Séminaire interne du LMT - 8 février 2001

Génération de code parallèle explicite, en espace, dans le cadre de réseaux d'automates cellulaires

leduc@lmt.ens-cachan.fr

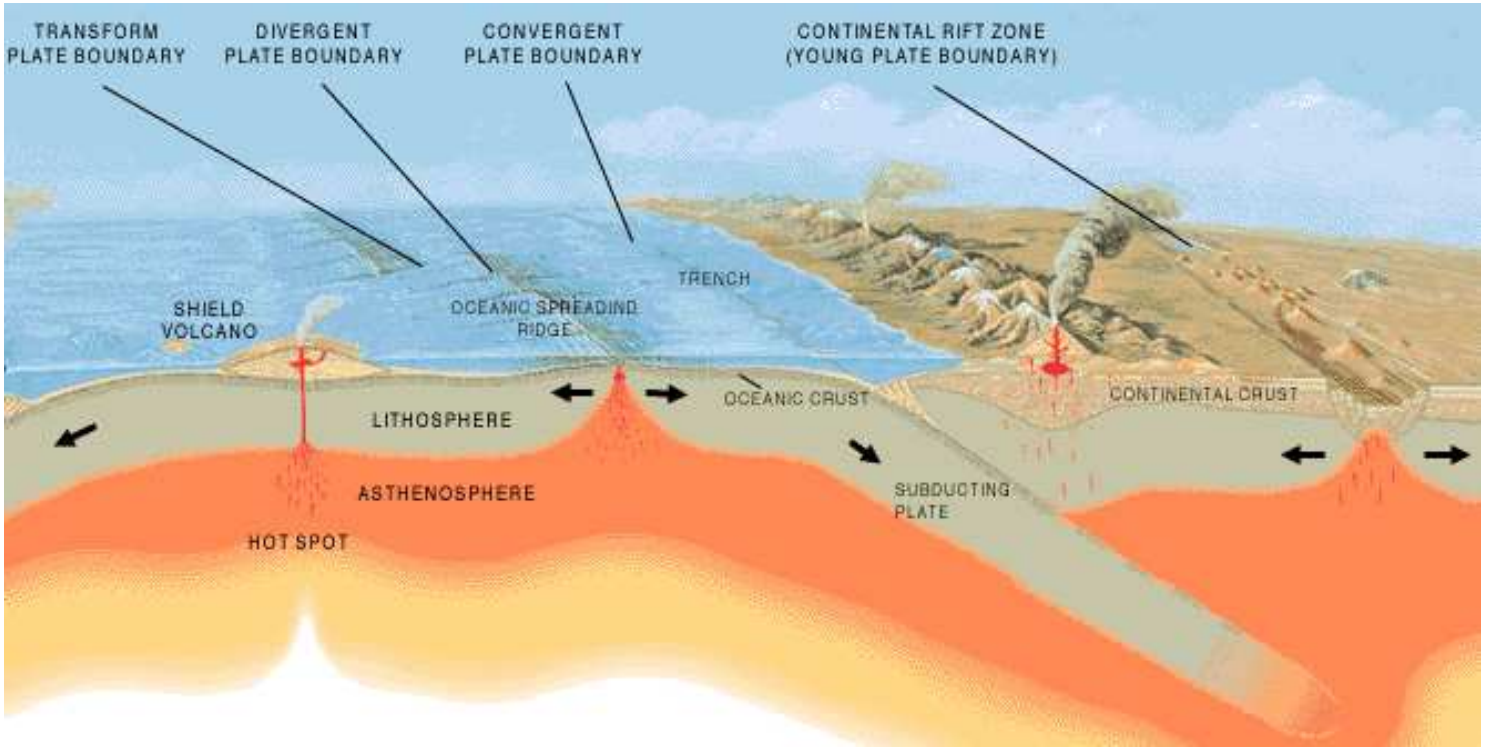
I. Modélisations par réseaux d'automates cellulaires et simulations parallèles du phénomène de subduction-érosion en tectonique des plaques

- rappels sur la tectonique des plaques,
- choix d'une modélisation discrète et SPM,
- présentation des modèles,
- présentation des simulations et de leurs optimisations parallèles,
- résultats et performances,
- conclusion et évolutions.

leduc@lmt.ens-cachan.fr

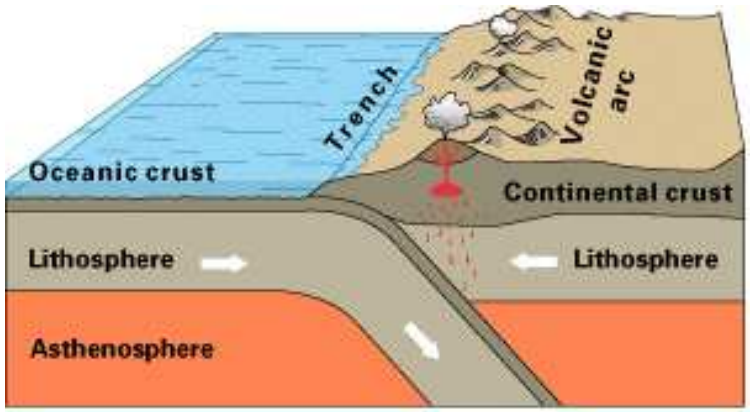
Le contexte géotectonique

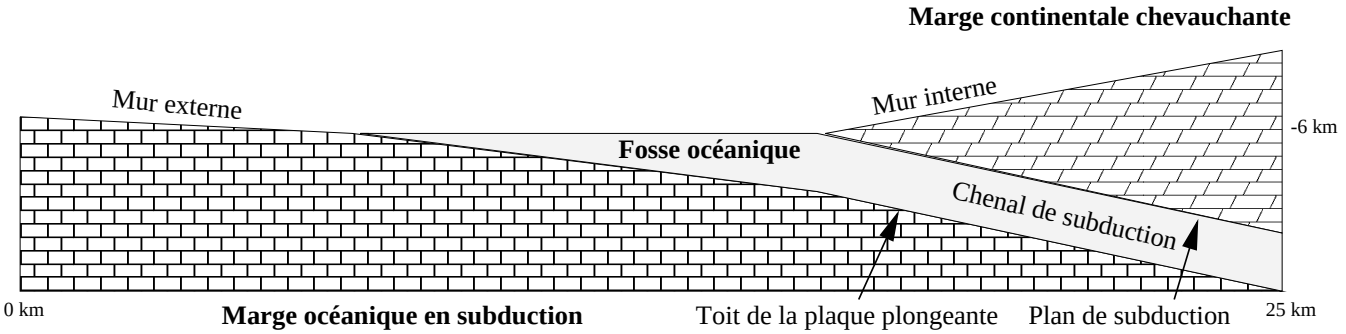
➥le globe terrestre est une structure stratifiée composée d'une multitude de calottes sphériques (les plaques) en mouvement permanent.



leduc@lmt.ens-cachan.fr

➥subduction de marge océanique sous une marge continentale (cas de la fosse d'Amérique centrale, au large de Mexico),





L'existant vs. notre stratégie de modélisation

➥l'existant en matière de modélisation : simulations analogiques expérimentales et simulations numériques avec résolution d'EDP par éléments finis. But : satisfaire des *critères de similarité physique*. Problèmes :
- transferts d'échelles et aspects thermiques,
- indications à petites échelles sur l'épaississement ou l'amincissement.

Comportement de *milieu*, pas de prise en compte des *individualités*.

➥modélisation par systèmes dynamiques discrets. La conjonction de règles élémentaires locales engendre des comportements d'ensemble complexes :
- modélisation de la convection par une méthode LGA,
- modélisations en sismologie, en érosion de terrain...

➥réseau d'automates cellulaires : juxtaposition d'une même règle de

transition, selon un mode opératoire synchrone, sur des cellules placées aux sommets d'un graphe simple orienté, connexe et d-régulier,

localité
uniformité
synchronisation
parallèlisme

leduc@lmt.ens-cachan.fr

Analogies avec le SPM

➥le *Sand Pile Model* 1D :

- physique de l'état granulaire (Bak, Tang et Wiesenfeld - 1980),
- réseau d'automates cellulaires en ligne,
- un automate correspond à une hauteur de pile de sable,
- loi de transition :

$$\text{avec : } \mathbb{I}(n) = \begin{cases} 0 \text{ si } n < 2 \\ 1 \text{ sinon} \end{cases}$$

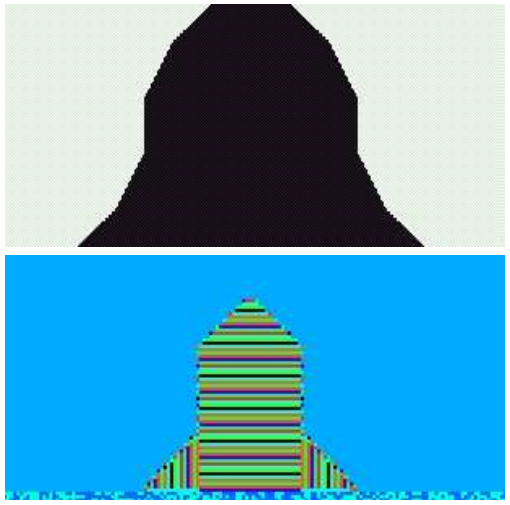
➥étendu au cas bi-dimensionnel :

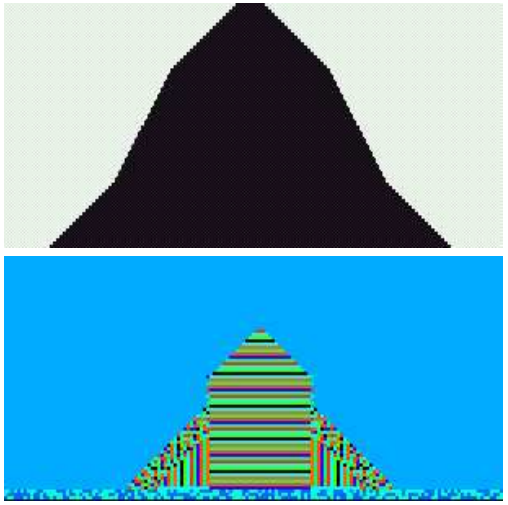
- grille régulière en 2 dimensions d'automates cellulaires,
- individualisation de chaque "grain" (avec attributs de nature et de couleur pour chaque automate),
- loi de transition plus complexe à écrire puisque fonction d'un voisinage plus grand (voisinage de Moore étendu),

