アルゴリズム 74－1
（2000．9．21）

有向グラフの最長路を求める効率良い並列アルゴリズム

多田 昭雄　　中村 良三
（崇城大学）　　（熊本大学）

概要

有向グラフ（節点数n，辺数m）に対して効率よく最長路を求める並列アルゴリズムを提案する．具体的には，はじめに有向グラフを入出次数が高々1の線形リストに分割し，各節点にランク値を設定する．そして節点の半順序関係を求めて，リストごとの節点のランク値の増分を修正しながら，グラフ全体の最長路を求める並列アルゴリズムである．このアルゴリズムの計算量は，CREW-PRAM並列計算機モデル上で，プロセッサ数が$O(m+n)$，計算時間が$O(\log^2 m)$である．

An Efficient Parallel Algorithm for Finding the Longest Simple Path in a Directed Graph

Akio Tada（Sojo University）
Ryozo Nakamura（Kumamoto University）

Abstract

This paper presents an efficient parallel algorithm for finding the longest simple path in a directed graph, using divide and conquer method. Namely this algorithm at first divides a directed graph into the several linked lists, in which each vertex has at most 1 both input and output edge degree. Then in combined process, each two vertices are merged to update an increment of a rank value on each linked list. Finally it gets the longest simple path in the given directed graph. The proposed algorithm requires $O(m+n)$ processors and $O(\log^2 m)$ time on a CREW-PRAM model.

1　はじめに

有向グラフ（節点数をn，辺数をmとする）の最長路を求める問題は，しばしばグラフの応用問題のひとつとして取り上げられる．

この問題に対する並列アルゴリズムの研究も以前から進められ，従来の提案では，CRCW-PRAM並列計算機モデルにおいて，行列を用いて$O(n^3)$台のプロセッサで$O(\log n)$時間で実行するアルゴリズムがある．このアルゴリズムは行列の計算に依存しているので，$O(n^3)$台のプロセッサ数を減少させることは困難である．

本論文では，分割統治法などの基本的な並列アルゴリズムを用いて，有向グラフに対して効率よく最長路を求める並列アルゴリズムを提案する．具体的には，はじめに，有向辺を出節点と入節点の組で表した有向グラフを入出次数が高々1の線形リストに分割し，各リスト上の節点にランク値を設定する．次に，各リストのすべての有向辺を半順序関係で表し，リストを併合しながら節点の半順序関係を求める．最後に，節点の半順序関係を用いて，節点の組を順次併合しながらリストごとの節点のランク値の増分を修正する．このようにして，グラフ全体の最長路を求める．このアルゴリズムの計算量は，CREW-PRAM 並列計算機モデル上で，プロセッサ数が$O(m+n)$，計算時間が$O(\log^2 m)$である．

以下，第2章では，提案する並列アルゴリズムの概要を述べる．第3章では，そのアルゴリズムの詳細を述べ，アルゴリズムの正当性と時間計算量を考察する．

2　最長路を求める並列アルゴリズムの概要

対象とする有向グラフは，自己閉路や多重辺を含まない有向単純グラフG（V，E），|V|= n，|E|= m とし，グラフの各節点は1から通し番号を付けて表す．有向グラフの入力は配列の形で与えられ，配列 OV が出節点，配列 IV が入節点を示し，各有向辺は OV[i]から IV[i]への向きで表される．すなわち有向辺は OV[1..m]と IV[1..m]の対で表され，出節点は節点番号の順で整列されていると仮定する．

具体例として，図1に有向グラフの例と図2にその入力例を示す。

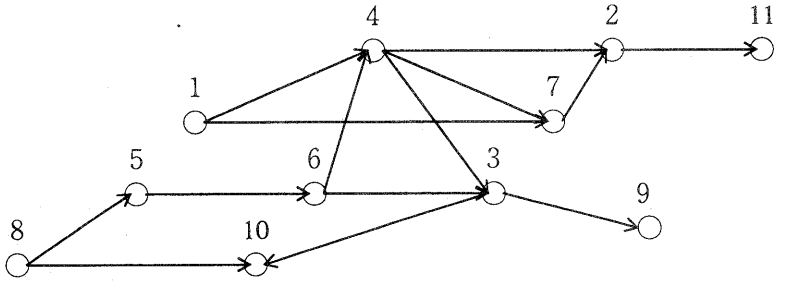

図1　有向グラフの例

Fig. 1　Instance of a directed graph.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
OV	1	1	2	3	3	4	4	4	5	6	6	7	8	8
IV	4	7	11	9	10	2	3	7	6	3	4	2	5	10

