PVM

Parallel Virtual Machine

Jalel Chergui

Messagerie : Jalel.Chergui@idris.fr

Jean-Michel Dupays

Messagerie : dupays@idris.fr

1 – Introduction

❶ Le modèle de programmation séquentiel :

- ☞ le programme est exécuté par un et un seul processeur ;
- ☞ toutes les variables et constantes du programme sont allouées dans la mémoire centrale du processeur.

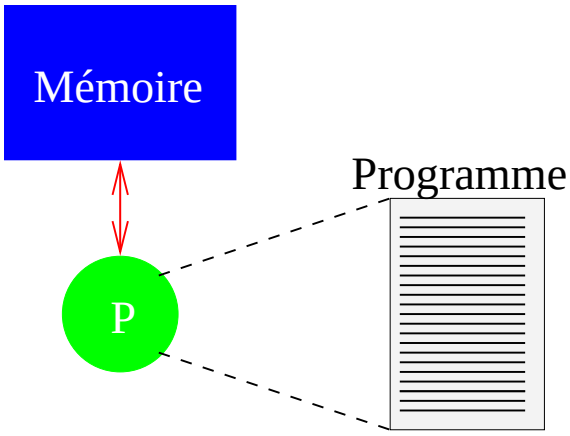

FIG. 1: Modèle séquentiel

❷ Le modèle de programmation par échange de messages :

- ☞ le programme est écrit dans un langage classique (`Fortran`, `C` ou `C++`) ;
- ☞ chaque processus exécute éventuellement des parties différentes d'un programme ;
- ☞ toutes les variables du programme sont privées et résident dans la mémoire locale de chaque processus ;
- ☞ une donnée est échangée entre deux ou plusieurs processus via un appel, dans le programme, à des sous-programmes particuliers.

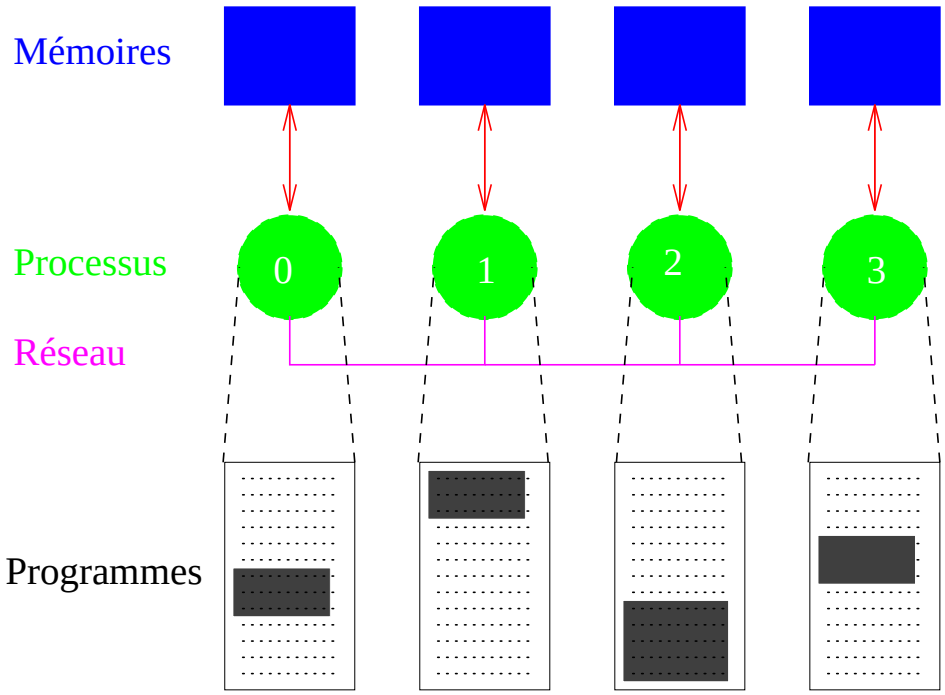

FIG. 2: Modèle à échange de messages

❸ Le modèle d'exécution SPMD :

- ☞ *Single Program, Multiple Data* ;
- ☞ le même programme s'exécute pour tous les processus ;
- ☞ toutes les machines supportent ce modèle de programmation et certaines ne supportent que celui-là ;
- ☞ c'est un cas particulier du modèle plus général MPMD, qu'il peut en revanche émuler.

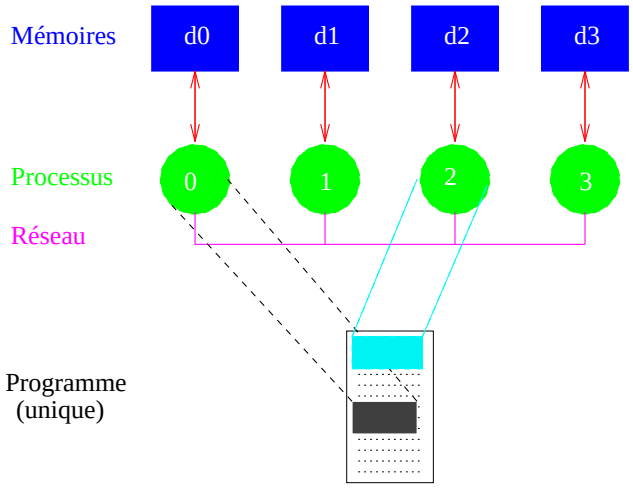

FIG. 3: Single Program, Multiple Data

❹ Exemple en Fortran d'émulation MPMD en SPMD

```
program spmd
if (ProcessusMaître) then
    call LeChef(Arguments)
else
    call LesOuvriers(Arguments)
endif
end program spmd
```

1.1 – Concepts de l'échange de messages

☞ Si un message est envoyé à un processus, celui-ci doit ensuite le recevoir

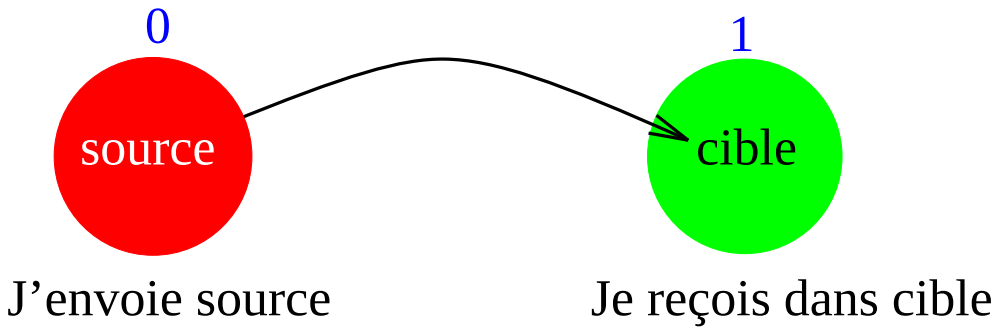

FIG. 4: L'échange de messages

- Un message est constitué de paquets de données transitant du processus émetteur au(x) processus récepteur(s)
- En plus des données (variables scalaires, tableaux, etc.) à transmettre, un message doit contenir les informations suivantes :
  - l'identificateur du processus émetteur ;
  - le type de la donnée ;
  - sa longueur ;
  - l'identificateur du processus récepteur.

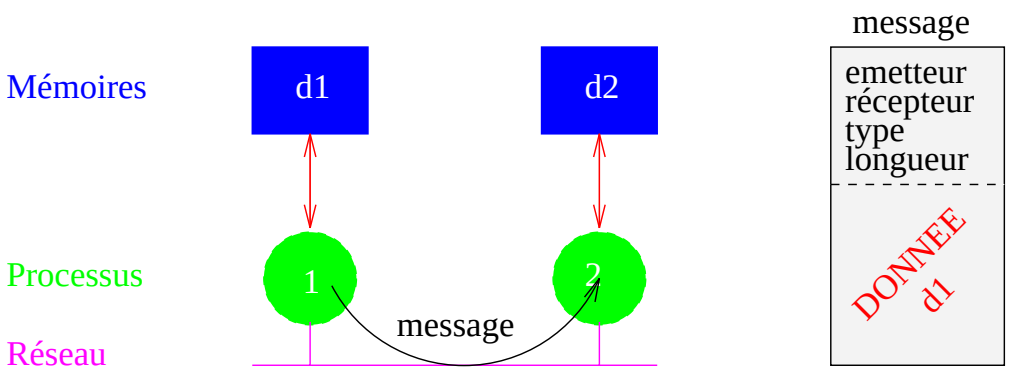

FIG. 5: Constitution d'un message

- Les messages échangés sont interprétés et gérés par un environnement tel que PVM, MPI ou autres
- Cet environnement peut être comparé :
  - à la téléphonie ;
  - à la télécopie ;
  - au courier postal ;
  - à une messagerie électronique ;
  - etc.

- Le message est envoyé à une adresse déterminée
- Cette adresse peut être comparée :
  - à un numéro de téléphone ;
  - à un numéro de télécopie ;
  - à une adresse postale ;
  - à une adresse électronique ;
  - etc.
- Le processus récepteur doit pouvoir classer et interpréter les messages qui lui ont été adressés.

L'environnement en question est PVM (*Parallel Virtual Machine*). Une application PVM est un ensemble de processus autonomes exécutant chacun leur propre code et communiquant via des appels à des sous-programmes de la bibliothèque PVM. Ces sous-programmes peuvent être classés dans les grandes catégories suivantes :

① gestion de l'environnement ;

② communications point à point ;

③ communications collectives ;

④ gestion des groupes.

1.2 – Historique

- Un projet de recherche universitaire démarré à l'université d'*Oak Ridge National Laboratory* en été 1989.
- **1990** : **version 1.0**. Un prototype, jamais rendu public.
- **1991** : **version 2.0**. Écrite à l'Université de *Tennessee*.
- **1992** : **versions 2.1 à 2.4**. Des modifications et des corrections.
- **1993** : **versions 3.0 à 3.3.11**. Une réécriture complète, des extensions et des corrections.
- **1997** : **version 3.4**. Des extensions majeures (introduction des contextes de communication et des groupes statiques de processus).

1.3 – Utilisation

- Pour employer les stations de travail du réseau pendant les heures creuses.
- Pour utiliser efficacement certains multi-processeurs à mémoire répartie (par exemple `CRAY T3E, IBM SP2`, etc.).
- Pour utiliser simultanément des super-calculateurs (ex. `T3E`, `C90`, `VPP`, `SP2`).
- Outil pédagogique pour la formation au parallélisme.

1.4 – Environnement hétérogène

- Les machines peuvent être séquentielles, parallèles, vectorielles, etc., avec des systèmes d'exploitation distincts.
- Les applications relèvent de différents modèles de programmation.
- L'interface réseau entre les machines peut-être du type Ethernet, FDDI, etc.

Dans cet environnement, PVM permet de gérer les machines, les processus et les communications.

1.5 – PVM versus MPI

- PVM est issu d'un projet de recherche universitaire. MPI est issu d'un consortium réunissant des constructeurs, des universitaires et des utilisateurs.
- Si PVM s'applique aussi bien aux grappes de stations qu'aux machines parallèles, MPI-1 ne concerne que les machines parallèles bien qu'il existe des implémentations dont les extensions permettent de l'utiliser sur grappe de stations.
- Les nouvelles fonctionnalités de MPI-2 en font cependant un sur-ensemble de PVM.
- Néanmoins, PVM continue également à évoluer de son côté et intégrera des fonctionnalités se trouvant dans MPI-1 et MPI-2.
- Leur modèle de programmation étant le même (échange de messages), la conversion d'un programme PVM en MPI est en général directe.

1.6 – Bibliographie

- La référence :

  PVM : Parallel Virtual Machine, MIT Press, 1994. Scientific and Engineering Computation Janusz Kowalik, Editor.

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

  ftp ://ftp.irisa.fr :/pub/netlib

- Une documentation complète sur PVM :

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

- Ce cours se trouve sur le serveur de l'IDRIS :

  http ://www.idris.fr/data/cours/parallel/pvm/

Position dans le cours

2 – Environnement

2.1 – Description

- Un processus démon `pvmd3` par machine.
- Une console `pvm` sur la machine hôte.
- Des variables d'environnement (`PVM_ARCH`, `PVM_ROOT`, etc.).
- Trois bibliothèques (`libpvm3.a`, `libfpvm3.a` et `libgpvm3.a`) contenant les modules objets des sous-programmes PVM.

2.2 – Le processus démon pvmd3

```
pvmd3 [ -options ] [ hostfile ]
```

- ☞ il coordonne les différentes machines composant la machine virtuelle ;
- ☞ un démon par utilisateur et par machine ;
- ☞ c'est le démon de la machine hôte qui active automatiquement les démons sur les machines distantes ;
- ☞ c'est une interface entre les processus s'exécutant sur les différentes machines (serveur).

