Design and Analysis of Parallel Algorithms
Exercise 2

s0840449

1 Question 1

1.1 Part (a)

The states that the sequence is transformed through is shown in Figure 1. The columns have been annotated with the part of the network that they belong to.

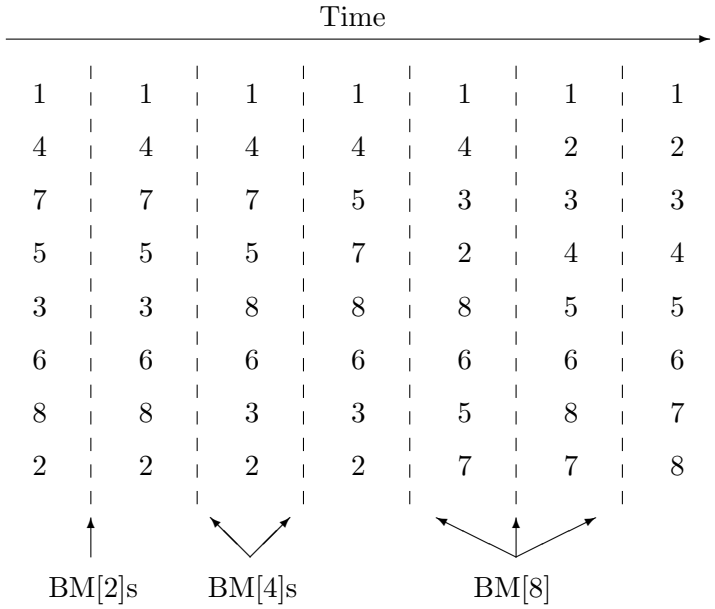

Figure 1: The states of the sequence from Question 1.a during a bitonic mergesort.

1.2 Part (b)

1.2.1 i.

In a ring processor network of $p = n$ processors, the furthest distance from one processor p' to another is $\frac{n}{2}$, or $2^{log(n)-1}$. A bitonic mergesort for n items proceeds through $log(n)$ recursively-built stages, where each stage has communication cost:

$$T_{comm} = \sum_{i=0}^{log(n)-1} communication\_cost(i)$$

The farthest that a processor needs to communicate during stage i is 2^i - during the initial stage ($i = 0$) only immediate-neighbour communication is needed ($2^0 = 1$), then in stage 1 processors need to communicate two neighbours over (e.g. wire 000 to wire 010, $2^1 = 2$), then 4, and so on. Additionally, for each stage i every stage $j < i$ is performed as well. This means that the communication cost for a stage i is the sum of all communication costs up to i, giving:

$$\begin{aligned}
T_{comm} &= \sum_{i=0}^{log(n)-1} \sum_{j=0}^{i} 2^j \\
&= \sum_{i=0}^{log(n)-1} (2^{i+1} - 1) \\
&= \left(\sum_{i=0}^{log(n)-1} 2^{i+1} \right) - log(n) \\
&= \left(2 \sum_{i=0}^{log(n)-1} 2^i \right) - log(n) \\
&= 2n - 2 - log(n) \\
&= \theta(n)
\end{aligned}$$

As the communication time $\theta(n)$ asymptotically dominates the computation time $\theta(log^2(n))$ (noting that these can be considered separately), the overall run time is $\theta(n)$.

1.2.2 ii.

There are three asymptotically significant differences in the new algorithm: an initial sequential sort, the replacement of exchange-swaps with compare-splits, and larger sized messages.

Initial sort: Sorting $\frac{n}{p}$ items takes time $\theta(\frac{n}{p} log(\frac{n}{p}))$.

Computation: The merge network now only has $log(p)$ steps at each depth, but also does a merge-split of $\frac{n}{p}$ items at every depth. This changes the recurrence to:

$$\begin{aligned}
d(p) &= d(\frac{p}{2}) + \frac{n}{p} log(p) \\
d(2) &= \frac{n}{p}
\end{aligned}$$

Solving gives the overall computation time:

$$\begin{aligned} d(p) &= \sum_{i=1}^{log(p)} \frac{n}{p} i \\ &= \frac{n}{p} \sum_{i=1}^{log(p)} i \\ &= \frac{n}{p} (\frac{log^2(p) - log(p)}{2}) \\ &= \theta(\frac{n}{p} log^2(p)) \end{aligned}$$