図2　有向グラフの入力

Fig. 2　Input array of Fig.1.

最長路を求める並列アルゴリズムは大きく分けて，次の3つのステージから構成される．

ｽﾃｰｼﾞ1．有向グラフを入出次数が高々1の線形リストに分割するアルゴリズム

ｽﾃｰｼﾞ2．線形リストを用いて節点の半順序関係を求めるアルゴリズム

ｽﾃｰｼﾞ3．節点の半順序関係を用いて最長路を求めるアルゴリズム

3　アルゴリズムの詳細化

ｽﾃｰｼﾞ１．有向グラフを入出次数が高々１の線形リストに分割するアルゴリズム

与えられた入力配列に基づいて，各節点の入出次数が高々１となるように節点を分割し，入力された有向グラフを線形リストに分割する．このとき，分割される節点を分割節点と呼び，一つの節点が分割され新たに生成された節点を兄弟節点と呼ぶ．

ｽﾃｯﾌﾟ［１.１］　各節点の出次数と入次数を求め，大きい方の値をその節点の最大次数とする．

具体的には，まず下記のアルゴリズムによって，各節点の出次数を出節点 OV[1..m]から配列 O[1..n]に求める．ここで B[1..n]は，出節点 OV[1..m]の節点番号が異なる境界の辺番号を記憶し管理する配列である．

```
for  1 ≦ e ≦ m-1  in parallel do
     if  OV[e] ≠  OV[e+1]
         then  B[OV[e]] ← e
for  2 ≦ v ≦ n  in parallel do
     O[v] ← B[v] - B[v-1]
```

また，各節点の入次数 I[1..n]は最初に入節点 IV[1..m]を整列して，その後は前述した出次数を求めるアルゴリズムと同様にして求める．

そして，各節点の出次数と入次数の大きい値を最大次数とし，各節点において最大次数だけ兄弟節点を生成する．各節点の兄弟節点数を示す配列を C[1:n]とする．

ｽﾃｯﾌﾟ［１.２］　各節点の兄弟節点数に基づき，その兄弟節点には連続した通し番号を付ける．さらに入力された有向グラフの節点番号の大きさの順序を保つように連続した新しい節点番号を付け，有向グラフを新節点番号に基づき線形リストに分割する．

具体的には，各節点の兄弟節点数の累計和を与える配列を，S[1..n]とすると，新節点番号と旧節点番号の対応表を配列 N[1..n+m]に下記の計算で求める．ここで配列 N の添字は新節点番号で内容が旧節点番号である．累計和 S[1..n]は C[1..n]を用いて部分和(Prefix sum)アルゴリズムより求める．

次に，各節点において兄弟節点の先頭に対する新節点番号と旧節点番号の対応を下記のアルゴリズムによって求める．

```
for  2 ≦ i ≦ n  in parallel do
         j ← S[i-1] + 1
         N[j] ← i
```

さらに各分割節点において先頭の兄弟節点が保持している旧節点番号を兄弟節点にﾌﾞﾛｰﾄﾞｷｬｽﾄすることによって，新節点番号と旧節点番号の対応表ができる．

次に，この対応表 N[1..n+m]に基づき，入力配列を新しい節点番号で作り直す．具体的には出節点の境界の辺番号を示す B[0..n]と節点の累計和 S[0..n]より，新しい出節点は配列 NOV[1..m]に下記のアルゴリズムによって求める．

for $1 \leqq e \leqq m$ in paralell do

$NOV[e] \leftarrow S[OV[e]-1] + (e - B[OV[e] - 1])$

また，新しい入節点 NIV[1..m]は入節点 IV[1..m]を整列して，出節点と同様にして求める．以後特にことわらない限り，新しい節点を単に節点と呼ぶ．

ｽﾃｯﾌﾟ［1.3］　各線形リストをダブリング技法を用いてなぞり，リストの先頭の節点を1として各節点のランク値を求める．リスト番号は始め先頭の節点番号とし，さらに昇順に整列して一連番号を与える．

