Synchronized Parallel Algorithms on RedBlack trees*

Xavier Messeguer and Borja Valles

Universitat Politècnica de Catalunya
Departament de Llenguatges i Sistemes Informàtics
Campus Nord–Mòdul C6
C/ Jordi Girona Salgado, 1–3
08034 Barcelona, Spain
Contact author: peypoch@lsi.upc.es

Abstract. We present an approach for designing synchronized parallel algorithms to update RedBlack trees. The resulting algorithms update k keys with k processors on trees of size n in time $O(\log n + \log k)$ which is very close to the optimal speedup of $O(\log n)$ (sequential time for one search or update). The algorithms are designed as a pipeline of waves of processors, which are created at the bottom of the tree and flow up to the root. The design is made following the E.W.Dijkstra approach by first choosing the invariant properties and then the rules to update the tree.

Keywords: Synchronized parallel algorithms, PRAM algorithms, RedBlack trees.

1 Introduction

The so called *Synchronized parallel algorithms* are those that manage data types in a synchronized manner (PRAM algorithms [Akl89]). They can be envisaged as many sequential algorithms running simultaneously and executing the same sentence at the same time. Therefore, it may happen that several processes read or write on the same memory location at the same time. Our goal is to avoid these concurrent accesses.

The first synchronized parallel algorithms on search trees were designed by W. Paul, U. Vishkin and H. Wagener for 2-3 trees in 1983 [PVW83]. They proved that the time needed to search or update k elements with k processors on a tree with n keys is $O(\log n + \log k)$ which is very close to the optimal speedup of $O(\log n)$.

They designed parallel algorithms to dynamically maintain a parallel dictionary working simultaneously with many keys. The algorithms first hang the keys from the leaves (*search phase*), and later rebalance the tree (*rebalancing phase*) using pipelines of processors. These pipelines can be envisaged intuitively in

* This work has been partially supported by ESPRIT LTR Project no. 20244 — ALCOM-IT and DGICYT under grant PB95-0787 (project KOALA).

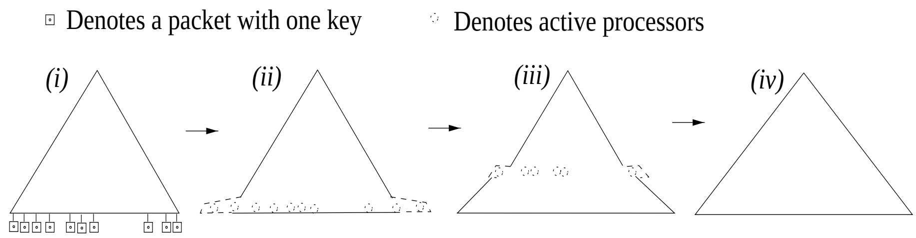

Fig. 1. Traveling wave for the basic insertion case on 2-3 trees

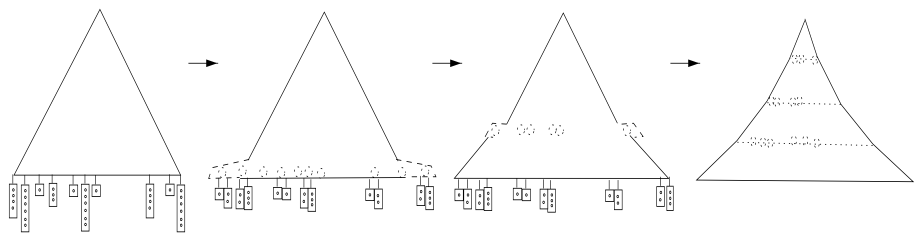

Fig. 2. Traveling waves for the general insertion case on 2-3 trees.

terms of traveling plane waves. Assume, for instance, the basic insertion case in which every leaf incorporates at most one new key (Figure 1.(i)). Something like a *wave* of processors is generated at the bottom of the tree, namely a *plane wave*, because all leaves of a 2-3 tree have the same depth (Figure 1.(ii)). This wave is sent up in further iterations (Figure 1(iii)) until it disappears (Figure 1.(iv)). Note that the wave goes to the root and at each iteration it strictly increases its height and decreases its depth. The *life-time* of each wave, i.e. the number of steps taken by a wave before it disappears, is an open problem, but some preliminary results [BYGM97] strongly suggest that it is logarithmic on k.

In the general insertion case (Figure 2), in which a packet of many new keys can hang from a single leaf, a pipeline of waves is generated to get something like harmonic traveling waves. Each new wave is created as follows: some iterations after the last wave has been created, the packets are split, the middle key of each one is attached as a new leaf and the remaining left and right subpackets are hung from the new leaf. This set of new leaves created by the middle keys constitute the new wave.

