

CS 467/567: Divide and Conquer on the PRAM

Stefan D. Bruda

Winter 2023



- **Problem:** Given a sequence $S_{1..n}$ sorted in increasing order and a value x , find the subscript k such that $S_k = x$
- If n processors are available the problem can be solved in constant time:
 - 1 All processors read x
 - 2 Each processor P_i compares x with S_i
 - 3 All processors P_i (if any) that found $x = S_i$ write j into k using min as combining operator
 - Good running time but far from optimal
 - Other CW models (Priority, Arbitrary, etc.) can also be used to break ties



BINARY SEARCH

- **Problem:** Given a sequence $S_{1..n}$ sorted in increasing order and a value x , find the subscript k such that $S_j = x$
- If n processors are available the problem can be solved in constant time:
 - ① All processors read x
 - ② Each processor P_i compares x with S_i
 - ③ All processors P_i (if any) that found $x = S_i$ write j into k using min as combining operator
 - Good running time but far from optimal
 - Other CW models (Priority, Arbitrary, etc.) can also be used to break ties
- Naïve divide and conquer approach for $N < n$ processors:
 - ① Divide the sequence S into N roughly equally sized subsequences (of length $O(n/N)$ each)
 - ② Each processor performs a sequential binary search to search for x in one subsequence
 - ③ Those procesors (if any) that found x write the respective index into k using min as combining operator
 - $O(\log(n/N))$ running time \rightarrow faster than the sequential algorithm but not considerably so



PARALLEL BINARY SEARCH

- N processors used to perform an $N + 1$ -way (rather than binary) search
- Sequence of stages; in the first stage all the sequence is under consideration, in subsequent stages only a subsequence will be under consideration
- At each stage the sequence under consideration is split into $N + 1$ subsequences
 - 1 Each processor P_i compares x with the elements s at the right boundary of the i -th subsequence
 - 2 If $x < s$ then all the elements in the $i + 1$ -st and higher subsequences can be discarded
 - 3 If $x > s$ then all the elements in the i -th and lower subsequences can be discarded
 - 4 If $x = s$ then the index has been found
- This process reduces the sequence under consideration N times rather than just halving it (like in the sequential case)
- The overall running time is thus $O(\log_N n)$



- **Problem:** Given two sequences of numbers (or more generally comparable values) $A = \langle a_1, a_2, \dots, a_r \rangle$ and $B = \langle b_1, b_2, \dots, b_s \rangle$ sorted in nondecreasing order, compute the sequence $C = \langle c_1, c_2, \dots, c_{r+s} \rangle$ such that each c_i belongs to either A or B , each a_i and b_j appear exactly once in C , and the sequence C is sorted in nondecreasing order

Algorithm RAM-MERGE(A, B) **returns** C :

- 1 $i \leftarrow 1, j \leftarrow 1$
 - 2 **for** $k = 1$ **to** $r + s$ **do**
 - 1 **if** $a_i < b_j$ **then** $c_k \leftarrow a_i, i \leftarrow i + 1$
 - 2 **else** $c_k \leftarrow b_j, j \leftarrow j + 1$
- $O(n)$ running time (optimal)
 - Requirements for the parallel algorithm:
 - Sublinear and adaptive number of processors
 - Running time substantially smaller than the sequential running time, and also adaptive
 - Optimal



PRAM MERGE

Assume (without loss of generality) that $r \leq s$

Algorithm PRAM-MERGE(A, B) **returns** C :

- Select $N - 1$ elements from A that divide A into N sequences of approximately equal size; call this sequence $A' = \langle a'_1, a'_2, \dots \rangle$. Similarly find the sequence $B' = \langle b'_1, b'_2, \dots \rangle$ that divide B into N sequences of roughly the same size (**constant time**):
 - 1 **for** $i = 1$ **do in parallel** $a'_i \leftarrow a_{i \lceil r/N \rceil}$, $b'_i \leftarrow b_{i \lceil s/N \rceil}$
- Merge A' and B' into a sequence of triples $V = \langle v_1, v_2, \dots, v_{2N-2} \rangle$, where each triple consists of an element of A' or B' , its position in A' or B' , and the name of the sequence of origin (A or B) (**$O(\log N)$ time**):
 - 1 **for** $i = 1$ **to** N **do in parallel**
 - 1 Processor P_i uses binary search on B' to find the smallest j such that $a'_i < b'_j$
 - 2 **if** j exists **then** $v_{i+j-1} \leftarrow (a'_i, i, A)$ **else** $v_{i+N-1} \leftarrow (a'_i, i, A)$
 - 2 **for** $i = 1$ **to** N **do in parallel**
 - 1 Processor P_i uses binary search on A' to find the smallest j such that $b'_i < a'_j$
 - 2 **if** j exists **then** $v_{i+j-1} \leftarrow (b'_i, i, B)$ **else** $v_{i+N-1} \leftarrow (b'_i, i, A)$