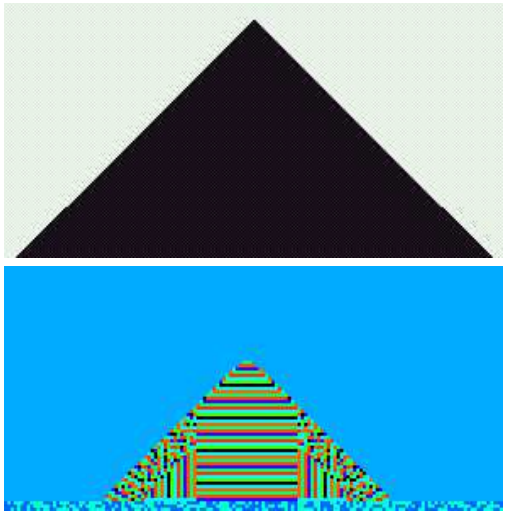
$$C_j^{t+1} = C_j^t - \mathbb{I}(C_j^t - C_{j-1}^t) - \mathbb{I}(C_j^t - C_{j+1}^t) + \mathbb{I}(C_{j-1}^t - C_j^t) + \mathbb{I}(C_{j+1}^t - C_j^t)$$

leduc@lmt.ens-cachan.fr

➥rendus visuels comparés :



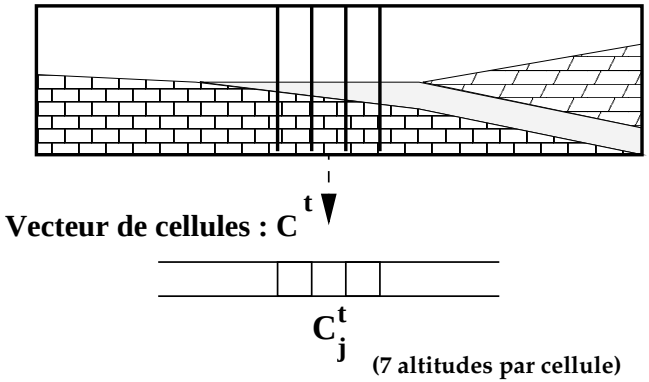




leduc@lmt.ens-cachan.fr

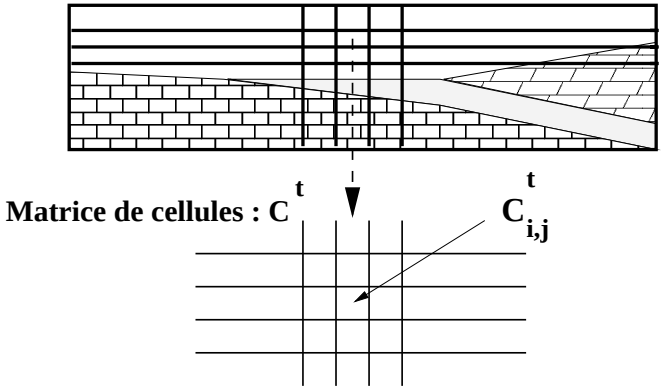
Les modèles

Automate cellulaire 1D



découpage en colonnes de la coupe étudiée

Automate cellulaire 2D



découpage en blocs de la coupe étudiée

démarche ascendante à base de règles d'interactions locales mutuelles.

leduc@lmt.ens-cachan.fr

➥modélisation 1D :

- ligne finie de 1000 AC,
- état d'une cellule : 7 hauteurs de couches, un coefficient de "marche" et un coefficient de "vieillissement" ⇒ nombre fini d'états,
- 2 voisins,
- imbrication de trois "moteurs" correspondant à trois échelles de temps,
- mécanisme additionnel pour la transmission d'une donnée globale ($Coeff_{Translation}$),

➥modélisation 2D :

- grille régulière de 200000 AC,
- état d'une cellule : 6 entiers bornés correspondant à la couleur, la nature, le vieillissement, la "courbure", la "mémoire" et l'altitude de la cellule courante,
- 24 voisins (voisinage de Moore étendu),
- mécanisme de réplication des cellules des bords internes singularisées par un état "frontière" invariant,

leduc@lmt.ens-cachan.fr

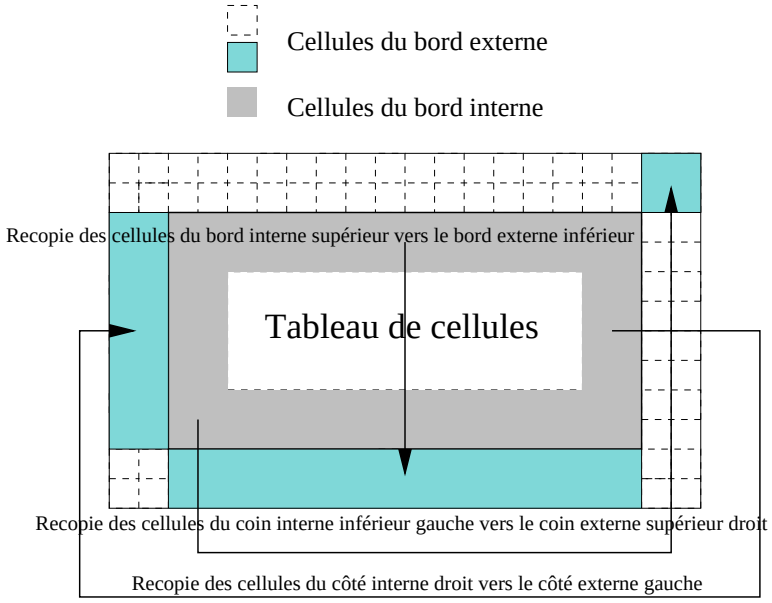
Représentation de $(C^t)_{t\in\mathbb{N}}$ par un "jeu de bascule" entre deux tableaux C et CC.

➥algorithme simplifié 1D

```
répéter
    CC ← C
    C ← F_lente(CC, Coeff_Tr)
    répéter
        CC ← C
        C ← F_moyenne(CC, Coeff_Tr)
        répéter
            CC ← C
            C ← F_rapide(CC)
        jusqu'à condition d'arrêt 1
    jusqu'à condition d'arrêt 2
    saisie conditionnelle d'image
jusqu'à condition d'arrêt 3
```

➥algorithme simplifié 2D

```
répéter
    CC ← C
    réplication des bords de CC
    C ← F(CC)
jusqu'à condition d'arrêt
```



➥3 "moteurs" imbriqués du modèle 1D :
(on les retrouve en 2D à l'exception de la translation du front de la marge océanique)

- échelle de temps lente :
  subduction et génération des "brisures de pente",
  dégradation et érosion de la base de la plaque continentale,
  émission ou non du signal $Coeff_{Translation}$,
- échelle de temps intermédiaire :
  translation éventuelle du front de la marge océanique,
  nivellement de la fosse océanique,
  affaissements de terrain au sein de la plaque continentale,
- échelle de temps rapide :
  avalanches superficielles sur la plaque continentale.

leduc@lmt.ens-cachan.fr

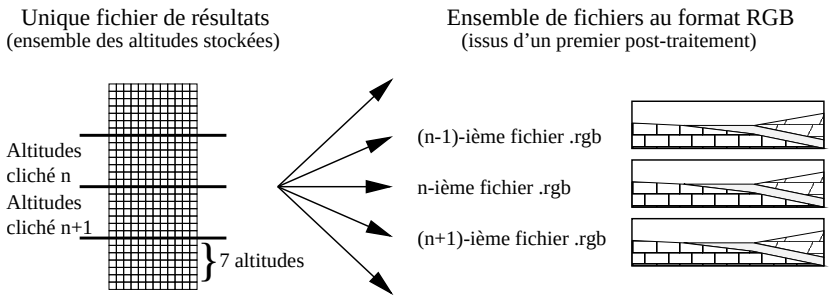
Les simulations

➥de la nécessité de développer nos propres outils...

la plate-forme `Cellular` (= langage `Cellang`, compilateur `cellc`, machine virtuelle `avcam` et viewer `cellview`) ne correspond pas à nos besoins.

➥et de bien distinguer les traitements des post-traitements graphiques.

➧en 1D



Raw → GIF → MPV

➧en 2D

conversion par `convert` de fichiers SunRaster ou Raw au format GIF, puis assemblage par `dmconvert` des images GIF en une animation MPEG-1 vidéo

➥stratégie (commune) d'optimisation parallèle :

- architectures MIMD (asynchrones, distribuées avec bus ou réseau de communication) - modèle par échange de messages
- méthode de décomposition de domaine

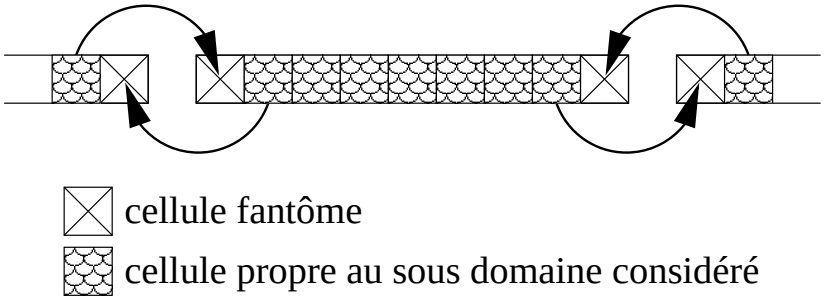
⇒ exploite la localité des données,

⇒ permet une résolution concurrente des sous-problèmes

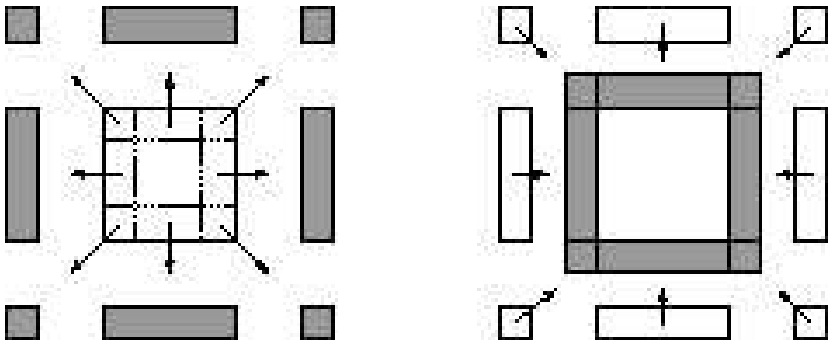
- méthode de décomposition de domaine avec recouvrement

⇒ problème : stockage en local d'informations non locales...

