日本ソフトウェア科学会第 26 回大会 (2009 年度) 講演論文集

自己調整二分木の並列操作

上田 和紀

Sleator と Tarjan による自己調整二分木 (splay tree) に対して並列操作を可能にする操作アルゴリズムを提案する．提案するアルゴリズムは，同一の木に対する複数の更新・挿入・削除操作のパイプライン的並列実行を許し，かつ操作系列のスループット (単位時間内に処理可能な操作の個数) とレスポンス (個々の操作の償却計算量 (amortized complexity)) を両立させることを目的としている．スループットの最適性と挿入操作の対数的レスポンスについては理論的結果を示す．削除操作は，木の形状に関する良い性質を保つにもかかわらず，Sleator らの枠組みでは最適性が証明できない．このことについても論じる．

1 はじめに

自己調整二分木 (スプレー木, splay tree) [2] は，アクセスした節点に対して扁平化 (splaying) 操作 (2.1 節) を施すことにより，木の形状を動的に最適化する二分探索木の総称であり，多くの強力な性質が成り立つことがわかっている．本論文では，同一のスプレー木に対する複数の挿入削除等の操作のパイプライン的並列実行を可能にする方法を検討する．目標は，下記の要請を満たす操作アルゴリズムを得ることである．

1. (**レスポンス**) 通常のスプレー木の操作と同様，対数的な償却計算量 (amortized complexity) [4] をもつ．
2. (**スループット**) 操作後の木の形状が，根に近い部分から葉に向かって漸増的に確定するようにすることで，個々の操作が同時に施錠しなければならない節点の数を高々O(1) 個におさえる．

もしスループットだけが目標ならば，二分木を用いなくても，線形リストを用いて容易に達成できる．したがって，レスポンスとスループットを同時に達成することが本質的に重要である．B 木やその変種に対する並列操作の研究は少なくない [1] が，スプレー木の並列性に関する研究は少なく，著者の知る限り，上記の二条件を満たす並列アルゴリズムはまだ提案されていない．

本論文では，二分探索木の各節点はキーと値の対を保持するものとし，節点はキーの対称順 (symmetric order) に並んでいるとする．基本操作として，次の二つを考える．単なる節点値の読出しは *update* の単純な変種と考えることができる．

update(i, v, v', t): キー i をもつ節点が木 t の中にあれば，その節点の現在の値を v に代入したあと，節点に新たな値 v' を格納する．なければ，キー i と値 v' をもつ節点を t に挿入し，v には節点がなかったことを示す特別の値を代入する．

delete(i, v, t): キー i をもつ節点が木 t の中にあれば，その節点の現在の値を v に代入したあと，節点を消去する．なければ v に特別の値を代入する．

2 関連研究

2.1 扁平化とトップダウン扁平化

スプレー木における扁平化とは，節点の探索操作においてアクセスしたパスの長さをおよそ半分にし

Concurrent Operations on Splay Trees.

Kazunori Ueda, 早稲田大学理工学部情報学科, Dept. of Information and Computer Science, Waseda University,

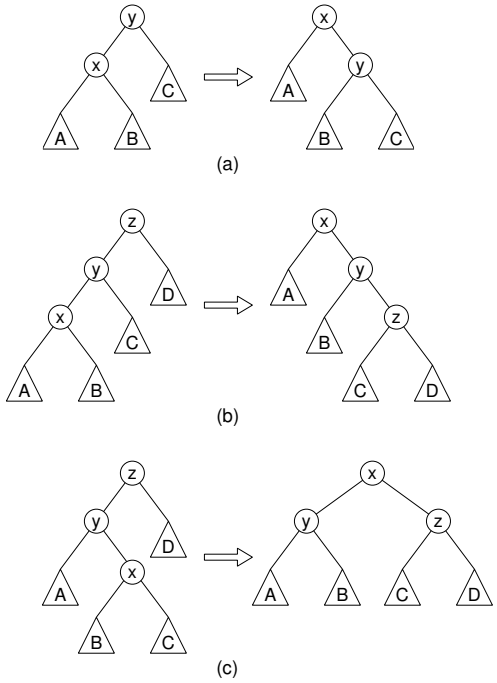

図 1　ボトムアップ扁平化操作の 1 ステップ． x **がアクセスした節点．(a) zig: 1 回の右回転 (**y **が根の場合のみ)，(b) zig-zig: 枝** yz **と枝** xy **をこの順に右回転，(c) zig-zag: 枝** xy **を左回転し，できた枝** xz **を右回転．**

