2-3木の高速並列マージ・アルゴリズム

柴山 悦哉（京都大学・数理解析研究所）

0. はじめに

本論では，2-3木（2-3 tree）で表現された長さmとn（$m \leq n$）の順序列（sorted list）を高々2m個のプロセッサを用いて，$O(\log n)$時間でマージするアルゴリズムを与える。この時間計算量を評価する際には，大小比較，プロセッサの割り当て，出力の2-3木の構築に要するすべての手間を考慮にいれた。また，このアルゴリズムの実行中に，複数のプロセッサが同一番地に同時にリード・アクセスすることはない。

ここではまず，2-3木の動的な挙動を自然に表現する並列計算モデルを導入する。このモデルは通信ラインを介して2-3木状に結合されたプロセッサよりなる。さらに，このモデルの通信ラインは動的に付け替えることが許される。この機能により，動的なデータ及び動的な制御が自然に表現できる。本報告で述べる並列マージ・アルゴリズムは，この計算モデル上で設計される。

このアルゴリズムと競合するものとしては，アレイで表現された順序列をマージするものがある。[ ]ところが出力としてアレイを用いるかぎり，マージアルゴリズムの時間計算量は$O(\frac{n+m}{p})$以下にはなりえない。ここでpは利用可能なプロセッサ数，nとmは2つの順序列の長さを表わす。一方，我々のアルゴリズムは高々2m個のプロセッサにより時間計算量$O(\log n)$を達成している。したがって，nが十分大きい時，アレイを出力するものより有利となる。

1. 2-3木

定義 1.1

2-3木とは次の条件を満たす木構造である[1]：

(1) 葉（leaf）以外の各節（node）がどれもまたは3つの子供を持つ

(2) 根（root）から葉へ至る道（path）の長さがすべて等しい。 ☒

定義 1.2

k個の葉を持つ2-3木Tを用いて，順序列$S=\{a_1, a_2, \cdots, a_k\}$（$a_1 < a_2 < \cdots < a_k$）を表現するには，次のようにすればよい。

(1) Tの葉に左から順番に$a_1, a_2, \cdots, a_k$を代入する。

(2) Tの葉以外の節Nが3つの子供を持つ時，Nには以下に定める4つの値$\min(N)$, $1\text{st}\max(N)$, $2\text{nd}\max(N)$及び$\max(N)$が代入される。またNが2つの子供を持つ時，Nには$\min(N)$, $1\text{st}\max(N)$, 及び$\max(N)$が代入される。

$\min(N)$＝"Nを根とする2-3木の葉に代入された最小の値"

$\max(N)$＝"Nを根とする2-3木の葉に代入された最大の値"

$1\text{st}\max(N)$＝"Nの1番目の子供を根とする2-3木の葉に代入された最大の値"

$2\text{nd}\max(N)$＝"Nの2番目の子供を根とする2-3木の葉に代入された最大の値" ☒

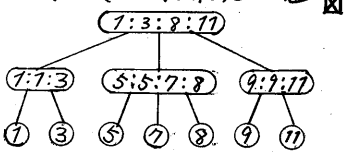


図 1.1 {1,3,5,7,8,9,11}を表現する2-3木

逐次的（sequential）なアルゴリズムを実行する際には，2-3木の葉以外の節Nに，$1\text{st}\max(N)$と$2\text{nd}\max(N)$を代入すれば十分である。[1]しかし，3章で述べる並列マージ・アルゴリズムを実行するためには，$\min(N)$及び$\max(N)$に相当する情報が必要になる。

順序列を2-3木で表現すると，探索

挿入，削除等の操作を高速に行なえる。[1]