La configuration de la machine virtuelle est réalisée grâce au fichier ¡¡ `hostfile` ¿¿...

```
> cat hostfile
```

```
# Deux stations bi-processeurs superscalaires IBM G30
* lo=dupont dx=$PVM_ROOT/lib/pvmd \
         ep=bin/$PVM_ARCH wd=$PWD sp=1000
mira.idris.fr
vega.idris.fr
& mach1.idris.fr
& mach2.idris.fr
& mach3.idris.fr
& mach4.idris.fr
# Cinq noeuds d'une machine MPP IBM SP2
* lo=dupuis dx=bin/mon_pvmd \
         ep=pvm3/bin/$PVM_ARCH sp=3000
p01.rhea.cnusc.fr
p02.rhea.cnusc.fr
p03.rhea.cnusc.fr
p04.rhea.cnusc.fr
p05.rhea.cnusc.fr
# Un supercalculateur vectoriel CRAY C94
atlas.idris.fr lo=mowgli dx=$PVM_ROOT/lib/pvmd \
         sp=2000
# Une machine MPP CRAY T3E
aleph.idris.fr lo=rbid001 sp=4000
```

```
> pvmd3 hostfile &
```

2.3 – La console pvm

```
> pvm
```

```
pvm> conf
9 hosts, 2 data formats
                    HOST     DTID     ARCH   SPEED
            mira.idris.fr    40000   RS6KMP    1000
            vega.idris.fr    80000   RS6KMP    1000
        p01.rhea.cnusc.fr    c0000      RS6K    3000
        p02.rhea.cnusc.fr   100000      RS6K    3000
        p03.rhea.cnusc.fr   140000      RS6K    3000
        p04.rhea.cnusc.fr   180000      RS6K    3000
        p05.rhea.cnusc.fr   1c0000      RS6K    3000
           atlas.idris.fr   200000      CRAY    2000
           aleph.idris.fr   280000      CRAY    4000
pvm>
```

```
pvm> help
help    - Print helpful information about a command
Syntax:  help [ command ]
Commands are:
  add     - Add hosts to virtual machine
  delete - Delete hosts from virtual machine
  halt     - Stop pvmds
  jobs     - Display list of running jobs
  kill    - Terminate tasks
  mstat   - Show status of hosts
  ps       - List tasks
  pstat   - Show status of tasks
  quit     - Exit console
  reset   - Kill all tasks
  spawn   - Spawn task
  version - Show libpvm version
  ...
pvm>
```

2.4 – Activation des processus

☞ Le sous-programme PVMFMYTID() permet d'intégrer le processus appelant dans l'environnement PVM :

```
integer, intent(out) :: MonTid

call PVMFMYTID(MonTid)
```

☞ Réciproquement, le sous-programme PVMFEXIT() extrait définitivement le processus appelant de cet environnement :

```
integer, intent(out) :: code

call PVMFEXIT(code)
```

☞ Un processus PVM quelconque peut activer d'autres processus sur la machine virtuelle en appelant le sous-programme PVMFSPAWN() :

```
include 'fpvm3.h'
integer,intent(in)            :: nprocs
character(len=5),intent(in) :: MonProg='a.out'
integer, intent(out)          :: numt
integer,intent(out),dimension(0:nprocs-1) :: tids

call PVMFSPAWN( MonProg, Drapeau, Valeur, &
                    nprocs-1,tids(1), numt )
```

Drapeau	Valeur
PVMTASKDEFAULT	'*'
PVMTASKHOST	Nom de machine
PVMTASKARCH	Nom de l'architecture

☞ Les processus (fils), activés à l'aide du sous programme PVMFSPAWN() ou à partir de la console pvm, peuvent connaître l'identificateur du processus qui les a généré (processus père) en appelant le sous-programme PVMFPARENT().

```
integer, intent(out) :: TidDuPere

call PVMFPARENT(TidDuPere)
```

Si le processus appelant ce sous-programme n'a pas de père, la valeur PVMNOPARENT est alors retournée.

```
1  program qui_je_suis
2    implicit none
3    include 'fpvm3.h'
4    integer, allocatable, dimension(:) :: tids
5    integer :: nprocs, MonTid, TidDuPere, numt, info
6
7    call PVMFMYTID(MonTid)
8    call PVMFPARENT(TidDuPere)
9
10   if( TidDuPere == PVMNOPARENT ) then
11
12     read(*,*) nprocs
13     allocate( tids(0:nprocs-1) )
14     tids(0) = MonTid
15     call PVMFSPAWN('QuiJeSuis',PVMTASKDEFAULT,  &
16                    '*', nprocs-1, tids(1), numt)
17     if( numt /= nprocs-1 ) stop
18
19   end if
20   print *,'Mon TID: ',MonTid, &
21           ' ; celui du pere: ',TidDuPere
22
23   call PVMFEXIT(info)
24 end program qui_je_suis
```

```
> cat Donnee
7
> QuiJeSuis < Donnee
Mon TID: 262148 ;  celui du pere: -23
```

2.5 – Gestion dynamique des machines

- Les sous-programmes `PVMFADDHOST()` et `PVMFDELHOST()` permettent respectivement d'intégrer une machine quelconque dans la machine virtuelle et de la retirer.
- Le sous-programme `PVMFMSTAT()` permet de tester si une machine donnée est disponible.

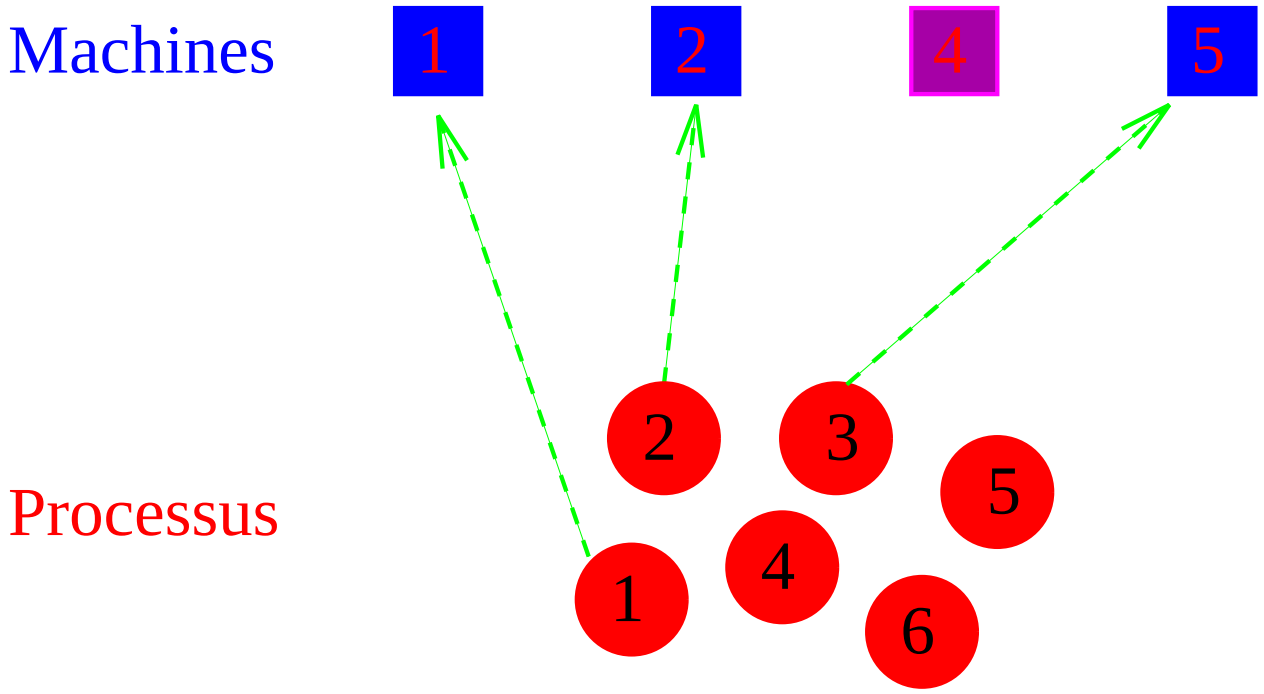

FIG. 6: Gestion des machines

```
1  program machines
2   implicit none
3   include 'fpvm3.h'
4   integer, parameter :: nprocs=6, nmach=4
5   integer, dimension(nprocs) :: tids
6   integer, dimension(nmach)  :: vrai
7   character(len=14),dimension(nmach):: machine
8   integer                    :: info, bonne, numt, k
9   integer                    :: MonTid,TidDuPere,etat
10
11  call PVMFMYTID(MonTid)
12  call PVMFPARENT(TidDuPere)
13  if( TidDuPere == PVMNOPARENT ) then
14    tids(1)=TidDuPere ; bonne = 0 ; k = 1
15    machine(1)='mach1.idris.fr'
16    machine(2)='mach2.idris.fr'
17    machine(3)='mach3.idris.fr'
18    machine(4)='mach4.idris.fr'
```

```
1     do i = 1, nmach
2      call PVMFMSTAT(machine(i), etat)
3      if(etat == 0) then
4        bonne = bonne + 1
5        vrai(bonne) = i
6        call PVMFADDHOST(machine(i), info)
7      endif
8     enddo
9     if (bonne == 0) then
10      print*,'Aucune machine n''est disponible'
11      call PVMFEXIT(info)
12      stop
13    endif
14    do i = 2, nprocs
15      call PVMFSPAWN('MonProg', PVMTASKHOST, &
16                 machine(vrai(k)),1,tids(i),numt)
17      if ( k == bonne ) then
18        k = 1
19      else
20        k = k+1
21      endif
22    enddo
23  endif
24  call PVMFEXIT(info)
25 end program machine
```

2.6 – Impressions sur la sortie standard

- Par défaut, seul le processus père imprime sur la sortie standard de la machine hôte.
- Par défaut, la sortie standard des processus fils est redirigée vers le fichier `/tmp/pvml.UID`.
- Il est possible de rediriger la sortie standard des processus fils sur celle associée à la console `pvm`.
- Un processus quelconque, appelant le sous-programme `PVMFCATCHOUT`(code,...) avec *code* = 1, redirige automatiquement sa sortie standard sur l'ecran du terminal associé à la machine hôte.

```
integer, intent(in)  :: code
integer, intent(out) :: info

call PVMFCATCHOUT(code, info)
```

```
1  program qui_je_suis
2    implicit none
3    include 'fpvm3.h'
4    integer, allocatable, dimension(:) :: tids
5    integer :: nprocs, MonTid, TidDuPere, numt, info
6
7    call PVMFMYTID(MonTid)
8    call PVMFPARENT(TidDuPere)
9    call PVMFCATCHOUT( 1, info)
10
11   if( TidDuPere == PVMNOPARENT ) then
12
13     read(*,*) nprocs
14     allocate( tids(0:nprocs-1) )
15     tids(0) = MonTid
16     call PVMFSPAWN('QuiJeSuis',PVMTASKDEFAULT,  &
17                    '*',nprocs-1, tids(1), numt)
18     if( numt /= nprocs-1 ) stop
19
20   end if
21   print *,'Mon TID: ',MonTid, &
22           ' ; celui du pere: ',TidDuPere
23
24   call PVMFEXIT(info)
25 end program qui_je_suis
```

```
> cat Donnee
7
> QuiJeSuis < Donnee
[t40005]  BEGIN
[t180002] BEGIN
[t140002] BEGIN
[tc0002]  BEGIN
[t100002] BEGIN
[t80002]  BEGIN
          Mon TID: 262148  ; celui du pere: -23
[t180002] Mon TID: 1572866 ; celui du pere: 262148
[t140002] Mon TID: 1310722 ; celui du pere: 262148
[tc0002]  Mon TID: 786434  ; celui du pere: 262148
[t100002] Mon TID: 1048578 ; celui du pere: 262148
[t180002] EOF
[t140002] EOF
[tc0002]  EOF
[t100002] EOF
[t40005]  Mon TID: 262149  ; celui du pere: 262148
[t40005]  EOF
[t80002]  Mon TID: 524290  ; celui du pere: 262148
[t80002]  EOF
```

2.7 – Regrouper des processus

- Un processus peut s'inclure dans un groupe (PVMFJOINGROUP()) ou le quitter (PVMFLVGROUP()).
- Tout processus appartenant à un groupe est identifié par un rang relatif à ce groupe.

```
character(len=7), intent(in)  :: groupe='UNIVERS'
integer, intent(out) :: rang, info

call PVMFJOINGROUP(groupe, rang)

call PVMFLVGROUP(groupe, info)
```

```
1  program qui_je_suis
2    implicit none
3    include 'fpvm3.h'
4    integer, allocatable, dimension(:) :: tids
5    integer :: nprocs,MonTid,TidDuPere,numt,info,rang
6
7    call PVMFMYTID(MonTid)
8    call PVMFPARENT(TidDuPere)
9    call PVMFCATCHOUT( 1, info)
10   call PVMFJOINGROUP('UNIVERS', rang)
11
12   if( TidDuPere == PVMNOPARENT ) then
13     read(*,*) nprocs
14     allocate( tids(0:nprocs-1) )
15     tids(0) = MonTid
16     call PVMFSPAWN('QuiJeSuis',PVMTASKDEFAULT,&
17                    '*',nprocs-1, tids(1), numt)
18   end if
19
20   print *,'Mon TID: ',MonTid, '; Mon rang: ',rang,&
21           '; TID du pere ',TidDuPere
22
23   call PVMFLVGROUP(info)
24   call PVMFEXIT(info)
25 end program qui_je_suis
```

```
> cat Donnee
7
> QuiJeSuis < Donnee
Mon TID: 262148 ; Mon rang: 0; TID du pere -23
Mon TID: 1572866; Mon rang: 3; TID du pere 262148
Mon TID: 1310722; Mon rang: 6; TID du pere 262148
Mon TID: 786434 ; Mon rang: 2; TID du pere 262148
Mon TID: 1048578; Mon rang: 5; TID du pere 262148
Mon TID: 262149 ; Mon rang: 4; TID du pere 262148
Mon TID: 524290 ; Mon rang: 1; TID du pere 262148
```

Pour la suite, on va supposer l'existence d'un sous-programme arbitrairement appelé PVM_INIT(rang, nprocs) dont le but est d'activer nprocs processus, d'inclure tous ces processus dans un groupe que l'on a nommé UNIVERS et de renvoyer dans rang le numéro du processus appelant et dans nprocs le nombre total de processus :

```
program MonProgramme
  implicit none
  integer, intent(out) :: rang, nprocs
  ...
  call PVM_INIT( rang, nprocs )
  ...
end program MonProgramme
```

```
1  subroutine PVM_INIT(rang, nprocs)
2    implicit none
3    include 'fpvm3.h'
4    integer, allocatable, dimension(:) :: tids
5    integer, intent(out) :: rang, nprocs
6    integer :: MonTid, numt, info
7
8    call PVMFMYTID(MonTid)
9    call PVMFCATCHOUT( 1, info)
10   call PVMFJOINGROUP('UNIVERS', rang)
11
12   if( rang == 0 ) then
13     read(*,*) nprocs ; allocate( tids(0:nprocs-1) )
14     tids(0) = MonTid
15     call PVMFSPAWN('MonProg',PVMTASKDEFAULT, &
16                    '*',nprocs-1, tids(1), numt)
17     if ( numt /= nprocs-1 ) stop
18   end if
19 end subroutine PVM_INIT
```

Attention, dans ce sous-programme, la variable nprocs n'est définie que sur le processus 0 !

