設 計 自 動 化 70－3
（1994．2．4）

グラフを2分割するハードウェアアルゴリズム

磯本 和典† 若林 真一‡ 小出 哲士‡ 吉田 典可‡
† マツダ株式会社技術研究所
〒730-91 広島県安芸郡府中町新地3-1
‡ 広島大学工学部
〒724 広島県東広島市鏡山一丁目4番1号

グラフ2分割問題は無向グラフG=(V,E)の節点集合を，2つの集合間を接続する枝の数が最小となるように2分割する問題である．この問題はVLSI設計では非常に重要な問題である．本稿では，グラフ2分割問題に対するハードウェアアルゴリズムを提案する．提案アルゴリズムはグラフ2分割問題の代表的なヒューリスティックアルゴリズムであるKernighan-Lin法を基にしており，$O(|V|)$個のプロセッシング素子から構成される1次元アレイ上で動作し，VLSIチップによる実現に適している．提案アルゴリズムの1回のパスの時間計算量は$O(|V|)$である．シミュレーション実験により，その解がKernighan-Lin法と同程度であることを示す．

A Hardware Algorithm for Graph Bisection

Kazunori ISOMOTO† Shin'ichi WAKABAYASHI‡ Tetsushi KOIDE‡ Noriyoshi YOSHIDA‡
† Technical Research Center, MAZDA Co.
3-1, Shinchi, Fuchu-cho, Aki-gun,
Hiroshima 730-91 Japan
‡ Faculty of Engineering, Hiroshima University
4-1, Kagamiyama 1 chome, Higashi-Hiroshima,
Hiroshima 724 Japan

The graph bisection problem is to partition the vertex set of an undirected graph G=(V,E) into two sets of equal size such that the number of edges between them is minimized. This problem is particularly important in several aspects of VLSI design. In this paper, we propose a hardware algorithm for graph bisection. The algorithm is based on the Kernighan-Lin heuristic algorithm, runs on a linear array consisting of $O(|V|)$ processing units, and is very suitable for direct VLSI implementation. Computation time of one pass of the proposed algorithm is $O(|V|)$. Simulation experiments showed that the proposed algorithm is as good as the original KL heuristic.

1 まえがき

回路分割はVLSI設計のあらゆる段階で必要とされる最も基本的な問題の1つである．回路分割は一般に，与えられたグラフGを等しいサイズの2つの部分グラフに分割するというグラフ分割問題として定式化できる．この問題はNP困難[5]のクラスに属することが示されており，いくつかのヒューリスティックアルゴリズムが提案されている[10]．それらの中で，KernighanとLinによって提案された手法（以下，KL法）が最も知られており，実用面でも多く使われている[6]．KL法の時間計算量はグラフの節点数をnとして$O(n^2 \log n)$である．

一方，最近の半導体技術の発達により，回路分割で扱う問題のサイズも大きくなってきた．このため，KL法は時間計算量の点から実用的でなくなり，より効率のよい分割手法の開発が望まれてきた．一般に，VLSI回路の分割にかかる時間を短縮するためには2つの手法が考えられる．まず第1に，従来の逐次計算機上で動作する新しいヒューリスティックアルゴリズムの開発である．例えば，FiducciaとMattheysesは回路分割を行う線形時間アルゴリズム（以下，FM法）を提案した[4]．しかし，FM法は時間計算量は小さいがKL法と同程度の解を求めることができない．回路分割の時間を短縮する第2の方法は並列処理の導入である．例えば，Banerjeeらはセル配置に対して，ハイパーキューブマルチプロセッサ上で動作する並列シミュレーティドアニーリングアルゴリズムを提案した[2]．また最近のVLSI技術の発達により，比較的低コストで並列アルゴリズムを直接VLSIチップとして実現することも可能になってきた．しかし，これまでに知られている多くの並列アルゴリズムは粒度が粗く，VLSIでの実現には向いていない．そこで，良質の解を求め，VLSIでの実現に適した回路分割並列アルゴリズムを開発することができればVLSIチップの設計時間とコストの短縮が可能となる．

本稿では，グラフを分割するハードウェアアルゴリズムを提案する．提案するアルゴリズムはn／2個のプロセッシング素子から構成される1次元アレイ上で動作し，$O(n)$計算時間でグラフを分割する．提案するアルゴリズムはKL法とオッドイーブントランスポジションソート(odd even transposition sort，以下，OETソート)[1]を基にしている．OETソートアルゴリズムはn個のデータをn個のプロセッシング素子からなる1次元アレイ上でnステップでソートする並列アルゴリズムである[1]．提案アルゴリズムをVLSIで実現することにより，1万個の節点をもつグラフを数秒で分割できることが期待できる．

