PARALLEL PROGRAMMING WITH PVM
TRAVAUX PRATIQUES

Philippe Marquet
marquet@lifl.fr
Laboratoire d'informatique fondamentale de Lille
Université des sciences et technologies de Lille
France

Maîtrise d'informatique

Janvier 1998

Table des matières

- Forme quadratique d'un vecteur 1
- Produit scalaire de deux vecteurs 13
- Communication sur un anneau 18
- Tri systolique 27

FORME QUADRATIQUE D'UN VECTEUR

Forme quadratique d'un vecteur

- Soit un vecteur X détenu par une tâche maître
- Le but du programme est de calculer la forme quadratique de ce vecteur
- La forme quadratique est définie comme la somme des carrés des éléments de ce vecteur
- L'idée de base d'une implémentation possible est la suivante :
  - Le maître lance autant d'esclaves que désiré
  - Chacun des esclaves reçoit une partie du vecteur X
  - Chacun des esclaves combine cette partie de vecteur et retourne le résultat (somme des carrés des éléments) au maître
  - À partir des ces différents résultats partiels, le maître produit le résultat global

```
      PROGRAM PFQ
      IMPLICIT NONE
*
*     .. Procedures externes ..
      EXTERNAL SGENVECT        ! Generation aleatoire de vecteurs
*
*     .. Fonctions externes ..
      DOUBLE PRECISION SDOT
      EXTERNAL SDOT            ! Produit scalaire en  sequentiel
*
*     .. Parameters ..
      INTEGER MAXN             ! taille maximale du tableau X
      PARAMETER (MAXN=8000)
      INTEGER MAXPROC          ! nombre max d'esclaves
      PARAMETER (MAXPROC=64)
      INTEGER METAG            ! tag des messages maitre-> escalve
      PARAMETER (METAG=1)
      INTEGER EMTAG            ! tag des messages escalve-> maitre
      PARAMETER (EMTAG=2)
*
*     .. Definition de la bibliotheque PVM ..
      INCLUDE 'fpvm3.h'
```



```
*     .. Scalaires ..
      INTEGER I, J, K, NEXT
      INTEGER IERR               ! Numero d'erreur
      INTEGER MYTID              ! Mon TID
      INTEGER NPROCS             ! Nombre d'escalves
      INTEGER N                  ! Longueur de X
      INTEGER LN                 ! Longueur locale a traiter
      DOUBLE PRECISION LFQ       ! Le resultat local
      DOUBLE PRECISION GFQ       ! Le resultat global
      INTEGER IBUF               ! Identifiant de Buffer PVM
*
*     .. Tableaux ..
      INTEGER TIDS (0:MAXPROC)
      DOUBLE PRECISION X(MAXN)
*
*     S'enroller dans PVM et recuperer le TID du maitre
*
      CALL PVMFMYTID (MYTID)
      CALL PVMFPARENT (TIDS(0))
```



```
*     Si je suis le maitre, je lance les autres
      IF (TIDS(0) .EQ. PVMNOPARENT) THEN
*
*        Dialogue
         WRITE (*,*) 'Combien de processus esclaves ?'
         READ (*,*) NPROCS
         WRITE (*,*) 'Longueur du vecteur ?'
         READ (*,*) N
         IF (N .GT. MAXN) THEN
            WRITE (*,*) 'ERREUR, taille trop grande'
            WRITE (*,*) 'ARRET'
            CALL PVMFEXIT (IERR)
            STOP
         END IF
*
*        Combien d'elements a traiter par chacun
         LN = N / NPROCS       ! les autres elements sont traites par
                               ! le maitre
         NEXT = 1              ! l'index du prochain element a envoyer
*
*        Generation aleatoire du vecteur
         CALL SGENVECT (X, N)
```



```
*        Pour chacun des esclaves
         DO 10 I=1, NPROCS
*
*           Lance l'esclave et test
            CALL PVMFSPAWN ('pfq', 0, 'anywhere', 1, TIDS(I), IERR)
            IF (IERR .NE. 1) THEN
               WRITE (*,*) 'ERREUR au lancement de l\'escalve', I
               WRITE (*,*) 'ARRET'
               CALL PVMFEXIT (IERR)
               STOP
            END IF
*
*           Passe les infos a l'escalve :
*           - La taille a traiter par l'esclave
*           - X
            CALL PVMFINITSEND (PVMDEFAULT, IBUF)
            CALL PVMFPACK (INTEGER4, LN, 1, 1, IERR)
            CALL PVMFPACK (REAL8, X(NEXT), LN, 1, IERR)
            CALL PVMFSEND (TIDS(I), METAG, IERR)
            NEXT = NEXT + LN
 10      CONTINUE
```

```
*        Le maitre fait sa part
*        - traitement des indices NEXT a N
         GFQ = SDOT (X(NEXT), X(NEXT), N-NEXT+1)
*
*        Le maitre recupere les resultats des esclaves
         DO 30 I = 1, NPROCS
            CALL PVMFRECV (-1, EMTAG, IBUF)
            CALL PVMFUNPACK (REAL8, LFQ, 1, 1, IERR)
            GFQ = GFQ + LFQ
 30      CONTINUE
*
*        Verification du resultat
*        On calcul le produit scalaire en local
         LFQ = SDOT (X, X, N)
         WRITE (*,*) '<X,X> - <X,X>p =', LFQ - GFQ
```



```
*        Le code d'un esclave
      ELSE
*
*        Reception des donnnes depuis le maitre :
*        - taille, X
         CALL PVMFRECV (TIDS(0), METAG, IBUF)
         CALL PVMFUNPACK (INTEGER4, LN, 1, 1, IERR)
         CALL PVMFUNPACK (REAL8, X, LN, 1, IERR)
*
*        Calcul de la forme quadratique locale
         LFQ = SDOT (X, X, LN)
*
*        Retour du resultat au maitre
*        - LFQ
         CALL PVMFINITSEND (PVMDEFAULT, IBUF)
         CALL PVMFPACK (REAL8, LFQ, 1, 1, IERR)
         CALL PVMFSEND (TIDS(0), EMTAG, IERR)
      END IF
*
*     C'est fini : on quitte proprement
      CALL PVMFEXIT (IERR)
      END
```


