Applied Algorithmics COMP526 – tutorial 10

L.A. Gasieniec and I. Lamprou

1 Questions

1.1 Parallel algorithms - revision

- Please make sure you understand duels in parallel computing. This includes string matching via witnesses, computing of maximum, and any other symmetric (order of arguments is irrelevant) binary operation.
- Have a good understanding of *mergesort* and *quicksort* algorithms. This includes sequential and parallel algorithms with their time and work complexity.

1.2 Parallel algorithms - longest sequence of zeros

Consider the problem of computing the longest sequence of consecutive 0s in a binary sequence $S[0,..,n-1]$ of length n. For example,

- Write a pseudocode of a sequential solution and comment on its correctness and complexity.
- Write a pseudocode of a parallel solution and comment on its correctness and complexity.

2 Solutions

2.1 Parallel algorithms - revision

See the relevant copies of slides and videos.

2.2 Parallel algorithms - longest sequence of zeros

Consider the problem of computing the longest sequence of consecutive 0s in a binary sequence $S[0,..,n-1]$ of length n. For example,

- *Write a pseudocode of a sequential solution and comment on its correctness and complexity.*

  The main idea is to swipe the input sequence S from the smallest index 0 to the largest $n-1$. The answer is stored in the variable *ans* which is originally set to 0 as if S contains only 1s we need to report value 0. In due course we keep info about the longest currently found sequence of 0s in *ans* but we also keep info on the length of the currently visited sequence of 0s in *temp*. When we discover the first 1 we check whether *ans* needs to be updated, i.e., when $ans < temp$, and we reduce the value of temp to 0. See below.

```
$ans, temp \leftarrow 0$;
(1) for ($i = 0; i < n; i{+}{+}$) {
(2)         if ($S[i] == 0$) {$temp{+}{+}$;}
(3)                 else {if ($temp > ans$) {$ans \leftarrow temp$;} $temp \leftarrow 0$; }
(4)         }
(5)  }
(6) if ($temp > ans$) {$ans \leftarrow temp$;}
(7) report($ans$);
```

  We need the last test in line (6) when S is concluded with a sequence of 0s which could be the longest. Since test each position in S exactly once the time complexity of the solution is $O(n)$.

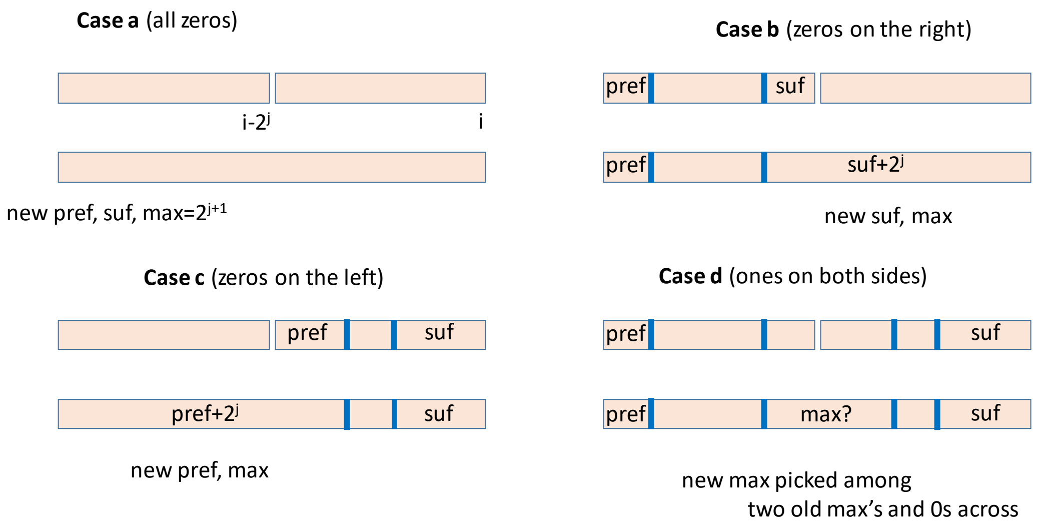
- *Write a pseudocode of a parallel solution, comment on its correctness and complexity.*

  The situation is more complex when we design a parallel solution. As discussed during the lecture we are often (if not always) limited to the computational methods which are available in parallel computing. And in particular the process of doubling our knowledge by considering block of size 2, 4,..., 2^j, ...