以下，2節では，グラフ分割問題を定式化し，KL法について説明する．3節では，グラフを分割するハードウェアアルゴリズムについて説明する．4節ではシミュレーション実験によって，提案アルゴリズムがKL法と同程度の解を求めることを示す．

2 準備

2.1 問題の定式化

本稿では，枝に重みをもつ無向グラフGが与えられ，カットされる枝の重みの総和が最小となるようなGの節点の分割を求めるグラフ2分割問題を扱う．G=(V,E)を無向グラフとし，Vを節点の集合，Eを枝の集合とする．Vの大きさをnで表わし，Eの大きさをmで表わす．枝についてはコスト関数c:E→Nが与えられる（Nは自然数の集合）．

GのカットとはVの互いに素であり空でない集合V_LとV_Rへの分割である．枝eの一方の節点がV_Lに属し，他方の節点がV_Rに属するとき，枝eはカットをクロスするという．カットのサイズとはカットをクロスする枝のコストの総和をいう．

Gの2分割は$||V_L|-|V_R|| \leq 1$を満たすカットである．Gの最小2分割は最小サイズのGの2分割である．

グラフ2分割問題とはグラフの最小2分割を求める問題である．ここで，グラフGの節点数が奇数のときは接続する枝を持たない節点を導入することにより，最小カットのサイズを変更することなく節点数を偶数とすることができる．これから，以下ではグラフの節点数は偶数であると仮定する．

2.2 Kernighan-Lin法

2.2.1 KL法の概要

グラフ2分割問題はNP困難のクラスに属する[5]．このため，多くのヒューリスティックアルゴリズムが開発されている[4],[6],[7]が，それらの中で文献[6]のKL法がグラフ2分割アルゴリズムの中で基本的なものとして知られており，比較的良い解を求めることから，VLSI設計におけるセル配置設計等で広く使われている[3]．

KL法は繰り返し改良法であり，ある初期解から開始し，以下のような解の改善を繰り返す．現在の解をC_0とする．ここで，交換することによってカットサイズの減少が最大となるような節点ペアを選ぶ．カットサイズが減少しないときは，増加が最小となる節点を選ぶ．ここで，このペアの選択は暫定的なものである．1つの節点が移動することが決まればこの節点はロックされる．次はロックされた節点集合を除いた節点集合から，交換することによってカットサイズの減少が最大となるような節点ペアを見つけ，これをロックする．このような操作を続けて行い，選択された節点ペアとカットサイズの記録を残しておく．この節点選択

```
1: compute the c and D values;
2: BestCost := Cost[0] := CutSize(L,R);
3: BestChange := 0;
4: for s:= 1 to n/2 do begin
5:    ndata := n/2 - s + 1;
6:    QuickSort(L);  /* D(l1)≧D(l2)≧ … ≧D(lndata) */
7:    QuickSort(R);  /* D(r1)≧D(r2)≧ … ≧D(rndata) */
8:    Cost[s] := ∞;
9:    i := 0;
10:   repeat
11:      i := i + 1;
12:      j := 0;
13:      repeat
14:         j := j + 1;
15:         LRGain := -(D[li] + D[rj]);
16:         ExchangeCost := 2c(li,rj) + LRGain;
17:         if ExchangeCost < Cost[s] then begin
18:            Pair[s] := (li,rj);
19:            Cost[s] := ExchangeCost
20:         end
21:      until (j=ndata) or (Cost[s]≦LRGain)
22:   until (i=ndata)
23:   (lmin, rmin) := Pair[s];
24:   L := L - {vlmin};
25:   R := R - {vrmin};
26:   for each i such that vi∈(L ∪ R) do
27:      if vi ∈ L then
28:         D[i] := D[i] - 2c(i, rmin) + 2c(i,lmin)
29:      else
30:         D[i] := D[i] - 2c(i, lmin) + 2c(i,rmin);
31:   Cost[s] := Cost[s-1] + Cost[s];
32:   if Cost[s] < BestCost then begin
33:      BestChange := s;
34:      BestCost := Cost[s]
35:   end
36: end;
37: for s:= 1 to BestChage do
38:   exchange Pair[s];
```

