アルゴリズム 39－1
（1994．5．13）

メッシュバス上での最小全域木アルゴリズム

堀川 豊　岩間 一雄

九州大学工学部情報工学科

本論文では，メッシュバス計算機（MBUS）上で無向グラフに対する最小全域木を求めるための $O(\log n(\log\log n)^2)$ 時間の確率アルゴリズムを与える．入力としては，n^2 個の各プロセッサに，与えられたグラフの枝と重さを与える．基本的方針は，各繰り返しでそれまで出来上がった部分木から出ている重み最小の枝を探して木に組み込むことである．繰り返し回数が $\log n$ になるので，重み最小の枝を $(\log\log n)^2$ 時間で捜すことがカギになる．また，最短経路問題のアルゴリズムも与えている．このアルゴリズムは，MBUS を 3 次元に拡張することにより，多少の変形で $O(\log^2 n\log\log n)$ 時間アルゴリズムを実現できる．

A Parallel Algorithm for Computing Minimum Spanning Trees on the Mesh of Buses

Yutaka Horikawa and Kazuo Iwama

Department of Computer Science and Communication Engineering
Kyushu University

In this paper, we present an $O(\log n(\log\log n)^2)$ algorithm on the Mesh of Buses for computing a minimum spanning tree of an undirected graph. At the beginning, each of n^2 processors holds the weight of each edge of the input graph. Basic idea is to find the minimum-weighted edge outgoing from the subtree made so far in each iteration and add the edge to the MST. Since the number of iteration is $\log n$, our key point is how to find the minimum-weighted edge in $(\log\log n)^2$ time. We also present an algorithm for the shortest path problem. We get an $O(\log^2 n\log\log n)$ algorithm by extending the mesh of buses into three-dimensional.

1. はじめに

並列アルゴリズム研究の究極的目標が，最高速でかつ最適な（並列計算機時間とプロセッサ数の積が最適な直列計算時間に等しい）アルゴリズムの発見であることは確かであるが，それがPRAM（並列乱アクセス機械）モデル上で遂行されたものであるならその実用的意義には疑問が残る．従って，より実用的なモデル上での同様の研究が重視される訳であるが，今度はモデル上の制限（主に通信機能）から来る計算時間の自明な下限が，このような研究の興味をそぐことになりかねない．例えば図1に示されるメッシュ計算機（MC）はn^2個のプロセッサ数に対して最も離れたプロセッサ間で$2n$の物理的距離があり，自明な問題を除いてはそれ以上の高速化は不可能である．キューブ網の場合は，この時間は$\log n$に減少はするが，それでも，並列計算における$\log n$は（定数係数を無視していることも考えれば）決して小さい値ではない．

メッシュバス計算機（MBUS，図2）は，MCの局所通信をバスによるグローバル通信に置き換えた並列モデルである．その物理的実現性はMCに比べてそれほど劣らないと考えられ，MCにおける通信距離による計算時間の自明な下限も存在しない．実際，グラフ問題に対しては，連結成分等の基本的問題に対して，PRAMモデルに匹敵する高速アルゴリズムが実現されることが知られている[3]．

本稿では，MBUS上で無向グラフに対する最小全域木（MST）を生成するための$O(\log n(\log\log n)^2)$時間の確率アルゴリズムを与える．入力は，n頂点の無向グラフが与えられ，各プロセッサ$P_{i,j}$には頂点i,j間の枝の重さ$w(i,j)$が与えられる．MSTアルゴリズムは，(i)連結成分を構成するアルゴリズム[5]と(ii)各代表頂点から出ている最小の重さを持つ枝を探すためのアルゴリズムを組み合わせることによって得られることが知られており，本稿のアルゴリズムもこの方針に従っている．