定義 1.3

(1) 2-3木の節Nの高さ(height)とはNから葉へ至る道の長さのことである。ただし、葉の高さは0とする。

(2) 2-3木Tの高さとは、Tの根の高さのことである。Tの高さを|T|と記す。

(3) 2-3木の節Nの深さ(depth)とは根からNへ至る道の長さのことである。ただし、根の深さは0とする。 ☒

例えば図1.7の2-3木の高さは2である。

2. 並列計算モデル

この章では2-3木状に結合されたプロセッサで構成される並列計算モデルを定義する。

定義 2.1

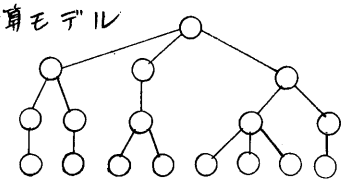
2-3計算モデルを構成するプロセッサは、次の条件を満たす木の形に通信ラインを介して結合される。

(1) 根のプロセッサは2つまたは3つの子供を持つ。

(2) 根でも葉でもないプロセッサは1つから3つの子供を持つ。ただし、あるプロセッサが1つしな子供を持たない時，このプロセッサの親は2つまたは3つの子供を持たなければいけない。

(3) 葉の深さはすべて等しい。 ☒

図2.1 2-3計算モデル



2-3計算モデルを構成するプロセッサ間の結合関係を動的に切り替えるために、次のような操作を定める。

操作1

葉を子供とするプロセッサがその一部の子供との間の通信ラインを切断する。尚，親とのリンクを切られたプロセッサは，以降活動を停止する。☒

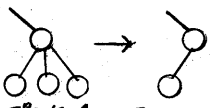


図2.2 操作1の例

操作2

(1) 根のプロセッサPE(r)の孫の個数が3以下の時、PE(r)は孫を直接の子供とし、今まで子供であったプロセッサを切り離す。

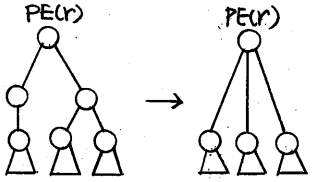


図2.3 操作2(1)の例

(2) 根のプロセッサPE(r)の子供の中に1つしか子供を持たないものが存在しても、PE(r)の孫の個数が4以上の時、次のような変形を行なう。まず，PE(r)は孫の一部をおじにあたるプロセッサの養子とし、PE(x)の各子供が0,2,または3個の子供を持つようにする。次に、PE(r)は子供を持たなくなったプロセッサを切り離す。

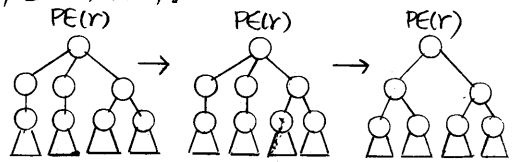


図2.4 操作2(2)の例

(3) 根以外のプロセッサPE(x)が2つまたは3つの子供を持ち，その中に，1つしな子供を持たないものが存在する時，次のような変形を行なう。まず，PE(x)は孫の一部をおじにあたるプロセッサの養子とし、PE(x)の各子供が0,2,または3個の子供を持つようにする。次に、PE(x)は子供を持たなくなったプロセッサを切り離す。 ☒

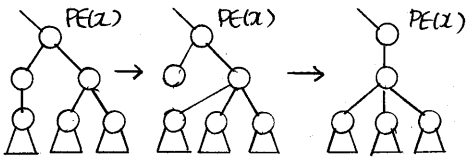


図2.5 操作2(3)の例

以下，操作2のことを平衡化操作と呼ぶ。あるプロセッサPE(x)が平衡化操作を行う時，PE(x)は自分自身と子供を結ぶ通信ラインおよび，自分の子供と孫を結ぶ通信ラインを付け替える。したがって、親子の関係にある2つのプロセッサが同時に平衡化操作を行なうことはできない。祖父と孫の関係にある2つのプロセッサに関しては，並列に平衡化操作を行なっても矛盾を生じることはない。図2.5において，PE(x)の親が1つしか子供を持たない場合，この変形は2-3計算モデルの条件に抵触する。この条件を保つためには，PE(x)の祖父が並列に平衡化操作を行なう必要がある。

定義2.2

2-3動的計算モデルとは，定義2.1の条件を満たし、葉以外の各プロセッサが操作1および平衡化操作を並列に行なえるようなものである。ただし，親子関係にある2つのプロセッサは、これらの操作を同時には行なえない。□