図1　Kernighan-Lin法

は全ての節点がロックされるまで続けられる．最終的に，カットサイズの減少が最大となるような節点ペアの系列を求め，それらを実際に交換する．ここまでの操作をパスと呼ぶ．ここまでの結果C_1が次のパスでさらに改善されうるならば，次のパスにおいて，ここまでの結果C_1を初期解として改善を行う．何度かパスを繰り返した後，改善ができなくなればアルゴリズムは終了する．図1にKernighan-Linアルゴリズムの詳細を示す．このアルゴリズムにおいて，ゲインDがコスト関数であり，各節点について計算される．(L,R)を初期分割とし，c(e)を枝eのコストとする．節点v_i∈Lのゲインを以下に示す．

［定義1］節点v_i∈Lの外部枝コストは

$$E(i) := \sum_{e=(v_i,v_j) \in E \wedge v_j \in R} c(e)$$

と定義され，内部枝コストは

$$I(i) := \sum_{e=(v_i,v_j) \in E \wedge v_j \in L} c(e)$$

と定義される．Rについても同様に定義され，節点v_iのゲインを以下のように定義する．

$$D(i) := E(i) - I(i)$$ □

D(i)はv_iが分割において属する部分集合を変更したときのカットサイズの減少を示す．これから，節点ペア$\{v_i,v_j\}$の交換に伴うカットサイズの変化分は，それらの間に枝(v_i,v_j)があれば枝のコストをc(i,j)，枝がなければc(i,j)=0とするとD(i)+D(j)-2c(i,j)となる．アルゴリズムにおいて，その時の最良のカットサイズをもつ交換ステップはBestChangeに記憶され，その時のコストはBestCostに記憶される．一時的な移動で選ばれた節点ペアと対応するカットサイズはPairとCostに記録される．

KL法の計算複雑度は$O(n^3)$である．しかし，平均的にはKL法は$O(n^2 \log n)$時間で解を求める[10]．

2.2.2　KL法の性質

図1に示したKL法の第9から23行までで求める節点ペア(lmin,rmin)はそれぞれソートされた節点リストLとR内において均一に分散しているわけでない．これらの節点ペアとして選ばれた節点の多くはゲインが高く，リスト内でも上位の方にあると考えられる．そこで，KL法を用いて実際のVLSI回路データに基づくグラフを分割し，このときの節点リストLとR内における節点lminとrminの出現位置を確認した．この実験結果を表1に示す．実験データは4節でシミュレーション実験を行うデータ（表3中のPrimSC1）を用いた．表1に示すように，節点lminとrminはそれぞれ節点リストLとRの1番目から非常に大きな確率で選ばれていることがわかる．この性質に基づいて3節でKL法に基づくハードウェアアルゴリズムを提案する．

3　アルゴリズム

3.1　シストリック・アルゴリズム

3.1.1　KL法の並列化

この節では，グラフを2分割するハードウェアアルゴリズムについて説明する．アルゴリズムは前節で説明したKL法の拡張である．KL法はP完全のクラスに属している[12]ため，KL法の直接的な並列化は困難であ

表1　リスト内の出現位置

リスト内の順番	L側の確率 (%)	R側の確率 (%)
1番	99.80	99.80
2番	0.17	0.17
3番	0.03	0.03
4番	0.0	0.0
5番以降	0.0	0.0

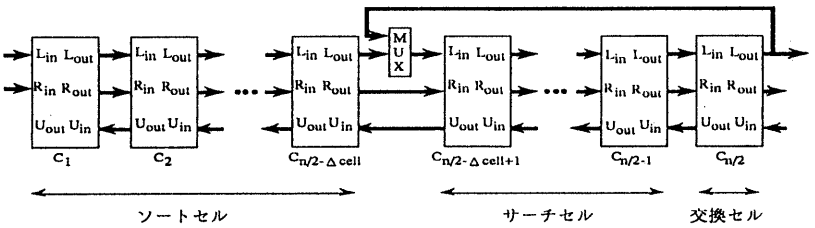

図2　1次元アレイの構造

ることが予想される．そこで，本稿ではKL法を解の質を保ったまま，アルゴリズムを一部変更することにより並列化することを目的とした．

並列化における問題はカットサイズを大きく改善する節点のペアをどのようにして見つけるかであり，節点のペアを見つけたあとにゲインをどのようにして更新するかである．さらに，元のアルゴリズムでは，1つの節点ペアを平均して$O(n \log n)$で求めている．しかし，提案する並列アルゴリズムでは全体の計算複雑度を$O(n)$としたいので，このステップは$O(1)$で行う必要がある．