en 1D



en 2D



➥algorithme simplifié :

- recouvrir les temps de communication par des temps de calcul,
- principes communs aux deux fonctions de transition globales :

1. Envoi non bloquant des bords internes,
2. Mise à jour du sous-domaine (bords internes et externes exclus),
3. Réception bloquante des bords externes,
4. Réévaluation des états des bords internes.

avec quelques sauvegardes périodiques de la configuration du réseau d'automates cellulaires...

leduc@lmt.ens-cachan.fr

➥gestion des entrées-sorties

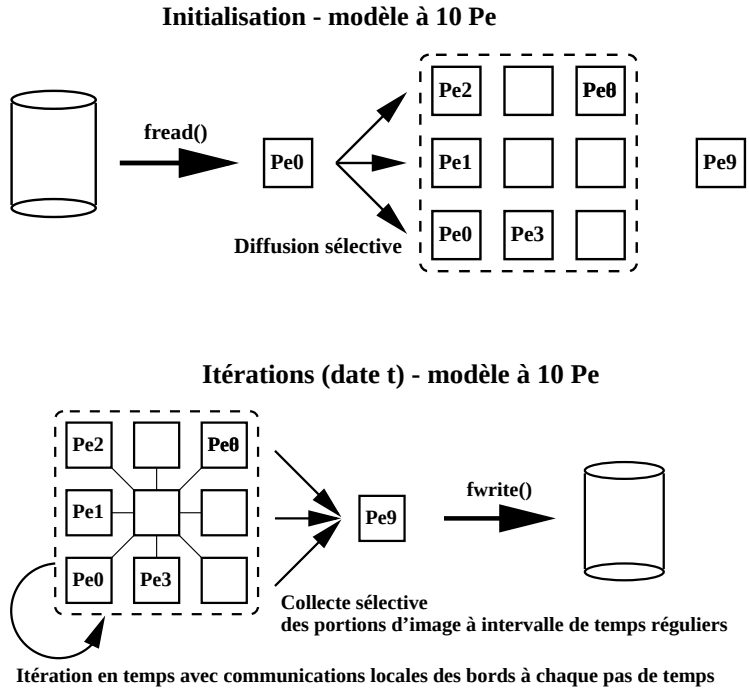
désynchronisation d'un processus "superviseur" du reste des "itérateurs".

◗Rôle du "superviseur" en 1D :

(c'est l'"itérateur" de gid nul)

1. il diffuse les paramètres d'initialisation,
2. il collecte, somme et rediffuse les valeurs de $Coeff_{Translation}$ (pvm_reduce, pvm_bcast),
3. il collecte périodiquement les attributs d'altitudes sur les sous-domaines (pvm_gather) et les stocke sur disque.

◗Rôle du "superviseur" en 2D :

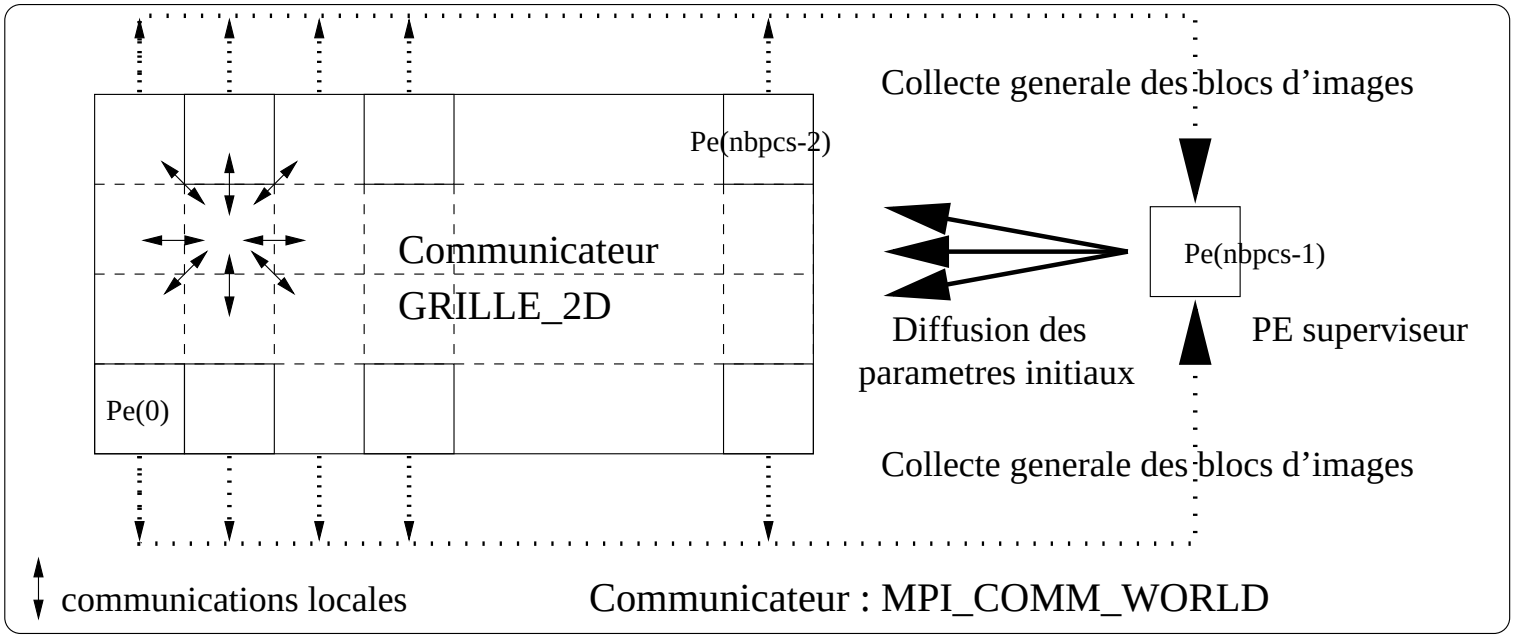


➥spécificités de la parallélisation 2D :

de l'intérêt d'utiliser MPI et non plus PVM.

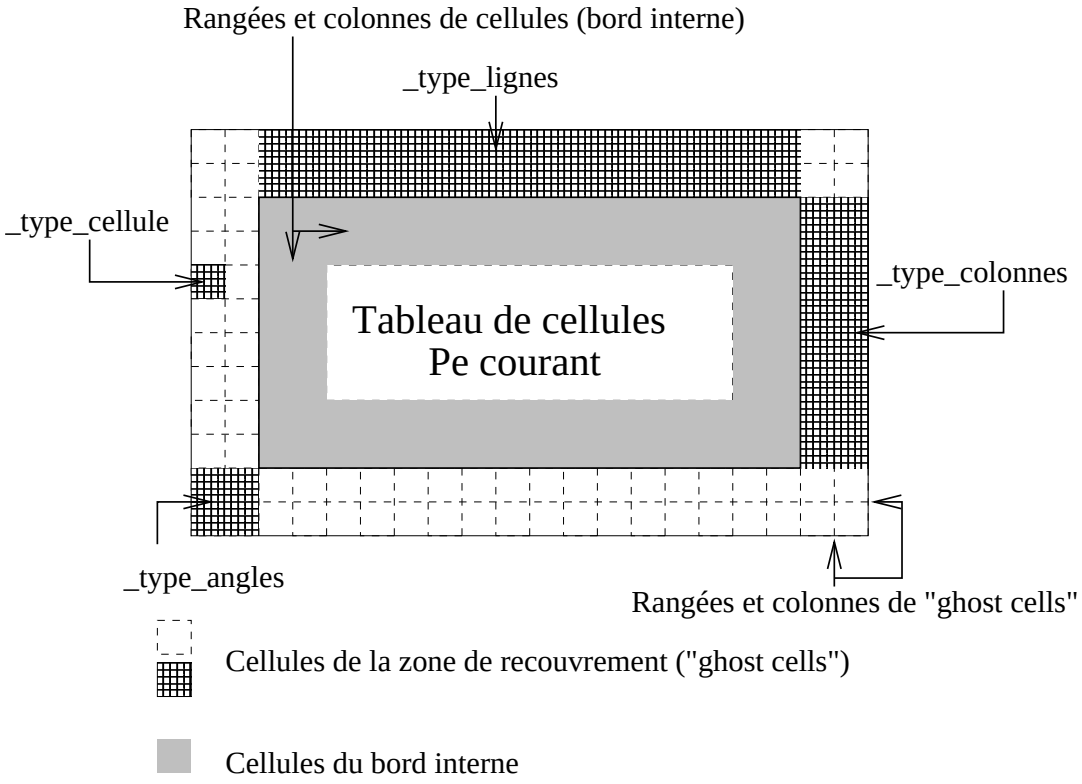
➧topologie cartésienne de processus :

- partitionnement de MPI_COMM_WORLD en deux sous-communicateurs,
- les "itérateurs" sont disposés sur une topologie cartésienne virtuelle régulière à 2D (MPI_Cart_create())



- types de données dérivés :

- le type _type_cellule (type hétérogène, MPI_Type_struct()),
- les types _type_lignes, _type_angles, _type_colonnes (types de données homogènes vectoriels à pas constant, MPI_Type_vector()),



leduc@lmt.ens-cachan.fr

Émission des résultats sur les "itérateurs" : il suffit de récupérer la valeur du champ `couleur` de la configuration courante
⇒ type _type_bloc_image (type homogène à pas variable où le tableau des cellules est considéré comme un tableau d'octets, MPI_Type_indexed() - tailles des champs alignés en mémoire obtenues par MPI_Type_extent()),

