TD 3: P-RAM 2

1 Acceleration factor

**Question 1** (Amdahl law). Consider an algorithm with a percentage $f$ of intrinsically sequential operations. Show that the acceleration factor is bounded by $1/f$, no matter how many processors. What to make of it?

**Question 2** (Gustafson factor). Gustafson introduced the acceleration factor $S_p = \frac{A(1)}{A(p)}$ where $A(p)$ is the average time of an arithmetic operation in a problem of the maximal size that can fit on $p$ processors. Compute $S_p$ for a $n \times n$ matrix operation with:

- $n^\alpha$ arithmetic operations,
- $w_1 n^2$ elements to be stored in memory,
- $w_2 n^2$ i/o (sequential) operations.

**Question 3.** Give examples of problems with super-linear acceleration factors.

2 Givens rotations on a linear network

To triangulate a matrix while staying numerically stable, we can use Givens rotations. The base operation is $\text{rot}(i,k)$ that combines lines $i$ and $i+1$, if they begin with $k-1$ zeros, to replace with 0 the element of coordinates $(i+1,k)$, for some magic $\theta$:

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

The sequential algorithm can be written:

```plaintext
for k = 1 to n-1:
    for i = n-1 downto k:
        rot(i,k)
```

**Question 4.** Distribute this algorithm on a linear network of $n$ processors. (Do not assume that $\text{rot}(i,k)$ gets faster when $k$ increases.)

**Question 5.** Same question for only $n/2$ processors.

3 Connected components on P-RAMs

We want a CREW algorithm that computes the connected components of a graph $(V,E)$. More precisely if $V = \llbracket 1,n \rrbracket$, we want an array $C$ of size $n$ such that $C(i) = C(j) = k$ if and only if $i$ and $j$ are in the same connected component and $k$ is the smallest node of this component.

**Definition 1.** At each step of the algorithm, we call pseudo-node labelled with $i$ the set of nodes $C^{-1}(i) = \{j \mid C(j) = i\}$. We will sometimes identify $i$ and $C^{-1}(i)$. 

The main invariant of the algorithm is that the smallest node of $C^{-1}(i)$ is $i$ and that the nodes inside the same pseudo-node are in the same connected component. This assertion holds if we initialise $C$ with $\forall i \in V \ C(i) = i$, meaning that each node considers itself the reference node for its connected component. The goal of the algorithm is to change this selfish attitude.

**Definition 2.** A $k$-cyclic pseudo-tree is a directed graph weakly connected (i.e. the induced undirected graph is connected) such that:

- the outdegree of every node is 1,
- there is exactly one cycle of length $k + 1$.

A star is a 0-cyclic pseudo-tree where all edges point to the smallest node (the root).

The invariant above is that the directed graph $G_C = (V, \{(i, C(i)) | i \in V\})$ is made of stars. The computation of connected components is done by iterating the following two functions:

**gather()**:

```
for i \in V in parallel do:
    N(i) := \min\{C(j) | \{i, j\} \in E, C(i) \neq C(j)\} or C(i) if the set is empty
```

**jump()**:

```
for i \in V in parallel do:
    B(i) := T(i)
repeat \log(n) times:
    for i \in V in parallel do:
        T(i) := T(T(i))
    for i \in V in parallel do:
        C(i) := \min\{B(T(i)), T(i)\}
```

**Question 6.** Apply **gather** on the following graph, then **jump**, then **gather**, etc. Keep track of the graphs $G_C$ and $G_T = (V, \{(i, T(i)) | i \in V\})$.

**Question 7.** Show that after an application of **gather**, connected components containing several pseudo-nodes induce 1-cyclic pseudo-trees in $G_T$. Note that the smallest pseudo-node of a 1-cyclic pseudo-tree is inside the cycle.

**Question 8.** Show that **jump** transforms a 1-cyclic pseudo-tree into a star.

**Question 9.** Show that after $\lceil \log n \rceil$ combinations of **gather** and **jump** the connected components of the graph are represented by $C$.

**Question 10.** What is the complexity of the algorithm? How many processors are used?