3T−5

グラフの最小全域木を求めるための メッシュバス上での並列アルゴリズム

堀川 豊　　岩間 一雄
九州大学工学部

1. まえがき

メッシュバス計算機(MBUS, 図2)は, メッシュ計算機(MC, 図1)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[1].

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$O(\log n(\log\log n)^2)$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$edge_{i,j}$が与えられる. また, MSTアルゴリズムは, (i)並列結合アルゴリズムで代表頂点を維持するアルゴリズム[2], (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いる.

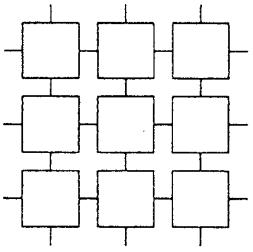

図1: MC

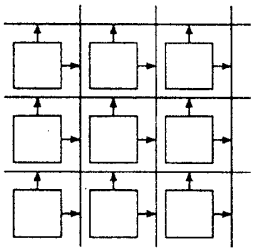

図2: MBUS

2. メッシュバス計算機モデル

図2に示すように, 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$P_{i,j}$と2次元メッシュ上に配置されたバスからなる.

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つ以上のプロセッサが1を有している場合には, バス上で実現できる値が少なくとも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.

A Parallel Algorithm for Computing Minimum Spanning Trees on the Mesh of Buses
Yutaka Horikawa and Kazuo Iwama
Faculty of Engineering, Kyushu University

3. アルゴリズムのための準備

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)根を持った木の集合(*pointer trees*)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, T_1(*pointer tree*)が完全であれば, 頂点$v_1(\in T_1)$と頂点$v_2(\in T_2 \neq T_1)$につながっ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, 他の枝(例えば, 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*hooking*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*pointer jumping*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. また, (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*hooking*と*pointer jumping*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[3])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[3])が知られている. これは木のサイズが十分に大きい時にはそのまま利用できる. 小さい時(例えば頂点数が3)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. この難しさは, "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.

4. 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 m^2 程度である.
出力： これらの整数値の中から最小値を出力する.
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, 二つのフェーズ (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.

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.

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 *pointer tree* 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 *hooking* 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 *pointer jumping* 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, *hooking* を実行する. この枝は, MST の一部になる.
ステップ6： *pointer jumping* を実行する.
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 サブ・フェーズのステップ1で PICK-ONE の失敗確率は十分に小さいので, ステップ3の後に, 各行に行きている枝の数は平均的には, 木を構成している頂点数 (m) を越えないことが判る. つまり, 生きている枝の数が m^2 まで大きくなる確率は十分に小さい. そのような時でも, 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. 他のステップでは, $O(1)$ もしくは $O(\log\log n)$(ステップ2) 必要である. 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. □

参考文献

[1] 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.
[2] Y. Shiloah and U. Vishkin. An $O(\log n)$ parallel connectivity algorithm. *J. Algor.*, pp. 57-67, 1982.
[3] 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.