補題2.3

次の条件(1)を満たす2-3動的計算モデルの高さ偶数のプロセッサが、並列に平衡化操作を行なうと(2)の条件を満たすようになる。逆に、(2)の条件を満たす2-3動的計算モデルの高さ奇数のプロセッサが、並列に平衡化操作を行なうと(1)の条件を満たすようになる。

(1) 高さ偶数の葉以外のプロセッサが、2つまたは3つの子供を持ち，高さ奇数のプロセッサが1つから3つの子供を持つ。

(2) 高さ偶数の葉以外のプロセッサが、1つなら3つの子供を持ち，高さ奇数のプロセッサが2つまたは3つの子供を持つ。

[証明] 省略 □

次に例題として，複数の2-3木の連接(concatenation)を2-3動的計算モデル上で求める方法を示す。まず，若の定義を行う。

定義2.4

(1) TとT'をそれぞれNとN'を根とする2-3木とする。$max(N)<min(N')$が成り立つ時，T<T'と書く。

(2) TとT' (T<T') をそれぞれ$\{t_1,t_2,\cdots,t_k\}$と$\{t'_1,t'_2,\cdots,t'_l\}$を表現する2-3木とする。この時，TとT'の連接とは、$\{t_1,t_2,\cdots,t_k,t'_1,t'_2,\cdots,t'_l\}$を表現する2-3木である。□

2つの2-3木T<T'の連接は次の操作をくり返し行なうことによって求めることができる。簡単のため，各節の情報の更新に関する手続きは省略する。

操作3

(1) |T|=|T'|の場合

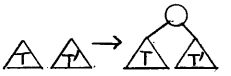


図2.6

図2.6のように新しい頂点を作り，TとT'をこの節の子供とする。

(2) |T|<|T'|の場合

T'の高さ|T|+1の節の中で，最も左側のものをNと名づける。Nが2つの子供を持てば，図2.7の変形を行なう。Nが3つの子供を持てば，図2.8の変形を行なう。

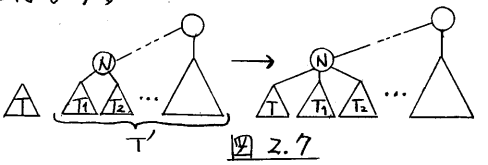


図2.7

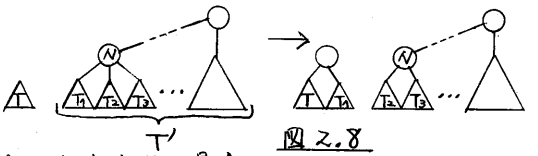


図2.8

(3) |T|>|T'|の場合

(2)と左右逆の操作を行なう。□

図2.8の場合，操作3をくり返し適用することにより，TとT'の連接を求めることができる。

[アルゴリズム] 2-3動的計算モデル上で，複数の2-3木の連接を求める．

図2.9において，各節は2-3木をあらわし、実線は通信ラインをあらわす

ものとする。また，三角形は2-3木をあらわし，破線で結ばれたプロセッサの管理化にあるものとする。さらに，各2-3木は，すべてのプロセッサに共有されたメモリ上に存在し，$T_1<T_2<\cdots<T_8$ を満たすものとする。

STEP 1: 高さ1のプロセッサが，2-3木を孫とみなして，平衡化操作を行なう。(図2.10)

