1/2	6/6/3/4.25
PATH-BASED	0/1/1/1/2	8/7/3/4.75
	2/0/0/1/2	6/6/3/4.25
WAKABA	0/1/1/1/1	7/7/5/5.75

표 1, 2 의 결과에서 보여 주듯이 조건부 분기가 복잡하게 얽혀 있을 경우, 조건부와 무조건부 자원 공유를 모두 생각한 기존의 스케쥴링 방법들은 clock 스텝 수를 줄이는데는 한계를 보여준다. 반면에 자원 공유문제를 두 단계로 처리한 우리의 알고리즘은 상대적으로 매우 능률적임을 실험 결과는 잘 나타내고 있다.

4. 결 론

이 논문에서는 중첩된 조건부 분기들이 있는 데이타-흐름 그래프에 대한 새로운 스케쥴링 알고리즘을 제시했다. 제안한 알고리즘은 조건부 분기를 비조건부 분기로 변형하는 과정을 통해 조건부와 비조건부 분기에서의 자원 공유를 동시에 고려하고자 하는 기존의 알고리즘들에 비해 복잡도를 줄였고, 또한 스케쥴 결과가 기존의 것에 비해 수행 속도 면에서 빠르다는 것을 실험을 통해 보였다. 궁극적으로 이는 제안한 알고리즘이 조건부 분기를 많이 가지는 데이타-흐름 그래프 스케쥴링에 매우 적합함을 입증하였다.

감사의 글

본 연구는 첨단정보기술 연구센터(AITrc)를 통하여 과학재단의 지원을 받았음.

참 고 문 헌

[1] D. W. Knapp, *Behavioral Synthesis* Prentice Hall, 1996

[2] C.-J. Tseng, R. W. Wei, S.G. Rothweiler, M. Tong and A.K. Bose, "Bridge: A Versatile Behavioral Synthesis System," *Proc. of Design Automation Conference,* pp. 415-420, 1988

[3] K. Wakabayashi and T. Yoshimura, "A Resource Sharing Control Synthesis Method for Conditional Branches," *Proc. of International Conference on Computer-Aided Design* pp. 62-65, 1989

[4] R. Camposano, "Path-Based Scheduling for Synthesis," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 1, pp. 85-93, Jan. 1991

[5] K. S. Huang, et al., "Constrained Conditional Resource Sharing in Pipeline Synthesis," *Proc. of International Conference on Computer-Aided Design,* pp. 52-55, 1988

[6] A. C. Parker, J. Pizarro and M. J. Mlinar, "MAHA: A Program for Data Path Synthesis," *Proc. of Design Automation Conference,* pp. 461-466, 1986

[7] P. G. Paulin and J. P. Knight, "Force-Directed Scheduling for Behavioral Synthesis of ASIC's," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 6, pp. 661-679, June, 1989

김 태 환
정보과학회논문지 : 시스템 및 이론
제 28 권 제 1 호 참조