次に，MBUS上でグラフの最短経路を求めるための$O(n\log^2 n\log\log n)$時間の確率アルゴリズムを与える．入力としては，n^2個の各プロセッサに，与えられたグラフの枝とその重さを与える．基本的方針は，各繰り返しで各頂点対の経路の長さを計算し，その中から最短の経路を探していくことである．MCでは，この問題に対して，n^2台のプロセッサを用いた$O(n)$時間アルゴリズム[4]，PRAMでは，n^3台のプロセッサを用いた$O(\log^2 n)$時間のアルゴリズム[1]が知られている．今回のアルゴリズムの特色は，2次元ではMCより性能が劣るが，MBUSを3次元（n^3台のプロセッサ）に拡張すれば，$O(\log^2 n\log\log n)$の計算時間を達成できるという点である．（つまり，PRAMに比べてもそれほど悪くない．MCの場合は，たとえ次元を上げても多項式対数時間で実現することは不可能である．）なお，[4]のアルゴリズムは多少の手直しでMBUS上での$O(n)$時間のアルゴリズムとなる．しかし，この場合も3次元への拡張は不可能である．

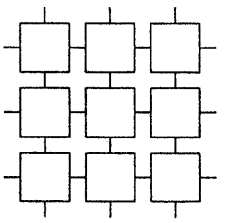

図1: MC

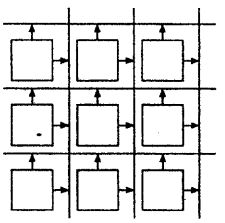

図2: MBUS

2. メッシュバス計算機モデル

図2に示されるように，MBUSは内部にローカルメモリを有するn^2個のプロセッサ$P_{i,j}(1\leq i\leq n)(1\leq j\leq n)$と2次元メッシュ上に配置されたバスからなる．例えばあるプロセッサ$P_{i,j}$の有するデータをすべてのプロセッサに伝えたいとき

は，先ず $P_{i,j}$ からつながる行バス上のプロセッサにバス通信を利用してそのデータを伝え，次に，その行の全てのプロセッサが列バスを介してそれぞれの列バス上のプロセッサ全てに伝えることにより，2ステップで実行できる．

バスには同時書き込みを許す．同時書き込みが生じたときはビットごとのOR演算が実行される（いわゆるwired-ORバス）と仮定する．例えばある行バス上のプロセッサが論理値0または1を有するとする．1を有するプロセッサが自分のプロセッサ番号（各プロセッサに唯一）をバスに書き込めば，2台以上のプロセッサが1を有している場合には，バス上で実現される値が少なくとも1つのプロセッサ番号と異なっているはずである．このことを点検することにより，1を有するプロセッサがただ1つであるかどうかの判定を定数ステップで行なうことができる．

3. 最小全域木問題

3.1 アルゴリズムのための準備

本稿で取り上げている最小全域木問題とは，与えられた無向グラフから，グラフ全域を覆っている木で，なおかつその木の枝の重さの総和が最小になるような木を探す問題である．今回は，MBUS上で無向グラフに対する最小全域木を生成するための $O(\log n(\log\log n)^2)$ アルゴリズムを与える．

MSTアルゴリズムは，次の基本的な操作により形成されている．(i) 根付き木の集合（ポインター木）が存在し，高さが1の時に“完全である”といわれる．(ii) 各繰り返しで，ポインター木 T_1 が完全であれば，頂点 $v_1(\in T_1)$ から他の木へ出ている最小の重さの枝を探す．この操作で対象となる枝は“生きている”といい，同じ木の二つの頂点間にある枝は“死んでいる”という．また選ばれた枝はMSTに含まれている．この操作をフックという．(iii) T_1 が完全でない時には，T_1 のすべての頂点がポインタージャンプ操作を実行する．この操作により，T_1 の高さは約半分になる．この(ii)(iii)の繰り返し回数は $O(\log n)$ である．(iii)の計算時間は $O(1)$ であり，(ii)は最小の重さの枝を探すために必要な時間に依存している．

このアルゴリズムの基本操作は，MBUS上で次のように実行される．各頂点 i は対角線プロセッサ $P_{i,i}$ に関連付けられる．頂点 $i_1, i_2, \cdots, i_j$ が木を構成していると仮定すると，この木の処理のために，各行（列）バス $i_1, i_2, \cdots, i_j$ とこれらのバス上のプロセッサを使える．この構造上でフック操作とポインタージャンプ操作を実行することは難しくはない．しかし，速い時間で最小の重さの枝を探す方法は明らかではない．

