1 PRAM algorithms

1.1 Give a CRCW PRAM algorithm to find the max of an array of N numbers. (1pt)

input: \( y[0 .. N] \);
output: max. in \( y[] \);

forall \( i = 0 \) to \( N-1 \)
  forall \( j = 0 \) to \( N-1 \)
    if \( y[i] > y[j] \)
      \( r[i] = 1; \)
    end if
  end forall
end forall

forall \( i = 0 \) to \( N-1 \)
  if \( (r[i] == N-1) \)
    max \( = y[i]; \)
  end if
end forall

update on write using summation

1.2 What are the speedup and efficiency as a function of \( N \)? (1pt)

Let assume that the best serial algorithm needs \( N \) time units (for example, read the array from
start to end using a for-loop and ignore all the overhead for process creations and if statements.
We also assume that every operation takes the same amount of time, 1 time unit). The parallel
algorithm above runs in 2 time units (\( r[i]=1; \) and \( \text{max} = y[i]; \)).

Then the speedup of this algorithm is

\[
S = \frac{T_1}{T_n} = \frac{N - 1}{2}
\]

And the efficiency is,

\[
E = \frac{S}{p} = \frac{N-1}{2N^2} = \frac{N - 1}{2N^2}
\]
1.3 An efficient Concurrent Read Exclusive Write (CREW) algorithm to compute the maximum \( N \) numbers (1pt)

Let assume that there exists \( i \) such that \( N = 2^i \). Then,

input: \( y[0 .. N] \);

var \( ub=N/2; \)

while (\( ub > 0 \))
do
  forall \( j = 0 \) to \( ub-1 \)
    if (\( y[j] < y[j + ub] \))
      swap(y[j], y[j + ub]);
  end if
  end forall
  \( ub = ub / 2; \)
end

return \( y[0] \);

This algorithm compares the first half and the second half of the array and picks up larger values between the two. (For example, if \( N = 16 \), compare 0 and 8, 1 and 9 and so on) After the first iteration, we have all the larger values in the first half. Then start the same procedure on the first half. (From 0 to 7 in the previous example) Keep iterating until the size of array we are looking shrink down to 1

1.4 What are the speedup and efficiency as a function of \( N \)? (1pt)

\( T_N = \log_2 N \) operations if we assume the swap operation takes one time unit. Then the speedup is,

\[
S = \frac{T_1}{T_N} = \frac{N - 1}{\log_2 N}
\]

\[
E = \frac{S}{p} = \frac{N - 1/\log_2 N}{N/2} = 2(N - 1) \quad \frac{N}{2 \log_2 N}
\]

2 Speedup

2.1 What is the parallel speedup? (1pt)

The given information is,

- \( T_1 = 64s \)
- \( T_{16} = 11.5s \)

The speedup \( S \) of a parallel algorithm is defined as \( S = \frac{T_1}{T_p} \), where \( T_1 \) denotes the execution time of the best serial algorithm and \( T_p \) denotes the execution time of the parallel algorithm. Therefore,

\[
S = \frac{T_1}{T_p} = \frac{64}{11.5} = 5.565
\]
2.2 If the program parallelized perfectly, what would $T_{16}$ be? (1pt)

The fact that one program has been perfectly parallelized, which means the overhead of the parallelization does not exist, implies $E = 1$. $E = 1 \Rightarrow \frac{S}{p} = 1 \Rightarrow S = p$. Therefore,

$$S = 16 \Rightarrow 16 = \frac{T_1}{T_{16}} = \frac{64}{T_{16}}$$

$$T_{16} = 4$$

2.3 What fraction of the serial running time $T_1$ is spent in the serial section? (1pt)

Let suppose $X$ denote the execution time of the perfectly parallelized part and $Y$ denote the execution time of the serial section. Then,

$$X + Y = 64 \quad (1)$$

$$\frac{X}{16} + Y = 11.5 \quad (2)$$

Solving these two equations yields,

$$X = 56s, Y = 8s$$

Therefore,

$$\frac{8}{64} = 12.5\%$$

3 Data Parallelism

3.1 Make a copy of $y[]$ in each row $i$ of $X[]$ ($X[i,:]$) (1pt)

forall $i = 0$ to $N-1$
    forall $j = 0$ to $N-1$
        $X[i,j] = y[j]$;
    end forall
end forall

3.2 permutation vector, (bonus 1pt)

4 Sorting

4.1 This strategy fails to produce the correct result. Why is this the case? (1pt)

The code has loop carried dependencies. That is, each process access the array $a[]$ simultaneously but the order of access can change the result.
4.2 Restructure the program so that it will parallelize (1pt)

for i = 0 to N-1
    if (i%2 == 0)
        forall j = 0 to N by 2
            Compare-exchange(a[j], a[j+1]);
        end forall
    else
        forall j = 0 to N by 2
            Compare-exchange(a[j+1], a[j+2]);
        end forall
    end if
end for