- Fichier `pfq.f`
  Compilation par `make pfq`

```
farm13% make init
mkdir -p ALPHA
mkdir -p /home/west/marquet/pvm3/bin/ALPHA
initialisations faites
(null command)

farm13% make pfq
f77 -I/home/west/marquet/pvm3/include  -c pfq.f
mv `basename ALPHA/pfq.o` ALPHA/pfq.o
f77 -I/home/west/marquet/pvm3/include  -c sdot.f
mv `basename ALPHA/sdot.o` ALPHA/sdot.o
f77  -o `basename /home/west/marquet/pvm3/bin/ALPHA/pfq` ALPHA/pfq.o ALPHA/sdot.
o /home/west/marquet/pvm3/lib/ALPHA/libfpvm3.a /home/west/marquet/pvm3/lib/ALPHA
/libpvm3.a
mv `basename /home/west/marquet/pvm3/bin/ALPHA/pfq` /home/west/marquet/pvm3/bin/
ALPHA/pfq
pfq done
```


- Fichier `Makefile`

```
# Makefile                                                              -*-Text-*-
################################################################################
#                                        Philippe Marquet --- <marquet@lifl.fr>
#                                        05 Dec 1994

# Il est necessaire d'avoir positionner les variables
#    PVM_ROOT : la racine de pvm3
#    PVM_ARCH : l'architecture pour laquelle compiler

IncludePath=${PVM_ROOT}/include
PvmLib=${PVM_ROOT}/lib/${PVM_ARCH}/libpvm3.a
PvmFLib=${PVM_ROOT}/lib/${PVM_ARCH}/libfpvm3.a
PvmGLib=${PVM_ROOT}/lib/${PVM_ARCH}/libgpvm3.a
BinDir=${HOME}/pvm3/bin/${PVM_ARCH}
ObjDir=${PVM_ARCH}

FFLAGS=-I${IncludePath}
# Ne pas changer l'ordre des librairies dans la definition suivante
LDFLAGS=
LDLIBS=${PvmFLib} ${PvmLib}
LINK=f77
```

```
#       .. pfq ..
pfq : ${BinDir}/pfq
        @echo "$@ done"
${ObjDir}/pfq.o : pfq.f
        ${FC} ${FFLAGS} -c $?
        mv `basename $@` $@
${BinDir}/pfq : ${ObjDir}/pfq.o ${ObjDir}/sdot.o
        ${LINK} ${LDFLAGS} -o `basename $@` ${ObjDir}/pfq.o ${ObjDir}/sdot.o ${LDLIBS}
        mv `basename $@` $@

#       .. sdot ..
${ObjDir}/sdot.o : sdot.f
        ${FC} ${FFLAGS} -c $?
        mv `basename $@` $@

#       .. initialisation ..
init :  ${ObjDir} ${BinDir}
        @echo "initialisations faites"
${ObjDir} :
        mkdir -p $@
${BinDir} :
        mkdir -p $@
```


- Travaux pratiques :
  - Compiler le programme
  - Lancer pvm sur une PVM composée de deux machines
  - Exécuter différents tests de `pfq`
  - Ajouter une nouvelle machine dans la PVM
  - Nouvelles exécutions

