최적 인터리빙을 이용한 배치 개선 알고리즘

성영태*, 오은경**, 허성우***
동아대학교 컴퓨터공학과
e-mail:{*saint, **ekoh}@donga.ac.kr, ***swhur@daunet.donga.ac.kr

An Algorithm for Improving Placement Using Optimal Interleaving

Young-Tae Sung*, Eun-Kyung Oh**, Sung-Woo Hur***
Dept. of Computer Engineering, Dong-A University

요 약

배치는 칩의 성능 및 레이아웃에 결정적인 영향을 주는 요인으로써 크게 광역배치와 상세배치 두 단계를 거쳐 이루어지며 수년 동안 다양한 기법들이 개발되어 왔다. 본 논문에서는 표준 셀 배치를 개선하기 위한 동적 프로그래밍 기법을 이용한 최적 인터리빙 알고리즘을 확장하여 셀을 동일 행 내에서만 움직이던 한계점을 극복하여 서로 다른 행 사이에서도 움직일 수 있도록 하였다. 즉, 셀들은 주어진 배치 내에서 임의의 위치로 움직일 수 있어 배치가 더욱 효율적으로 최적화 될 수 있도록 하였다.

1. 서론

VLSI/CAD분야의 물리적 설계 단계에서 수행되는 표준 셀 배치 문제는 NP-완전문제로 알려져 있으며[1], 이것을 해결하기 위하여 simulated annealing, force-directed technique, min-cut algorithm, numerical optimization, evolution-based placement 와 같은 다양한 알고리즘이 발표되었다[2]. 최근 배치 기법은 여러 가지 알고리즘을 혼합한 복합적 방식(hybrid technique)을 채택함으로써 향상된 배치 결과를 보여주고 있다[3,4]. 배치 알고리즘은 크게 광역배치와 상세배치 두 단계로 나눌 수 있으며, 최종 배치를 향상시키기 위해 최적의 광역 배치를 얻는 것이 필수적이다. S.W Hur와 John Lillis는 최소 절단 네트웍 플로우 기법(min-cut network flow)을 통해 최적의 광역배치를 얻은 후 동적 프로그래밍 기법(dynamic programming technique)을 이용하여 최적의 상세 배치를 얻었으며[3], Yand et.al은 n x n 격자(grid) 결정에 중점을 둔 분할기법을 이용하여 광역배치를 얻고, low temperature annealer를 사용하여 최적의 상세배치를 얻었다[4].

표준 셀 배치의 목적으로 회로 면적의 최소화, 배선 길이의 최소화, 라우팅 가능성의 최대화 그리고 신호 지연의 최소화등을 들수 있다. 이 중 배선 길이를 최소화 하는 것은 회로의 성능에 결정적인 영향을 주는 넷(critical net)을 줄일 수 있으며 회로의 전체적인 배선 가능성을 높일 수 있다[6].

본 논문에서는 행(row)과 열(column)의 2단계 최적 인터리빙을 이용한 개선된 상세배치 알고리즘을 제안하며 배선 길이의 최소화를 그 목적 함수로 선택하였다.

본 논문의 구성은 다음과 같다. 2절에서는 배치와 관련된 알고리즘을 설명하고 3절에서 실제 회로를 이용한 실험을 통하여 개선된 배치를 확인한 후 마지막으로 결론 및 향후 연구 과제를 제시한다.

2. 배치 알고리즘

2.1 개요

최적 인터리빙을 통한 배치개선 알고리즘은 행(row) 기반의 최적 인터리빙과 열(column) 기반의 최적 인터리빙을 번갈아 수행하며 각 셀의 최적 위치는 동적 프로그래밍 기법(dynamic programming)을 통해 계산된다. 전체 알고리즘의 구조는 그림 1과 같다.

Algorithm *Optimal Interleaving*
Input : P_d, w, v_{begin}, v_{end}
Output : an optimized detailed placement
P_d ← initial detailed placement
while(v_{begin} != v_{end}) {
  v_{begin} ← v_{begin} + next vertical position offset
  P_d ← columnInterleave(P_d, v_{begin})
  P_d ← optimize each rows by rowInterleave(P_d, w)
}
return P_d