Extraction et retournement des pixels d'un Pe quelconque

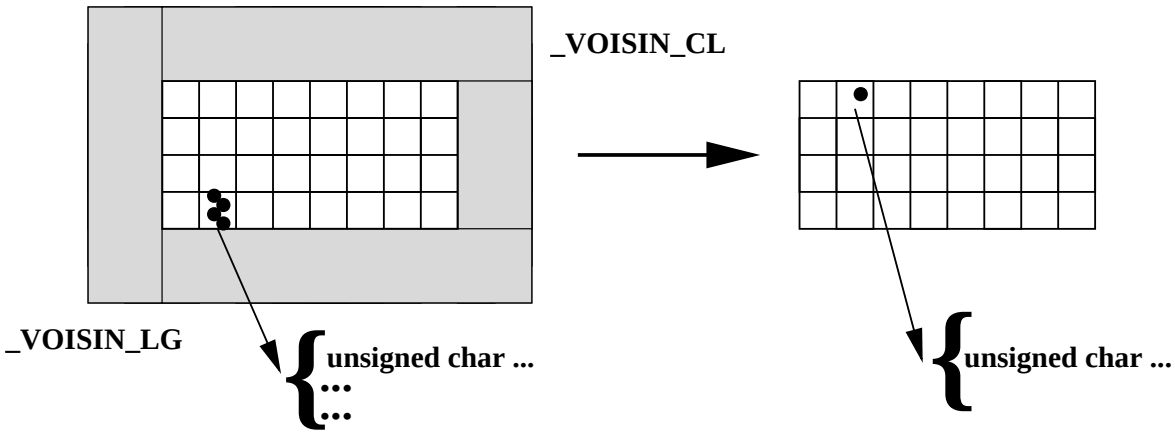


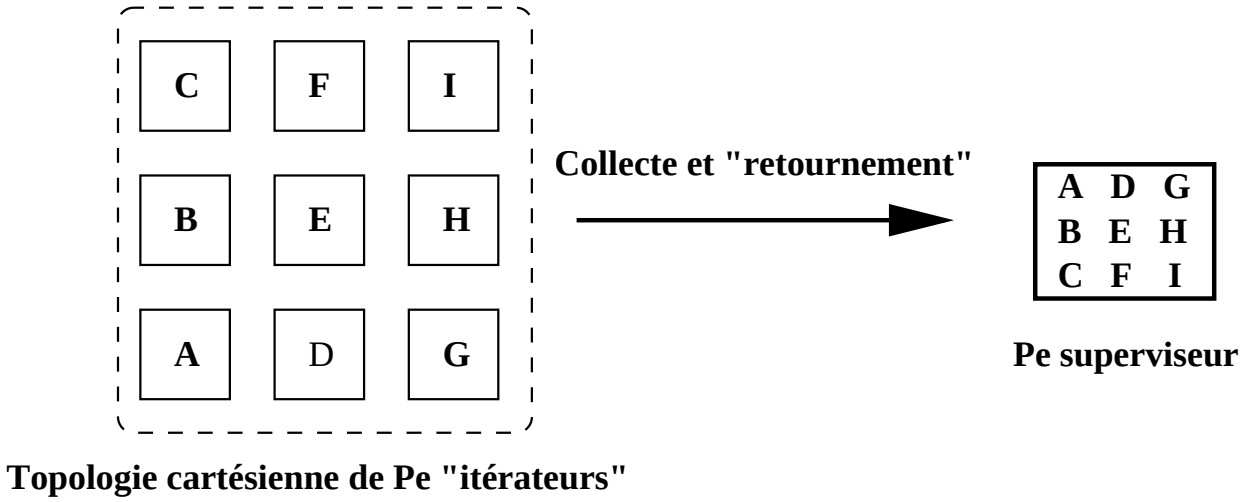
Tableau de cellules

Tableau de pixels
(retournement de l'image)

leduc@lmt.ens-cachan.fr

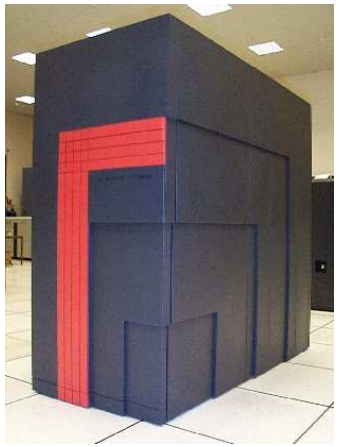
collecte des résultats sur le "superviseur" : _type_image_globale (type homogène à pas constant, primitive MPI_Type_vector()),

Collecte et retournement des blocs d'image sur le superviseur



Résultats et performances

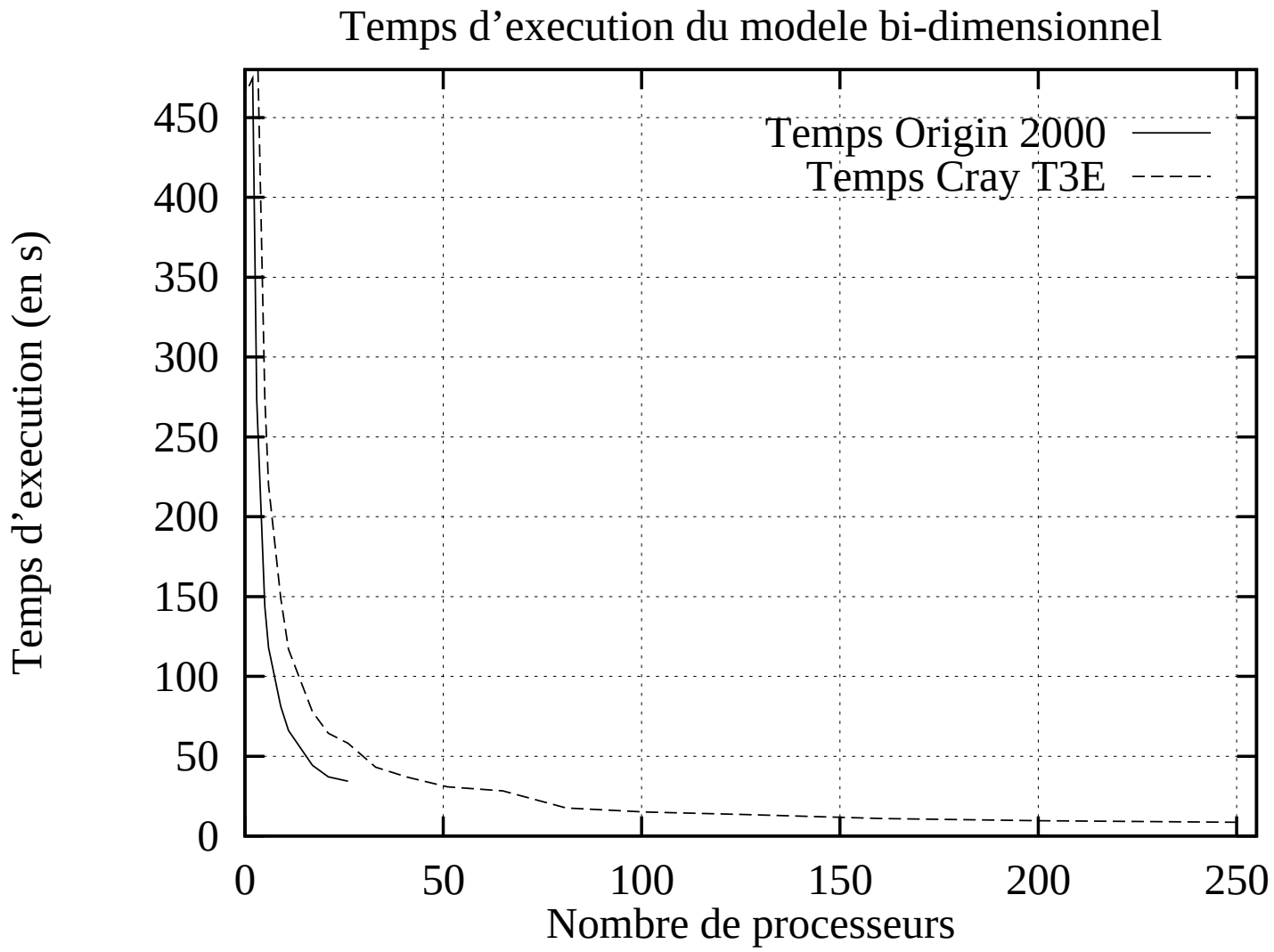
- le Cray t3e :

