삼각행렬과 접합블럭을 이용한 마이크로프로그램의 광역적 최적화

(A Global Compaction of Microprograms Using Triangular Matrices and Junction Blocks)

崔 基 浩*, 林 寅 七**
(Ki Ho Choi and In Chil Lim)

要 約

본 논문은 마이크로프로그램의 최적화 과정에서 필요한 MOP's(microoperations)의 데이타 종속 관계와 자원상충관계를 DAG(Directly Acyclic Graph)로 나타내던 종래의 방법대신 삼각행렬상에 보다 간단하게 나타내어 이용할 수 있는 데이타 종속행렬(Data dependency Matrix:DDM) 표현 방법을 제안하고, 또한 접합블럭으로 트레이스(trace)를 분할함으로써 블럭복사를 방지할 수 있는 마이크로프로그램의 광역적 최적화 알고리즘을 제안하였다.

제안된 DDM과 최적화 알고리즘을 Lah의 예에 적용시켰으며, 제안된 알고리즘이 실행시간과 제어기억 용량을 감소시키는데 있어서 종래의 알고리즘 보다 더 효율적임을 보였다.

Abstract

To represent the relations of the data dependency and resource conflict among micro-operations (MOP's) in the compaction process of microprograms, we propose a DDM (data dependent matrix) representation method instead of the DAG (conventional directed acyclic graph). Also, we propose a global compaction algorithm of microprograms to prevent a kind of block copying by cutting the trace at a junction block. The DDM method and compaction algorithm have been applied to the Lah's example. The result shows that the proposed algorithm is more efficient than the conventional algorithms in reducing the total execution time and control memory space.

I. 序 論

1951년 M. V. Wilkes의 마이크로그래밍(microprogramming) 개념소개 이후, LSI/VLSI등 반도체 IC기술의 발달과 함께 마이크로프로그래밍 제어방식은 성능/가격비가 우수하고 복잡한 디지탈 시스템 개발에 이바지해 왔다.

마이크로프로그램 제어방식은 제어부(control unit)의 체계적인 설계, 유연성(flexibility), 확장성(extensibility), 에뮬레이터에 의한 소프트웨어의 호환성(compatibility) 등의 장점을 갖기 때문에 기능이 복잡하고 다양한 명령 세트를 갖는 컴퓨터(complex in-

*正會員, 光云大學 電子計算機工學科
(Dept. of Computer Eng., Kwangwoon Univ.)
**正會員, 漢陽大學校 電子工學科
(Dept. of Electronic Eng., Hanyang Univ.)
接受日字 : 1986年 5月 3日

struction set computer:CISC)의 제어에 널리 이용되고 있다. 또한, 고속 메모리의 가격 감소와 더불어 WCS(Writable Control Store)의 이용과 이용자(user) 마이크로프로그래밍이 가능해짐에 따라 보다 나은 마이크로프로그래밍 도구(tool) 개발이 요구되고 있다.[1]

마이크로프로그래밍 도구개발의 궁극적인 목표는 여러가지 다양한 머신(machine)에 대해 최소 실행시간을 갖는 마이크로코드(microcode)를 생성할 수 있는 컴파일러(compiler)와 HLML(High Level Microprogramming Language)을 개발하는데 있다고 할 수 있다.

그간 HLML을 실현하고자 하는 많은 연구가 있었으나 아직도 일반적으로 널리 이용될 수 있는 컴파일러는 없는 상태이며, 일반적으로 컴파일러에 의해 생성된 마이크로코드가 고도의 숙련된 마이크로프로그래머에 의해 작성된 것 보다 실행속도가 아직도 상당히 느린편이다.[2]

마이크로코드로 컴파일하는 것을 복잡하게 만드는 주 원인은 재래의 머신 코드(code)의 구조보다 수평(horizontal) 마이크로코드의 구조가 훨씬 복잡 다양하고, 또한, 최소 실행시간을 갖도록 마이크로코드화 하는 것이 대단히 중요한 문제이기 때문이다.

따라서, 수평 마이크로구조(microarchitecture)에서 마이크로프로그램의 실행시간과 제어기억용량을 줄이기 위해 병렬 수행성을 갖는 MOP's(microoperations)를 찾아 MI's(microinstructions)로 구성하는 과정인 마이크로프로그램의 최적화(compaction) 문제에 대한 연구가 필요하게 되었다. 이러한 연구로는 마이크로 프로그램의 국소적 최적화(local compaction)와 광역적 최적화(global compaction)가 있다.

국소적 최적화는 기본블럭(basic block) 내에서만 최적화를 수행하고, 광역적 최적화는 한개 이상의 기본블럭들을 최적화하는 것으로 MOP들이 블럭 경계를 넘어서 최적화 될 수 있기 때문에 국소적 최적화시 보다 전체 마이크로프로그램 실행시간을 더욱더 감소시킨다.[3-5]

마이크로프로그램의 광역적 최적화에 대해 Wood,[5] Tokoro,[6] Fisher,[4,7] Isoda,[8] Lah,[9] Su[2] 등이 활발히 연구하여 왔다. 이러한 연구중 Fisher의 Trace scheduling[4,7]이 가장 일반적인 방법으로, 최적화된 마이크로코드 실행시 가장 빠른 실행을 보이고 있다. Trace scheduling 방법은 실행시간을 상당히 감소시킬 수 있지만, 블럭복사(block copy)로 인해 제어기억 용량을 매우크게 증가시킬 수 있다. 최악의 경우 블럭복사에 의한 제어기억 용량은 지수함수적으로 증가할 수 있고 복잡한 북키핑(bookkeeping) 단계가 Trace scheduling을 실현하는데 큰 문제점으로 남아있다.

또한, Lah의 Tree compaction[9]은 Fisher의 Trace scheduling의 단점인 제어기억 용량의 지수함수적인 증가 가능성과 복잡한 북키핑단계를 해결하고자 마이크로프로그램 블럭 집합을 트리 모양으로 나누어 최적화를 수행한 결과 제어기억 용량은 크게 줄일 수 있었으나 실행시간은 약간 증가하였다.