つつ，目標節点 (*delete* においては，目標節点の直前または直後のキーをもつ節点) を木の根まで浮上させる操作である．扁平化は枝の回転 (rotation) を基本操作としており，図 1 に示す zig, zig-zig, zig-zag のうちの適切な操作をボトムアップに繰り返す．以下本論文では，左右対称な操作群はその片方のみを示す．また図中の小文字は節点，大文字は部分木を示す．*update*, *delete* 等の個別の操作アルゴリズムについては多くの変種がある．扁平化の大きな特徴は，アクセスしたパス上の各節点の深さを約半分にする一方で，アクセスしたパスの上にない節点を，高々O(1) 段しか深くしないことである．

扁平化はボトムアップな変形操作であるため，並列操作には適さない．文献 [2] はトップダウン扁平化も提案しているが，これは実装の容易化が主な目的であり，木の根は操作終了の直前まで確定しない．

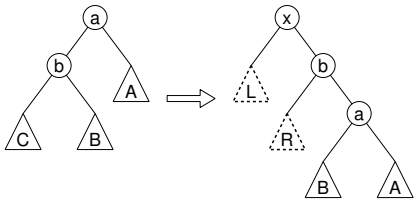

図 2　トップダウン扁平化による *update*

2.2　並列操作に関する過去の研究

和田 [5] は，並行論理型言語 [3] の論理変数を用いた扁平化アルゴリズムを提案している．これは，論理変数を利用して，トップダウン扁平化を in-place で行なうようにしたものと見なすこともできるが，*update* のように，対象となる節点が操作終了後の木に存在することがわかっている場合は，木の根のキーを操作の最初に確定させる点が大きな特徴である．しかしこの技法は，*delete* のように，操作結果の木の根が事前にわからない場合には適用できない．

2.3　トップダウン扁平化の問題点

トップダウン扁平化による *update* は，2.2 節のように根のキーを最初に確定させるようにしても，並列処理の観点からは問題が残る．たとえば，節点 $x(< b)$ ($<$ はキーによる順序関係) の *update* によって起きる図 2 の zig-zig 操作 [5] を考える (L と R は，木 C をトップダウン扁平化した結果の左 (右) 部分木で，*update* 完了時までに確定)．

この *update* の後，$y(< x)$, $z(> b)$ へのアクセスがこの順に続くとする．最初の x へのアクセス時に x が部分木 C の左の方にあったために zig-zig 操作が続く場合，L の根が確定するのは遅くなる．しかし L の根が確定するまでは，次の y へのアクセスが zig, zig-zig, zig-zag のどれをまず適用するか決められない．そこで 3 番目の z へのアクセスが，2 番目の操作によって影響を受けることのない b の右部分木に向かうにもかかわらず，長時間ブロックしてしまう．

削除操作はさらに問題である．一般に，二分木から節点 x を削除するには，x の左部分木の最大の節点 y を探してそれを x の場所に移すことが基本となる．しかし，扁平化の有無にかかわらず，y が見つかるま

では x の場所にくる新たなキーは確定せず，後続の操作をブロックしてしまう．以下のような解決法も考えられるが，いずれもうまく動作しない．

1. y が見つかるまで，x を一時的なキーとして利用すると，$y \leq z \leq x$ であるような節点 z への操作を誤った方向へ導く．
2. 各節点が直前と直後のキーをもつ節点へのポインタを保持することによって，x の直前の要素 y に O(1) 時間でアクセスできるようにすることが考えられる．これらのポインタは木の扁平化時に変更する必要がないという特徴がある．しかしこの方法は逐次操作のときしかうまく動作しない．なぜならこの削除操作の前の操作が x と y を結ぶパスを下降中で，いずれ y に到達するかもしれないからである．