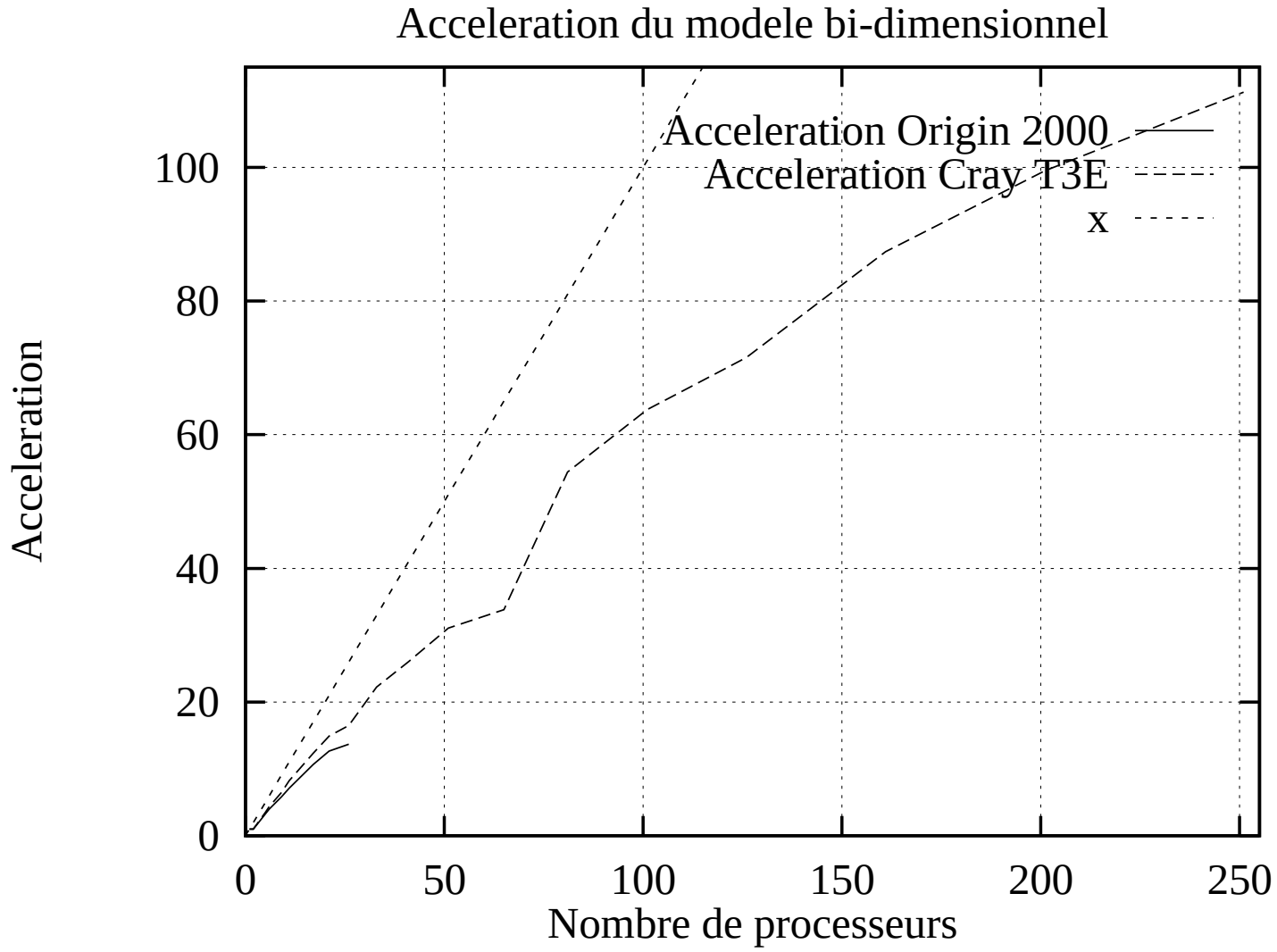


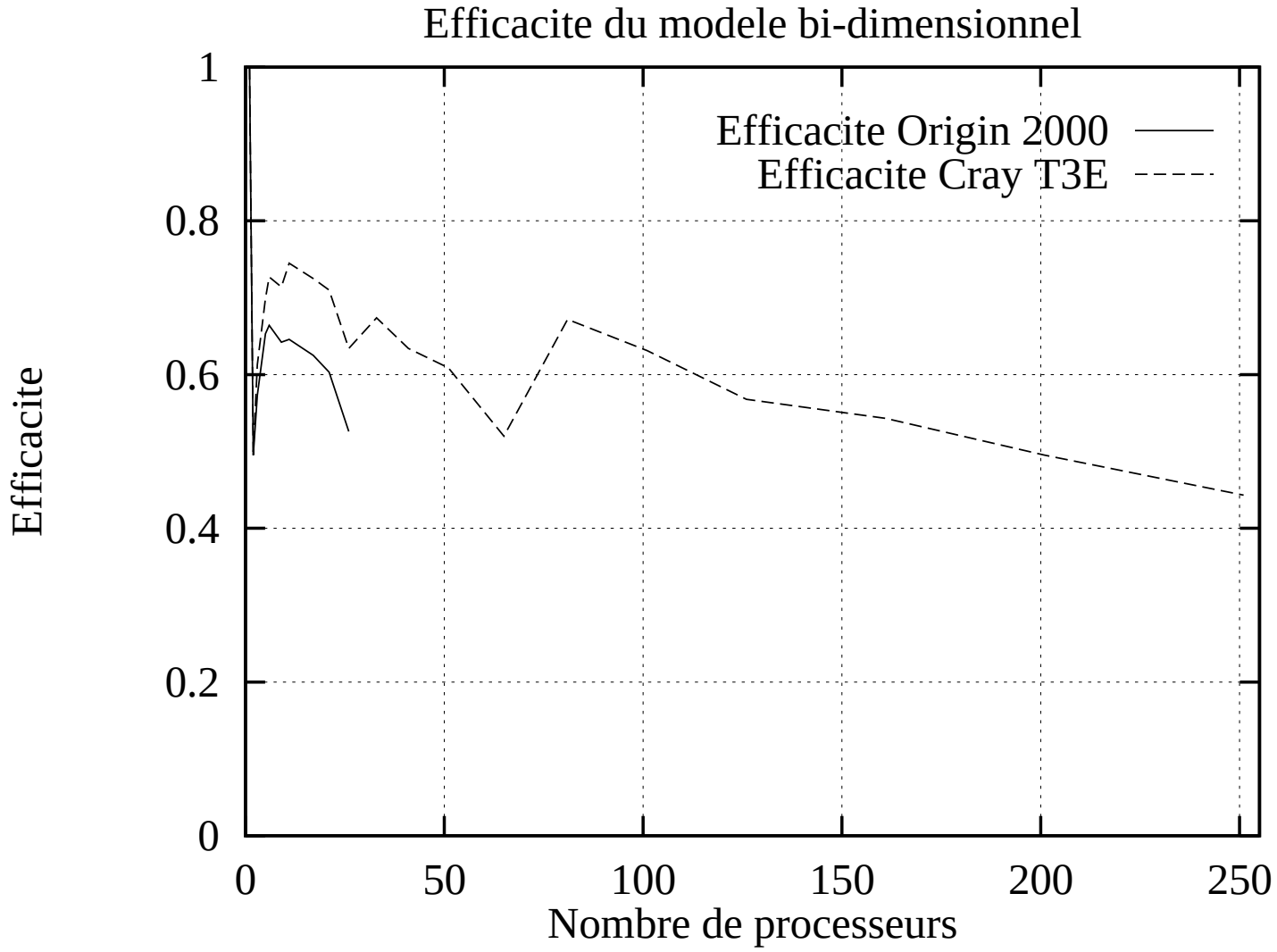
256 Dec Alpha 21164 - 600 Mflop/s,
caches de données de 8 ko et 96 ko,
latence : 1 μs,
bande passante : 480 Mo/s,

- l'Origin 2000 :

32 R10000 - 390 Mflop/s,
caches de données de 32 ko et 4 Mo,
latence : 773 ns,
bande passante : 780 Mo/s,


Temps d'execution du modele bi-dimensionnel
Temps Origin 2000
Temps Cray T3E
Temps d'execution (en s)
Nombre de processeurs
0
50
100
150
200
250
300
350
400
450


Acceleration du modele bi-dimensionnel
Acceleration Origin 2000
Acceleration Cray T3E
x
Acceleration
Nombre de processeurs
0
20
40
60
80
100
0
50
100
150
200
250



leduc@lmt.ens-cachan.fr

➥minimiser le ratio taille des interfaces / taille des sous-domaines :

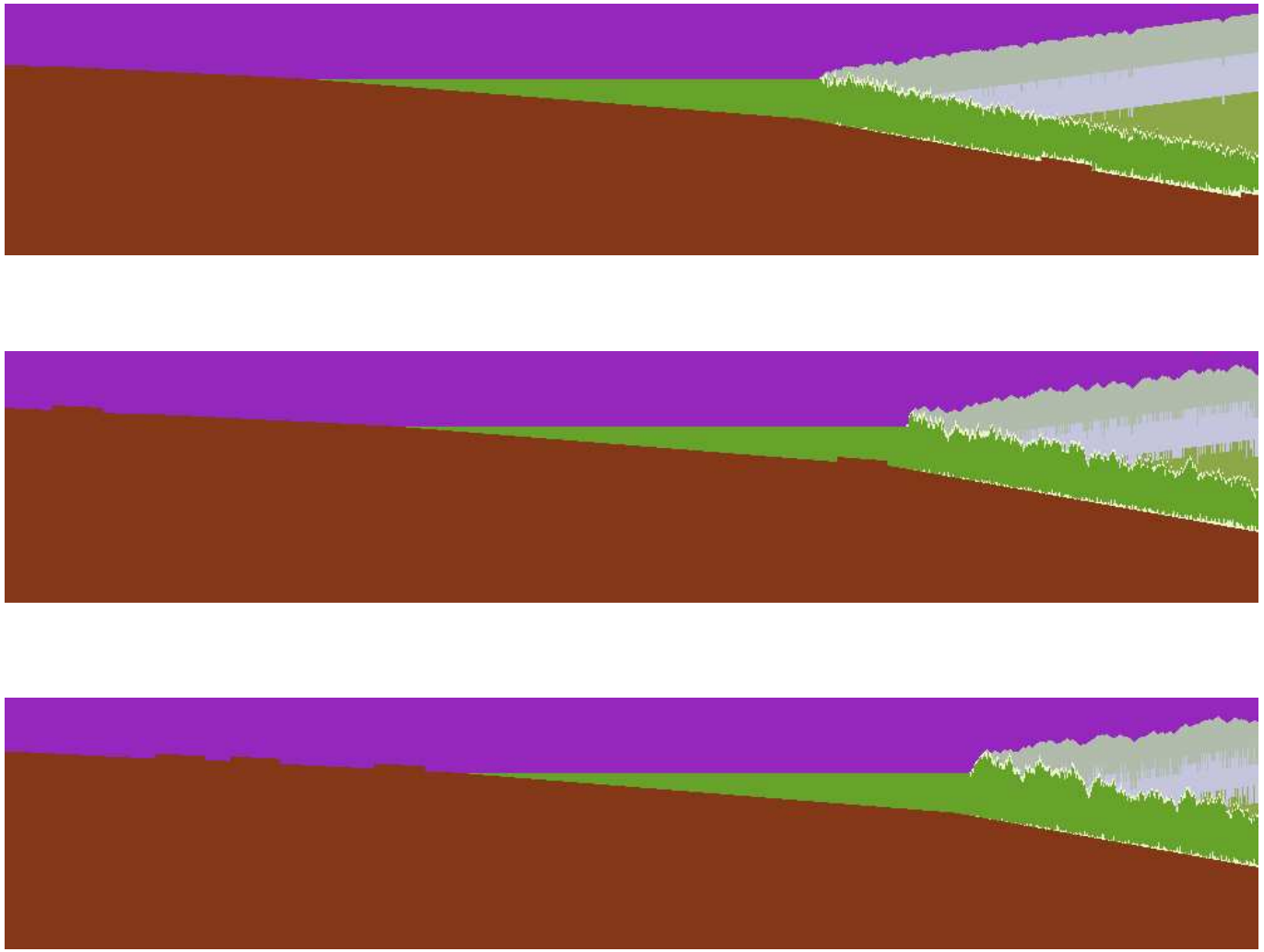
nb proc	Topologies associées (nb de processeurs en ligne * nb de processeurs en colonnes) Ratios associés (nb de cellules aux interfaces sur le nb total de cellules d'un sous-domaine)											
2	1 * 2 4,23	2 * 1 **2,74**										
4	1 * 4 7,78	2 * 2 4,61	4 * 1 **3,50**									
5	1 * 5 9,45	5 * 1 **3,88**										
8	1 * 8 14,14	2 * 4 8,14	4 * 2 5,36	8 * 1 **5,00**								
10	1*10 17,00	2 * 5 9,81	5 * 2 **5,73**	10*1 **5,73**								
16	2 * 8 14,48	4 * 4 8,87	8 * 2 **6,83**									
20	1*20 28,86	2*10 17,33	4 * 5 10,52	5 * 4 9,22	10*2 **7,54**	20*1 9,22						
25	1*25 33,60	5 * 5 **10,87**	25*1 **10,87**									
32	4 * 8 15,15	8 * 4 **10,28**										

leduc@lmt.ens-cachan.fr

nb proc	Topologies associées (nb de processeurs en ligne * nb de processeurs en colonnes) Ratios associés (nb de cellules aux interfaces sur le nb total de cellules d'un sous-domaine)											
40	1*40 44,67	2*20 29,14	4*10 17,98	5 * 8 15,48	8 * 5 11,91	10*4 **10,97**	20*2 **10,97**	40*1 15,48				
50	1*50 50,20	2*25 33,86	5*10 18,30	10*5 **12,59**	25*2 **12,59**	50*1 18,30						
64	8 * 8 **16,47**											
80	2*40 44,89	4*20 29,70	8*10 19,25	10*8 17,11	20*4 **14,27**	40*2 17,11						
100	1 * 100 66,80	2*50 50,40	4*25 34,38	5*20 29,97	10 * 10 19,87	20*5 **15,82**	25*4 **15,82**	50*2 19,87	100*1 29,97			
125	5*25 34,64	25*5 **17,36**	125*1 34,64									
160	4*40 45,32	8*20 30,79	20*8 **20,18**	40*4 **20,18**								
200	1 * 200 80,08	2 * 100 66,93	4*50 50,79	5*40 45,53	8*25 35,40	10 * 20 31,32	20 * 10 22,84	25*8 **21,63**	40*5 **21,63**	50*4 22,84	100*2 31,32	200*1 45,53
250	5*50 50,98	10 * 25 35,90	25 * 10 **24,24**	50*5 **24,24**	125*2 35,90	250*1 50,98						

