論文99-36C-6-1

클러스터 재배치를 이용한 회로분할

(Clusters Re-placement for Circuit Partitioning)

金 相 鎭 *, 尹 泰 鎭 **, 李 唱 熙 ***, 安 光 善 *

(Sang Jin Kim, Tae Jin Yun, Chang Hee Lee, and Gwang Seon Ahn)

요 약

회로 분할 문제에 있어서 서열화는 k분 분할의 좋은 해를 얻는 방법으로 사용되어 왔다. 서열화는 구획화 과정을 거쳐 클러스터를 구성함으로서 분할이 완료된다. 본 논문에서는 이렇게 구성된 클러스터를 재배치하여 다시 구획함으로서 향상된 해를 얻는 알고리즘을 제시하였으며, 이를 여러 가지 대상회로에 대해 실험하여 향상된 결과를 얻었다.

Abstract

In circuit partitioning problem, work on vertex ordering have used to get good results for k-way partitioning. Body of work constructs a partitioning by first constructing a vertex ordering, then splitting it. We present a re-placement algorithm for enhanced results by replacing and splitting the clusters repeatedly. Experimental results on several circuits show that our approach achieves enhancement.

I. 서 론

회로 분할문제는 VLSI 칩이나 FPGA(Field Programmable Gate Array), MCM(Multi Components Module) 등의 설계자동화를 위하여 중요한 문제로 취급되어왔다.[5] 분할 될 회로의 수를 k라 할 때 이를 k분 분할이라 하며 k의 크기가 클 (k>10)때는 특히 클러스터링이라고도 한다. 시스템의 분할이나 레이아웃, 모의실험 등의 문제에 있어서 하이퍼그래프의 클러스터링은 VLSI CAD의 복잡성을 줄이기 위하여 즐겨 사용되고 있다. 예를 들어 20,000개의 노드를 갖는 논리회로를 k=5,000개의 클러스터로 분할하고 분할된 각 클러스터를 하나의 노드로 취급하면 문제의 크기는 1/4 줄어 자동화의 시간이 그만큼 단축되는 효과를 얻을 수 있다. 이러한 예로, 회로의 나누어진 각 클러스터를 노드로 보고 이분 분할 알고리즘인 FM(Fiduccia-Mattheyses)을 수행한 후, 클러스터링 된 노드를 풀어서 다시 한번 FM을 수행하여 분할 효율을 높이는 이 단계 FM 이분분할 알고리즘[6]이 연구되었으며, 비슷한 이 단계 접근법으로 회로 배치문제의 효율을 향상시키는 등 여러 가지 분야에서 응용되어왔다.

사용하려는 응용영역에 따라 각 클러스터를 구성하는데 드는 비용을 정의하고 이를 목적함수라 한다. 목적함수 값의 최적화는 이 함수의 성격에 따라 그 값을 최대화하거나 최소화하는 것을 일컫는다. 노드의 서열

* 正會員, 慶北大學校 컴퓨터工學科
(Department of Computer Engineering, Kyungpook National University)
** 正會員, 慶雲大學校 電算情報工學科
(Department of Computer & Information Engineering, Kyungwun University)
*** 正會員, 啓明文化大學校 事務自動化科
(Department of Office Automation, Kaimyung college)
接受日字:1998年11月19日, 수정완료일:1999年5月25日

화는 이러한 목적함수를 만족하기 위해서 많이 응용되어온 방법 중의 하나이다. 이는 주어진 그래프의 각 노드에 일련번호를 붙여 순서를 정하고 이웃한 노드들의 집합으로 클러스터를 형성하도록 하는 것으로 목적함수에 맞도록 적당히 서열을 정해야 한다. 서열화의 방법에는 전통적인 그래프 탐색 알고리즘인 깊이 우선 탐색(DFS), 너비 우선 탐색(BFS)과 최대 이웃 선택법(Max-Adjacency), 그리고 이를 개선한 WINDOW법 등이 연구되었다. 이렇게 서열화 된 노드는 목적함수에 맞도록 최적 구획화를 하는데, 이때는 동적 프로그래밍을 도입한 DP-RP가 사용된다.[3]

본 논문에서는 이렇게 구획화 된 클러스터를 재배치법을 이용하여 목적함수의 값을 개선하는 알고리즘을 소개하며 다음과 같이 구성되어있다. 다음 장에서는 이와 관련된 연구내용으로 목적함수, 서열화 그리고 구획화 알고리즘에 대해서 설명한다. 3장에서는 이전의 방법들에 대해 새로운 개념인 재배치법을 설명하고 이에 대한 예를 들겠으며, 4장에서는 이전의 분할 방법과 재배치법을 사용하여 이를 개선한 실험결과를 보인다. 마지막, 5장에서는 결론으로 논문을 맺는다.

