Algorithmique avancée et parallélisme

Laurent Lemarchand
LabSTICC/UBO
Laurent.Lemarchand@univ-brest.fr

https://moodlesciences.univ-brest.fr/moodle/course/view.php?id=706#section-4

Plan du cours

- Modèle passage de message
- PVM : fonctionnalités
- Utilisation de PVM

- Exemples d'utilisation en optimisation combinatoire
  - Méthodes exactes (Branch & Bound)
  - Méthodes heuristiques (Algorithmes génétiques)

Evaluation

- Partie Algorithmique parallèle
  - note CC (PVM sur 10 points) L Lemarchand

Besoins en puissance de calcul
besoins et possibilités

- Problématique
  - ***Calculs trop gros***
  - ***Calculs trop long***
- ***Possibilités***
  - Matériel plus rapide
    - évolutions processeurs et mémoire
  - Meilleurs algorithmes
    - algorithmique
  - Travailler plus
    - parallélisme

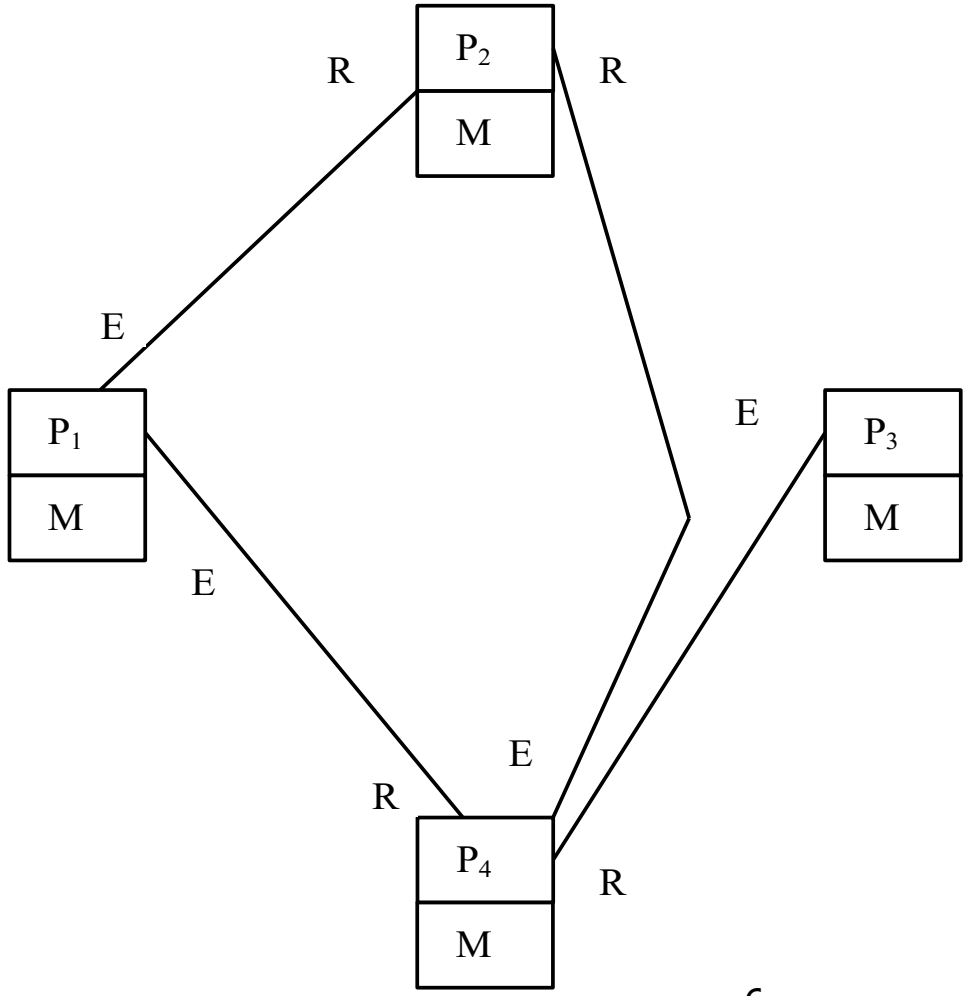
Applications envisagées
MIMD/SPMD

- Sur grappe de calcul (parallélisme à gros grain)
  - Avec mémoire distribuée
    - pas cher, répandu (réseau de stations)
  - Ou système multipro/multicoeur
    - Pas cher, répandu aussi !

- Au niveau fonctionnel ou programme
  - SPMD (1 programme)
  - client/serveur (MIMD 2 programmes)

Introduction au Passage de Messages

- Modèle à passage de messages :
  - Plusieurs processus s'exécutent en parallèle
  - Processus attribués à des processeurs distincts
  - Chaque processeur possède sa propre mémoire (pas de mémoire partagée)
  - Les communications entre processeurs se font par envoi (send) et réception (receive) de messages



Introduction

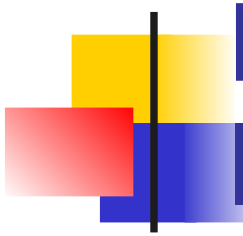
- Principales fonctions de l'échange de messages
  - Échange de données
  - Synchronisation
- Particulièrement adapté aux architectures à mémoire distribuée (de type MIMD)

Caractéristiques du modèle

- Difficulté de programmation
  - Gestion explicite de
    - La distribution des données
    - L'ordonnancement des tâches
    - La communication entre les tâches
  - Conséquences
    - Peut prendre beaucoup de temps
    - Peut mener à plusieurs erreurs
    - Coûts additionnels de développement

Caractéristiques du modèle

- Efficacité
  - Beaucoup de liberté laissée au programmeur
  - Possibilité d'optimisation et de « fine-tuning »
  - Conséquence : si les ressources sont disponibles, une grande efficacité peut être obtenue
- Portabilité
  - Maturité du modèle
    - Popularité depuis plusieurs années : standards établis
    - Standards implémentés sur plusieurs machines
  - Conséquences
    - Un programme écrit en respectant les standards peut être facilement transposé sur un autre ordinateur
    - Portabilité du code <> Portabilité de la performance



Développement d'applications basées sur le passage de messages

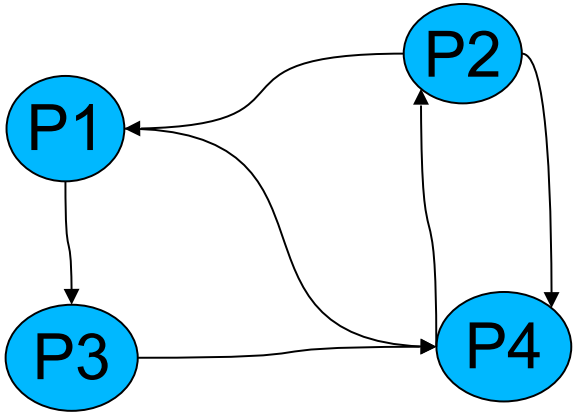
- 2 principaux outils :
  - Message Passing Interface (MPI)
  - Parallel Virtual Machine (PVM)
- Englobent une grande proportion des applications parallèles existant aujourd'hui
- Basés sur des bibliothèques permettant d'étendre des langages séquentiels existants
  - C
  - C++
  - Fortran

Modèle de programmation réseau faiblement couplé

- Modèle asynchrone MPMD (au mieux SPMD)
- Modèle de programmation
  - Services offerts ?
  - Utilisation de librairies
- Parallel Virtual Machine
  - Réseau hétérogène (clusters/grille/constellations)
  - Services via librairie C ou Fortran

Modèle de programmation processus communicants

- Graphe de processus asynchrones communicants
- Modèle indépendant des ressources
  - Localisation des processus
  - Média de communication
- Communications
  - Bi-point
  - Par groupe

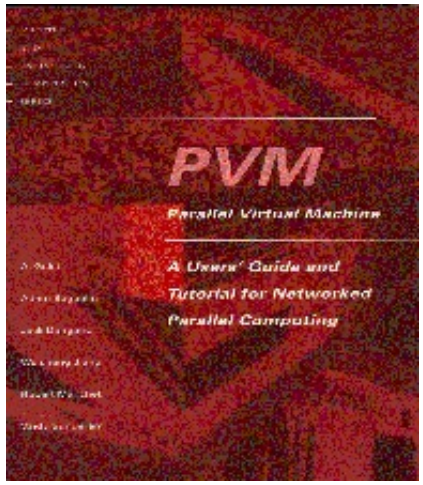


Parallel Virtual Machine
caractéristiques

- Sur réseau hétérogène faiblement couplé
- Services
  - Création/Destruction de processus
  - Communications (transport XDR)
    - Asynchrones bi-point FIFO ou multicast
    - Synchrones bi point (RdV) ou groupe (barrière de synchronisation)
  - Signaux distants
- Gestion de la machine et des processus