Su[2]는 북키핑을 위해 블럭간의 MOP 이동법칙을 제시한 Fisher의 메뉴(menu)방법을 수정 보완한 ITSC (Improved Trace Scheduling Compaction) 알고리즘을 제안하였으나 재결합(rejoin)되는 MOP가 조건부 점프(conditional jump) MOP 아래에서 최적화 되었을 때 블럭 복사를 해야하는 문제점은 남아 있다.

Atkins[10]는 게이트(gate)를 최소화하는 Quine-Mc Clusky방법에 착안하여 마이크로코드의 국소적 최적화를 위한 QM(Quine Modified) 알고리즘을 제안하였는데, 이것은 DAG를 입력하여 MOP 상충행렬(conflict matrix)을 구성하고 LA(Look-Ahead) 세트와 PI(Prime Implicant) 세트를 구하고 BAB(Branch and Bound) 알고리즘을 적용시키기 때문에 수행시간이 길어진다는 단점을 갖고 있다.

국소적 최적화나 광역적 최적화 과정에서, MOP들간의 데이타 종속관계와 자원 상충(resource conflict) 관계를 나타내는데 이용되어 온 종래의 DAG(또는 data dependency graph:DDG)는 그 구성과 이용면에서 알고리즘이 매우 복잡하기 때문에, 본 논문에서는 QM 알고리즘의 MOP 상충행렬에 착안하여, DAG를 구성하지 않고 원시 마이크로프로그램으로부터 삼각행렬상에 데이타 종속관계와 자원 상충관계를 보다 용이하게 나타내고 이를 최적화 과정에 직접 이용할 수 있는 방법을 제안한다.

또한, 본 논문에서는 Trace scheduling이 갖는 블럭복사의 문제점을 해결하기 위하여, 수행빈도가 높은 트레이스(trace)를 구하고 그 트레이스를 접합블럭으로 분할함으로써 대량의 블럭복사를 방지할 수 있는 마이크로프로그램의 광역적 최적화 알고리즘을 제안한다. 또, 본 알고리즘을 Lah의 예에 적용하여 종래의 알고리즘과 비교함으로써 본 알고리즘의 최적화를 입증한다.

Ⅱ. 마이크로프로그램의 모델링 및 기본정의

본 장에서는 본 논문에서 제안되는 알고리즘 수행에 필요한 마이크로프로그램의 모델링(modeling)과 기본정의[3-4]에 대해 서술한다. 알고리즘을 간단히 하기 위해 각 MI는 단일위상(monophase) 동안 실행된다고 가정한다.

(정의 1) 머신에 있는 여러 자원들의 최소단위의 기본동작을 MOP(microoperation)라고 한다. 마이크로프로그램은 MOP's의 시퀀스로 구성된다.

(정의 2) MOP는 6-tuple(label, instruction, next-address, destination-registers, source-registers, resource vector)로 모델링 한다. 여기서 label은 각 MOP를 지정하는 번호이며, instruction은 CONT, CJMP, GOTO등과 같은 마이크로시퀀서(microaddress controller)에의 명령을 표시하며, next-address는 다음에 수행될 MOP의 타겟트 번지(target address)이다. Destination-registers와 source-registers는 각 MOP에서 write되거나 read되는 레지스터를 나타낸다. Resource vector는 MOP를 수행하는데 필요한 하드웨어 요소(hardware component)를 나타낸다. 해당요소가 사용되면 "1"이고 사용되지 않으면 "0"으로 표시한다.

(정의 3) 주어진 사이클동안 함께 수행되는 MOP들의 집합을 MI(microinstruction)라 한다.

(정의 4) 시작점에서만 진입점(entry point)을 갖고 종점에서만 분기(branch)할 수 있으며 내부 루프(loop)를 갖지 않는 MOP's의 시퀀스를 기본블럭이라 한다.

(정의 5) 시작점에 재결합 MOP를 갖고 종점에 조건부 분기 MOP를 갖는 기본블럭을 접합블럭 이라 한다.

(정의 6) 기본블럭을 절점(node)으로 하고 기본블럭 간의 데이타 흐름을 마디(edge)로 하는 방향성그래프(directed graph)를 흐름 그래프(flow graph)로 정의한다.

(정의 7) 흐름 그래프상에서 어떤 경로(path)에서도 반드시 수행되는 기본블럭을 dominant 블럭이라 한다.

(정의 8) 흐름 그래프에서 각 기본 블럭마다 수행빈도를 측정하여 그 블럭의 기대값으로 정의한다.

(정의 9) 주어진 마이크로프로그램이나 그 일부에서 어떤 다른 경로보다 높은 기대값을 갖고 실행 경로를 구성하는 블럭들의 집합을 트레이스라 한다.

(정의10) 각 블럭의 실행시간에 각 블럭이 실행할 기대값을 곱한 값의 합을 그 마이크로프로그램의 가중실행시간(weighted execution time)이라고 한다.

(정의11) MPO_i와 MOP_j가 다음 조건을 만족할 때 데이타종속관계가 있다고 말한다.

(조건 1) $O_i \cap I_j \neq \phi$

(조건 2) $I_i \cap O_j \neq \phi$

(조건 3) $O_i \cap O_j \neq \phi$

여기서 I와 O는 각 MOP의 source-registers와 destination-registers의 집합을 나타낸다.

(정의12) MOP_i와 MOP_j가 동일한 하드웨어 요소를 사용할 때 MOP_i와 MOP_j는 자원 상충관계가 있다고 한다.

(정의13) MOP's간의 데이타 종속관계와 자원 상충관계를 나타낸 삼각행렬을 데이타 종속행렬(data dependency matrix:DDM)이라 정의한다. 이때 각 행(row)과 열(column)은 원시 마이크로프로그램에서의 MOP순으로 배열한다.

(정의14) 프로그램상의 어떤 점에서 레지스터 내용이 overwrite되기전에 읽혀지면 그 레지스터는 그 점에서 live하다고 하고 그렇지 않으면 dead하다고 한다.

(정의15) 트레이스상에 조건부 분기 MOP가 있을 경우 이 MOP이후에 트레이스로 부터 분리된 블럭의 시작점(IN)에서 live한 레지스터 집합을 조건부 source 레지스터 집합이라 한다.