그림 1. 행-열 최적 인터리빙 알고리즘

최적 인터리빙의 입력으로 초기 상세배치 P_d와 행(row) 인터리빙을 위한 윈도우 크기 w, 그리고 열(column) 인터리빙을 위한 초기 수직 좌표 값이 주어진다. 이때 $min_bound \le v \le max_bound$가 되며 최소, 최대 한계값은 표준 셀이 위치하는 핵심 영역(core region)의 한계값이 될 수 있다. 알고리즘은 열(column) 인터리빙을 위한 수직 좌표가 유효한 값인 동안 계속 수행된다. 먼저 초기 상세배치 P_d에 대한 열(column) 인터리빙이 수행된다. 열(column) 인터리빙이 끝나고 나면 각 행(row)에 대하여 행(row) 인터리빙이 수행되며 열(column) 인터리빙에서 발생하는 셀 중첩을 해결한다. 열(column) 인터리빙의 입력인 수직 좌표값 v가 더 이상 유효하지 않다면 알고리즘은 종료하고 최종 배선 길이를 계산한다.

2.2 최적 열 인터리빙(Optimal Column Interleaving)

그림 1에서 보인 바와 같이 최적 열(column) 인터리빙은 초기 상세배치 P_d와 가상의 수직선에 대한 좌표값 v을 입력으로 받으며 알고리즘은 v를 왼쪽에서 오른쪽으로 이동시키며 수행된다. 이것을 그림 2에 나타내었다.

Algorithm Column *Interleaving*
Input : P_d, v
Output : an optimized P_d
while(v != v_{end}) {
  T ← find a sequence of cells using vertical line vl
  A ← find a sequence of cells from T, s.t $|A|=n$,
  B ← find a sequence of cells from T, s.t
    $B \cap A=\emptyset$, $|B \cup A| = |T|$
  optimalInterleave(A, B)
  v ← move v to next vertical point
}

그림 2. 최적 열(column) 인터리빙 알고리즘

초기 부분열 T를 선택하는 과정은 다음과 같다. 각 행(row)에서 가상의 수직선 vl에 교차하는 모든 셀을 부분열 T에 포함시킨다. 임의의 열(row)에 대하여 수직선 vl에 교차하는 셀이 없다면 vl에서 가장 가까운 셀을 포함시킨다. 일단 T가 초기화 되었다면 두 부분열 A와 B로 다시 분할하고 동적 프로그래밍 기법(dynamic programming)을 이용하여 각 셀에 대한 열(column)상의 최적 위치를 찾는다. 이 과정에서 한 행(row)에 있는 셀을 다른 행(row)으로 옮기는 도중에 셀의 중첩이 발생할 수 있으나 이것은 최적 행(row) 인터리빙을 통해서 해결될 수 있다. 그림 3은 행의 갯수가 4인 경우 최적 열(column) 인터리빙의 과정을 보여주고 있다.

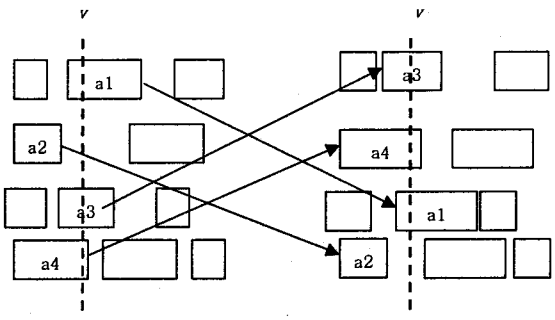

그림 3. 최적 열(column) 인터리빙의 동작 원리

최적 인터리빙에 사용되는 동적 프로그래밍 기법(dynamic programming)은 다음 단락에서 설명한다.

2.3 최적 행 인터리빙(Optimal Row Interleaving)

한 행(row)에 대한 최적 인터리빙 알고리즘은 그림 4와 같다.

Algorithm Row *Interleaving*
Input : R, w
Output : an optimized legal row
while(w != w_{end}) {
A ← find a sequence of cells within window size w
B ← find a sequence of cells within window size w, $B \cap A = \varnothing$, $B \cup A$=all cells within window
optimalInterleave(A, B)
w ← move window to right
}

그림 4. 최적 행(row) 인터리빙 알고리즘

최적 인터리빙의 입력으로 상세배치 P_d의 각 행과 윈도우 크기 w가 주어지며 다음과 같은 단계로 알고리즘이 수행된다. 먼저 주어진 윈도우 크기 내에서 n개의 부분열 A를 찾는다. 이때 부분열 A의 각 셀들은 원래의 상대적 순서를 유지한다. 다시 주어진 윈도우 크기 내에서 부분열 A에 포함되지 않은 셀들을 부분열 B로 선택하고 마찬가지로 원래 상대적 셀 위치를 유지한다. 그런 다음 선택된 부분열 A와 B에 대하여 각 셀의 상대적인 위치를 유지하면서 두 부분열을 최적 인터리빙 시키고 크기 w인 윈도우를 오른쪽으로 이동시킨다. 위 과정을 첫째 행(row)부터 마지막 행(row)까지, 그리고 각 행(row) 내에서는 윈도우를 오른쪽으로 계속 이동하면서 반복 적용시킨다.

