Parallel Computing

Exercise sheet 1 + Reference Solution

April 12, 2019

Disclaimer

This document contains the assignments from exercise-sheet 1 of the lecture *184.710 Parallel Computing 2019S*, the reference solution as well as my personal solution.

The reference solution is given directly in the assignments in *italics*, my personal solution is always located in the **Solution** subsection of an exercise.

I cannot guarantee the correctness of any solution provided in this document.

Exercise 1

You are given the following PRAM algorithm (in pseudo-C code):

```
par (0 <= i<n) {
    a[i] = b[i] + c[i];
}
```

1. What does the PRAM algorithm do?

   Element wise addition of the input arrays.

2. What is the asymptotic running time for inputs of size n?

   Time $O(1)$

3. What is the asymptotic number of operations?

   Operations $O(n)$

4. Which PRAM model is needed, which will suffice?

   EREW PRAM suffices

Solution

1. Given two input-arrays b, c of size n the algorithm calculates the sum of the two corresponding elements from the input-arrays.
2. $T_{par}(n) = O(1)$ (under the assumption that there are n processors).
3. $\texttt{Ops}(n) = O(n)$.
4. EREW PRAM is sufficient, because no processes access the same memory locations (only access to the i'th position).

Exercise 2

Consider the following PRAM program ($n, p > 0$):

```
par (i=0; i<n; i+= n/p ) {
    for (j=0; j<n/p; j++) {
        a[i+j] = b[i+j] + c[i+j];
    }
}
```

1. What does this PRAM algorithm do on inputs b, c and n, p?

   Same as program before

2. For which inputs does it work considering n and p?

   Works only for n divisible by p

3. What is the asymptotic running time and number of operations as a function of n and p?

   Time $O(n/p)$, operations $O(n)$

4. Can this restriction (from 2) be lifted? Extend the program accordingly (within the same time bound).

   One possible solution as follows:

```
par (i=0; i<n; i+= n/p ) {
        for (j=0; j<n/p && i+j<n; j++) {
                a[i+j] = b[i+j] + c[i+j];
        }
}
```

Solution

1. Given two input-arrays b, c of size n the algorithm splits the input arrays to chunks of size $\frac{n}{p}$ and calculates the sum of the two corresponding elements from the input-arrays sequentially in each chunk (and the chunks are worked in parallel).

2. $p \mid n$

3. $T_{par}(n, p) = O(\frac{n}{p})$, $\texttt{Ops}(n, p) = O(n)$

4. Pseudo-code as below:

```
par (i=0; i<p; i++) {
        elems = n/p;
        rest = n%p;
        first = i * elems;
        if (i < rest){
                first+=i;
                elems++;
        } else {
                first+=rest;
        }
        for (j=0; j<elems; j++) {
                a[first+j] = b[first+j] + c[first+j];
        }
}
```

Exercise 3

Consider the following PRAM program, where n is a power of two:

```
for (k=1; k<n; k<<=1) {
        par (0<=i<k) {
                a[i+k] = a[i];
        }
}
```

1. What does this PRAM algorithm do?

   Copies the value from $a[0]$ to all other elements of a.

2. What is the asymptotic running time of the algorithm?

   Time $O(\log n)$

3. What is the largest number of processors used in any step?

   In the last iterations uses $\frac{n}{2}$ processors.

4. What is the asymptotic, total number of operations performed by the algorithm?

   $O(n)$ operations, $\Sigma_{i=0}^{\log(n)-1} 2^i = 2^{\log n} - 1 = n - 1$

5. Which PRAM model is required?

   EREW PRAM

6. Assuming a stronger PRAM model, can the operation of the algorithm be done faster? How fast?

   Time with CREW PRAM $O(1)$

Solution

1. The algorithm "flood-fills" the input array a with the value in $a[0]$ so that after the algorithm $a[i] = a[0] \forall i$ with $0 < i < n$.
2. $T_{par}(n) = O(\log n)$
3. $\#\texttt{processor} = \frac{n}{2}$
4. $\texttt{Ops}(n) = O(n)$
5. EREW PRAM is sufficient because read-operations to the same cell are sequential in the outer `for`-loop.
6. By choosing CREW PRAM $a[0]$ can be read parallel and the task can be completed in $T_{par}(n) = O(1)$ assuming $n - 1$ processors.

Exercise 4

Devise an EREW PRAM algorithm the sum of n numbers (n is a power of two) stored in an array a. The output can be stored as $a[0]$, and the algorithm is allowed to destroy the input elements.