したがって本論文では，高々O(1) 個の節を施錠しつつ，厳密にトップダウンに木を変形してゆくアルゴリズムを考えることとする．

3 並列更新アルゴリズム

本節では，後続の操作をブロックしない *update* 操作を与える．基本的なアイデアは，zig-zig と zig-zag の両方について，目標節点をその深さの半分までしか浮上させない半扁平化 (semi-splaying) を用いることである (文献 [2] の半扁平化は，zig-zig のみが扁平化と異なっていた)．x を更新対象の節点とすると，アルゴリズムは以下のようになる．

(a) 空の木に対する挿入は図 3 (a1) の操作，(空でない) 木の根に対する更新は図 3(a2) の操作を行なう．

(b) zig: x が左部分木の根である場合は図 3(b1) の操作，x が存在すべき左部分木が空の場合は図 3(b2) の操作を行なう．

(c) zig-zig: 図 3(c) 左の木における $x(< b)$ の探索では，枝 ba の右回転を行なってアクセスしたパスの長さを 1 短縮する．次は 1 レベル (短縮前の長さでは 2 レベル) 下降して，部分木 A に対して再帰的に探索を行なう．

(d) zig-zag: 図 3(d) 左の木における節点 x ($b < x < a$) の探索では，枝 cb の左回転と，できた枝 ca の右回転を行ない，アクセスしたパスを 1 短縮する．$x = c$ ならばこれで探索終了である．$x < c$ ならば 2 レベル (短縮前の長さでは 3 レベル) 下降して B の中から x を再帰的に探索する．$x > c$ ならば同様に C の中から再帰的に探索する．$x \geq c$ の場合には枝 ca の回転操作を省略することも考えられる．b の右部分木が空の場合は，そこに節点 x を挿入したあと，上に述べた回転操作を行なう．

以上の操作で，アクセスしたパスの長さは最悪でも約 2/3 になる．半分でなくて 2/3 なのは，上記 zig-zag 操作の性質によるものである．

4 並列削除アルゴリズム

並列削除のための基本的な着想は，扁平化操作を，削除すべき節点を下降させるために利用することである．これまでは，扁平化操作はもっぱら，再度アクセスしそうな節点を浮上させるために用いられてきた．ここで重要なことは，削除対象の節点以外は高々 O(1) レベルしか下降させないようにすることである．以下では，z を削除対象の節点とする．

まず，根節点が削除対象節点 z である場合を考える．この場合，zipping と呼ぶ操作によってそれを “容易に” 削除できる場所まで下降させる．節点が “容易に” 削除できるとは，その左部分木，右部分木，左部分木の右部分木，右部分木の左部分木のいずれかが空であることである．根節点の下降によって，その左部分木と右部分木の縫い合せが起きる．

(a) “容易に” 削除できる場合 : 図 4(a1) または (a2) のように変形する．

(b) “容易に” 削除できない場合 : 図 4(b) のように zig-zag を施し，その結果できる b の右部分木に，(一つめとは左右対称な) zig-zag を施す．

4 回の回転で z は 2 レベル下降する．z の新たな部分木 C と F は，同じレベルにとどまる．それ以外の節点も高々1 レベルしか下降しない．z を根とする新たな部分木に対して再帰的に削除操作を行なうが，z の子孫でない節点がそれによってさらに下降することはない．