Communication: The number of communications and the distance between processors is now bounded by $log(p)$, but each message costs $\frac{n}{p}$ rather than 1. This gives a run-time:

$$\begin{aligned} T_{comm} &= \sum_{i=0}^{log(p)-1} \sum_{j=0}^{i} \frac{n}{p} 2^j \\ &= \frac{n}{p} \sum_{i=0}^{log(p)-1} \sum_{j=0}^{i} 2^j \\ &= \frac{n}{p} (2p - 2 - log(p)) \\ &= 2n - 2\frac{n}{p} - \frac{n}{p} log(p) \\ &= \theta(n) \end{aligned}$$

(It can be seen that n dominates $\frac{n}{p} log(p)$ by multiplying by p: np clearly dominates $nlog(p)$.) The overall runtime is:

$$T_{all} = \theta(\frac{n}{p} log(\frac{n}{p})) + \theta(\frac{n}{p} log^2(p)) + \theta(n)$$

Calculating the cost:

$$Cost = \theta(nlog(\frac{n}{p})) + \theta(nlog^2(p)) + \theta(np)$$

This is cost optimal for $p = log(n)$:

$$\begin{aligned} Cost &= \theta(nlog(\frac{n}{log(n)})) + \theta(nlog^2(log(n))) + \theta(nlog(n)) \\ &= \theta(nlog^2(log(n))) + \theta(nlog(n)) \\ &= \theta(nlog(n)) \end{aligned}$$

(The final step can be seen by noting that $n = 2^b$; substituting this gives $nlog^2(b)$ and nb, the latter of which clearly dominates the former.)

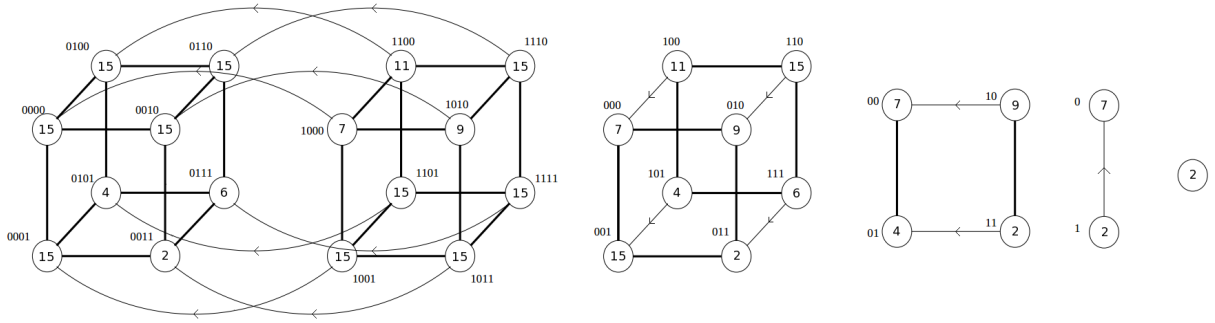

Figure 2: An example of the index distribution method for question 2 on the 16 character string 'abcqergstguevcba'. The method finds the correct answer, 2, in $log(16) = 4$ steps. Note only the relevant parent hypercube is shown in each step - there are actually two 3 dimensional cubes, four 2 dimensional cubes, and so on.

2 Question 2

First notice that any two elements $A[i]$ and $A[n-i-1]$ $(0 \leq i \leq n-1)$ on the hypercube are in opposite 'corners', i.e., have maximal distance between them. Since these must be compared, they must be on the same processor at some step. As the distance between two 'corners' on a hypercube grows logarithmically in the number of processors $p = n$, the communication time to place the two elements in the same place will always be $\theta(log(n))$[1]. Therefore, start by letting all pairs of processors i and $n-i-1$ exchange their elements with each other.

Having made this exchange, each processor can compare the two values it holds, and determine if it represents a 'matching' index in the array or not. The processors cannot tell whether there have been any previous non-matching indices or not, only if they are a match. Each processor can however guess at the 'best' (last) matching index in the array. For processors who represent matching indices, the conservative guess is that all indices match, i.e. the best index is $n-1$. Processors that represent non-matching indices know that they are an upper bound on the best index, so set their best index to their label (position) minus 1.

