La trieuse systolique

P. Quinton

Novembre 2002

1 Introduction

Ce document décrit un programme Java qui simule le comportement d'un architecture systolique pour le tri. Ce programme a été réalisé par Lucien Ungaro[1].

On peut en voir le fonctionnement à l'url

`http://www.irisa.fr/cosi/Quinton/dea/java-applets/TriPara.html`

Le programme décrit dans ce document a le même comportement que cette applet, mais c'est un "vrai" programme Java.

La réalisation de la trieuse en Java repose sur l'utilisation de processus (*threads*) dont le comportement est synchronisé par des tampons accessibles en mode producteur-consommateur. La réalisation ainsi obtenue est donc une version asynchrone d'une architecture systolique[2].

L'architecture simulée comprend N processeurs (ou cellules, suivant la terminologie systolique). Chaque cellule a deux registres, appelés `min` et `max`, comme le montre la figure 1.

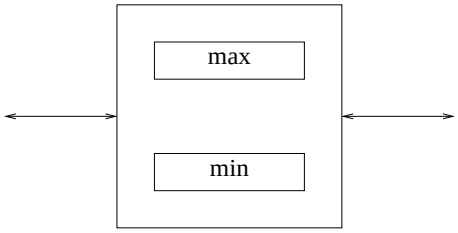

Figure 1: Cellule de la trieuse

On agence les cellules en réseau linéaire, comme le montre la figure 2.

Les valeurs à trier sont fournies au réseau par la gauche. Supposons qu'on veuille trier n nombres. Pendant n cycles, on présente les n valeurs à la cellule de gauche : ceci constitue une première phase *montante*. À l'issue de cette phase, les cellules émettent leurs données vers la gauche, et les données triées sont ainsi récupérées. Ceci constitue la phase *descendante*.

[1]Ifsic, Campus de Beaulieu, 35042, Rennes-Cedex

[2]En général, on considère qu'une architecture systolique est synchrone, c'est-à-dire que tous ses processeurs sont synchronisés par une horloge commune. C'est toutefois une hypothèse restrictive inutile, même si elle simplifie bien la vie !

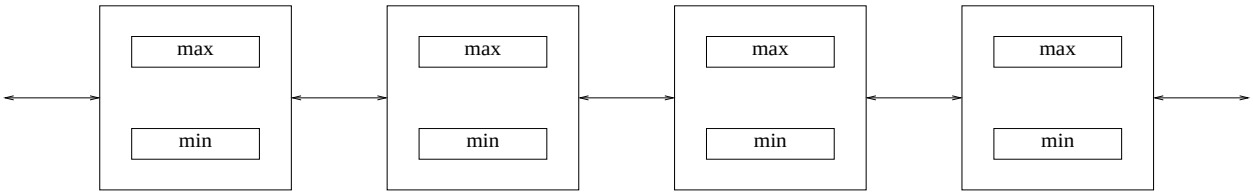

Figure 2: Trieuse systolique

Si on suppose un fonctionnement synchrone, à chaque cycle de la phase montante, la cellule :

- lit une valeur v qui lui est fournie par la gauche,
- émet simultanément le contenu de `max` vers la droite,
- compare v avec le contenu du registre `min`, et place la plus petite des deux valeurs dans `min`, et la plus grande dans `max`.

Lors de la phase descendante, une cellule :

- lit une valeur v qui lui est fournie par la droire
- émet simultanément le contenu de `min` vers la gauche,
- compare v avec le contenu du registre `max`, et place la plus petite des deux valeurs dans `min`, et la plus grande dans `max`.

2 Description succincte du programme

L'architecture est réalisée dans le programme appelé `TriPara.java`. Ce programme utilise deux librairies, appelées `es.java` et `lecture.java`.

Il comprend les classes suivantes : `Tamp`, `Trieuse`, `Emplacement`, `Bouchon` et `TriPara` lui-même.

2.1 La classe `Tamp`

Cette classe représente un tampon à une place, qu'on peut considérer comme un registre. Elle est utilisée pour permettre les communications entre processus, de façon synchronisée. Elle utilise des méthodes `prod` et `cons` permettant la production d'une valeur dans le tampon ou la consommation d'une valeur.

2.2 La classe `Trieuse`

C'est le modèle d'une trieuse à `N` emplacements. Utilise des fonctions d'affichage `Frame`, `txtResul` pour créer une fenêtre de visualisation.

Elle contient elle-même une classe `BiAfficheur` décrite en 2.3, qui permet l'affichage d'un processeur.

Elle contient aussi les méthodes `entrer` et `consulter` qui permettent d'entrer une donnée dans le réseau, et de lire le résultat fourni par le réseau.

Elle contient aussi la classe `Emplacement` décrivant le fonctionnement d'un processeur (voir 2.4).

Elle instancie un tampon d'entrée et un tampon de sortie, un bouchon qui est un processeur spécial placé à la droite de l'architecture.

