□特 輯□

최적화 문제에서 그래프의 응용

홍익대학교 전자계산학과 정 균 락*

목 차

I. 서 론

주어진 문제를 해결하기 위해서는 문제에 대한 정확한 정의와 표현이 필요하다. 문제의 표현을 위해서는 문제의 성격에 따라 수학적 모델, 트리, 그래프등이 많이 사용되는데, 그 이유는 이미 잘 알려진 모델로 표현이 가능하거나 변형될 수 있으면 문제의 해결이 훨씬 용이해지기 때문이다. 예를 들어, 어떤 문제가 그래프 상에서 해를 구하는 문제로 변형된다면, 문제 자체에 대한 이해도 쉽고 이미 많이 개발된 그래프에 관련된 알고리즘들을 적용시킬 수가 있다.

본 논문에서는 최적화 컴파일러에서 매우 중요한 단계인 레지스터 할당(register allocation) 문제를 그래프로 표현하고, 그래프 착색(graph coloring) 알고리즘을 사용해서 해결하는 방법, VLSI(Very Large Scale Integration) 설계 과정 중의 하나인 floorplanning에서 면적이 최소가 되도록 각 블럭의 dimension을 결정할 때, floorplan을 X-그래프와 Y-그래프로 표현하여 면적을 구하는 방법과 전체 선의 길이의 합이 최소가 되도록 모듈의 방향(module orientation)을 결정하는 문제에서 다단계 그래프(multistage graph)를 구성하여 각 모듈들의 최적 방향을 결정하는 방법을 고찰하기로 한다.

* 종신회원

II. 레지스터 할당 문제

레지스터 할당이란 컴파일러 후반부(back end)에서 수행되는 최적화 단계 중의 하나로, 변수들과 컴파일러가 발생시키는 임시변수들 중에 레지스터에 상주시킬 것들을 선택하여 물리적 레지스터(hardware register)에 할당하는 작업을 말한다. 레지스터 할당의 목적은 프로그램의 수행중 레지스터를 참조(access)하는 것이 메모리를 참조하는 것보다 훨씬 빠르므로 사용 빈도가 많은 변수들의 값을 가능하면 레지스터에 둠으로써 레지스터에서 메모리로 또는 메모리에서 레지스터로의 이동을 최소화하여 프로그램의 실행 시간을 줄이는 것이다.

레지스터 할당 문제는 그래프 착색(graph coloring) 문제로 변환될 수 있는데, 그래프 착색이란 주어진 그래프의 각 노드에 색을 할당하는데 인접한 노드 사이에는 같은 색을 줄 수가 없다. 주어진 그래프가 k색으로 착색이 가능한가 하는 문제는 NP-complete[4]이나 우수한 휴리스틱이 있고, 또 항상 최적해가 필요한 것이 아니므로 실제적으로는 큰 문제가 되지 않는다. Chaitin은 실제로 IBM System/370의 실험적 PL/I 컴파일러에서 그래프 착색 문제를 레지스터 할당에 적용하였다[1]. Chow는 우선순위 개념을 도입하여 우선순위가 높은 변수에 먼저 색을 할당하는 개선된 방법을 제안하였다

[3].

레지스터 할당 문제는 다음과 같은 과정을 통해 그래프 착색 문제로 변환될 수 있다[3].