Références PVM

- Site web www.epm.ornl.gov/pvm/pvm_home.html



- Livre

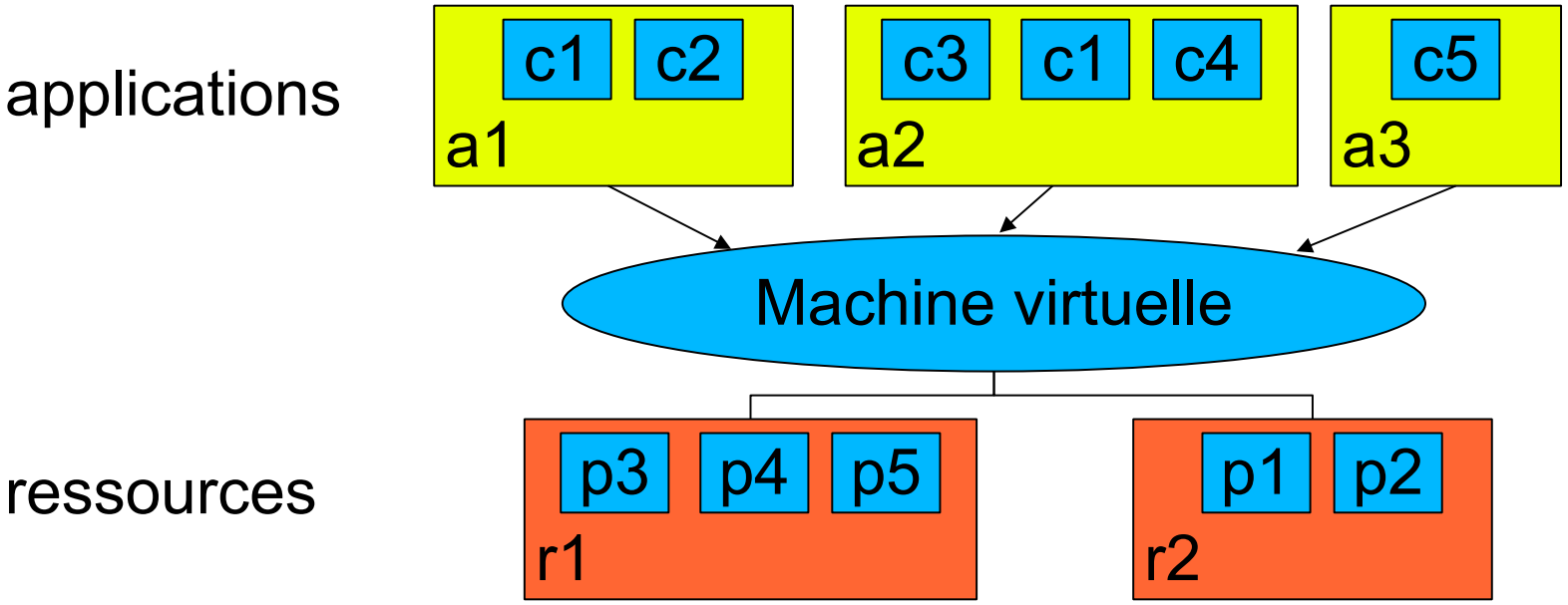
PVM: Parallel Virtual Machine
A Users' Guide and Tutorial for Networked Parallel Computing

Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, Vaidy Sunderam

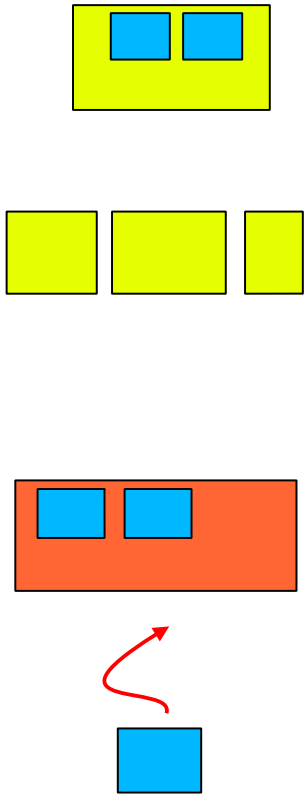
www.netlib.org/pvm3/book/pvm-book.html

Parallel Virtual Machine
machine virtuelle

- Vue unifiée d'un réseau hétérogène, indépendant
  - Des ressources physiques de calcul variées
  - Des média de communication



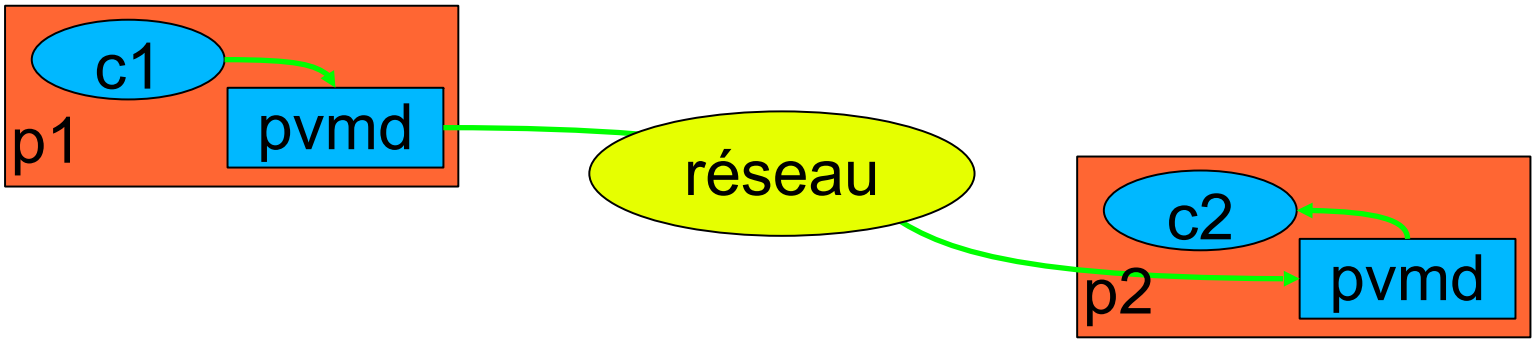
Parallel Virtual Machine
machine virtuelle



- Une application
  - Plusieurs composantes (processus) interagissant librement
  - Plusieurs applications simultanées
- Une machine virtuelle
  - Enrollement préalable des ressources physiques
  - Management dynamique possible

Parallel Virtual Machine
composition

- Pour les applications
  - Accès aux ressources via une librairie d'interface
- Pour les ressources processeurs
  - Enrollement préalable des ressources physiques
  - Démon pvmd local



Parallel Virtual Machine
mise en route

- Console pvm
  - Gestion des machines

```
% pvm
pvm> conf
localhost
pvm> add distHost
```

  - Gestion des processus

```
% pvm
pvm> ps -ef
...
pvm> reset
pvm> halt
pvm> quit
```

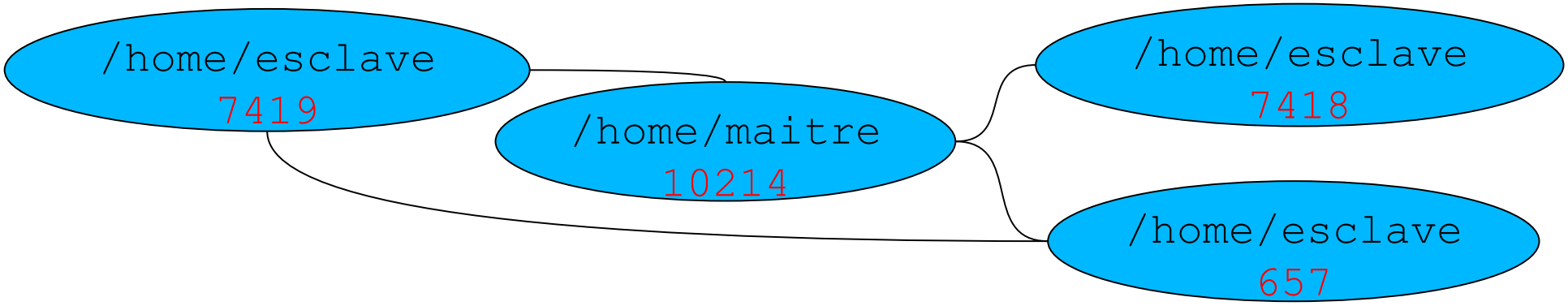
Parallel Virtual Machine
console

```
% pvm
pvm> help
```

```
add         Add hosts to virtual machine
alias       Define/list command aliases
conf        List virtual machine configuration
delete      Delete hosts from virtual machine
echo        Echo arguments
export      Add environment variables to spawn export list
getopt      Display PVM options for the console task
halt        Stop pvmds
help        Print helpful information about a command
id          Print console task id
jobs        Display list of running jobs
kill        Terminate tasks
mstat       Show status of hosts
names       List message mailbox names
ps          List tasks
pstat       Show status of tasks
put         Add entry to message mailbox
quit        Exit console
reset       Kill all tasks, delete leftover mboxes
setenv      Display or set environment variables
setopt      Set PVM options - for the console task *only*!
sig         Send signal to task
spawn       Spawn task
trace       Set/display trace event mask
unalias     Undefine command alias
unexport    Remove environment variables from spawn export list
version     Show libpvm version
```