これらの問題点を解決するために，提案アルゴリズムでは節点のゲインによるソートに並列ソートアルゴリズムを適用した．提案するアルゴリズムでは，オッドイーブントランスポジションソート(odd-even transposition)[1]を適用した．第2に，2.2.2で説明したように節点ペアとして選ばれた節点の多くはリスト内でも比較的上位の方にあることが多い点に注目した．そこで，元のKL法のようにリスト全体をソートし直してゲインが最大の節点ペアを見つけるのではなく，高いゲインをもつある定数個の節点集合のみから見つけることにした．第3に，節点ペアを選ぶときに，他の節点が部分集合を移ったためにゲインを更新しなければならない節点の数は少ないことに注目した．実際に，グラフ分割の多くのアプリケーションでは，節点の最大次数は小さな定数となることが一般的である．さらに，選ばれた節点ペアの交換コストは正確に計算されるべきであるが，それ以外の節点については常に正確に計算する必要はない．すなわち，例え節点のゲインが正確でないままで分割を行っても，最終結果は大きな差はないと思われる．

3.1.2　セルとデータ構造

グラフを2分割するアルゴリズムについて説明する．アルゴリズムはシストリック・アルゴリズム[8],[9]であり，セルと呼ばれるプロセッシング素子から構成される1次元アレイ上で実行される．アレイの構成を図2に示す．アレイはソートセル，サーチセル，交換セルの3種類のセルからなっている．グラフの節点数をnとして全部でn／2個のセルからなっており，内訳はn／2-Δcell個のソートセルとΔcell-1個のサーチセルと1個の交換セルである．ただし，Δcellは定数である．説明を簡単にするため，アレイは図2のように水平方向に並んでいるものとし，右端のセルを交換セルとし，左端のセルから$C_1, C_2, \cdots, C_{n/2}$という順番に番号がついているものとする．ソートセルが基本的なセルでサーチセルと交換セルはソートセルの拡張である．

アルゴリズムの中で用いるデータ構造について説明する．図3に3種類のセルで共通に使用するリストデータ構造を示す．リストは固定長とし，(1)節点ID，(2)節点次数，(3)接続枝のリストからなる．節点IDは節点IDを表わす整数である．節点次数はその節点に接続している節点の個数を表わす．接続枝のリストは3項組(v,c,side)のリストである．vは隣接する節点，cはvとの枝のコスト，sideはvがL側に属すると"L"，R側に属すると"R"という値をもつサイドフラグである．以降ではこのリストのことを節点データと呼ぶ．

アルゴリズムの実行中，各セルは2つの節点データVLとVRをもつ．VLは一時的にLに属している節点の

```
type
   VertexID:        integer;
   EdgeData = record
      AdjVertex:    VertexID;
      EdgeCost:     integer;
      Side:         char
   end;
   VertexData = record
      Vertex:       VertexID;
      VertexDegree: integer;
      EdgeList:     array[1..MaxDegree] of EdgeData
   end;
var
   VL:              VertexData;
   VR:              VertexData;
   LD:              integer;
   RD:              integer;
   LNew:            integer;
   RNew:            integer;
```

図3　セルのデータ構造

```
1:initialize;
2:input data;
3:for i:=1 to n/4 - Δsort/2 do begin
4:   for all cells do in parallel OddTransposition;
5:   for all cells do in parallel EvenTransposition;
6:end;
7:for k:=1 to n/2 do begin
8:   for s:=1 to Δsort/2 do begin
9:      for all cells do in parallel OddTransposition;
10:     for all cells do in parallel MoveNewData;
11:     for all cells do in parallel EvenTransposition;
12:     for all cells do in parallel MoveNewData
13:  end;
14:  do in parallel begin
15:     for search & exchage cells do in parallel
                                        FindBestSwap;
16:     for s:=1 to Δcell do begin
17:        for sort cells do in parallel OddTransposition;
18:        for sort cells do in parallel MoveNewData;
19:        for sort cells do in parallel EvenTransposition;
20:        for sort cells do in parallel MoveNewData
21:     end
22:  end;
23:  for all cells do in parallel MoveData
24:end;
```

図4　シストリックアルゴリズム

データであり，VRは一時的にRに属している節点のデータである．L側とR側の節点のゲインはそれぞれLDとRDがもつ．さらに，ゲインを更新するために一時的な交換候補として選ばれた節点IDはそれぞれLNewとRNewがもつ．