PRAM MERGE (CONT'D)

- Each processor merges and inserts into C the elements of two subsequences, one from A and one from B . The indices of the two elements (one in A and one in B) at which each processor begins merging are first computed and stored in an array Q of pairs ($O(r + s/N)$ time):

1 $Q_1 \leftarrow (1, 1)$

2 **for** $i = 2$ **to** N **do in parallel**

1 **if** $v_{2i-2} = (a'_k, k, A)$ **then** processor P_i

uses binary search on B to find the smallest j such that $b_j > a'_k$

$Q_i \leftarrow (k \lceil r/N \rceil, j)$

2 **else** processor P_i

uses binary search on A to find the smallest j such that $a_j > b'_k$

$Q_i \leftarrow (j, k \lceil s/N \rceil)$

3 **for** $i = 1$ **to** N **do in parallel**

1 Processor P_i uses RAM-MERGE and $Q_i = (x, y)$ to merge two subsequences beginning at a_x and b_y and places the result in C beginning at index $x + y - 1$. The merge continues until either

(a) an element larger than or equal to the first component of v_{2i} in each of A and B (when $i \leq N - 1$), or

(b) no elements are left in either A or B (when $i = N$)

Running time: $O(n/N + \log n) \rightarrow$ optimal algorithm for $N \leq n / \log n$



Algorithm CRCW-SORT($S_{1..n}$) returns $S'_{1..n}$:

- 1 **for $i = 1$ to n do in parallel**
 - for $j = 1$ to n do in parallel**
 - 1 **if $s_i > s_j \vee s_i = s_j \wedge i > j$**
 - 2 **then P_{ij} writes 1 in c_i using + as combining operation**
 - 3 **else P_{ij} writes 0 in c_i using + as combining operation**
 - 2 **for $i = 1$ to n do in parallel**
 - 1 P_{i1} stores S_i into S'_{1+c_i}
- Method called **enumeration** or **rank sorting**
 - Constant running time
 - $O(n^2)$ processors $\rightarrow O(n^2)$ cost (**not optimal**)
 - Likely of not a great practical value (number of processors very high)



- What about the CREW PRAM?
- Can still compare one pair of values (s_i, s_j) in each processor, but we cannot write all the results c_i in a single memory location



- What about the CREW PRAM?
- Can still compare one pair of values (s_i, s_j) in each processor, but we cannot write all the results c_i in a single memory location
- Solution:
 - ① If $s_i > s_j \vee s_i = s_j \wedge i > j$ then processor P_{ij} writes 1 into c_{ij} ; otherwise P_{ij} writes 0 into c_{ij}
 - ② Set c_i to $\sum_{j=1}^n c_{ij}$ then continue as in the CRCW algorithm
- Extra step: $c_i \leftarrow \sum_{j=1}^n c_{ij}$
 - Keep adding (in parallel) pairs of values until a single value remains
 - $O(\log n)$ time using n processors
- Overall running time: $O(\log n)$ using $O(n^2)$ processors



Algorithm CREW-SORT($S_{1..n}$) **returns** $S_{1..n}$:

- 1 Distribute equal size subsequences of S to the N processors. Each processor will then sort its subsequence sequentially ($O((n/N) \log(n/N))$ time)
 - 2 Keep merging pairwise adjacent subsequences (in parallel) until one sequence (of length n) is obtained (using PRAM-MERGE)
 - N/k subsequences (of length kn/N each) to merge in iteration k
 - Allocate $O(k)$ processor per pair of subsequences for each merge $\rightarrow O(n/N + \log(kn/N)) = O(n/N + \log n)$ time per iteration
 - $O(\log N)$ iterations $\rightarrow O((n/N) \log N + \log n \log N)$ overall time
-
- Running time: $O((n/N) \log n + \log^2 n)$
 - Cost: $O(n \log n + N \log^2 n) \rightarrow$ optimal for $N \leq n / \log n$



Algorithm CREW-SORT($S_{1..n}$) **returns** $S_{1..n}$:

- 1 Distribute equal size subsequences of S to the N processors. Each processor will then sort its subsequence sequentially ($O((n/N) \log(n/N))$ time)
 - 2 Keep merging pairwise adjacent subsequences (in parallel) until one sequence (of length n) is obtained (using PRAM-MERGE)
 - N/k subsequences (of length kn/N each) to merge in iteration k
 - Allocate $O(k)$ processor per pair of subsequences for each merge $\rightarrow O(n/N + \log(kn/N)) = O(n/N + \log n)$ time per iteration
 - $O(\log N)$ iterations $\rightarrow O((n/N) \log N + \log n \log N)$ overall time
- Running time: $O((n/N) \log n + \log^2 n)$
 - Cost: $O(n \log n + N \log^2 n) \rightarrow$ optimal for $N \leq n / \log n$
 - We can also sort faster ($O(\log n)$ time with $O(n)$ processors, still optimal), but such an algorithm does not scale well