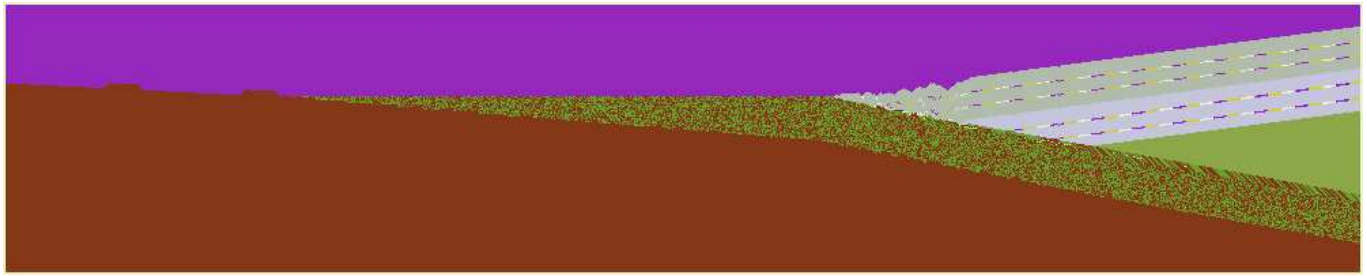
leduc@lmt.ens-cachan.fr

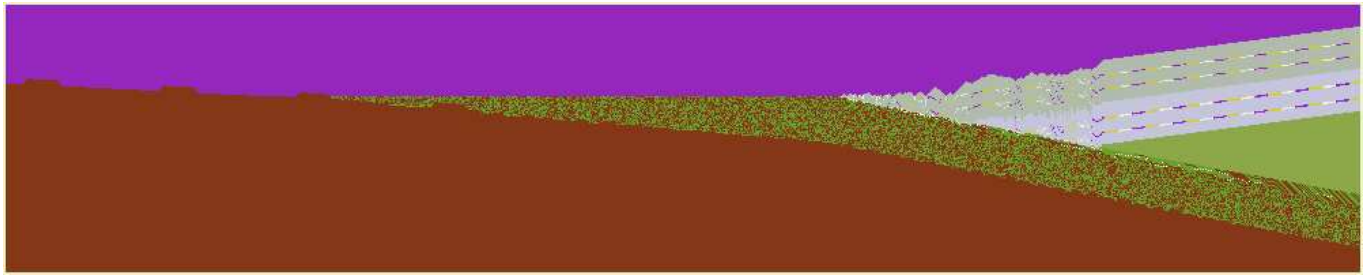
➥Copies d'écran de la simulation 1D :

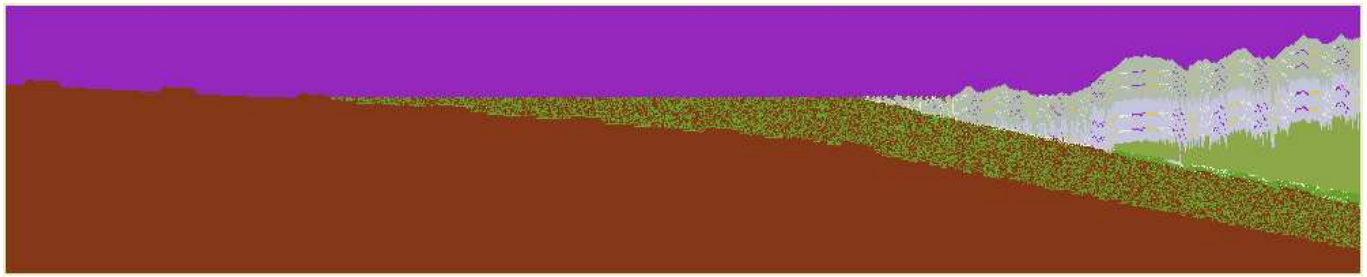


leduc@lmt.ens-cachan.fr

➥Copies d'écran de la simulation 2D :







leduc@lmt.ens-cachan.fr

leduc@lmt.ens-cachan.fr

Conclusions

- les vertus de l'approche discrète,
- la 1D intermédiaire facilitant le passage à la 2D,
- du bon "rendement" du modèle 1D en terme de rendu-visuel et de la simulation 2D en terme d'optimisation parallèle,
- justification de l'intérêt des nombreuses fonctionnalités de MPI,
- des simulations à valeur didactique pour l'illustration d'un phénomène géotectonique.

Évolutions

- analyser le comportement des simulations,
- plate-forme logicielle spécifique $\mathcal{LAC}$, envisager la 3D...

leduc@lmt.ens-cachan.fr

II. Écriture d'un outil générique pour la manipulation de réseaux d'automates cellulaires

- contexte, motivations et enjeux,
- les atouts de la programmation orientée objets,
- cas particulier d'une parallélisation sur architectures à mémoire partagée,
- conclusion et évolutions.

leduc@lmt.ens-cachan.fr

Contexte, motivations et enjeux

➥les outils classiques (`Cellular`, `Cellsim`, `CAPOW`, `StarLogo`...) ne permettent pas de simuler et représenter tous les types de RAC,

➥au cours des premiers développement de la plate-forme $\mathcal{LAC}$ quelques grands principes communs ont été identifiés ($\mathcal{LAC}$: langage dédié aux rac2d, traducteur en Lex et Yacc pour générer du C avec appel à des primitives MPI, frontal graphique en Perl/Tk),

➥but : simplifier le travail d'un expérimentateur futur...

leduc@lmt.ens-cachan.fr

Réutilisation et factorisation

un RAC : un système discret + un mode opératoire,

- représentation logique d'une réalité géométrique physique sous forme d'un "graphe simple orienté, connexe et d-régulier" (typiquement $\mathbb{Z}^n$; [Róka, 94] propose une généralisation aux graphes de Cayley),

- juxtaposition d'une même règle homogène de transition locale à tous les automates du réseau qui ont tous le même voisinage (d'influences) indépendamment du temps,
- application itérée selon un mode opératoire synchrone de ces règles de transition.

➥mise en place des notions de réseau, voisinage, initialisation, transitions globale et locale et (accessoirement) représentation graphique,

➥seuls l'initialisation, la transition locale et le mode de représentation graphique sont réellement RAC-dépendant !

leduc@lmt.ens-cachan.fr

Polymorphisme et "liaison retardée" : la POO comme élément de réponse

➥Polymorphisme : faculté d'une méthode donnée à pouvoir être appliquée à divers objets instanciant chacun des classes différentes,
➥Liaison dynamique : à l'exécution (et non à la compilation comme en C++), le code de la méthode invoquée par envoi de message est recherché par parcours ascendant de la hiérarchie de classes (du graphe d'héritage) de l'objet récepteur. La première méthode trouvée dans ce parcours "ascendant" de graphe, dont la signature correspond à celle du message, est exécutée.
➥Conséquences : factorisation maximale et "instanciation dynamique" d'objets (dont on a aucune connaissance a priori ⇒ chargement dynamique de nouvelles classes !) à l'exécution à l'aide d'une instruction du type :

```
objet = (AcGénérique)Class.forName("classeDApplication").newInstance();
```


leduc@lmt.ens-cachan.fr

Le code générique

➥notion de “contrat” à implémenter :

```
1   import java.awt.image.WritableRaster;
2   public interface Ac
3   {
4       void initialiser (int indice,Proprietes propriétés,Voisinage voisins);
5       void transitionLocale();
6       void afficher(WritableRaster tableauPixels);
7   }
```

➥code générique du réseau :

```
1   public class Rac
2   {
3       private Proprietes propriétés;
4       private Ac[] configuration0,configuration1;
5       private Voisinage voisins0,voisins1;
6       Rac ([...])
7       {
8           for (int i=0 ; i<nbCellules ; i++) {
9               configuration0[i]=(Ac)Class.forName(propriétés.classeDApplication).newInstance();
10              configuration1[i]=(Ac)Class.forName(propriétés.classeDApplication).newInstance();
11          }
12          voisins0 = new Voisinage(propriétés.tabDimensions,configuration0);
13          voisins1 = new Voisinage(propriétés.tabDimensions,configuration1);
14      }
15      public void initialiser () throws DimensionException
16      {
```

```
17          for(int i=0 ; i<nbCellules ; i++) {
18              configuration0[i]. initialiser (i,propriétés,voisins1);
19              configuration1[i]. initialiser (i,propriétés,voisins0);
20          }
21      }
22      public void transitionGlobale()
23      {
24          permuterLogiquement(configuration0,configuration1);
25          for(int i=0 ; i<nbCellules ; i++)
26              configuration1[i].transitionLocale();
27      }
28      private void afficher()
29      {
30          for(int i=0 ; i<nbCellules ; i++)
31              configuration1[i]. afficher (tableauPixels);
32      }
33  }
```

➥code générique du voisinage :

```
1   public class Voisinage
2   {
3       /* les méthodes de cette classe permettent de "mapper" un vecteur sur un
4        * tableau multi-dimensionnel et réciproquement */
5       private Ac[] réseau;
6       private int[] tabDimensions;
7       Voisinage(int[] tabDimensions,Ac[] réseau)
8       {
9           // corps du constructeur ...
10      }
11      public Ac getVoisin(int indice,int[] tabDécalages)
```


leduc@lmt.ens-cachan.fr

```
12        {
13            int [] coordonnées = indice2coordonnées(indice);
14            for (int i=0 ; i<dimension ; i++)
15              coordonnées[i]=(coordonnées[i]+tabDécalages[i]+tabDimensions[i])%tabDimensions[i];
16            return réseau[ coordonnées2indice(coordonnées) ];
17        }
18    }
```

➥la partie relative au traitement graphique (conversationnel ou post-traitement), à la lecture des *Properties*, au lancement de l'application...(et, plus tard, la gestion du *multithreading*) est aussi écrite de manière "générique".

