Parallel and Distributed Algorithms and Programs
TD n°1 - P-RAM

Hadrien Croubois
hadrien.croubois@ens-lyon.fr

Aurélien Cavelan
aurelien.cavelan@ens-lyon.fr

16/10/2015

All documents are available on my website: `http://hadriencroubois.com/#Teaching`

Part 1

Selection in a list

Question 1

a) Let L be a list containing n objects colored either in blue or red. Design an effective EREW algorithm that separates the blue elements from the red elements (i.e. that builds a new list containing only the blue elements).

Part 2

Mystery Procedure

We define the following two operators for a table $A = [a_0, a_1, \dots, a_{n-1}]$ of n integers:

- Prescan(A) returns the table: $[0, a_0, a_0 + a_1, a_0 + a_1 + a_2, \dots, a_0 + a_1 + \dots + a_{n-2}]$
- Scan(A) returns the table: $[a_0, a_0 + a_1, a_0 + a_1 + a_2, \dots, a_0 + a_1 + \dots + a_{n-1}]$

These two operators can be computed in $O(\log n)$ time on P-RAM EREW.
Given a table *Flags* we define the following Split procedure:

Algorithm 1: Mystery Procedure 1

```
def Split(A, Flags):
    Iup ← n − Reverse(Scan(Reverse(Flags)));
    Idown ← Prescan(1 − Flags);
    for i = 1 to n do in parallel
        if Flags(i) then
            Index[i] ← Iup[i]
        else
            Index[i] ← Idown[i]
    Result ← Permute(A, Index);
    return Result
```

The names of the different functions are relatively intuitive. In particular, Reverse reverse the table, and Permute(A,*Index*) reorders table A according the permutation *Index*. The horrible expression Reverse(Scan(Reverse(*Flags*) does a simple Scan but from the end of table *Flags* (of which the elements are considered as integers).

Question 2

a) Apply the procedure on this input:

$$\begin{array}{lcl} A & = & [\ 5\ \ 7\ \ 3\ \ 1\ \ 4\ \ 2\ \ 7\ \ 2\ ] \\ Flags & = & [\ 1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ ] \end{array}.$$

b) What is the purpose of the Split procedure?

c) What is the computational time of the Split procedure?

Question 3

a) We consider the following Mystery procedure:

Algorithm 2: Mystery Procedure 2

```
def Mystery(A, Number_Of_Bits):
    for i = 0 to Number_Of_bits − 1 do
        bit(i) ← table containing the i^th bit of the elements of A;
        A ← Split(A, bit(i));
```

(a) Run the procedure on $A = [5, 7, 3, 1, 4, 2, 7, 2]$ with $Number\_Of\_Bits = 3$.

(b) What is the purpose of procedure Mystery 2?

(c) Given entries of size $O(\log n)$ bits, what is the complexity with n processors? With p processeurs?

Part 3

Connected components

We would like to design a CREW algorithm to compute the connected components of a graph $G = (V, E)$ with vertices numbered from to 1 to n. In particular, we are looking for an algorithm that returns a table C of size n, such that $C(i) = C(j) = k$ if and only if i and j are in the connected component and k is the smallest index among the vertices from this component.

Definition 1 *For all iteration of the algorithm, we call the pseudo-vertex labeled by i the set of vertices $j, k, l, \cdots \in V$ such that $C(j) = C(k) = C(l) = \cdots = i$. In other words, we consider the pseudo-vertex labeled by i to be the same as the vertex labeled by i.*

One of the invariants of the algorithm is that the smallest index of the vertices from the pseudo-vertex labeled by i is i and the vertices belonging to a pseudo-vertex are in the same connected component. This assertion is true if we initialize C by: for all $i \in V = [\![1, n]\!] : C(i) = i$. This means that at the begining, each processor considers itself as the pseudo-vertex of its connected component. The goal of the algorithm is to change this egocentric point of view.

Definition 2 *A k-cyclic tree ($k \geqslant 0$) is a weakly connected oriented graph such that:*

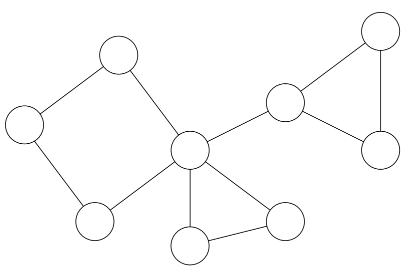
- *Each vertex has an out-degree of* 1
- *There is exactly one circuit of length* $k + 1$.

We call a star a 0*-cyclic tree.*

Therefore, the previous invariant is that the oriented graph $(V, \{(i, C(i)) \mid i \in V\})$ consists of stars only. We can identify pseudo-vertex and stars, the center of the star being the index of the pseudo-vertex. Computing the connected components is done by running the following procedures several times:

Question 4

a) We consider the following graph:



Apply the function Gather on this graph, then the function Jump, and the Gather function again, etc.

Algorithm 3: Procedures to compute the connected components.

```
def Gather():
    for i ∈ S do in parallel
        T(i) ← min {C(j) | {i,j} ∈ E, C(j) ≠ C(i)} ;     // si l'ensemble est vide, on associe C(i)
    for i ∈ S do in parallel
        T(i) ← min {T(j) | C(j) = i, T(j) ≠ i} ;        // si l'ensemble est vide, on associe C(i)
def Jump():
    for i ∈ S do in parallel
        B(i) ← T(i)
    for j = 1 to log n do
        for i ∈ S do in parallel
            T(i) ← T(T(i))
    for i ∈ S do in parallel
        C(i) ← min {B(T(i)), T(i)}
```

b) Show that after using the Gather function, connected components containing several pseudo-vertices induce 1-cyclic trees in the oriented graph $(V, \{(i, T(i)) \mid i \in V\})$. Note that the smallest pseudo-vertex of a 1-cyclic tree belongs to the cycle.

c) Show that the function Jump transforms a 1 cyclic tree into a 1-cyclic star (or pseudo-vertex).

d) Show that after $\lceil \log n \rceil$ iterations, the connected components of the graph are represented by pseudo-vertices induced by C.

e) What is the complexity of the algorithm? How many algorithms are used?