1. Give the pseudo-code of your PRAM algorithm.

   A possible solution is to reverse the broadcast algorithm:

```
for (k=1; k<=log(n); k++) {
        par (i=0; i<n; i+=2^k) {
                a[i] = a[i] + a[i+2^(k-1)];
        }
}
```

   Another possible solution:

```
for (k=n/2; k>0; k>>=1) {
        par (i=0; i<k; i++) {
                a[i] = a[i] + a[i+k];
        }
}
```

2. How fast can your algorithm be in number of parallel steps?

   works in $O(\log n)$ time

3. How many operations does it perform?

   $O(n)$ operations

4. Can it be improved (sped-up) by using a stronger CREW PRAM model?

   CREW does not help

Solution

1. Pseudo-code as below:

```
for (k=n/2; k>=1; k>>=1) {
        par (0<=i<k) {
                a[i] = a[i] + a[i+k];
        }
}
```

2. $T_{par}(n) = O(\log n)$
3. $\texttt{Ops}(n) = O(n)$
4. No, because the addition can only work on two inputs and the algorithm already operates on every input element.

Exercise 5

A problem of size n can be solved sequentially in at most $Cn \log n$ operations for some constant C independent of n, but only $cn \log n$ operations can be efficiently parallelized to run in time $\frac{cn \log n}{p}$ time steps for some other constant c with $c < C$.

1. Assume n sufficiently large (larger than p), what is the maximum speed-up that this algorithm can achieve?

   Amdahl, sequential fraction $s = \frac{(C-c)}{C}$, speed-up at most $1/s = \frac{C}{C-c}$

2. Compute the maximum speed-up for $C = 100$, $c = 10$ and $C = 100$, $c = 99$.

   $C = 100, c = 10 \rightarrow 1.11, C = 100, c = 99 \rightarrow 100$

Solution

1. Calculation of speed-up:

$$\begin{aligned} S_p(n) &= \frac{T_{seq}(n)}{T_{par}(p,n)} \\ &= \frac{Cn\log n}{\frac{cn\log n}{p}} \\ &= \frac{Cp}{c} \end{aligned}$$

2. For $C = 100, c = 10$:

$$\begin{aligned} S_p(n) &= \frac{100}{10}p \\ &= 10p \end{aligned}$$

   For $C = 100, c = 99$:

$$S_p(n) = \frac{100}{99}p$$

Exercise 6

An algorithm is running in $O(n^2)$ operations. We want a speed-up of 30 using all cores of a 32-core processor.

1. How large can the sequential fraction be at the most to achieve this speed-up?
   Amdahl, $\frac{1}{s+\frac{1-s}{p}} = S,\ \frac{1}{s+\frac{1-s}{32}} = 30,\ s = \frac{32/30-1}{31} \approx 0.002$

Solution

1. Starting with Amdahl's law:

$$\begin{aligned} S_p(n) &= \frac{1}{s+\frac{r}{p}} && \text{Amdahl's law} \\ &= \frac{1}{s+\frac{1-s}{p}} && \text{Amdahl's law: } s = (1-r) \\ s &= \frac{p - S_p(n)}{S_p(n)\cdot(p-1)} && \text{standard math transformation} \\ &= \frac{32-30}{30\cdot(32-1)} \\ &= \frac{1}{465} \end{aligned}$$

Exercise 7

A work-optimal algorithm ($O(n^2)$ sequential time) is running in parallel time $O(\frac{n^2}{p}+\sqrt{n})$ operations (example: matrix-vector multiplication for square matrices). A speed-up of 90 with 100 cores is required.

1. How large must the input matrix and vector be?

$$\begin{aligned} \frac{n^2}{n^2/p+\sqrt{n}} &= S \\ \frac{n^2}{n^2/100+\sqrt{n}} &= 90 \\ n^{-3/2} &= \frac{1}{90}-\frac{1}{100} \\ \sqrt{n^3} &= 900 \\ n &= \sqrt[3]{900^2} \approx 93.2 \end{aligned}$$

vector: $n > 93$*, matrix:* $n^2 > 8649$

Solution

1. To archive the desired speed-up the matrix M and the vector v need to be of size $|M| = n \times n, |v| = n$ for $n \geq 94$.

$$\begin{aligned} S_p(n) &= \frac{T_{seq}(n)}{T_{par}(p,n)} && \text{speed-up} \\ &= \frac{n^2}{\frac{n^2}{p}+\sqrt{n}} \\ n &\approx 94 && \text{calculated with solver} \end{aligned}$$

Exercise 8

Another parallel algorithm is running in time $O(\frac{n^2 \log n}{p} + n)$, and the best known sequential counterpart runs in $O(n^2)$ operations.

