Parallel and Distributed Algorithms and Programs
TD n°3

Julien Braine   Laureline Pinault

9/10/2018

Part 1

Tree Root Finding

Let $\mathcal{F}$ be a forest of binary trees. Each node i of a tree is associated to a processor $P(i)$ and has a pointer toward its father $father(i)$.

Question 1

a) Give a P-RAM CREW algorithm so that each node finds $root(i)$. Show that your algorithm uses concurrent reads and gives its complexity.

Part 2

Givens Rotations on a Ring of Processors

In order to triangularise a matrix A of order n, one can use Givens rotations. The basic operation $\textsc{Rot}(i,j,k)$ consists in combining the two lines i et j, where each of them must start with $k-1$ zeros, to cancel the element at position (j,k):

$$\begin{pmatrix} 0 & \dots & 0 & \mathbf{a'_{i,k}} & a'_{i,k+1} & \dots & a'_{i,n-1} \\ 0 & \dots & 0 & \mathbf{0} & a'_{j,k+1} & \dots & a'_{j,n-1} \end{pmatrix} \leftarrow \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 0 & \dots & 0 & \mathbf{a_{i,k}} & a_{i,k+1} & \dots & a_{i,n-1} \\ 0 & \dots & 0 & \mathbf{a_{j,k}} & a_{j,k+1} & \dots & a_{j,n-1} \end{pmatrix}$$

Computation of θ is left to the astute reader. :-) The sequential algorithm can be written as follows:

Algorithm 1: Givens Rotation Procedure

```
def Givens(A):
    for k = 1 to n − 1 do
        for i = n downto k + 1 step −1 do
            Rot(i − 1, i, k)
```

We assume that a rotation $\textsc{Rot}(i,j,k)$ can be executed in constant time, independently of k.

Question 2

a) Adapt this algorithm to a linear network of n processors $\rightarrow P_1 \rightarrow P_2 \ldots \rightarrow P_n$.

b) Same question with a bidirectional linear network of processors with only $\lfloor \frac{n}{2} \rfloor$ processors $\rightleftarrows P_1 \rightleftarrows P_2 \ldots \rightleftarrows P_{n/2}$.

Part 3

Acceleration Factor

Question 3

a) Consider a problem to solve, which necessitates a percentage f of inherently sequential operations. Show that the acceleration factor is limited by $1/f$, regardless of the number of processors used. What lesson can be learned for the parallelization of a fixed size problem?

b) We assume that to solve a problem of size $n \times n$:

- the number of arithmetic operations to execute n^α, with α a constant;
- the number of elements to store in memory is $w_1 n^2$, with w_1 constant;
- the number of input/output operations (intrinsically sequentials) is $w_2 n^2$, with w_2 a constant.

How can we estimate the acceleration obtained with p processors on a problem of large size? (Hint: do not hesitate to introduce constants and assume that every process has memory M.) What lesson can be learned for the parallelization of a problem with variable size?

c) Practical/cultural question: do superlinear acceleration factors exist? (i.e. with an efficiency strictly greater than 1)