2.8 – Spécificités sur T3E

- Le T3E est vu comme une machine unique homogène.
- L'exécution d'une application parallèle sur le T3E ne nécessite pas le lancement préalable d'un quelconque démon pvmd3.
- Le suivi et la gestion des processus sur le T3E ne nécessitent pas l'utilisation de la console pvm.
- L'activation des processus sur les nœuds du T3E ne nécessite pas obligatoirement l'appel au sous-programme PVMFSPAWN().
- Les sous-programmes pour la gestion dynamique des machines sont inutiles.
- Tous les processus peuvent imprimer sur la sortie standard.

- Une fois démarrés sur le T3E, les processus sont automatiquement inclus dans un groupe statique dont le nom est affecté à la constante chaîne de caractères PVMALL déclarée dans le fichier d'en-tête **fpvm3.h**.
- Notre sous-programme PVM_INIT() se trouve largement simplifié sur T3E :

```
1  subroutine PVM_INIT(rang, nprocs)
2    implicit none
3    include 'fpvm3.h'
4    integer, intent(out) :: rang, nprocs
5    integer :: info
6
7    call PVMFMYTID(MonTid)
8    call PVMFGETPE(MonTid,rang)
9    call PVMFGSIZE(PVMALL, nprocs)
10
11 end subroutine PVM_INIT
```

Ici, la variable nprocs est définie sur tous les processus.

2.9 – Spécificités sur VPP300

- Le VPP300 est vu comme un ensemble de processeurs vectoriels homogènes interconnectés par un réseaux « crossbar ».
- Pour générer un exécutable parallèle, l'application doit-être chargé avec l'option -Wl,-P à l'édition de liens.
- L'exécution d'une application parallèle SPMD sur le VPP300 se fait sans le lancement préalable d'un quelconque démon pvmd3.
- La soumission, le suivi et la gestion des processus sur le VPP300 se font généralement en « batch » avec NQS.

- L'activation des processus sur les processeurs du VPP300 nécessite obligatoirement l'appel au sous-programme PVMFSPAWN().
- Tous les processus peuvent imprimer sur la sortie standard.
- Notre sous-programme PVM_INIT() sur VPP300 est identique à celui généralement utilisé sur une grappe de stations :

```
1  subroutine PVM_INIT(rang, nprocs)
2    implicit none
3    include 'fpvm3.h'
4    integer, allocatable, dimension(:) :: tids
5    integer, intent(out) :: rang, nprocs
6    integer :: MonTid, numt, info
7
8    call PVMFMYTID(MonTid)
9    call PVMFJOINGROUP('UNIVERS', rang)
10
11   if( rang == 0 ) then
12     read(*,*) nprocs
13     allocate( tids(0:nprocs-1) )
14     tids(0) = MonTid
15     call PVMFSPAWN('MonProg',PVMTASKDEFAULT, &
16                    '*',nprocs-1, tids(1), numt)
17     if ( numt /= nprocs-1 ) stop
18   end if
19 end subroutine PVM_INIT
```

Position dans le cours

3 – Communications point à point

3.1 – Notions générales

☞ Une communication dite point à point a lieu entre deux processus, l'un appelé processus émetteur et l'autre processus récepteur.

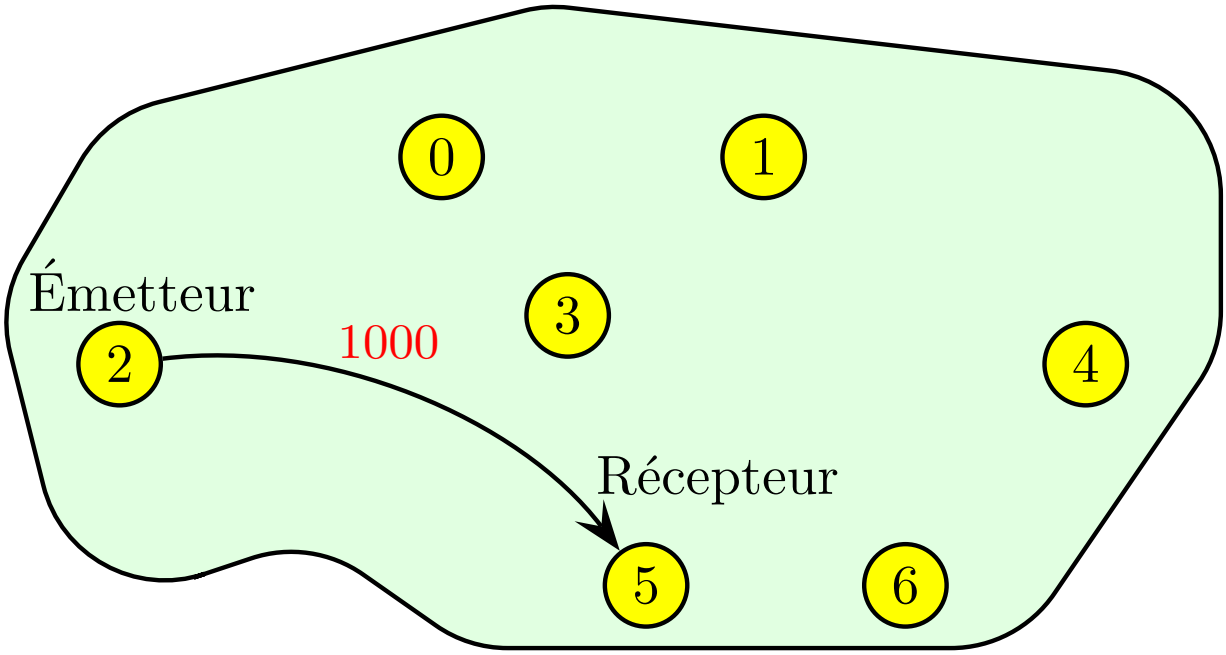

FIG. 7: Communication point à point

- L'émetteur et le récepteur sont identifiés par leur TID (*Task IDentification*).
- Ce que l'on appelle l'enveloppe d'un message est constituée :
  1. du TID du processus émetteur ;
  2. du TID du processus récepteur ;
  3. de l'étiquette (*tag*) du message ;
  4. du type et de la taille des données.
- Les données échangées sont typées (entiers, réels, etc.)
- Dans ce chapitre, nous ne traiterons que de communications bloquantes en réception, mais des communications non-bloquantes sont bien sûr possibles.

```
1  program point_a_point
2   implicit none
3
4   include 'fpvm3.h'
5   integer, parameter :: etiquette=100
6   integer     :: valeur,rang,nprocs,bidon,info
7   integer     :: tid_recepteur,tid_emetteur
8
9   call PVM_INIT(rang,nprocs)
10
11  if (rang == 1) then
12    valeur=1000
13    call PVMFGETTID('UNIVERS', 5, tid_recepteur)
14    call PVMFPSEND(tid_recepteur,etiquette,valeur,1,&
15                   INTEGER4,info)
16  elseif (rang == 5) then
17    call PVMFGETTID('UNIVERS', 1, tid_emetteur)
18    call PVMFPRECV(tid_emetteur,etiquette,valeur,1,&
19                   INTEGER4,bidon,bidon,bidon,info)
20    print *,'Moi, processus 5, j''ai reçu ', &
21            valeur,' du processus 1.'
22  end if
23
24  call PVMFEXIT(info)
25 end program point_a_point
```

```
> cat donnee
7
> point_a_point < donnee
Moi, processus 5, j'ai reçu 1000 du processus 1
```

3.2 – Types de données de base

TAB. 1: Principaux types de données de base (Fortran)

Type PVM	Type Fortran
INTEGER2	INTEGER*2
INTEGER4	INTEGER*4
REAL4	REAL*4
REAL8	REAL*8
COMPLEX8	COMPLEX
COMPLEX16	COMPLEX*16
STRING	CHARACTER
BYTE1	OCTET

TAB. 2: Principaux types de données de base (C)

Type PVM	Type C
PVM_BYTE	byte
PVM_SHORT	signed short
PVM_INT	signed int
PVM_LONG	signed long int
PVM_FLOAT	float
PVM_DOUBLE	double
PVM_CPLX	complex
PVM_DCPLX	double complex

3.3 – Ex. : anneau de communication

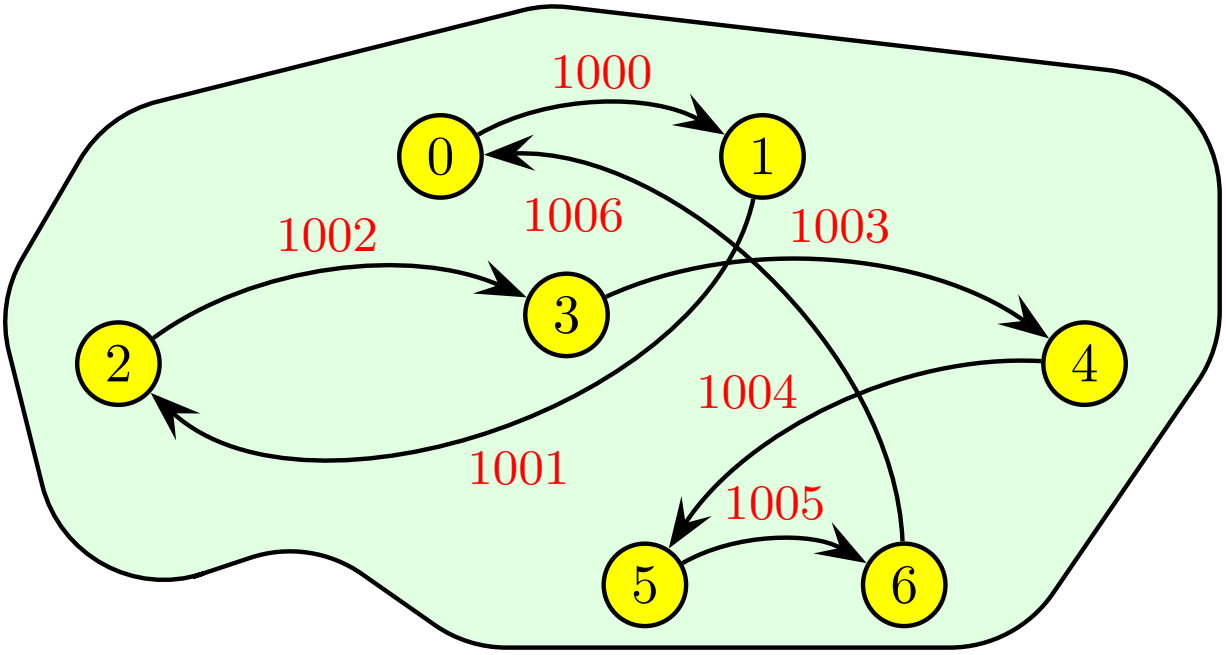

FIG. 8: Anneau de communication

Exemple : anneau de communication


```
1  program anneau
2   implicit none
3   include 'fpvm3.h'
4   integer,parameter :: etiquette=100
5   integer :: nprocs,rang,valeur,bidon,info
6   integer :: num_proc_precedent,num_proc_suivant
7   integer :: tid_proc_precedent,tid_proc_suivant
8
9   call PVM_INIT(rang, nprocs)
10
11  num_proc_precedent=mod(nprocs+rang-1,nprocs)
12  num_proc_suivant=mod(rang+1,nprocs)
13
14  call PVMFGETTID('UNIVERS',num_proc_suivant, &
15                  tid_proc_suivant)
16  call PVMFPSEND(tid_proc_suivant,etiquette,  &
17            rang+1000,1,INTEGER4,info)
18  call PVMFGETTID('UNIVERS',num_proc_precedent, &
19                   tid_proc_precedent)
20  call PVMFPRECV(tid_proc_precedent,etiquette,  &
21           valeur,1,INTEGER4,bidon,bidon,bidon,info)
22
23  print *,'Moi, processus ',rang,', j''ai reçu ', &
24         valeur,' du processus ',num_proc_precedent
25  call PVMFEXIT(info)
26 end program anneau
```

```
> cat donnee
7
> anneau < donnee
Moi, processus 1, j'ai reçu 1000 du processus 0
Moi, processus 3, j'ai reçu 1002 du processus 2
Moi, processus 5, j'ai reçu 1004 du processus 4
Moi, processus 0, j'ai reçu 1006 du processus 6
Moi, processus 2, j'ai reçu 1001 du processus 1
Moi, processus 4, j'ai reçu 1003 du processus 3
Moi, processus 6, j'ai reçu 1005 du processus 5
```

```
include 'fpvm3.h'
integer :: tid_recepteur,etiquette,nb,info
PVMFPSEND(tid_recepteur,etiquette,variable,nb, &
                  type,info)
```

```
include 'fpvm3.h'
integer :: tid_emetteur,etiquette,nb,info, &
           stid,setiquette,staille
PVMFPRECV(tid_emetteur,etiquette,variable,nb,type,&
                 stid,setiquette,staille,info)
```

Les sous-programmes PVMFPSEND() et PVMFPRECV() supposent que :

- ☞ les données échangées sont homogènes ;
- ☞ les données échangées sont contigües en mémoire ;
- ☞ les deux processus impliqués se trouvent sur des machines sur lesquelles la représentation des nombres est identique.

3.4 – Échange de données hétérogènes

☞ L'envoi de message se fait en trois étapes :

① attribution d'un identificateur (ou pointeur) associé à un espace mémoire temporaire (*buffer*) ;

② compactage des données éventuellement hétérogènes (entières, réelles, caractères, etc.) dans cet espace mémoire temporaire ;

③ envoi proprement dit du message contenu dans cet espace mémoire temporaire.