This rebalancing phase synchronizes the processors that belong to the same wave, and these processors locally manage the data and test the conditions to become inactive or to continue one step more. For this reason we say that processors are controlled by *Local Rules*. These are sequential algorithms composed by a small and fixed number of sentences that access a small number of neighbor nodes. The rebalancing phase can be written:

```
While there are active processors do
    For all waves do
        For all active processors of a wave do
            Select and apply rules
        endforall
    endforall
endwhile
```

These ideas were applied on B trees by L. Higham and E. Schenk [HS94], on Skip lists by J. Gabarró, C. Martínez and X. Messeguer [GMM96], and on AVL trees by J. Gabarró and X. Messeguer [GM96].

The RedBlack trees are an important basic data structure, namely a *balanced binary search tree*, which implements the *dictionary* abstract data type. The balancing criterion differentiates RedBlack trees from 2-3 trees, because it does not force the tree to be perfectly balanced: it is possible to deal with RedBlack trees whose leaves have significantly different depth. Therefore, it could be difficult to synchronize the processors of a wave because there is no obvious way to create plane waves.

We address in this paper the design of the synchronized insertion parallel algorithm on RedBlack trees with

- the same cost $O(\log n + \log k)$, and
- the *exclusive-read* and *exclusive-write* policy (EREW [Akl89]).

We omit the search phase of the update algorithms because it is well known (see previous references). We only design the rebalancing phase of this algorithm in which k keys are updated with k processors. The deletion algorithm can easily be designed using the same technique.

We prove the algorithm correctness following the approach developed by E.W.Dijkstra, in which the proofs are based on

- the preservation of some properties, called *invariants*, at each iteration, and
- the strict decreasement of a function, called *variant* function, at each iteration.

This approach, very common in basic sequential algorithmic courses, has not been applied yet on parallel algorithms on balanced search trees.

The rest of paper is organized as follows. Section 2 recalls RedBlack trees. Section 3 addresses the synchronized insertion algorithm. Finally section 4 shows the local rules of the algorithm.

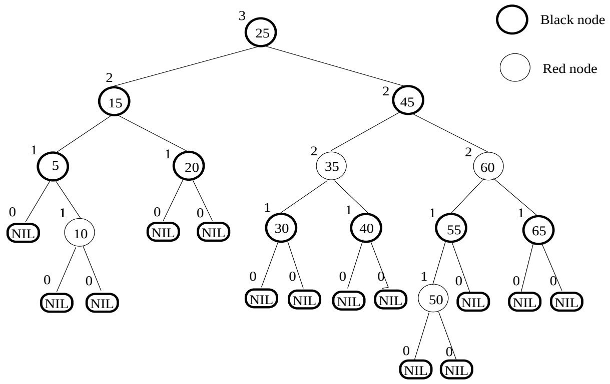

Fig. 3. An example of a RedBlack tree. The number drawn left-above the node denotes its black-height

2 RedBlack trees.

Following [CLR90], each node n of a RedBlack tree stores a key, denoted $\mathbf{key}(n)$, and each internal node has three pointers $\mathbf{left}(n)$, $\mathbf{right}(n)$ and $\mathbf{parent}(n)$ pointing respectively to its sons and parent (see Figure 3). A RedBlack tree satisfies the following properties:

P_1 : Every node is either red or black.
P_2 : Every leaf (NIL) is black.
P_3 : If a node is red then both its children are black. This is equivalent to, no path from the root to a leaf contains two consecutive red nodes.
P_4 : Every simple path from a node to a leaf contains the same number of black nodes.

The last condition P_4 allows the definition of the function called black-height in [CLR90]:

$$\begin{aligned}\mathbf{blackh}(n) = &\text{ the number of black nodes on any path from,}\\ &\text{ but not including, a node } n \text{ to a leaf.}\end{aligned}$$

We recall the sequential insertion algorithm:

1. *Search phase.* The key to be inserted falls until it is attached to a new red node n at the bottom of the tree. As this new node n is red, the property P_4 is maintained.

