PRAM Divide and Conquer Algorithms

(Chapter Five)

Introduction:

- Really three fundamental operations:
    - *Divide* is the partitioning process
    - *Conquer* the the process of (eventually) solving the eventual base problems (without dividing).
    - *Combine* is the process of combining the solutions to the subproblems.
- Merge Sort Example
    - *Divide* repeatedly partitions sequence into halves.

- *Conquer* sorts the base sets of one element.
- *Combine* does most of the work. It repeatedly merges two sorted halves.

- Quicksort: The *divide* stage does most of the work.

Search Algorithms

- Usual Format: Have a file of n records. Each record has several *data* fields and a *key* field.
- Problem Statement: Let $S = \{s_1, s_2, \ldots, s_n\}$ be a sorted sequence of integers. Given an integer x, determine if $x = s_k$ for some k.
- Possibilities and actions:
  - **Case 1.** $x = s_k$ for some k.
    - Action: Return k.
  - **Case 2.** There is no k with $x = s_k$.
    - Action: Return
  - **Case 3.** There are several successive records, say $s_k, s_{k+1}, \ldots, s_{k+i}$, whose key field is x.
    - Action: Depends upon the application. Perhaps k is returned.
- **Recall:** Sequential Binary Search.
  - Key of middle record in file is compared to x.
  - If equal, procedure stops.
  - Otherwise, top or bottom half of the

file is discarded and search continues on other half.

- Searching using CRCW PRAM with n PEs.
  - One PE, say P_1, reads x and stores it in shared memory
  - All other PEs read x
  - Each processor P_i compares x to s_i for $1 \leq i \leq n$.
  - Those P_j (if any) for which $x = s_j$ use a min-CW to write j into k.
    - Can easily modify for PRIORITY or ARBITRARY, but not COMMON.
- Searching using PRAM and N PEs with $N < n$.
  - Each P_i is assigned the subsequence $s_{(i-1)\frac{n}{N}+1} \leq x \leq s_{i\frac{n}{N}}$
  - All PEs read x.
  - Any P_i with $s_{(i-1)\frac{n}{N}+1} \leq x \leq s_{i\frac{n}{N}}$ performs a binary search.
  - All P_i with a hit (if any) use MIN-CW

to write the index of its hit to k.

- **Problem:** Preceding algorithm is slow, as often all PEs but one are idle for most of the algorithm.

PRAM BINARY SEARCH

- Using N processors, we can extend the binary search to become an $(N + 1)$-way search.
- An increasing sequence is partitioned into $N + 1$ blocks and each PE compares a partition point s with the search value x.
- If $s > x$, then x can not occur to the right of s, so all elements following S are discarded.
- If $s < x$, then x can not occur to the left of s, so all elements preceding x are discarded.
- If $s = x$, then the index of s is returned.
- Diagram: (Figure 5.3, page 200)

drop.. s_1..drop.. s_2 ..keep.. s_3 ..drop.. s_4 ..drop...s

ptrs $\rightarrow$ $\uparrow$ $\uparrow$ $\uparrow$ $\uparrow$ $\uparrow$

P_1 P_2 P_3 P_4 P

- If x is not found, the search is narrowed to one block, identified by two successive pointers.
- This procedure continues recursively.
- Number of stages required:
  - Let m_t be the length of largest block at stage t.
  - The maximum length of blocks in stage 1 is

$$m_1 = \left\lceil \frac{n}{N+1} \right\rceil$$

  - The $(N+1)$ blocks of indices at stage 1 are

$[1, .., m_1], [m_1+1, .., 2m_1], .., [(N-1)m_1 + 1, .., Nm_1], [Nm_1+1, ..$

-
  - We can let P_i point to the value $i \cdot m_1$
  - Clearly $Nm_1 < n \le (N+1)m_1$ and $m_1 < \frac{n}{N}$ since n is in the (N+1)th

block.

- Similarly, $m_2 < \frac{m_1}{N}$ at stage 2, so $m_2 < \frac{n}{N^2}$.
- Inductively, $m_t = \frac{n}{N^t}$.
- Let g be the least integer t with $\frac{n}{N^t} \leq 1$.
- Then,

$$g = \left\lceil \frac{\lg n}{\lg N} \right\rceil = \Theta(\lg_N n)$$

- If n items are divided into $N + 1$ equal parts g successive times, then the maximum length of the remaining segment is 1.

• **Analysis of Algorithm:**

- The time for each stage is a constant.
- There are at most g iterations of this algorithm so

$$t(n) \in O[\lg_N(n)]$$

- The sequential binary search algorithm for this problem has a $O(\lg n)$ running time.

- To show optimality of the running time of this algorithm using this sequential time, we would need to show its running time is $O(\frac{\lg n}{N})$.
  - Trivial, if N is a constant.
  - Not obvious in general, as N is usually a function of n (e.g., $N = \sqrt{n}$).
- Instead, here optimality is established by a direct proof in the next lemma.
- Much better running time than previous naive parallel search algorithm with running time of

$$\lg\left(\frac{n}{N}\right) = \lg n - \lg N = \Theta(\lg n).$$

Lemma: As defined above, g is a lower bound for the running time of all PRAM comparison-based search algorithms.

- At the first comparison step, N processors can compare x to at most N elements of S.
- Note that $n - N$ elements are not checked, so one of the $N + 1$ groups created by the

partition by these N points has size at least $\lceil (n - N)/(N + 1) \rceil$.

- Moreover,

$$\left\lceil \frac{n - N}{N + 1} \right\rceil \geq \frac{n - N}{N + 1} = \frac{n + 1}{N + 1} - 1$$

- Then the largest unchecked group could hold the key and its size could be at least

$$m = \frac{n + 1}{N + 1} - 1.$$

- Repeating the above procedure again for a set of size at least m could not reduce the size of the maximal unchecked sequence to less than

$$\frac{m + 1}{N + 1} - 1 \geq \frac{n + 1}{(N + 1)^2} - 1.$$

- After t repetitions of this process, we can not reduce the length of the maximal unchecked sequence to less than

$$\frac{n + 1}{(N + 1)^t} - 1.$$

- Therefore, the number of iterations required by any parallel search algorithm is not less than the minimal value h of t with

$$(n+1)/(N+1)^t - 1 \leq 0$$

or, equivalently, h is the minimum t such that

$$\frac{n+1}{(N+1)^t} \leq 1$$

- So at least h iterations will be required by any parallel search algorithm, where

$$\lg(n+1) - h\lg(N+1) \leq \lg 1 = 0.$$

or

$$h \geq \frac{\lg(n+1)}{\lg(N+1)}.$$

- Recall that the running time of PRAM Binary Search is

$$g = \left\lceil \frac{\lg n}{\lg N} \right\rceil$$

  - ASIDE: It is pretty obvious that $h \leq g$ since h partitions into $N+1$groups each time, while g partitions into N groups each time (as rightmost

g –group could always have size 1).

- However, g and h have the same complexity, as

$$g \in \Theta(\frac{\lg n}{\lg N}) = \Theta(\frac{\lg(n+1)}{\lg(N+1)}) = \Theta(h)$$

- This can be formally by proving that

$$\lim_{n\to\infty} \left\{ \left[\frac{\lg n}{\lg N} \right] \Big/ \left[\frac{\lg(n+1)}{\lg(N+1)} \right] \right\} = 0$$

using L'Hospital's rule (assuming that $N = N(n)$ is a differentable function of n).