テクニカルノート

2 分探索木を通りがけ順になぞる並列アルゴリズム

右田雅裕[†] 中村良三[††]

2 分探索木を通りがけ順になぞる並列アルゴリズムを並列計算機モデル CREW PRAM のもとで提案する．このアルゴリズムでは，はじめにオイラーツアー技法を用いて 2 分探索木のオイラー閉路を求め，その走査リストから簡潔で効率良く通りがけ順の値を算定する並列アルゴリズムを示す．この並列アルゴリズムの時間計算量は，節点の数を N とすると，$O(N)$ のプロセッサを用いて通りがけ順の値を $O(\log N)$ 時間で求めることができる．

Parallel Algorithm for Inorder Traversal of a Binary Search Tree

Masahiro Migita[†] and Ryozo Nakamura[††]

We propose an efficient parallel algorithm to number the vertices in inorder on a binary search tree by using Euler tour technique. The proposed algorithm can be implemented in $O(\log N)$ time with $O(N)$ processors in CREW PRAM, provided that the number of nodes in the tree is N.

1. はじめに

2 分探索木は，順序が付いた要素の挿入・探索・削除が頻繁に行われる集合に対して，アルゴリズムを効率良く実現できるデータ構造である．この木のすべての要素を系統的に効率良くなぞる方法として，行きがけ順（preorder），帰りがけ順（postorder）および通りがけ順（inorder）があり，特に通りがけ順は木の各要素を昇順になぞる重要ななぞりである．

上記のなぞりを並列に実行するアルゴリズムでは，オイラーツアー技法（Euler tour technique）を用いて，走査リスト（traversal list）を生成し，そのリストから行きがけ順と帰りがけ順になぞる各サブリストを作り，それらのなぞりを求める方法が文献 1) で示されているが，通りがけ順のなぞりを求めるアルゴリズムは明らかでない．一方，文献 2), 3) の平衡 2 分探索木を構成する並列アルゴリズムの中では，各節点において親や子の存在条件に基づき，通りがけ順を構成する走査リストの要素には重み 1，それ以外の要素には重み 0 を付け，この重みに対して並列 prefix-sum を適用して通りがけ順の値を求めている．また，文献 4) においても，走査リストの各要素における親子の関係や子の存在を参照することによって，文献 2), 3) と同様な重みづけを行い，並列 prefix-sum を用いて通りがけ順の値を求めている．

本稿で提案する並列アルゴリズムは，まず隣接リストで表された 2 分探索木にオイラーツアー技法を用いてオイラー閉路を求め，その閉路から根がリストの先頭になるような走査リストを作る．次に，その走査リストの各要素に簡単な情報を持たせるだけで通りがけ順になぞるサブリストを生成し，そのサブリストに対して並列リストランキングアルゴリズムを適用して通りがけ順の値を求めるアルゴリズムである．この提案する並列アルゴリズムは CREW PRAM モデルのもとで節点数の対数オーダの時間計算量で実行できる．

2. 提案する並列アルゴリズム

提案する並列アルゴリズムは，オイラーツアー技法を用いて 2 分探索木のオイラー閉路から走査リストを生成し，そのリストに基づき通りがけ順の値を効率良く算定する並列アルゴリズムである．提案するアルゴリズムでは，共有記憶領域に次のようなデータ構造を定義し，2 分探索木を配列 T によって次のように表す．

```
type index = 1..N;
     node  = record key:keytype; num:index;
  { 節点 :  keytype は定義済，通りがけ順の値 }
            parent,left,right:index;  end;
            { 親，および左と右の子 }
var   T = array[index] of node;
```

文献 1), 5) に提示されている木をなぞる並列アルゴリズムでは，隣接リスト（このリストでは，辺 (i,j) と辺 (j,i) は相互に参照可能であるように構成する）で表された木にオイラーツアー技法を用いて，オイラー閉路を求め，その閉路から根がリストの先頭になるような走査リストを作る．オイ

† 熊本大学総合情報処理センター
Information Processing Center, Kumamoto University
†† 熊本大学工学部数理情報システム工学科
Department of Computer Science, Faculty of Engineering, Kumamoto University

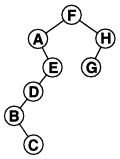

図 1 2 分探索木の例
Fig. 1 Example of a binary search tree.