1. What is the speed-up of this algorithm?

   $S = \frac{n^2}{\frac{n^2 \log n}{p}+n}$, *(possibly* $p \to \infty : n$*)*

2. What is the relative speed-up?

   $S = \frac{n^2 \log n + n}{\frac{n^2 \log n}{p}+n}$, *(possibly* $p \to \infty : n \log n + 1$*)*

3. For a problem of size n (fixed), how many processors does the parallel algorithm need in order to be faster than the sequential algorithm?

$$\frac{n^2 \log n}{p} + n < n^2$$
$$p > \frac{n^2 \log n}{n^2 - n} = \frac{n \log n}{n - 1}$$

Solution

1.

$$\begin{aligned} S_p(n) &= \frac{T_{seq}(n)}{T_{par}(p, n)} \\ &= \frac{n^2}{\frac{n^2 \log n}{p} + n} \\ &= \frac{np}{n \log n + p} \end{aligned}$$

2.

$$\begin{aligned} SRel_p(n) &= \frac{T_{par}(1, n)}{T_{par}(p, n)} \\ &= \frac{n^2 \log n + n}{\frac{n^2 \log n}{p} + n} \\ &= \frac{pn \log n + p}{n \log n + p} \end{aligned}$$

3.

$$T_{par}(p, n) < T_{seq}(n)$$
$$\frac{n^2 \log n}{p} + n < n^2$$
$$p > \frac{n \log n}{n - 1}$$

Exercise 9

You are given the following running times for work-optimal parallel algorithms ($O(n^2)$ sequential time): $O(\frac{n^2}{p} + \log p)$, $O(\frac{n^2}{p} + \log^2 p)$, $O(\frac{n^2}{p} + p)$, and $O(\frac{n^2}{p} + p\sqrt{p})$.

1. State the isoefficiency functions for each running time for fixed p.

   Efficiency needs to stay constant.

$$e = \frac{n^2}{p\left(\frac{n^2}{p} + \log p\right)} = \frac{n^2}{n^2 + p\log p}$$

$$n^2(1-e) = ep\log p$$

$$n = \sqrt{\frac{e}{1-e}p\log p} \qquad \text{Solved for n}$$

Solution for other running-times:

$$n = \sqrt{\frac{e}{1-e}p\log^2 p}$$
$$n = \sqrt{\frac{e}{1-e}p^2}$$
$$n = \sqrt{\frac{e}{1-e}p^2\sqrt{p}}$$

2. For $O(\frac{n^2}{p} + \log p)$ and $O(\frac{n^2}{p} + p\sqrt{p})$, compute this required input size n for $p = 10$ and $p = 100$ to maintain an efficiency of either $e = 0.5$ or $e = 0.9$. Round to the next larger integer.

$$n = \sqrt{\frac{e}{1-e}p\log p}$$

e/p	10	100
0.5	6	26
0.9	18	78

$$n = \sqrt{\frac{e}{1-e}p^2\sqrt{p}}$$

e/p	10	100
0.5	18	317
0.9	54	949

Solution

Note: $E(p,n)$ denotes the efficiency function, while e denotes a constant efficiency.

1. a) $O(\frac{n^2}{p} + \log p)$:

Efficiency:
$$E(p,n) = \frac{T_{seq}(n)}{p \cdot T_{par}(p,n)} = \frac{n^2}{n^2 + p\log p}$$

Express n in terms of p:
$$e = \frac{f(p)^2}{f(p)^2 + p\log p}$$
$$f(p) = \sqrt{\frac{ep\log p}{1-e}}$$

b) $O(\frac{n^2}{p} + \log^2 p)$:

$$E(p,n) = \frac{n^2}{n^2 + p\log^2 p}$$
$$f(p) = \sqrt{\frac{ep\log^2 p}{1-e}}$$

c) $O(\frac{n^2}{p} + p)$:

$$E(p,n) = \frac{n^2}{n^2 + p^2}$$
$$f(p) = \sqrt{\frac{ep^2}{1-e}}$$

d) $O(\frac{n^2}{p} + p\sqrt{p})$:

$$E(p,n) = \frac{n^2}{n^2 + \sqrt{p^5}}$$
$$f(p) = \sqrt{\frac{e\sqrt{p^5}}{1-e}}$$

2. a) $O(\frac{n^2}{p} + \log p)$:

e/p	10	100
0.5	6	26
0.9	18	78

b) $O(\frac{n^2}{p} + p\sqrt{p})$:

e/p	10	100
0.5	18	317
0.9	54	949

Exercise 10

You are given the following running times for work-optimal parallel algorithms ($O(n^2)$ sequential time): $O(\frac{n^2}{p} + \log n)$, $O(\frac{n^2}{p} + \log^2 n)$, $O(\frac{n^2}{p} + n)$, and $O(\frac{n^2}{p} + n\sqrt{n})$.

1. What is the maximum number of processors each of these algorithms can productively be used given a fixed but variable input size n? We look for the number of processors for which asymptotically the running time is bounded. For example, in case of $O(\frac{n^2}{p} + \log^2 n)$: how many processors can be used before the running time becomes $O(\log n)$?

2. For $n = 100$, compute the number of processors that can productively be used for each algorithm?

Remember, in big O notation $O(n^2/p + \log n) = O(\max\{n^2/p, \log n\})$*, thus the question is: When is* $\log n$ *larger than* n^2/p*?*

$$p = \frac{n^2}{\log n}, \text{ for } n = 100 \rightarrow 1505$$
$$p = \frac{n^2}{\log^2 n}, 226$$
$$p = \frac{n^2}{n}, 100$$
$$p = \frac{n^2}{n\sqrt{n}}, 10$$