이러한 집합을 그 조건부 분기 MOP의 source 레지스터에 첨가함으로써 이 MOP 이후의 모든MOP와의 데이타종속 여부를 확인하여 데이타 종속관계가 있으면 그 분기 MOP행의 해당 열에 D를 삽입한다. 이렇게 함으로써 조건부 분기 MOP이후 선택된 경로상에서 조건부 source레지스터 집합을 write하는 MOP와 분기 MOP가 마치 데이타 종속관계가 있는 것 처럼 취급하여 해당 MOP가 분기 MOP위로 올라가서 다른 경로 상에서 write되어 변형된 데이타 값을 read하여 프로그램이 변경되는 것을 방지한다.

(정의16) 한 트레이스의 시작점이 되는 블럭을 트레이스헤드(tracehead)라 하고 이러한 블럭들의 집합을 트레이스헤드 집합이라 한다.

Ⅲ. 최적화 알고리즘

마이크로프로그램의 전체 수행시간 및 제어 기억용량을 감소시키기 위해 본 논문에서 제안되는 광역적 최적화 알고리즘에서는 먼저, 원시 마이크로프로그램을 기본 블럭들로 나누어 흐름그래프를 구성하고 live 레지스터를 분석한 후 접합블럭을 고려하여 최적화 시킬 트레이스를 선택하며, 본 논문에서 제안되는 삼각행렬 구성 방법에 의해 선택된 트레이스에 대한 원시 마이크로프로그램의 데이타 종속관계와 자원 상충관계를 나타내고 이를 이용하여 각 MOP의 우선 순위를 정한 후, 이 우선 순위에 따라 선택된 트레이스의 각 MOP를 최적화한다. 이러한 최적화 알고리즘을 기술하면 다음과 같다.

입력 : MOP's(label, instruction, next-address, destination-registers, source-registers, resource vector)의 시퀀스.

출력 : 각 MI가 하나 이상의 MOP들을 갖는 MI's의 시퀀스.

절차 :

(1) 초기화 및 전처리

(2) 모든 블럭들이 최적화될 때까지 다음 과정을 반복 수행한다.

- 최적화 될 트레이스선택
- 선택된 트레이스의 최적화
- 최적화된 MOP's에 대한 북키핑

1. 흐름 그래프구성

전처리과정으로 마이크로프로그램의 데이타 흐름을 분석하여 각 기본블럭을 구한 후 기본블럭을 절점으로 하고 데이타 흐름의 방향을 마디로 하는 흐름 그래프를 구성하되 그림 1과 같이 조건부 분기와 재결합이 직접 연결되는 곳에 모조블럭(dummy block)을 부가함으로써 북키핑시에 복사되는 MOP's를 수용할 수 있도록 한다. 최적화 후 빈 보조블럭은 제거한다.

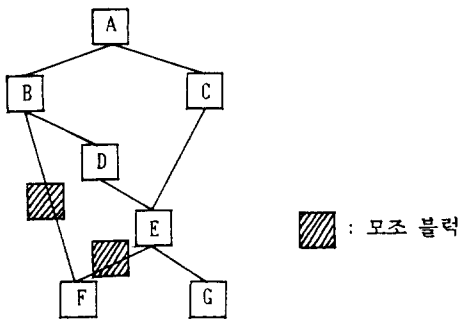


그림 1. 모조 블럭 삽입
Fig. 1. Insertion of dummy blocks.

2. Live레지스터 분석

구성된 흐름 그래프상에서 아래와 같이 각 기본블럭에 대해 데이타 흐름식(data flow equation)[11-12]을 역방향으로 적용시켜 각 기본 블럭의 IN과 OUT 에서의 live레지스터 집합을 구한다.

$$OUT[B] = \bigcup_{S \in SUC(B)} IN[S]$$

$$IN[B] = (OUT[B] - WR[B]) \cup RD[B]$$

여기서

B : 기본 블럭

SUC(B) : 블럭 B의 후행(successor) 블럭들의 집합

OUT[B] : 블럭 B의 출력점에서의 live레지스터 집합

IN[B] : 블럭 B의 입력점에서의 live 레지스터 집합

WR[B] : 블럭 B안에서 write된 레지스터들의 집합

RD[B] : 블럭 B안에서 read된 레지스터들의 집합

단, 블럭 B에서 read되기전에 먼저 write된 read레지스터는 제외한다.

3. 트레이스의 선택

흐름 그래프 상에서(정의 9)에 의해 구하여진 기대값 중 가장 높은 기대값을 가진 진입블럭을 선택하고 그 블럭으로 부터 시작하여 접합블럭이나 출구블럭 또는 이미 최적화된 블럭에 이를 때 까지 계속 가장높은 기대값을 가진 후행 블럭을 선택하여 트레이스를 구성한다. 접합 블럭에 의해 트레이스가 절단되므로 재결합되는 MOP나 그 위의 MOP가 조건부 점프MOP 아래에서 최적화될 수 없다. 따라서 Trace scheduling 에서와 같은 블럭복사는 발생되지 않는다. 이러한 트레이스 선택 알고리즘은 다음과 같다.

입력 : 각 블럭들이 블럭 레이블과 기대값과 블럭의 최적화 수행 여부를 나타내는 플래그(flag)를 갖는 3-tuple(블럭레이블, 기대값, 프래그) 의 블럭 집합

출력 : 실행 경로를 이루는 블럭들의 집합

절차 :

(1) 트레이스헤드 집합으로 부터 빈 블럭 집합을 제거하고 그 후속 블럭을 트레이스헤드 집합에 부가한다.

(2) 각MOP의 조건부 source 레지스터와 재결합 destination레지스터를 빈 상태가 되게 한다.

(3) 트레이스헤드 집합에서 가장 기대값이 큰 블럭을 현 블럭으로 하고 트레이스에 부가한다.

(4) 현 블럭이 출구 블럭, 접합 블럭, 또는 이미 최적화된 블럭이 될 때 까지 다음 과정을 반복한다.

- 후행 블럭들 중에서 가장 기대값이 큰 블럭을 현 블럭으로 바꾼다.
- 만일 현 블럭이 최적화 안된 블럭이면 최적화된 것으로 마크하고 그 트레이스에 부가한다.