PVM
création de processus

- Localisés de machine transparente/par archi/par machine au choix
- Le tid unique est similaire à un pid Unix




```
NumPs = pvm_spawn("/home/esclave", ..., 3, tabTids)
// tabTids : {7419,7418,657}
```

PVM
processus

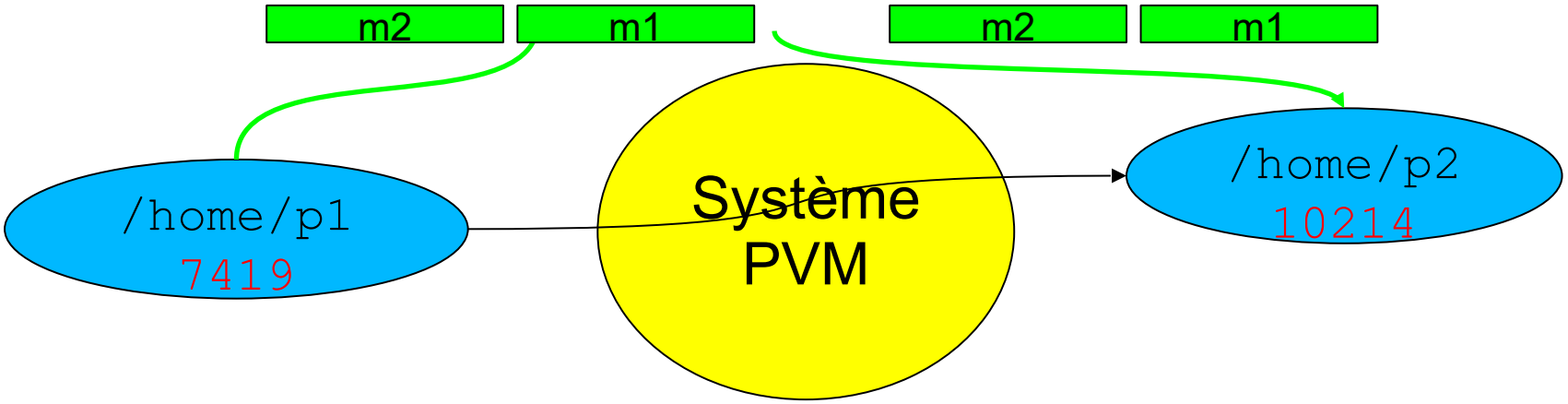
- Constituent les applications
- Identifiés par un tid unique sur l'ensemble de la PVM
- Correspondent à l'exécution de processus Unix


```
mtid = pvm_mytid();
// mtid : 10214
```

```
prt = pvm_parent();
// prt : tid du père
```

PVM
communications

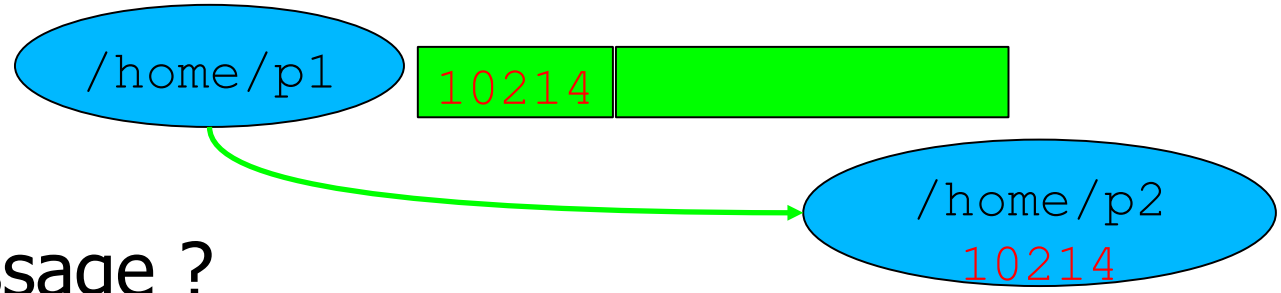
- Communications par passage de message
- Communication de base biPoint FIFO



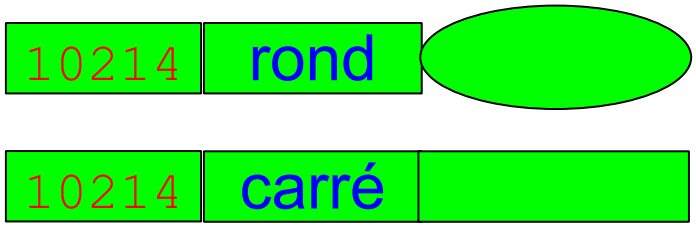
- Tamponnés dans le système

PVM
composition d'un message

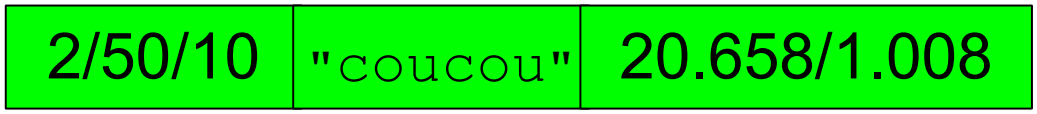
- Pour qui ?
  - tid



- Quel genre de message ?
  - filtre possible à l'arrivée

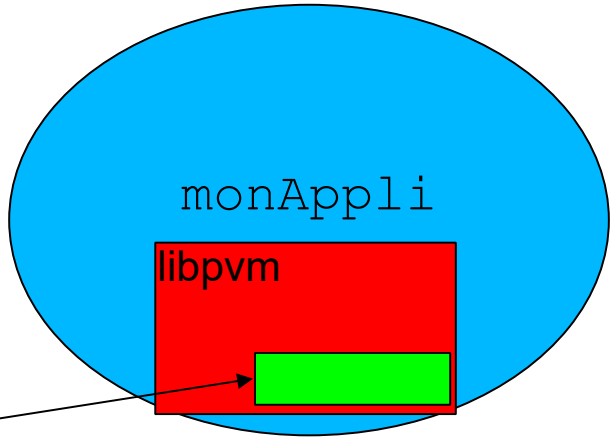


- Quelles données ?
  - typage (machines hétérogènes)
  - message composite



PVM
envoi – initialisation

```
...
pvm_initsend(PvmDataDefault);
```

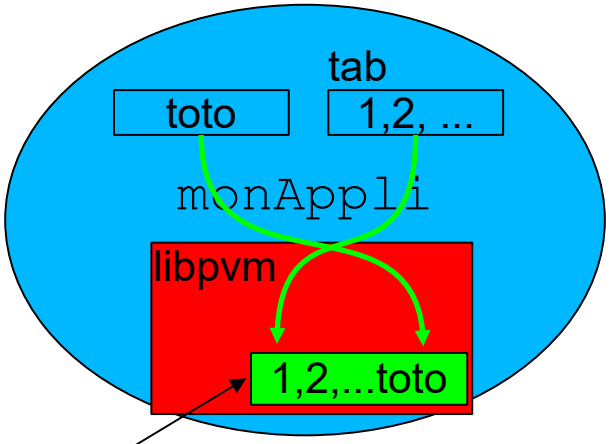


Initialise le buffer de communication

- Type d'encodage
  - Libre, XDR
- Suivant machine virtuelle prévue

PVM
envoi – constitution du message

```
...
pvm_initsend(PvmDataDefault);
pvm_pkint(tab, 10, 1);
pvm_pkstr("toto");
```



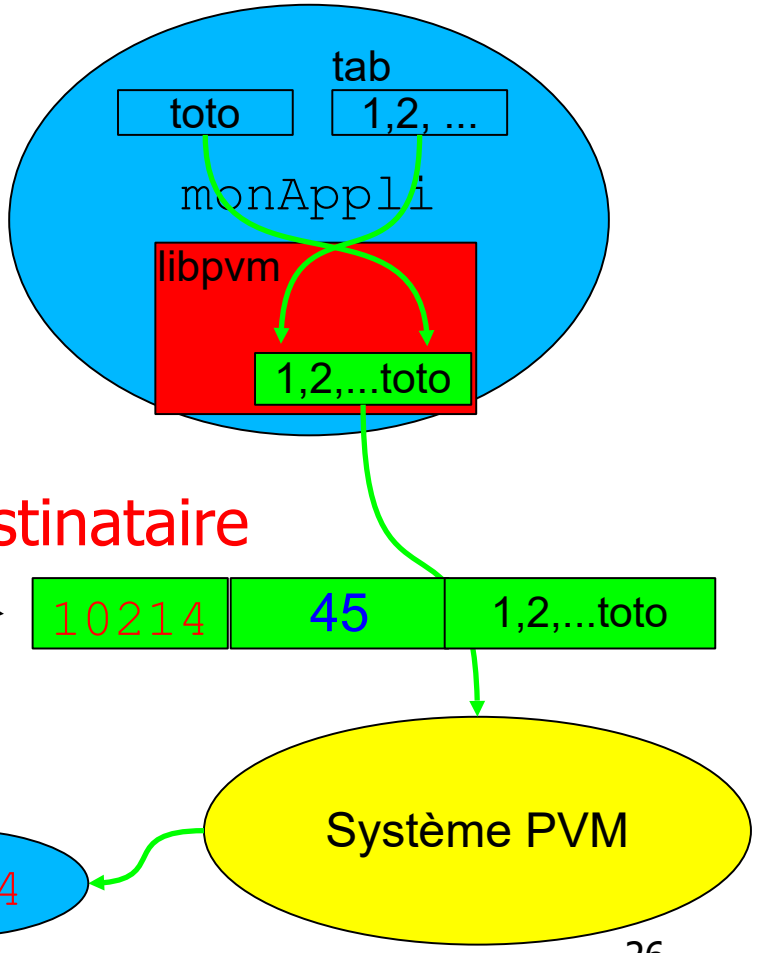
recopie des données dans le buffer

- Type des données associées au nom de fonction
  - pkint, pkfloat, pkstring, ...
  - Plusieurs empaquetages consécutifs possibles