先ず，各行から1台のプロセッサを選択するための $O(\log\log n)$ アルゴリズム（PICK-ONE[2]）を利用して，$O(\log n(\log\log n))$ 時間で各行の最小値を探すことができる．即ち，MST生成に対して $O((\log n)^2\log\log n)$ アルゴリズムを得ることができるが，これは我々が目標とする時間よりは遅い．

MBUS全体を使って $n\times n$ の整数値の中から最小値を見つけるために $O((\log\log n)^2)$ アルゴリズム（FIND-MINI[2]）も知られている．つまり，アルゴリズムの終盤で各ポインター木が十分に大きくなっている場面では，このアルゴリズムをそのまま利用することが考えられる．しかしポインター木が小さい時，例えば頂点数が3の時は，9個のプロセッサと各3本の行（列）バスのみを使ってほぼ $3n$ 個の整数値の中から最小値を見つけなければならない．この問題は，“候補者”となる生きている枝の数を制限することにより解消される．

3.2 アルゴリズム

前記の二つの主要なサブルーチン（PICK-ONE，FIND-MINI）を用いる．

PICK-ONE.

入力： ある行の各プロセッサに0か1を与える．

出力： その行で1を有するプロセッサの中から任意の1台を選択する．

プロセッサ： その行のすべてのプロセッサが使える．通信手段として，行バスが使える．

FIND-MINI.

入力： m 本の行バス上のプロセッサに整数値を与える．この整数値の数は，行毎に n より小さい m^2 程度とする．

出力： これらの整数値の中の最小値を出力する．

プロセッサ： この m 行のすべてのプロセッサが使える．通信手段として，この m 本の行バスと対応する m 本の列バスが使える．

MSTアルゴリズムは，サブ・フェーズ，メイン・フェーズに分かれている．

サブ・フェーズ

初めにプロセッサ $P_{i,j}$ は，（頂点 i,j 間の枝の重さである）整数値か ∞ を保持していて，∞ 以外のすべての整数値は生きている．各行 i で，次のステップ1～3を $c\log n$ 回繰り返す．

ステップ1： 生きている整数（プロセッサ）に対して，ランダムな整数 $a_1, a_2, \cdots, a_{\log\log n}$ を得るために，PICK-ONEを $\log\log n$ 回実行する．（PICK-ONEが失敗した場合は，これらの整数は $\log\log n$ 個より少なくなる．）

ステップ2： 上で求めた $\log\log n$ 個の（もしくはそれより少ない）整数値の中から中央値 a を見つけて，現在の繰り返しが j 回目の時は $D_i(j) \leftarrow a$ とする．

ステップ3： a より大きい整数は死ぬ．

以下の議論では，2項分布のすそ野の可能性を議論するために，便利な以下のようなシェルノフの近似式をしばしば用いる．

$$Pr\left[X \leq (1-\theta)pn\right] \leq e^{-\frac{\theta^2 pn}{2}}$$

$$Pr\left[X \geq (1+\theta)pn\right] \leq e^{-\frac{\theta^2 pn}{2}}$$

このフェーズでは，MBUS上で各行から最小の重さの枝を選ぶための準備を行なう．各行のすべての枝を1つずつチェックしていては時間がかかり過ぎるので，速い時間で軽い枝を探す操作（バイナリーサーチ）を行なうために，次のようなデータ $D_i(1), D_i(2), \cdots$ を求めている．

最初に m 個の整数値が第 i 行に生きているとすれば，$D_i(1)$ は最小値から $\frac{m}{2}$ 番目の値に近い値になり，$D_i(2)$ は $\frac{m}{4}$ 番目に，$D_i(3)$ は $\frac{m}{2^3}$ 番目に近い値になることが期待できる．もし c が十分に大きければ，$D_i(c\log n)$ はその行の最小値といえる．このことは，以下の議論により保証される．