Ⅱ. 관련 연구

하이퍼그래프 $H(V,E)$는 노드의 집합 $V=\{v_1, v_2, \ldots, v_n\}$와 에지의 집합 $E=\{e_1, e_2, \ldots, e_m\}$로 구성되어 있다. 하이퍼그래프 내의 모든 에지 e는 그에 인접한 노드들의 집합으로 나타내며 $e \subset V$, $e \in E$, $|e| \geq 2$ 를 만족한다. 분할된 각 클러스터 $C_i \neq \emptyset$는 $C_i \subset V$이다.

k분 분할은 각 클러스터의 집합 $P^k=\{C_1, C_2, \ldots, C_k\}$로 표현한다. 각 클러스터는 임의의 i, j에 대하여 $C_i \cap C_j = \emptyset, (i \neq j)$를 만족한다. 각 클러스터의 크기는 $L \leq |C_i| \leq U$, $(1 \leq i \leq k)$와 같이 한계치 내에 있어야 한다. 임의의 클러스터 C_i에 원소를 가지면서 다른 클러스터 $C_{\neq i}$에도 원소를 갖는 에지들의 집합 $cut(C_i)=\{e \in E \mid 0 < |e \cap C_i| < |e|\}$을 컷이라 한다.

정점 v에 이웃한 정점의 집합을 $adj(v)$라하고, v에 인접한 간선의 집합을 $net(v)$라 한다. 또, 클러스터 C에 이웃한 정점의 집합을 $adj(C)=\{v \in V - C \mid \exists v_j \in C,\ v \in adj(v_j)\}$라 정의한다.

1. 분할 목적함수

이러한 분할의 결과를 평가하기 위해 분할 비용을 정의할 필요가 있다. 분할에 관한 연구는 분할 비용 함수 $f(P^k)$의 값을 최적화하는 P^k를 구하는 것이 목적이다.[1] 이분 분할(k=2)의 경우, 전체 컷의 크기 $|cut(C_1)|$나, 균형비를 감안한 컷(ratio cut)의 크기 $\frac{|cut(C_1)|}{|C_1| \cdot |C_2|}$를 주로 분할 비용 함수 $f(P^2)$로 사용한다.

일반적인 클러스터링의 경우, 분할 문제의 비용 $f(P^k)$는 보통 각 클러스터 비용의 합으로 나타낸다. 예로써, 각 클러스터의 크기에 비한 클러스터 컷의 수의 합인 최소 크기비(SC : Scale Cost), 균형비를 감안한 전체 컷의 크기를 나타내는 최소 클러스터비(Minimum cluster ratio), 각 에지들이 단일 클러스터에 속한 비의 합인 흡수비(Absorption), 클러스터내 평균 최단경로에 비한 클러스터 디그리를 최대화하고자 하는 디그리비(Degree Separation), 한 클러스터 내에 완전히 속하게 되는 에지의 수를 최대화 하고자 하는 밀도(Density), 등이 이에 해당된다.

그밖에 특수한 응용 영역에 따라 최대 지연시간을 최소화하는 Min-Delay, 동일 종류의 FPGA에 적용할 수 있는 최소 소자의 수를 얻고자 하는 Single-Device FPGA, 다른 종류의 FPGA에 적용되는 Multi-Device FPGA등이 있다.

2. 노드의 서열화

노드의 서열화(Vertex Ordering)는 주어진 노드의 집합 $V=\{v_1, v_2, \ldots, v_n\}$의 각 노드에 서열을 매겨 $v_{\pi_1}, v_{\pi_2}, \ldots, v_{\pi_n}$으로 나타낼 수 있도록 하는 전단사함수 $\pi : [1 \ldots n] \rightarrow [1 \ldots n]$ 로서, $\pi(j)=i$ 에 의해 노드 v_i를 j 번째로 서열을 매기며, 편의상 $\pi(j)$를 π_j로 표기할 수도 있다. 이러한 서열화는 서열 내에서 연속된 부분들을 k개의 클러스터로 형성할 때 분할 함수 $f(P^k)$의 값을 최적화 할 수 있도록 모든 노드의 서열을 정하는 것을 목적으로 한다.

이 과정에서 서열화되지 않은 노드들은 서열화된 노드들로부터 끌리는 인력(attraction)을 갖게 되고 이 값이 최적인 노드가 다음 순으로 서열화된다. 서열화된 노드들의 집합을 S라 하고 현재 서열화되기 위해 선택된 노드를 v_{i*}라고 하며, 이때의 $|S|$을 $index$라 하고, 다음 서열화되는 노드에 서열을 붙이기 위해서 사용된다. 그림 1에 서열화 알고리즘을 표현하였다.