  Also in this case we will compute information about the longest sequence of zeros in the block $B[i,j]$ of size 2^j (if such exist) finishing at position i, for all $i = 0,\ldots,n-1$ in $S[0\ldots n-1]$. Lets denote this information by $L(i,j,max)$. Apart from having $L(i,j,max)$ (for a similar reason of having line (6) in the sequential code) we also need the length sequences of 0s in front of this block and at its conclusion denoted by $L(i,j,\mathit{pref})$ and $L(i,j,\mathit{suf})$ respectively.

When we apply a single round of doubling (of blocks) process the information about block $B[i,j]$ will be concatenated with $B[i-2^j,j]$ to form $B[i,j+1]$ where we need to compute values $L(i,j+1,\mathit{max})$, $L(i,j+1,\mathit{pref})$ and $L(i,j+1,\mathit{suf})$, for all $i=0,\ldots,n-1$ simultaneously. Several cases apply, see the pseudocode below.

```
(1) for all $i = 0, \ldots, n-1$ do in parallel {
(2)     if ($S[i] == 0$) {$L(i,0,\mathit{max}), L(i,0,\mathit{pref}), L(i,0,\mathit{suf}) \leftarrow 1$;}
(3)        else {$L(i,0,\mathit{max}), L(i,0,\mathit{pref}), L(i,0,\mathit{suf}) \leftarrow 0$;}
(4)     }
(5)     for ($j = 1; j < \log n; j{+}{+}$) {
(6)         if ($i \geq 2^j$) { // this block has a predecessor
(6)a            case ($L(i,j,\mathit{max}), L(i-2^j,j,\mathit{max}) == 2^j$) {
                        // both blocks are full of 0s
                    $L(i,j+1,\mathit{max}), L(i,j+1,\mathit{pref}), L(i,j+1,\mathit{suf}) \leftarrow 2^{j+1}$};
                        // the new (twice as big) block is full of 0s too
                }
(6)b            case ($L(i,j,\mathit{max}) == 2^j \neq L(i-2^j,j,\mathit{max})$) {
                        // only block $B[i,j]$ is full of 0s
                    $L(i,j+1,\mathit{max}), L(i,j+1,\mathit{suf}) \leftarrow L(i-2^j,j,\mathit{suf}) + 2^j$;
                    $L(i,j+1,\mathit{pref}) \leftarrow L(i-2^j,j,\mathit{pref})$;
                        // new $\mathit{max}$ and $\mathit{suf}$ extended by $2^j$, and $\mathit{pref}$ copied
                }
(6)c            case ($L(i-2^j,j,\mathit{max}) == 2^j \neq L(i,j,\mathit{max})$) {
                        // only block $B[i-2^j,j]$ is full of 0s
                    $L(i,j+1,\mathit{max}), L(i,j+1,\mathit{pref}) \leftarrow L(i,j,\mathit{pref}) + 2^j$;
                    $L(i,j+1,\mathit{suf}) \leftarrow L(i,j,\mathit{suf})$;
                        // new $\mathit{max}$ and $\mathit{pref}$ extended by $2^j$, and $\mathit{suf}$ copied
                }
(6)d            case ($L(i,j,\mathit{max}), L(i-2^j,j) \neq 2^j$) {
                        // both blocks contain 1s
                    $L(i,j+1,\mathit{max}) \leftarrow \mathit{MAXIMUM}\{L(i-2^j,j,\mathit{max})$,
                       $L(i,j,\mathit{max}), L(i-2^j,j,\mathit{suf}) + L(i,j,\mathit{pref})\}$;
                    $L(i,j+1,\mathit{pref}) \leftarrow L(i-2^j,j,\mathit{pref})$;
                    $L(i,j+1,\mathit{suf}) \leftarrow L(i,j,\mathit{suf})$;
                        // new $\mathit{max}$ can go across, but $\mathit{pref}, \mathit{suf}$ are copied
                }
(7)         }
(8)     }
(9) report ($L(n-1,\log n,\mathit{max})$); // the last entry knows the answer
```



Case a (all zeros)
i-2^j
i
new pref, suf, max=2^j+1
Case b (zeros on the right)
pref
suf
pref
suf+2^j
new suf, max
Case c (zeros on the left)
pref
suf
pref+2^j
suf
new pref, max
Case d (ones on both sides)
pref
suf
pref
max?
suf
new max picked among two old max's and 0s across

The time complexity of this solution is $O(\log n)$. Since we use n processing units the total work is $O(n \log n)$. One can reduce this work to linear (optimal) when using only $n/\log n$ processors, each initially processing a sub-block of size $\log n$ sequentially.