Ce qui reste à la charge de l'expérimentateur...

➥génération du fichier des *Properties* :

```
1   classeDApplication=Fdf
2   tabDimensions=200,200
3   trtGraphique=post-traitement
4   nbc=200
5   nbl=200
6   grossissementX=1
7   grossissementY=1
8   periodeEntre2Cliches=3
9   nombreDeCliches=200
10  nbThreads=20
```

➥écriture de la classe chargée d'implémenter le "contrat" :

```
1   class Fdf implements Ac
2   {
3       int état;
4       private int x,y;
5       private Fdf N,E,courant,S,W;
6       /*
7        * Selon la valeur de "état", la portion boisée qui correspond à la
8        * cellule courante est : soit complètement consumée (0), soit en train de
9        * bruler (13 -> 1), soit indemne (14).
10       */
11
12      public void initialiser(int indice,Proprietes propriétés,Voisinage voisins)
13      {
```

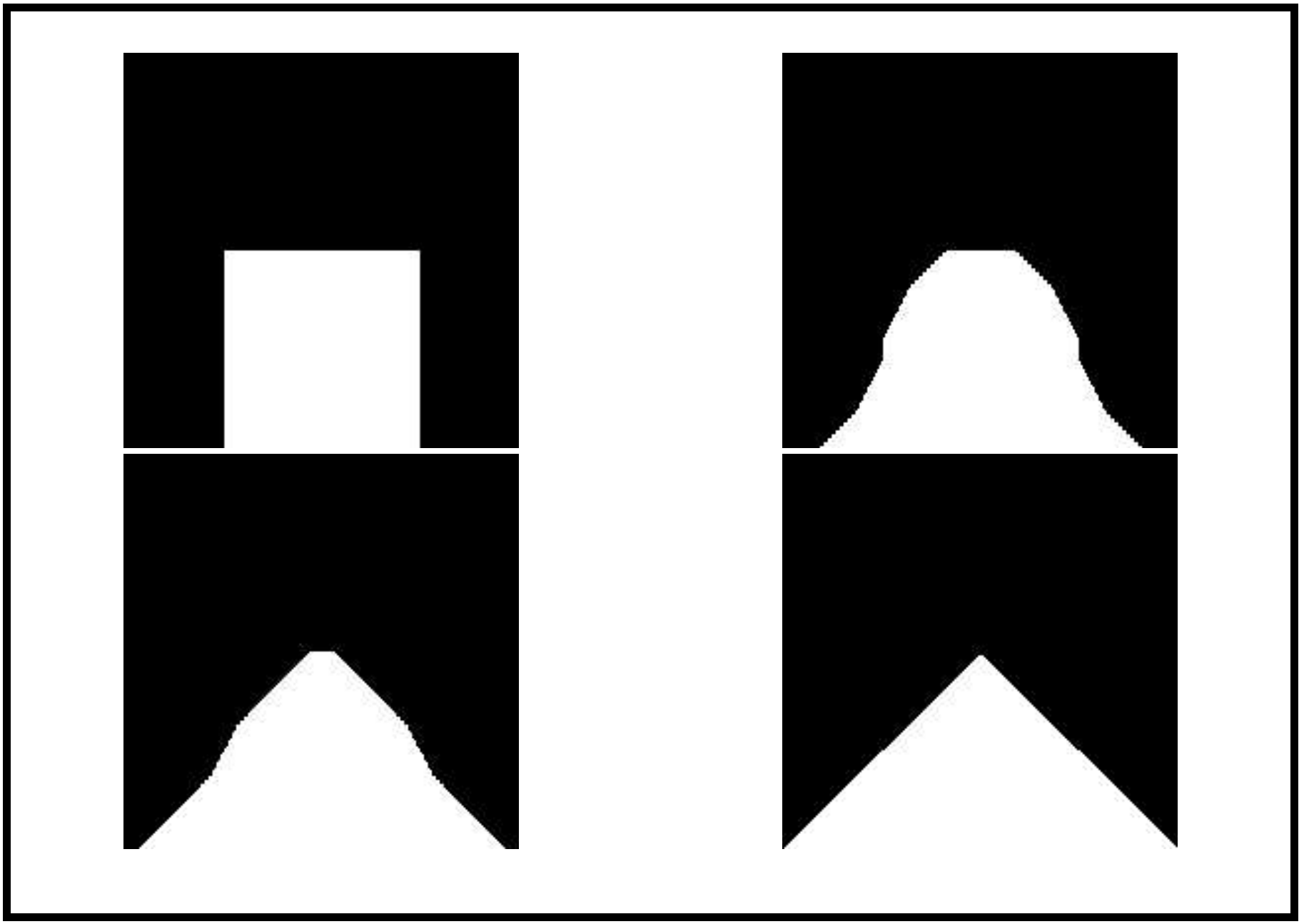
```
14          x = voisins.getCoordonnées(indice,0);
15          y = voisins.getCoordonnées(indice,1);
16
17          if ( voisins.estSurUnBord(new int[] {x,y}) )
18              état = 0;
19          else if (x == 100)
20              état = 13;
21          else état = (aléatoire.nextInt(100) < 60) ? 14 : 0;
22
23          N = (Fdf) voisins.getVoisin(indice,new int[] {0,1});
24          W = (Fdf) voisins.getVoisin(indice,new int[] {-1,0});
25          courant = (Fdf) voisins.getVoisin(indice,new int[] {0,0});
26          E = (Fdf) voisins.getVoisin(indice,new int[] {1,0});
27          S = (Fdf) voisins.getVoisin(indice,new int[] {0,-1});
28      }
29
30      public void transitionLocale()
31      {
32          if ((courant.état >= 1 && courant.état <= 13) ||
33              (courant.état == 14 &&
34               (N.état == 13 || E.état == 13 ||
35                S.état == 13 || W.état == 13)))
36              état = courant.état-1;
37          else état = courant.état;
38      }
39
40      public void afficher(WritableRaster tableauPixels)
41      {
42          int rouge=0 , vert=0 , bleu=0;
43          switch(état) {
44          case 0:
```


leduc@lmt.ens-cachan.fr

```
45                  bleu = 255;
46                  break;
47              case 14:
48                  vert = 255;
49                  break;
50              default:
51                  rouge = (255 * (13-état)) / 13;
52              }
53              tableauPixels.setPixel(x,y,new int[]{rouge,vert,bleu});
54          }
55      }
```

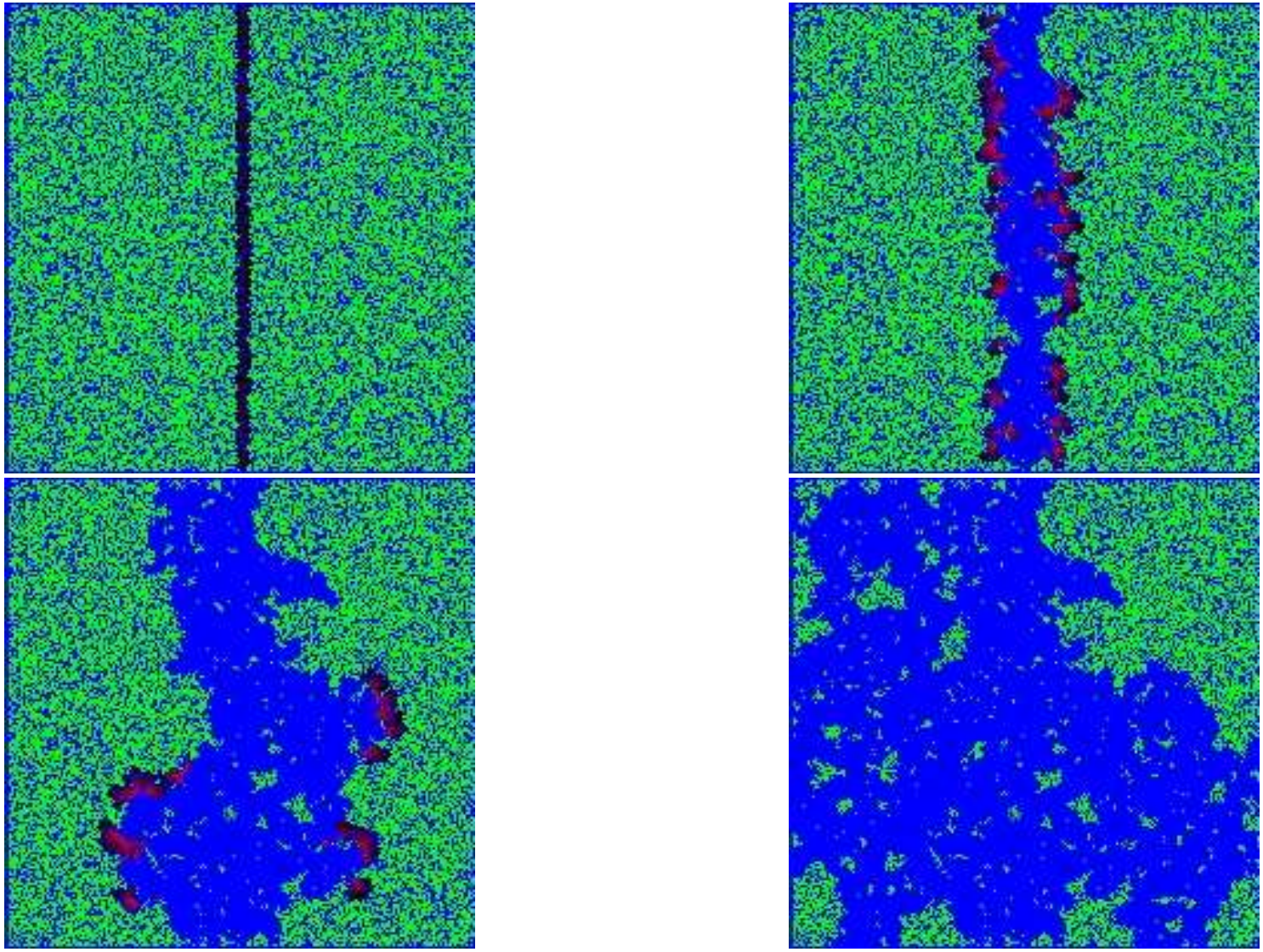
➥quelques illustrations : Spm 1D, feux de forêt, flocon...