- La réception d'une message se fait en deux étapes :
  1. réception du message dans un espace mémoire temporaire (*buffer*) ;
  2. décompactage des données éventuellement hétérogènes (entières, réelles, caractères, etc.) en mémoire centrale.
- Le message peut-être constitué d'un ensemble d'éléments d'un tableau espacés d'un pas constant en mémoire.

```
1  program heterogene
2    implicit none
3    include 'fpvm3.h'
4    integer,parameter :: etiquette=100
5    integer           :: rang,nprocs,bufid,info,genre
6    integer           :: tid_recepteur,tid_emetteur
7    character(len=7)  :: nom
8    real              :: poids
9    call PVM_INIT(rang,nprocs)
10   if (rang == 1) then
11      nom='chameau' ; genre=1010 ; masse=100.8
12      call PVMFINITSEND(PVMDATADEFAULT, bufid)
13      call PVMFPACK(STRING, nom, 1, 1, info)
14      call PVMFPACK(INTEGER4, genre, 1, 1, info)
15      call PVMFPACK(REAL4, poids, 1, 1, info)
16      call PVMFGETTID('UNIVERS', 5, tid_recepteur)
17      call PVMFSEND(tid_recepteur, etiquette, info)
18   elseif (rang == 5) then
19      call PVMFGETTID('UNIVERS', 1, tid_emetteur)
20      call PVMFRECV(tid_emetteur, etiquette, bufid)
21      call PVMFUNPACK(STRING, nom, 1, 1, info)
22      call PVMFUNPACK(INTEGER4, genre, 1, 1, info)
23      call PVMFUNPACK(REAL4, poids, 1, 1, info)
24   end if
25   call PVMFEXIT(info)
26 end program heterogene
```

```
include 'fpvm3.h'
integer, intent(out) :: bufid
PVMFINITSEND( Encodage, bufid )
```

TAB. 3: Les valeurs du paramètre Encodage

Encodage	Opérations/Utilités
PVMDATADEFAULT	Formatage XDR. Message dupliqué en mémoire. Pour machines hétérogènes.
PVMDATARAW	Pas de formatage XDR. Message dupliqué en mémoire. Pour machines homogènes.
PVMDATAINPLACE	Pas de formatage XDR. Message non dupliqué. Pour machines homogènes.

3.5 – Gestion de *buffer* multiples

☞ Par défaut, il existe deux *buffers* actifs par processus :

① un pour l'envoi ;

② un pour la réception.

- Les sous-programmes `PVMFMKBUF()` et `PVMFFREEBUF()` permettent de gérer plusieurs *buffers* aussi bien pour l'envoi que pour la réception.
- Les sous-programmes `PVMFSETSBUF()` et `PVMFSETRBUF()` permettent de désigner un nouveau *buffer* actif respectivement pour l'envoi et pour la réception. L'ancien *buffer* actif est alors sauvegardé.
- Les sous-programmes `PVMFGETSBUF()` et `PVMFGETRBUF()` permettent de connaître les identificateurs de *buffers* actifs courants respectivement pour l'envoi et pour la réception.
- Le sous-programme `PVMFINITSEND()` est en réalité une combinaison des sous-programmes `PVMFFREEBUF()` et `PVMFMKBUF()`.

3.6 – Distribution sélective d'un message

- Un processus donné peut distribuer (`PVMFMCAST()`) un message à un ensemble prédéterminé de processus.
- Le processus distributeur appel le sous-programme `PVMFMCAST()`.
- Les processus récepteurs font une réception explicite du message en appelant, par exemple, le sous-programme `PVMFPRECV()` ou le sous-programme `PVMFRECV()`.
- La distribution d'un message à l'aide du sous-programme `PVMFMCAST()` est plus efficace qu'une boucle explicite sur un appel au sous-programme `PVMFSEND()` ou `PVMFPSEND()`. Ceci est d'autant plus vrai que le nombre de processus impliqué dans cette distribution est plus élevé.

```
1  program distribution
2    implicit none
3    include 'fpvm3.h'
4    integer, parameter :: etiquette=100
5    integer            :: rang,nprocs,bufid,info
6    integer            :: tid_emetteur,tids(3)
7    real               :: A(500,600), B(600)
8
9    call PVM_INIT(rang,nprocs)
10   if (rang == 0) then
11      call random_number(a)
12      call PVMFINITSEND(PVMDATARAW, bufid)
13      call PVMFPACK(REAL4, A(4,1), 600, 500, info)
14      call PVMFGETTID('UNIVERS', 1, tids(1))
15      call PVMFGETTID('UNIVERS', 3, tids(2))
16      call PVMFGETTID('UNIVERS', 5, tids(3))
17      call PVMFMCAST(3,  tids, etiquette, info)
18   elseif (rang==1 .or. rang==3 .or. rang==5) then
19      call PVMFGETTID('UNIVERS', 0, tid_emetteur)
20      call PVMFRECV(tid_emetteur, etiquette, bufid)
21      call PVMFUNPACK(REAL4, B, 600, 1, info)
22      print *,'Moi, processus ',rang,', j''ai reçu',&
23                 ' A(4,:) dans B(:)'
24   end if
25   call PVMFEXIT(info)
26 end program distribution
```

```
> cat donnee
7
> distribution < donnee
Moi, processus 3, j'ai reçu A(4,:) dans B(:)
Moi, processus 1, j'ai reçu A(4,:) dans B(:)
Moi, processus 5, j'ai reçu A(4,:) dans B(:)
```

3.7 – Optimisation des communications

- L'optimisation doit être un souci essentiel lorsque la part des communications par rapport aux calculs devient assez importante
- L'optimisation des communications peut s'accomplir à différents niveaux dont les principaux sont :
  1. éviter si possible la duplication du message en mémoire (*buffering*) en choisissant le mode `PVMDATAINPLACE` à l'envoi du message ;
  2. éviter si possible les surcoûts induits par le formatage des données lorsque les machines sont homogènes en choisissant le mode `PVMDATARAW` à l'envoi du message ;

③ transférer les grands volumes de données en mode `PVMROUTEDIRECT` pour éviter l'engorgement des communications au niveau des démons `pvmd3`.

④ chevaucher les communications avec les calculs en utilisant le sous-programme `PVMFNRECV()` pour la réception non-bloquante.

```
1  program non_bloquant
2   implicit none
3   include 'fpvm3.h'
4   integer, parameter     :: na=256,nb=200
5   integer, parameter     :: m=2048,etiquette=1111
6   real, dimension(na,na):: a
7   real, dimension(nb,nb):: b
8   real, dimension(m,m)   :: c
9   real, dimension(na)    :: pivota
10  real, dimension(nb)    :: pivotb
11  integer                :: rang,nprocs,info, &
12                            drapeau,bufid,oldval &
13                            tid_emetteur,tid_recepteur
14
15 !*** Initialisation PVM
16  call PVM_INIT( rang, nprocs )
17
18 !*** Initialisation des tableaux
19  call random_number(a)
20  call random_number(b)
21
22 !*** Choix du protocole de communication
23  if (rang == 0 .or. rang == 1) &
24    call PVMFSETOPT(PVMROUTE,PVMROUTEDIRECT,oldval)
```

```
25  if (rang == 0) then
26    call random_number(c)
27 !*** J'envoie un gros message
28    call PVMFINITSEND(PVMDATAINPLACE, bufid )
29    call PVMFPACK(REAL4,c,m*m,1,info)
30    call PVMFGETTID('UNIVERS', 1, tid_recepteur)
31    call PVMFSEND(tid_recepteur, etiquette, info)
32 !*** Ce calcul modifie le contenu du tableau C
33    c(1:nb,1:nb) = matmul(a(1:nb,1:nb),b)
34  elseif (rang == 1) then
35    drapeau = 0 ; bufid = 0
36    call PVMFGETTID('UNIVERS', 0, tid_emetteur)
37    do while ( bufid == 0 )
38 !*** Je reçois le gros message, si arrivé
39      call PVMFNRECV(tid_emetteur, etiquette, bufid)
40      if ( bufid > 0 ) then
41 !*** Le message est reçu, je le décompacte
42        call PVMFUNPACK(REAL4,c,m*m,1,info)
43 !*** Ce calcul dépend du message précédent
44        a(:,:) = transpose(c(1:na,1:na))
45      elseif( bufid == 0 .and. drapeau == 0) then
46 !*** Ce calcul est indépendant du message
47        call sgetrf(nb, nb, b, nb, pivotb, info)
48        drapeau = 1 ; end if ; end do ; end if
49  call PVMFEXIT(info)
50 end program non_bloquant
```

- Il est possible de tester l'arrivée d'un message avant sa réception (`PVMFPROBE()`).
- Une fois un message reçu, le sous-programme `PVMFBUFINFO()` renvoit des informations sur son contenu (taille, étiquette et TID du processus émetteur).

```
 ...
 elseif (rang == 1) then
   drapeau = 0 ; bufid = 0
   do while ( bufid == 0 )
!*** Je teste l'arrivée d'un message
     call PVMFPROBE(-1,etiquette,bufid)
     if ( bufid > 0 ) then
!*** Un message est arrivé
       call PVMFBUFINFO(bufid,taille,etiquette, &
                        tid_emetteur,info)
       call PVMFRECV(tid_emetteur,etiquette,bufid)
!*** Le message est reçu, je le décompacte
       call PVMFUNPACK(REAL4,c,taille/4,1,info)
!*** Ce calcul dépend du message précédent
       a(:,:) = transpose(c(1:na,1:na))
     elseif( bufid == 0 .and. drapeau == 0) then
!*** Ce calcul est indépendant du message
       call sgetrf(nb, nb, b, nb, pivotb, info)
       drapeau = 1
     end if
   end do
 end if
```

3.8 – Spécificités sur T3E

- Sur T3E, la famille des sous-programmes send et recv admettent en argument aussi bien un TID qu'un numéro de processus.
- Sur T3E, le type INTEGER8 est une extension CRAY aux types INTEGER2 et INTEGER4 prédéfinis par PVM.

```
1  program distribution_t3e
2    implicit none
3    include 'fpvm3.h'
4    integer, parameter :: etiquette=100
5    integer            :: rang,nprocs,bufid,bidon,info
6    integer            :: rangs(3), valeur
7    real               :: A(500,600), B(600)
8    call PVM_INIT(rang,nprocs)
9    if (rang == 0) then
10
11      call random_number(a) ; valeur = 1010
12      call PVMFINITSEND(PVMDATARAW, bufid)
13      call PVMFPACK(INTEGER8, valeur, 1, 1, info)
14      call PVMFPACK(REAL8, A(4,1), 600, 500, info)
15      rangs(:) = (/ 1, 3, 5 /)
16      call PVMFMCAST(3,  rangs, etiquette, info)
17
18   elseif (rang==1 .or. rang==3 .or. rang==5) then
19
20      call PVMFRECV(0, etiquette, bufid)
21      call PVMFUNPACK(INTEGER8, valeur, 1, 1, info)
22      call PVMFUNPACK(REAL8, B, 600, 1, info)
23
24   end if
25   call PVMFEXIT(info)
26 end program distribution_t3e
```

- Le mécanisme d'échange de messages sur T3E est basé sur un dispositif de copie (shmem_put()) ou chargement (shmem_get()) de mémoire à mémoire.
- L'envoi de messages avec PVMFPSEND() s'effectue en un ou deux temps selon que la taille du message est respectivement $\leq$ ou $>$ à la valeur contenue dans la variable d'environnement PVM_DATA_MAX (par défaut, 4096 octets).

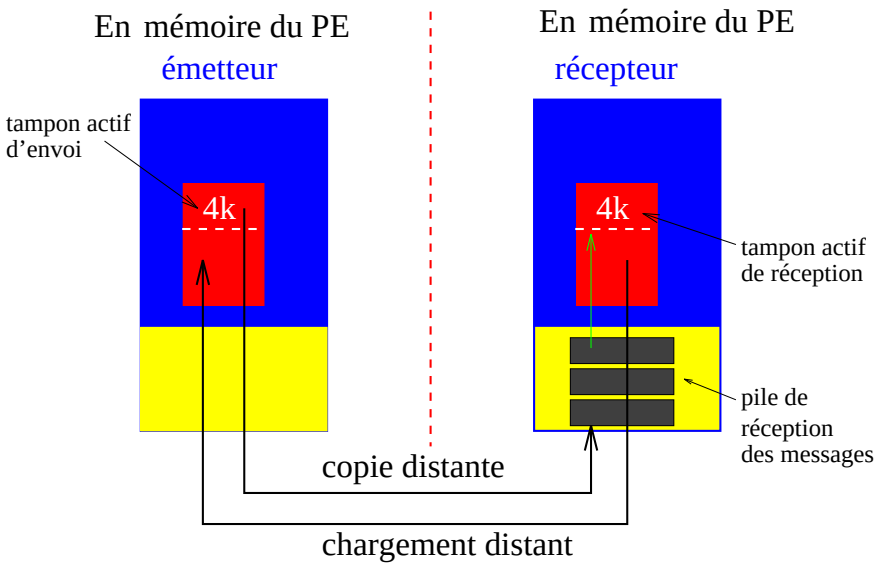

FIG. 9: Implémentation sur T3E

☞ L'attribution des ressources PVM sur un processeur du T3E peut être contrôlée par des variables d'environnement :

① PVM_DATA_MAX : taille (en octets) du message initial (multiple de 8). Par défaut, 4096 octets.

② PVM_RETRY_COUNT : nombre maximum de tentatives d'envoi du message à un autre processus. Au-delà de ce nombre PVM abandonne les tentatives et renvoit un message d'erreur du type PvmOutOfResSMP. Par défaut ce nombre est égal à 500.

③ PVM_SM_POOL : nombre de *buffers* système dans la queue de réception de messages. Par défaut max(10,2*nprocs).

- La variable d'environnement PVM_DATA_MAX est un des moyens d'optimiser les communications point à point sur T3E.
- Certaines variables peuvent être positionnées via le sous-programme PVMFSETOPT(). Parmi celles présentées, seule la valeur de PVMRETRYCOUNT peut être modifiée via ce sous-programme.

