Msc in High-Performance Computing

DESIGN AND ANALYSIS OF PARALLEL ALGORITHMS

Formative Assessment

B142302

1. Give the asymptotic time complexity when the time complexity is of each of the following forms. In each case, n is the input size.

(a) T = 5n^4 – 2n^2 + n. [1 mark]

BigO(n^4)

(b) T = 2^n + n^2 . [1 mark]

BigO(2^n)

(c) T = 7 log(n^2). [1 mark]

BigTheta(log(n))

(d) T = n^2 log(n) + 1000. [1 mark]

BigO(n^2 log(n))

(e) Which of the above represents the worst scaling with input size? [1 mark]

b. T = 2^n + n^2

Describe what each of the following algorithms is doing, and state the asymptotic time complexity.

(f) [1 mark]

Algorithm 1:

```
for i ← 0 to n^2 do
        A[i] ← 0;
end
```

Set the values of every element in array A[0:n^2] to 0.
BigO(n^2).

(g) [1 mark]

Algorithm 2:

```
 i ← 0;
while i < n^2 do
        A[i] ← 0;
        i ← i + 2;
end
i ← 1;
while i < n^2 do
        A[i] ← 1;
        i ← i + 2;
end
```

Set the first 2 values of every 3 elements in array A[0:n^2] to [0,1].
BigO(n^2).

(h) [1 mark]

Algorithm 3:

```
 sum ← 0;
```

```
for i ← 0 to n do
        j ← 1;
        while j ≤ n do
                sum ← sum + A[i, j – 1];
                j ← 2j;
        end
end
```

Set the first 2 values of every 3 elements in array A[0:n^2] to [0,1].
BigO(n^2).

2. Recall Amdahl's and Gustafson's scaling laws.

(i) State the assumptions implicit in both laws and in what situations they are applicable. [5 marks]

Amdahl's Law assumes the problem size won't change even if we increase the number of processors. Because the run time of the serial fraction is constant, the speed-up depends on the part of the problem which can not be parallelised. It does not consider the scalability factor (e.g. overheads for parallel scheduling), as the workload cannot scale to match the available computing power as the machine size increases.

$$S_A(\alpha, P) = \frac{1}{\alpha + \frac{(1-\alpha)}{P}}$$

for P is the number of processors, and α is the serial fraction of the problem. Amdahl's Law is useful when the number of processors is small.

Gustafson's law scaling law assumes the problem scales up linearly with the number of processes.

$$S_G(\alpha, P) = \alpha + (1-\alpha)P$$

for P is the number of processors, α is the serial fraction of the problem, and the problem size is P. Gustafson's law is applicable to the cloud or distributed computing environment, where it is possible to provide computing resources on demand to distribute massive data sets with a latency.

(ii) A parallel program has a serial runtime of 1220.0055s, and 80% of it is parallelised, while the rest is serial. How long would you expect it to take on 1024 processes? Show your work. [1 mark]

1220.0055s * 20% + 1220.0055s * 80% / 1024 = 244.9542s

(iii) How long would you expect it to take when the input was 1024 times larger than the input to the serial job? [1 mark]

1024 times longer.

You have previously seen the CREW PRAM parallel algorithm given in algorithm 4 for the Fractional Knapsack Problem.

Algorithm 4: CREW PRAM Fractional Knapsack

```
calculate profitabilities (independently in parallel);
sort in Divide and Conquer;
A = (a_1, . . . , a_n); B = (b_1, . . . , b_n);
log n integral;
k ← n/log n integral;
j_0 ← 0;
j_k ← n;
for 1 ≤ i ≤ k − 1 in parallel do
        j_i ← rank(b_(i log n) : A)
for 0 ≤ i ≤ k − 1 in parallel do
        B_i ← (b_(i log n+1), . . . , b_((i+1) log n))
        A_i ← (a_(j_i+1), . . . , a_j_(i+1) )

compute prefix sum of weights;
Knapsack capacity = c;
for each object independently in parallel do
        if myprefix ≤ c then the object is completely included;
        end
        else if myprefix > c && left neighbour's prefix < c then
                an easily determined fraction of the object is included;
        end
        else
                object is not included;
        end
end
```

(l) Modify algorithm 4 for a distributed parallel architecture. You may assume that a $\Theta(n)$ parallel sort is possible on all architectures. You may also assume that an $O(1)$ function exists which can provide the current process with its own process ID. State your chosen parallel architecture and the number of processes required. You may describe your algorithm using a mixture of diagrams and pseudocode. At the end of your algorithm, every process should have a copy of the solution to the fractional knapsack problem. [4 marks]

Choosing PRAM. Here we assume the problem size p = n, so we require n processes in which we have n objects.

(m) What is the asymptotic time complexity of your algorithm? What is the cost? [3 marks]
Divide and conquer requires time O(log n) to generate the subproblems and work in O(n). So the whole work can be done in O(n log n).

(n) Is your algorithm cost optimal? Explain why. [2 marks]
It is cost-optimal.