図 5 に，根節点 z の削除による木の形状の変化を

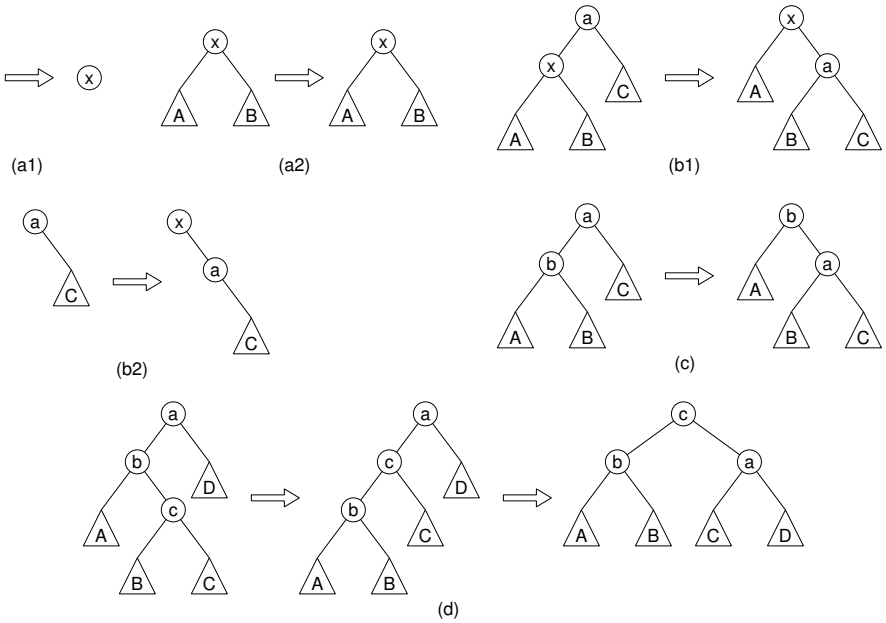

図 3　後続操作をブロックしない更新アルゴリズムの 1 ステップ

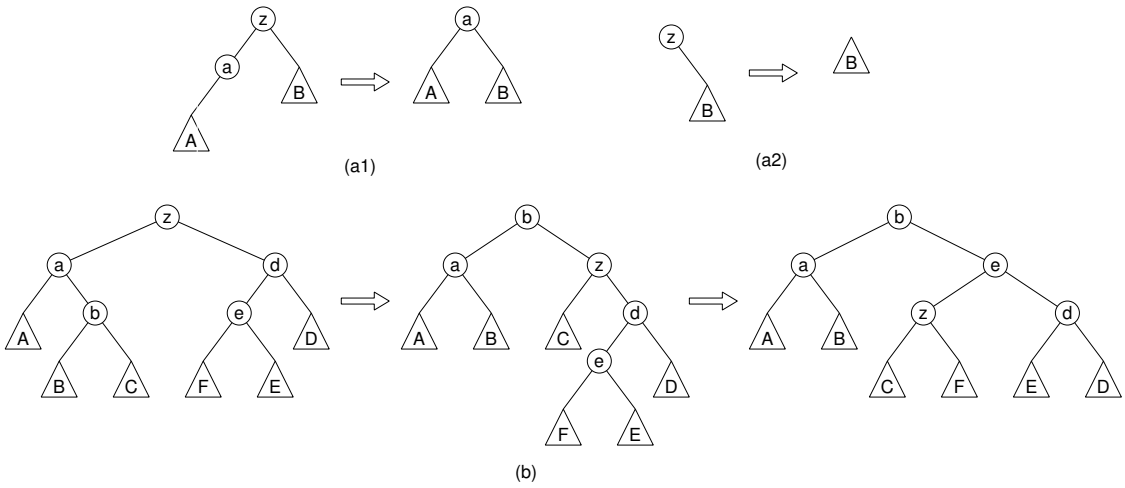

図 4　後続操作をブロックしない削除アルゴリズムの 1 ステップ

示す.

削除対象節点 z が根であるとは限らない場合は，まず第 3 節の方法で z を探索する．これは根から z に至るパスを短縮する効果をもつ．つぎに，z を zipping によって下降させて削除する.

Zipping 操作はパスの短縮を行なわないが，アクセスした節点は浮上させるという原則にしたがうならば，zipping に先だって，左部分木の最大要素に至るパスと右部分木の最小要素に至るパスをそれぞれトップダウンの半扁平化 (zig-zig (図 3(c)) の繰返し) によって短縮すればよい．この短縮化は zipping と並行して行なうことができる.