PRODUIT SCALAIRE DE DEUX VECTEURS

Produit scalaire de deux vecteurs

- À partir du programme de calcul de la forme quadratique d'un vecteur X

$$\sum_{i=1,N} X(i)^2$$

- Calcul du produit scalaire de deux vecteurs X et Y :

$$\sum_{i=1,N} X(i) \times Y(i)$$

- Programme `psdot.f` :

```
farm13% cp pfq.f psdot.f
farm13% emacs psdot.f &
...
farm13% make psdot
```

Décomposition dichotomique en PVM

- Nouvelle version du calcul du produit scalaire de deux vecteurs X et Y
- L'idée est de choisir une taille L telle que
  - si X et Y sont de longueur inférieure à L, on effectue le traitement local
  - sinon, on partage le travail entre deux tâches PVM identiques

  On itère ensuite ce processus jusqu'à effectuer tous les traitements localement
- Programme `pdic.f`

Bag of tasks en PVM

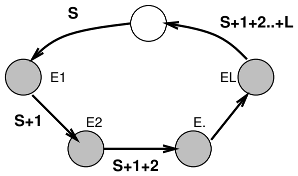
- Troisième version du produit scalaire de deux vecteurs X et Y
- On choisit un nombre N de tâches PVM pour effectuer ce travail (par exemple, le nombre de machines constituant la PVM)
- Comme précédemment, on choisit une taille L à partir de laquelle on décide de partager le travail
- Un maitre lance N esclaves puis répond aux requêtes de ces N esclaves
- Réponses (message tag) :
  - `work` suivit de L valeurs de X et de L valeurs de Y
  - `end` : le travail est terminé
- Un esclave itère la boucle suivante jusqu'à recevoir un ordre `end` :
  - Émission d'une requête au maître
  - Réception d'un ordre
  - Si l'ordre est `work`, lecture des données, production du résultat et retour du résultat au maître

- Le maître a terminé lorsqu'il a envoyé N ordres `end`
- Programmes
  - `botm.f` : le maître
  - `bote.f` : les esclaves

COMMUNICATION SUR UN ANNEAU

Communication sur un anneau

- Exemple illustratif de communication en anneau
- Soit un maître et L esclaves connectés en anneau



- Le maître émet une valeur S vers le premier esclave
- Chaque esclave (numéro i) reçoit une valeur V depuis son voisin de gauche et transmet la valeur $V + i$ à son voisin de droite
- Le maître collecte la valeur émise par le dernier esclave

Code du maître (fichier `anneau.f`)

```
*
      PROGRAM ANNEAU
*
      IMPLICIT NONE
*
*     ..Definition de la bibliotheque PVM..
      INCLUDE 'fpvm3.h'
*
*     ..Definitions communes avec les esclaves..
      INCLUDE 'anneau.h'
*
*     ..Variables..
      INTEGER I, J, K
      INTEGER IERR                ! Erreur PVM
      INTEGER TIDS (0:NPROCS)     ! TIDS des esclaves
                                  ! TIDS (0) = TID du maitre
      INTEGER S                   ! La valeur emise au premier esclave
      INTEGER ATTENDUE            ! La valeur attendue depuis le dernier esclave
```


Fichier `anneau.h`

```
*    Usage: definitions communes a anneau.f et anne.f
*
      INTEGER EMTAG             ! tag des messages maitre-> esclaves
      INTEGER METAG             ! tag des messages esclaves-> maitre
      INTEGER EDTAG             ! tag des messages esclaves-> esclave droit
      PARAMETER (EMTAG=0, METAG=1, EDTAG=3)
*
      INTEGER NPROCS            ! Nombre d'esclaves
      PARAMETER (NPROCS=4)
*
```


Code du maître (suite)

```
*
*     ..On entre dans PVM et on lance les esclaves..
      CALL PVMFMYTID (TIDS (0))
      CALL PVMFCATCHOUT (1, IERR)
      CALL PVMFSPAWN ('anne', 0, '*', NPROCS, TIDS(1), IERR)
*
*     ..Distribution du tableau TIDS aux esclaves..
      CALL PVMFINITSEND (PVMDATADEFAULT, IERR)
      CALL PVMFPACK (INTEGER4, TIDS (0), NPROCS+1, 1, IERR)
      CALL PVMFMCAST (NPROCS, TIDS(1), METAG, IERR)
*
*     ..On emet une valeur au premier esclave
      CALL PVMFINITSEND (PVMDEFAULT, IERR)
      CALL PVMFPACK (INTEGER2, S, 1, 1, IERR)
      CALL PVMFSEND (TIDS (1), EDTAG, IERR)
```

```
*
*     ..Calcul de la valeur qui devrait etre recue
      ATTENDUE = S + (((NPROCS + 1) * NPROCS)/2)
*
*     ..On attend une valeur depuis le dernier escalve
      CALL PVMFRECV (TIDS (NPROCS), EDTAG, IERR)
      CALL PVMFUNPACK (INTEGER2, I, 1, 1, IERR)
*
*     ..Quitte PVM..
      CALL PVMFEXIT (IERR)
*
*     .. OK ?..
      IF (I .EQ. ATTENDUE) THEN
         WRITE (*,*) 'Ok, j ai recu la bonne valeur'
      ELSE
         WRITE (*,*) 'Pb, j attendais', ATTENDUE, ' recu', I
      END IF
      END
```