選ばれた $\log\log n$ 個のデータの中央値が，n 個のデータの中央値に近い値になるということは，言い換えれば，確率 $\frac{\log\log n}{n}$ で n 個の各プロセッサに手を挙げさせたとき，中央値より小さいデータを有する $\frac{n}{2}$ 個のプロセッサの中から $\frac{\log\log n}{2}$ 個が手を挙げるということである．ここで，$\frac{\log\log n}{2}$ 個から $\frac{\log\log n}{4}$ 個まで減る確率は，チェルノフの近似式を適用すると，

$$Pr\left[X \leq \frac{\log\log n}{4}\right] \leq e^{-\frac{1}{16}\log\log n}$$

となり，十分小さくなる．生きているデータの数が減ってくると，各プロセッサが手を挙げる確率は大きくなるので，上の式の値はもっと小さくなり，選ばれた $\log\log n$ 個の中央値が生き残っているデータの中央値から大きくずれることがないことが保証される．

メイン・フェーズ

このフェーズがアルゴリズムのメインになっている．入力はサブ・フェーズと同様で，さらに $D_i(1), D_i(2), \cdots, D_i(c\log n)$ が加えられる．各プロセッサは変化が起こらなくなるまで次のステップ1～7を繰り返す．最初 ∞ 以外のすべての整数値は生きている．各繰り返しで，n 行はいくつかのグループに分割され，各グループは各ポインター木に相当する．

このアルゴリズムの概要を説明する．サブ・フェーズで求めた$D_i(j)$を用いて各行でバイナリーサーチを行ない，$D_i(j+1)$よりも小さい整数値がなく，$D_i(j)$より小さい整数値が存在するような$D_i(j)$を見つけて，$D_i(j)$より大きい整数値が死ぬことにより，各行で軽い方からk個の枝(整数値)を選択できる（ステップ2，3）．そして，同じ木を構成している行のグループに対して，生きている整数値にFIND-MINIを適用して，これらの最小値を求める．そしてこの最小値を持つ枝が，この完全な木から他の木へ出ている最小の重さの枝である（ステップ4）．ステップ4で選ばれた枝を用いてフック操作を行なうことにより，異なる木が連結され，新たな1つの木が生成される（ステップ5）．そして，この新たな木を高さ1の完全な木にするためにポインタージャンプ操作を実行する（ステップ6）．これらの操作を繰り返すことにより，MSTが生成される．以下に，このアルゴリズムを示す．

ステップ1： 木が完全でなければステップ6へ．

ステップ2： 各行でバイナリーサーチを使い，$D_i(j+1)$よりも小さい整数値がなく，$D_i(j)$より小さい整数値が存在するような$D_i(j)$を見つける．

ステップ3： 各行で，$D_i(j)$より大きい整数値は死ぬ．

ステップ4： 各（完全な）木に対して，生きている整数値にFIND-MINIを適用する．もしFIND-MINIが失敗すれば，たんにステップ6へ．

ステップ5： ステップ4で得られた枝を使って，フック操作を実行する．この枝は，MSTの一部になる．

ステップ6： ポインタージャンプ操作を行なう．

ステップ7： 木が完全であれば，枝の状態を“生きている”から“死んでいる”に変える．

[定理] メイン・フェーズが停止すれば，MSTは完成している．また，$c\log n(\log\log n)^2$時間以内に停止しない確率は十分に小さい．

[証明] 今，ポインター木がm個の頂点から構成されているとする．この木に含まれる頂点から他のポインター木の頂点に出ている枝の重さの最小値をd_{mini}とする．$D_i(j) > d_{mimi} > D_i(j+1)$であれば，重さが$D_i(j+1) \sim D_i(j)$の枝がステップ2，3で選ばれる．この選ばれた枝数が少なければ，ステップ4での最小値の計算は簡単であるが，多くなればそうはいかない．よって，この枝数について議論する．