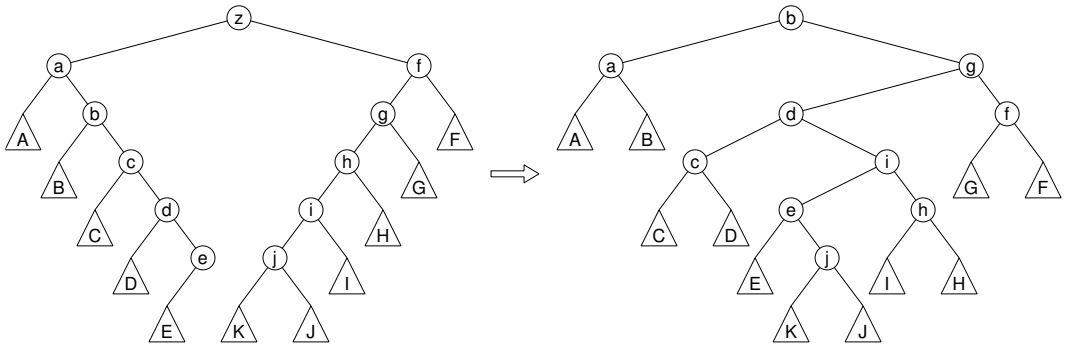

図 5　Zipping による節点 z の削除

Zipping は更新操作と異なり，各節点のキー値を読むことなく木を下降する．また zipping は，木 T_1 と木 T_2 (T_1 のどのキーも，T_2 のどのキーよりも小さいものとする) とのトップダウン併合操作にも応用できる．すなわち，新たな節点 (キーは任意) を調達し，その左部分木を T_1，右部分木を T_2 として一つの木を構成したのち，調達した根節点を消去すればよい．

5　計算量に関する結果と考察

効率の二つの尺度のうち，スループットについては容易に議論ができる．すなわち，二つの操作は，レベル l (根をレベル 0 として) の節点を $O(l)$ 回 — *update* は高々 $(l+2)$ 回，zipping は高々 $(2l+2)$ 回 — の回転操作ののちに確定させる．さらにどちらの操作も，連続する高々3 レベルの節点を同時に施錠するだけでよい．これらのことから，木の大きさや深さによらないスループットで，操作系列をパイプライン的に並列処理することができる．

レスポンスは，*update* については，通常のスプレー木と同等の償却計算量をもつことが証明できる．具体的には，節点 x の**大きさ** $s(x)$ を x を根とする部分木の節点数と定義し，**ランク** $r(x)$ を $\log_2(s(x))$ とする．そして木の**ポテンシャル**を，すべての節点のランクの和と定義する．すると，*update* の償却時間，つまり回転操作の回数で測った所要時間に操作前後のポテンシャルの変化を加えたものは，n を木の節点数として，$O(\log n)$ であることを示すことができる．このことから，十分長い操作系列の平均レスポンスは，最悪でも対数的であることがわかる．文献[2] のように，節点に異なる重みをつけて s や r を定義することにより，より強い性質を示すこともできるが，本論文では省く．

一方，*delete* については，文献[2] の解析方法では，対数的償却計算量を導くことはできない．そのことを示すために，図 4 (b) の 4 回の回転によるポテンシャル変化を考える．

図 4(b) の一番右側の木のランク関数を r' とする．一番左側の木からのポテンシャルの変化を，k をある正定数として $k(r'(b)-r'(z))$ 以内に押さえることができることを示すのが，文献[2] における償却計算量の証明技法の基本であった．しかし，これらの木について $s(A)=s(B)=s(C)=h \gg t=s(D)=s(E)=s(F)$ を仮定すると，ポテンシャル変化が h/t に関して $O(\log(h/t))$ となる．一方 $r'(b)-r'(z)$ は h/t に関して $O(1)$ であるので，上記の要請を満たす k は存在しないことがわかる．Zipping に先立ってパス短縮化を行なった場合についても，同様のことが示せる．