2.1 생존 범위(live range)의 계산

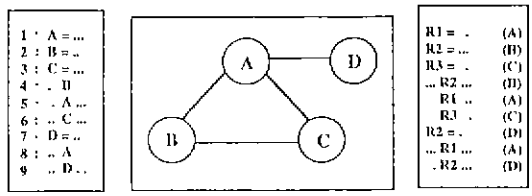
생존 범위라는 것은 변수가 프로그램 상에서 제일 처음 정의된 곳으로부터 마지막 사용된 곳까지를 의미한다. 그림 1(a)에서 변수 다음에 =이 있으면 변수가 정의(definition)된 것이고, 없는 경우는 사용(use)된 것이라고 가정하면 변수 A는 첫 번째 줄에서 정의되고 다섯 번째와 여덟 번째에서 사용되었으므로 생존 범위는 (1, 8)이 된다. 같은 방법으로 B의 생존 범위는 (2, 4), C의 생존 범위는 (3, 6), D의 생존 범위는 (7, 9)가 된다. 두 개의 생존 범위가 서로 겹치게 되면 간섭(interfere)한다고 하는데, A와 B는 서로 간섭하고 B와 D는 서로 간섭하지 않는다. 여기서 간섭의 의미는 서로 간섭하는 두 생존 범위에는 같은 레지스터를 할당할 수 없음을 뜻한다.

2.2 간섭 그래프의 구성

간섭 그래프의 각 노드는 생존 범위가 되고, 두 생존 범위가 서로 간섭하면 대응되는 노드 사이에 에지가 존재하게 된다. 그림 1(a)에 해당하는 간섭 그래프가 그림 1(b)에 있다.

2.3 그래프 착색

사용할 수 있는 색의 수가 k라고 하면 (즉 k개의 레지스터를 사용할 수 있으면) 그래프에서 인접한 노드의 수(차수)가 k보다 작은 노드를 순서를 기억하며 제거해 나간다. 노드를 제거하면 그 노드에 인접해 있던 노드들의 차수가 하나씩 감소하게 되므로, 이 과정을 반복하면 더 이상 그래프에 노드가 남아 있지 않거나, 그래프에 남아 있는 모든 노드들이 k보다 많은 이웃 노드들을 가지고 있게 된다. 노드가 모두 제거된 경우에는 제거된 순서의 역순으로 색을 할당함으로써 착색을 모두 끝낼 수 있다. 예를 들어 그림 1(b)에서 K=3이라고 할 때 D의 차수가 1이므로 D를 먼저 제거하면 A, B, C 모두 차수가 2가 되고 모두 제거가 가능하다. A, B, C 순으로 제거했다고 가정하면 C, B, A, D 순으로 색을 주게된다. 그림 1(c)는 C가 R_3, B와 D가 R_2, A가 R_1 색(레지스터)을 할당받았을 때의 결과를 나타내고 있다. 모든 노드가 다 제거되지 않았을 때는 이 중 데피 노드(spilled node)를



a) 프로그램의 일부분 b) 간섭 그래프의 구성 c) 레지스터 할당 결과

(그림 1)

선택하여 그래프로부터 제거한다. 데피 노드는 레지스터를 할당받지 않으며 그 변수의 값을 메모리에 유지하고 필요할 때마다 load/store하게 된다. 여기서는 레지스터 할당의 기본 개념만을 소개하며 자세한 내용은 [3]을 참조하기 바란다.

위에서 본 바와 같이, 레지스터 할당 문제는 그래프에서의 착색 문제로 잘 표현되고 있다.

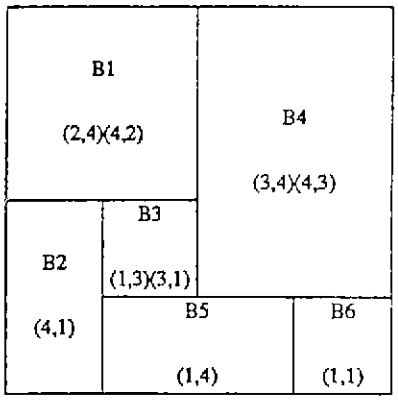
III. Floorplanning 문제

VLSI 칩(chip) 설계에 있어서 floorplan은 블럭(building block)들로 이루어져 있다. 설계 초기 단계에 있어서 각 블럭들의 상대적 위치는 고정되어 있고 대강의 면적은 추정될 수 있으나, 실제로 실현(realization) 했을 때의 dimension은 달라질 수 있다. 예를들어, 면적이 12인 블럭이 있다면 1×12, 2×6, 3×4, 4×3, 6×2, 12×1 등으로 실현될 수 있다.