Choose a vertex v_{i*} and set $\pi(1)=i*$
Set *index*, the current size of S to 1
for each $v_i \in V-S$, compute $Attract(i)$
while $V-S \neq \emptyset$
  choose $v_{i*} \in V-S$ with optimal $Attract(i*)$
  Increment *index* and set $\pi(index)=i*$
  Update $Attract(i*)$ for each $v_i \in V-S$
end while

그림 1. 서열화 알고리즘
Fig. 1. Ordering algorithm.

인력을 계산하는 함수 $Attract(i)$에는 다음과 같은 것들이 있다. 인력함수로 $Attract(v)=\max\{j \mid v_{\pi_j} \in adj(v)\}$를 사용하는 깊이우선법(DFS)과 $Attract(v)=\min\{j \mid v_{\pi_j} \in adj(v)\}$를 사용하는 너비우선법(BFS)은 전통적인 그래프 방문 알고리즘을 이용하여 탐색하며 방문된 순서대로 서열화한 것이다. 최소 주변 노드법(min-perimeter)은 구성된 클러스터 C의 주변 노드의 수 $|adj(C)|$가 최소화 되도록 노드를 하나하나 포함해 나가며 서열을 매긴 것으로 인력함수 $Attract(v)=|adj(v)\cup\{v\}| - (S\cup adj(S))|-1$를 사용한다. 최대 이웃 선택법(max-adjacency)은 서열화된 노드들에 가장 많은 공통 에지를 가지는 노드를 선택해가며 서열화한 것으로 인력함수 $Attract(v)=|\{e \in net(v) \mid e\cap S\neq\emptyset\}|$를 사용한다.

서열화된 노드를 클러스터링할 때 필요한 클러스터의 크기 L,U를 감안하여 서열화되지 않은 노드들은 가장 최근에 서열화된 몇 노드 $\{v_{\pi i} \mid index-W < i \le index\}$로부터 전적인 인력을 받으며 또, 그 이전에 서열화된 몇 노드 $\{v_{\pi i} \mid index-W-T < i \le index-W\}$로부터는 *index*로부터의 거리에 비례하는 부분적인 인력을 받는다는 착안으로 각각 윈도우(window) W와 테일(tail) T를 설정하고 이에 대해서만 인력함수를 적용하는 윈도우법(WINDOW)이 연구되었으며 이는 대부분의 인력함수에 적용되어 좋은 성능을 보였다.[2]

3. 서열노드의 구획화

구획화(Splitting)는 서열화를 통해 얻어진 서열노드를 제한된 크기 L,U 범위 내에서 연속된 노드들의 집합으로 구성된 클러스터 $C_i=\{v_{\pi j} \mid m_{i-1}<j\le m_i\}$ $(m_0=0, m_k=n)$를 형성해 나갈 때, 분할 함수 $f(P^k)$의 값을 최적화 할 수 있는 m_i $(1\le i\le k)$를 구하는 것을 목적으로 한다. 이러한 제한된 분할 조건(RP : Restricted Partitioning)에서 최적화된 해를 구하기 위해 동적 프로그래밍(DP : Dynamic Programming)을 사용하는데 이것을 DP-RP라 한다.

구간 $[i,j]$내에 있는 노드들로 구성된 클러스터를 $C_{[i,j]}$라 한다. 또한 $P^k_{[i,j]}$를 구간 $[i,j]$내에서의 k분 분할이라고 하고, $\hat{P}^k_{[i,j]}$를 최적해를 갖는 $P^k_{[i,j]}$라고 한다. 그러면 $P^1_{[i,j]}=\hat{P}^1_{[i,j]}=\{C_{[i,j]}\}$로 나타낼 수 있다. $\hat{P}^k_{[i,j]}$는 $[i,j]\subset[i',j']$, $k<k'$인 조건에서 $\hat{P}^{k'}_{[i',j']}$를 구하기 위해 사용될 수 있다. 각 클러스터 $C_{[i,j]}$는 시작 노드 $v_{\pi i}$와 끝 노드 $v_{\pi j}$에 의해 결정되므로 RP문제는 주어진 제한 조건에 의하여 가능한 $(U-L+1)n$ 개의 클러스터 가운데 모든 노드가 중복되지 않고 포함되도록 k개의 클러스터를 선택함으로서 분할의 해를 얻는다. $(U-L+1)n$ 개의 각 클러스터는 자신이 $P^1_{[i,j]}$에 대한 최적해이다. DP-RP는 $P^1_{[i,j]}$로부터 이분 분할의 최적해 $\hat{P}^2_{[i,j]}$를 구하게되고 이것이 계속 확장되어 결국 $[1,n]$ 범위 내에서 k분 분할의 최적해 $\hat{P}^k_{[1,n]}$까지를 유도하게 된다.