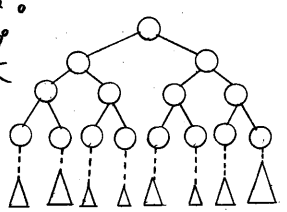


図2.9

STEP 2: 高さ2のプロセッサが平衡化操作を行なう。これと並列に，高さ0のプロセッサが操作3を行なう。(図2.11) ☒

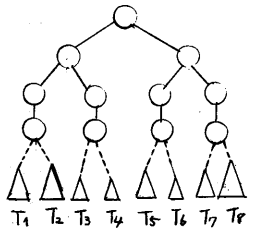


図2.10

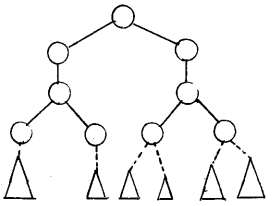


図2.11

STEP 2の終了後，高さが奇数のプロセッサによる平衡化操作と，高さが偶数のプロセッサによる平衡化操作及び操作3が交互に行なわれる。そして，葉のプロセッサの持つ2-3木が，唯一つしか存在しなくなった時，この手続きは終了する。一般に，複数の2-3木を連接するプログラムは，次のようになる。ただし，このプログラムの実行前に，葉以外の各プロセッサは2つまたは3つの子供を持ち，葉のプロセッサは，それぞれ1つの2-3木を持つものとする。

[連接を求めるプログラム]

while 葉の持つ2-3木の数が2以上 do
  step 1 高さ奇数のプロセッサが平衡化操作を行う。
  step 2 葉のプロセッサが操作3を行なう。葉以外の高さが偶数のプロセッサが平衡化操作を行なう。
end while ☒

定理2.5

合計n個の葉を持つm個の2-3木の連接は，高々2m個のプロセッサを用いて，$O(\log n)$ 時間で求められる。

[証明の概略] 補題2.3より，連接を求めるプログラムのSTEP 2を実行する前には，葉のプロセッサが2つまたは3つの2-3木を持つ。よって，これらの2-3木に対し，操作3を適用することができる。ところが，葉のプロセッサの管理する2-3木の中で，最も低いものの高さは，STEP 2を一回実行すると，少なくとも一増加する。(ある葉のプロセッサが3つの2-3木を管理する時は，操作3を2回適用すればよい。)よって，STEP 2は高々 $\log_2 n$ 回しか実行されない。 ☒

このアルゴリズムの実行に際して，葉のプロセッサのみが，直接的に連接を求める作業を行なった。他のプロセッサは，子供に仕事の割り当てを行なった。

3. 並列マージ・アルゴリズム

この章では，2つの順序列 $A=\{a_1,a_2,\cdots,a_m\}$ $(a_1<a_2<\cdots<a_m)$ と $B=\{b_1,b_2,\cdots,b_n\}$ $(b_1<b_2<\cdots<b_n)$ を並列にマージするアルゴリズムを設計し解析する。簡単のため，AとBは同じ要素を含まず，$m\le n$ 及び $b_n<a_m$ が成りたつものとする。また，$\bar{A}$ と $\bar{B}$ でそれぞれAとBを表現する2-3木をあらわす。ここでは，処理のパイプライン化を図るために，2-3木の列で順序列を表現することを考える。

定義3.1

(1) 2-3木の列 $\{T_1,T_2,\cdots,T_k\}$ が $T_1<T_2<\cdots<T_k$ を満たす時，この列は $T_1,\cdots,T_k$ の連接と同じ順序集合を表現しているものと考える。

(2) $\bar{B}[u,v]$ で順序列 $\{b_j\in B \mid u<b_j<v\}$ を表現する，ある2-3木の列をあらわす。 ☒

一般に，$\bar{B}[u,v]$ は一意的には定まらない。後程，$\bar{B}[u,v]$ の求め方を示すが，

それまでは，$B[u,v]$ は $\{b_j \in B \mid u < b_j < v\}$ を表現する任意の2-3木の列を意味するものとする。

この章で導入するアルゴリズムは，まず"Bを分割して，m個の2-3木の列 $B[-\infty,a_1]$, $B[a_1,a_2]$, …, $B[a_{m-1},a_m]$ を作る。ただし，$-\infty$ はAもBに属するどんな要素よりも小さい値とする。次に，これらとAから $\{b_j \mid b_j < a_1\} \cup \{a_1\}$, $\{b_j \mid a_1 < b_j < a_2\} \cup \{a_2\}$, …, $\{b_j \mid a_{m-1} < b_j < a_m\} \cup \{a_m\}$ を表現するm個の2-3木を作る。そして，2章で述べたアルゴリズムにより，これらの連接を求めれば，マージ操作は完了する。

