SOME COMMENTS ON BUILDING HEAPS IN PARALLEL

Carlo Luchetti - M.Cristina Pinotti

Nota interna B4-71

Dicembre 1992

COMMENT

SOME COMMENTS ON BUILDING HEAPS IN PARALLEL

Carlo LUCHETTI* and M.Cristina PINOTTI**

1. Introduction and background

A parallel work-optimal heap construction algorithm has been recently presented by Rao and Zhang [2]. However, as shown in the next section, there are some cases in which the algorithm does not produce the correct result. Here an amended version is proposed which builds a heap from a set of n elements in time O(n/p) using p processors, for $1 \leq p \leq n/\log n \log\log n$, on the EREW-PRAM model of computation. This algorithm is work-optimal for a range of processors smaller than other parallel makeheap presented in literature [1] but, it preserves the main feature, in our opinion, of algorithm [2], that is, different processors operate upon disjoint segments of the structure.

Recall that a heap H of n elements is a complete binary tree exhibiting the heap-shape property (i. e., all the leaves occur on the last two levels) and the max-ordering property (i.e., every node stores a value which is no smaller than the values stored in its children). H is implemented in situ by an array H[1..n], without additional pointers, with the root at position 1, and the left and right children of node in position i (briefly referred as *node i*) stored in position 2i and 2i+1, respectively [3]. The value of node i is indicated by H[i].

Finally, familiarity with [2] will be assumed.

2. The counterexample

Rao and Zhang [2] present two EREW-PRAM algorithms to build heaps: the not work-optimal *basic algorithm* [1] [2, Sect. 2] and the *optimal heap construction* [2, Sect. 3]. The latter is an incorrect extension of the basic algorithm as its second phase violates Lemma 2. Specifically, during the second phase a pair of alternating steps is not sufficient to guarantee that every leaf, when it is compared with the value in its list, contains the minimum among all the values on the path from the root to itself. Consequently, the complete execution of phase two does not ensure that the final value of every leaf be at least as large as any element in its list.

---

* Dipartimento di Informatica, Università di Pisa, Pisa, Italy

** Istituto di Elaborazione dell'Informazione, C.N.R., Pisa, Italy

[1] More precisely, Lemma 1 of [2] can be rewritten:
A heap of $n = 2^l - 1$ elements, for some l, can be constructed in at most (2l-3) steps of an EREW PRAM using (n+1)/2 processors.

In Figure 1, a counterexample is given: Figure 1(a) shows the state of a heap H at the beginning of phase two. After the first comparison leaves-lists, both values stored in the nodes 4 and 5 (Figure 1(b)) violate the max-ordering property. H[2] is exchanged with H[5] (Figure 1(c)) while H[4] remains in the leaf 4 disclaiming Lemma 2(i). During the subsequent comparison leaves-lists (Figure 1(d)), the list value '69' is rejected because it is compared with H[4] instead of the minimum on the path from the root to the node 4. The second phase is so completed without success (Figure 1(e)). Namely, the final value H[4] is no greater than or equal to all the elements in its list.

It can be easily proved that Lemma 2(i) holds if and only if *(k-1) pairs* of alternating steps are executed after every comparison leaves-lists, where k is the leaves level. The example in Fig. 2 shows that this condition is necessary.

Hence, the second phase terminates in $O(k \cdot \log n)$ instead of $O(k+\log n)$ as claimed in [2] and Theorem 3 in [2] is restated as:

Theorem 3. *A heap of a set S of n elements, chosen from a total order, can be constructed in $O((n \cdot \log p)/p)$ time on an EREW PRAM of p processors.*

3. Makeheap

Several revised versions of [2] are possible. However, none of them exhibits optimal work for $1 \le p \le n/\log n$. Here, we present a solution very close to the algorithm by Rao and Zhang.

The processors are essentially allocated to the data as in [2]. Some minor changes are carried out to satisfy the heap-shape property.

A heap is induced on the array H[1..n] using processors $P_1,\ldots,P_p$ for some $p=2^k-1$ with $1 \le k \le \lceil \log n \rceil$. A single item, referred as H[i], is assigned to P_i. Moreover, a list L_i of $(|H_i| - 1)$ items, where $|H_i|$ is the cardinality of the (implicit) binary tree rooted at node i at level k in H, is allocated to every *leaf processors* P_i (i.e., $2^{k-1} \le i \le 2^k-1$).

The algorithm operates in three phases, as in [2].

In phase one the basic algorithm is applied on $H[1 \ldots 2^k-1]$.

In phase two, every leaf processor, at first, selects locally the k biggest elements of L_i and puts them in the first k positions of L_i in non increasing order (i.e., $L_i[1] \le L_i[2] \le \ldots \le L_i[k]$).