3.9 – Spécificités sur VPP300

L'attribution des ressources PVM sur un processeur du VPP300 peut être contrôlée par des variables d'environnement :

① À l'exécution, la zone tampon totale utilisée pour le transfert des messages (c-a-d. la zone de stockage temporaire correspondant au maximum des données en cours de transfert à un instant donné) a une taille fixée par la variable VPP_MBX_SIZE. Sa valeur par défaut est de 4 Moctets. Elle doit être exprimée en octets et en multiple de 4.
Pour diverses raisons, cette zone peut-être saturé notamment par une fréquence élevée d'envois de gros messages à destination d'un même processus pendant que celui-ci calcule. L'exécution peut alors s'arrèter avec un message d'erreur. Une parade serait de positionner la variable d'environnement VPP_MBX_SIZE à une valeur supérieure à 4 Moctets avant de relancer l'exécution.

② Une autre zone tampon propre à PVM sert à stocker chacun des messages (une boîte aux lettres). Sa longueur maximale a une taille fixée par la variable PVM_BUF_SIZE. Sa valeur par défaut est 400 Koctets. Elle doit être exprimée en octets et être egale à un multiple de 4.

Position dans le cours

4 – Communications collectives

4.1 – Notions générales

- Les communications collectives permettent de faire en une seule opération une série de communications point à point.
- Une communication collective concerne toujours les processus d'un groupe donné.

☞ Il y a trois types de fonctions :

❶ celle qui assure les synchronisations globales : `PVMFBARRIER()`.

❷ celles qui ne font que transférer des données :

- ❑ diffusion globale de données : `PVMFBCAST()` ;
- ❑ diffusion sélective de données : `PVMFSCATTER()` ;
- ❑ collecte de données réparties : `PVMFGATHER()` ;

❸ celles qui, en plus de la gestion des communications, effectuent des opérations (somme, produit, maximum, minimum, etc.) sur les données transférées : `PVMFREDUCE()`

4.2 – Synchronisation : PVMFBARRIER

```
integer :: info, nprocs

call PVMFBARRIER("groupe",nprocs,info)
```

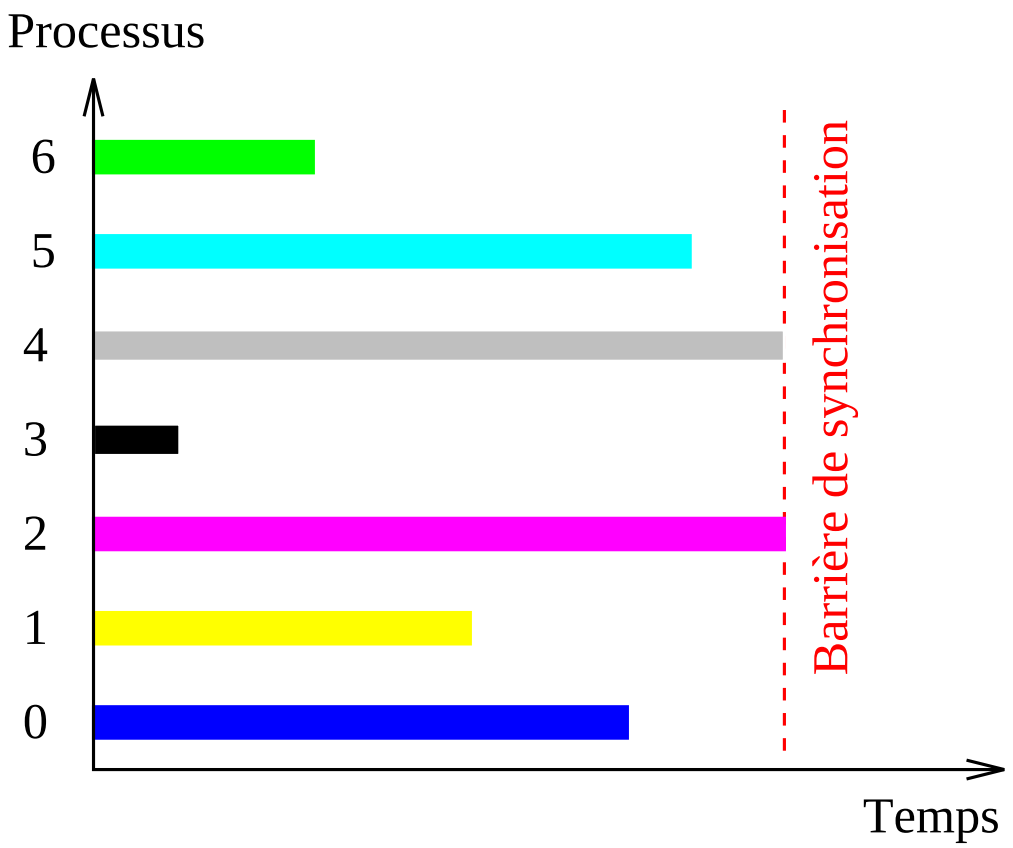

FIG. 10: Barrière globale de synchronisation

4.3 – Diffusion générale : PVMFBCAST

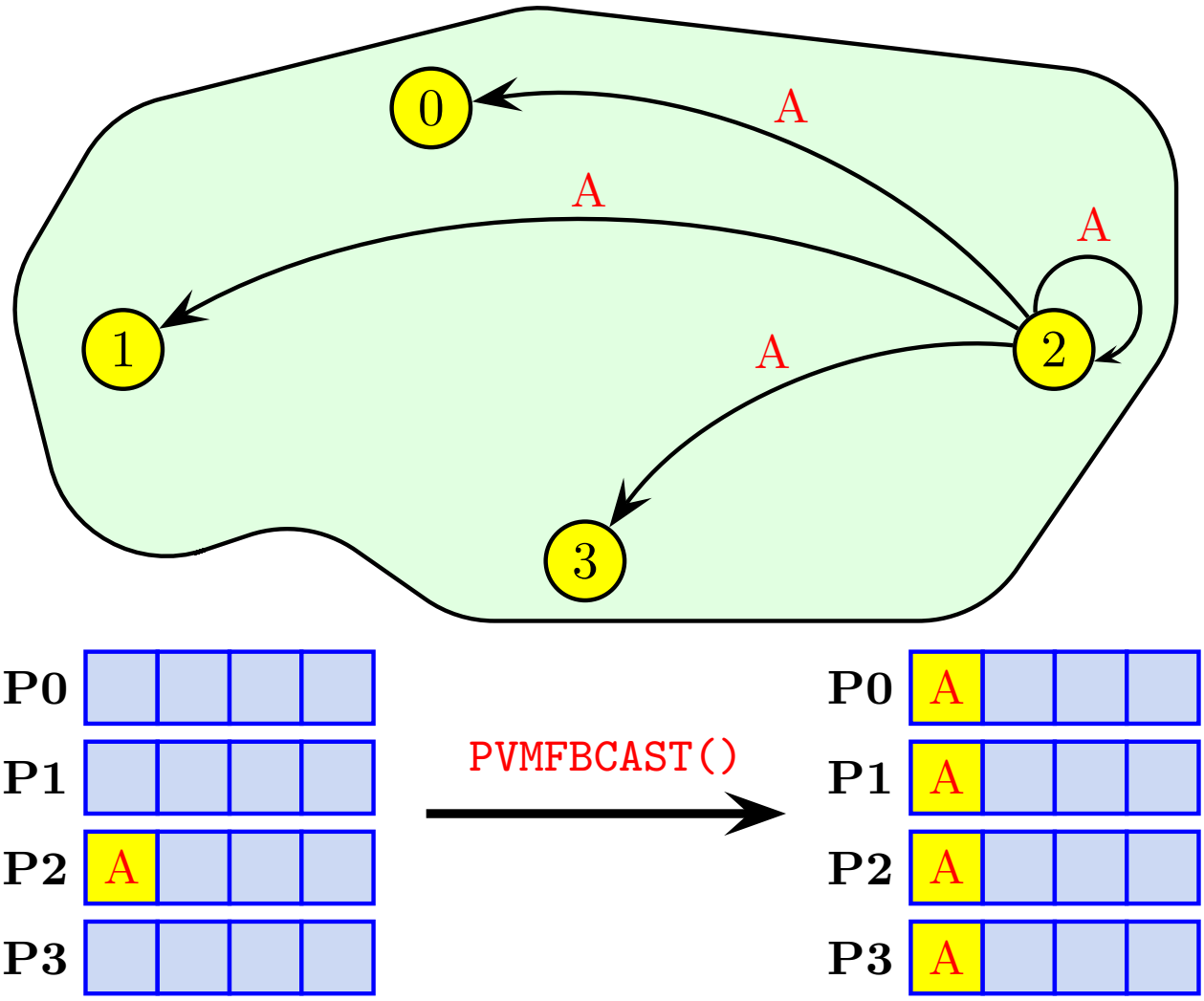

FIG. 11: Diffusion générale : PVMFBCAST

```
1  program bcast
2    implicit none
3    include 'fpvm3.h'
4    integer :: nprocs=4,etiquette=100
5    integer :: rang,valeur,bufid,info, &
6               bidon,tid_emetteur
7
8    call PVM_INIT(rang, nprocs)
9    call PVMFBARRIER("UNIVERS", nprocs, info)
10
11   if (rang == 2) then
12     valeur=rang+1000
13     call PVMFINITSEND(PVMDATADEFAULT, bufid)
14     call PVMFPACK(INTEGER4, valeur,1, 1, info)
15     call PVMFBCAST("UNIVERS", etiquette, info)
16   else
17     call PVMFPGETTID("UNIVERS", 2, tid_emetteur)
18     call PVMFPRECV(tid_emetteur,etiquette, valeur,&
19             1,INTEGER4,bidon,bidon,bidon,info)
20     print *,'Moi, processus ',rang,', j''ai reçu ',&
21           valeur,' du processus 2'
22   end if
23   call PVMFBARRIER("UNIVERS", nprocs, info)
24   call PVMFLVGROUP("UNIVERS",info)
25   call PVMFEXIT(info)
26 end program bcast
```

```
> cat donnee
4
> bcast < donnee
Moi, processus 0, j'ai reçu 1002 du processus 2
Moi, processus 1, j'ai reçu 1002 du processus 2
Moi, processus 3, j'ai reçu 1002 du processus 2
```

4.4 – Diffusion sélective : PVMFSCATTER

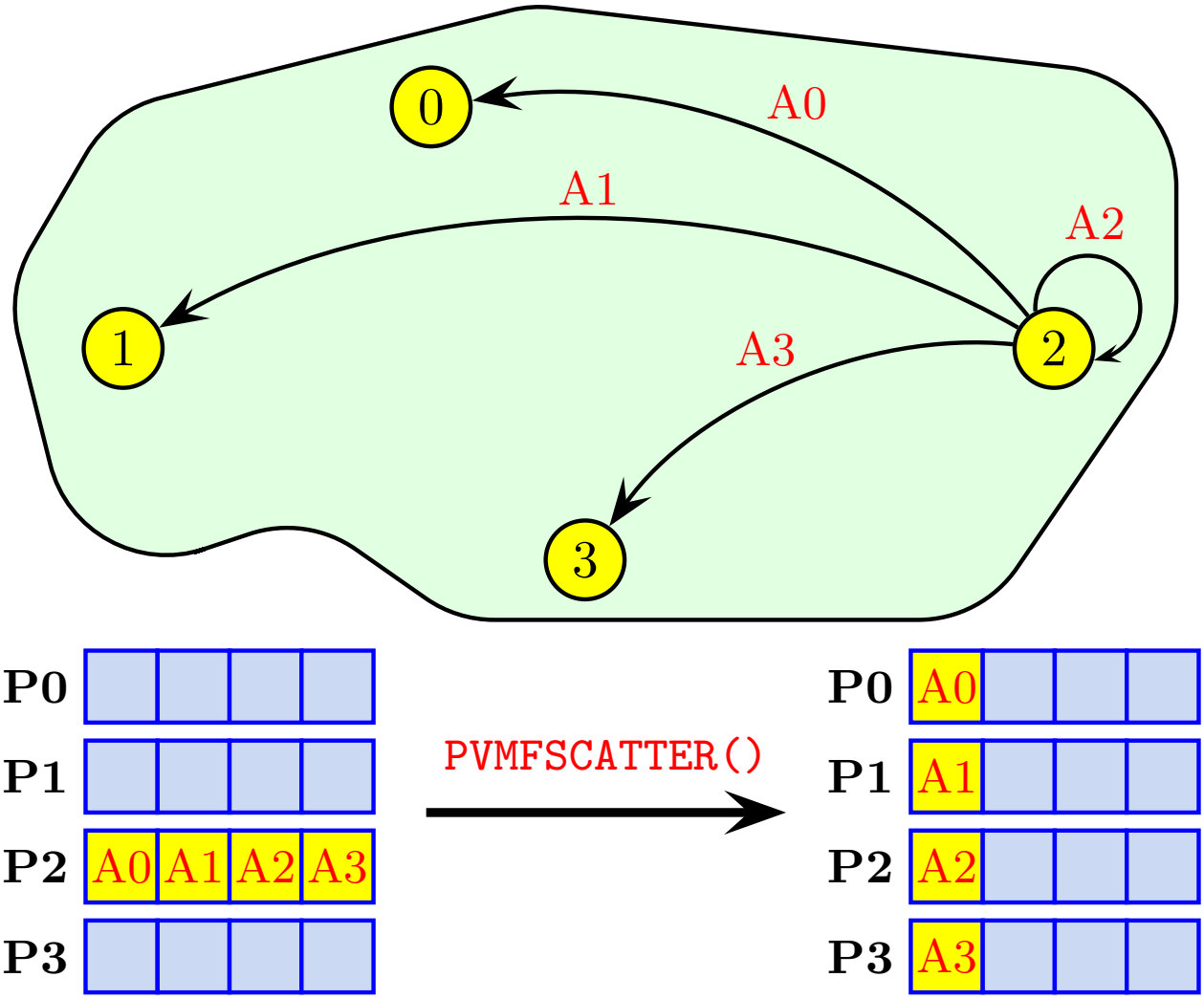

FIG. 12: Diffusion sélective : PVMFSCATTER

```
1  program scatter
2   implicit none
3
4   include 'fpvm3.h'
5   integer,parameter :: nb_valeurs=128
6   integer           :: nprocs=4,etiquette=100
7   integer           :: rang,longueur_tranche,i,info
8   real,allocatable,dimension(:) :: valeurs,donnees
9
10  call PVM_INIT(rang, nprocs)
11  call PVMFBARRIER("UNIVERS", nprocs, info)
12
13  longueur_tranche=nb_valeurs/nb_processus
14  allocate(donnees(longueur_tranche))
15
16  if (rang == 2) then
17    allocate(valeurs(nb_valeurs))
18    valeurs(:)=(/(1000.+i,i=1,nb_valeurs)/)
19  end if
```

```
20  call PVMFSCATTER(donnees,valeurs,longueur_tranche,&
21         REAL4,etiquette,"UNIVERS",2,info)
22
23  print *,'Moi, processus ',rang,', j''ai reçu ', &
24    donnees(1),' à ',donnees(longueur_tranche), &
25    ' du processus 2'
26
27  call PVMFBARRIER("UNIVERS", nprocs, info)
28  call PVMFLVGROUP("UNIVERS",info)
29  call PVMFEXIT(info)
30 end program scatter
```

```
> cat donnee
4
> scatter < donnee
Moi, pr. 0, j'ai reçu 1001. à 1032. du pr. 2
Moi, pr. 1, j'ai reçu 1033. à 1064. du pr. 2
Moi, pr. 3, j'ai reçu 1097. à 1128. du pr. 2
Moi, pr. 2, j'ai reçu 1065. à 1096. du pr. 2
```

