PVM
Parallel Virtual Machine

Thomas LEDUC
Thomas.Leduc@lip6.fr
http://quartz.dgs.jussieu.fr:8080/

Octobre 1997

Table des matières

1 Introduction

1.1 La programmation parallèle

Un peu de vocabulaire :

- un *calcul* c'est un enchaînement d'opérations sur des objets. Ces opérations peuvent-être mutuellement exclusives (calcul séquentiel) ou simultanées (calcul parallèle),
- un *programme* c'est la description syntaxique d'un calcul,
- un *processeur* c'est un organe de réalisation d'un calcul par exécution d'un programme,
- un *processus* (une *tâche*), c'est un objet informatique représentant en mémoire l'exécution d'un programme sur un processeur pour réaliser un calcul.

La notion de programmes parallèles, coopérants et communicants a d'abord été envisagée pour la conception d'algorithmes comprenant des activités concurrentes (simulations à événements discrets) et pour partager des ressources entre utilisateurs concurrents sur des systèmes d'exploitation. Maintenant, il s'agit de gagner en efficacité et en place mémoire et de résoudre des problèmes intrinsèquement parallèles (contrôle de procédés industriels en temps réel) et/ou répartis...

Les modèles d'exécution parallèle

- Le parallélisme des données
  A partir d'un unique exécutable, avec un ou plusieurs flots de contrôles
  - SIMD (Single Instruction Multiple Data)
  - MIMD (Multiple Instruction Multiple Data)
  - SPMD (Single Program Multiple Data)
- Le parallélisme de contrôle ou de tâches
  Un (ou plusieurs) exécutables sont distribués sur un (ou plusieurs) processeurs et donnent naissance à plusieurs tâches.
  - MIMD (Multiple Instruction Multiple Data)
  - SPMD (Single Program Multiple Data)
  - MPMD (Multiple Program Multiple Data)

Du point de vue logique et physique, une machine physique mono-processeur (séquentielle donc) peut simuler tout à tour l'exécution des plusieurs processus logiquement parallèles. De même, un processus logiquement séquentiel peut-être exécuté en parallèle par des processeurs fonctionnant simultanément.

Pourquoi PVM? Pour s'affranchir de la (ou des) machine(s) physique(s) justement. Il s'agit d'une architecture virtuelle pour chaque utilisateur. Avec plusieurs "réalisations" matérielles possibles à la clef : machine massivement parallèle (grain fin, nombreuses communication, CM-5) ou réseau de stations de travail (grosse granularité, peu de communications).
Utilisations possibles : employer le parc de machine pendant les heures creuses pour faire de gros calculs, utiliser efficacement certains multi-processeurs à mémoire répartie (T3E...), utiliser simultanément des super-calculateurs, outil pédagogique pour la formation au parallélisme.

1.2 Le Message Passing

Dans le cas du modèle de programmation par échange de messages, le programme est écrit dans un classique (type C, Fortran...), chaque processus exécute éventuellement des parties différentes du programme, toutes les variables du programme sont privées et résident dans la mémoire locale de chaque processus et une donnée est échangée entre deux ou plusieurs processus par appel explicite à une primitive de la librairie de communication au sein même du programme.

1.3 Historique de PVM

été 89	projet de recherche universitaire démarre au Oak Ridge National Laboratory.
1990	version 1.0 (prototype jamais rendu public)
février 1991	version 2.0 écrite à l'université du Tennessee
1992	version 2.1 à 2.4 (des modifications et des corrections)
1993	version 3.0 à 3.3.11 (réécriture complète, des extensions et des corrections)
1997	version 3.4 (extensions majeures, introduction des contextes de communication et des groupes statiques de processus)

Nous utiliserons la version 3.3.11. Pour plus d'information concernant les différentes versions, consulter :

http://www.epm.ornl.gov/pvm/changes.html

1.4 La machine virtuelle PVM

Il s'agit de parallélisme asynchrone[1], distribué, avec réseau de communication. Les conditions :

- un nombre quasi illimité[2] de machines hétérogènes communiquant par message au travers d'un réseau hétérogène,
- machines organisées en une ou plusieurs grappes locales, où chaque grappe partage un système de fichiers commun,
- programmes écrits en Fortran, C ou C++.