윈도우 크기가 7인 경우 인터리빙 과정을 그림 5에 나타내었다.

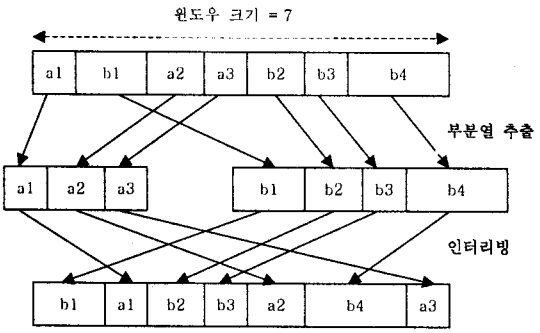

그림 5. 최적 행(row) 인터리빙의 동작 원리

열(column)과 행(row) 인터리빙에서 다루는 두 부분열의 최적 조합 과정은 동적 프로그래밍 기법을 통해 효율적으로 해결될 수 있다[7].

주어진 윈도우 크기 $n + m$에 대하여, $A=a_1, a_2, ..., a_n$ 그리고 $B=b_1, b_2, ..., b_m$라고 하자. $S_{i,j}$를 $a_1,a_2,...,a_i (i \le n)$와 $b_1,b_2,...,b_j (j \le m)$로 부터 얻은 최적 순서라고 하고, $C(S_{i,j})$는 $S_{i,j}$의 비용 즉, 부분열로부터 얻은 배선 길이의 총합이라 한다면 부분열 A와 B를 인터리빙 하는 것은 최소 비용의 $S_{n,m}$을 찾는 것과 같다. $S_{i,j}$의 부분적 배치 비용 $C(S_{i,j})$는 윈도우 사이즈 w내의 $S_{i,j}$에 속한 모든 셀들의 배선 길이의 합이다. 이것은 $S_{i,j}$로 커버된 부분 윈도우에 대한 배선 밀집도의 일반화된 합으로 볼 수 있다. 앞서 수행된 $S_{i,j}$의 최적 인터리빙은 이후에 일어나는 부분열에 대한 셀들의 순서를 결정하는데 독립적이라는 것이 이 아이디어의 핵심이며 이러한 독립성은 최적 인터리빙을 동적 프로그램으로 접근 가능하게 한다.

앞에서 설명한 동적 프로그래밍을 재귀 관계식(recurrence relation)으로 나타내면 식 1과 같이 된다.

$$S_{0,0} = 0$$
$$C(S_{0,0}) = 0$$
$$S_{i,j} = \begin{cases} S_{i-1,j}a_i \text{ if } C(S_{i-1,j}a_i) < C(S_{i,j-1}b_j) \\ S_{i,j-1}b_j, \textit{ otherwise} \end{cases}$$

식 1. 동적 프로그래밍의 재귀 관계

동적 프로그래밍 기법은 표(table)를 이용함으로써 이전까지의 국부 최적 $S_{i,j}$를 항상 저장하고 있으며 이러한 값들은 각 단계마다 독립적인 최소값이다. 최적의 $S_{n,m}$은 각 표의 최소 $S_{i,j}$의 조합으로 얻을 수 있다[7].

3. 실험 및 고찰

본 논문에서 제시한 알고리즘의 효율성을 검증하기 위해 I사의 실험용 회로를 사용하였다. 모든 회로는 표준 셀 배치를 목적으로 설계되었으며 광역배치는 다른 툴을 이용하여 얻었고, 이 광역배치의 결과를 입력으로 사용한다. 실험은 팬디엄4 2Ghz, 리눅스 머신상에서 수행되었으며 아래 각각의 결과는 각 회로에 대하여 행(row) 인터리빙을 수행했을 경우, 열(column) 인터리빙을 수행했을 경우 그리고 행(row)과 열(column) 인터리빙을 혼합하여 수행한 경우에 대하여 요약하였다. 표 1은 실험에 사용된 각 회로의 사양을 나타낸다.