このアルゴリズムの実行前に，プロセッサはAと同じ形に並べられていると仮定する。まず，Aが図3.1のような2-3木である場合を考えてみよう。簡単のため，Aの各節はプロセッサをあらわし，節と節を結ぶ辺は，通信ラインをあらわすものとする。PE(1), PE(2), …, PE(8) はプロセッサの名前で，節の内部に書かれた値は対応するプロセッサのローカル・メモリに記憶されているものとする。この時，$B[-\infty,a_1]$, $B[a_1,a_2]$, …, $B[a_4,a_5]$ は次のようにして求めることができる。図3.2はデータの流れをあらわす。

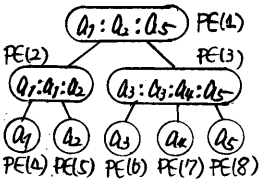


図3.1

[アルゴリズム]

STEP 1：PE(1)はBを分割し，$B[-\infty,a_2]$ および $B[a_2,a_5]$ を求める。そして，これらに属する2-3木の根をさすポインタを，それぞれPE(2)とPE(3)へ送る。

STEP 2：PE(2)は $B[-\infty,a_2]$ を分割し，$B[-\infty,a_1]$ と $B[a_1,a_2]$ を求める。そして，これらに属する2-3木の根をさすポインタを，それぞれ，PE(4)とPE(5)へ送る。並列に，PE(3)は $B[a_2,a_3]$, $B[a_3,a_4]$, $B[a_4,a_5]$ に属する2-3木の根をさすポインタを，それぞれ，PE(6), PE(7), PE(8)へ送る。 ☒

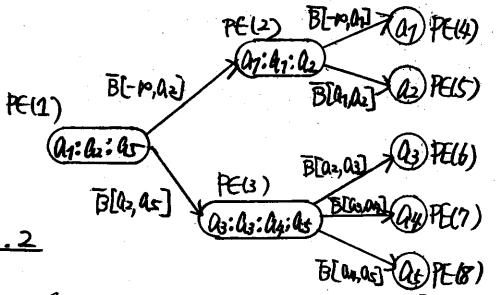


図3.2

この操作を，$A=\{a_1,a_2,\cdots,a_m\}$ の場合に一般化すると，次のプログラムのようになる。ただし簡単のため，Aの各節はちょうど2つの子供を持つと仮定する。また，プログラム中，PE(N)はAの節Nに対応するプロセッサであり，$pred(a_i)=a_{i-1}$ $(i \neq 2)$, $pred(a_2)=-\infty$ と定める。プロセッサ間の同期は，メッセージ・パッシングにより取られる。以下，"2-3木を送る"とは，その2-3木の根をさすポインタを送ることを意味するものとする。"2-3木を受け取る"についても同様に定める。

[Bを分割するプログラム]

if Nが根でない *then*
　PE(N)は親々から $B[pred(min(N)), max(N)]$ に属する2-3木を受け取る。
end if
if Nが葉でない *then*
　PE(N)は，それぞれの子供へ $B[pred(min(N)), lsemax(N)]$ と $B[lsemax(N), max(N)]$ に属する2-3木を送る。
end if ☒

上のプログラムにおいて，Aの根に対応するプロセッサ PE(N_r) は，Bを2つの2-3木の列に分割する。この作業は，AとBが図3.3のような2-3木である時，次のように行なわれる。