Le constructeur de la trieuse crée les emplacements, etc.

2.3 La classe `BiAfficheur`

Cette classe permet l'affichage des données d'un processeur. Elle contient deux méthodes `afficher` et `interpEnt`.

2.4 La classe `Emplacement`

Elle décrit un processeur de la trieuse, et constitue un processus (`Thread`).

Elle commence par le constructeur, par la méthode `run` qui décrit l'exécution proprement dite du processus.

2.5 La classe `Bouchon`

Cette classe est une extension de `Tamp` et permet à la fois d'absorber les valeurs qui lui viennent de la gauche, et de fournir des valeurs infinies vers la gauche en phase ascendante.

3 Le programme `Lecture`

Ce programme contient quelques classes permettant la lecture de diverses données. Voir B.

A Programme TriPara

```
import es.*; import ihm.*; import java.awt.*;

//=====================================================================
class Tamp { // modele de tampon a une place

int tampon; boolean estVide=true;

synchronized void prod(int v) {
  if(!estVide){try {wait();} catch(InterruptedException e){};}
  tampon=v; estVide=false; notify();
}

synchronized int cons() {
  if(estVide){try {wait();} catch(InterruptedException e){};}
  int resul=tampon; estVide=true; notify(); return resul;
}
}
//=====================================================================


//=====================================================================
class Trieuse {

int N; // nombre d'emplacements
static final int infini = 999999;

// fenetre de visualisation de la trieuse
Frame fTrieuse = new Frame("TRIEUSE");
TextField txtResul= new TextField(25);

//---- modele d'afficheurs pour la paire min-max d'un emplacement -------
class BiAfficheur extends Panel {
TextField t1=new TextField(4); TextField t2=new TextField(4);

public BiAfficheur(String s1, String s2) {
  Placement.p(this,new Label(s2),1,1,1,1); Placement.p(this,t2,2,1,1,1);
  Placement.p(this,new Label(s1),1,2,1,1); Placement.p(this,t1,2,2,1,1);
  t1.setEditable(false); t1.setBackground(Color.white);
  t2.setEditable(false); t2.setBackground(Color.white);
}

public void afficher(int v1, int v2, int duree) {
  t1.setText(interpEnt(v1)); t2.setText(interpEnt(v2));
  try {Thread.sleep(duree); } catch(InterruptedException e){};
}
```

```
String interpEnt(int v) { // pour un affichage discret des infinis
  if (v==infini) return ""; else return String.valueOf(v);
}
}
//---------------------------------------------------------------------

public void entrer(int v) {entreeTrieuse.prod(v);}

public int consulter(){
  int resul=sortieTrieuse.cons();
  txtResul.setText(txtResul.getText()+resul+" "); return resul;
}


//---------------- modele de processus emplacement ----------------------
class Emplacement extends Thread {
BiAfficheur v=new BiAfficheur("min","max");

Tamp entreeG; Tamp sortieD; Tamp sortieG; Tamp entreeD;
int min; int max;

// Constructuer d'Emplacement
public Emplacement(Tamp eg, Tamp sg, Tamp sd, Tamp ed){
  entreeG=eg; sortieG=sg; sortieD=sd; entreeD=ed;
}

// Méthode d'exécution du thread
public void run() {
  min=infini; max=infini;
  for(int i=0; i<2*N; i++) { // phase montante
    sortieD.prod(max); max=entreeG.cons();    v.afficher(min,max,500);
    if(max<min) {int t=max; max=min; min=t;}; v.afficher(min,max,1000);
  }
  for(int i=0; i<2*N; i++) { // phase descendante
    sortieG.prod(min); min=entreeD.cons();    v.afficher(min,max,500);
    if(max<min) {int t=max; max=min; min=t;}; v.afficher(min,max,1000);
  }
}

}
//---------------------------------------------------------------------


Tamp entreeTrieuse = new Tamp(); // tampon d'entree de la trieuse
Tamp sortieTrieuse = new Tamp(); // tampon de sortie de la trieuse
```

```
class Bouchon extends Tamp {  // tampon de droite (tampon bidon)
  void prod(int v) {}         // prod : ne fait rien
  int cons() {return infini;} // cons : delivre des infinis
}

Tamp bouchon= new Bouchon();

// Construction de la trieuse
Trieuse(int nbEmplacements) {
  N=nbEmplacements;
  Placement.p(fTrieuse,txtResul,0,3,1,1);

   // cree N emplacements interconnectes par tampons
  Tamp entreeG=entreeTrieuse; Tamp sortieG=sortieTrieuse;
  for(int i=1; i<N; i++) {
    Tamp sortieD= new Tamp(); Tamp entreeD= new Tamp();
    Emplacement emp=new Emplacement(entreeG,sortieG,sortieD,entreeD);
    Placement.p(fTrieuse,emp.v,i,1,1,1); // positionne les afficheurs
    emp.start();
    entreeG=sortieD; sortieG=entreeD;
  };

  Emplacement emp= new Emplacement(entreeG,sortieG,bouchon,bouchon);
  Placement.p(fTrieuse,emp.v,N,1,1,1);
  emp.start();

  fTrieuse.pack(); fTrieuse.setVisible(true);
}
}
//=====================================================================


// programme principal (instanciation et utilisation d'une trieuse)
//=====================================================================
public class TriPara {
  static public void main(String argv[]){
  int N=5;
  Trieuse t=new Trieuse(N);

  Thread.currentThread().setPriority(Thread.currentThread().getPriority()-1);

  System.out.println("entrer "+2*N+" valeurs entieres");

  for(int i=0; i<2*N; i++) { t.entrer(Lecture.unEntier());};
  for(int i=0; i<2*N; i++) { int x=t.consulter();}
  System.out.println("termine");
  Lecture.unCarCmde(); // attente que l'utilisateur acquitte le resultat
```

```
    System.exit(0);
  }
  }
  //=====================================================================
```