Une application PVM, c'est : un ensemble de processus autonomes exécutant chacun leur propre code et communiquant via des appels aux routines de PVM. Il existe des routines[3] de gestion de l'environnement, de communication point à point, de communication collective et de gestion des groupes.
Le produit : un démon pvmd par machine physique, une console sur la machine hôte, des variables d'environnement, trois bibliothèques (libpvm3.a, libfpvm3.a et libgpvm3.a correspondant aux routines C, Fortran et gestion des groupes dynamiques).

1.5 De la documentation sur le Net

- http://www.infop6.cicrp.jussieu.fr/~leduc/
- http://www.epm.ornl.gov/pvm/
- ftp://ftp.irisa.fr/pub/netlib/pvm3/book/pvm-book.html

1. non bloquant
2. Au plus 4095 machines exécutant au plus $2^{18} - 1$ tâches chacunes
3. Plus d'une soixantaine...

- ftp://ftp.irisa.fr/pub/netlib/pvm3/faq_html/faq.html
- http://www.idris.fr/data/cours/parallel/pvm/
- http://www.ens.fr/~cousot/cours/parallelisme/parallelismePVM/index.html

1.6 Erreurs typiques de la programmation parallèle

2 PVM - introduction pratique

2.1 Quelques remarques concernant "l'installation" de PVM au niveau utilisateur

Pour chacun des systèmes de fichiers accessibles par une des machines de PVM, faire :

- créer un répertoire $\sim /pvm3$, un sous-répertoire $\sim /pvm3/bin$, et autant de sous-répertoires $\sim /pvm3/bin/PVM\_ARCH$ que de type d'architectures de machines différentes accédant, à partir d'une des machines de PVM, à ce système de fichiers,
- créer un fichier $\sim /.rhosts$ dans lequel figurera la liste des machines de PVM, et modifier ses droits d'accès comme suit : chmod go-rwx $\sim /.rhosts$,
- vérifier que les variables d'environnement PATH, PVM_ROOT... sont correctement initialisées et ci ce n'est le cas mettre à jour le fichier $\sim /.cshrc$ (veiller à faire un *source* $\sim /.cshrc$),
- supprimer le fichier $\sim /.tcshrc$ auquel cas le tcsh utilisera le $\sim /.cshrc$, mettre les alias dans un autre fichier que le $\sim /.cshrc$ et les messages type "post-it" dans le fichier $\sim /motd$ (que vous appellerez dans votre $\sim /.login$).

De plus, pour le bon déroulement des séances de TD, on prendra le tcsh pour login-shell [4] et on ajoutera les lignes suivantes au fichier $\sim /.cshrc$:

```
if ($?PVM_ROOT) then
        setenv PVM_ARCH `$PVM_ROOT/lib/pvmgetarch`
        set path=($path $PVM_ROOT/bin/$PVM_ARCH ~/pvm3/bin/$PVM_ARCH)
endif
```

2.2 La console pvm

La commande *pvm* lance le démon pvmd, si ce n'est déjà fait et une tâche de contrôle de la machine virtuelle (prompt associé : *pvm* >). Pour s'en convaincre, lancer la console en arrière-plan et lister la table des processus Unix.
Note : pour spécifier un parc de machines physiques donné à votre machine virtuelle PVM, tout en lançant la console, vous pouvez taper la commande : pvm hostfile (hostfile étant un nom de fichier).

Commandes utilisables à partir de la console pvm :

- **add** ou **delete** machine(s) : ajouter ou supprimer des machines physiques à la configuration de la machine PVM,
- **conf** : lister l'ensemble des machines de la configuration

4. Utiliser la commande : *chsh −s /bin/tcsh nom_de_login*

- **quit** : quitter la console (le(s) démon(s) reste(nt) toujours actif(s))
- **halt** : arrêt complet de la machine virtuelle PVM.
- **help** : pour lister les commandes.
- **ps** : pour obtenir la liste des tâches PVM.
- **version** : pour obtenir le numéro de version de la machine PVM utilisée.