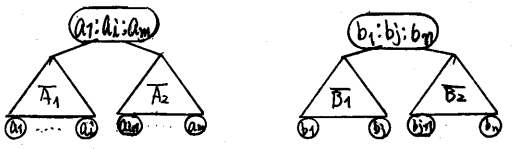


図3.3

PE(N)は，まずBの根を見に行く。こ

の時，次の4つの可能性が考えられる。
(1) $a_i < b_1$ (2) $b_1 < a_i < b_j$ (3) $b_j < a_i < b_n$ (4) $b_n < a_i$
(1)の場合，PE(N_r)はBを1番目の子供へ送る。(4)の場合，PE(N_r)はBを1番目の子供へ送る。(2)の場合，PE(N_r)はBを2番目の子供へ送り，再帰的にB_1の分割を行なう。(3)の場合，PE(N_r)はB_nを1番の子供へ送り，再帰的にBの分割を行なう。Bの根が3つの子供を持つ時も，ほぼ同様の操作が行なえる。この時，次の補題が成りたつ。

補題 3.2

Aの根をN_rとする。PE(N_r)は$B[-\infty, 1\text{stmax}(N)]$または，$B[1\text{stmax}(N), \max(N)]$に属する高さh ($0 < h < |B|$)の2-3木を，$O(|B|-h)$時間内に，それぞれの子供へ送ることができる。

[証明] hに関する帰納法で示せる。 ☒

一方，Aの根でも葉でもない節Nに対し，$B[\text{pred}(\min(N)), \max(N)] = \{T_1, T_2, \cdots, T_k\}$ ($T_1 < T_2 < \cdots < T_k$)とする。前員のプログラムによると，PE(N)は親から$T_1, T_2, \cdots, T_k$を受け取ってから，これらを分割し，その結果をそれぞれの子供へ送る。しかし，処理の高速化をはかるためには，受信と送信を交互に，くり返し行なった方がよい。例えば，PE(N)がT_ℓ ($1 \le \ell \le k$)を受け取ったとしよう。この時，T_ℓの根をN_ℓとすると，次の3つの可能性が考えられる。
(1) $1\text{semax}(N) < \min(N_\ell)$
(2) $\min(N_\ell) < 1\text{semax}(N) < \max(N_\ell)$
(3) $\max(N_\ell) < 1\text{semax}(N)$
(1)の場合，PE(N)はT_ℓを直ちに，2番目の子供へ送る。(3)の場合，PE(N)はT_ℓを直ちに，1番目の子供へ送る。(2)の場合，PE(N)はT_ℓを分割する。この分割は，Aの根に対応するプロセッサが，Bを分割する場合と同様に行なえる。ここで注意しておくが，PE(N)は$T_1, T_2, \cdots, T_k$を，この順番に受け取るわけではない。次の補題のように，高さの順番に受け取ると考えるとよい。また，(2)を満たす2-3木は，$\{T_1, T_2, \cdots, T_k\}$の中に高々1個しか存在しない。

補題 3.3

Aの根でも葉でもない1つの節Nの深さをdとする。PE(N)が，$O(|B|+d-h)$時間内に，$B[\text{pred}(\min(N)), \max(N)]$に属する高さh ($0 \le h \le |B|$)のすべての2-3木を受け取るなら，このプロセッサは，$B[\text{pred}(\min(N)), 1\text{semax}(N)]$または，$B[1\text{stmax}(N), \max(N)]$に属する高さhの2-3木を$O(|B|+d-h+1)$時間内に，それぞれの子供へ送ることができる。

[証明] $B[\text{pred}(\min(N)), 1\text{semax}(N)]$または，$B[1\text{stmax}(N), \max(N)]$に属する1つの2-3木をTとする。TがPE(N)の親から送られて来たものである場合，この補題は明らかに成りたつ。TがPE(N)の親から送られて来た2-3木T'を，分割してできたものである時，補題3.2と同様。 ☒

前員のプログラムをパイプライン化すると，次のようになる。ただし，$B[u,v,h]$とは$B[u,v]$に属す高さhの2-3木の集合である。既に述べたように，$B[u,v]$は一意的には定まらない。しかし，今まで述べてきた方法で，Bを分割して，$B[u,v]$を作ると，これは一意的に定まる。