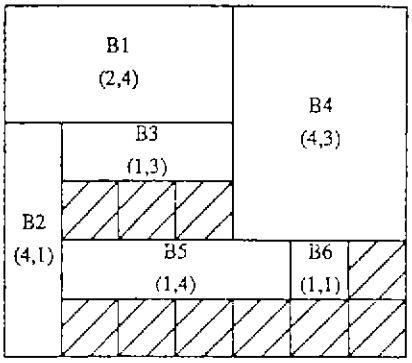
그림 2에 6개의 블럭으로 이루어진 floorplan의 예가 나타나 있다. 여기서 B_1은 2×4 또는 4×2로 실현될 수 있고 B_5는 1×4로만 실현 가능하다. Floorplanning의 목적은 floorplan의 전체 면적이 최소가 되거나 원하는 aspect ratio(가로와 세로의 비율)를 갖도록 각 블럭들의 dimension을 결정하는 데 있다.

Floorplan의 목적이 전체 면적을 최소화 하는 것이라 가정할 때, 그림 2에 해당하는 floorplan의 최적해가 그림 3처럼 주어진다. 이 예에서는 높이가 6이고 폭이 7로 전체 면적은 42로 최소가 되며, 빗금친 부분은 사용되지 않는 공간을 의미한다.

Floorplan은 두 개의 dag(directed acyclic graph)를 사용해서 표현할 수 있는데 하나는 X-그래프, 다른 하나는 Y-그래프라 불린다[7]. X-그래프의 각 정점(vertex)은 floorplan의 수직선에 대응한다. 정점 p에서 정점 q로의 방향성 에지(edge)를 왼쪽 변이 수직선 p의 일부이고 오른쪽 변이 수직선 q의 일부가 되는 블럭이 존재할 때 긋는다.



(그림 2)

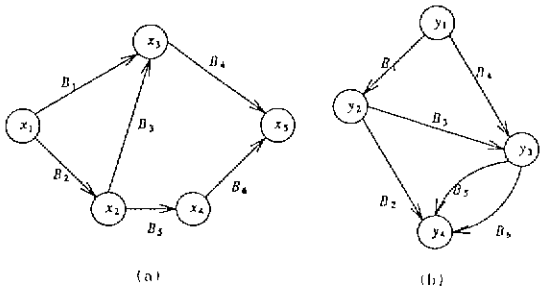


(그림 3)

그림 2에 주어진 floorplan의 X-그래프가 그림 4(a)에 나타나 있다. 그림 2에서 5개의 수직선을 왼쪽으로부터 x_1, x_2, x_3, x_4, x_5로 이름을 붙이면 그림 4(a)에는 이 5개의 수직선에 대응되는 5개의 정점 x_1, x_2, x_3, x_4, x_5가 존재하게 된다. 수직선 x_1과 x_3 사이에 블럭 B_1이 있으므로 정점 x_1에서 정점 x_3으로 가는 에지가 존재하게 된다. 이 에지에는 B_1이라는 라벨(label)이 붙어 있다.

Y-그래프의 각 정점은 floorplan의 수평선에 대응된다. 정점 p에서 정점 q로의 방향성 에지는 위쪽 변이 수평선 p의 일부이고, 아래쪽 변이 수평선 q의 일부가 되는 블럭이 존재할 때 긋는다. 그림 2에 주어진 floorplan의 Y-그래프가 그림 4(b)에 나타나 있다. 여기서 수평선 Y_3과 Y_1 사이에는 두 개의 블럭 B_5와 B_6이 있으므로 정점 y_3에서 y_4로 가는 두 개의 에지가 존재한다.