Le fichier hostfile Permet de définir la liste des composants de notre machine virtuelle PVM. Il doit résider sur la machine "père" et peut-être stocké à n'importe quel endroit.

```
ippc1.infop6.cicrp.jussieu.fr{leduc}71: cat hostfile-134
ippc1.infop6.cicrp.jussieu.fr
ippc3.infop6.cicrp.jussieu.fr
ippc3.infop6.cicrp.jussieu.fr
ippc1.infop6.cicrp.jussieu.fr{leduc}72:
```

Le fichier $\sim$ */.pvmrc* Au lancement de la console pvm, il y a lecture du fichier $\sim$ */.pvmrc*. Celui-ci permet de définir quelques alias et de préciser quelques commandes comme par exemple :

```
$ cat $HOME/.pvmrc
alias ? help
alias ps ps -lx
alias p1 add ippc1.infop6.cicrp.jussieu.fr
...
echo "###########################################"
echo "Lancement d'une console PVM d'identifiant :"
id
echo "###########################################"
$
```

Un exemple de session (avec le fichier $\sim$ */.pvmrc* **ci-dessus) :**

```
ippc1.infop6.cicrp.jussieu.fr{leduc}26: pvm
###########################################
Lancement d'une console PVM d'identifiant :
t40001
###########################################
pvm> p3
1 successful
                    HOST     DTID
ippc3.infop6.cicrp.jussieu.fr    80000
pvm> conf
2 hosts, 1 data format
                    HOST     DTID     ARCH   SPEED
ippc1.infop6.cicrp.jussieu.fr    40000    LINUX    1000
ippc3.infop6.cicrp.jussieu.fr    80000    LINUX    1000
pvm> ps
                    HOST      TID     PTID    PID   FLAG 0x COMMAND
ippc1.infop6.cicrp.jussieu.fr   (cons)        -   9368       4/c -
```

```
pvm> quit
pvmd still running.
ippc1.infop6.cicrp.jussieu.fr{leduc}27:
```

2.3 Le démon pvmd

Il est lancé en arrière plan directement sur la machine "père" : pvmd hostfile &. Il suffit de le lancer sur la machine père pour que, automatiquement, il soit lancé à même les fils. Vous ne pouvez en lancer qu'un seul par machine pour un utilisateur donné. Si vous utilisez la spécification pw ("password") dans votre fichier hostfile, ne le lancez pas en arrière plan !

2.4 les directives de compilation et d'éditions de liens

Pour la compilation, ajouter : $-I\$PVM\_ROOT/include$
Pour l'édition de liens, ajouter : $-L\$PVM\_ROOT/lib/\$PVM\_ARCH\ -lpvm3$
Si vous utilisez les "groupes dynamiques", vous devez ajouter :
$-L\$PVM\_ROOT/lib/\$PVM\_ARCH\ -lgpvm3 -lpvm3$

Un Makefile simplifié

2.5 Lancer une application PVM

A partir de la console Procéder comme suit :

```
ippc1.infop6.cicrp.jussieu.fr{leduc}77: pvm
##########################################
Lancement d'une console PVM d'identifiant :
t40001
##########################################
pvm> help spawn
spawn  - Spawn task
Syntax:  spawn [ options ] file [ arg ... ]
Options:  -(count)  number of tasks, default is 1
          -(host)   spawn on host, default is any
          -(ARCH)   spawn on hosts of ARCH
          -?        enable debugging
          ->        redirect output of job to console
          ->(file)  redirect output of job to file
          ->>(file) append output of job to file
          -@        trace job, display output on console
          -@(file)  trace job, output to file
pvm> spawn -2 -> exemple
[1]
2 successful
t40002
t40003
pvm> [1:t40002] Tid = 40002     Pid = 13877
[1:t40002] Le processus Unix 13877 existe toujours !
[1:t40002] EOF
[1:t40003] Tid = 40003  Pid = 13878
[1:t40003] Le processus Unix 13878 existe toujours !
```

```
[1:t40003] EOF
[1] finished

pvm> halt
ippc1.infop6.cicrp.jussieu.fr{leduc}78:
```

A partir d'un shell Commencer par lancer le démon, puis lancer le processus serveur sur la machine père.

2.6 Les entrées-sorties

Il n'y a pas d'entrée standard (stdin) en PVM, il faut donc se contenter de lire des fichiers accessibles en lecture. Les sorties (stdout et stderr) sont redirigées vers la console pour le processus père ou vers les fichiers /tmp/pvml.UID [5] des machines PVM respectives pour les processus fils.

2.7 Quitter PVM

Ne pas confondre *halt* et *quit*. Veiller à ne laisser aucun fichier /tmp/pvm*.UID ni aucun démon pvmd ou tâches PVM sur les différentes machines utilisées.
Au sein d'un code source, veillez à ne pas oublier la primitive pvm_exit(). Si vous stoppez le démon père pvmd, alors tous les démons fils et tous les processus enrolés seront tués aussi. En cas de problème, vous devrez faire tout manuellement : tuer les processus sur chaques clients et nettoyer les répertoires /tmp de toutes les machines.