$\forall\, i,j$ compute $f(P^1_{[i,j]})=c(C_{[i,j]})$
  using *Cluster_Costs*()
for $k'=2$ to k **do**
  for each i,j **do**
    $f_{best}=\infty$
    for $m=j-U$ to $j-L$ **do**
      if $f_{best} < f(\hat{P}^{k'-1}_{[i,m]}\cup\{C_{[m+1,j]}\})$ **then**
        $f_{best}=f(\hat{P}^{k'-1}_{[i,m]}\cup\{C_{[m+1,j]}\})$
        $\hat{P}^{k'}_{[i,j]}=\hat{P}^{k'-1}_{[i,j]}\cup\{C_{[m+1,j]}\}$
      end if
    end for
  end for
end for
return $\hat{P}^{k'}=\hat{P}^{k'-1}_{[i,i-1]}$ for the i that minimizes
  $f(\hat{P}^k_{[i,i-1]})$, $1\le i\le n$

그림 2. 구획화 알고리즘
Fig. 2. Splitting algorithm.

이 과정을 그림 2에 나타내었다. DP-RP 알고리즘

은 먼저 $(U-L+1)n$ 개 클러스터에 대해 클러스터비용 $c(C_{[i,j]})$를 구함으로 시작되며, 클러스터 비용은 분할 목적에 따라 다르게 계산되므로 여기서는 **Cluster_Costs**를 호출하는 것으로 되어있다. 이 알고리즘은 Cluster_Costs를 수행하는데 $O(nU)$, 분할을 위해 $O(k(U-L)n^2)$의 시간 복잡도를 가진다.[3]

Ⅲ. 재배치법

2.2절에서 설명된 여러 가지 서열화는 DP-RP에 의해 최적의 해를 갖는 클러스터들로 구획될 수 있다. 이때, DP-RP는 주어진 구획조건과 서열 $[1,n]$을 입력으로 받아 항상 최적의 해를 구해준다. 그러나 분할 문제의 최적해가 구해지지 않는 것은 최적의 서열화를 얻지 못했기 때문이다. 대부분의 서열화 알고리즘은 순간 순간 최적의 노드를 선택하여 서열을 정하는데, BFS와 같이 넓은 범위를 건너다니며 서열화하여 비슷한 시기에 서열화된 노드간의 연관성을 약하게 한다든지, DFS와 같이 깊이 방문하려는 경향으로 현재의 중요한 노드 덩어리를 지나쳐 버리게되는 경우가 있다. 이를 얼마나 조화롭게 처리하느냐가 관건이지만 이를 최적화하는 알고리즘은 개발되지 않았으며 아직까지 NP-문제로 알려져 있다.

```
M = ordering(V)
MS = splitting(M)
do
    M = replacing(MS)
    MS = splitting(M)
while f(P^k_M) updated
```

그림 3. 분할 알고리즘
Fig. 3. Partitioning algorithm.

이러한 서열화의 성질로 인하여 큰 인력을 가지는 노드가 서열상 멀리 떨어져 하나의 클러스터를 형성하지 못하게된다. 따라서, 서열화한 후 DP-RP에 의해 구획화된 분할을 클러스터단위로 연관이 깊은 클러스터끼리 이웃할 수 있도록 재배치하여 다시 노드단위에서 DP-RP를 수행하면 이전보다 나은 분할을 얻을 수 있다. 또 이와 같은 과정을 반복적으로 수행하여 계속적으로 분할 문제의 해를 향상시켜나갈 수 있다. 이것을 재배치(Re-placement)라하고 이를 분할과정에 적용한 전체적인 알고리즘은 그림 3에 보였으며, 여기에 사용된 서열 M과 서열의 집합 MS는 3.1절에 설명된다.

1. 재배치의 정당화

노드 집합 V를 서열화하여 생성된 새로운 서열인 $[\pi_1,\pi_n]$을 M이라 한다. 이때 서열 M을 최적으로 구획화하여 생성되는 k개의 부서열은 $[\pi_1,\pi_{m_1}]$, $[\pi_{m_1+1},\pi_{m_2}]$, ... , $[\pi_{m_{k-1}+1},\pi_{m_k}]$이며 이를 각각 M_1, M_2, ..., M_k로 정의한다. 서열 연결 연산자 &는 두 개의 서열을 하나로 연결한다. 즉, $M_1\&M_2\&\ldots\&M_k=M$이라 할 수 있으며 교환법칙이 성립하지 않는다. 또한 서열 해체함수 δ는 서열을 집합으로 바꾼다. 즉, 서열 M를 구획화하여 얻은 서열 M_i와 분할 P^k에 대해 $\delta(M)=V$, $\delta(M_i)=C_i$, $\hat{P}^k_M=\{\delta(M_i)\ |1\le i\le k\}$ 이다.