PVM
envoi – envoi effectif

```
...
pvm_initsend(PvmDataDefault);
pvm_pkint(tab, 10, 1);
pvm_pkstr("toto");
pvm_send(10214, 45);
```

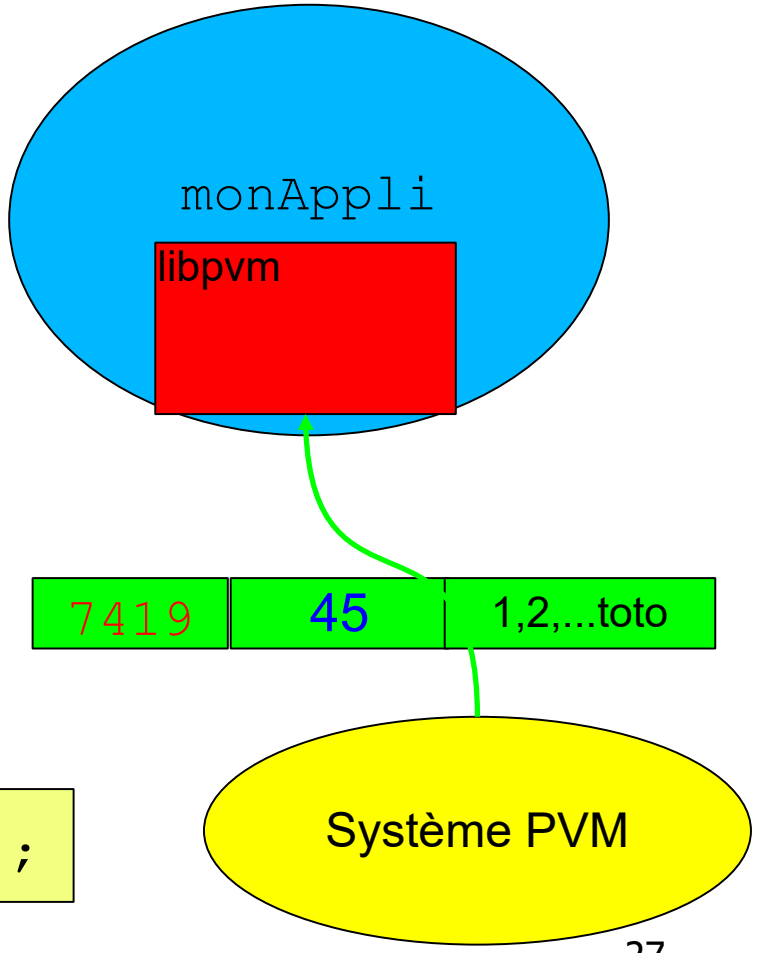
Envoi du message avec un genre au destinataire



- Recopie dans l'espace d'adressage du démon pvmd local
  - Envoi non bloquant

PVM
réception bloquante - filtrage

```
...
pvm_recv(exp, gre);
```



- Filtrage
  - Sur l'expéditeur exp
  - Sur le genre de message envoyé
- Indifférent possible :

```
pvm_recv(-1, gre);
```

```
pvm_recv(exp, -1);
```

PVM
réception non bloquante ou test

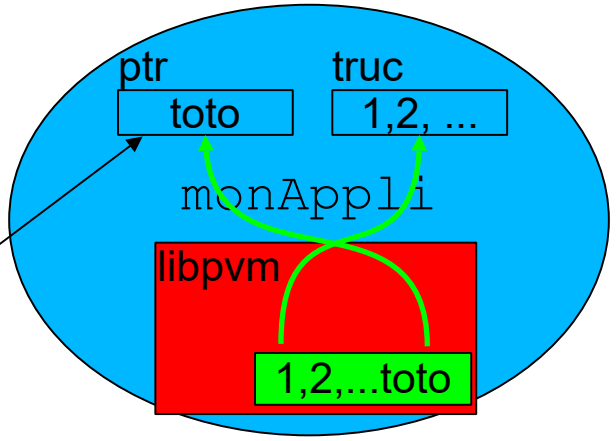
```
...
pvm_nrecv(exp, gre);
```

```
...
pvm_probe(exp, gre);
```

- Renvoient une valeur < 0 en cas d'erreur

PVM
réception - désempaquetage

```
...
pvm_recv(exp,gre);
pvm_upkint(truc, 10, 1);
pvm_upkstr(ptr);
```



recopie des données en local

- Dans le même ordre que lors de l'envoi

PVM
groupes de processus

```
...
pvm_join("groupe");
```

```
...
pvm_leave("groupe");
```

- Renvoient une valeur < 0 en cas d'erreur
- Pour des communications/synchronisations globales
- Implantées suivant ressources physiques

PVM
envoi direct de tableaux

PVM_STR PVM_FLOAT
PVM_BYTE PVM_CPLX
PVM_SHORT PVM_DOUBLE
PVM_INT PVM_DCPLX
PVM_LONG PVM_UINT
PVM_USHORT PVM_ULONG

```
pvm_psend(int tid, int msgtag,
          void *vp, int cnt, int type )
```

```
pvm_precv(int tid, int msgtag, void *vp, int cnt,
          int type, int *rtid, int *rtag, int *rcnt)
```

- vp, cnt : données, type: type des tableaux
- En réception: renvoie les mêmes infos que pvm_bufinfo() : tid réel, tag réel et taille du message

PVM
communications de groupe

- Envoi simultanné d'un message à plusieurs processus
  - Typage du message, comme avec pvm_send()

```
pvm_initsend(...);
pvm_pk...(...);
pvm_bcast("groupe", genre);
```

- Barrière de synchronisation
  - Tous bloqués jusqu'au n^{eme} appel

```
...
pvm_barrier("groupe", n);
```

Principe du paradigme maître/esclave

- Le maître a le contrôle de l'application globale et des données

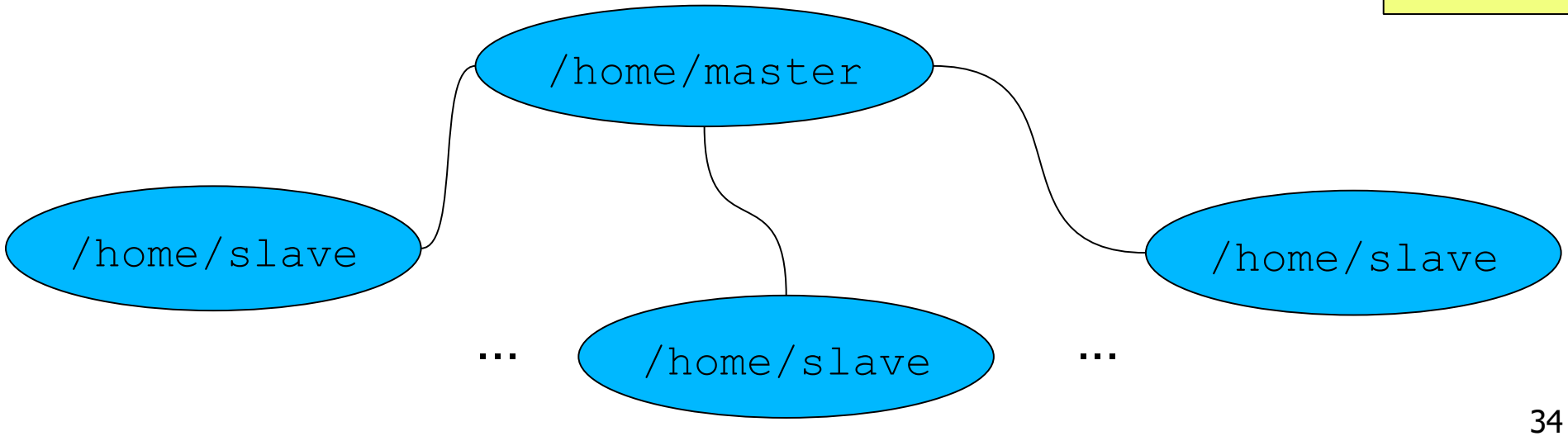
```
PROGRAM
IF (process = master) THEN
      master-code
ELSE
      slave-code
ENDIF
END
```