This best index information can then be passed 'back' through the hypercube to processor 0 via a recursive process. If a d dimensional hypercube represents a sub-array, then its two $d-1$ dimensional 'parent' hypercubes represent the first and second halves of the sub-array. The initial case is $d = log(n)$, for the hypercube that contains the entire array. To find the minimum index in the hypercube, let each processor in the second parent hypercube pass its best index to its first-parent neighbour, and let the neighbour choose the minimum of the received index and its own best index. This process then repeats recursively with the $d-2$ dimensional grandparents, and so on. The base case occurs when $d = 0$, at which point processor 0 has the index i of the last element in the array prefix such that $A[i] == A[n-i-1]$. This process is illustrated in Figure 2. The resulting algorithm is given in Algorithm 1.

The time complexity of the algorithm is $\theta(log(n))$. Lines 1-2 are $\theta(1)$. Line 3 is $\theta(log(n))$, as *other* is always the furthest processor away, so communication in both directions of a single integer

[1]It does not matter if one is moved to the other, or both to a third location; the time will always be some fraction of $log(n)$, and thus $\theta(log(n))$.

Algorithm 1 A $\theta(log(n))$ time, $p = n = 2^d$ processor parallel algorithm for Question 2. Each processor runs this function in parallel.

```
1:  id ← processor's label
2:  other ← id XOR (1)_d                          ▷ (1)_d is the binary vector of d 1s.
3:  exchange A[id], A[other] with processor labelled other
4:  if A[id] == A[other] then
5:      best_index ← n − 1
6:  else
7:      best_index ← id − 1
8:  for i = log(n) − 1 downto 0 do
9:      if ith bit set in id then
10:         send best_index to neighbour with label id XOR 2^i
11:     else
12:         recieve neighbour_index from neighbour with label id XOR 2^i
13:         best_index ← min(best_index, neighbour_index)
14: if id = 0 then
15:     result ← best_index
```

is $\theta(2 * (1 * log(n))) = \theta(log(n))$. Lines 4-7 are $\theta(1)$. Lines 8-13 are $\theta(log(n))$, as the 'neighbour' processor is always a direct neighbour and so communication is $\theta(1)$, and the loop repeats $log(n)$ times. Finally, lines 14-15 are $\theta(1)$. Therefore, the overall runtime is:

$$\begin{aligned} time &= \theta(1) + \theta(log(n)) + \theta(1) + \theta(log(n)) + \theta(1) \\ &= \theta(1 + log(n) + 1 + log(n) + 1) \\ &= \theta(log(n)) \end{aligned}$$

The algorithm is not cost optimal. As it uses n processors, the cost is $n*\theta(log(n)) = \theta(n*log(n))$. The best known sequential algorithm is $\theta(n)$ (Appendix A), which is asymptotically faster than $\theta(n * log(n))$.

The algorithm cannot be cost optimal on $p = n$ processors. The cost of an n processor parallel algorithm has a lower bound of $\theta(n)$. Since the cost of the sequential algorithm is also $\theta(n)$, the run-time of a parallel algorithm with $p = n$ processors would have to be $\theta(1)$, for a cost of $\theta(1 * n) = \theta(n)$. As the pairs of elements $A[i]$ and $A[n - i - 1]$ must be compared and each pair are $log(n)$ links apart, it is impossible to have a $\theta(1)$ run-time algorithm.

I do not think that the algorithm can be directly scaled down to be cost optimal, due to the placement of the data. As long as elements are placed on a hypercube such that processor 0 holds elements $0, ..., k$, processor 1 holds elements $k + 1, ..., 2 * k$, and so on, comparable elements of the array are always going to be on hypercube 'corners'. The communication run-time, therefore, will be $\frac{n}{p}log(p)$ for p processors and n items in the array. Translating this to a cost gives $nlog(p)$, which will always be asymptotically greater than n, no matter what p is set to be.

A Sequential Analysis of Question 2

Lines 1-3 are $\theta(1)$. Lines 4-7 are $\theta(n)$, as lines 3-7 are $\theta(1)$ and the loop can repeat at most n times (j starts as $n-1$, the loop terminates if $j < 0$, and j is decremented on each iteration.) Therefore, overall the run-time is $\theta(1) + \theta(n) = \theta(n)$.

Algorithm 2 A $\theta(n)$ time sequential algorithm for Question 2.

```
1: i ← 0
2: j ← n − 1
3: count ← −1
4: while j ≥ 0 and A[i] == A[j] do
5:     i ← i + 1
6:     j ← j − 1
7:     count ← count + 1
```