3.1.3　アルゴリズムの概要

提案アルゴリズムの概要を図4に示す．第1行の初期化ステップはすべてのレジスタとフラグを初期値にセットする．第2行のデータ入力ステップは節点データが入力され，各セルのレジスタに格納される．ここで，入力されるグラフは前もって分割されているとする．そして，節点データにおけるサイドフラグはその初期分割に応じてあらかじめ値をもっているものとする．データ入力後，第3から6行目でアルゴリズムはゲイン（LDとRDの値）によって節点をソートする．LとRの各節点集合のゲインは独立にOETソートを用いてソートされる．ここでOddTranspositionでは，各奇数番目のセルC_iはセルC_{i+1}からデータD_{i+1}をもらう．ここで，$D_i > D_{i+1}$ならば，C_iとC_{i+1}のデータを交換する．EvenTranspositionでは，偶数番目のセルがOddTranspositionの奇数番目のセルと同様の動作を行う．ここで，OETソートはOddTranspositionとEvenTranspositionをn／2回繰り返すことによりソートを行う．各セルのゲインの更新もOddTranspositionとEvenTranspositionの動作中に同時に行われる．

第7から24行のforループが本アルゴリズムのメインループである．ループの各ステップでは，節点ペアとその交換コストを出力する．(v_l, v_r)を交換する候補として選ばれた節点ペアとする．この時，v_lがR側の節点となり，v_rがL側の節点となると，これらに隣接している全節点のゲインは更新されなければならない．すなわち，各セルのゲインを更新する際，v_lとv_rが交換されたことを全節点に放送しなければならない．しかし，放送はシストリックの性質を破ってしまうので，アルゴリズムでは放送の代わりに，データをセルに順番に回している．これによりゲインの更新では時間遅れが生じてしまうが，これは結果にあまり影響を与えないと予想される．手続きMoveNewDataは交換セルで選ばれた節点IDを一時に1セルのみ左にシフトする．各セルが得た新しいデータはOddソートとEvenソートで節点データを更新するのに使われる．

各セルC_iについてl^i_{new}とr^i_{new}を直前の手続きMoveNewDataによって右隣りのセルからもらった節点IDとする．Dの値を正確に計算するためにはl^i_{new}とr^i_{new}だけではなくセルC_{i-1}のl^{i-1}_{new}とr^{i-1}_{new}と，セルC_{i-2}のl^{i-2}_{new}とr^{i-2}_{new}が必要である．これらのデータを1次元アレイ上で必要とする各セルに送るためにはそれらのセル間のリンクが必要である．

第8から13行のforループはソートセルにおいてゲインの部分ソートを行うと共に，ゲインの更新のためにLNewとRNewのデータをシフトする．第16から21行に示すようにソートセルは部分ソーティングを続ける．サーチセルと交換セルでは最適な交換コストをもつペアが選ばれる．この操作は手続きFindBestSwapで行な

```
/* Stage 1 */
for i:=1 to Δcell do
  for all search & exchage cells do in parallel begin
    ExchageCost := 2c(l_i, r_j)-LD-RD;
    if (ExchageCost, l_i, r_j) <_L (Cost^L, lmin^L, rmin^L) then begin
      Cost^L := ExchageCost;
      lmin^L := l_i;
      rmin^L := r_j
    end;
    if (ExchangeCost, l_i, r_j)<_L (Cost^R, lmin^R, rmin^R) then begin
      Cost^R := ExchageCost;
      lmin^R := l_i;
      rmin^R := r_j
    end;
    move VL, LD, (Cost^L, lmin^L, rmin^L) to the right neighbor
                                                    cell
  end;
/* Stage 2 */
for i:=1 to Δcell/2 do begin
  for all search & exchage cells do in parallel
                                  OddTranspositionWithCost;
  for all search & exchage cells do in parallel
                                  EvenTranspositionWithCost
end;
```

図5　手続き *FindBestSwap*

われる．手続き FindBestSwap の詳細は3.1.4で説明する．最後に，第23行で示した手続き MoveData ですべての節点が右にシフトされ，交換セルから節点ペアが交換コストとともに出力される．ここで，説明したアルゴリズム中には実際に交換する節点ペアを見つける部分がない．これについてはアルゴリズムの外の部分で行なわれるものと仮定する．

3.1.4 手続き FindBestSwap

この手続きはサーチセルと交換セルにおいてソートされた節点データのすべてのペアの中で最適の交換コストをもつペアを見つける．交換される節点ペアが交換セルでLNewとRNewに入れられる以外は，サーチセルと交換セルの働きは同じである．この手続きを図5に示す．この手続きは以下に示す2つのステージからなる．第1ステージでは，LとR間のすべての節点ペアの交換コストを計算する．これは節点データとLDと交換コストをサーチセルと交換セルにおいて回転させることによって行なわれる．この手続きではL側とR側の交換コストがそれぞれ ($Cost^L$, $lmin^L$, $rmin^L$) と ($Cost^R$, $lmin^R$, $rmin^R$) によって表わされる．前者はセルの間を回転し，後者はセルに残される．2つのコストの比較 ($<_L$で表す) は辞書式順序で行われる．ここで，ステージ1終了後では2つの節点が同じコストをもつ最小の交換コストをもつペアとして選ばれる．第2ステージでは，最小の交換コストをもつ節点ペアが交換セルに移るように第1ステージで計算された交換コストによって節点データがソートされる．ここでのソートはOETソートアルゴリズムによってなされ，LとRのデータは別々にソートされる．