PVM
un exemple - application

- Création de n processus avec 2 programmes distincts
  - Un maitre, qui créera les autres processus
  - n-1 esclaves, exécutant le même traitement sur des données différentes

`master.c`

`slave.c`



PVM
un exemple - exécution

- ***Création d'une machine virtuelle avec 2 stations***

```
[sur mach1]% pvm
pvm> add mach2
pvm> quit
[sur mach1]%
```

- Éxecution de l'application puis terminaison de la machine virtuelle

```
[sur mach1]% master
...
trace
...
[sur mach1]% pvm
pvm> halt
[sur mach1]%
```

PVM
un exemple – code du maitre

```
#include "pvm3.h"
#define SLAVENAME "/home/slave"

main()
{
   int tids[32];          /* slave task ids */
   int n, nproc, numt, i, who, msgtype, nhost, narch;
   float data[100], result[32];

   puts("How many slave programs (1-32)?");
   scanf("%d", &nproc);
   /* start up slave tasks */
   numt=pvm_spawn(SLAVENAME, (char**)0, 0,
                              "", nproc, tids);


   /* Begin User Program -- dummy data */
   n = 100;
   for( i=0 ; i<n ; i++ ) {
     data[i] = i*10.8;
   }
```

```
   /* Broadcast initial data to slave tasks */
   pvm_initsend(PvmDataDefault);
   pvm_pkint(&nproc, 1, 1);
   pvm_pkint(tids, nproc, 1);
   pvm_pkint(&n, 1, 1);
   pvm_pkfloat(data, n, 1);
   pvm_mcast(tids, nproc, 0);

   /* Wait for results from slaves */
   msgtype = 5;
   for( i=0 ; i<nproc ; i++ ) {
         pvm_recv( -1, msgtype );
         pvm_upkint( &who, 1, 1 );
         pvm_upkfloat( &result[who], 1, 1 );
         printf("I got %f from %d\n", result[who], who);
   }

    /* Program Finished exit
       PVM before stopping */
   pvm_exit();
}
```

PVM
un exemple – code de l'esclave

```
#include <stdio.h>
#include "pvm3.h"

main()
{
   int mytid;       /* my task id */
   int tids[32];    /* task ids   */
   int n, me, i, nproc, master, msgtype;
   float data[100], result;

   /* enroll in pvm */
   mytid = pvm_mytid();

   /* Receive data from master */
   msgtype = 0;
   pvm_recv( -1, msgtype );
   pvm_upkint(&nproc, 1, 1);
   pvm_upkint(tids, nproc, 1);
   pvm_upkint(&n, 1, 1);
   pvm_upkfloat(data, n, 1);
```

```
   /* Determine which slave I am (0 -- nproc-1) */
   for( i=0; i<nproc ; i++ )
      if( mytid == tids[i] ){
         me = i; break;
      }

   /* Do calculations with data */
   result = work( ... );

   /* Send result to master */
   pvm_initsend( PvmDataDefault );
   pvm_pkint( &me, 1, 1 );
   pvm_pkfloat( &result, 1, 1 );
   msgtype = 5;
   master = pvm_parent();
   pvm_send( master, msgtype );

   /* Program finished. Exit PVM before stopping */
   pvm_exit();
}
```

PVM
configuration multipostes hétérogène

- ***Lancement de pvmd distant***
  - ***rsh/ssh avec login sans mot de passe***
    - ***~/.rhosts***
    - ***Pas d'affichage (echo) dans .cshrc***
  - ***Trouver pvmd adéquat (PVM hétérogène)***
    - ***$PVM_ROOT / $PVM_ARCH***

```
setenv PVM_ROOT /usr/share/pvm3
setenv PVM_ARCH `$PVM_ROOT/lib/pvmgetarch`
setenv PATH "${PATH}:${PVM_ROOT}/lib"
```

PVM
compilation des exécutables

- Compiler pour chacune des architectures cibles
  - Par exemple, un répertoire par architecture

```
setenv PATH "${PATH}:${HOME}/bin/${PVM_ARCH}"
```

  - Makefile adapté

```
BDIR   =   $(HOME)/bin/$(PVM_ARCH)

PVM_LIB = $(PVM_ROOT)/lib/$(PVM_ARCH)/libpvm3.a
CFLAGS =   -I$(PVM_ROOT)/include
include $(PVM_ROOT)/conf/$(PVM_ARCH).def

myprog: myprog.c
    cc $(CFLAGS) -o $(BDIR)/myprog myprog.c $(PVM_LIB) $(ARCHLIB)
```

PVM
lancement des exécutables

- Définir le chemin pour les exécutables de chaque machine lors du lancement de PVM
  - Fichier de configuration pour le démon pvmd3
  - Si, par exemple, $HOME sur NFS visible par tous

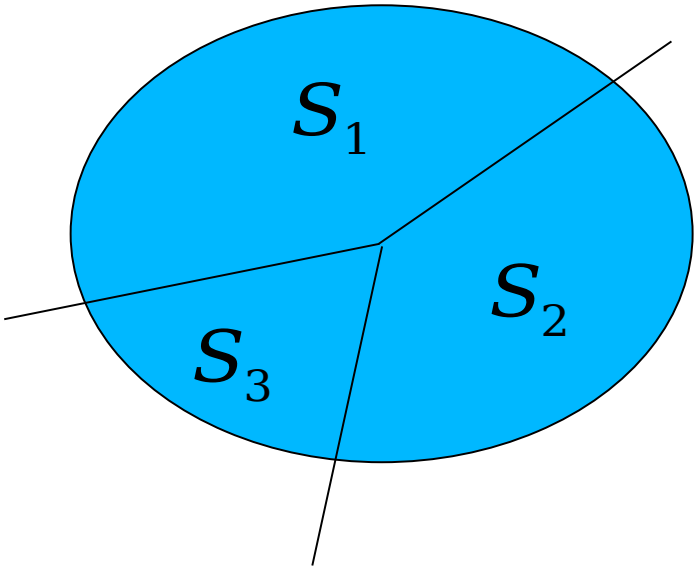
```
*   ep=$HOME/bin/$PVM_ARCH wd=$HOME
machine1
machine2
machine3
```

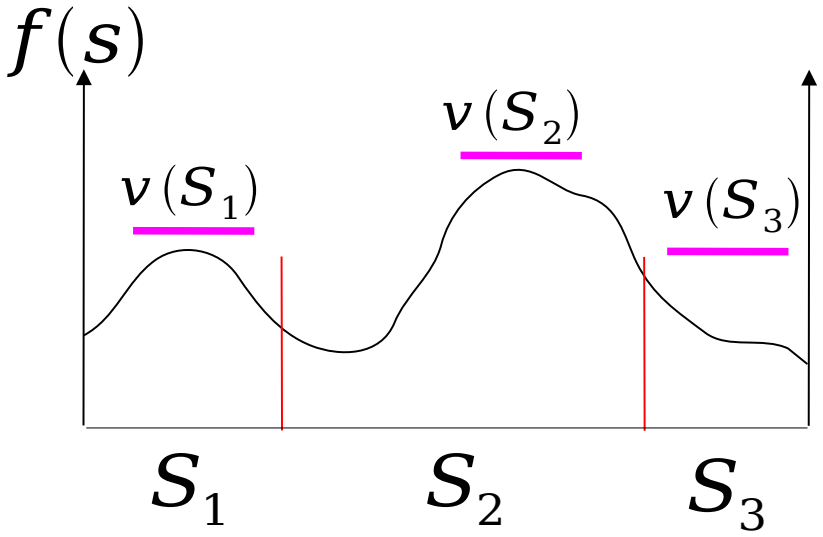
  - Lancement

```
pvm_spawn("myprog", (char **)0, 0, "", 5, tids)
```

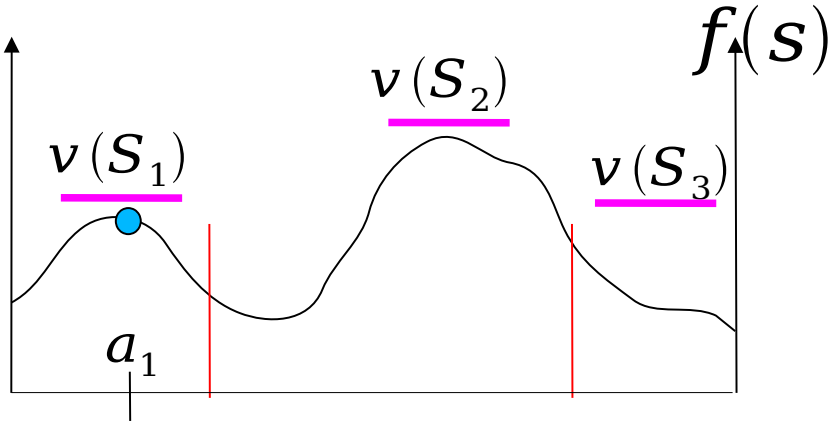
Exemples d'applications
Méthodes par séparation et évaluation
principe B&B