3 Premiers exemples

3.1 Ne pas confondre getpid() et pvm_mytid(), processus Unix et tâche PVM

Une tâche PVM possède un identifiant global unique (le **TID**) au sein de la machine virtuelle. Cette tâche est exécutée sous forme d'un processus Unix qui possède, lui, un identifiant local unique (le **PID**), propre à la machine physique sur lequel il s'exécute. Le TID propre à chaque tâche permet une designation sans ambiguïté lors de l'échange de messages.
Les commandes:

#include "pvm3.h"	Utilisation de la bibliothèque PVM
int pvm_mytid()	commande de connection avec le démon local de PVM
int pvm_exit()	commande de déconnection avec le démon local de PVM

Dans ces deux cas, si les valeurs de retour sont négatives, cela signifie que la primitive a échouée.

```
#include <stdio.h>
#include "pvm3.h"
/* Compilation :
 * gcc -o $HOME/pvm3/bin/LINUX/exemple exemple-1.c
 * -I$PVM_ROOT/include -L$PVM_ROOT/lib/LINUX -lpvm3
 * Execution :
 * pvmd & ; $HOME/pvm3/bin/LINUX/exemple
 */
main()
{
  int montid = pvm_mytid();
```

5. Ou /tmp/pvml.UID.MACHINE dans le cas d'un système de gestion de fichiers commun à plusieurs architectures différentes.

```
  int monpid = getpid();
  printf("Tid = %x\tPid = %d\n",montid,monpid);
  pvm_exit();
  printf("Le processus Unix %d existe toujours !\n",getpid());
  exit();
  printf("Le processus Unix %d n'existe plus !\n",getpid());
}
```

Autre exemple:

```
#include <stdio.h>
#include "pvm3.h"
/* Compilation :
 * gcc -o $HOME/pvm3/bin/LINUX/exemple exemple-2.c
 * -I$PVM_ROOT/include -L$PVM_ROOT/lib/LINUX -lpvm3
 * Execution :
 * pvmd & ; $HOME/pvm3/bin/LINUX/exemple
 */
main()
{
  int montid = pvm_mytid();
  int monpid = getpid();
  printf("Tid = %x\tPid = %d\n",montid,monpid);
sleep(30);
  pvm_exit();
  printf("Le processus Unix %d existe toujours !\n",getpid());
sleep(30);
  exit();
  printf("Le processus Unix %d n'existe plus !\n",getpid());
}
```

En cours d'exécution, à partir de la console pvm, taper la commande $ps - l$, puis, sous un shell, lister l'ensemble des processus vous concernant. Comparer.

3.2 Le spawn programmé

Les commandes:

int pvm_spawn()	lancer plusieurs tâches
int pvm_catchout()	récupérer les sorties (standard et d'erreur) des fils sur la machine père
int pvm_parent()	récupérer le TID de la tâche père (ou PvmNoParent sinon)

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

/* Compilation :
 * gcc -o $HOME/pvm3/bin/LINUX/exemple exemple-3.c
 * -I$PVM_ROOT/include -L$PVM_ROOT/lib/LINUX -lpvm3
 * Execution :
 * pvmd & ; $HOME/pvm3/bin/LINUX/exemple
 */
```

```
main(int argc,char *argv[])
{
  int montid,info,numt,*tids;

  if (argc == 2)
    {
      pvm_catchout(stdout);