しかし，第 4 節の削除操作は，アクセスしたパス上の節点の深さが約半分になり (事前にパス短縮化を施した場合)，それ以外の節点も高々定数レベルしか沈まないという，節点の浮き沈みについてのスプレー木一般の性質は満たしている．では一般に，この二つの性質を満たす自己調整的な木アルゴリズムで，平均レスポンスが対数時間で押さえられないような，十分長い操作系列は存在するのだろうか？ これは未解

決であるが，本論文で提案した二操作については，平均レスポンスは少なくとも $O(\sqrt{n})$ (更新のみならば $O(\log n)$) と予想される．

その根拠として，各節点の削除しやすさの変化を考える．節点 x の**削除困難度** $d(x)$ を，x からその直前のキー x_- をもつ節点へ至るパス長 (x_- が存在しない場合や，x_- が x の子孫でない場合は 0 と定める) と直後のキー x_+ をもつ節に至るパス長の最小値と定めると，第 4 節の *delete* は，d の大きな節点の消去には時間がかかるものの，残った各節点の d を高々 $O(1)$ しか大きくしない．また第 3 節の *update* で新たに挿入した節点の d は 0 であり，*update* はすでに存在していた各節点の d も高々$O(1)$ しか大きくしない．(ボトムアップ扁平化における節点の d の増加は，定数で押えることができない．) これらのことから

1. 新たな節点の d の値が k まで成長するには，他の節点の $\Omega(k)$ 回の挿入削除が必要

であることがわかる．さらに

2. 二分木における各節点の d の総和は，木をトラバースしたときに通る枝の延べ本数を上回ることはないから $O(n)$

である．1. と 2. から，新たな節点の挿入と，d の大きな節点の消去が繰り返されるという最悪の操作系列を考えても，操作の平均の手間は $O(\sqrt{n})$ であり，実用上の効率は更新操作のみの場合とほとんど変わらないと予想される．

6 まとめと今後の課題

節点の浮き沈みに関する望ましい性質を保ち，かつ計算量の意味で最適なスループットをもつ自己調整二分木の並列操作 (更新，挿入，削除，併合) アルゴリズムを提案した．節点の更新や挿入に関しては対数的償却計算量を持つことが証明できており，さらにアクセスパターンの偏りや変化に対する追従性など，スプレー木の持つ強力かつ頑健な性質の多くを引き継いでいる．削除の償却計算量のより良い理論的限界を導く (またはその不存在を示す) ことは今後の課題である．また，アルゴリズムの実際的効率，並列分散環境での実装，応用の検討も今後の課題である．

謝辞 本論文の初期の版について議論していただいた Robert Tarjan 氏 (Princeton 大)，毛受哲氏 (NEC)，中谷祐介氏 (早稲田大) に感謝する．

参 考 文 献

[1] Lanin, V. and Shasha, D. : A Symmetric Concurrent B-Tree Algorithm, Proc. 1986 Fall Joint Computer Conference, IEEE, 1986, pp. 380–389.

[2] Sleator, D. D. and Tarjan, R. E. : Self-Adjusting Binary Search Trees, *J. ACM*, Vol. 32, No. 3 (1985), pp. 652–686.

[3] Shapiro E. : The Family of Concurrent Logic Programming Languages. *ACM Computing Surveys*, Vol. 21, No. 3 (1989), pp. 413–510.

[4] Tarjan, R. E. : Amortized Computational Complexity, *SIAM J. Alg. Disc. Math.*, Vol. 6, No. 2 (1985), pp. 306–318.

[5] 和田久美子:スプレイ木の並列データ探索, Proc. KL1 Programming Workshop '90, Tokyo, ICOT, 1990, pp. 42–49.

A 付録: LaTeX による論文作成のガイド

ここに，以前の `sample.tex` では，論文作成のガイドがあったが，その内容は `guide.tex` に移動した．