각 블럭들의 dimension이 임의로 주어졌을 때, X-그래프의 에지에는 대응되는 블럭의 폭이 가중치로 주어지고 Y-그래프의 에지에는 높이가 주어진다. 이 때 전체 floorplan의 면적을 구하는 문제는 대응되는 X-그래프와



(그림 4)

Y-그래프에서 최장 경로의 길이를 구하는 문제와 같게 된다. 즉 X와 Y-그래프의 최장 경로의 길이를 각각 x, y라 하면 면적은 xy가 된다. 면적이 최소가 되는 floorplan을 찾기 위해서는 블럭들이 가질 수 있는 모든 가능한 dimension들에 대해 X와 Y-그래프를 구성하고 그 중 최적해를 구해야 하는데, 이 문제는 NP-hard이고 branch-and-bound와 같은 방법을 사용해서 해결할 수 있다[7].

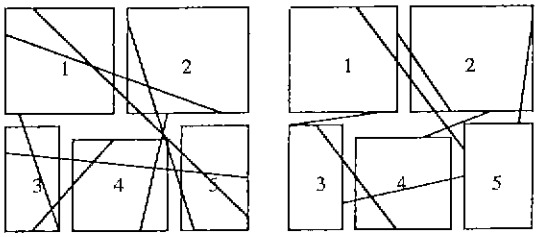
IV. 모듈 방향 결정 문제

4.1 개요

초고밀도 집적회로(VLSI) 설계 과정의 한 단계인 모듈의 방향 결정 문제는, 각 모듈들의 위치가 어떤 Placement 알고리즘에 의해 결정된 후 각 모듈의 위치는 변화시키지 않고, 모듈들을 수평축 또는 수직축을 중심으로 뒤집거나(flipping) 회전(rotation)시킴으로써 회로의 효율성과 연결성(routability)을 향상시키는 방법으로 최근에 연구되기 시작했다. 속도가 빨라야 하는 고성능 회로에서는 가장 긴 선(wire)의 길이가 최소가 되게 각 모듈의 방향을 결정하는 것이 바람직하고, 그렇지 않은 경우에는 전체 선들의 길이의 합을 최소화하는 것이 바람직하다.

Libeskind-Hadas와 Liu[6]는 전체 선들의 길이의 합이 최소가 되도록 모듈의 방향을 결정하는 문제가 NP-hard임을 보였고, 신경회로망을 이용한 휴리스틱(heuristic)을 개발하였다. Yamada와 Liu[9]는 이 문제를 해결하기 위한 여러 가지 분석적 방법과 simulated annealing 방법을 실험하였다. K. Chong과 S. Sahni[2]는 가장 긴 선의 길이가 최소가 되도록 모듈의 방향을 결정하는 문제가 NP-hard임을 증명하였다.

모듈의 갯수가 5일 때의 예가 그림 5에 있다. 5(a)는 주어진 모듈들의 초기 상태이며, 5(b)는 모듈 방향이 결정된 후의 모듈들의 상태로서 모듈 1과 2는 수직축을



a) 초기 모듈들의 상태 b) 모듈 방향이 결정된 후의 모듈들의 상태

(그림 5) 모듈 방향 결정 문제

중심으로, 3과 5는 수직축과 수평축을 중심으로, 4는 수평축을 중심으로 뒤집어져 있다.

같은 선에 속해 있는 두 핀(pin)들 사이의 거리는 일반적으로 Manhattan 거리나 Euclidean 거리를 사용하여 구할 수 있다. 본 논문에서는 Manhattan 거리를 사용한다고 가정한다.

p, q를 같은 선에 속하는 두 개의 핀이라 하고, (x_p, y_p)와 (x_q, y_q)를 이 두 핀의 좌표라 하자. N을 모든 선들의 집합이라 하면

$$L=\Sigma[|x_p-x_q|+|y_p-y_q|]$$
$$(p,\ q)\in N$$

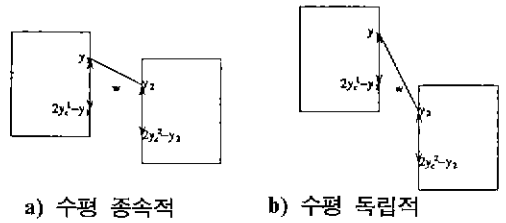
는 모든 선들에 대한 Manhattan 거리의 합이 된다. 모듈 방향 결정 문제는 L이 최소값을 가지도록 각 모듈들의 방향을 결정하는 문제이다.