표 1. 회로 사양

	#nodes	#cells	#pads	#nets	#rows
Ckt1	12649	7473	5176	7769	113
Ckt2	15275	11241	4034	11569	111
Ckt3	20680	16210	4470	13067	100
Ckt4	5423	3352	2071	3425	58
Ckt5	12195	7469	4726	7970	52

표 2에서는 최적 행(row) 인터리빙만을 적용했을 때의 결과를 보여주며 알고리즘 적용 전의 값은 광역 배치로 부터 얻은 결과이다.

표 2. 최적 행(row) 인터리빙을 적용한 결과

표2	알고리즘 적용 전	알고리즘 적용 후	개선율(%)
Ckt1	4.38736e+07	4.24217e+07	3.31
Ckt2	9.75889e+07	9.34727e+07	4.22
Ckt3	1.03904e+08	1.02754e+08	1.11
Ckt4	1088376e+07	1.80782e+07	4.03
Ckt5	6.23296e+07	5.97664e+07	4.11

표 3은 최적 열(column) 인터리빙만을 수행한 결과를 보여주며 알고리즘 적용 전의 결과는 표2와 동일하다.

표 3. 최적 열(column) 인터리빙을 적용한 결과

표3	알고리즘 적용 전	알고리즘 적용 후	개선율(%)
Ckt1	4.38736e+07	4.22289e+07	3.75
Ckt2	9.75889e+07	9.52991e+07	2.35
Ckt3	1.03904e+08	1.02289e+08	1.55
Ckt4	1.88376e+07	1.83533e+07	2.57
Ckt5	6.23296e+07	6.13191e+07	1.62

표 4에서는 행(row)와 열(column)을 혼합한 인터리빙의 결과이다.

표 4. 최적 행과 열 인터리빙을 같이 적용한 결과

표4	알고리즘 적용 전	알고리즘 적용 후	개선율(%)
Ckt1	4.38736e+07	4.08274e+07	6.94
Ckt2	9.75889e+07	9.06269e+07	7.13
Ckt3	1.03904e+08	9.66658e+08	6.97
Ckt4	1.88376e+07	1.75366e+07	6.91
Ckt5	6.23296e+07	5.86133e+07	5.96

4. 결론 및 향후 과제

본 논문에서는 상세배치를 개선하기 위해 최적 인터리빙(optimal interleaving) 알고리즘을 확장하여 적용함으로써 모든 셀이 배치 내에서 임의의 위치로 이동할 수 있도록 하였다. x-축 또는 y-축에 평행한 일차원적인 이동만을 통해서도 배치는 개선될 수 있으나 이를 동시에 적용할 경우 더욱 효과적으로 개선되는 것을 실험을 통해 보였다. 표 2와 3에서 보였듯이 최적 행(row) 인터리빙을 통해 평균 3.35%의 배치 개선을 얻었으며 열(column) 인터리빙을 통해 평균 2.37% 배치 개선을 얻었다. 그리고 표 4에서는 각각의 단일 인터리빙의 미비점을 극복한 결과 평균 6.77%의 향상된 배치를 확인할 수 있었다.

5. 참고문헌

[1] M.R Garey and D.S Johnson, *Conputers and Intractability: A Guide to the Theory of NP-Completeness*, W.H.Freeman and Company, 1979.

[2] K. Shahookar and P. Mazumder, *"VLSI Cell Placement Techniques,"* ACM Computing Surveys(CSUR), vol 23, Issue2, 1991.

[3] Sung-Woo Hur and John Lillis, *"Mongrel: Hybrid Techniques for Standard Cell Placement,"* Proc. of ICCAD, pp.165-170, 2000.

[4] X. Yang, M. Wang, K. Egur, and M. Sarrafzadeh, *"A Snap-on Placement Tool,"* Proc. of Intl. Symposium on Physical Design, pp. 153-158, 2000.

[5] Sung-Woo Hur and John Lillis, *"Relaxation and Clustering in a Local Search Framework: Application to Linear Placement,"* Proc. of the 36th ACM/IEEE conference on Design Automation, 1999.

[6] M. Wang and M. Sarrafzadeh, *"Behavior of Congestion Minimization During Placement,"* Proc. of International Symposium on Physical Design, pp. 145-150, 1999.

[7] Thomas H. Cormen and Charlse E. Leiserson, Introduction to Algorithms, The MIT Press, pp. 323, 2001.