      montid = pvm_mytid();
      tids = (int *) calloc (atoi(argv[1]) , sizeof(int));
      if(pvm_parent()==PvmNoParent)
        {
          /* code du pere */
          if (atoi(argv[1]) == pvm_spawn(argv[0],argv+1,
                      PvmTaskDefault,NULL,atoi(argv[1]),tids))
            printf("Le pere (de TID = %x) a lance %d taches\n",
                      montid,atoi(argv[1]));
        }
      else
        {
          /* code du fils */
          printf("Un fils de tid : %x, de pere : %x\n",montid,pvm_parent());
        }
      info = pvm_exit();
      printf("INFO rendue : %d\n",info);
    }
  else printf("Usage : %s <nombre de fils>\n",argv[0]);
}
```

3.3 L'envoi de message - communications point à point

Un message c'est une enveloppe constituée du TID de l'émetteur, du TID du récepteur, du TAG (ou étiquette) du message, du type et de la taille des données à transférer, plus un contenu (le message lui-même). Commençons par les communications point-à-point.
Pour chaque envoi d'un message en PVM, faire :

- initialiser le tampon d'émission (pvm_initsend()),
- ranger les données (éventuellement hétéroggènes) à envoyer une par une, dans le tampon d'expédition (pvm_pk...()),
- envoyer le message du tampon au(x) destinataire(s) (pvm_send()).

Pour chaque réception d'un message en PVM, faire :

- attendre (bloquant : pvm_recv()) ou vérifier (non bloquant : pvm_nrecv()) que les données sont bien arrivées dans un tampon de réception,
- extraire les données reçues, une par une, du tampon de réception (pvm_upk...())

Les commandes:

int pvm_initsend(C)	le codage C peut-être PvmDataDefault (codage XDR[6] pour un réseau hétérogène, duplication du message en mémoire), PvmDataRaw (pas d'encodage, duplication du message en mémoire) ou PvmDataInPlace (pas d'encodage ni de duplication).
int pvm_pkTYPE(TYPE* dbt,int nbtotal,int pas)	compactage de données d'un même type TYPE au sein du buffer d'émission. On peut appeler cette primitive plusieurs fois pour envoyer des données de types différents au sein d'un même buffer.
int pvm_send(int TID,int TAG)	envoi asynchrone
int pvm_recv(int TID,int TAG)	réception asynchrone (wildcard: valeur -1)
int pvm_upkTYPE(TYPE* dbt,int nbtotal,int pas)	décompactage des données reçues à faire dans l'ordre inverse du compactage correspondant.

En cas d'erreur, les primitives renvoient une valeur entière négative.

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

#define MSG_ALLER 100
#define MSG_RETOUR 101

/* Compilation :
 * gcc -o $HOME/pvm3/bin/LINUX/exemple exemple-4.c
 * -I$PVM_ROOT/include -L$PVM_ROOT/lib/LINUX -lpvm3
 * Execution :
 * pvmd & ; $HOME/pvm3/bin/LINUX/exemple
 */

main(int argc,char* argv[])
{
  int montid,resultat,tids[2];
  pvm_catchout(stdout);

  montid = pvm_mytid();
  if(pvm_parent()==PvmNoParent)
    {
      /* code du pere */
      if (1 == pvm_spawn(argv[0],NULL,PvmTaskDefault,NULL,1,tids))
        printf("Le pere (de TID = %d) a lance la tache %d\n",montid,tids[0]);

      tids[1] = montid;

      pvm_initsend(PvmDataDefault);
      pvm_pkint(tids,2,1);
      pvm_send(tids[0],MSG_ALLER);

      pvm_recv(tids[0],MSG_RETOUR);
      pvm_upkint(&resultat,1,1);
      printf("%d + %d = %d\n",tids[0],tids[1],resultat);
```

```
    }
  else
    {
      /* code du fils */
      pvm_recv(pvm_parent(),MSG_ALLER);
      pvm_upkint(tids,2,1);
      resultat = tids[0] + tids[1];

      pvm_initsend(PvmDataDefault);
      pvm_pkint(&resultat,1,1);
      pvm_send(pvm_parent(),MSG_RETOUR);
    }
  pvm_exit();
}
```

3.4 TD : ensemble de Mandelbrot

Versions avec et sans équilibrage dynamique de tâches.

3.5 Les groupes de processus

Les groupes de processus permettent de partitionner l'ensemble des processus.

int pvm_joingroup(char *group)	cette primitive permet d'incorporer le Pe courant dans le groupe "group" elle renvoie un entier compris entre 0 et pvm_gsize()-1
int pvm_lvgroup(char *group)	cette primitive permet d'exclure le Pe courant du groupe "group".
int pvm_gsize(char *group)	cette primitive renvoie le nombre exact de tâches inclues dans le groupe "group"
int pvm_gettid(char *group,int inum)	cette primitive renvoie le TID de la inum-ième tâche du groupe "group"
int pvm_getinst(char *group,int tid)	cette primitive renvoie le GID (ou rang du processus dans le groupe) de la tâche de TID "tid" au sein du groupe "group"
int pvm_barrier(char *group,int n)	synchronisation des n Pe du groupe "group"

Attention : vous allez créer des groupes dynamiques, n'oubliez pas, à l'édition de liens, d'insérer :