4.5 – Collecte : PVMFGATHER

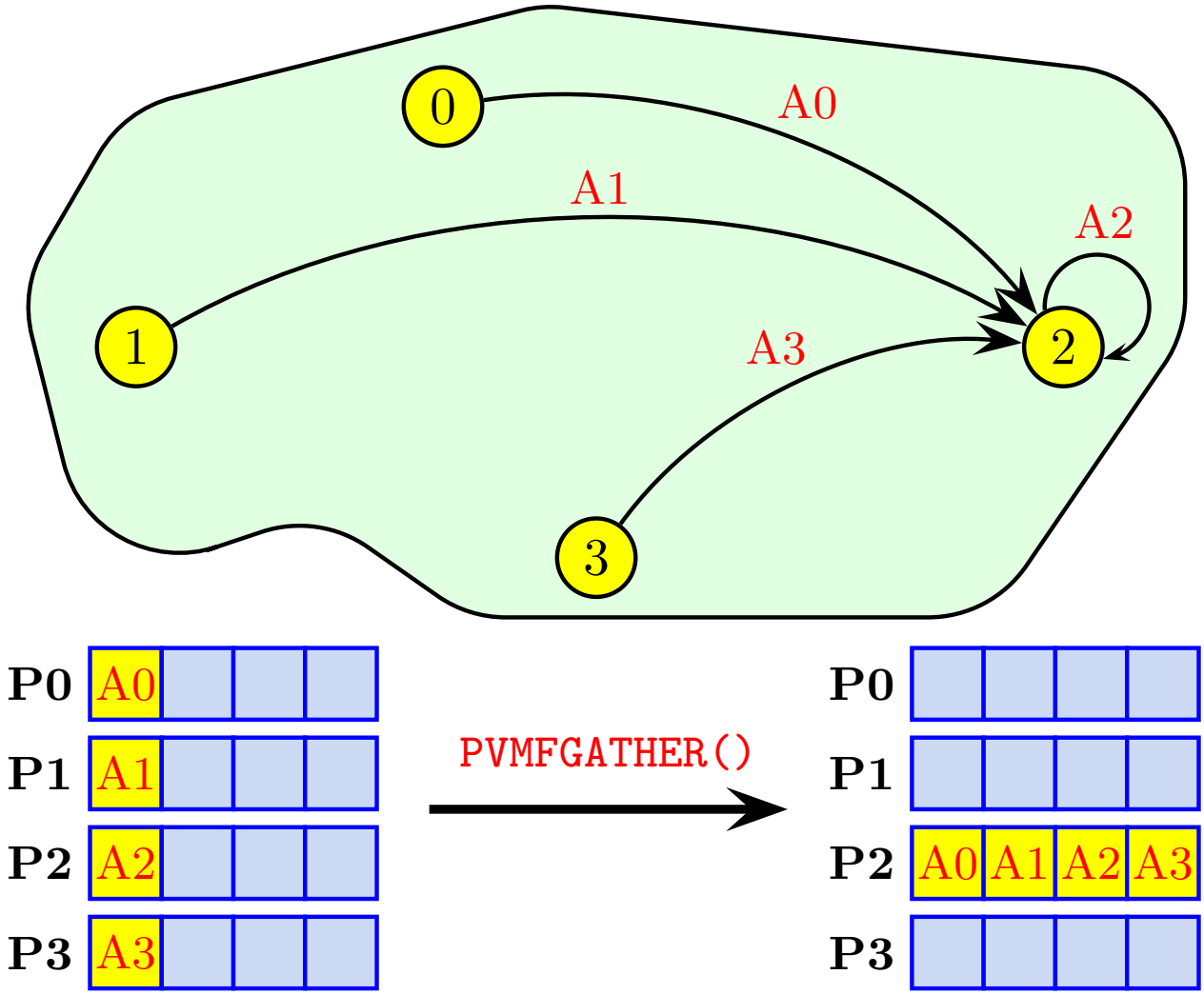

Fig. 13: Collecte : PVMFGATHER

```
1  program gather
2   implicit none
3
4   include 'fpvm3.h'
5   integer, parameter :: nb_valeurs=128
6   integer            :: nprocs=4, etiquette=100
7   integer            :: rang,longueur_tranche,i,info
8   real, dimension(nb_valeurs)     :: donnees
9   real, allocatable, dimension(:) :: valeurs
10
11  call PVM_INIT(rang, nprocs))
12  call PVMFBARRIER("UNIVERS", nprocs, info)
13
14  longueur_tranche=nb_valeurs/nb_processus
15  allocate(valeurs(longueur_tranche))
16
17  valeurs(:)=(/(1000.+rang*longueur_tranche+i, &
18                 i=1,longueur_tranche)/)
19
20  call PVMFGATHER(donnees,valeurs,longueur_tranche,&
21         REAL4,etiquette,"UNIVERS",2,info)
```

```
22  if (rang == 2) &
23     print *,'Moi, processus 2', j''ai reçu ', &
24             donnees(1),
25             ' ... ',donnees(longueur_tranche+1), &
26             ' ... ',donnees(nb_valeurs)'
27
28  call PVMFBARRIER("UNIVERS", nprocs, info)
29  call PVMFLVGROUP("UNIVERS",info)
30  call PVMFEXIT(info)
31 end program gather
```

```
> cat donnee
4
> gather < donnee
Moi, pr. 2, j'ai reçu 1001. ... 1033. ... 1128.
```

4.6 – Réductions réparties : PVMFREDUCE()

- Une réduction est une opération appliquée à un ensemble d'éléments pour en obtenir une seule valeur. Des exemples typiques sont la somme des éléments d'un vecteur (`SUM(A( :))` ou la recherche de l'élément de valeur maximum dans un vecteur (`MAX(V( :))`.
- PVM propose un sous-programme (`PVMFREDUCE()`) de haut-niveau pour opérer des réductions sur des données réparties sur un ensemble de processus, avec récupération du résultat sur un seul processus.
- Si plusieurs éléments sont concernés par processus, la fonction de réduction est appliquée à chacun d'entre eux, en faisant d'abord les réductions locales sur chaque processus.

Tab. 4: Les opérations de réduction prédéfinies

Nom	Opération
PVMSUM	Somme des éléments
PVMPROD	Produit des éléments
PVMMAX	Recherche du maximum
PVMMIN	Recherche du minimum

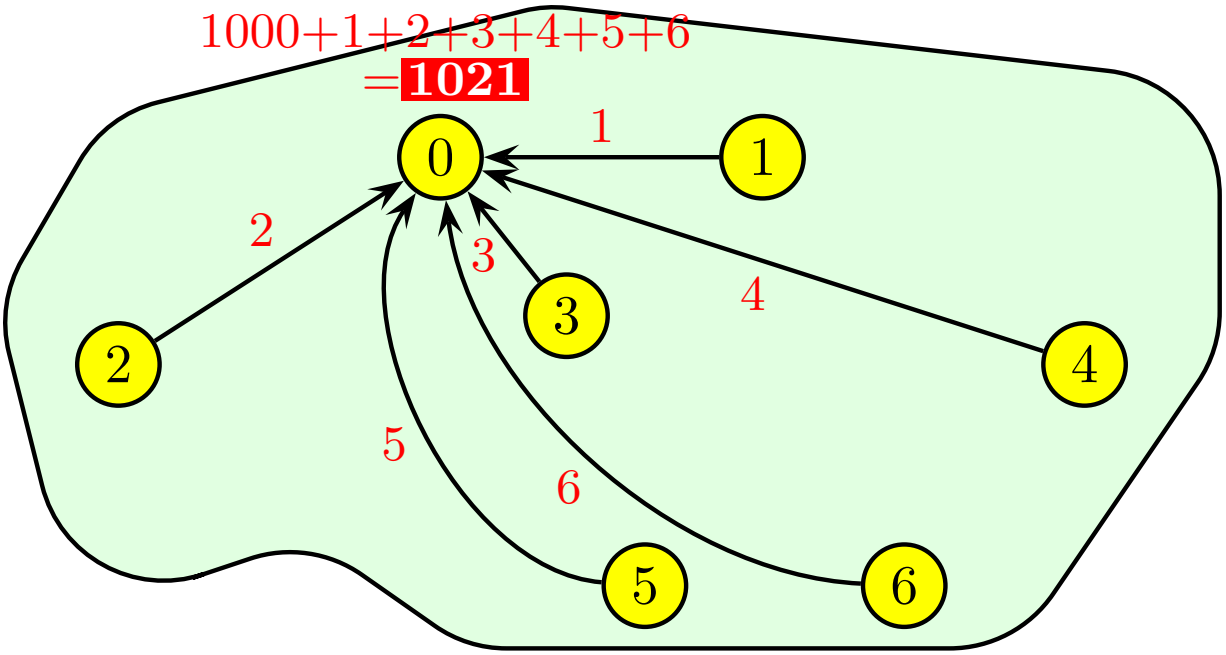

FIG. 14: Réduction répartie (somme)

```
1  program reduce
2   implicit none
3   include 'fpvm3.h'
4   integer :: nprocs=4,etiquette=100
5   integer :: rang,valeur,info
6   external PVMSUM
7
8   call PVM_INIT(rang, nprocs)
9   call PVMFBARRIER("UNIVERS", nprocs, info)
10  if (rang == 0) then
11     valeur=1000
12  else
13     valeur=rang
14  endif
15
16  call PVMFREDUCE(PVMSUM,valeur,1,INTEGER4, &
17                  etiquette,"UNIVERS",0,info)
18
19  if (rang == 0) &
20     print *,'Moi, processus 0, j''ai pour ', &
21              'valeur de la somme globale ',valeur
22  call PVMFBARRIER("UNIVERS", nprocs, info)
23  call PVMFLVGROUP("UNIVERS",info)
24  call PVMFEXIT(info)
25 end program reduce
```

```
> cat donnee
7
> reduce < donnee

Moi, pr. 0, j'ai pour v. de la somme globale 1021
```

Position dans le cours

5 – Les groupes de processus

Il s'agit de partitionner un ensemble de processus afin de créer des sous-ensembles sur lesquels on puisse effectuer des opérations collectives.