(5) 트레이서헤드 집합을 갱신하되 현 블럭이 접합 블럭이면 그 블럭도 트레이스헤드 집합에 추가한다.

(6) 원시 마이크로프로그램의 MOP 순서대로 MOP's의 시퀀스를 정한다.

(7) 트레이스에서 떨어져 나간 후행블럭의 IN에서 live한 레지스터들을 조건부 source레지스터로 추가한다.

(8) 트레이스에서 떨어져나간 재결합점 위의 조상(ancestor) 블럭들에서 write되고 재결합MOP를 갖는 블럭의 IN에서 live한 레지스터들을 재결합 MOP의 destination레지스터에 추가한다.

4. DDM의 구성

종래는 원시 마이크로프로그램으로 부터 데이타의

종속관계와 자원상충관계를 DDG(또는 DAG)에 나타내던 것을 본 논문에서는 (정의11)~(정의13)에 따라 삼각행렬상에 다음과 같은 방법으로 나타낸다.

(1) 선택된 트레이스상에 나타난 원시 마이크로 프로그램의 MOP 순서대로 각 MOP에 행렬의 행과 열의 인덱스를 가한다.

(2) 원시 프로그램상에서 모든 i에 대해 i<j인 MOP_i와 MOP_j가 데이타 종속관계가 있을 때 DDM에서 i번 행의 j번 열에 D로써 데이타 종속관계를 대각선 상위부분에 나타낸 초기 DDM을 구한다.

(3) 각행의 자손 MOP의 집합을 구하고 D가 없는 해당 열에 D를 삽입한다. 이때 i번째행의 자손 MOP 집합을 DS(i)라 할 때
i<j<k…<m이고 i 행의 j, k, …, m열에 D가 있다면
DS(i)={j, k, …, l} ∪ DS(j) ∪ DS(k) ∪…∪ DS(m)가 된다.

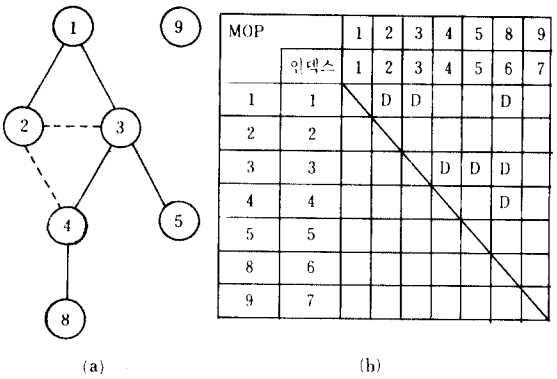
(4) 모든 i에 대해 i<j인 MOP_i와 MOP_j가 데이타 종속관계는 없으나 자원상충관계가 있으면 DDM에서 i번 행의 j번 열에 H로써 대각선 상위부분에 나타내어 최종 DDM을 구한다.

(정리 1) 초기 DDM에서 i번째 행의 자손 MOP 집합을 DS(i)라 할 때
i<j<k…<m이고 i행의 j, k, …, m열에 D가 있다면
DS(i)={j, k, …, m} ∪ DS(j) ∪ DS(k) ∪…∪ DS(m)
이다.

(증명) MOP_j, MOP_k, …, MOP_m 각각에 대해 MOP_i가 선행하므로 MOP_i와 MOP_j, MOP_k, …, MOP_m은 각각 ddd(directly data dependent)[3] 관계에 있다. 또한 MOP_j, MOP_k…, MOP_m 각각에 후행하면서 ddd관계에 있는 MOP들은 MOP_i와 dd(data dependent)관계에 있다.[3] 마찬가지방법을 차 하위의 MOP's에 대해서도 반복적으로 적용하면 MOP_i와 최하위의 MOP's와도 dd관계가 성립하므로, MOP_i와 ddd나 dd관계를 갖는 MOP들은 MOP_i로 부터 모두 경로를 갖는 자손 MOP 집합이다. Q. E. D.

label	inst.	next-addr.	dest-reg.	src-reg.	resource-vec.
1	CONT		R 1	R1, R3	0 1 1 0
2	CONT		R 3	R1, R3	0 1 0 0
3	CONT		R 6	R1, R4	0 1 0 1
4	CONT		R 4	R2, R5	0 1 0 0
5	GOTO	8	R 7	R5, R6	0 0 1 0
8	CONT		R 4	R1	0 0 0 1
9	STOP		R10	R5	1 0 0 0

그림 2. 원시 마이크로 프로그램 예
Fig. 2. An example source microprogram.



MOP		1	2	3	4	5	8	9
	인덱스	1	2	3	4	5	6	7
1	1		D	D			D	
2	2							
3	3				D	D	D	
4	4						D	
5	5							
8	6							
9	7							

그림 3. (a) DDG와 (b) 초기 DDM
Fig. 3. (a) DDG and. (b) Inital DDM.

그림 2는 DDM구성 예를 보이기 위한 원시 마이크로프로그램이고, 그림 3(a)는 데이타 종속관계와 자원상충관계를 나타내는 종래의 DAG이며 그림3(b)는 초기 DDM이다. 그림3(b)로 부터 구한 각 행의 자손 MOP집합 DS는 다음과 같다.

DS(7)={ϕ}
DS(6)={ϕ}
DS(5)={ϕ}
DS(4)={6} ∪DS(6)={6}
DS(3)={4, 5, 6} ∪DS(4) ∪DS(5) ∪DS(6)={4, 5, 6}
DS(2)={ϕ}
DS(1)={2, 3, 6} ∪DS(2)∪DS(3)∪ DS(6)
={2, 3, 4, 5, 6}

이와 같은 연산은 resource벡터로 부터 자원 상충관계를 구할 때와 마찬가지로 bit벡터를 이용하여 쉽게 구할 수 있다.

위 집합으로 부터 해당 행의 해당 열(각 집합의 원소)에 D가 없는 곳에 D를 삽입하고 자원 상충관계를 나타내면 그림 4 와 같이 최종 DDM이 쉽게 구해진다.

