ICS 443: Parallel Algorithms Fall 2017

Lecture 9 — September 25, 2017

Prof. Nodari Sitchinava *Scribes: Isaac DeMello, Brandon Doan, Robert Figgs*

1 Overview

In the last lecture we detailed an algorithm that finds the minimum of an array in O($\log_2(\log_2(n))$) time, and with O($n(\log_2(\log_2(n))$) work.

In this lecture we define a work-efficient algorithm that finds the minimum in O($\log_2(\log_2(n))$) and with O(n) work. We also begin to look at searching algorithms. . . . .

2 Work Efficient Min

We begin by breaking down the algorithm into parts of size k, where k = O($\log_2(\log_2(n))$)

2.1 WE-CRCW-min

In order to break down the size of the array and apply the previously defined CRCW-min algorithm covered in lecture 9. We use the following algorithm:

Algorithm 1 WE-CRCW-min(A, n)

```
k = ⌈log_2(log_2(n))⌉
n' = ⌈n/k⌉
B = new array of size n'
for i = 1 to n' in parallel do
  B[i]=A[(i-1)k+1]
  for j = (i − 1)k + 2 to ik in parallel do
    B[i] = min(B[i], A[j])
  end for
end for
return CRCW-min(B, 1, n')
```

2.1.1 WE-CRCW-min Algorithmic Analysis

The first 3 lines of the algorithm can be done in constant time, this means that the runtime of the algorithm is dominated by the for-loop.

Work Analysis : Work of this algorithm can be shown as follows:

$\sum_{i=1}^{n} \sum_{j=(i-1)k+2}^{ik} O(1) = \theta(n' * k) = \theta((n/k) * k) = \theta(n)$

3 Searching

In this part of the lecture we look at how to both efficiently and inefficiently search through an array for a specified element.

3.1 Inefficient Search

Algorithm 2 Searching(A, n, x)

answer = -1
for $i = 1$ to n in parallel **do**
  if A[mid] ==x
  answer = i
end for
return answer

3.2 Recursive Search

While searching a sorted input, it is impossible to improve the run time of the search. The algorithm is as follows:

Algorithm 3 Binary-Search($A, left, r, x$)

mid = $\lfloor (l + r)/2 \rfloor$
if r ¡ l **then**
  return -1
end if
if $A[mid] == x$ **then**
  return x
end if
if x ¿ A[mid] **then**
  return return Binary-Search($A, l, mid + 1, r, x$)
else
  return Binary-Search($A, l, mid - 1, x$)
end if