今，ステップ3の後に生きているプロセッサの数は，m行それぞれにほぼm個である．（普通は，この数はm個よりは少なくなる．）最悪の場合は，ある行で死んでいる枝の数が$m-1$本で，それらの枝がその行で軽い方から$m-1$本の枝であるときである．この時，これらすべての死んでいる枝は，$D_i(j+1)$（この行の軽い方からほぼ$m-1$番目の枝の重さ）よりも軽くなっていて，$D_i(j)$と$D_i(j+1)$の間には，$m-1$本とほぼ同数の枝が存在している．よって，生きている最小の重さの枝は$D_i(j)$と$D_i(j+1)$の間にあり，ステップ3で選ばれる枝数はmを越えないことが判る．ここで，枝数がm^2まで増える確率は，

$$Pr[X \geq m^2] \leq e^{-\frac{m(m-1)^2}{2}}$$

となり十分小さくなることから，ステップ3で選ばれる枝数はm^2まで増えることはない．もし，このような時には，FIND-MINIの入力を$m \times m^2$と考える．しかし，このような場合においても，FIND-MINIはたいした問題なく作動する．よって，ステップ4でFIND-MINIは高い確率で$O((\log\log m)^2)$時間で停止する．

他のステップの計算時間は，$O(1)$もしくは$O(\log\log n)$（ステップ2）である．主ループの繰り返し回数が，$O(\log n)$時間であることはよく知

られている． □

4. 最短経路問題

4.1 アルゴリズム

本稿で取り上げている最短経路問題とは，与えられたグラフから，各頂点間の経路の長さを計算して，その中から最短の経路を探していく問題である．

各枝 (i,j) は，各プロセッサ $P_{i,j}$ に割り当てられている．入力データ $w_0(i,j)$ として，枝が実在すれば枝の重さが，枝がなければ ∞ が，$i=j$ の時には0が与えられる．以下にMBUS上で最短経路を探すアルゴリズムを示す．(基本的な考え方は[1]と同じである．)

ステップ**1**． $l=0$ とする．

ステップ**2**． $i=1 \sim n$ まで，ステップ3，4を繰り返す．

ステップ**3**． $1 \leq j,k \leq n$ に対して，

$$q_l(i,k,j) \leftarrow w_l(i,k)+w_l(k,j)$$

ステップ**4**． $1 \leq k \leq n$ に対して，

$$w_{l+1}(i,j) \leftarrow \min\{w_l(i,j), q_l(i,1,j), \cdots, q_l(i,n,j)\}$$

ステップ**5**． $l=\log n$ であれば終了．それ以外は，$l=l+1$ としてステップ2へ．

このアルゴリズムの概要を説明する．$i=1$ の時(頂点1と他の頂点間の経路)について考える．まず，1行目の各データ $w_l(1,j)$ を各対角線プロセッサ $P_{j,j}$ を介して j 行の各プロセッサ $P_{j,k}$ に通信する．そして，各プロセッサでステップ3を行なう．次に，k 列のすべてのプロセッサがそれぞれ持つデータ $q_l(i,1,k), q_l(i,2,k), \cdots, q_l(i,n,k)$ の中から最小値を見つけて $P_{1,k}$ に通信する（ステップ4）．これらの操作を，$1 \leq i \leq n$ について行なう．この一連の動作がアルゴリズムのメインループの1ステップである．

最初の1ステップでは，異なる2頂点間の経路については，枝を2本介した状態での最短経路が求まり，次のステップでは，枝を最高4本介した状態での最短経路が，k ステップ目では，枝を最高 2^k 本介した状態での最短経路が求まる．つまりメインループの繰り返し回数は $\log n$ 回になることが判る．

次に各ステップの計算時間について考える．ステップ3は定数時間で実行できることは明らかである．ステップ4の最小値の計算には，各列から最小値を見つけるための確率アルゴリズム（SEARCH-MINI）を導入する．SEARCH-MINIは，PICK-ONEを用いることによって，$O(\log n \log\log n)$ 時間の確率アルゴリズムを実現できる．

SEARCH-MINI.