Séminaire interne du LMT - 8 février 2001



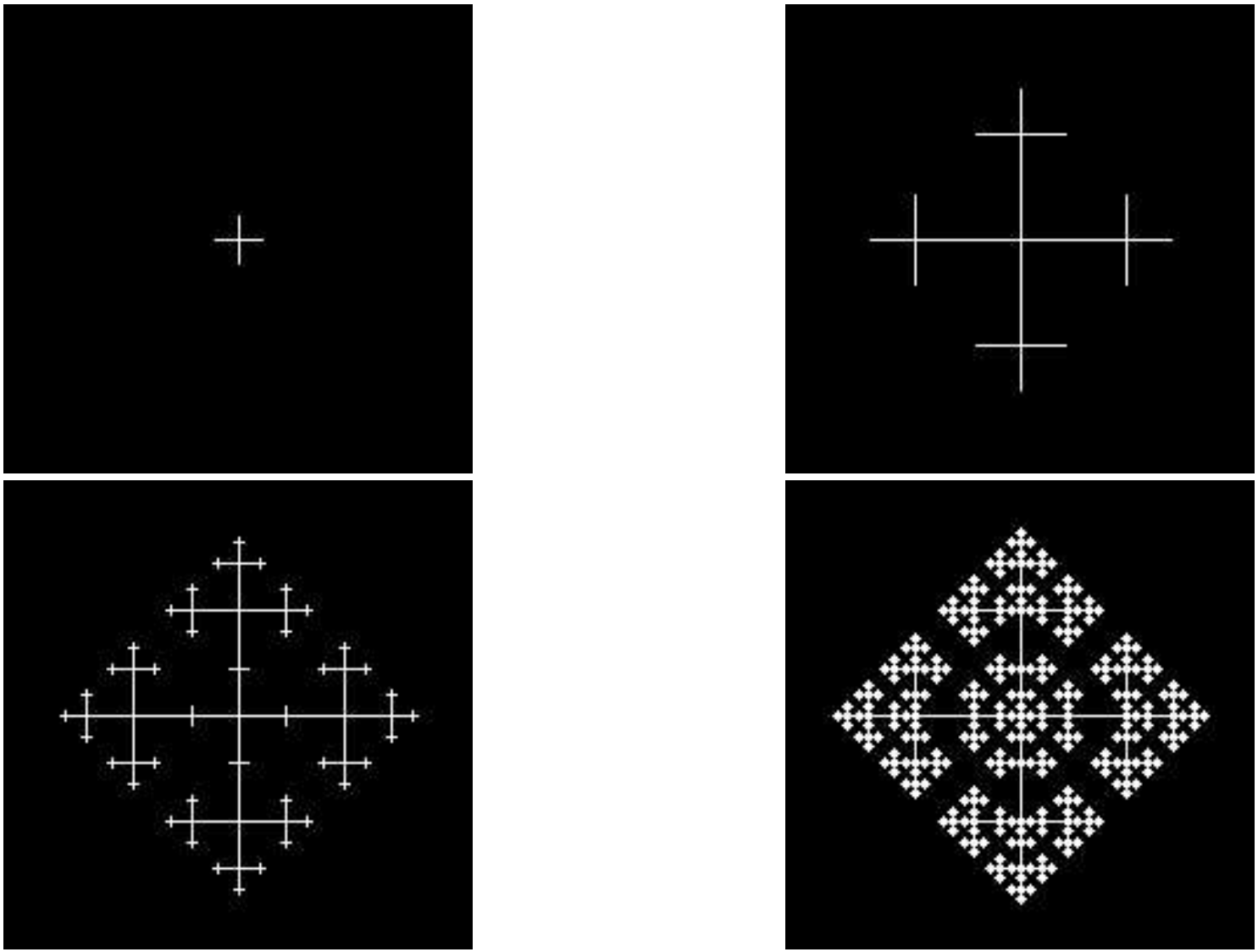
leduc@lmt.ens-cachan.fr

Séminaire interne du LMT - 8 février 2001



leduc@lmt.ens-cachan.fr

Séminaire interne du LMT - 8 février 2001



leduc@lmt.ens-cachan.fr

Parallélisation du simulateur générique - de l'intérêt des *native threads*

➥restriction au cas des architectures à mémoire partagée,

➥VM Java de bacchus : JDK 1.2.2, *green threads*, *mipsjit*,

"*Green threads are user-level threads, implemented within a single Unix process, running on a single processor. This release allows the use of native threads (implemented using POSIX threads, or pthreads), therefore taking full advantage of multiprocessors*"

➥stratégie employée :

◗décomposition par blocs d'automates consécutifs, de l'ensemble du réseau (pas de notion de localité ni d'exploitation du voisinage... toutes les données sont globalement adressables),

◗applications itérées concurrentes des fonctions de transition "globale" pour chaque sous-domaine,

- synchronisation globale à chaque pas de temps.

- en ce qui concerne l'implémentation :
- mise au point d'une classe générique `RacAux` :

```
 1  class RacAux extends Thread
 2  {
 3      RacAux([...])
 4      {
 5          // lancement automatique du thread courant :
 6          start();
 7      }
 8      public void run()
 9      {
10          initialiser ();
11          compteurCliché++;
12          afficher ();
13          for (int compteur=1 ; compteurCliché<propriétés.nombreDeCliches ; compteur++) {
14              Rac.incrementerRDV(1);
15              synchronized (this) {
16                  try {
17                      wait();
18                  } catch (InterruptedException e) {}
19              }
20              transitionGlobale();
21              if ((compteur%propriétés.periodeEntre2Cliches) == 0) {
22                  compteurCliché++;
23                  afficher ();
```

```
24              }
25          }
26      }
27      private void afficher()
28      {
29          for(int i=début ; i<fin ; i++)
30              configuration1[i]. afficher (tableauPixels);
31      }
32      private void  initialiser ()
33      {
34          // même  traitement ...
35      }
36      public void transitionGlobale()
37      {
38          // même  traitement ...
39      }
40  }
```

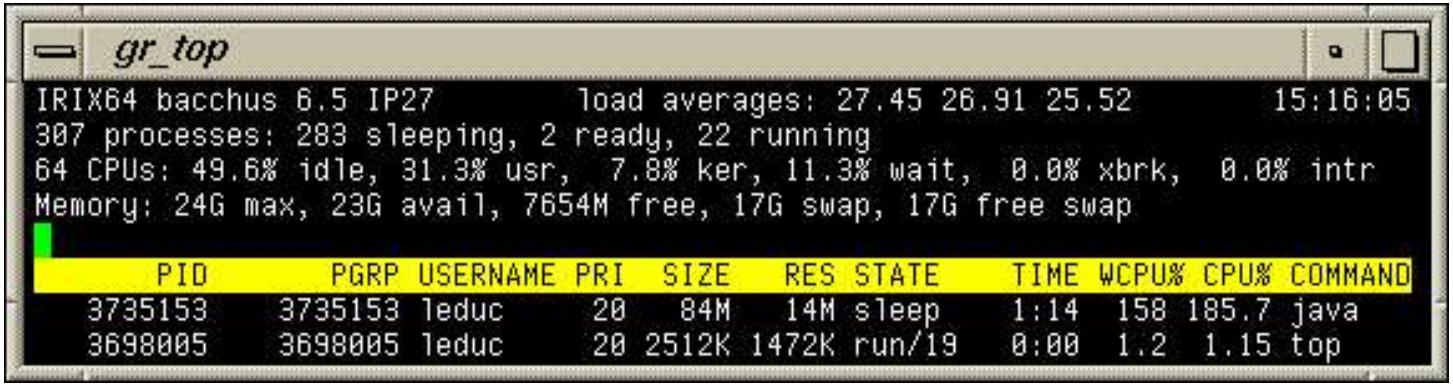
- qui est instanciée au niveau de la classe `Rac` (modifiée) :

```
1   public class Rac
2   {
3       private static int RDV = 0;
4
5       public void itérer() throws IOException
6       {
7           tableauThreads = new RacAux[nbThreads];
8           for (int i=0 ; i<tableauThreads.length ; i++) {
9               début = i*nbCellParThread;
10              fin = (i+1 != nbThreads) ? (i+1)*nbCellParThread : nbCellules;
11              tableauThreads[i] =
```

```
12                  new RacAux(début,fin,[...]);
13          }
14          for (int compteur=0 ; compteurCliché<propriétés.nombreDeCliches-1 ; compteur++) {
15              // première  synchronisation  des  threads :
16              try {
17                  while (RDV != tableauThreads.length)
18                      Thread.currentThread().sleep(10); // paramètre  à  ajuster
19              } catch (InterruptedException e) {}
20              // remise  à  zéro  du  compteur  de "rendez-vous" :
21              incrementerRDV(-tableauThreads.length);
22              // traitement  d'affichage  ou  stockage  de  la  configuration  courante :
23              compteurCliché = représenterGraphiquement(compteurCliché,compteur);
24              // relance  globale :
25              // notifyAll ();          (ne  marche  qu'avec  la  version  1.3  du  JDK)
26              for (int i=0 ; i<tableauThreads.length ; i++)
27                  synchronized(tableauThreads[i])
28                      { tableauThreads[i].notify (); }
29          }
30      }
31      synchronized static void incrementerRDV(int n)
32      {
33          RDV += n;
34      }
35  }
```


leduc@lmt.ens-cachan.fr

À propos des performances...



- de l'utilité de l'option `-T` (*List data for individual kernel threads*) de la commande `ps`

Conclusions et évolutions

- écriture d'une version "objets distribués" ou "invocation de méthodes à distances" de la transition globale, pour prendre en compte le cas d'architectures à mémoire distribuée non partagée,
- envisager une version par échange de messages explicites (couplage Java-MPI ?),
- développement d'une GUI et d'un mini-langage pour l'aide à la mise au point de réseaux d'automates cellulaires...
- une adresse : `http://bacchus.lmt:8080/~leduc/doctorat/`.