5. 우선순위

앞에서 구성된 DDM상에서 각 MOP의 자손 MOP수(각 행에서의 D수)와 조상 MOP수(각 열에서의 D수)를 이용하여 아래와 같이 우선순위를 정한다. 최적화시 모든 선행MOP가 최적화된 MOP중 우선순위가 높은 순으로 MOP를 최적화하기 위함이다.

(1) 자손 MOP수가 큰 MOP를 우선한다.

(2) 자손 MOP수가 같은 경우 조상 MOP수가 큰 MOP를 우선한다.[13-14]

(3) 둘다 동일할 경우 원시 프로그램에서 먼저 오는 MOP를 우선한다.

MOP		1	2	3	4	5	8	9
	인덱스	1	2	3	4	5	6	7
1	1		D	D	D	D	D	
2	2			H	H			
3	3				D	D	D	
4	4						D	
5	5							
8	6							
9	7							

그림 4. 최종 DDM
Fig. 4. The final DDM.

(4) 자손 MOP와 조상 MOP가 없는 MOP는 우선순위 결정에서 제외한다.

(5) 마지막 MOP가 접합 블럭의 조건부 분기이거나 출구 MOP이면 우선순위결정에서 제외시킨다.

그림 5는 그림 4의 DDM에 대한 우선순위 결정 예를 나타낸다.

MOP		1	2	3	4	5	8	9
	인덱스	1	2	3	4	5	6	7
1	1		D	D	D	D	D	
2	2			H	H			
3	3				D	D	D	
4	4						D	
5	5							
8	6							
9	7							

자 손 MOP수 (DN)	조 상 MOP수 (AN)	우 선 순 위
5	0	1
0	1	6
3	1	2
1	2	3
0	2	5
0	3	4
0	0	/

그림 5. 우선순위 결정 예
Fig. 5. An example for the priority decision.

(정리 2) 자손 MOP와 조상 MOP가 없는 MOP는 우선순위를 갖는 MOP들이 최적화된 이후에 최적화시키는 것이 MI수를 줄일 수 있다.

(증명) 자손 MOP와 조상 MOP가 없는 MOP는 다른 MOP와 데이타 종속 관계가 없기 때문에 함께 놓일 수 있는 MI수가 많다. 만약 이 MOP가 먼저 최적화 되면 자원 상충에 의해 다른 MOP가 최적화되지 않는 경우가 발생하여 MI수가 증가하게 된다. 따라서 그 트레이스상의 다른 모든 MOP를 먼저 최적화시키고 데이타종속관계가 없는 MOP를 자원 상충이 없는 적절한 MI에 놓음으로써 MI수를 줄일 수 있다. Q. E. D.

(정리 3) 현 트레이스상에서 마지막 MOP가 접합 블럭의 조건부 분기MOP이면, 현 트레이스의 맨 마지막 MI로 최적화 시킴으로써 블럭 복사를 없앨 수 있다.

(증명) 만일 분기MOP 위에 있던 MOP가 아래로 이동하여 최적화 되었다면 이동된 MOP는 다음에 오는 한쪽 경로상에서는 실행될 수 있지만 분기되는 다른 경로상에는 실행될 수 없고, 또한, 이동된 MOP가 분기MOP 이전의 다른 경로상의 레지스터값을 overwrite한다면 원래의 MOP 이후의 모든 MOP를 복사해야 한다. Q. E. D.

(정리 4) 현 트레이스상에서 마지막MOP가 원시프로그램상의 출구MOP이면 맨 마지막 MI로 최적화 시킴으로써 원시프로그램 의미의 변경을 방지할 수 있다.

(증명) 출구MOP 이후로 이동된 MOP는 그 프로그램에서는 실행이 되지 않으므로 원래의 프로그램 의미를 변경시킬 수 있다. Q. E. D.

6. 트레이스의 최적화

선택된 트레이스를 최적화하는 알고리즘은 다음과 같다.

입력 : 선택된 트레이스에 있는 MOP's의 시퀀스

출력 : 최적화된 MI's의 시퀀스

절차 :

(1) DDM구성

(2) 각 MOP의 우선순위를 결정하고 우선순위가 지정된 모든 MOP들의 플래그를 1로 set시킨다.

(3) i = 1로 하여 MI(i)를 현MI로 한다.

(4) 플래그가 set된 MOP에 대해 열순으로 그 열에 D가 없는 열들의 집합S를 구한다.

(5) S의 모든 MOP에 대해 다음 과정을 수행한다.

① 현MI와 자원상충을 나타내는 H가 없고 S중 우선순위가 제일 높은 MOP를 현MI에 부가하고 해당 프래그를 clear한다.

② 부가된 MOP의 해당 행에서 플래그를 갖는 모든 MOP의 D를 삭제한다.

(6) 플래그가 clear되지 않은 MOP가 존재하면 i를 1증가시키고 (4)로 간다.

(7) 접합 블럭의 조건부 분기 MOP가 있으면 다음 과정을 수행한다.

① 조건부 분기 MOP의 열에서 현MI에 있는 MOP's의 행에 D나 H가 없으면 ③으로 간다.

② i를 1증가시켜 현MI로 한다.

③ 현MI에 조건부 분기 MOP를 부가한다.

(8) 자손 MOP수(DN)와 조상 MOP수(AN)가 모두 0인 MOP가 존재하면 자원 상충이 없는 MOP가 있는 MI중에서 가능한 복사를 하지 않는 MI에 부가하고 최적화 절차를 마친다.

위(8)에서 dominant 블럭에 있던 MOP이면 가능한 dominant 블럭에, 그렇지 않은 경우는 가능한 원래 위치에 가까운 MI에 두도록 한다.

7. 북키핑

북키핑에 앞서 최적화 과정을 수행한 트레이스를 진입블럭으로 부터 조건부 분기 MOP나 재결합 MOP가 있는 MI를 기준으로 블럭의 경계를 정하여 기본블럭들로 분할한다.

그 다음, 최적화 과정에 의해 본래의 프로그램 의미를 변경 시키는 경우를 방지하기 위해 참고문헌[13], [14]에서와 같은 북키핑 법칙을 최적화된 트레이스에 적용 시킨다.