ステップ**1**： PICK-ONEを実行して，1個データを選ぶ．

ステップ**2**： ステップ1で選ばれたデータより大きいデータを持つプロセッサは死ぬ．

SEARCH-MINIの計算時間について考える．ステップ1が $O(\log\log n)$ であることは，PICK-ONEを用いていることから明らかである．ステップ2はバス通信により定数時間でできる．これらのことから，メインループの繰り返し回数が $O(\log n)$ であることを示せばよい．

各列からランダムに1個のデータを選んだ時，そのデータが中央値より小さくなる確率は $\frac{1}{2}$ であり，$\log n$ 回繰り返せば，平均で $\frac{\log n}{2}$ 回中央値より小さいデータが選ばれることが期待される．例えば，中央値より小さいデータが選ばれる回数が $\frac{\log n}{4}$ 回ぐらいまで減少する確率は，チェルノフの近似式を用いると，

$$Pr[X \leq \frac{\log n}{4}] \leq e^{-\frac{\log n}{16}}$$

となり，この確率は十分小さいことが判る．つまり，$\log n$ 回この操作を行なえば，$\frac{\log n}{4}$ 回は中央値より小さいデータが選ばれることが保証され

る．いいかえれば，$4\log n$ 回ぐらいこの操作を行なえば，生きているデータは定数個まで減るといえる．後は，PICK-ONE を定数回行ない生きているデータすべてを比べることにより最小値が求まる．

4.2 3次元 MBUS 上でのアルゴリズム

今度は，3次元 MBUS 上でのアルゴリズムについて考える．各枝 (i,j) は，各プロセッサ $P_{1,i,j}$ に割り当てられている．この枝データをバスにより各プロセッサ $P_{h,i,j}$ に通信する．2次元のアルゴリズムの $i=1$ の時の計算を n^2 個のプロセッサ $P_{1,i,j}$ で，$i=2$ の時の計算を n^2 個のプロセッサ $P_{2,i,j}$ で，$\cdots$ というように同時に行ない，n^2 個のプロセッサ $P_{l,i,j}$ で新たに計算された $w_k(l,j)$ をバスにより $1 \leq l \leq n$ の各プロセッサに通信して，1ステップが終了する．このようにして，2次元のアルゴリズムでの n 回の繰り返しが回避できる（つまり，ステップ2が省略できる）．ステップ3，4は，2次元のアルゴリズムと同様である．これによって，$O(\log^2 n \log\log n)$ アルゴリズムが達成できる．

4.3 $O(n)$ アルゴリズム

最短経路問題に対する MC 上での $O(n)$ アルゴリズム [4] を以下に示す．

ステップ1： $k=1$ とする．

ステップ2： $w_{k+1}(i,j) \leftarrow \min(w_k(i,j), w_k(i,k)+w_k(k,j))$

ステップ3： $k=n$ であれば終了．それ以外は，$k=k+1$ としてステップ2へ．

このアルゴリズムはまた，MBUS 上での $O(n)$ アルゴリズムにもなる．どのように実現されるかを，以下に示す．

ステップ2について考える．まず，k 行目のすべてのプロセッサがそれぞれ有しているデータ $w_k(k,j)$ を列バスによって，各列すべてのプロセッサに通信する．次に，k 列目のすべてのプロセッサがそれぞれ有しているデータ $w_k(i,k)$ を行バスによって，各行すべてのプロセッサに通信する．ここで，プロセッサ $P_{i,j}$ には，$w_k(i,k)$ と $w_k(k,j)$ が通信されてきて，この2つのデータの和と，もともとこのプロセッサが有していたデータ $w_k(i,j)$ の大小を比較して，小さい方のデータを $w_{k+1}(i,j)$ に読み込むことにより，ステップ2が終了する．これらの操作は，すべて定数ステップで行なわれるので，ステップ2は定数時間で計算される．アルゴリズムのメインループは，n ステップなので，全体の計算時間は $O(n)$ 時間になることが判る．しかし，このアルゴリズムは，3次元に拡張した MBUS 上でもこれ以上計算時間は速くならない．

4.4 最短経路の探索

最短経路問題では，経路の長さを計算するとともに，経路の道筋を見つけることも重要である．ここでは，MBUS 全体を使って一つの最短経路を見つける方法について考える．