부서열의 재배치는 주어진 부서열의 집합 $MS=\{M_1,M_2,\ldots,M_k\}$의 각 부서열에 새로운 서열을 매겨 $M_{\rho_1},M_{\rho_2},\ldots M_{\rho_k}$로 나타낼 수 있도록 하는 전단사함수 $\rho:[1\ldots k]\rightarrow[1\ldots k]$ 로서, $\rho(j)=i$ 에 의해 부서열 M_i를 j 번째로 배치하며 편의상 $\rho(j)$를 ρ_j로 표기할 수도 있다. 이는 재배치로 인해 생성된 새로운 서열 $M_\rho=M_{\rho_1}\&M_{\rho_2}\&\ldots\&M_{\rho_k}$를 구획화하여 분할 비용을 줄이는 것을 목적으로 한다. 이때 목적 함수 $f(P^k)$에 대해 다음과 같은 정리를 세울 수 있다.

정리) 주어진 서열 M에 대해 목적 함수$f(P^k)$의 값을 최대(최소)화하는 구획화의 경우, $f(\hat{P}^k_M)\le f(\hat{P}^k_{M_\rho})$ ($f(\hat{P}^k_M)\ge f(\hat{P}^k_{M_\rho})$)가 성립한다.

증명) 분할 조건 L,U를 만족한 각 부서열은 재배치에 의해 M_ρ를 구성할 때, 이는 동일한 조건하에서 이전의 부서열의 집합 MS와 같게 최적 구획화 될 수 있으며, 이때는 $f(\hat{P}^k_M)=f(\hat{P}^k_{M_\rho})$가 성립한다. 이는 분할 비용함수 $f(\hat{P}^k_M)=\sum_{1\le i\le k}c(\delta(M_i))$에서 인덱스 i가 주어진 범위 내에서 어떤 값을 먼저 취하든지 결과가 같다는 것과 일맥상통한다. 이에 반해 $\hat{P}^k_M\neq\hat{P}^k_{M_\rho}$라면, $MS\neq MS_\rho$이며 재배치 후의 구획은 좀 더 나은 해를 얻게 된다.

2. 재배치를 통한 성능향상

함수 ρ는 $f(\hat{P}^k_{M_\rho})$의 값을 향상시킬 수 있도록 재배

치를 수행해야 한다. 서열화된 노드의 집합 S가 노드 v를 끄는 인력함수 $Attract(v)$에 대하여 2.2절에서 설명한바 있다. 분할의 목적에 따라 여러 가지 인력함수를 정의할 수 있었는데, 클러스터간의 인력함수 $Attract(C_i, C_j)$에서도 분할 목적에 따라 여러 가지 인력함수가 있을 수 있다. 이때, 재배치된 서열 M_ρ를 구하는 알고리즘을 그림 4에 나타내었다.

$M_{\rho_1} = M_i$, where minimize

$$\underset{1\le j\le k,\, j\neq i}{MAX}\ Attract(\delta(M_i), \delta(M_j))$$

$M_\rho = M_{\rho_1}$

$MS = MS - \{M_{\rho_1}\}$

for $m = 2$ **to** k **do**

  $M_{\rho_m} = M_i$, where maximize

  $Attract(\delta(M_i), \delta(M_{\rho_{m-1}})), M_i \in MS$

  $M_\rho = M_\rho$ & M_{ρ_m}

  $MS = MS - \{M_{\rho_m}\}$

end for

그림 4. 재배치 알고리즘
Fig. 4. Re-placement algorithm.

알고리즘은 먼저 자신 이외의 다른 모든 서열 M_j에 대한 최대 인력이 가장 작은 한 서열 M_i을 구하여 M_{ρ_1}으로 둔다. 이는 인력이 큰 서열끼리 이웃할 때 다른 분할을 얻을 확률이 비교적 높으므로 인력이 작은 서열을 선두에 두어 다른 분할의 가능성을 높이고자 함이다. 그 후 다른 서열 M_{ρ_m}을 배치할 때는 바로 전에 배치된 서열 $M_{\rho_{m-1}}$과 가장 인력이 큰 서열을 찾아 그 다음으로 배치하도록 하며, 이를 모든 서열이 재배치될 때까지 반복한다.