Ⅳ. 적용 예

본 알고리즘을 VAX-11/750시스템을 이용하여 FORTRAN으로 프로그램하고 Lah의 예[9]에 적용시켜서 종래의 알고리즘과 비교 검토하였다.

block	label	instruction	next-addr.	dest-reg.	src-reg.	resource-vec.
A	1	CONT		1	2, 3	0 1 1 0
	2	CONT		10	1, 5	0 0 1 0
	3	CJMP	7	2	3, 4	1 0 1 0
B	4	CONT		8	9, 5	0 1 0 0
	5	CONT		2	2, 5	0 0 0 1
	6	GOTO	10	15	4, 8	0 1 0 0
C	7	CONT		9	8, 8	0 1 0 1
	8	CONT		1	1, 5	0 1 0 0
	9	CONT		8	11, 12	0 0 0 1
D	10	CONT		14	2, 13	0 1 0 1
	11	CONT		7	3, 5	0 1 0 0
	12	CJMP	15	6	4, 10	1 0 0 0
E	13	CONT		9	3, 4	1 0 1 0
	14	GOTO	19	6	3, 6	0 1 0 0
F	15	CONT		2	6, 8	0 0 0 1
	16	CONT		7	2, 14	0 1 0 1
	17	CONT		4	2, 15	0 1 0 0
	18	CONT		14	11, 12	0 0 1 0
G	19	CONT		9	6, 10	0 0 0 1
	20	CONT		13	7, 3	0 0 0 1
	21	STOT		3	4, 11	0 1 1 0

그림 6. Lah예의 원시 마이크로 프로그램
Fig. 6. Source microprogram of Lah's example.

Lah의 예는 그림6과 같이 21개의 MOP's로 작성된 마이크로 프로그램이고 흐름 그래프는 그림7과 같다.

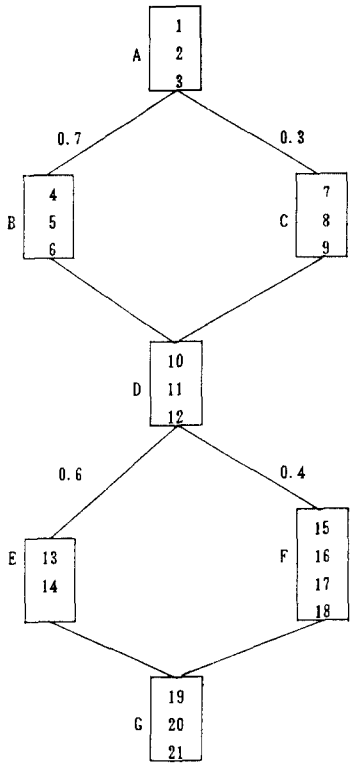


그림 7. Lah 예의 흐름 그래프
Fig. 7. The flow graph of Lah's example.

Live레지스터 분석 결과는 그림8과 같다.

IN [A] = 2 3 4 5 8 9 11 12
OUT[A] = 1 2 3 4 5 8 9 10 11 12 13
IN [B] = 2 3 4 5 9 10 11 12 13
OUT[B] = 2 3 4 5 8 10 11 12 13
IN [C] = 1 2 3 4 5 8 10 11 12 13
OUT[C] = 2 3 4 5 8 10 11 12 13
IN [D] = 2 3 4 5 8 10 11 12 13
OUT[D] = 3 4 5 6 7 8 10 11 12 14
IN [E] = 3 4 6 7 10 11
OUT[E] = 3 4 6 7 10 11
IN [F] = 3 5 6 8 10 11 12 14
OUT[F] = 3 4 6 7 10 11
IN [G] = 3 4 6 7 10 11
OUT[G] =

그림 8. 각 블럭의 live레지스터
Fig. 8. Live registers of each block.

먼저 그림 7 의 흐름 그래프에서 기대값에 따라 처음 선택된 트레이스는 A-B-D이다. 이때 D블럭이 접합 블럭D에서 트레이스는 잘린다. 선택된 트레이스A-B-D에 대한 초기 DDM은 그림 9 와 같다.

MOP		1	2	3	4	5	6	10	11	12
	인덱스	1	2	3	4	5	6	7	8	9
1	1		D	D		D				
2	2									D
3	3				D	D		D		
4	4						D			
5	5							D		
6	6									
10	7									
11	8									
12	9									

그림 9. 트레이스A-B-D의 초기 DDM
Fig. 9. The initial DDM of trace A-B-D.

이때 조건부 분기 MOP 3에 대해서 이 MOP의 source-registers에 조건부 source레지스터 집합 IN[C]가 포함되었다.

초기 DDM에서 각 MOP에 대한 자손 MOP집합은 다음과 같다.

$DS(9)=\{\phi\}$

$DS(8)=\{\phi\}$

$DS(7)=\{\phi\}$

$DS(6)=\{\phi\}$

$DS(5)=\{7\} \cup DS(7)=\{7\}$

$DS(4)=\{6\} \cup DS(6)=\{6\}$

$DS(3)=\{4,5,7\} \cup DS(4) \cup DS(5) \cup DS(7)=\{4,5,6,7\}$

$DS(2)=\{9\} \cup DS(9)=\{9\}$

$DS(1)=\{2,3,5\} \cup DS(2) \cup DS(3) \cup DS(5)$

$=\{2,3,4,5,6,7,9\}$

따라서, 최종 DDM은 그림10(a)와 같고 이것의 우선순위는 그림10(b)와 같다.

트레이스 A-B-D에 대해서 최적화된 결과는 그림11과 같고 조건부 분기 MOP 3과 12, 재결합MOP 10에 의해 블럭A′, B′, D′가 분할 되었다.

이때 MOP 2가 조건부 분기 MOP 3 아래로 이동했기 때문에 블럭C에 복사된다. 트레이스 D-E-G에 대해서도 최종 DDM과 우선순위는 그림12와 같다. 이를 최적화한 결과는 그림13과 같다.