4.2 모듈의 독립성 (Independence)

[a, b]들 두 점 a, b를 포함하는 폐구간이라 하자. 두 구간 [a, b]와 [c, d]는 서로 접치지 않으면 독립적이라 하고 아니면 종속적이라고 정의한다. M_1과 M_2가 중심이 각각 (x_c^1, y_c^1)과 (x_c^2, y_c^2)인 두 개의 모듈이라 하고 (x_1, y_1)이 M_1에 있는 핀이고 (x_2, y_2)가 M_2에 있는 핀이라 때 두 핀을 연결하는 선을 w라 하자.

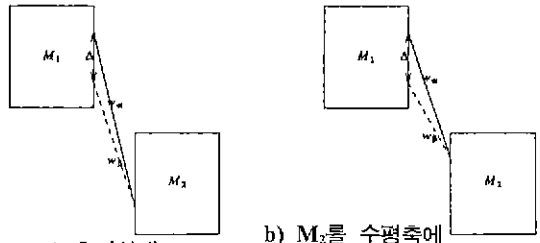
만약에 $[x_1, 2x_c^1-x_1]$과 $[x_2, 2x_c^2-x_2]$가 독립적이면 모듈 M_1과 M_2는 w에 관해 수직 독립적(vertically independent)이라 하고, $[y_1, 2y_c^1-y_1]$과 $[y_2, 2y_c^2-y_2]$가 독립적이면 w에 관해 수평 독립적(horizontally independent)이라 한다. 그림 6에 w에 관해 수평 종속적인 경우와 수평 독립적인 경우가 나타나 있다. M_1과 M_2가 두 모듈을 연결하는 모든 선에 관해 수직 독립적이고 수평 독립적이면 독립적이라 하고 아니면 종속적이라 한다.

모듈의 독립성 개념은 매우 중요한 의미를 가지고 있는데, 그림 7에서 (a)는 두 모듈의 초기상태이고 (b)는



a) 수평 종속적 b) 수평 독립적

(그림 6)



a) 초기상태 b) M_2를 수평축에 대해 뒤집은 후

(그림 7)

M_2를 수평축에 대해 뒤집었을 때의 결과를 나타내고 있다. 여기서 보면 모듈 M_2의 방향에 관계없이 모듈 M_1을 수평축에 대해 뒤집어야 w의 길이가 짧아짐을 알 수 있다. 즉 M_1과 M_2가 독립적이면 M_2의 방향에 관계없이 M_1의 방향을 결정할 수 있다. 그러므로 다음의 정리를 얻게 된다.

[정리 1]

M_1과 M_2가 서로 독립적인 모듈이면 M_1과 M_2의 최적 모듈 방향은 서로 독립적으로 결정될 수 있다.

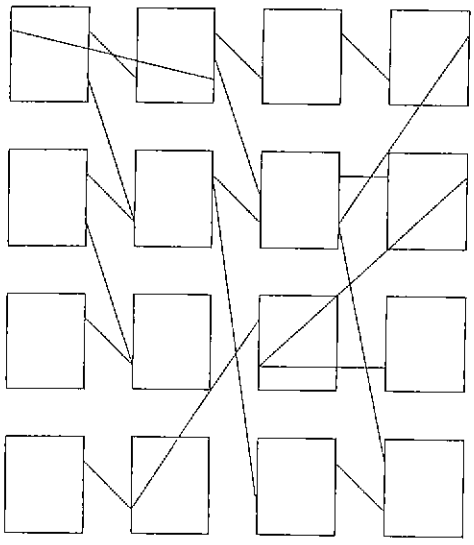
(증명) [2] 참조.

4.3 행렬 형태의 모듈

nm개의 모듈 M_{ij}가 그림 8과 같이 행렬 형태로 정렬되어 있고 선들은 서로 인접한 열에 있는 모듈들 사이에서만 연결되어 있다고 할 때, 전체 선의 길이의 합을 최소로 하는 문제는 다음과 같은 방법에 의해 선형시간에 해결될 수 있다.

