DS 2013 Parallélisme

Damien Crémilleux, Lauriane Holy

25 mars 2014

1 Questions de cours

1.1 Question 1

Les compteurs de synchronisation possèdent plusieurs avantages :
- synchronisation moins restrictives que l'exclusion mutuelle (on peut laisser passer plus d'un processus dans la zone protégé)
- les variables de synchronisation et les réveils sont gérés automatiquement
- leurs spécifications sont plus abstraites et concises (mécanisme de haut niveau)

Ils ont cependant plusieurs inconvénients :
- difficile à mettre en oeuvre
- réévalutions parfois inutiles dans conditions

1.2 Question 2

La différence entre les wait/notify Java et les moniteurs de Hoare réside dans le réveil du thread après que celui-ci ait été bloqué. Dans le cas des moniteurs de Hoare, lors de l'instruction reprendre(), le thread faisant l'appel est bloqué pour laisser la place à un processus précédement bloqué. Le processus débloqué est donc sûr que sa condition de réveil est vrai. Au contraire, dans le cas d'un notify() en java, le thread réveillé se contente d'aller dans la queue d'entrée du moniteur. Au moment où il rentrera, la condition d'entrée ne sera plus nécessairement vraie, d'où la nécéssité de mettre le wait() dans une boucle while.

1.3 Question 3

La communication par rendez vous n'est pas le moyen de communication le plus rapide dans un programme parallèle exécuté sur un réseau de machine car l'émission et la réception de message sont bloquantes, et il faut attendre la confirmation de réception (principe de l'accusé de réception). Ainsi le premier thread arrivé attend l'autre. Il est possible d'utiliser des buffers de communication pour aller plus vite, et ne plus avoir de réception bloquante.

2 Questions de TP

2.1 Question 1

Lors du TP MPI, chaque processeur s'occupe d'une tranche de la plaque. Cependant, comme un processeur a besoin des plaques calculées par ses voisins, il y un recouvrement aux extrémitées. Ces parties sont donc en double, et sont mises à jour au fur et à mesure de l'avancement des threads.

2.2 Question 2

On suppose que la matrice possède M ligne et N colonnes. Soit P le nombre de processus, il va donc y avoir nbPlaques =2 * (N / P) (arrondi à l'entier supérieur), et (nbPlaques - 1) recouvrements. En proportion, il y a donc (2*(N/P)-1)/N recouvrement.

3 Problème : Synchronisation

```
package ds2013;

public class Facteur {

    private static final int nbThread = 10;

    private int[][][] valeur; //la valeur a transmettre
    private int[][][] etat; //indique si une valeur est a lire
    private boolean[][][] tabReady; //indique l'etat d'attente

    public Facteur() {
        valeur = new int[nbThread][nbThread];
        etat = new int[nbThread][nbThread];
        tabReady = new boolean[nbThread][nbThread];
    }

    //envoi de la valeur value au processeur tid
    void send(int tid, int value) throws Exception {
        synchronized(this) {
            etat[(int)Thread.currentThread().getId()][tid] = 1;
            valeur[(int)Thread.currentThread().getId()][tid] =
                value;
            this.notifyAll();

            while(etat[(int)Thread.currentThread().getId()][tid] ==
                 1) {
                tabReady[(int)Thread.currentThread().getId()][tid]
                    = true;
                this.wait();
                tabReady[(int)Thread.currentThread().getId()][tid]
                    = false;
            }
        }
    }

    // reception de la valeur du message envoye par le processus
        tid
    int recv(int tid) throws Exception {
        synchronized(this) {
```

```
            while(etat[tid][(int)Thread.currentThread().getId()] ==
                 0) {
                tabReady[tid][(int)Thread.currentThread().getId()]
                    = true;
                this.wait();
                tabReady[tid][(int)Thread.currentThread().getId()]
                    = false;
            }

            int res = valeur[tid][(int)Thread.currentThread().getId
                ()];
            etat[tid][(int)Thread.currentThread().getId()] = 0;
            this.notifyAll();
            return res;
        }
    }

    //dit si le recepteur est pret (bloque en attente)
    //et si on peut donc emettre sans se bloquer
    boolean sendReady() {
        synchronized(this) {
            boolean send = false;
            for (int i = 0; i < nbThread; i++) {
                send = send || (tabReady[(int)Thread.currentThread
                    ().getId()][i] == true); //on regarde si au
                    moins un thread est pret a recevoir
            }
            return send;
        }
    }

    //dit si l'emetteur est pret (bloque en attente)
    //et si on peut donc recevoir sans se bloquer
    boolean recvReady() {
        synchronized(this) {
            boolean recv = false;
            for (int i = 0; i < nbThread; i++) {
                recv = recv || (tabReady[i][(int)Thread.
                    currentThread().getId()] == true); //on regarde
                     si au moins un thread est pret a envoyer
            }
            return recv;
        }
    }

    //affiche le(s) cycles des interblocages s'il y en a
    //rend vrai s'il y a un interblocage
    /*boolean deadlockDetect() {
    //?
    }*/


}
```