트레이스D-E-G의 최적화에서도 MOP20이 재결합

MOP		1	2	3	4	5	6	10	11	12
	인덱스	1	2	3	4	5	6	7	8	9
1	1		D	D	D	D	D	D	H	D
2	2			H						D
3	3				D	D	D	D		H
4	4						D	H	H	
5	5							D		
6	6							H	H	
10	7								H	
11	8									
12	9									

(a)

자손 MOP수 (DN)	조상 MOP수 (AN)	우선순위
7	0	1
1	1	5
4	1	2
1	2	3
1	2	4
0	3	6
0	3	7
0	0	/
0	2	/

(b)

그림10(a). 최종 DDM (b) 우선순위
Fig. 10(b). The final DDM. (b) The priority.

사이클	MOP's	블 럭
1	1	A′
2	3, 11	
3	2, 4, 5	B′
4	6	
5	10, 12	D′

그림11. 트레이스 A-B-D의 최적화 결과
Fig. 11. The compaction result of the trace A-B-D.

MOP		10	12	13	14	19	20	21
	인덱스	1	2	3	4	5	6	7
10	1				H	H	D	D
12	2			H	D	D		D
13	3					D		D
14	4					D		D
19	5						H	
20	6							D
21	7							

(a)

자손 MOP수 (DN)	조상 MOP수 (AN)	우선순위
2	0	3
3	0	1
2	0	4
2	1	2
0	3	6
1	1	5
0	5	/

(b)

그림12. (a) 트레이스 D-E-G의 최종 DDM (b) 우선순위
Fig. 12. (a) The final DDM of trace D-E-G.
(b) The priority.

MOP 19 위로 이동하였기 때문에 블럭F에 복사된다. 블럭C와 블럭F에 대해서도 마찬가지 방법으로 최적화된 결과는 그림14와 같다.

따라서 Lah예에 대한 본 알고리즘을 적용시켜 최적화된 결과는 그림15와 같다.

사이클	MOP's	블 럭
1	10, 12	D′
2	13, 14, 20	E′
3	19, 21	G′

그림13. 트레이스 D-E-G의 최적화 결과
Fig. 13. The compaction result of the trace D-E-G.

사이클	MOP's	블럭
1	2, 7	C′
2	8, 9	

(a)

사이클	MOP's	블럭
1	15	F′
2	16	
3	17, 18, 20	

(b)

그림14. (a) 블럭C의 최적화 결과 (b) 블럭F의 최적화 결과
Fig. 14. (a) The compaction result of block C. (b) The compaction result of block F.

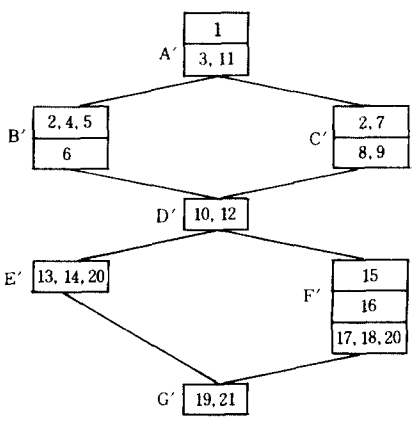


그림15. Lah 예에 본 알고리즘 적용후 최적화된 결과
Fig. 15. The compaction result of Lah's example applied to the proposed algorithm.

V. 종래 알고리즘과의 비교 및 검토

Lah의 예에 대해 본 알고리즘과 Trace scheduling, Tree compaction을 적용했을 때 비교해 보면 표1과 같다.

표1에서 비교된 바와 같이 사이클수, 복사 MOP수 및 총 space*시간에 있어서 모두 줄어듬을 알 수 있다.

여기서 HWFC 알고리즘[13-14]과는 같은 결과이다. 그러나 HWFC 알고리즘에서는 A-B-D-E-G를 하나의 트레이스로 하여 최적화하므로, 원시 프로그램의 MOP6이 D-E-G 트레이스상의 어떠한 MOP와도

표 1. Lah 예에 대한 본 알고리즘과 trace schduling, tree compaction과의 비교
Table. 1. The comparison of the t proposed algorithm with trace scheduling and tree compaction for Lah's example.

	Trace scheduling	Tree compaction	본 알고리즘
A-B-D-E-G사이클수	7	8	7
총 사이클 수	33	34	32
총 실행시간	8.1	8.4	7.8
총복사 MOP수	17	3	2
총 space수	21	13	12
총 space *시간	170.1	109.2	93.6

데이타 종속관계가 없으나 모두 자원 상충관계 때문에 원래의 MOP순서에 맞게 B′블럭에 놓여진 것이므로 만일 MOP21이 MOP6과 자원 상충관계가 없다면 Trace schduling시와 같은 많은 블럭복사를 요한다. 그러나 이러한 문제는 접합블럭으로 트레이스를 분할함으로써 블럭복사를 방지할 수 있음을 알 수 있다.

MOP's의 source와 destination 레지스터로부터 데이타 종속관계와 자원 상충관계를 나타내기 위해, DAG구성시는 링크 조정이 복잡할 뿐아니라, 구성된 DAG로 부터 우선순위를 정하기 위해 자손 노드수와 조상 노드수를 구하는 운행 알고리즘이 또한 간단하지 않다.[15]

그러나 삼각행렬상에서는 데이타 종속관계는 bit벡터의 ORing을 이용하여 해당 난에 D로써 쉽게 나타낼 수 있고 자원 상충관계도 bit벡터의 ANDing을 이용하여 해당 난에 H로써 쉽게 표시할 수 있다. 또한, 자손 MOP수는 해당 행의 D수가 되고 조상 MOP 수는 해당 열의 D수가 되므로 우선순위도 간단히 구해진다.

한편, 지금까지는 초기 DDM으로 부터 최종 DDM을 구하여 해당 행과 열의 D수로써 자손 MOP수와 조상 MOP수를 계산하고 우선순위를 결정한 후에 DDM과 우선순위를 이용하여 최적화를 수행하였으나, 초기 DDM에 자원 상충관계만 포함하여 최적화에 이용할 수도 있다. 이때 자손 MOP수는 앞에서와 같이 (정리1)을 이용하여 구한 집합의 원소 수(cardinality) 가 된다.

즉, i번째 행의 자손MOP수=|DS(i)|; DS(i)의 원소수 또한 조상 MOP수도 같은 원리로 구할 수 있다.