これは既成のダブリング技法と並列整列アルゴリズムを用いて求める．

ｽﾃｰｼﾞ1のアルゴリズムの正当性と計算量に関する考察

ｽﾃｯﾌﾟ[1.1]のアルゴリズムの正当性は，CREW-PRAM モデル上で，自明である．また，計算量としては，各節点の出次数を求めるには，プロセッサ数$O(m)$で計算時間は$O(\log n)$であるが，入次数を求めるには，配列 IV[1..m]を昇順に整列する必要がある．整列は CREW-PRAM モデル上の単純な並列整列アルゴリズムで$O(\log^2 m)$であるので，このステップの計算量は，プロセッサ数は$O(m)$で，計算時間は整列を入れて$O(\log^2 m)$である．

ｽﾃｯﾌﾟ[1.2]の処理では，既知な部分和，ﾌﾞﾛｰﾄﾞｷｬｽﾄ及び整列の並列アルゴリズムのみを用いてアルゴリズムを設計しているので，その正当性は明らかである．計算量は，分割したグラフの新節点数は高々 n+m-1 個であるので，各節点に1台のプロセッサを割り当てると，プロセッサ数は$O(m+n)$で，計算時間は部分和，ﾌﾞﾛｰﾄﾞｷｬｽﾄ及び整列に要する時間$O(\log^2 m)$となる．

ｽﾃｯﾌﾟ[1.3]では，自明な並列アルゴリズムであるダブリング技法と並列整列アルゴリズムを用いて処理できる．計算量は，節点数が高々 n+m-1 個であるので，プロセッサ数は$O(m+n)$である．また，整列並列アルゴリズムが含まれるので，$O(\log^2 m)$となる．

以上の考察から，ｽﾃｰｼﾞ1は CREW-PRAM モデル上でプロセッサ数$O(m+n)$で，計算時間$O(\log^2 m)$で可能である．

図1の具体例では，ｽﾃｯﾌﾟ[1.1]およびｽﾃｯﾌﾟ[1.2]の兄弟節点の累計和は図3のようになる．

節点番号	1	2	3	4	5	6	7	8	9	10	11
出次数:O[]	2	1	2	3	1	2	1	2	0	0	0
入次数:I[]	0	2	2	2	1	1	2	0	1	2	1
兄弟節点数:C[]	2	2	2	3	1	2	2	2	1	2	1
累計和:S[]	2	4	6	9	10	12	14	16	17	19	20

図3　ｽﾃｯﾌﾟ[1.1]の結果

Fig. 3　Result after step[1.1].

また，ｽﾃｯﾌﾟ[1.2]の新節点番号と旧節点番号の対応表は，図4のようになり，図2の入力配列は図5のように新しい節点番号に置き換わる．

新節点番号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
旧節点:N[]	1	1	2	2	3	3	4	4	4	5	6	6	7	7	8	8	9	10	10	11

図4　新節点番号/旧節点番号対応表

Fig. 4　New/old vertex number corresponding table.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
NOV	1	2	3	5	6	7	8	9	10	11	12	13	15	16
NIV	7	13	20	17	18	3	5	14	11	6	8	4	10	19

図５　新しく構成された有向グラフの入力

Fig. 5　New input array.