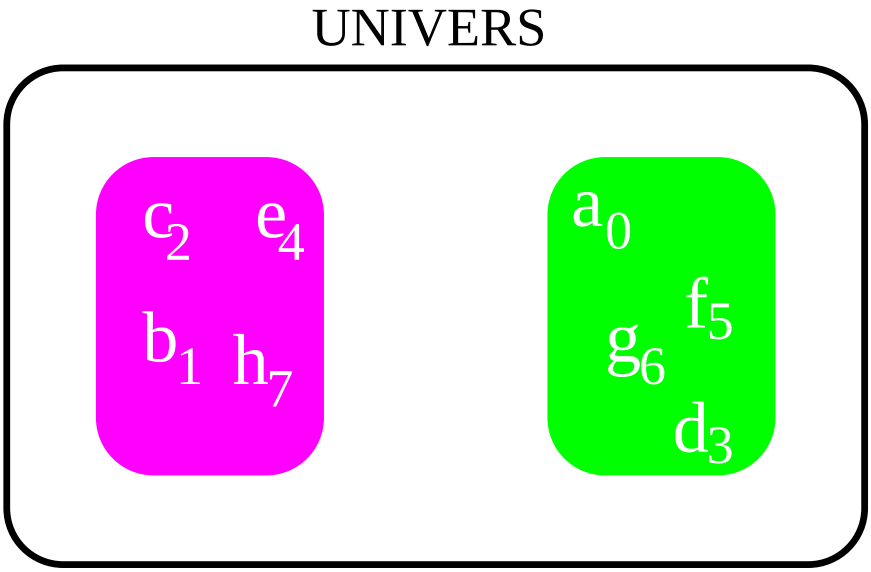

FIG. 15: Partitionnement d'un ensemble de processus

5.1 – Exemple pratique

Dans l'exemple suivant, le processus 2 collecte un vecteur "A" de tous les processus (une fois activés) appartenant au groupe unique dénommé arbitrairement « `UNIVERS` » et créé lors de l'appel à notre sous-programme `PVM_INIT`.

```
1  program monde
2   include 'fpvm3.h'
3   integer, parameter :: m=100,etiquette=100
4   integer            :: nprocs=8,rang,info
5   real, dimension(m)        :: A
6   real, dimension(m,nprocs) :: B
7
8   call PVM_INIT(rang, nprocs)
9   call PVMFBARRIER('UNIVERS', nprocs, info)
10
11  a(:)=real(rang)
12
13  call PVMFGATHER(B,A,m,REAL4,etiquette,'UNIVERS', &
14                 2,info)
15
16  if(rang == 2) print *,B(:,:)
17  call PVMFBARRIER('UNIVERS', nprocs, info)
18  call PVMFLVGROUP('UNIVERS', info)
19  call PVMFEXIT(info)
20 end program monde
```

```
> cat donne
8
> monde < donne
```

UNIVERS

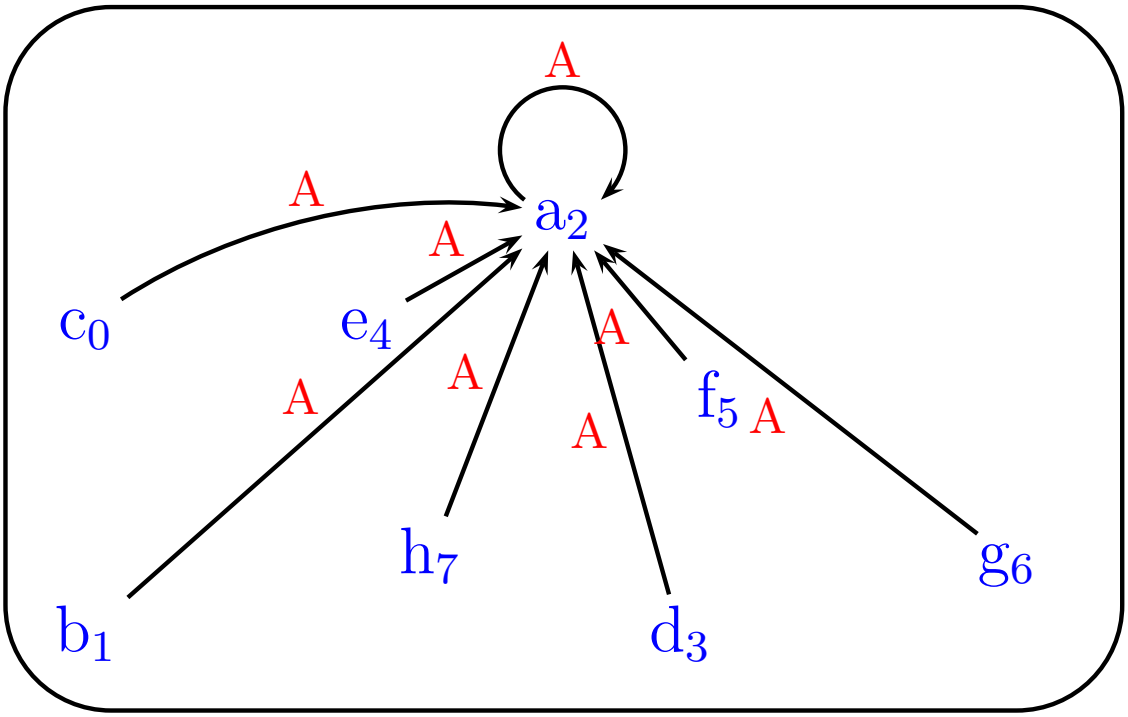

FIG. 16: Constitution d'un groupe unique dénomé ici UNIVERS

Que faire pour que le processus 2 ne collecte les données que des processus de rangs pairs par exemple ?

- ☞ Utiliser les sous-programmes *send/recv* peut être très pénalisant surtout si le nombre de processus est élevé.
- ☞ Une solution efficace consiste à inclure ces processus dans un groupe et à faire en sorte que seuls ces processus appellent le sous-programme `PVMFGATHER()` pour participer à cette collecte.

5.2 – Création/destruction d'un groupe

Dans la bibliothèque PVM, il existe un sous-programme pour :

① créer et inclure (une fois créé) un processus dans un groupe : PVMFJOINGROUP() ;

② qu'un processus puisse quitter et détruire (s'il est le dernier à sortir) un groupe : PVMFLVGROUP().

5.3 – Définition d'un groupe

- Un groupe est un ensemble ordonné de N processus.
- Chaque processus du groupe est identifié par un entier $0,\ 1,\ \ldots,\ N-1$ appelé rang ou instance.
- Le rang d'un processus est relatif au groupe auquel il appartient.
- Les sous-programmes `PVMFGSIZE()` et `PVMFGETINST()` permettent de connaître respectivement la taille d'un groupe et le rang d'un processus dans un groupe à partir de son `TID`.
- Inversement, le sous-programme `PVMFGETTID()` permet de connaître le `TID` d'un processus connaissant son rang dans un groupe.

5.4 – Exemple pratique (suite)

Dans cette exemple, nous allons :

- ☞ regrouper les processus de rang pair dans un groupe ;
- ☞ ne collecter les données que des processus de ce groupe.

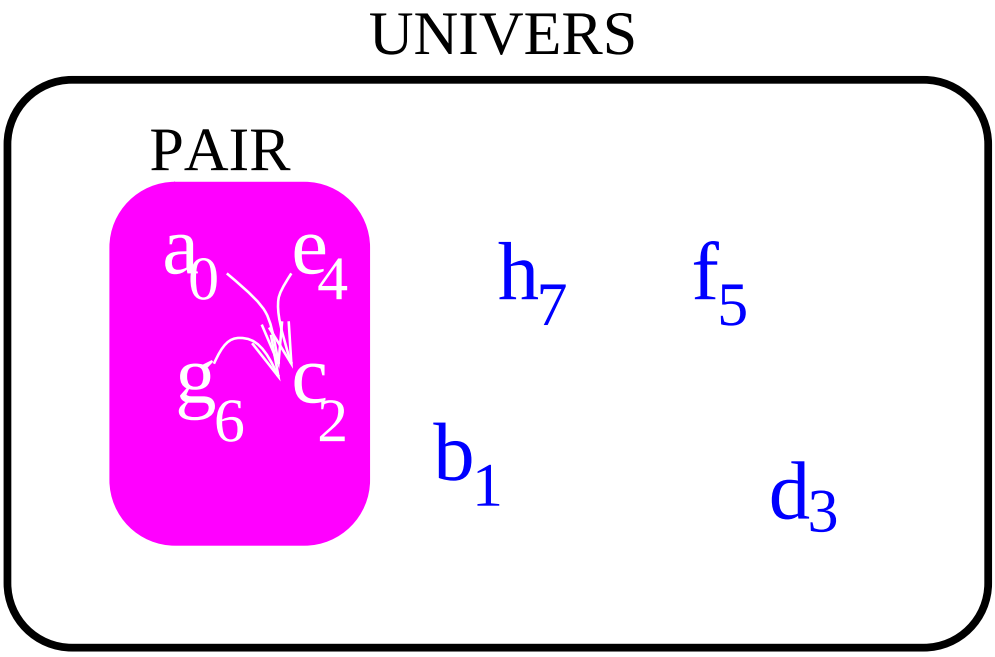

FIG. 17: Un groupe dans un autre

```
1  program GroupePair
2   include 'fpvm3.h'
3   integer,parameter :: m=16,etiquette=100
4   integer           :: nprocs=8,rang,info,tid_de_2
5   integer           :: rang_de_2_dans_pair, &
6                        nprocs_pair,rang_dans_pair
7   real, dimension(m)           :: A
8   real, dimension(m,nprocs/2) :: B
9
10  call PVM_INIT(rang, nprocs)
11 !*** Initialisation du vecteur A
12  A(:) = real(rang)
13 !*** Le nombre de processus de rang pair
14  nprocs_pair = int(nprocs/2) + 1
```

```
15  if( mod(rang,2) == 0 ) then
16    call PVMFJOINGROUP('PAIR', rang_dans_pair)
17    call PVMFBARRIER('PAIR', nprocs_pair, info)
18    call PVMFGETTID('UNIVERS', 2, tid_de_2)
19    call PVMFGETINST('PAIR',tid_de_2, &
20              rand_de_2_dans_pair)
21
22    call PVMFGATHER(B,A,m,REAL4,etiquette, &
23              'PAIR',rand_de_2_dans_pair,info)
24
25    if(rang_dans_pair == rand_de_2_dans_pair) &
26        print *,B(:,:)
27    call PVMFBARRIER('PAIR', nprocs_pair, info)
28    call PVMFLVGROUP('PAIR', info)
29  end if
30
31  call PVMFBARRIER('UNIVERS', nprocs, info)
32  call PVMFLVGROUP('UNIVERS', info)
33  call PVMFEXIT(info)
34 end program GroupePair
```

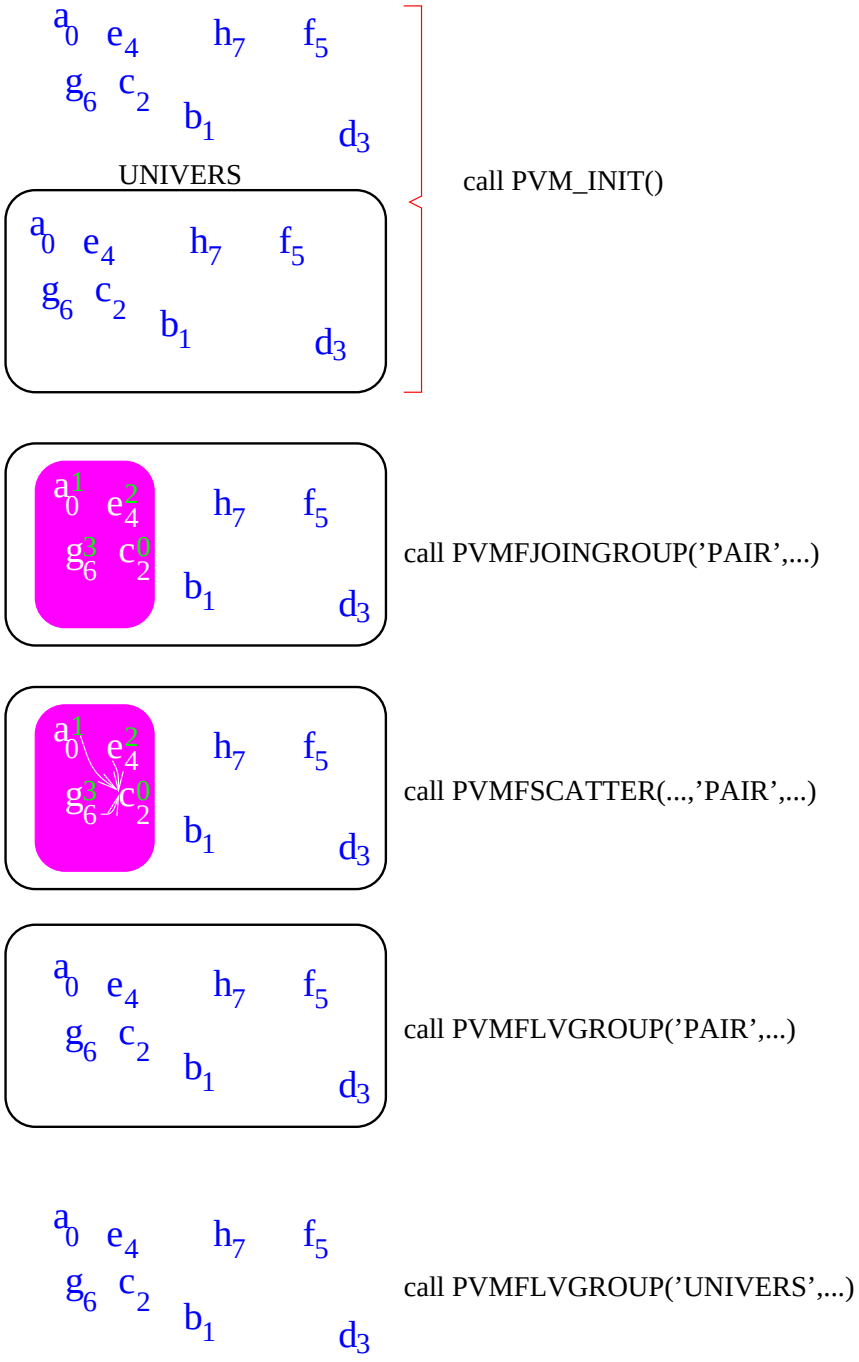

FIG. 18: Création/destruction du groupe PAIR

5.5 – Spécificités sur T3E

- Tous les processus de l'application sont inclus automatiquement dans le groupe `PVMALL` dès le démarrage de l'exécution.
- Le groupe `PVMALL` est statique. Il ne peut être par conséquent ni modifié (en taille avec `PVMFJOINGROUP()`) ni détruit (avec `PVMFLVGROUP()`).
- Il est en revanche possible de gérer dynamiquement des sous-groupes à l'intérieur du groupe `PVMALL`.

Position dans le cours

6 – Les outils

- Du domaine publique ➪ l'interface `XPVM`.
- Propriétaires CRAY sur `T3E` ➪ `apprentice` et `totalview`.
- Propriétaires Fujitsu sur `VPP300` ➪ `mpt` et `totalview`.

6.1 – L'interface XPVM

C'est un outil X qui permet la visualisation des événements PVM :

- ☞ pour les performances ;
- ☞ pour le débogage des communications.