재배치 문제는 서열에 대한 서열화로 볼 수 있으며, 이에 대해서도 여러 가지 다른 방법을 사용할 수 있을 것이다. 본 논문에서 사용한 재배치법을 2.2절에서 소개한 서열화에 비유한다면 서열을 노드로 본 DFS법이라 할 수 있다. WINDOW법과 같은 향상된 방법을 응용하지 않은 것은 서열하나의 크기가 이미 윈도우의 크기에 준해서 구획되었기 때문에 윈도우내에 하나 이상의 서열이 포함되기 어려울 것이며 이 경우, WINDOW법은 DFS와 같은 알고리즘이 되기 때문이다.

재배치 알고리즘은 $k\times k$ 배열에 각 클러스터간의 인력을 테이블로 작성하여 두고 여기에서 가장 큰 값을 찾아내어 배치를 수행할 수 있으므로 각 에지를 기준으로 배열을 작성하는데 걸리는 시간 $O(e)\simeq O(n)$ 그리고, k개의 클러스터 배치에 대해 각각 테이블을 검색하는 시간이 필요하고 이것이 $O(k^2)$이므로, 전체 시간 복잡도는 $O(n+k^2)$이다.

3. 예제

본 절에서는 그림 5와 같이 n=8 개의 노드를 가진 하이퍼그래프를 $U=3, L=2, k=3$ 인 조건하에서 재배치 알고리즘을 사용하여 분할하는 예를 든다.

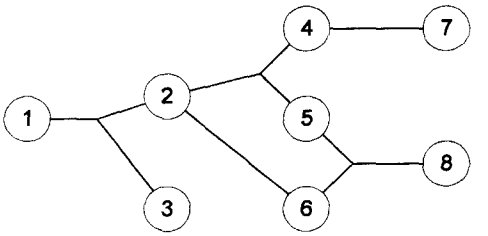

그림 5. 예제 회로
Fig. 5. Example circuit.

회로분할 문제의 대표적인 연구대상은 컷의 수를 줄이는 것이며, 각 클러스터의 크기도 균형을 이루는 것이 바람직하므로 본 논문의 예와 실험에서는 이를 잘 반영한 크기비(SC)를 목적함수로 사용하였다. 크기비의 경우, 각 클러스터의 크기에 대한 클러스터 컷의 크기를 일반화한 함수로 볼 수 있으며,

$$f(P^k) = \frac{1}{n(k-1)} \sum_{i=1}^{k} \frac{|cut(C_i)|}{|C_i|}$$

를 최소화하려는 목적을 가진다. 예제의 이해를 돕기 위하여 간단한 DFS를 서열화를 위한 인력함수로 두고, 재배치를 위한 클러스터간 인력함수로는 크기비를 고려하여 다음과 같이 정의한다.

$$Attract(C_i, C_j) = \sum_{\{e\in E \mid e\cap C_i\neq\emptyset,\, e\cap C_j\neq\emptyset\}} \frac{|(C_i\cup C_j)\cap e|}{|e|}$$

분할 과정은 먼저 v_1으로부터 DFS방법으로 방문하여 [1,2,4,7,5,8,6,3]으로 서열화 하고, DP-RP는 이를 주어진 조건에 따라 가능한 세가지 방법으로 구획화를 시도하여 표 1의 Pass1과 같은 결과를 얻는다. 이때 세가지 방법의 크기비는 주어진 조건에 따라 계산된다. 예를 들어 1번 분할의 경우, $SC=$

$\frac{1}{8(3-1)}\frac{3}{2}+\frac{2}{3}+\frac{3}{3}=0.198$ 이다. 이 중 $SC=0.198$로 가장 작은 1번을 택하였다고 하자. 이는 $C_1=\{v_1,v_2\}$, $C_2=\{v_4,v_7,v_5\}$, $C_3=\{v_8,v_6,v_3\}$ 로 하이퍼그래프를 분할한 것으로 클러스터의 컷 $cut(C_i)$을 컴마 뒤에 각각 표시하였다. 이중 클러스터 비용이 가장 작은 C_2를 가장 먼저 선택하여 맨 앞에 두고, C_2와 인력이 큰 C_3를 그 다음으로, C_1을 마지막으로 두면 Pass2와 같이 배열된다. 이를 또다시 DP-RP를 사용하여 분할하면 $C_1=\{v_4,v_7\}$, $C_2=\{v_5,v_8,v_6\}$, $C_3=\{v_3,v_1,v_2\}$의 결과를 얻게된다. 이것은 Pass3와 같이 동일한 기준으로 재배치 한 후 DP-RP를 수행하여도 분할 비용이 줄지 않으므로 알고리즘을 종료한다.