3.1.5 時間計算量

提案アルゴリズムの時間計算量について説明する．以下では，アルゴリズムの記述においてMaxDegreeで表わされていたグラフGの最大節点次数を定数であると仮定する．空間計算量については以下の通りである．各セルはデータを格納するために O(MaxDegree) が必要である．節点次数の仮定から，各セルは定数オーダの空間が必要である．これから，グラフGの節点数をnとしてアルゴリズム全体では O(n) の空間が必要である．

時間計算量については，ΔsortとΔcellの両方とも定数であるとする．これから，アルゴリズム中のすべての手続きは定数時間で処理される．よって，図4に示すアルゴリズムでは，アルゴリズムの1回のパスの実行に O(n) 時間が必要である．

時間計算量について詳細な評価を行うために，ハードウェアのクロックを導入し，1クロックをステップと表わす．ここで，各手続きは以下に示すように数ステップ必要である．データ入力ではMaxDegreeステップ必要である．手続き OddTranspositionでは，ゲインの更新にMaxDegreeステップ必要であり，隣のセルとゲインを比較するのに1ステップ，隣のセルとデータを交換するためにMaxDegreeステップ必要である．これから2MaxDegree+1ステップ必要である．手続きEvenTranspositionはOddTranspositionと同数のステップが必要である．手続きMoveNewDataは手続きOddTranspositionやEvenTranspositionと同時に実行できるので，余分なステップは必要ではない．手続きFindBestSwapは2 Δcell $\times$MaxDegreeステップ必要である．ここで，この手続きは図4の第16から21行の forループと並列に行われる．最後に，手続きMoveDataはMaxDegreeステップ必要である．これから，アルゴリズム全体で必要なステップ数は，

$$(n/2)\times MaxDegree+$$
$$((n/2)-\Delta sort)\times(2MaxDegree+1)+$$
$$(n/2)((\Delta sort+\Delta cell)\times(2MaxDegree+1)+MaxDegree) \cdots (1)$$

となる．

ここで，現在のVLSI技術では，提案したアルゴリズムをクロックステップ100ns 以下でVLSIチップとして簡単に実現可能である．

3.2 アルゴリズムの拡張

前の節で説明したシストリックアルゴリズムはいくつかの方法に拡張できる．第1に，提案したアルゴリズムでは，アレイへのデータの入力は独立して行われている．しかし，n個のデータを順に入力し，全データの入力後にはソートを完了する1次元アレイ上で動作する O(n) ソートアルゴリズムがある．例えば，Mirankerらによって提案されたzero-timeソータ[11]である．このようなソータを各節点のゲインをソートするOETソートの代わりに使うと，データ入力フェーズと初期ソートフェーズがオーバラップし，全体の計算量を減らすことができる．ここで，OETソートの機能はゲインの部分ソートに必要であるため，OETソートとzero-timeソータの2つのタイプのソータを用意することになり，ハードウェアコストは増加する．

第2の拡張はBestChangeを計算する機能の導入である．提案したアルゴリズムは交換コストとともに節点のペアを出力する．そこで，交換コストの総和を計算して節点ペアの交換を実行する外部のハードウェア／ソフトウェアが必要である．これはハードウェアで実現できる．図6に拡張アルゴリズムの構成を示す．ここでは1つのFIFOといくつかの追加回路をもつ．アルゴリズムの出力は FIFO に入れられ，各節点IDはvertexlistと呼ぶインデックスアレイに入れられる．このアレイでは節点IDがインデックスとして使われ，FIFOでの節点の位置がデータとして格納される．コン