Code de l'esclave (fichier anne.f)

```
*
      PROGRAM ANNE
*
      IMPLICIT NONE
*
*     ..Definition de la bibliotheque PVM..
      INCLUDE 'fpvm3.h'
*
*     ..Definitions communes avec le maitre..
      INCLUDE 'anneau.h'
*
*     ..Variables..
      INTEGER I, J, K
      INTEGER IERR                ! Erreur PVM
      INTEGER TIDS (0:NPROCS)     ! TIDS des esclaves
                                  ! TIDS (0) = TID du maitre
      INTEGER ME                  ! Mon numero d'esclave (1 a NPROCS)
      INTEGER MYTID               ! Mon TID
      INTEGER NONEXT              ! L'esclave suivant ou le maitre
      INTEGER NOPREV              ! L'esclave precedent ou le maitre
      INTEGER RECU                ! La valeur recue
      INTEGER EMIS                ! La valeur emise
```



```
*
*     ..Qui suis-je..
      CALL PVMFMYTID (MYTID)
*
*     ..Recoit TIDS depuis le maitre..
      CALL PVMFRECV (-1, -1, IERR)
      CALL PVMFUNPACK (INTEGER4, TIDS (0), NPROCS+1, 1, IERR)
*
*     ..Recherche de ME..
      ME = 0
      DO WHILE (MYTID .NE. TIDS (ME) .and. ME .LE. NPROCS)
         ME = ME + 1
      END DO
      IF (ME .GT. NPROCS) THEN
         WRITE (*,*) 'Erreur fatale, je ne suis pas dans TIDS'
         STOP
      END IF
      NONEXT = ME + 1
      IF (ME .EQ. NPROCS) NONEXT = 0     ! le maitre
      NOPREV = ME - 1
      IF (ME .EQ. 1)      NOPREV = 0     ! le maitre
```



```
*
*     ..Attente d'une valeur depuis le precedent..
      CALL PVMFRECV (TIDS(NOPREV), EDTAG, IERR)
      CALL PVMFUNPACK (INTEGER2, RECU, 1, 1, IERR)
*
*     ..Emission au suivant..
      EMIS = RECU + ME
      CALL PVMFINITSEND (PVMDEFAULT, IERR)
      CALL PVMFPACK (INTEGER2, EMIS, 1, 1, IERR)
      CALL PVMFSEND (TIDS(NONEXT), EDTAG, IERR)
*
*     ..C'est fini..
      CALL PVMFEXIT (IERR)
      END
```

TRI SYSTOLIQUE

Tri systolique

- Soit une liste de $N \times L$ valeurs à trier
- Un maître et L esclaves connectés en anneau
- Le maître initie le tri en émetant les valeurs au premier esclave
- Un esclave reçoit N premières valeurs qu'il trie
- Soit max la plus grande valeur alors détenue
- Lors de la réception d'une nouvelle valeur v
  - si $v > max$, cette valeur est transmise à l'esclave suivant
  - sinon, la valeur max est transmise à l'esclave suivant
    la valeur v est insérée dans les valeurs locales
- Le tri est terminée lorsque l'esclave i (i de 1 à L) a reçu $(L - i + 1) \times N$ valeurs

Implémentation sous PVM

- Mise en place
  - Le maître initie L esclaves
  - Il leur transmet les TIDS des autres esclaves
  - Chaque esclave retrouve son numéro et identifie son voisin de droite
- Le maître émet la liste des valeurs au premier esclave
- Chaque esclave attend N premières valeurs
- Il trie ces N valeurs
- Pour chaque esclave, tant qu'il n'a pas fini de trier
  - Recevoir une valeur de l'escalve précédent
  - Suivant sa valeur ($<$ ou $>$ à max)
    - ◇ Transmettre cette valeur ou max à l'esclave suivant
    - ◇ Insérer cette valeur si nécessaire
    - ◇ Mettre à jour max
- Retour des valeurs triées au maître

Duplication des esclaves

- Modification de l'implémentation précédente
- Chaque esclave est associé à un esclave trieur