표 1. 예제회로의 분할 과정
Table 1. Partitiionig example circuit.

Pass	Case	C_1	C_2	C_3	SC
1	1	$\{v_1,v_2\}$, 3	$\{v_4,v_7,v_5\}$, 2	$\{v_8,v_6,v_3\}$, 3	0.198
	2	$\{v_1,v_2,v_4\}$, 4	$\{v_7,v_5\}$, 3	$\{v_8,v_6,v_3\}$, 3	0.240
	3	$\{v_1,v_2,v_4\}$, 4	$\{v_7,v_5,v_8\}$, 3	$\{v_6,v_3\}$, 3	0.240
2	1	$\{v_4,v_7\}$, 1	$\{v_5,v_8,v_6\}$, 2	$\{v_3,v_1,v_2\}$, 2	0.115
	2	$\{v_4,v_7,v_5\}$, 2	$\{v_8,v_6\}$, 2	$\{v_3,v_1,v_2\}$, 2	0.146
	3	$\{v_4,v_7,v_5\}$, 2	$\{v_1,v_2,v_5\}$, 3	$\{v_1,v_2\}$, 3	0.198
3	1	$\{v_4,v_7\}$, 1	$\{v_3,v_1,v_2\}$, 2	$\{v_5,v_8,v_6\}$, 2	0.115
	2	$\{v_4,v_7,v_3\}$, 2	$\{v_1,v_2\}$, 3	$\{v_5,v_8,v_6\}$, 2	0.177
	3	$\{v_4,v_7,v_3\}$, 2	$\{v_1,v_2,v_5\}$, 4	$\{v_8,v_6\}$, 2	0.188

Ⅳ. 실험과 고찰

본 실험에서는 [2] 에서 사용한 8개의 벤치마크 회로(http://vlsicad.cs.ucla.edu/~cheese/)에 대해서 재배치 알고리즘을 실험하였다. 크기비를 목적함수로 사용하고, DFS, BFS, Max-Adj, WINDOW에 대하여 원래의 방법과 각 알고리즘에 재배치법을 적용하여 얻은 결과를 표 2에 나타내었다. 이 실험에서는 $L=10$, $U=100$을 사용하였고 분할 수는 노드 수에 따라 다른 값을 적용하였다. 그리고 WINDOW 방법에서의 윈도우 크기는 $W=\lceil \frac{n}{k} \rceil$, 테일의 크기는 $T=U-W$, 인력함수는 $Attract(v)=\sum_{e\in M(v)}\frac{|S\cap e|}{|e|-1}$로 실험되었다. 또, 재배치법에 사용된 클러스터 인력함수는 3.3절에서 예제와 함께 정의된 것을 그대로 사용하였다.

실험에 사용한 시스템은 PC이며 Visual C++로 구현하였고 9개의 다른 벤치마크 회로[1]에 대하여 실험하였다. 표에는 회로명, 각회로의 노드수 n, 분할 수 k를 표시하였으며 점선 위에 각 알고리즘을 사용한 결과를 그 아래에 재배치법을 이용한 결과를 나타내었다.

이 실험 결과를 통해서 볼 때, 재배치법은 DFS, BFS, Max-Adj, WINDOW법 대하여 각각 90.319%, 91.157%, 93.878%, 94.997%, 평균 92.588%로 크기비를 줄인 결과를 보였다. 재배치의 회수는 WINDOW의 경우 각 회로에 대해 5회, 50회, 22회, 16회, 8회, 12회, 27회, 20회, 4회 반복하였으며 회로의 크기에 대략 비례하는 것으로 나타났으나 Max-Adj의 경우 Pr1-SC, Pr1-SC에 대해 27회, 8회의 반복회수를 보여 실험에 따라 다양한 차이를 보여주었으며 평균 9.8배의 CPU 시간이 걸린 것으로 나타났다.

또한, 분할 수 k가 작은 경우($k<10$)의 실험에서는 좋은 향상을 보여주지 못하였다. 예로 대표적인 이분 분할 알고리즘인 FM에 재배치를 적용한 결과 2% 미만의 향상만을 나타내었다.

Ⅴ. 결 론

본 논문에서는 서열화를 이용하는 알고리즘의 결과를 향상시키는 재배치법을 소개하였다. 이것은 본 논문에서 실험한 크기비 이외의 다른 목적함수를 위해서도 다양하게 적용될 수 있는 범용성을 가지고 있으며, 서열화 및 DP-RP를 적용할 수 있는 기존의 어떤 방법과도 같이 사용될 수 있다.