- Enumération implicite de toutes les solutions d'un problème d'optimisation
  - Sous espaces de recherche S_1, S_2, ..., S_n
  - *Qualité* pour chaque sous espace $v(S_1)$, $v(S_2)$, ..., $v(S_n)$





Méthodes par séparation et évaluation
principe B&B



- Supposons
  - f(s) à **maximiser**
  - Une solution réalisable a_1 connue, avec $v(a_1) = 12$
- $v(S_3) = 11$ implique que il n'est pas nécessaire d'explorer S_3

- $v(S_i)$ est une **borne supérieure** sur la qualité des solutions réalisables de S_i

Méthodes par séparation et évaluation
algorithme B&B

- Il faut, en plus de la fonction objective f(s) (*max*)
  - Une fonction de partitionnement d'un espace S en sous espaces

branch

$$S_1, S_2, \ldots, S_n : S = \bigcup_n S_i$$

bound

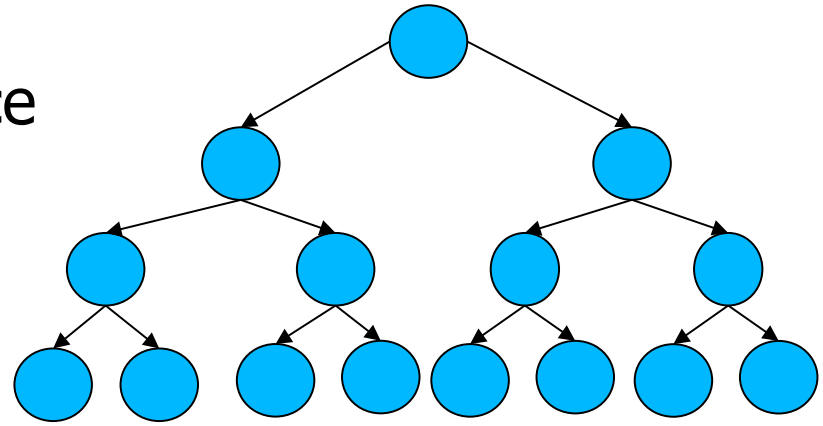
- Une fonction d'évaluation de qualité pour un espace donnant une borne supérieure sur S :

$$f : \quad \forall s \in S \;\; f(s^*) \leq v(S)$$

  - Une stratégie de recherche pour choisir quel sous espace S_i évaluer à l'itération suivante.

Méthodes par séparation et évaluation
arbre de recherche

- Le déroulement de l'algorithme correspond à l'exploration d'un arbre de recherche :
  - Chaque noeud correspond à un sous espace
  - Les feuilles sont des solutions ou des espaces sans solution réalisable
  - Les fils S_1, S_2, ..., S_n de S résultent du partitionnement de S



Méthodes par séparation et évaluation
algorithme

Espace S,

Évaluation v(S)(*min*),

séparer()

suivant()

liste sommets L = {S}

Borne initiale U = ∞

Meilleure solution

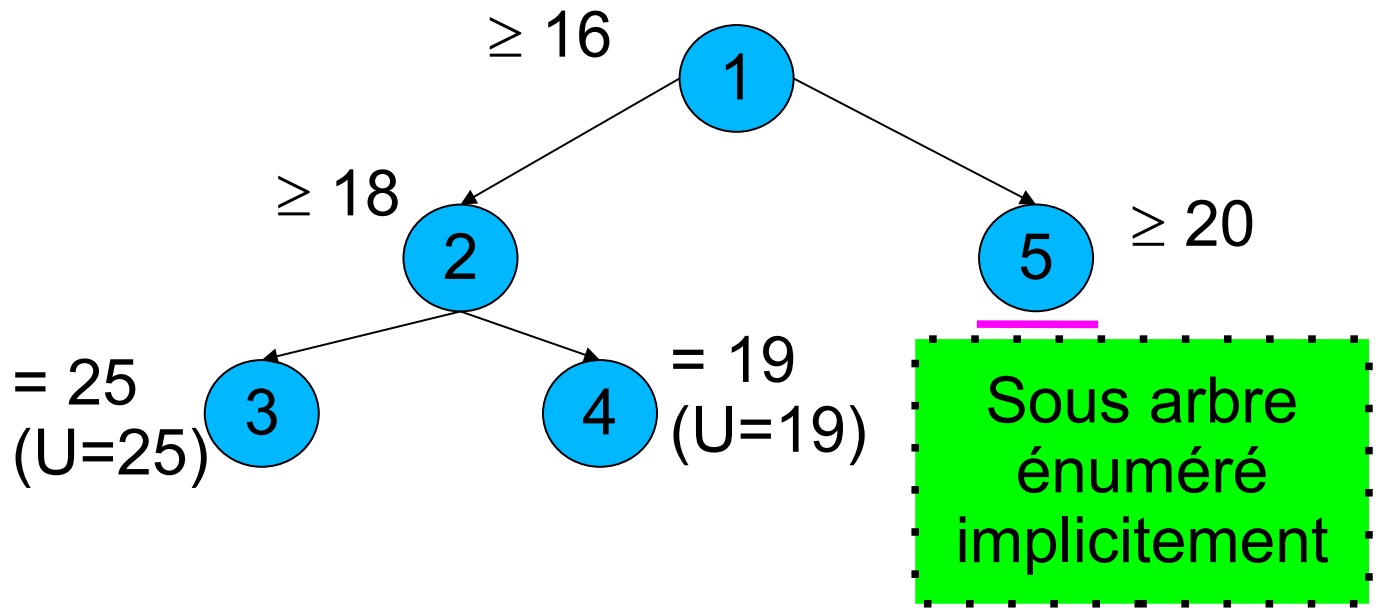
$S_{best} = \infty$

Algo B&B
tant que L $\neq \varnothing$ **faire**
    S = suivant(L)
    $S_1, S_2, ..., S_n$= séparer(S)

    pour chaque S_i **faire**
        si $v(S_i) > U$ ou S_i non réalisable
            éliminer S_i
        sinon si S_i réalisable
            $S_{best} = S_i$
            $U = v(S_i)$ (= $f(S_i)$)
        sinon
            $L = L \cup \{S_i\}$
fin

Méthodes par séparation et évaluation
coupes

- Coupes dans l'arbre de recherche si $v(S_i) > U$ (si *min*)
  - Énumération implicite



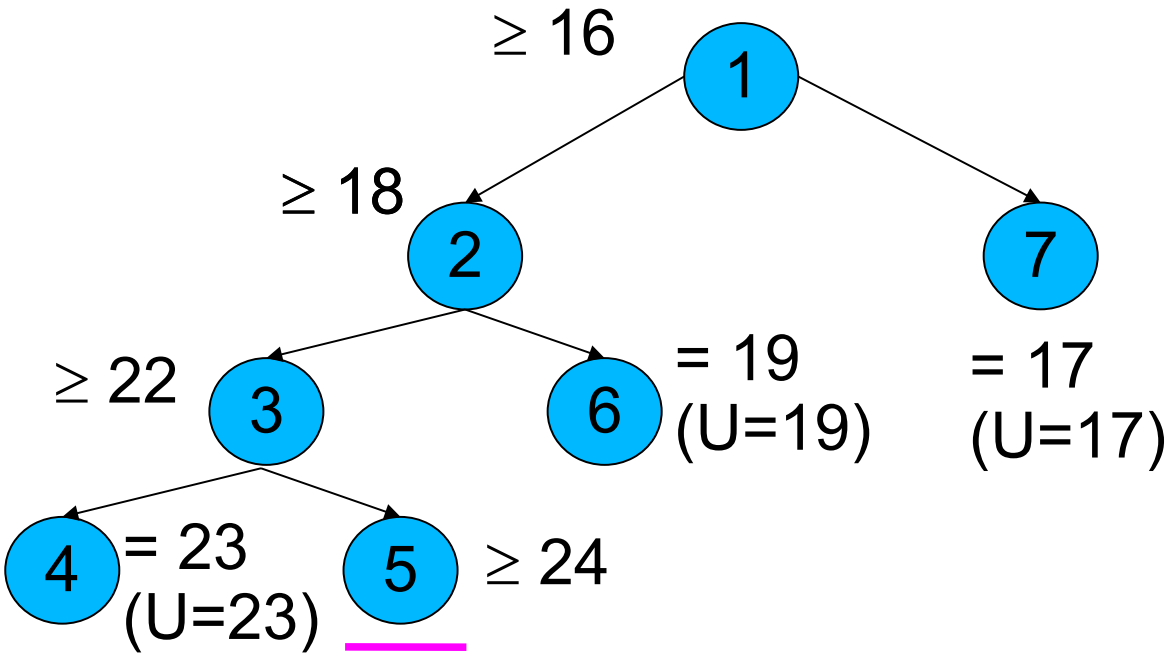
Méthodes par séparation et évaluation
parallélisation