[「Bを分割するプログラム(その2)」]

```
for h from |B| to 0 step -1
  if Nが根でない then
    PE(N)は親からB[pred(min(N)), max(N), h]を受け取る.
  end if
  if Nが葉でない then
    PE(N)はそれぞれの子供へB[pred(min(N))), 1semax(N), h]とB[1stmax(N), max(N), h]を送る
  end if
end for
```
☒

この時，次の定理が成りたつ。

定理 3.4

$B[-\infty, a_1]$, $B[a_1, a_2]$, …, $B[a_{m-1}, a_m]$ は前項のプログラムにより，$O(\log n)$ 時間で求めることができる。

[証明]

補題 3.2, 3.3 より，d に関する帰納で次のことが示せる。

※ A の根以外のある節 N の深さを d とすると，$FE(N)$ は $O(|B|+d-h)$ 時間内に $B[pred(min(N)), max(N)]$ に属する高さ h $(0 \le h \le |B|)$ の 2-3 木を，すべて受け取る。

したがって，葉のプロセッサは $O(|B|+|A|) = O(\log n + \log m) = O(\log n)$ 時間内に，すべての 2-3 木を受け取り終える。 ☒

さらに，次の補題が成りたつ。

補題 3.5

今まで述べてきた方法で求められた $B[pred(a_i), a_i]$ $(1 \le i \le m)$ を $T_1, T_2, \cdots, T_k$ とすると，ある k' $(1 \le k' \le k)$ が存在し，次の (1)(2)(3) が満たされる。

(1) $|T_1| \le |T_2| \le \cdots \le |T_{k'}| \ge \cdots \ge |T_{k-1}| \ge |T_k|$

(2) $T_1, T_2, \cdots, T_{k'}$ の中に，同じ高さの 2-3 木は高々2つしか存在しない。

(3) $T_{k'+1}, \cdots, T_k$ の中に，同じ高さの 2-3 木は高々2つしか存在しない。

[証明の概略]

(1)(2)(3) の条件を満たす 2-3 木の列を，今まで述べた方法で分割すると，(1)(2)(3) の条件を満たす 2-3 木の列が，2つできることを示せばよい。 ☒

補題 3.5 の条件 (1)(2)(3) を満たす，ある 2-3 木の列が，長さ n' の順序列を表現する時，この列に属する 2-3 木の連接は $O(\log n')$ 時間で求められる。[1] したがって，$\{b_j \mid b_j < a_1\} \cup \{a_1\}$, $\{b_j \mid a_1 < b_j < a_2\} \cup \{a_2\}$, …, $\{b_j \mid a_{m-1} < b_j < a_m\} \cup \{a_m\}$ を表現する m 個の 2-3 木は，$O(\log n)$ 時間で求めることができる。さらに，2章で導入したアルゴリズムを用いて，これらの連接を求めると，A と B のマージ操作の時間計算量は $O(\log n)$ となる。

さらに，次の定理が成りたつ。

定理 3.6

この章で述べたアルゴリズムを実行する際に，複数のプロセッサの同一番地に対する，同時アクセス（リード・コンフリクト）は生じない。

[証明の概略]

2-3 木の連接を求める際には，明らかに，リード・コンフリクトは生じない。一方，2-3 木の列の分割を求める時には，次のことが成りたつ。

(1) 根以外の各プロセッサは，親から送られてきた 2-3 木の節のみにアクセスする。

(2) 葉以外の各プロセッサは，ある 2-3 木を子供へ送った後，その 2-3 木の節にアクセスすることはない。

(3) 葉以外のプロセッサが，各子供へ送る 2-3 木の列は，同じ節を含まない。

(1)(2) より，2つのプロセッサが，先祖と子孫の関係にある時，これらの間にリード・コンフリクトは生じない。一方 (3) より，2つのプロセッサが，先祖と子孫の関係にない時も，これらの間にリード・コンフリクトは生じない。

☒

4. 付録

2-3 動的計算モデル上では，結合律を満たす演算の列に対し，動的なスケジューリングを行なうことができる。

[例題] 2-3 動的計算モデル上で，$3+1+5+6+8+7+2+11+4$ を計算する。

図 4.7 において，各節はプロセッサ，辺は通信ラインをあらわし，数字は対応するプロセッサのローカル・メモリに貯えられた値を示す。ここでは，小数と整数の加算は，整数同志の加算に比べ，約10倍の時間を要するものとする。この問題は次のようにして解け