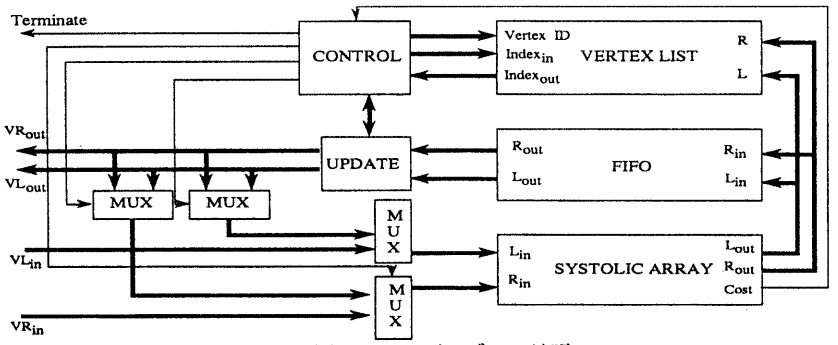

図6　アルゴリズムの拡張

トローラは BestChangeの値を決定する．次のパスでは，データはFIFOからマルチプレクサを通してセルの1次元アレイに入力される．コントローラはBestChangeの値によってマルチプレクサで節点データを交換する．マルチプレクサはまたvertexlistデータによって各節点の隣接データを更新する．このようにして，アルゴリズムは効率良く数パスを繰り返す．BestChangeが0のときはコントローラは終了信号を出力する．

アルゴリズムはマルチプロセッサ上の並列アルゴリズムとしても使用できる．アルゴリズムはデータ転送に厳密な同期は必要としない．そこで，同期システムと非同期システムのどちらの上でも実現できる．特に，ハイパーキューブマシンのような超並列マルチプロセッサシステムに向いている．ここで，ハイパーキューブマシンには常にハミルトンパス (Hamiltonian path) が存在するので，アルゴリズムの実現では1次元アレイはハミルトンパス上に埋め込まれる．

4　実験結果

4.1　Δcellの影響

提案したアルゴリズムでは，3.1.1に示したように高いゲインをもつある定数Δcell個のセルに注目し，それらの中で交換コストの高いものを見つけるようにしている．提案したシストリックアルゴリズムの振る舞いをシミュレートするプログラムを開発し，定数Δcellの値のアルゴリズムに対する影響を実験によって調べた．実験には表3に示すデータPrimSC1を用いた．Δsortは10とした．実験結果を表2に示す．実験において，各データに対して前もって10種類の初期分割をランダムに用意しておき，各初期値に対してそれぞれ実験を行った．表中で，カットサイズは10回の実験における平均値を表わす．PTimeは3.1.5で説明した式(1)により，1ステップを100nsで実行できるという仮定で計算した仮想並列実行時間を表わす．Δcellが大きいほどKL法に近い動作を行うことから，解はよくなるが，計算時間は大きくなる傾向にあることがわかる．しかし，同表から，Δcell は20程度で十分であることがわかる．

表2　実験

Δcell	カットサイズ	PTime	Δcell	カットサイズ	PTime
1	10739	0.27	15	9643	1.01
5	10179	0.57	20	9709	0.98
10	9826	0.82	30	9666	1.25

4.2　シミュレーション実験

提案したアルゴリズムを評価するためにシミュレーション実験をおこなった．実験では，Δsort とΔcell をそれぞれ10と20に設定した．提案したアルゴリズムと比較するために，KL法も実現した．両方のアルゴリズムともNECのワークステーションEWS4800/220 (30MIPS, 192MBytes) 上でC言語で実現した．表3にテストデータを示す．ここで，データ1からデータ3は一様乱数を用いて作成したデータであり，残りはMCNCのベンチマークデータである．ここで，MCNCのベンチマークデータは実際のVLSI回路のデータであるため，元のデータはハイパーグラフである．ハイパーグラフからグラフへの変換法には一般的なネット変換法を用いた．表中で，#cell, #net, #edgeはそれぞれ，節点数，ネット数，枝数を表わしている．MaxDegreeは節点次数の最大値を表わしている．

表4と表5に実験結果を示す．表中で，Min., Max., Ave.はそれぞれ，10回の実験におけるカットサイズの最小値，最大値，平均値を表わす．CPUは10回の実行の平均実行時間を表わす．diff.はKL法に対する提案したアルゴリズムのカットサイズの改善率を表わす．

表4はアルゴリズムを1パスのみ適用した後の結果を示し，表5は最終結果を示す．アルゴリズムの1パスのみの適用では，提案したアルゴリズムはKL法と比較して平均2.16%悪い解を求めているが最終結果では0.52%の解の悪化にとどまっている．これらの結果か

表3 実験データ

データ	#cell	#net	#edge	Max Degree
Data1	1000	—	10000	21
Data2	2000	—	20000	21
Data3	3000	—	30000	11
PrimSC1	914	1266	4789	45
PrimSC2	3122	3817	27183	62
Biomed	6514	7052	52055	88
Industry2	12142	13419	128914	83

