2015 FALL CSE 603 MIDTERM EXAM

Saturday, September 25, 2015

Problem 1 (15pt). Consider time measurements reported in the table below for a certain parallel algorithm executed on some parallel computer (time unit is irrelevant). n indicates the size of the problem solved, and p indicates the number of processors used. Please, write a brief analysis of the algorithm (is it scalable, if yes in what sense, if not why, etc.)

$p \backslash n$	**100**	**200**	**400**	**800**	**1600**
2	9	20	42	87	180
4	4	10	21	43	80
8	3	5	11	20	40
16	2	3	5	11	20
32	2	2	3	6	10

The algorithm is weakly scalable (for fixed n/p ratio time stays roughly constant). Given sufficiently large n the algorithm exhibits strong scalability (time decreases roughly $2\times$ as the number of processors p doubles).

Problem 2 (25pt). The best sequential algorithm for solving a problem runs in $\Theta(n^2)$. A parallel algorithm designed for solving the same problem runs in:

$$T_p = \Theta\left(\frac{n^2}{p} + \frac{n}{\sqrt{p}}\log p\right) \text{ for } p \leq n^2.$$

Find the maximum number of processors as a function of n that can be used, such that the algorithm runs with the maximum possible efficiency. Suppose now that you have $x = n$ instances to solve, each of size n. Is it better to use $p = n$ or $p = n^2$ processors – why?

$T_1 = n^2$, hence $E_p = \Theta(\frac{n^2}{p \cdot \left(\frac{n^2}{p} + \frac{n}{\sqrt{p}}\log p\right)} = \frac{n^2}{n^2 + n\sqrt{p}\log p}$. To achieve $E_p = 1$ the term $n\sqrt{p}\log p$ must be of order $O(n^2)$, hence $\sqrt{p}\log p = O(n)$:

$\sqrt{p}\log p = O(n)$
$2\sqrt{p}\log\sqrt{p} = O(n)$
$\sqrt{p} = O\left(\frac{n}{\log n}\right)$
$p = O\left(\frac{n^2}{\log^2 n}\right)$

In case $p = n$ we can run all jobs at the same time, hence the total time:

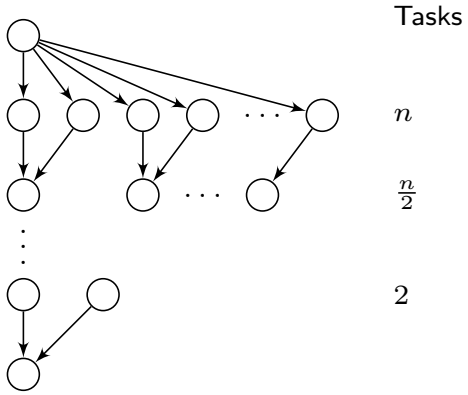
$$R_n = T_n = O\left(n + \sqrt{n}\log n\right).$$

In case $p = n^2$ we run x jobs sequentially, each using all available processors:

$$R_{n^2} = n \cdot T_{n^2} = O\left(n \log n\right).$$

Since $R_n < R_{n^2}$ for sufficiently large n, it is better to use $p = n$ processors.

Problem 3 (25pt). Consider the directed acyclic graph given below – it describes dependencies between tasks in some parallel algorithm. What is the maximal parallelism this algorithm can achieve?



We can directly apply work and span reasoning: first we observe that $n = 2^d$, then span (length of the critical path) is $T_\infty = \log n + 2$, and we can simplify to $T_\infty = \log n$. $T_1 = 1 + (n + \frac{n}{2} + \frac{n}{4} + \ldots + 1) = 1 + (2n - 1) = 2n$. Hence, $S_\infty = \frac{2n}{\log n + 2} = O\left(\frac{n}{\log n}\right)$.

Problem 4 (35pt). You are given a sequence of n integers $[x_0, x_1, \ldots, x_{n-1}]$ and $p \leq n$ processors where $p|n$ (i.e. n divides p). Processor p_i is assigned $[x_{i \cdot \frac{n}{p}}, \ldots, x_{(i+1) \cdot \frac{n}{p}-1}]$. Propose an efficient parallel algorithm to generate sequence $[s_0, s_1, \ldots, s_{n-1}]$ where $s_i = x_0 - x_1 - \ldots - x_i$. You can assume that two basic parallel primitives are given to you:

- $z \leftarrow \text{broadcast}(rank, z)$ in which processor p_{rank} communicates value z to all other processors.
- $s_{rank} \leftarrow \text{scan}(rank, x_{rank}, \otimes)$ in which p processors execute parallel prefix on p values x_i with operator $\otimes$ (processor p_{rank} stores x_{rank} and s_{srank}).

You can assume that all common sequential algorithms are given to you. For example, you can use function $[y_0, y_1, \ldots, y_n] \leftarrow \text{prefix}([x_0, x_1, \ldots, x_n], \otimes)$ to sequentially compute $[y_0 = x_0, y_1 = x_0 \otimes x_1, \ldots]$. Please, try to explain your algorithm using succinct pseudo-code, and avoid lengthy verbal explanations.

Each processor is assigned $\frac{n}{p}$ consecutive elements of x, for example p_0 is assigned $[x_0, x_1, \ldots, x_{\frac{n}{p}-1}]$. Because $-$ is not associative, we cannot compute s_i directly using prefix. However, we observe that $s_i = x_0 - (x_1 - x_2 - \ldots x_i) = 2x_0 - (x_0 + x_1 + \ldots + x_i)$. Now, we are given a prefix operation that works for case where $p = n$, hence we have to adopt it slightly to case where $p \ll n$.

1. Compute prefix sequentially with $+$ on each processor, store it in y, e.g. p_0 will have $[y_0 = x_0, y_1 = x_0 + x_1, \ldots, y_{\frac{n}{p}-1}]$
2. Compute parallel prefix over $[y_{\frac{n}{p}-1}, y_{\frac{2n}{p}-1}, \ldots, y_{n-1}]$ (which is distributed over p processors) and store it in y' (y'_0 stored on processor 0, y'_i on processor i): $y'_i \leftarrow \text{scan}(i, y_{\frac{(i+1 \cdot n)}{p}-1}, +)$
3. Update y with y' locally on each processor: on processor i we have $d = y'_i - y_{\frac{(i+1 \cdot n)}{p}-1}$ and we add d to every element of y
4. Broadcast x_0 from p_0 to all processors: $x_0 \leftarrow \text{broadcast}(0, x_0)$
5. Compute locally on each processor $s_i \leftarrow 2x_0 - y_i$