Subsequently *k pairs* of alternating steps are carried out, i.e.,

the leaf processors, which operate synchronously with the processors on the levels k-2, k-4, ..., compare their value H[i] with $L_i[c]$, the current value in L_i, starting from $L_i[1]$. They execute the following algorithm:

if H[i] < L_i[c] **then begin**
exchange these two values;
c : c+1 {i.e., advance to the next element in the list}
end
else skip

In the third phase, each leaf processor builds a heap of the elements in its list.

The correctness of the algorithm follows immediately observing that:

a) the values in L_i can only replace the values on the path from the root to node i. Then, it is sufficient to know the k biggest elements in every list;

b) a leaf value is preceded by at least k elements in H, when it is exchanged with a list value;

c) after the comparison leaves-lists in the j-th pair of alternating steps, the values in the leaves are candidates to sift-up at most as far as the j-th level of H.

Finally, recalling that every list has cardinality O(n/p) [1] and that the selection can be executed in linear time; the three phases take respectively O(k), O(n/p +klogk + k), O(n/p) time where O(klogk) is the time spent to sort the first k elements of every list. The overall time complexity is:

$$O(n/p + \log p \log\log p)$$

i.e., the product time and processors number is optimal for $p = O(n/\log n \log\log n)$.

Acknowledgment

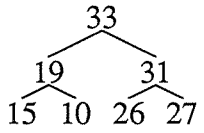
We thank Ferruccio Barsi, Linda Pagli and Geppino Pucci for helpful discussions concerning this note.

References

[1] S. Olariu and Z. Wen, Optimal Parallel Initialization Algorithms for a Class of Priority Queues, *IEEE Trans. Parallel and Distributed Systems*, **2** (1991), no. 4, 423-429.

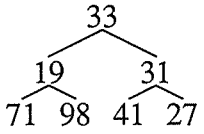
[2] N.S. Rao and W. Zhang, Building heaps in parallel, *Inform. Process. Lett.* **37** (1991) 355-358.

[3] R. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, PA, 1983.



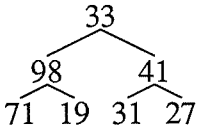
71 98 41 19
69 9 5 21

(a) H: at the beginning of the second phase



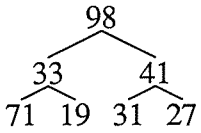
15 10 26 19
69 9 5 21

(b) Working at levels {3, 1}



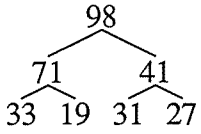
15 10 26 19
69 9 5 21

(c)Working at level {2}



15 10 26 19
69 9 5 21

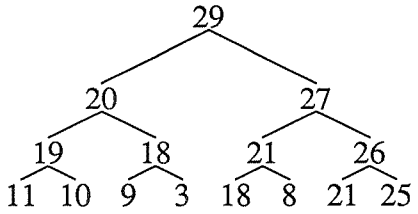
(d)Working at levels {1,3}



15 10 26 19
69 9 5 21

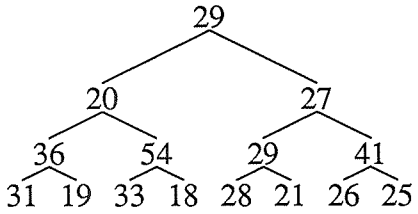
(e)The final state of H

Figure 1



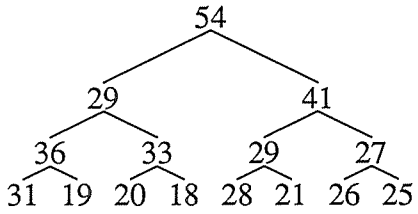
31 36 33 54 28 29 41 12
30 64 72 73 29 18 37 19

(a) At the beginning of the second phase



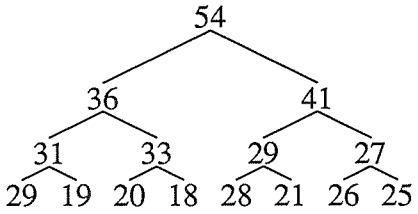
11 10 9 3 18 8 21 12
30 64 72 73 29 18 37 19

(b) After the first pair of alternating steps.
Lemma 2 (i) is violated.



11 10 9 3 18 8 21 12
30 64 72 73 29 18 37 19

(c) After the second pair of alternating steps (without working at level k).
Lemma 2 (i) is violated.



11 10 9 3 18 8 21 12
30 64 72 73 29 18 37 19

(d) After the third pair of alternating steps (without working at level k).
Lemma 2 (i) is verified.

Figure 2