실험을 통해서 볼 때, 이전의 결과를 향상시키는 것을 관찰할 수 있었으며, 최적해에 가까운 알고리즘일수록 향상율이 줄어드는 것도 관찰되었다.

앞으로의 연구 방향은 좀 더 나은 위치에 클러스터를 배치할 수 있도록 알고리즘을 개발하는 것이라 하겠다. 이 문제 역시 분할 수 k에 대한 NP문제로 보여진다. 이에 대한 연구는 여러 종류의 서열화 알고리즘이 개발된 것에 비견될 수 있을 것이다. 또한 재배치의 반복 수 개선을 통한 전체 수행 속도의 향상도 기대해 볼 만하다. 더불어 배치된 클러스터 내부의 노드를 재배열하는 방법도 좋은 결과를 보여줄 것으로 기대된다.

표 2. 실험결과
Table 2. Experimental results.

Case	n	k	DFS		BFS		Max-Adj		WINDOW	
			SC	time	SC	time	SC	time	SC	time
Prl-SC	833	19	108.512	7.8	206.938	7.8	87.418	7.7	68.266	7.2
			94.733	23.4	197.977	23.4	76.523	18.5	67.786	21.6
Pr2-SC	3014	70	44.209	23.2	62.677	23.2	32.290	23.2	26.714	22.8
			42.598	487.2	57.379	320.2	31.309	348.0	25.606	684.0
Test02	1663	44	73.422	12.8	121.603	12.8	53.648	12.9	42.411	13.1
			62.317	138.2	107.221	153.6	50.998	123.8	38.958	172.9
Test03	1607	32	71.589	12.7	121.632	12.8	42.196	12.8	34.849	12.3
			65.477	76.2	110.815	53.8	39.474	92.2	33.090	118.1
Test04	1515	31	74.994	12.3	143.755	12.3	43.798	12.4	35.559	12.1
			65.594	59.0	131.838	73.8	41.134	67.0	33.888	58.1
Test05	2595	42	43.899	20.1	81.784	20.1	28.818	20.5	19.770	19.8
			40.909	84.4	75.579	132.7	26.382	123.0	18.916	142.6
Test06	1752	47	70.781	14.2	118.751	14.1	54.275	14.5	44.029	14.0
			61.972	136.3	107.109	126.9	52.745	174.0	41.759	226.8
19ks	2844	73	31.679	22.8	56.802	22.7	27.093	23.1	22.870	22.3
			30.086	205.2	51.732	190.7	25.972	221.8	22.286	267.6
bm1	882	21	84.089	8.1	167.857	8.1	77.644	8.2	60.762	7.8
			75.425	24.3	148.602	29.2	72.434	24.6	58.184	18.7

참 고 문 헌

[1] C. J. Alpert., "Multi-way Graph and Hypergraph Partitioning" Ph.D thesis, Dep't of Computer Science. UCLA. 1996.

[2] C. J. Alpert and A. B. Kahng. "A General Framework for Vertex Orderings, With Application to Netlist Clustering." Technical Report 940018, UCLA, 1994.

[3] C. J. Alpert and A. B. Kahng. "Multi-Way Partitioning Via Space-filling Curves and Dynamic Programming." In Proc. of the ACM/IEEE Design Automation Conf. pp. 652-657, 1994.

[4] S. Dutt and W. Deng. "A probability-based approach to VLSI circuit partitioning." In Proc. of the ACM/IEEE Design Automation Conf. pp. 100-105. 1996.

[5] F. M. Johannes. "Partitioning of VLSI Circuits and Systems." In Proc. of the ACM/IEEE Design Automation Conf. pp. 775-778, 1996.

[6] T. N. Bui, "Improving the Performance of the Kernighan-Lin and Simulated Annealing Graph Bisection Algorithms", In Proc. of the ACM/IEEE Design Automation Conf. pp. 775-778, 1989.

[7] W. Sun and C. Sechen, "Efficient and Effective Placements for Very Large Circuits" Proc. of the IEEE Intl. Conf. on Computer-Aided Design, pp. 170-177. Santa Clara, Nov. 1993.

저 자 소 개

金 相 鎭(正會員)
1994년 계명대학교 전자계산학과 졸업(공학사). 1996년 경북대학교 대학원 컴퓨터공학과 졸업(공학석사). 1998년 경북대학교 대학원 컴퓨터공학과 박사과정 수료. 주관심 분야는 VLSI 설계, DFT 기법, 회로분할

尹 泰 鎭(正會員) 第 35卷 C編 第 2號 參照

李 唱 熙(正會員) 第 35卷 C編 第 2號 參照

安 光 善(正會員) 第 35卷 C編 第 2號 參照