-L$PVM_ROOT/lib/$PVM_ARCH -lgpvm3 -lpvm3

Pairs et impairs

```
#include <stdio.h>
#include "pvm3.h"
#define UNIVERS "Univers"
#define PAIRS "UniversPair"

/* Compilation :
 * gcc -o $HOME/pvm3/bin/$PVM_ARCH/exemple exemple-5.c
 * -I$PVM_ROOT/include -L$PVM_ROOT/lib/$PVM_ARCH -lgpvm3 -lpvm3
 * Execution :
 * pvmd & ; $HOME/pvm3/bin/$PVM_ARCH/exemple
```

```
 */

void pvm_initialisation(int* rang,int * montid,int nbpcs,char *argv[])
{
  /* Cette primitive permet d'activer "nbpcs-1" processus et de les
   * inclure dans le groupe UNIVERS */
  /* int tids[nbpcs-1]; */
  int tids[255];

  pvm_catchout(stdout);
  (*montid) = pvm_mytid();

  if(pvm_parent() == PvmNoParent)
    {
      /* code du pere */
      if ((nbpcs-1) != pvm_spawn(argv[0],argv+1,PvmTaskDefault,NULL,nbpcs-1,tids))
          {
            printf("Erreur au lancement des Pe !\n");
            pvm_exit();
            exit();
          }
    }

  (*rang) = pvm_joingroup(UNIVERS);
  pvm_barrier(UNIVERS,nbpcs);
}

void pvm_finalisation(int nbpcs)
{
  pvm_barrier(UNIVERS,nbpcs);
  pvm_lvgroup(UNIVERS);
  /* pvm_catchout(0); */
  pvm_exit();
}

main(int argc,char* argv[])
{
  int rang,rangPair,montid,nbpcs,nbpcsPairs;
  if (argc == 2)
    {
      nbpcs = atoi(argv[1]);
      pvm_initialisation(&rang,&montid,nbpcs,argv);
      /* ***** ***** ***** ***** ***** ***** ***** */
      nbpcsPairs = nbpcs/2 + (nbpcs%2);

      if ((rang%2) == 0)
        {
          rangPair = pvm_joingroup(PAIRS);
          pvm_barrier(PAIRS,nbpcsPairs);
```

```
          printf("UNIVERS[%d] == PAIRS[%d] == %x\n",rang,rangPair,montid);

          pvm_barrier(PAIRS,nbpcsPairs);
          pvm_lvgroup(PAIRS);
        }
      /* ***** ***** ***** ***** ***** ***** ***** */
      pvm_finalisation(nbpcs);
    }
  else printf("Usage : %s <nb de pcs>\n",argv[0]);
}
```

3.6 L'envoi de message - communications collectives

Faire en une seule opération une série de communications point-à-point.

Type PVM	type C
PVM_BYTE	byte
PVM_SHORT	short
PVM_INT	int
PVM_LONG	long
PVM_FLOAT	float
PVM_DOUBLE	double

int pvm_mcast(int *tids,int ntask,int msgtag)	distribution d'un message d'étiquette msgtag aux Pe de TID : tids[0],... tids[ntask-1]. La réception doit-être programmée explicitement par des pvm_recv()
int pvm_bcast(char *group,int msgtag)	diffuser à l'ensemble du groupe "group" le message d'étiquette "msgtag"
int pvm_scatter(void *result,void *data,int n,int datatype,int msgtag,char *group,int superviseur)	distribution sélective de données (d'une matrice vers des vecteurs par exemple) au sein du groupe "group". Le Pe "superviseur" envoie "n" valeurs (de type datatype) data[0],...,data[n-1] aux Pe du groupe "group" qui les stockent dans result[0],...,result[n-1]
int pvm_gather(void *result,void *data,int n,int datatype,int msgtag,char *group,int superviseur)	collecte sélective de données (de vecteurs vers une matrice par exemple) au sein du groupe "group". Le Pe "superviseur" reçoie n valeurs (de type datatype) data[0],...,data[n-1] des Pe du groupe "group" et les stocke dans result[0],...,result[n-1].
int pvm_reduce(void (*func)(),void *data,int n,int datatype,int msgtag,char *group,int superviseur)	réduction répartie sur un ensemble de processeurs. Opérations de réduction prédéfinies : PVMSUM, PVMPROD, PVMMAX, PVMMIN. Permet d'effectuer une opération de réduction sur les n données data[0],...,data[n-1] (de type datatype) réparties sur chaque Pe du groupe "group" et d'envoyer le résultat global sur le Pe "superviseur"

En cas d'erreur, les primitives renvoient une valeur entière négative.