図５に基づいてグラフを描くと，図１のグラフは図６のようになり，６つの線形リストに分割される．

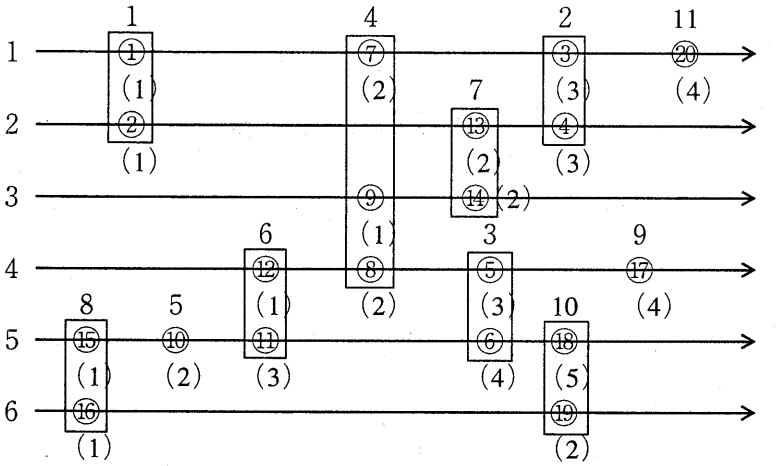

図６　分割されたグラフ

Fig. 6　Divided graph.

ｽﾃｰｼﾞ２．線形リストを用いて節点の半順序関係を求めるアルゴリズム

ｽﾃｯﾌﾟ［２.１］ｽﾃｰｼﾞ１で作成した線形リストから，分割節点のみからなるリストを作成する．これは兄弟節点数が１の節点を削除することで作成できる．

ｽﾃｯﾌﾟ［２.２］

（１）すべての有向辺に対して，次のような半順序関係（＜）で表す．例えば，　$a_1 < a_2$　と表す．ここで，a_1は出節点の分割節点番号で，a_2は入節点の分割節点番号である．

（２）各リスト上の有向辺の半順序関係について，推移律に従って圧縮する．例えば，　$a_1 < a_2$，$a_2 < a_3$　ならば，　$a_1 < a_2 < a_3$　と表す．

repeat　log(線形リストの数＝Ｌ)　do

｛　/*　以下のｽﾃｯﾌﾟ[2.3]から[2.5]までを繰り返す　*/

ｽﾃｯﾌﾟ［２.３］重複しない２つの線形リストからなる組を作る．すなわち，（ｉ，ｉ＋１），i=2j-1,(j=1,2,… $L/2^{loop}$)なる線形リストの組を作る．

ｽﾃｯﾌﾟ［２.４］線形リストの組で，ｽﾃｯﾌﾟ[2.2]で求めた半順序関係を比較し，併合する．このとき，

（1） $a_i < a_j$, $a_j < a_r$ が成立すれば，推移的に新たな半順序関係 $a_i < a_j < a_r$ を作る．

（2） $a_i < a_r$, $a_j < a_r$ など順序関係が不定な場合は任意な並び $(a_i, a_j) < a_r$ とする．

（3）リスト上に同一の番号が存在すれば，ループとなるので，同一の番号が現れる直前で打ち切り印を付ける．

ｽﾃｯﾌﾟ［2.5］リストの組（i, i＋1），i=2j-1,(j=1,2,… $L/2^{loop}$)の2つのリスト番号を ⌈(i＋1)／2⌉ に更新する．

} /* end repeat */

このようにして，分割節点の半順序関係を求める．

ｽﾃｰｼﾞ2のアルゴリズムの正当性と計算量に関する考察

ｽﾃｯﾌﾟ[2.1]の処理の正当性は自明で，計算量はプロセッサ数 $O(m+n)$ で，計算時間は線形リスト上の節点の数が高々 n-1 個なので，ダブリング技法で $O(\log n)$ で処理できる．

ｽﾃｯﾌﾟ[2.2]の処理の正当性は自明で，計算量はプロセッサ数 $O(m+n)$ で定数時間 $O(1)$ である．

ｽﾃｯﾌﾟ[2.3]の処理の正当性は自明で，計算量はプロセッサ数 $O(m)$ で計算時間は定数時間 $O(1)$ である．

ｽﾃｯﾌﾟ[2.4]の処理の正当性は推移律より明らかである．計算量はプロセッサ数 $O(m+n)$ で，計算時間は(1)(2)は定数時間 $O(1)$，(3)は $O(\log n)$ である．