配列 T	key	num	parent	left	right
1	F	6	0	2	3
2	A	1	1	0	5
3	H	8	1	4	0
4	G	7	3	0	0
5	E	5	2	6	0
6	D	4	5	7	0
7	B	2	6	0	8
8	C	3	7	0	0

図 2 図 1 の 2 分探索木の表現
Fig. 2 Binary Search tree Representation of Fig. 1.

ラーツアー技法とは，木の各辺を互いに逆向きの 2 辺で置換したようなオイラー閉路を構成する並列計算である．次に，この走査リストの各要素（辺）のランクを算定する．ここで，リストの要素 k のランクとは，リストの先頭から要素 k までの要素の個数として定義される．このランク付けはダブリング技法（doubling technique）を用いて，n 個の要素からなるリストを EREW PRAM 計算機モデルで，プロセッサ数 $O(n/\log n)$ を用いて $O(\log n)$ の時間計算量で計算できることが知られている[1),5)]．

走査リスト上において各要素のランクが求まると，木の任意の枝 (i,j) は，走査リスト上ではランクの値の小さい辺 (i,j) とランクの値の大きい辺 (j,i) とが対になって現れる．ここで，そのランクの値の小さい辺を先行辺（advance edge），ランクの値の大きい辺を後退辺（retreat edge）とよぶ．

提案する並列アルゴリズムでは，まず前述のオイラーツアー技法を用いて，2 分探索木のオイラー閉路から走査リストを生成し，その走査リスト上の各辺の先行辺と後退辺を求める．

たとえば，**図 1** の 2 分探索木を**図 2** のようなデータ構造で表現すると，2 分探索木の隣接リストは，この木が順序木であることを考慮して，そのリストの要素は親（`parent`），左の子（`left`），右の子（`right`）の順に欄を連結して考える．ただし，欄の値が零（0）の場合はその要素が欠けているものと解釈する．

次に，この隣接リストを用いてオイラー閉路を求め，その 2 分探索木の根がリストの先頭になるような走査リストを作り，リスト上のランク付けを用いて走査リスト上の先行辺と後退辺を算定する．

最後に，走査リスト上の先行辺と後退辺が決まれば，2 分探索木の行きがけ順，帰りがけ順および通りがけ順の節点のなぞりは次のように効率良くできる．

i) 行きがけ順（preorder）

走査リスト上から先行辺となる要素のサブリストを下記のように生成する．

$$(i_1,j_1) \longrightarrow (i_2,j_2) \longrightarrow \cdots \longrightarrow (i_n,j_n)$$

上記のサブリストの先頭の要素 (i_1,j_1) は最初に i_1，次に j_1 をなぞり，第 k の要素 (i_k,j_k) $(k \geq 2)$ からは j_k のみをなぞる．すなわち行きがけ順の節点のなぞりは，次のようになる．

$$i_1 \longrightarrow j_1 \longrightarrow j_2 \longrightarrow j_3 \longrightarrow \cdots \longrightarrow j_n$$

ii) 帰りがけ順（postorder）

走査リスト上から後退辺となる要素のサブリストを下記のように生成する．

$$(i'_1,j'_1) \longrightarrow (i'_2,j'_2) \longrightarrow \cdots \longrightarrow (i'_n,j'_n)$$

上記のサブリストの先頭から順次に辺 (i'_k,j'_k), $(k \in L)$ の i'_k を取り出し，リストの最後の要素は (i'_n,j'_n) を取り出す．その結果，節点のなぞりは次のようになる．

$$i'_1 \longrightarrow i'_2 \longrightarrow i'_3 \longrightarrow \cdots \longrightarrow i'_n \longrightarrow j'_n$$

上記の i)， ii) のなぞりの骨子はすでに文献 1) に提示されているが，通りがけ順のなぞりについては明らかでない．

提案するアルゴリズムでは，上記の i)， ii) のアルゴリズムと同様に，走査リストから通りがけ順を構成するサブリストを生成し，そのサブリストの要素である辺の先頭の節点を順次取り出せば通りがけ順のなぞりが求められる．このアルゴリズムでは，走査リストの各要素に，先行辺または後退辺と右枝または左枝という単純な情報を持たせ，その情報のみで各要素が自分自身が通りがけ順の要素であるかどうか判断する．

iii) 通りがけ順（inorder）

i) 行きがけ順， ii) 帰りがけ順の節点のなぞりは容易にできる．しかし，通りがけ順では，左部分木，根，右部分木の順に節点をなぞる必要がある．通りがけ順のなぞりを注意深く考察すると，左枝のなぞりは帰りがけ順，右枝のなぞりは行きがけ順のなぞりに類似している．この点を考慮して，走査リスト上の要素である各辺に先行辺か後退辺かの情報のほかに，その辺が左枝か右枝かの情報を付加する．その結果，通りがけ順に節点をなぞるアルゴリズムは次のようになる．