B Programme Lecture

```
package es;
import java.io.*;

public class Lecture {
public static char unCar() {
  // lecture d'un caractere
  char c;
  try { c = (char) System.in.read();
  } catch(IOException e) {c= (char) 0;};
  return c;
}

public static char unCarCmde() {
  // lecture d'un caractere et consommation jusqu'au retour chariot
  char c;
  try { c = (char) System.in.read();
  while(System.in.read()!='\n'){};
  } catch(IOException e) {c= (char) 0;};
  return c;
}

public static String chaine(String delimiteurs) {
  // lecture d'une chaine comprise entre delimiteurs
  StringBuffer b = new  StringBuffer();
  char c=unCar();
  // ignore les delimiteurs de tete
  while (delimiteurs.indexOf(c)!=-1) {c=unCar();};
  // lit jusqu'au prochain delimiteur
  while (delimiteurs.indexOf(c)==-1) {b.append(c); c=unCar();};
  return b.toString();
}

public static String chaine() {
  // lecture d'une chaine comprise entre " ", ",", "." ou "\n"
  return  chaine(" ,.\n");
}

public static int unEntier() {
  // lecture d'un entier represente en decimal
  String s=Lecture.chaine(" ,.\n");
  try { return Integer.parseInt(s);
  } catch(NumberFormatException e) {return 0;};
}

public static float unReel() {
```

```
  // lecture d'un flottant represente en decimal
  String s=Lecture.chaine(" \n");
  try { return  Integer.parseInt(s);
  } catch(NumberFormatException e) {return 0;};
}

}
```

C Programme Placement

```
package ihm;
import java.awt.*;

// procedures de placement de composants visuels dans un receptacle

public class Placement {

  static GridBagLayout placeur= new GridBagLayout();
  static GridBagConstraints c = new GridBagConstraints();

  // procedure generale de positionnement
  //-------------------------------------
  public static void p(
         Container cont, Component comp, int x, int y, int w, int h, int pos,
         int t, int l, int b, int r, double wx, double wy, int fill) {

     cont.setLayout(placeur);
     c.gridx=x; c.gridy=y; // position (en nbre de cellules) du coin nord-est
     c.gridwidth=w; c.gridheight=h; // largeur et hauteur (en nbre de cellules)
     c.fill=fill;  // directions d'expansion : NONE, BOTH, HORIZONTAL, VERTICAL
     c.anchor=pos; // position du composant dans ses cellules :
                   // CENTER, EAST,NORTHEAST, NORTH, NORTHWEST,
                   // WEST, SOUTHWEST, SOUTH, SOUTHEAST ...
     c.weightx=wx; c.weighty=wy; // ponderation de la distribution de l'espace
                                 // supplementaire en cas d'agrandissement
     c.insets = new Insets(t,l,b,r); // marges en pixels
     placeur.setConstraints(comp, c); cont.add(comp);
  };

  // placement d'un composant qui ne grossit pas
  //--------------------------------------------
  public static void p(Container cont, Component comp,
            int x, int y, int w, int h, int pos, int t, int l, int b, int r) {
     p(cont, comp, x, y, w, h, pos, t, l, b, r,
                           0.0, 0.0, GridBagConstraints.NONE);
  };

  // positionnement d'un composant sans marges qui ne grossit pas
  //-------------------------------------------------------------
  public static void p(Container cont, Component comp,
                    int x, int y, int w, int h, int pos) {
     p(cont, comp, x, y, w, h, pos,
                              0, 0, 0, 0, 1.0, 1.0,
                                              GridBagConstraints.NONE);
  };
```

```
  // positionnement au centre d'un composant sans marges qui ne grossit pas
  //-------------------------------------------------------------------------
  public static void p(Container cont, Component comp,
                   int x, int y, int w, int h) {
     p(cont, comp, x, y, w, h,
                       GridBagConstraints.CENTER, 0, 0, 0, 0, 1.0, 1.0,
                                                  GridBagConstraints.NONE);
  };
}
```