ｽﾃｯﾌﾟ[2.5]のリスト番号の更新はプロセッサ数 $O(m)$ で，計算時間は定数時間 $O(1)$ である．

ｽﾃｯﾌﾟ[2.3]からｽﾃｯﾌﾟ[2.5]までの繰り返し回数は，分割された線形リストの数は高々 m-1 なので，log m 回となる．従って，ｽﾃｰｼﾞ2の計算量はプロセッサ数が $O(m+n)$ で計算時間は $O(\log m \log n)$ である．

具体例では，分割節点の半順序関係は図7のようになる．

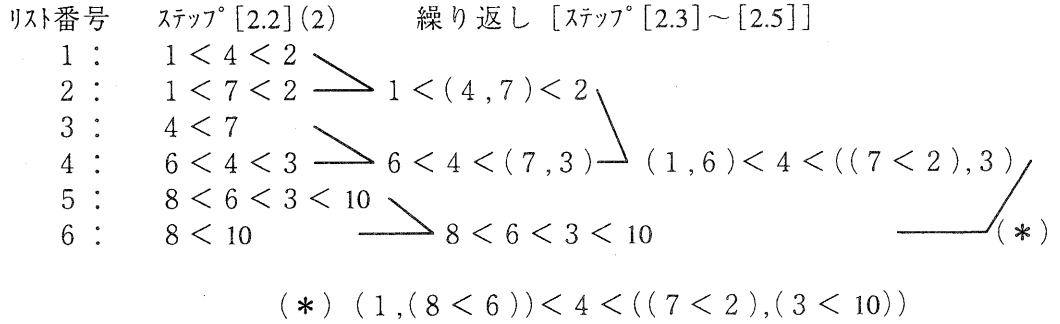

図7　分割節点の半順序関係

Fig. 7　Partial ordering of vertices.

ｽﾃｰｼﾞ 3．節点の半順序関係を用いて最長路を求めるアルゴリズム

ｽﾃｯﾌﾟ［3.0］各分割節点の兄弟節点のランク値がその最大値に等しくなるような増分値 g を算定する．

repeat log(分割節点の数) do
{ /* 以下のｽﾃｯﾌﾟ[3.1]から[3.5]までを繰り返す */
ｽﾃｯﾌﾟ［3.1］分割節点の隣接した組を作る．
これはｽﾃｰｼﾞ 2 で求めた分割節点の半順序関係より，内側の（ ）中の半順序関係を優先して組み合わせる．
ｽﾃｯﾌﾟ［3.2］各組での処理：
同一の線形リスト上で次のような処理を行う．
（1）同一リスト上に対応する節点があるとき
a → b （a，b は増分値，→は有向辺の流れを示す）
if a ＞ b then 純増分 p ＝ a － b を求める．
（2）同一リスト上に対応する節点がないとき
空→空 => 空
空→ b => p ＝ 0
a →空 => p ＝ 0，g ＝ a
ｽﾃｯﾌﾟ［3.3］併合した各兄弟節点において，純増分の最大値を求める．
ｽﾃｯﾌﾟ［3.4］純増分の最大値を用いて，各兄弟節点の増分値を修正する．
増分値は有向辺に沿った流れの節点に属する兄弟節点に対して修正する．このため，兄弟節点が属する最小のリスト番号を記憶しておく．
ｽﾃｯﾌﾟ［3.5］分割節点が記憶された配列のインデックスを更新する．
} /* end repeat */

ｽﾃｯﾌﾟ［3.6］繰り返しの結果，各線形リストの最大の増分値が求まる．
（1）そして，各リストの最大のランク値をもつ節点に増分値を加えて最終ランク値を求める．
（2）すべての節点の中で最大のランク値をもつ節点までのパスがそのグラフの最長路となる．