ら，提案したアルゴリズムはKL法と同程度の解を求めていることがわかる．さらに，本アルゴリズムをVLSIによって実現することにより，大規模VLSI回路の分割を効率良く求めることがわかる．

5 あとがき

本稿では，グラフを分割するハードウェアアルゴリズムを提案した．今後の課題として，本アルゴリズムを実現するVLSIチップの設計，グラフをk分割するハードウェアアルゴリズムの開発，ハイパーグラフを分割するハードウェアアルゴリズムの開発などがある．

謝辞

プログラム作成に協力頂いた本学学部生廣瀬啓君に感謝します．本研究の成果の一部は文部省科学研究費補助金一般研究(C)(課題番号05680274)による．

文献

[1] S.G.Akl:"The Design and Analysis of Parallel Algorithms," Prentice-Hall (1989).
[2] P.Banerjee, M.H.Jones and J.S.Sargent :"Parallel simulated annealing algorithm for cell placement on hypercube multiprocessors," IEEE Trans. on Parallel and Distributed Systems, 1, 1, pp.91-106 (1990).
[3] A.E.Dunlop and B.W.Kernighan:"A procedure for placement of standard-cell VLSI circuits," IEEE Trans. on Computer-Aided Design, CAD-4, 1, pp.92-98 (1985).
[4] C.M.Fiduccia and R.M.Mattheyses:"A linear-time heuristic for improving network partitions," Proc. 19th Design Automation Conf., pp.175-181 (1982).
[5] M.R.Garey and D.S.Johnson:"Computers and Intractability: A Guide to the Theory of NP-Completeness," W.H.Freeman and Company (1979).
[6] B.W.Kernighan and S.Lin:"An efficient heuristic procedure for partitioning graphs," Bell Systems Technical Journal, 49, 2, pp.291-307 (1970).
[7] B.Krishnamurthy:"An improved min-cut algorithm for partitioning VLSI networks," IEEE Trans. on Computers, C-33, 5, pp.438-446 (1984).
[8] H.T.Kung:"Let's design algorithms for VLSI systems," Carnegie-Mellon University Technical Report, CMU-CS-79-151 (1979).
[9] H.T.Kung:"Why systolic architectures?," IEEE Computer, 15, 1, pp.37-46 (1982).
[10] T.Lengauer:"Combinatorial Algorithms for Integrated Circuit Layout," John Wiley & Sons (1990).
[11] G.Miranker, L.Tang and C.K.Wong:"A 'zero-time' sorter," IBM Journal of Research and Development, 27, 2, pp.140-148 (1983).
[12] J.E.Savage and M.G.Wolka:"Parallelism in graph-partitioning," Journal of Parallel and Distributed Computing, 13, pp.257-272 (1991).

表4 実験結果（1パス終了後）

データ	KL				提案アルゴリズム				diff.
	カットサイズ			CPU	カットサイズ			PTime	
	Min.	Max.	Ave.	(sec.)	Min.	Max.	Ave.	(sec.)	(%)
Data1	169559	172331	170880	18.1	169646	172492	170936	0.09	-0.03
Data2	335428	344922	341207	84.3	336920	343866	341270	0.18	-0.02
Data3	205429	209283	207443	168.0	205450	209665	207999	0.14	-0.27
PrimSC1	9848	12980	11869	9.9	10193	13134	12124	0.17	-2.15
PrimSC2	26703	31635	30028	209.0	28502	31709	30386	0.82	-1.18
Biomed	25445	60005	48378	969.0	25235	61442	51037	2.42	-5.50
Industry2	60806	71917	68533	5764.3	70106	75305	72611	3.24	-5.95

表5 実験結果（最終）

データ	KL				提案アルゴリズム				diff.
	カットサイズ			CPU	カットサイズ			PTime	
	Min.	Max.	Ave.	(sec.)	Min.	Max.	Ave.	(sec.)	(%)
Data1	160564	164097	164097	137.0	161274	165735	163443	0.66	-0.69
Data2	320951	327207	324317	721.0	323620	327334	325140	1.38	-0.25
Data3	188030	193100	192154	1439.0	191580	195353	193222	1.08	-0.56
PrimSC1	9134	11106	9738	54.7	9062	10851	9709	0.98	0.30
PrimSC2	23071	27592	26284	866.0	22976	27854	26719	3.35	-1.65
Biomed	24365	27170	25814	4513.0	24365	29486	26227	14.24	-1.60
Industry2	35799	45282	40467	28217.2	35467	42679	40148	22.70	0.79