［**通りがけ順になぞる並列アルゴリズム**］

走査リストにおいて下記に述べる条件 a) から e) までを満たすサブリストを作り，そのリストの先頭から各要素すなわち辺の先頭の節点を順次なぞれば，通りがけ順になぞることができる．

このアルゴリズムでは，記述を簡潔にするため，走査リスト上のすべての辺に各プロセッサを割り当てる．

まず，次の条件 a)，b)，c) を満たす辺に目印を付加する．
a) 左枝の後退辺，**b)** 右枝の先行辺．**c)** 右枝の先行辺の直後の右枝の後退辺，または，左枝の後退辺の直後の右枝の後退辺．次に，

d) 上記の a)，b)，c) で目印を付加した辺のみからなるサブリストを作る．このとき，このサブリストの最後の辺 (i_n,j_n) が左枝の後退辺ならば，ダミー辺 (j_n,j_{n+1}) をこのサブリストの最後尾に追加する．

e) 条件 a)〜d) で生成されたサブリスト上で，右枝の後退辺の直後に辺があれば，その辺（左枝の後退辺）を削除する．

最後に，条件 e) で生成されたサブリストに対して，その各要素すなわち各辺の先頭の節点からなるリストを作り，並列リストランキングアルゴリズムを適用すれば，各節点の通りがけ順の値が求まる．

上記アルゴリズムを**図 3** に示す．図 3 では走査リストの最後尾は自分自身を指し示していると仮定し，走査リストの各要素 k には次のような配列を割り付ける．son(k) は左枝/右枝の識別，trav(k) は先行辺/後退辺の識別，reverse(k) は要素 k（辺 (i,j)）の逆順の辺 (j,i) へのポインタ，mark(k) は条件 a)，b)，c) を満たす辺の目印をそれぞれ格納する．また，head(L) はリスト L の先頭の要素を取り出す関数とし，next(k) は要素 k のポインタを示す．

［**アルゴリズムの正当性**］

はじめに，条件 a)，b) を満たす辺が通りがけ順になぞる辺の候補になることは自明である．また，条件 c) では，まず右枝の先行辺 (u,v) の直後の右枝の後退辺 (v,u) を通りがけ順になぞる辺に含めることで，右枝の葉節点 v をなぞることができる．次に，左枝の後退辺 (u,v) の直後の右枝の後退辺 (v,w) を通りがけ順になぞる辺に含めることによって，右枝を持たない節点 v をなぞることができる．

次に条件 d) では，条件 a)，b)，c) を満たす辺のみからなるサブリストを作る．このサブリストの最後の辺 (i_n,j_n) が後退辺でかつ左枝であれば，すなわち，根 j_n が右枝を持たないならば，新たにダミー辺 (j_n,j_{n+1}) を追加して根 j_n をなぞるようにする．

条件 e) では，条件 a) から d) までによって生成されたサブリスト上で，右枝の後退辺の直後に辺があれば，その辺は必ず左枝の後退辺である．なぜなら右枝は先行辺としてすでにそのサブリスト上に登録済みであり，次にくるのは左枝の後退辺しかありえない．このとき，この左枝の後退辺 (v,u) の節点 v は右枝を持つので，右枝の先行辺 (v,w) としてすでにサブリスト上に存在している．そのため，節点 v は 2 回なぞられることになる．したがって，右枝の後退辺の直後の左枝の後退辺はこのサブリスト上から削除する必要がある．

たとえば，**図 4** で条件 d) を満たすサブリストにおいて，

右枝 右枝 右枝 **左枝** 左枝
$\rightarrow(v_3,v_4)\rightarrow(v_4,v_5)\rightarrow(v_5,v_4)\rightarrow(\boldsymbol{v_3},\boldsymbol{v_2})\rightarrow(v_2,v_1)\rightarrow$
先行辺 先行辺 後退辺 **後退辺** 後退辺