- Coupes dans l'arbre de recherche si $v(S_i) > U$
  - Énumération implicite

- Parallélisation possible
  - Anomalies favorables ou défavorables suivant la fonction d'évaluation et la stratégie de parcours
  - Mise en oeuvre centralisée ou distribuée
  - Problème mise à jour de U

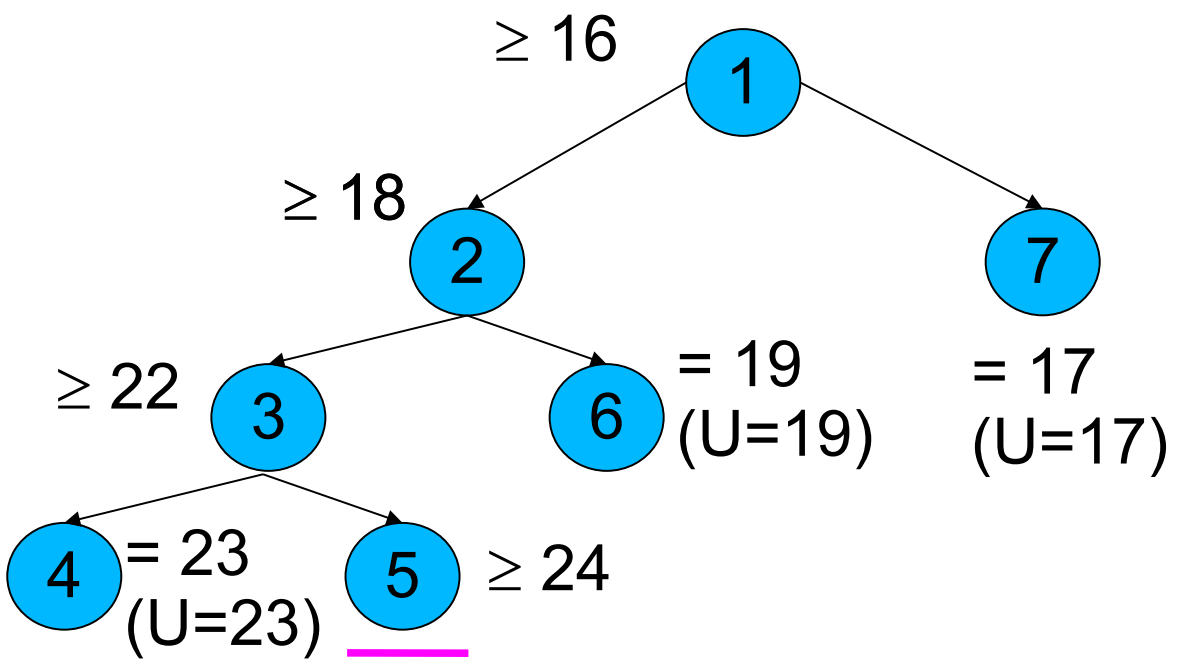
Méthodes par séparation et évaluation
accélération et parallélisation

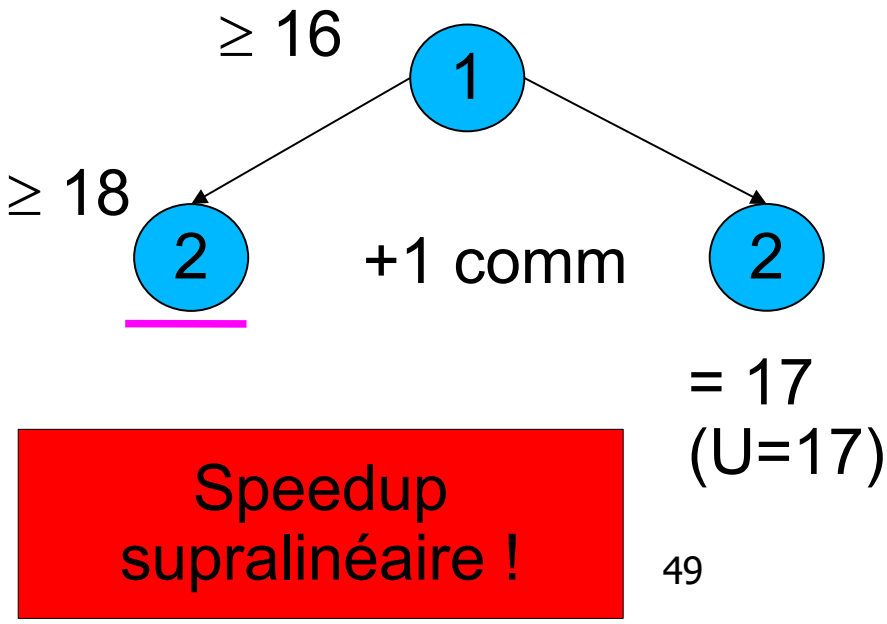
- Exemple avec mise en oeuvre centralisée
  - 1 maitre, 2 esclaves, comms 1 unité de temps en // ...
  - Parcours profondeur d'abord de l'arbre de recherche
    7 unités de temps en séquentiel



Méthodes par séparation et évaluation
accélération et parallélisation

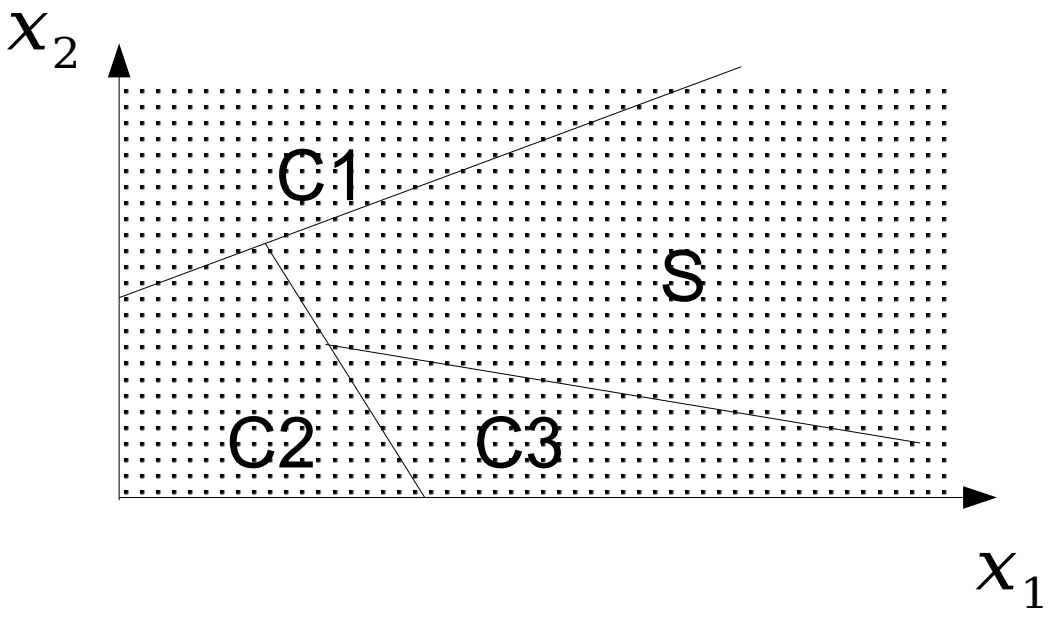
- Exemple avec mise en oeuvre centralisée
  - 1 maitre, 2 esclaves, comms 1 unité de temps en // ...
  - Parcours profondeur d'abord de l'arbre de recherche
    7 unités de temps en séquentiel
    2 unités de temps en // sur 2 processeurs





Exemples d'utilisation B&B
application à la programmation linéraire

- Le simplexe donne des valeurs dans $\mathbb{R}$
  - Comment résoudre un problème dans $\mathbb{N}$?