$F=\{f_{ij} \mid f_{ij}\in\{\phi, v, h, b\}, 1\le i\le n, 1\le j\le m\}$를 각 모듈들의 방향이라 하자. 여기서 ϕ는 모듈이 그대로 있을 때, v는 모듈을 수직축에 대해 뒤집었을 때, h는 수평축에 대해 뒤집었을 때, b는 수직축과 수평축 모두에 대해 뒤집었을 때를 나타낸다.

L_{ij} $(f_{ij}, F-F_{ij})$를 모듈 M_{ij}의 방향이 f_{ij}이고 나머지 모듈들의 방향이 각각 $F-\{f_{ij}\}$일 때, 선의 한 핀이 모듈 M_{ij}에 연결된 모든 선들의 길이의 합이라 하면 L_{ij}는 R_{ij}와



(그림 8)

O_{ij}로 나누어 질 수가 있다. R_{ij}는 한 핀이 M_{ij}에 있고 다른 핀이 같은 i행에 있는 모듈들에 연결된 선들의 길이의 합이 되고, O_{ij}는 M_{ij}에 연결된 나머지 선의 길이의 합이 된다. 그러므로

$$L_{ij}(t_{ij},\ F-F_{ij})=R_{ij}(f_{ij},\ F-f_{ij})+O_{ij}(f_{ij},\ F-f_{ij}).$$

F_i, L_i, R_i와 O_i를 다음과 같이 정의한다.

$$F_i=\{f_{ij} \mid 1\le j\le m\}$$

$$L_i(F_i,\ F-F_i)=\sum_{j=1}^{m} L_{ij}(f_{ij},\ F-f_{ij})$$

$$R_i(F_i,\ F-F_i)=\sum_{i=1}^{m} R_{ij}(f_{ij},\ F-f_{ij})$$

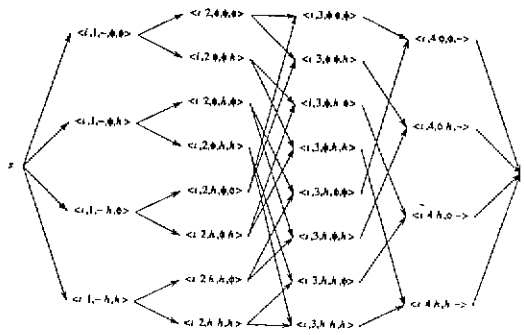
$$O_i(F_i,\ F-F_i)=\sum_{j=1}^{m} O_{ij}(f_{ij},\ F-f_{ij})$$

즉, $R_i(F_i,\ F-F_i)$는 i행에 있는 두 모듈들을 연결하는 선들의 길이의 합의 2배가 되고, $O_i(F_i,\ F-F_i)$는 한 핀이 i행에 있는 모듈에 있고 다른 핀은 i가 아닌 행에 있는 모듈에 있는 선들의 길이의 합이 된다. 위의 정의로부터 다음 식이 성립함을 쉽게 알 수 있다.

$$L_i(F_i,\ F-F_i)=R_i(F_i,\ F-F_i)=O_i(F_i,\ F-F_i)$$

모듈들의 방향이 F일 때 전체 선의 길이의 합을 ETL이라 하면 ETL은 다음과 같이 표현된다.

$$ETL=\tfrac{1}{2}\sum_{i=1}^{n} L_i(F_i,\ F-F_i)$$



(그림 9)

[정리 2]

$A=\{a_{ij}\}$와 $B=\{b_{ij}\}$를 nm개의 모듈 방향의 서로 다른 두 집합이라 하고, $A_i=\{a_{ij} \mid 1\le j\le m\}$와 $B_i=\{b_{ij} \mid 1\le j\le m\}$을 i행에 모듈들의 방향이라 한다. $\Delta A=O_i(A_i,\ A-A_i)-O_i(B_i,\ A-A_i)$이고 $\Delta B=O_i(A_i,\ B-B_i)-O_i(B_i,\ B-B_i)$라 하면 $\Delta A=\Delta B$.