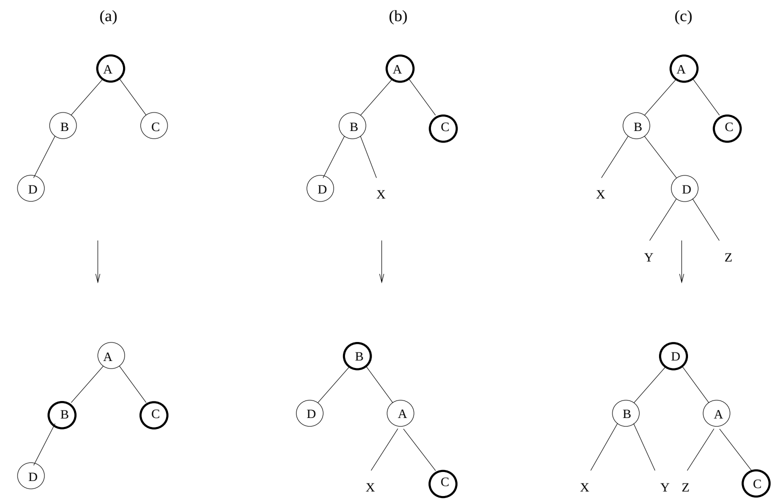

Fig. 4. The three basic local rules (under symmetry) of the sequential algorithm. The first rule (a) propagates up the redness and the following two rules, cases (b) and (c), rotate down the blackness.

2. *Rebalancing phase.* If the *parent* of n is black P_3 holds and the insertion is over. Otherwise, n and $\mathbf{parent}(n)$ are red and the bottom-up *rebalancing* phase really starts. By performing rotations and node recoloring, the redness of consecutive nodes disappears or rises up. Finally, if the root becomes red it is colored black. Figure 4 depicts the local rules applied in this phase.

3 Synchronized parallel insertion

Assume that the parallel search phase has ended and that the packets of keys hang from the leaves. We force each iteration of the rebalancing phase to hold the following invariants:

I_1: *Properties P_1, P_2 and P_4 of RedBlack trees.*
I_2: *Only those red nodes whose parent is also red have an active processor.* We identify the node with its processor, then we sometimes talk about "active nodes". Therefore, when there are no active nodes property P_3 holds, and by I_1 the tree is a RedBlack tree.
I_3: *All active processors of a wave have the same black-height,*

$$\forall p, q \in \text{wave} : \mathbf{blackh}(p) = \mathbf{blackh}(q).$$

This property allows us to define the black-height of a wave w:

$$\mathbf{blackh}(w) = \mathbf{blackh}(p) \text{ for any } p \text{ such that } p \in w.$$

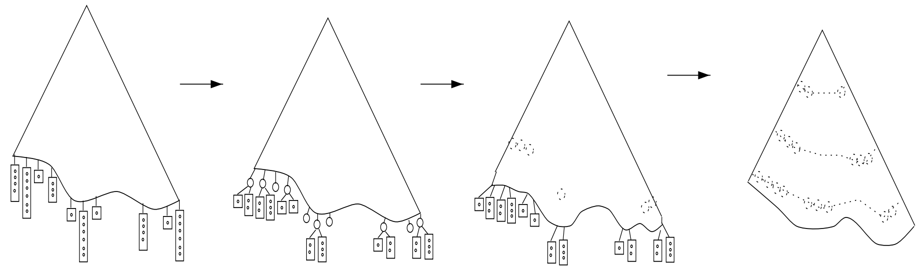

Fig. 5. Traveling waves for the general synchronized parallel insertion on RedBlack trees

I_4: *The black-height of the last created wave is at least two.* This property means that if the black-height of every wave gets increased by one unit at each iteration, then between two consecutive waves there is at least one black node. Therefore, if an active node has a grandparent gr, then gr is black.

The variant function involves the number of keys hanging at leaves, denoted $NKEYS$, and the sum of the depths of all existing waves, denoted $DEPTH$. Namely, it is defined by the ordered pair $(NKEYS, DEPTH)$. It strictly decreases at each iteration because new keys are attached to the tree, and when there are no keys, we force waves to strictly increase their black-height.

Each iteration is composed of two separate actions: (i) the creation of a new wave and (ii) the moving up of all waves.

(i) A wave is created by selecting the middle key of each packet and attaching it into a new red node, so I_1 holds. Each new red node n is controlled by an active processor p_n. Then active processors test their parents color and become inactive if it is black, so I_2 holds. As all nodes of the last created wave satisfy $\mathbf{blackh}(n) = 1$ (black leaves hang from them), I_3 holds.

(ii) Active processors run local rules which will be showed in the following section. We design them so they satisfy the the previous invariant and so they increase the black-height of all waves. Finally, we again update the active nodes.

The Figure 5 depicts the performance of the synchronized algorithm.