(정리5) 초기 DDM에서 j번째 열의 조상 MOP집합을 AS(j)라 할 때, g<h<…<i<j이고 j열의 g, h, …, i행에 D가 있다면 AS(j)={g, h, …, i} ∪ AS(g) ∪ AS

(h) ∪ … ∪ AS(i) 이다(증명생략).

그러므로, j번째 열의 조상 MOP수 = | AS(j) | ; AS(j)의 원소 수.

따라서 Lah 예의 트레이스 A-B-D에 대해 최적화 알고리즘을 적용하기 위한 이같은 형식의 DDM은 원시 프로그램으로 부터 그림16과 같이 되고 각열의 조상MOP 수는 다음과 같이 계산된다.

MOP		1	2	3	4	5	6	10	11	12
	인덱스	1	2	3	4	5	6	7	8	9
1	1		D	D	H	D	H	H	H	
2	2			H						D
3	3				D	D		D		H
4	4						D	H	H	
5	5							D		
6	6							H	H	
10	7								H	
11	8									
12	9									

그림16. 트레이스 A-B-D에 대해 다른 형태의 DDM
Fig. 16. Anther DDM for trace A-B-D.

$AS(1)=\{\phi\}$ $|AS(1)|=0$

$AS(2)=\{1\}\cup AS(1)=\{1\}$ $|AS(2)|=1$

$AS(3)=\{1\}\cup AS(1)=\{1\}$ $|AS(3)|=1$

$AS(4)=\{3\}\cup AS(3)=\{1,3\}$ $|AS(4)|=2$

$AS(5)=\{1,3\}\cup AS(1)\cup AS(3)=\{1,3\}$ $|AS(5)|=2$

$AS(6)=\{4\}\cup AS(4)=\{1,3,4\}$ $|AS(6)|=3$

$AS(7)=\{3,5\}\cup AS(3)\cup AS(5)=\{1,3,5\}$ $|AS(7)|=3$

$AS(8)=\{\phi\}$ $|AS(8)|=0$

$AS(9)=\{2\}\cup AS(2)=\{1,2\}$ $|AS(9)|=2$

그림10과 같은 DDM은 행렬을 수정함으로써 자손과 조상 MOP를 쉽게 찾거나 계산할 수 있는 반면, 그림16은 DDM을 수정은 안하나 자손 MOP수를 다시 계산해야 한다는 장, 단점이 있다.

여기서는, 데이타 종속관계 및 자원상충관계를 DAG 대신 DDM으로 구성하는 방법만을 제안하였으나 행렬을 흐름 그래프 대신 이용할 수도 있으며 루프를 갖는 경우도 이용 가능하다.

Ⅵ. 결 론

본 논문에서는 마이크로프로그램의 최적화 과정에서 필요한 MOP's의 데이타 종속관계와 자원 상충관계를 삼각행렬 상에 간단하게 나타내어 이용할 수 있는 방법을 제시하였고, 또한 접합블럭으로 트레이스를 분할함으로써 블럭복사를 방지할 수 있는 마이크로프로그램의 광역적 최적화 알고리즘을 제안하였다.

제안된 DDM과 최적화 알고리즘을 Lah의 예에 적용시켰으며, 제안된 알고리즘이 종래의 알고리즘보다 실행시간과 제어기억용량을 더 효율적으로 감소시킬 수 있음을 보였다.

루프를 갖는 마이크로프로그램에도 제안된 DDM과 최적화 알고리즘을 확장 적용하는 것은 앞으로의 연구 대상이 될 수 있으며, HLML과 그 컴파일러와 같은 마이크로프로그래밍의 도구 개발은 더욱더 연구해야 할 분야로 남아있다.

參 考 文 獻

[1] D.K. Banerji et al., *Elements of Microprogramming,* Prentice Hall, 1982.

[2] B. Su and S. Ding, "Some experiments in global microcode compaction," *IEEE Proceedings the 18th Annual Workshop on microprogramming,* pp. 175-180, Dec. 1985.

[3] D. Lanskov et al., "Local microcode compaction techniques," *ACM computing Surveys,* vol. 12, no. 3, pp. 261-294, Sept. 1980.

[4] J.A. Fisher, "Trace scheduling: A technique for global microcode compaction," *IEEE Trans. on Computers,* vol. C-30, no. 7, pp. 478 - 490, July 1981.

[5] W.G. Good, "Global optimization of microprograms through modular constructs," *IEEE Proceedings the 12th Annual Microprogramming Workshop,* pp. 1-6, Nov. 1979.

[6] M. Tokoro et al., "Optimization of microprograms," *IEEE Trans. on Computers,* vol. C-30, no. 7, pp. 491-504, July 1981.

[7] J.A. Fisher, *The Optimization of Horizontal Microcode within and beyond Basic Blocks: An Application of Processor Scheduling with Resources,* Ph.D. thesis, New York Univ., Oct. 1979.

[8] S. Isoda et al., "Global compaction of horizontal microprograms based on the generalized data dependency graph," *IEEE Trans. on Computers,* vol. C-30, no. 10, pp. 922-933, Oct. 1983.

[9] J. Lah and D.E. Atkins, "Tree compaction

of microprograms," *IEEE Proceedings the 16th Annual Microprogramming Workshop*, pp. 11-22, Oct. 1983.

[10] R.P. Atkins, "Improved instruction formation in the exhaustive local microcode compaction algorithm," *IEEE Proceedings the 17th Annual Microprogramming Workshop*, pp. 105-111, Oct. 1984.

[11] A.V. Aho and J.D. Ullman, *Principles of Compiler Design*, Addison Wesley, 1977.

[12] M.S. Hecht, *Flow Analyais of Computer Programs*, North Holland, N.Y., 1977.

[13] 이상정외 3인, "Total execution time 및 control memory space의 감소를 위한 microprogram의 광역적 최적화 기법", 한국정보과학회논문지, vol. 11, no. 3, pp. 210-222, 8월 1984년.

[14] 조영일, 마이크로프로그램의 최적화에 관한연구, 한양대학교 대학원, 박사학위 논문, 6월 1985년.

[15] B. Kolman and R.C. Busby, *Discrete Mathematical Structures for Computer Science*, Prentice Hall, 1984.