(증명) [2] 참조.

[정리 2]는 i행에 있는 모듈들의 방향을 바꾸었을 때, 한 핀이 i행에 있고 다른 핀이 다른 행에 있는 선의 길이의 합에 미치는 영향은 다른 행에 있는 모듈들의 방향에는 독립적임을 의미하고 있다

[정리 3]

$L_i'(F_i,\ F-F_i)=O_i(F_i,\ F-F_i)+\tfrac{1}{2}R_i(F_i,\ F-F_i)$이라 하면 $L_i'(A_i,\ A-A_i)-L_i'(B_i,\ A-A_i)=L_i'(A_i,\ B-B_i)-L_i'(B_i,\ B-B_i)$이다.

(증명) [2] 참조.

[정리 4]

F를 모듈들의 방향이라 하고 X_i가 가능한 X_i값 중 $L_i'(X_i,\ F-F_i)$를 최소로 한다고 하면 $X=\cup X_i$는 ETL을 최소로 한다.

(증명) [2] 참조.

전체 길이의 합의 최소가 되는 모듈들의 최적 방향을 찾는 알고리즘은 [정리 4]에 그 토대를 둔다. 임의의 i에 대해 L_i'를 최소로 하는 X_i는 먼저 모든 모듈들의 방향을 임의로 결정해 준 다음 구할 수가 있다. 이 모듈 방향들의 집합을 F라 하고 편의상 $f_{ij}=\phi$, $1\le i\le n$, $1\le j\le m$이라 한다. 또 L''를 다음과 같이 정의한다.

$$L''(f_{ij},\ f_{i,j-1},\ f_{i,j-1},\ F-\{f_{ij},\ f_{i,j-1},\ f_{i,j+1}\})$$
$$=O_{ij}(f_{ij},\ F-f_{ij})+\tfrac{1}{2}R_{ij}(f_{ij},\ F-f_{ij}).$$

각 행에 대하여 m+2 단계로 이루어진 다단계 그래프 (multistage graph) G_i를 다음과 같이 만든다.

1) 0단계에는 s라고 불리는 정점이 있고, (m+1) 단계에는 t라고 불리는 정점이 있다.

2) 1부터 m단계에는 i행에 있는 m개의 모듈들이 대응되는데 정점들은 5-튜플 형태 $\langle i, j, f_p, f_c, f_n\rangle$를 갖는다. 여기서 i와 j는 모듈의 인덱스를 나타내고, f_c는 모듈 M_{ij}의 방향이고, f_p는 모듈 $M_{i,j-1}$의 방향 (j=1일 때는 f_p='-'), f_n은 모듈 M_{ij+1}의 방향을 의미한다 (j=m일 때는 f_n='-'). f_p, f_c와 f_n은 각각 4개의 모듈 방향 ϕ, v, h, b를 가질 수 있으므로 각 단계에는 최대 4^3=64개의 정점들이 존재한다. m=4일 때의 예가 그림 9에 나타나 있는데, 이 예에서는 지면 관계상 방향을 ϕ와 h로만 제한하였다.

3) G_i에서 에지는 다음과 같이 얻어진다.

(가) 정점 s에서 1단계에 있는 모든 정점으로 에지를 긋는데 이 에지들의 가중치는 모두 0이다.

(나) m단계에 있는 모든 정점 $\langle i, m, f_p, f_c, -\rangle$로부터 정점 t로 에지를 긋는데 이 에지들의 가중치는 $L_{im}''(f_c, f_c, -, F\text{-}\{f_{i,m-1}, f_{i,m}\})$이다

(다) 각 정점 $\langle i, j, f_p, f_c, f_n\rangle$에서 $\langle i, j+1, f_c, f_n, f'\rangle$로 에지를 긋는데 이 에지들의 가중치는 $L_{ij}''(f_c, f_p, f_n, F\text{-}\{f_p, f_c, f_n\})$이다. 여기서 $f'\in\{\phi, v, h, b\}$이다.