4 Local rules for insertion

Let us deal now with the rules we apply to make the waves go up. If there are active nodes without a grandparent, we simply turn the root black. For each active node n with a grandparent (that is black, by I_4) we consider the area

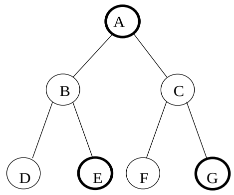

Fig. 6. Area defined by active nodes D and F

defined by its grandparent gp (node A in Figure 6), and the sons and grandsons of gp. In this area we can have active nodes other than n, but in any case they are all grandsons of gp and belong to the same wave, by I_4. Depending on the number of active nodes in the area we apply one rule or another.

If the grandparent of an area has only one active grandson we are in the same situation as the sequential case so we can try the same rules (see [CLR90]) and check if they satisfy the invariants. If the grandparent has more of one active grandson we are in a specifically parallel case so we need new rules. For every area in this situation we need to select one representative of its active nodes so we can apply the rules with only one processor. Note that counting the active nodes in an area and selecting a representative may lead us to a concurrent read situation. We avoid that possibility by just properly sequentializing that process.

In the sequential case we have three rules (see Figure 4): in (a) we move the wave up just by recoloring. Note that the number of black nodes of each path does not change but the variant function decreases, because the black-height of the wave (whose only node is now the grandparent) is one unit higher than before. In (b) and (c) we need both rotations and recoloring. The number of black nodes of each path does not change and the active nodes become inactive.

In the parallel case we find two new situations: if we have two active nodes that are brothers (Figure 7(a)) we need one rotation and recoloring; otherwise (Figure 7(b)) recoloring is enough, because both parents are red. Again the wave moves up one level without changing the number of nodes of any path.

Summing everything up, in all cases the active nodes of a wave move up one level (their black-height increases one unit) or they become inactive, which means that the variant function actually decreases and I_3 holds. The last created wave has now black height two (I_4). We also guarantee that every path from every node to a leaf has the same number of black nodes, so we preserve I_1. Finally, as we keep updating the active nodes, we also satisfy I_2.

References

[Akl89] S. Akl. *The design and analysis of parallel algorithms.* Prentice-Hall, 1989.

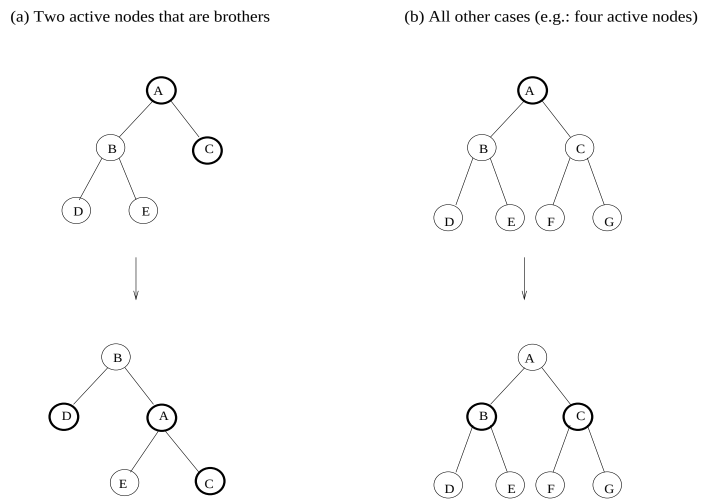

Fig. 7. The two basic new local rules (under symmetry) of the parallel algorithm

[BYGM97] R.A. Baeza-Yates, J. Gabarró, and X. Messeguer. Fringe analysis for parallel macrosplit insertion algorithm in 2–3 trees. Technical Report LSI-97-38-R, Universitat Politècnica de Catalunya. Dep. de Llenguatges i Sistemes Informàtics, 1997.

[CLR90] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* McGraw Hill, MIT, 1990.

[GM96] J. Gabarró and X. Messeguer. Massively parallel and distributed dictionaries on AVL trees and brother trees. In ISCA, editor, *Proc. of 9th International Conference on Parallel and Distributed Computing Systems*, pages 14–17, 1996. An extended version appeared as Technical Report LSI-96-27-R, LSI-UPC,1996.

[GMM96] J. Gabarró, C. Martínez, and X. Messeguer. A design of a parallel dictionary using skip lists. *Theoretical Computer Science*, (158):1–33, 1996.

[HS94] L. Higham and E. Schenks. Maintaining B-trees on an EREW PRAM. *J. of Parallel and Dist. Comp.*, 22:329–335, 1994.

[PVW83] W. Paul, U. Vishkin, and H. Wagener. Parallel dictionaries on 2–3 trees. In J. Díaz, editor, *Proc. 10th International Colloquium on Automata, Programming and Languages, LNCS 154*, pages 597–609. Springer-Verlag, 1983. Also appeared as "Parallel computation on 2–3 trees" in RAIRO Informatique Théorique, pages 397-404, 1983.