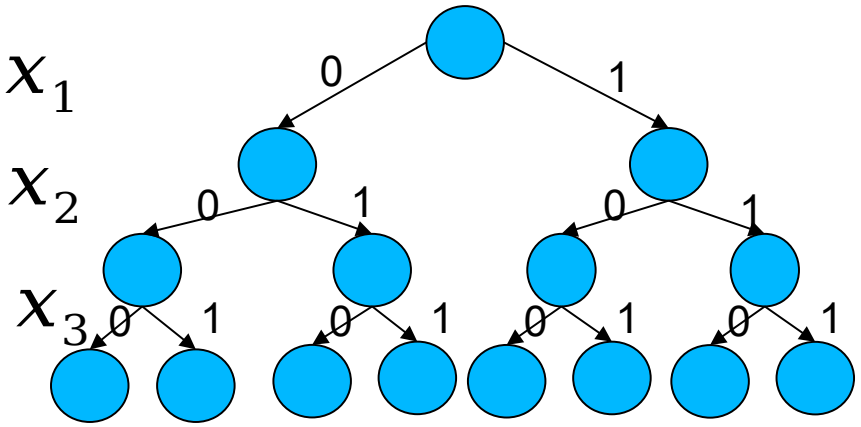


$$x_1 + 3x_2 \leqslant 15$$
$$-2x_1 - x_2 \leqslant -12$$
$$-3x_1 - 11x_2 \leqslant -66$$
$$min \quad 2x_1 + 4x_2$$
$$x_1, x_2 \in \mathbb{R}$$

$$x_1, x_2 \in \mathbb{N}$$

Exemples d'utilisation B&B
application à la programmation linéraire

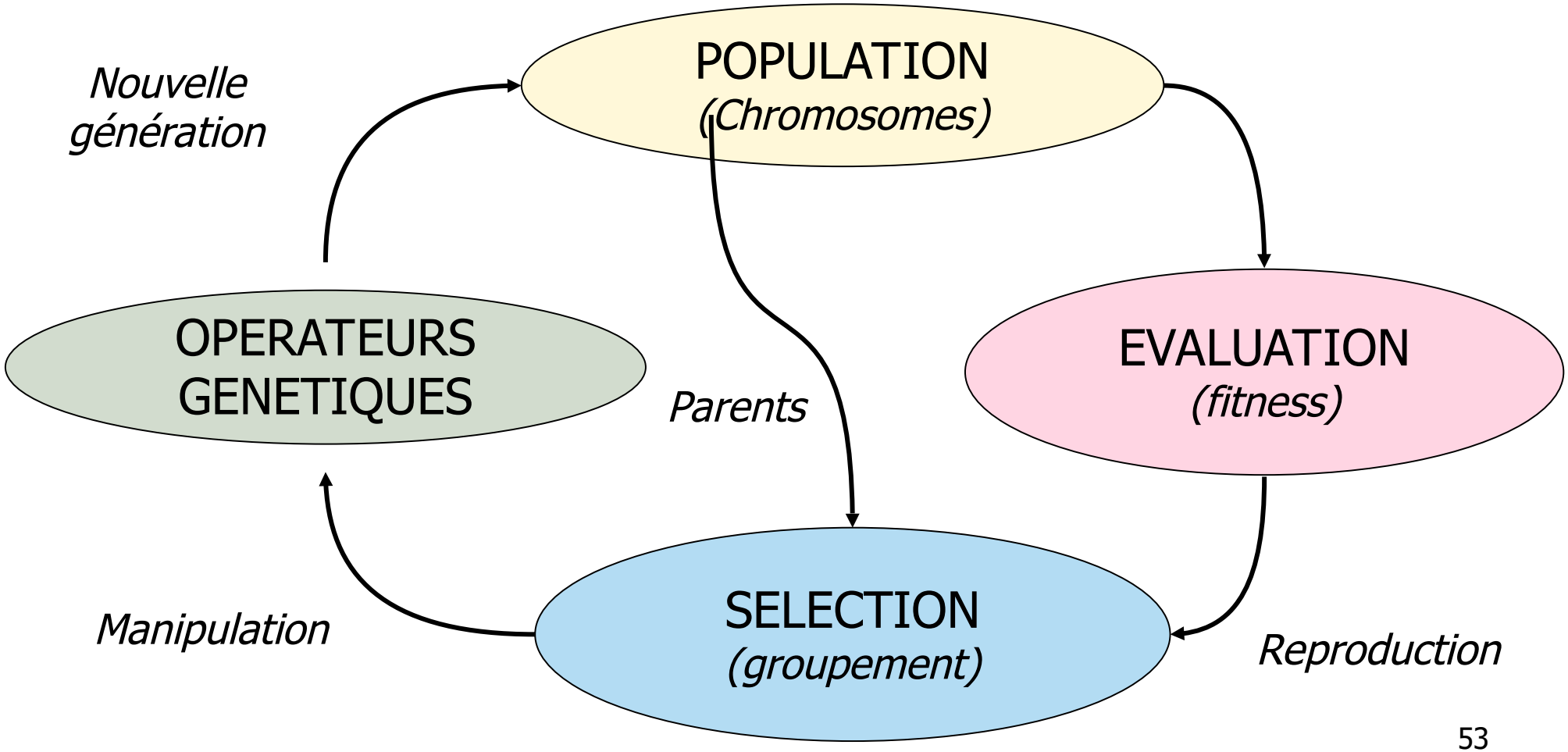
- $v(S_{\mathbb{R}}) \leq v(S_{\mathbb{N}})$ car $\mathbb{N} \subset \mathbb{R}$
  - Le simplexe fournit une borne pour la solution dans $\mathbb{N}$
- On sépare sur les variables
  - bivalentes (*0-1 IP*)
  - ou dans un intervalle entier quelconque (*IP*)



Exemples d'applications

Algorithmes génétiques

principe de base



Algorithmes génétiques
Algorithme

Espace S,
Évaluation f(s)(*min*),
crossover(P)
mutation(P)

taux de mutation et
de crossover

Algo GA
générer population initiale P dans S
tant que critère fin non atteint **faire**
reproduction(P) suivant f()
crossover(P)
mutation(P)
fin tant que
meilleure solution de population
fin

Algorithmes génétiques
parallélisme

- Possibilités de parallélisme
  - Modèle à grain fin maitre/esclave (// sur l'évaluation des individus)
    - Application d'heuristiques d'amélioration locale possible
  - Modèle à gros grain (// sur plusieurs populations distinctes et indépendantes)
    - Modèle des ilots (transfert éventuel d'individus dans des populations distantes)

Algorithmes génétiques
parallélisme – un exemple

- Sur des problèmes de test de TSPLIB
  http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/index.html
  - Instances en 2D (distances euclidiennes)
  - Modèle à ilots, 10 stations en ethernet 100Mb/s, MPI (// interprocs)
    - 1000 générations
    - 1000 individus
    - Probabilité de mutation 30%, (cross over ?)
    - Migration circulaire, 100 itérations, 20% de la pop.
    - Crossover à 1 point, sélection proportionnelle

Algorithmes génétiques
parallélisme – un exemple

Comparison of Parallel Metaheuristics for Solving the TSP
M. Lazarova, P. Borovska
CompSysTech'08

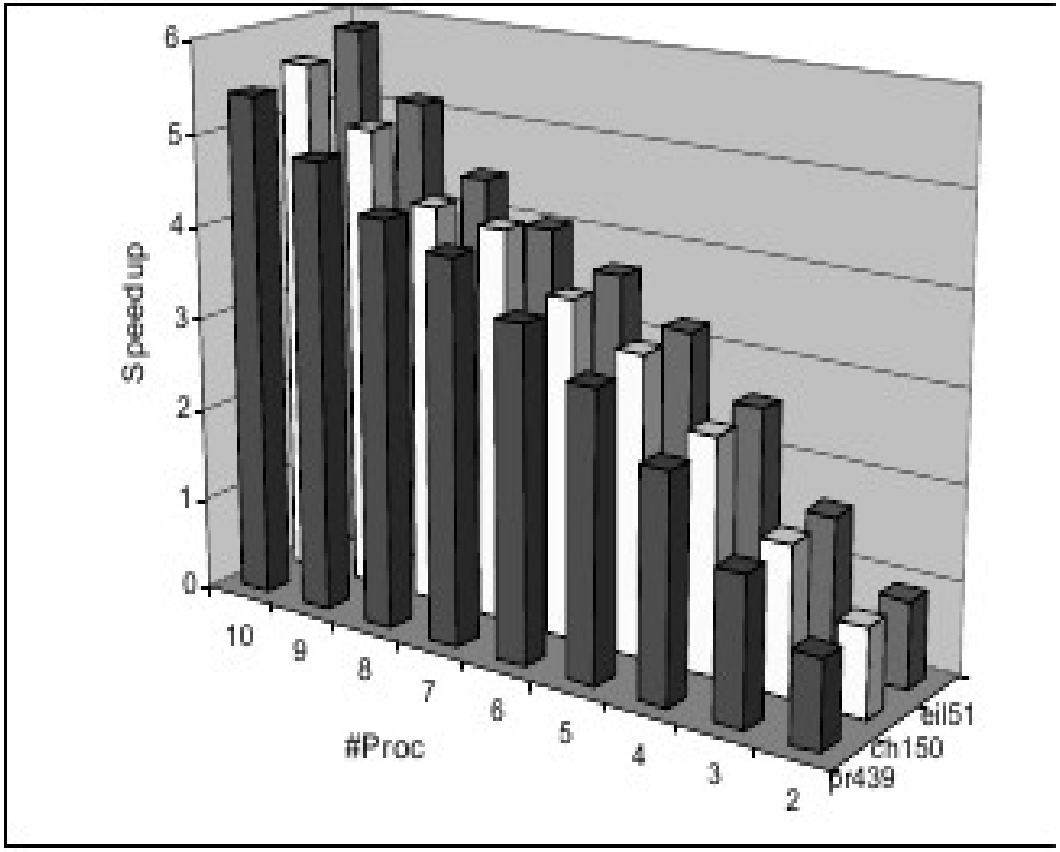


Fig.6. Speedup of parallel GA