앞에서 언급한 과정을 통해 그래프 Gi를 구성하였을 때 s에서 t로 가는 경로 (path)에 있는 정점들에서 f_c 값들이 각 모듈들의 방향을 결정한다는 것을 쉽게 알 수 있다. 이 경로의 길이가 f_c들에 의해 정의된 모듈 방향이 주어졌을 때의 L_i'의 값과 같게 된다. 그러므로 s에서 t로의 최단 경로를 찾으면 모듈들의 최적 방향 X_i를 구할 수가 있다.

다단계 그래프에서의 최단 경로는 dynamic programming 기법을 사용하여 정점과 에지의 수에 선형 시간(linear time)으로 찾을 수가 있다[5]. 각 단계에 최대 64개의 정점이 있으므로 G_i에 있는 모든 정점의 수는 O(m)이고 각 단계 사이에 최대 256개의 에지가 존재하므로 에지의 수도 O(m)이 된다.

그러므로 G_i에서 X_i는 O(m) 시간에 찾을 수가 있고, 각 행에 대해 G_i를 구성하고 X_i를 찾아야 하므로 O(nm) 시간에 X를 구할 수 있다. 즉, 모듈의 수에 선형 시간으로 X를 구할 수 있다. 위의 분석은 각 모듈이 고정된 수의 선을 가지고 있을 때라고 가정한 경우이고, 그렇지 않은 경우는 시간 난이도(time complexity)가 O(mn+선의 수)이 된다.

V. 결 론

우리가 어떤 문제를 해결하려 할 때, 이 문제가 그래프 상에서 해를 구하는 문제로 변형 가능하면 이미 개발된 그래프 알고리즘을 사용하여 쉽게 문제를 풀 수가 있다. 본 논문에서는 레지스터 할당 문제, floorplanning 문제와 모듈의 방향 결정 문제에서 그래프를 사용해서 이러한 문제들을 표현하고, 이미 알려진 그래프 알고리즘을 적용하여 원하는 해를 구하는 방법을 고찰하였다.

참 고 문 헌

1. G. J. Chaitin, M. A. Auslander, A. K. Chandra, J. Coche, M. E. Hopkins, and P. Marktein, "Register Allocation Via Coloring", Comput. Lang. 6, pp. 47~51, 1981.
2. K. R. Chong and S. Sahni, "Algorithms for Floorplanning and Module Orientation", Ph.D. dissertation, University of Minnesota, 1991.'
3. Fred C. Chow and John L. Hennessy, "The Priority-based coloring approach to register allocation", ACM Transaction on Programming Language and Systems, 12(4), pp. 501~536, January 1990.
4. M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-completeness", W. H. Freeman and Co., New York, 1979.
5. E. Horowitz and S. Sahni, "Fundamentals of Computer Algorithms", Computer Science Press, Potomac, MD, 1978.
6. R. Libeskind-Hadas and C. L. Liu, "Solutions to the Module Orientation and Rotation Problems by Neural Computation Network", Proc. 26th Design Automation conference, pp. 400~405, 1989.
7. S. Wimer, I. Koren and I. Cederbaum, "Optimal aspect ratios of building blocks in VLSI", IEEE Trans. on Computer-Aided Design, Vol.18, No. 2, pp. 139~145, 1989.
8. M. Yamada and C. L. Liu, "An Analytical Method for Optimal Module Orientation", Proc. 1988 International Symp. on Circuits and Systems, pp. 1679~1682. 1988.

정 균 락

1978 서울대학교 계산통계학과(학사)
1980 한국과학기술원 전산학과(석사)
1991 Univ of Minnesota(박사)
1980 ~1984 한국과학원 연구원
1991 ~현재 홍익대학교 컴퓨터 공학과 조교수
관심 분야 : Algorithm
VLSI design algorithm
Parallel algorithm