3.6.1 Le allreduce - un peu d'aléatoire

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "pvm3.h"

#define UNIVERS "Univers"
#define MSG_REDUCE 100
#define MSG_BCAST 101

#define MAX_ITER 100


/* Compilation :
 * gcc -o $HOME/pvm3/bin/$PVM_ARCH/exemple exemple-6.c
 * -I$PVM_ROOT/include -L$PVM_ROOT/lib/$PVM_ARCH -lgpvm3 -lpvm3
 * Execution :
 * pvmd & ; $HOME/pvm3/bin/$PVM_ARCH/exemple
 */

void pvm_initialisation(int* rang,int * montid,int nbpcs,char *argv[])
{
  /* Cette primitive permet d'activer "nbpcs-1" processus et de les
   * inclure dans le groupe UNIVERS */
  /* int tids[nbpcs-1]; */
  int tids[255];

  pvm_catchout(stdout);
  (*montid) = pvm_mytid();

  if(pvm_parent() == PvmNoParent)
    {
      /* code du pere */
      if ((nbpcs-1) != pvm_spawn(argv[0],argv+1,PvmTaskDefault,NULL,nbpcs-1,tids))
          {
            printf("Erreur au lancement des Pe !\n");
            pvm_exit();
            exit(1);
          }
    }

  (*rang) = pvm_joingroup(UNIVERS);
  pvm_barrier(UNIVERS,nbpcs);
}

void pvm_finalisation(int nbpcs)
{
  pvm_barrier(UNIVERS,nbpcs);
```

```
  pvm_lvgroup(UNIVERS);
  /* pvm_catchout(0); */
  pvm_exit();
}

main(int argc,char* argv[])
{
  int rang,montid,nbpcs;
  int valeur,cpt=0;

  if (argc == 2)
    {
      nbpcs = atoi(argv[1]);
      pvm_initialisation(&rang,&montid,nbpcs,argv);
      /* ***** ***** ***** ***** ***** ***** ***** */
      srand((int) clock());

      do {
        valeur = rand()%2;

        pvm_reduce(PvmSum,&valeur,1,PVM_INT,MSG_REDUCE,UNIVERS,0);
        if (rang == 0)
          {
            pvm_initsend(PvmDataDefault);
            pvm_pkint(&valeur,1,1);
            pvm_bcast(UNIVERS,MSG_BCAST);
          }
        else
          {
            pvm_recv(pvm_gettid(UNIVERS,0),MSG_BCAST);
            pvm_upkint(&valeur,1,1);
          }
        cpt++;

      } while ((cpt < MAX_ITER) && (valeur != 0) && (valeur != nbpcs));