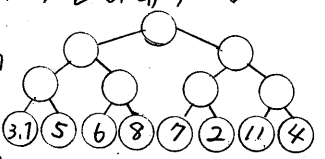


図 4.7

STEP1: 高さ1のプロセッサは，2番目の子供の持つ値を受け取り，それを1番目の子供へ伝える。そして，2番目の子供を切り離す。（図4.2）

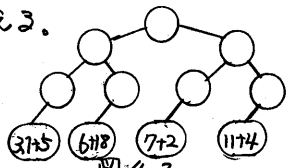


図4.2

STEP2: 葉のプロセッサは，加算を行なう。これと並列に，高さ2のプロセッサが平衡化操作を行なう。（図4.3）

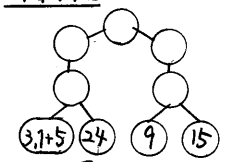


図4.3

STEP3: 1番左の葉のプロセッサは，引き続き加算を行なう。高さ1のプロセッサのうち，右側のものは，STEP1と同様の作業を行なう。（図4.4）

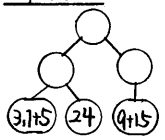


図4.4

以下，同様に図4.5から4.10までの変形を行なう。図

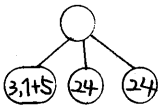


図4.5

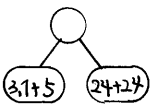


図4.6

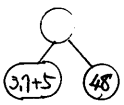


図4.7

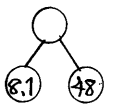


図4.8

図4.9

図4.10

図4.7において，解くべき問題は，((3,1+5)+(6+18))+((7+2)+(11+4))とみなされている。ところが，最終的には，(3,1+5)+((6+18)+((7+2)+(11+4)))という形で問題は解かれた。平衡化操作及び，プロセッサ間の通信に要する手間を無視すれば，明らかに後者の方が効率的である。

謝辞

本研究に対し，有益な助言を与えて下さった，中島玲二助教授（京大・数理解析研究所）及び，本論の草稿を通読し，適切なる指摘を行ってくれた，萩野達也，秋山豊行，桜川貴司の3氏に深く感謝いたします。

[参考文献]

1. AHO, A.V., HOPCROFT, J.E., and ULLMAN, J.D. The Design and Analysis of Computer Algorithms. Addison-Wesley, (1974)

2. BROWN, M.R. and TARJAN, R.E.: A Fast Merging Algorithm. JACM, Vol. 26(1979), No. 2, pp. 211-226.

3. DEKEL, D. and SAHNI, S.: Binary Trees and Parallel Scheduling Algorithms. Lecture Notes on Computer Science, Springer-Verlag, Vol. 111 pp. 480-492.

4. GAVRIL, F.: Merging with Parallel Processors. CACM, Vol. 18(1975) No. 10, pp. 588-591.

5. HIRSCHBERG, D.S.: Fast Parallel Sorting Algorithms. CACM, Vol. 21(1978), No. 8, pp. 657-661.

6. HWANG, F.K. and LIN, S.A.: A Simple Algorithm for Merging Two Disjoint Linearly Ordered Sets. SIAM J. on Computing, Vol. 1(1972) No. 1, pp. 31-39.

7. PREPARATA, F.P.: New Parallel Sorting Schemes. IEEE Trans. on Computers, C-27 No. 7 (1978) pp. 669-673.

8. SHILOACH, Y. and VISHKIN, U.: Finding the Maximum, Merging, and Sorting in a Parallel Computation Model. J. of Algorithms, Vol. 1 (1981), No. 2, pp. 81-102.

9. VALIANT, L.G.: Parallelism in Comparison Problem. SIAM J. on Computing, Vol. 4(1975) No. 3, pp. 348-355.