まず，次のような変数を準備する．

$p_k(i,j)$： 最短経路を計算する時に経路が通過する頂点を表す．

$e(i,j)$： 枝 (i,j) が最短経路に使われているかどうかを表す．

ここで，i,j は頂点番号を，k はアルゴリズムの繰り返し回数を表している．また，最初 $p_0(i,j)$，$e(i,j)$ には，それぞれに0が与えられている．

枝 (i,j) の最短経路の長さ $w_k(i,j)$ をステップ4で計算する際に，$w_k(i,j) \leftarrow q_{k-1}(i,l,j)$ であれば，$p_k(i,j) \leftarrow l$ とし，$w_k(i,j) \leftarrow w_{k-1}(i,j)$ であれば，$p_k(i,j) \leftarrow 0$ とする．このようにして，ステップ4の計算時に，経由する頂点を覚えさせておく．つまり，このアルゴリズムでは，最短経路の長さ $w_k(i,j)$ の計算をする時には，後にどの経路が欲しいのかは言う必要がなく，どのような

経路の探索にも対応できるようになっている．

ここで，頂点i,j間の最短経路の探索方法について説明する．すでに，最短経路の長さは求まっているとする．

まず，プロセッサ$P_{i,j}$で，最終的な最短経路の長さが$w_k(i,j)$で，$p_k(i,j) = l$であれば，$P_{i,j}$から$P_{i,l}$と$P_{l,j}$にそれぞれ$p_{k-1}(i,l)$，$p_{k-1}(l,j)$を調べに行く．そして，$p_{k-1}(i,l) = m$であれば，$P_{i,l}$から$P_{i,m}$と$P_{m,l}$にそれぞれ$p_{k-2}(i,m)$，$p_{k-2}(m,l)$を調べに行く．また，$p_{k-1}(l,j) = 0$の時には，$p_{k-2}(l,j)$を調べる．このようにして，$p_0(x,y)$まで調べる．この時，$p_0(x,y)$を有するプロセッサ$P_{x,y}$に割り当てられている枝(x,y)は最短経路に使われており，$e(i,j) \leftarrow 1$とする．これらの操作は，アルゴリズムの全く逆の操作を行なっているので，実行時間はアルゴリズムのメインループと同じ$O(\log n)$となる．

上記のようにして，最短経路に使われている枝は分かった．次は，その枝がどのような順番でつながっているかについて考える．まず，$e(x,y) = 1$を有するプロセッサ$P_{x,y}$が行バスを用いて対角線プロセッサ$P_{x,x}$に自分のプロセッサ番号yを通信する．これにより，経路上で頂点xに来れば，次は頂点yに行くことにする．つまり，頂点xから頂点yにポインターを引くことにする．また，頂点i,j間の最短経路を見つける時には，頂点iをポインターの始まりとして，頂点jをポインターの終りとする．こうして，ポインターをたどって，対角線上のプロセッサを見ていけば，最短経路が見つかることが分かる．

5. おわりに

本論文では，代表的なグラフ問題である最小全域木問題と最短経路問題をとりあげて，MBUS上でそれぞれの問題に対して，$O(\log n(\log\log n)^2)$時間，$O(\log^2 n \log\log n)$時間の確率アルゴリズムを与えた．今後は，他のグラフ問題に対して，MBUS上でのアルゴリズムを考えていきたい．

参考文献

[1] E. Dekel, D. Nassimi and S. Sahni. Parallel matrix and graph algorithms. *SIAM J. Comput.*, pp.657-675,1981.

[2] 堀川，岩間．探索問題に対するメッシュバス上での並列アルゴリズム．並列処理シンポジウムJSPP '93．pp.207-214.

[3] K. Iwama and Y. Kambayashi. An $O(\log n)$ parallel connectivity algorithm on the mesh of buses. *Proc. 11th IFIP World Computer Congress*, pp.305-310,1989.

[4] K. N. Levitt and W. H. Kautz. Cellular arrays for the solution of graph problems. *Commun. ACM*, pp.789-801,1972.

[5] Y. Shiloah and U. Vishkin. An $O(\log n)$ parallel connectivity algorithm. *J. Algor.*, pp. 57-67, 1982.