      if (rang == 0)
        {
          if ((valeur == 0) || (valeur == nbpcs))
            printf("Egalite sur tous les Pe en %d iterations\n",cpt);
          else printf("Egalite non realisee en %d iterations\n",MAX_ITER);
        }
      /* ***** ***** ***** ***** ***** ***** ***** */
      pvm_finalisation(nbpcs);
    }
  else printf("Usage : %s <nb de pcs>\n",argv[0]);
}
```

3.6.2 Un algorithme de tri...

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

#define MAX(a,b) ((a)>(b) ? (a) : (b))
#define MIN(a,b) ((a)<(b) ? (a) : (b))

#define UNIVERS "Univers"
#define STOP -1

#define MSG_SENDRECV 100
#define MSG_GATHER 101

/* Compilation :
 * gcc -o $HOME/pvm3/bin/LINUX/tri tri.c
 * -I$PVM_ROOT/include -L$PVM_ROOT/lib/LINUX -lgpvm3 -lpvm3
 * Execution :
 * pvmd & ; $HOME/pvm3/bin/LINUX/tri
 */

void pvm_initialisation(int* rang,int * montid,int nbpcs,char *argv[])
{
    /* Cette primitive permet d'activer "nbpcs-1" processus et de les
     * inclure dans le groupe UNIVERS */
    /* int tids[nbpcs-1]; */
    int tids[255];

    /* pvm_catchout(stdout); */
    (*montid) = pvm_mytid();

    if(pvm_parent() == PvmNoParent)
    {
        /* code du pere */
        if ((nbpcs-1) != pvm_spawn(argv[0],argv+1,PvmTaskDefault,NULL,nbpcs-1,tids))
        {
            printf("Erreur au lancement des Pe !\n");
            pvm_exit();
            exit(1);
        }
    }

    (*rang) = pvm_joingroup(UNIVERS);
    pvm_barrier(UNIVERS,nbpcs);
}

void pvm_finalisation(int nbpcs)
{
    pvm_barrier(UNIVERS,nbpcs);
```

```
    pvm_lvgroup(UNIVERS);
    /* pvm_catchout(0); */
    pvm_exit();
}

main(int argc, char *argv[])
{
    int precedant,rang,suivant,montid,nbpcs,rang_du_pere;
    int i,*tab;
    int recue = STOP;
    int envoyee = STOP;
    int stockee = STOP;

    if (argc >= 3)
    {
        nbpcs = argc-1;
        pvm_initialisation(&rang,&montid,nbpcs,argv);
        /* ***** ***** ***** ***** ***** ***** ***** */
        precedant = ((rang == 0) ? STOP : pvm_gettid(UNIVERS,rang-1));
        suivant = ((rang == nbpcs-1) ? STOP : pvm_gettid(UNIVERS,rang+1));

        if (rang == 0)
        {
            stockee = atoi(argv[1]);

            for(i=2 ; i<=nbpcs ; i++)
            {
                /* reception d'une nouvelle valeur */
                recue = atoi(argv[i]);
                /* tests min-max */
                envoyee = MAX(stockee,recue);
                stockee = MIN(stockee,recue);
                /* envoi du max des deux valeurs */
                pvm_initsend(PvmDataDefault);
                pvm_pkint(&envoyee,1,1);
                pvm_send(suivant,MSG_SENDRECV);
            }
            /* recoi l'ordre de terminer du dernier des Pe du groupe */
            pvm_recv(pvm_gettid(UNIVERS,nbpcs-1),MSG_SENDRECV); /**/
            pvm_upkint(&recue,1,1); /**/
        }
        else if (rang == nbpcs-1)
        {
            /* reception d'une nouvelle valeur */
            pvm_recv(precedant,MSG_SENDRECV);
            pvm_upkint(&recue,1,1);
            /* traitement de cette valeur */
            stockee = recue;
            /* envoi l'ordre de terminer aux autres */
            envoyee = STOP;
```

```
            pvm_initsend(PvmDataDefault);
            pvm_pkint(&envoyee,1,1);
            pvm_bcast(UNIVERS,MSG_SENDRECV);
        }
        else
        {
          do {
              /* reception d'une nouvelle valeur */
              pvm_recv(-1,MSG_SENDRECV);
              pvm_upkint(&recue,1,1);
              /* tests */
              if (stockee == STOP)
              {
                  /* premiere valeur recue */
                  stockee = recue;
              }
              else if (recue != STOP)
              {
                  /* tests min-max */
                  envoyee = MAX(stockee,recue);
                  stockee = MIN(stockee,recue);
                  /* envoi du max des deux valeurs */
                  pvm_initsend(PvmDataDefault);
                  pvm_pkint(&envoyee,1,1);
                  pvm_send(suivant,MSG_SENDRECV);
              }
          } while (recue != STOP);
        }

        tab = (int*) calloc (nbpcs,sizeof(int));

        rang_du_pere = pvm_getinst(UNIVERS,pvm_parent());
        if (rang_du_pere < 0)
            rang_du_pere = rang;

        pvm_gather(tab,&stockee,1,PVM_INT,MSG_GATHER,UNIVERS,rang_du_pere);

        if (rang == rang_du_pere)
        {
            for (i=0 ; i<nbpcs ; i++)
                printf("%d, ",tab[i]);
            printf("\n");
        }
        free(tab);
        /* ***** ***** ***** ***** ***** ***** ***** */
        pvm_finalisation(nbpcs);
    }
    else printf("Usage : %s <liste d'au moins 2 valeurs>\n",argv[0]);
}
```

3.7 TD : convolution en imagerie

4 Optimisation

- chevaucher les communications et les calculs (faire en sorte de recouvrir les temps de communication par les temps de calcul).
- éviter la duplication des buffers en mémoire en choisissant le mode PVMDATAINPLACE à l'émission.
- éviter, dans la mesure du possible, le surcoût induit par l'encodage préalable à toute émission de données, dans le cas d'architectures homogènes, en choisissant le mode PVMDATARAW à l'envoi du message.