Le source est disponible dans le domaine public à l'adresse :

http ://www.netlib.org/pvm3/xpvm/index.html

Une documentation intéressante se trouve à l'adresse :

http ://www.netlib.org/utk/icl/xpvm/xpvm.html

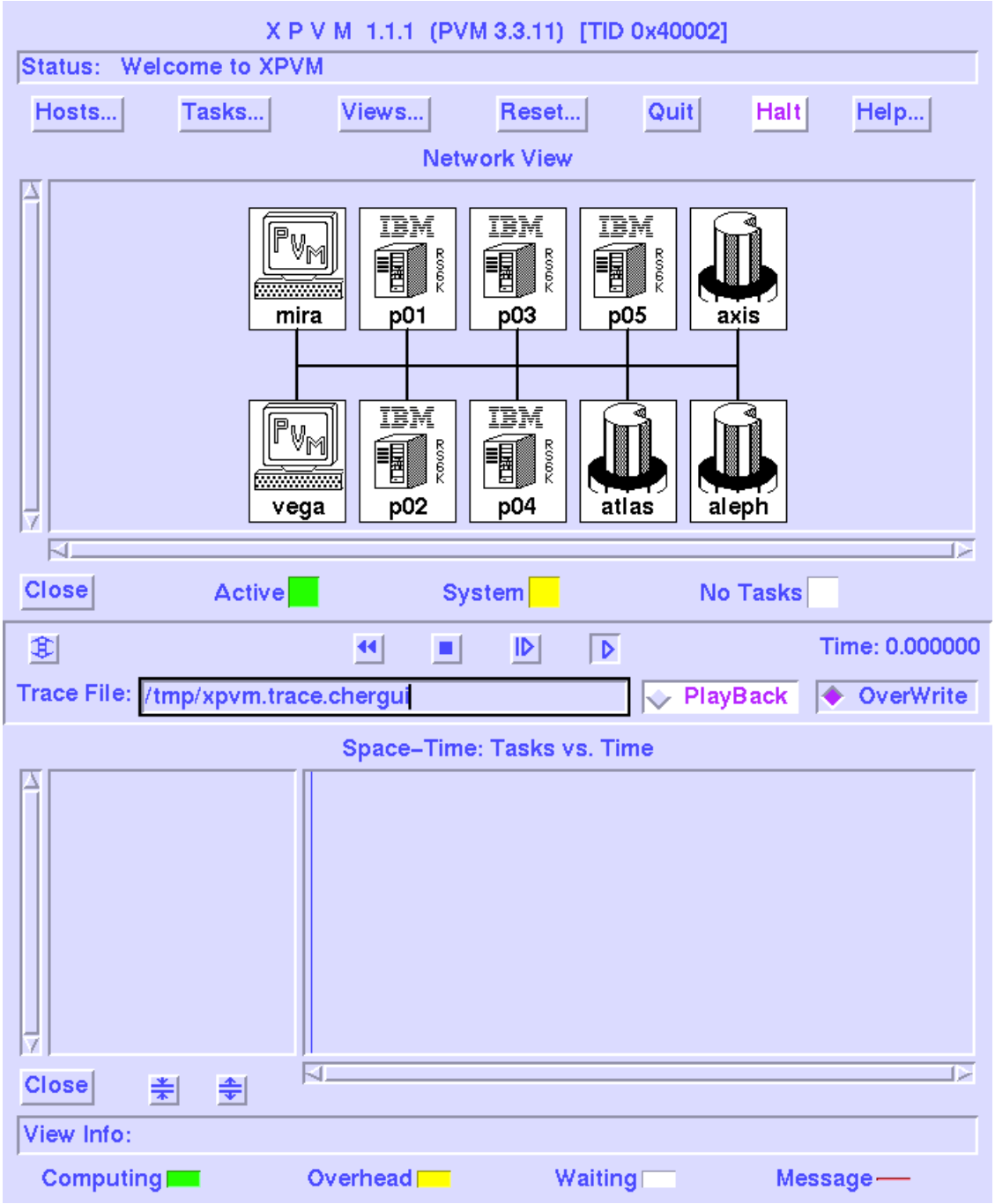

FIG. 19: La machine virtuelle

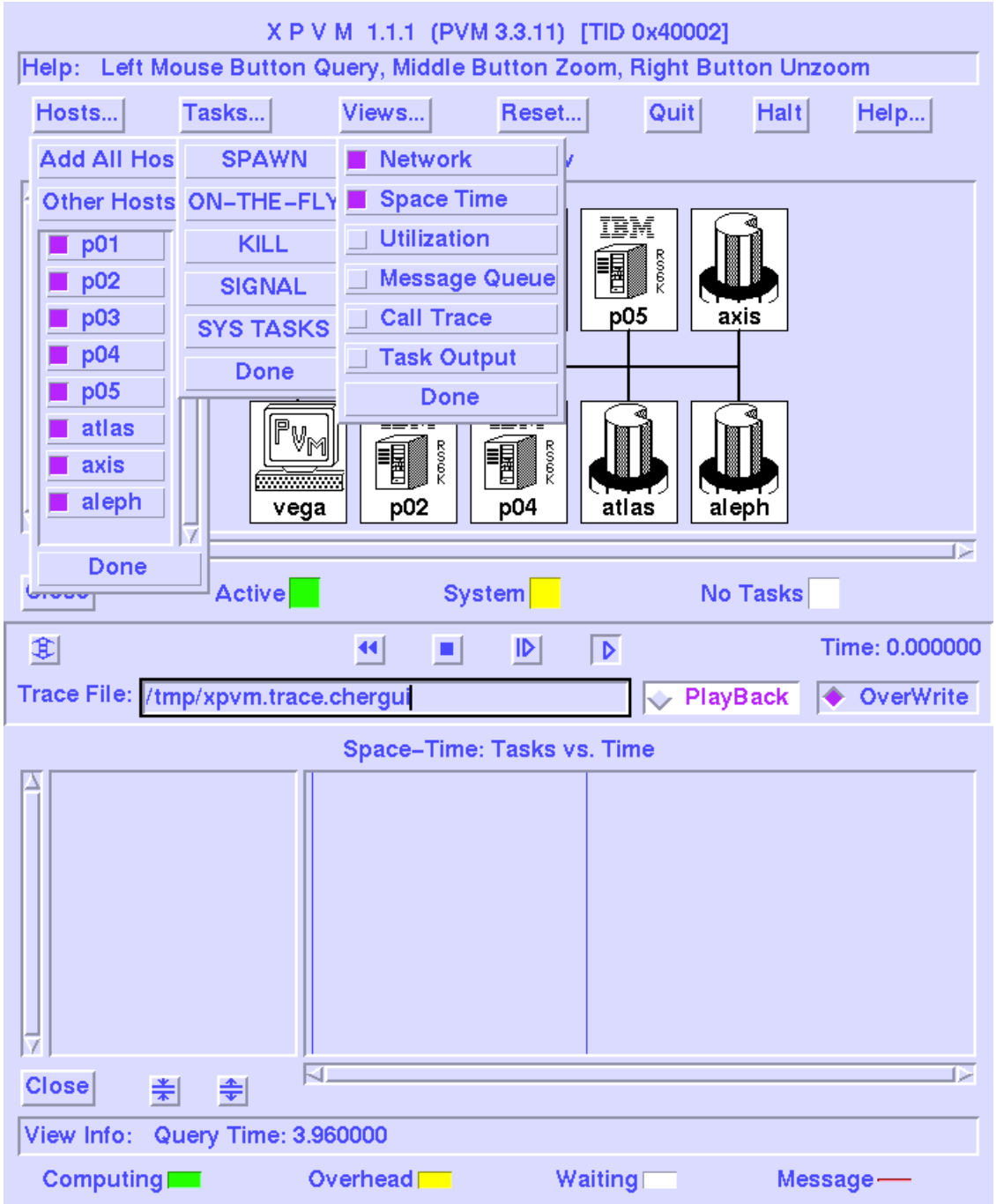

FIG. 20: Une brève description

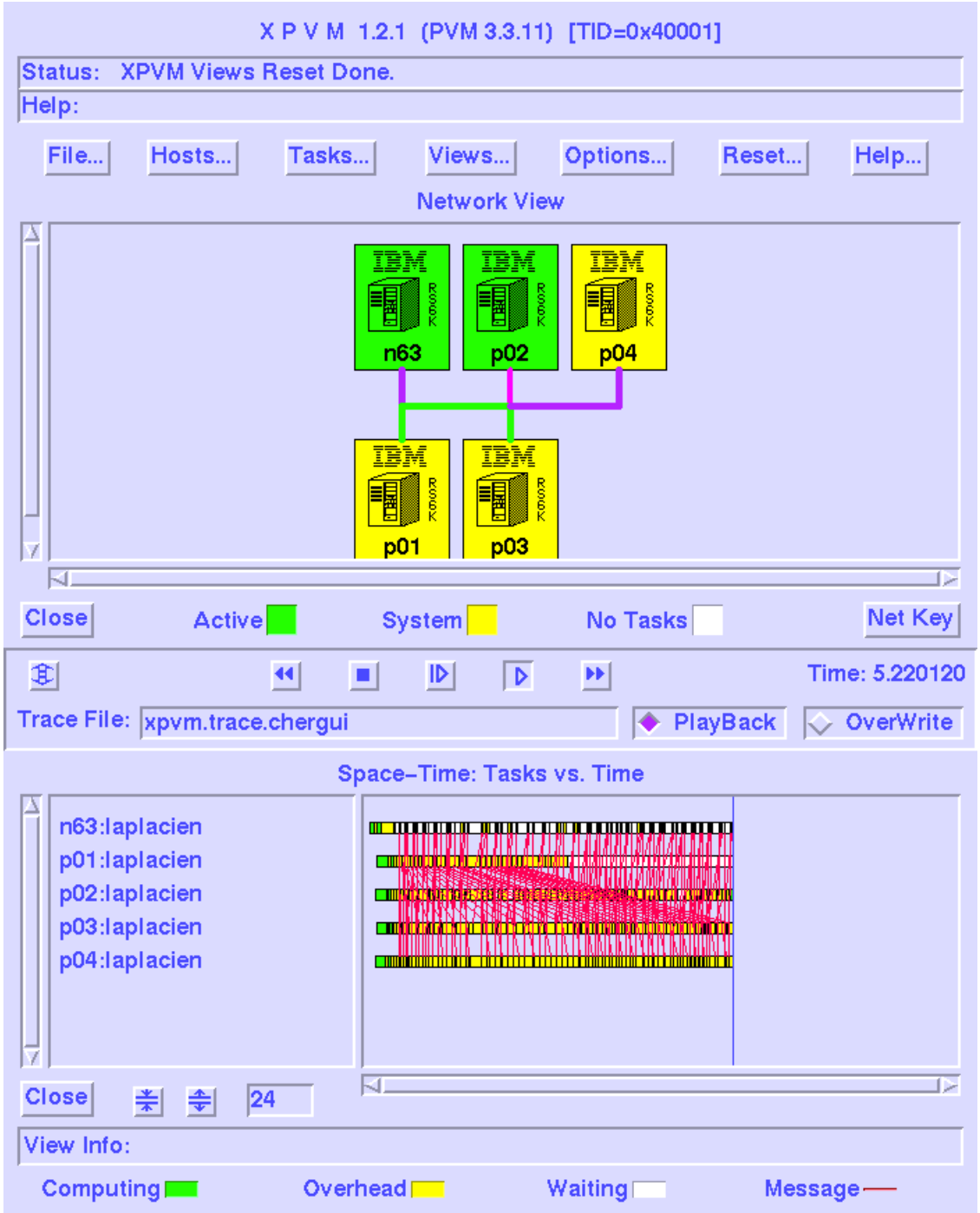

FIG. 21: Visualisation des performances

6.2 – Sur T3E

XPVM n'est pas implémenté sur le T3E. Il existe cependant d'autres outils qui permettent :

- ☞ de mesurer les performances ➪ apprentice
- ☞ de déboguer un programme ➪ totalview

TAB. 5: Utilisation d'apprentice et de totalview (en Fortran)

apprentice	totalview
❶ `f90 -eA -lapp source.f90`	❶ `f90 -G0 -X2 source.f90`
❷ `mpprun -n4 a.out`	❷ `totalview a.out`
❸ `apprentice app.rif`	

Un suppport détaillé d'utilisation d'`apprentice` se trouve à l'adresse :

http ://www.idris.fr/su/Parallele/utilitaires_aleph/outil/apprentice.html

De même pour `totalview` à l'adresse :

http ://www.idris.fr/su/Parallele/utilitaires_aleph/outil/totalview.html

6.3 – Sur VPP300

XPVM n'est pas implémenté sur le VPP300. Il existe cependant d'autres outils qui permettent :

- ☞ de mesurer les performances ➪ mpt
- ☞ de déboguer un programme ➪ totalview

Un suppport détaillé d'utilisation d'mpt se trouve à l'adresse :

http ://www.idris.fr/su/Hybride/utilitaires/
performance/mpt.html

De même pour totalview à l'adresse :

http ://www.idris.fr/su/Hybride/utilitaires/
outil/totalview.html/

Position dans le cours

7 – Conclusion

- PVM est un environnement de programmation et d'exécution.
- La bibliothèque PVM est un outil pour la programmation parallèle par échange de messages.
- Les applications PVM s'exécutent aussi bien sur des machines hétérogènes en grappe que sur des machines MPP autonomes comme le T3E ou le SP2 utilisées conjointement.
- Les machines peuvent disposer d'un système d'exploitation différent.
- Le réseau d'interconnexion entre les machines peut être du type ETHERNET, FDDI, HIPPI, ATM, etc.

7 – Conclusion

- La bibliothèque PVM contient des sous-programmes qui permettent essentiellement de gérer la machine virtuelle, les processus, les communications point à point et collectives, les *buffers* d'envoi et de réception de messages et les groupes de processus.
- Le modèle de programmation peut être du type SPMD ou, plus général encore, du type MPMD.
- L'arrivée de MPI a largement compromis son avenir.
- Il continue cependant à évoluer pour intégrer en particulier des fonctionnalités qui existent déjà dans MPI-1 ou sont prévues dans MPI-2.
- La conversion d'un code PVM en MPI est triviale, l'inverse n'étant pas vrai si l'on utilise des fonctions évoluées (types dérivés, topologies, etc.) de MPI.