右枝の後退辺 (v_5,v_4) の直後にくる左枝の後退辺 $(v_3,\boldsymbol{v_2})$ は，サブリストから削除する．その結果，条件 e) を満たすサブリストの先頭から各要素である辺の先頭の節点をなぞれば通りがけ順の値が得られる． □

図 1 の 2 分探索木について，通りがけ順のなぞりを示す．まず，走査リストを先行辺，後退辺ならびに左右の子を識別する記号を付記して表すと次のようになる．

```
const N = { 節点の総数 };
type cell = record  start,end:  1..N+1;
            next:  list  end;
     list = ↑cell;

procedure Para_Inorder_Traverse(traversal_list:  list);
{ 通りがけ順の値を算定する並列アルゴリズム． }
var rank:  array of integer;  k, sublist:  list;
begin  { 走査リストのランクを計算 }
   List_Ranking(traversal_list, rank);
   for all k, k ∈ traversal_list in parallel do
   begin  { 先行辺または後退辺の識別 }
      if rank(k) < rank(reverse(k)) then
         trav(k) ← advance
      else
         trav(k) ← retreat;
      mark(k) ← 0;
      if trav(k) = retreat and son(k) = left then
         mark(k) ← 1  { 条件 a) }
      else if trav(k) = advance and son(k) = right then
         mark(k) ← 2;  { 条件 b) }
      if mark(k) ≠ 0 and next(k) ≠ k then
         if trav(next(k)) = retreat and
            son(next(k)) = right then
               mark(next(k)) ← 3  { 条件 c) }
   end;
   sublist ← traversal_list;  { ここから条件 d) }
   repeat log 2(N − 1) times
   { 目印の辺のみのサブリストを生成 }
      for all k, k ∈ sublist in parallel do
         if mark(next(k)) = 0 then
            next(k) ← next(next(k));
   for all k, k ∈ sublist in parallel do begin
      if k=head(sublist) then  { サブリストの先頭の処理 }
         if mark(k) = 0 then
            head(sublist) ← next(head(sublist));
      if mark(next(k)) = 0 then
         next(k) ← k;  { サブリストの最後尾の処理 }
      if next(k) = k and mark(k) = 1 then
      begin  { 最後尾が左枝の後退辺のとき }
         new(next(k));  { 最後尾にダミー辺を追加 }
         next(next(k)) ← next(k);
         next(k)↑.start ← k↑.end; next(k)↑.end ← N+1
      end
   end;
   for all k, k ∈ sublist in parallel do
      if mark(k) = 3 and next(k) ≠ k then
         next(k) ← next(next(k));  { 条件 e) }
   List_Ranking(sublist, rank);  { 通りがけ順の値の算定 }
   for all k, k ∈ sublist in parallel do
      T[k↑.start].num ← rank(k)  { 値の格納 }
end;
```

図 3 通りがけ順になぞる並列アルゴリズム

Fig. 3 Parallel algorithm for numbering the vertices in inorder using the traversal list.

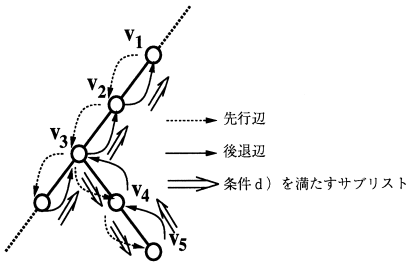

図 4 走査リストと条件 d) を満たすサブリスト
Fig. 4 Traversal list and sublist satisfied the condition d).

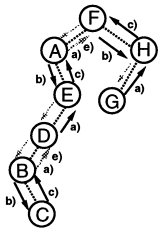

図 5 条件 a)〜e) を満たす図 1 のサブリスト
Fig. 5 Sublist satisfied conditions a)〜e) in Fig. 1.

$$\begin{array}{lllll} \underset{advance}{(F,A)} \rightarrow & \underset{advance}{(A,E)} \rightarrow & \underset{advance}{(E,D)} \rightarrow & \underset{advance}{(D,B)} \rightarrow & \\ \underset{advance}{(B,C)} \rightarrow & \underset{retreat}{(C,B)} \rightarrow & \underset{retreat}{(B,D)} \rightarrow & \underset{retreat}{(D,E)} \rightarrow & \underset{retreat}{(E,A)} \rightarrow \\ \underset{retreat}{(A,F)} \rightarrow & \underset{advance}{(F,H)} \rightarrow & \underset{advance}{(H,G)} \rightarrow & \underset{retreat}{(G,H)} \rightarrow & \underset{retreat}{(H,F)} \end{array}$$