Algorithm PRAM-CONVEX-HULL(n, Q):

- 1 Sort the points in Q according to their x coordinate
 - 2 Partition Q into $n^{1/2}$ sets $Q_1, Q_2, \dots, Q_{n^{1/2}}$ of $n^{1/2}$ points each such that the sets are separated by vertical lines and Q_i is to the left of Q_j iff $i < j$
 - 3 **for** $i = 1$ to $n^{1/2}$ **do in parallel**
 - 1 **if** $|Q_i| < 3$ **then** $CH(Q_i) \leftarrow Q_i$
 - 2 **else** $CH(Q_i) \leftarrow$ PRAM-CONVEX-HULL($n^{1/2}, Q_i$)
 - 4 **return** PRAM-MERGE-CH($CH(Q_1), CH(Q_2), \dots, CH(Q_{n^{1/2}})$)
- Let the algorithm use $O(n)$ processors
 - Step 1 doable in $O(\log n)$ time
 - Step 2 takes constant time (the sets Q_i are all subsequences of Q)
 - Step 4 takes $O(\log n)$ time
 - Overall the running time is $t(n) = t(n^{1/2}) + c \log n$ and so $t(n) = O(\log n)$
 - Therefore cost is $O(n \log n) \rightarrow$ **optimal** (non-output sensitive complexity)



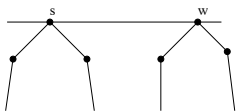
Algorithm PRAM-MERGE-CH($CH(Q_1), CH(Q_2), \dots, CH(Q_{n^{1/2}})$):

- Let u be the leftmost point of $CH(Q_1)$ and v the rightmost point of $CH(Q_{n^{1/2}})$
- Identify the upper hull:
 - 1 Assign $O(n^{1/2})$ processors to each $CH(Q_i)$
 - 2 Each processor assigned to $CH(Q_i)$ finds the upper tangent common between $CH(Q_i)$ and $CH(Q_j)$ for some $i \neq j$
 - 3 Between all common tangents between $CH(Q_i)$ and $CH(Q_j)$, $j < i$ let L_i (tangent with $CH(Q_i)$ at point l_i) be the tangent with the smallest slope
 - 4 Between all common tangents between $CH(Q_i)$ and $CH(Q_j)$, $j > i$ let R_i (tangent with $CH(Q_i)$ at point r_i) be the tangent with the smallest slope
 - 5 If the angle formed by L_i and R_i is smaller than 180 degrees then no points from $CH(Q_i)$ are in the upper hull; otherwise include in the upper hull all the points between l_i and r_i (inclusive)
 - 6 Identify the upper hull as all the points from u to r_1 , then all the points identified above, then all the points from $r_{n^{1/2}}$ to v (inclusive)
- Identify the lower hull (similar to the upper hull)
 - The lower hull is identified as above but this time u and v are excluded
- Return the union of the upper and lower hulls (array packing)

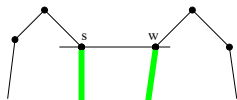
CONVEX HULL ON THE PRAM (CONT'D)

Computing the upper tangent of $CH(Q_i)$ and $CH(Q_j)$ in $O(\log n)$ time:

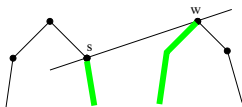
- Let s and w be the middle points in the (sorted) upper hulls from $CH(Q_i)$ and $CH(Q_j)$
- If \overline{sw} is the upper tangent of $CH(Q_i)$ and $CH(Q_j)$ then we are done (Case a)
- Otherwise repeat from Step 1 but excluding at least half the points of at least one upper hull (Cases b–h)



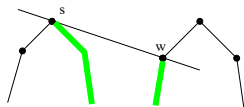
(a)



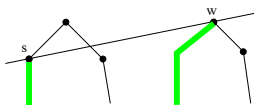
(b)



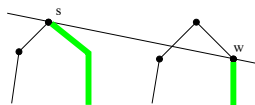
(c)



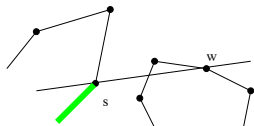
(d)



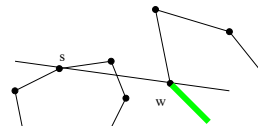
(e)



(f)



(g)



(h)