ｽﾃｰｼﾞ 3 のアルゴリズムの正当性と計算量に関する考察

ｽﾃｯﾌﾟ[3.0]の正当性は既知のアルゴリズムを用いているので自明である．計算量としては，プロセッサ数 $O(m+n)$ で，計算時間は兄弟節点のランク値の最大値は平衡 2 分木法により $O(\log n)$ で求められる．

ｽﾃｯﾌﾟ[3.1]の正当性は自明である．プロセッサ数は $O(m+n)$ で定数時間 $O(1)$ で処理できる．

ｽﾃｯﾌﾟ[3.2]は有向辺の流れに沿って処理するので正当性は自明である．計算量はプロセッサ数 $O(m+n)$ で定数時間 $O(1)$ で処理できる．

ｽﾃｯﾌﾟ[3.3]の正当性は自明で，プロセッサ数 $O(n)$，計算時間 $O(\log m)$ である．

ｽﾃｯﾌﾟ[3.4]は有向辺の流れに沿ってそれ以降に含まれる兄弟節点に増分値を修正する．計算量はプロセッサ数 $O(m+n)$ で計算時間は $O(\log m)$ である．

ｽﾃｯﾌﾟ[3.5]のインデックス更新はプロセッサ数$O(n)$で定数時間$O(1)$である．

ｽﾃｯﾌﾟ[3.1]から[3.5]までの繰り返しの回数は log n 回であるので，繰り返しの部分は$O(\log m \log n)$時間となる．

ｽﾃｯﾌﾟ[3.6]の正当性は自明である．計算量はプロセッサ数$O(m)$で計算時間は定数時間$O(1)$である．

従って，ｽﾃｰｼﾞ3の計算量はプロセッサ数$O(m+n)$で計算時間は$O(\log m \log n)$となる．

具体例では，ｽﾃｰｼﾞ3は図8のようになる．従って図6の節点⑳（分割節点 11）のランクが最大値7となり，その節点までのパスがグラフの最長路となる．

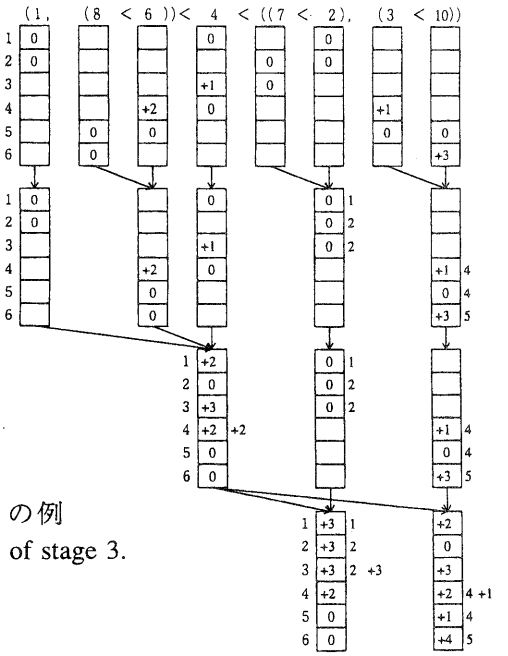

図8　ｽﾃｰｼﾞ3の例

Fig. 8　Example of stage 3.

4　まとめ

有向グラフにおける最長路を求める問題について，分割統治法などの基本的な並列アルゴリズムを用いて求める効率的な並列アルゴリズムを提案した．このアルゴリズムは CREW-PRAM 並列計算機モデル上で，プロセッサ数が$O(m+n)$で計算時間が$O(\log^2 m)$である．

今後の課題として，このアルゴリズムを応用して，有向グラフの強連結成分を求める並列アルゴリズムについても考察していきたい．

参考文献

1）Gibbons, A. and Rytter, W.: Efficient Parallel Algorithms, *Cambridge University Press,* pp.6-18（1988）.

2）Xavier, C. and Iyegar, S.S.: Introduction to Parallel Algorithms, *Wiley-inter science,* pp.188-201（1998）.

3）Cole, R.: Parallel Merge Sort, *SIAM J. Comput.* Vol.17,No.4, pp. 770-785（1988）