上記の走査リストから，条件 a)，b)，c) および d) を満たすサブリストは次のようになる．

$$\begin{array}{lllll} \underset{advance}{(A,E)} \rightarrow & \underset{advance}{(B,C)} \rightarrow & \underset{retreat}{(C,B)} \rightarrow & \underset{retreat}{(B,D)} \rightarrow & \underset{retreat}{(D,E)} \rightarrow \\ \underset{retreat}{(E,A)} \rightarrow & \underset{retreat}{(A,F)} \rightarrow & \underset{advance}{(F,H)} \rightarrow & \underset{retreat}{(G,H)} \rightarrow & \underset{retreat}{(H,F)} \end{array}$$

次に，e) の条件によって，(B,D) と (A,F) の辺が削除され，サブリストは次のようになる．

$(A,E) \longrightarrow (B,C) \longrightarrow (C,B) \longrightarrow (D,E) \longrightarrow$
$(E,A) \longrightarrow (F,H) \longrightarrow (G,H) \longrightarrow (H,F)$

上記のサブリストは適用された条件を付記した矢印つき実線で**図 5** に示される．

このサブリストの各要素すなわち辺の先頭の節点をなぞれば通りがけ順が次のように得られ，さらに並列リストランキングを適用して通りがけ順の値が求まる．

$$A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F \rightarrow G \rightarrow H$$

［通りがけ順になぞる並列アルゴリズムの時間量］

節点数 N の 2 分探索木を表す隣接リストからオイラー閉路を求めるアルゴリズムは $O(\log N)$ の時間を要するが，2 分探索木の場合には隣接リストの長さはたかだか 3 であるので，プロセッサ数 N を用いれば定数時間 $O(1)$ で算定できる．また走査リスト上でのランクの算定および先行辺および後退辺の識別に用いたダブリング技法は $O(\log N)$ 時間を要する．

次に通りがけ順の値を求めるアルゴリズムは，リストの各要素にプロセッサを割り当てると，各プロセッサは自分が先行辺であるか後退辺であるか，また右枝であるか左枝であるかの情報に基づき，a)，b)，c) の条件に合否するかどうかを $O(1)$ 時間で判別できる．条件 d) におけるマークされた要素の部分リストの生成は $O(\log N)$ で構成することができる．また条件 e) は $O(1)$ 時間でできる．さらに，生成したサブリストのランクは $O(\log N)$ 時間で求まる．前述の結果，通りがけ順の値を算定する並列アルゴリズムは $O(\log N)$ 時間で可能である．このアルゴリズムは，Brent の定理[1)]を用いればプロセッサ数を $O(N/\log N)$ に減少させることも容易である．

3. おわりに

通りがけ順の値を求めるためには，2 分探索木を通りがけ順になぞるという逐次性の強い処理が必要となる．この処理をどのように並列化するかが問題である．単に 2 分探索木の通りがけ順の値を求めるだけなら，見出しの値を並列に整列することによって求めることができる．しかし本稿では，オイラーツアー技法を用いて 2 分探索木を通りがけ順になぞる汎用性のある並列アルゴリズムを考察した．すなわち，提案した並列アルゴリズムでは，まず，オイラーツアー技法を用いて 2 分探索木のオイラー閉路を求め，その走査リストからリスト操作を行うだけで効率良く通りがけ順の値を算定する簡潔な並列アルゴリズムを示した．

参考文献

1) Gibbons, A. and Rytter, W.: *Efficient parallel algorithms*, pp.21–24, Cambridge University Press (1988).
2) 拓植宗俊，藤本典幸，萩原兼一：パイプライン化による平衡 2 分探索木に対する並列操作，信学技報，COMP94-60, pp.61–70 (Nov. 1994).
3) 拓植宗俊，藤本典幸，萩原兼一：平衡 2 分探索木に対する並列オンライン操作，電子情報通信学会論文誌，Vol.J80-D-I, No.7, pp.582–590 (1997).
4) Smith, J.R.: *The Disign and Analysis of Parallel Algorithms*, p.319, Oxford University Press (1993).
5) Tarjan, R.E. and Vishkin, U.: An Efficient Parallel Biconnectivity Algorithm, *SIAM Journal of Computing*, Nol.14, No.4, pp.862–874 (1985).

(平成 12 年 4 月 18 日受付)
(平成 12 年 9 月 7 日採録)