ENSERB 3ème Année Filière Informatique **Année 2000-2001**
D.E.A Informatique

Expression du Parallélisme

Durée 2h
Documents de cours autorisés

Exercice A

On s'intéresse au calcul des $N(N-1)$ interactions entre N données, X_1, X_2, ..., X_N et au calcul des N cumuls :

$$F_i = \sum_{j=1, j \neq i}^{N} Interaction(X_i, X_j)$$

où $Interaction(X_i, X_j)$, $i \neq j$, est l'interaction de X_i avec X_j.

Les interactions sont supposées symétriques :

$$\forall i, j, i \neq j, Interaction(X_i, X_j) = Interaction(X_j, X_i)$$

et il y a donc seulement $N(N-1)/2$ interactions à calculer.

I - Code HPF

Une formulation HPF de ce problème est donnée dans le programme *interactions.hpf* fourni en annexe.

Pour illustrer simplement ce problème, les données sont dans un tableau d'entiers, X, initialisé avec les valeurs $1, 2, ..., n$ et l'interaction entre X_i et X_j est le produit $X_i * X_j$.

Dans ce code, la taille n du tableau X est supposée impaire.

1) Donner les valeurs successives prises par les tableaux X, CX, CA et F dans le cas où n vaut 5. Préciser le rôle de chaque tableau.

2) Décrire le placement de chaque tableau sur les processeurs (par exemple, pour $n = 10001$ et 5 processeurs).

3) Expliquer le code SPMD généré par le compilateur *Adaptor* (voir le programme *cube.f*). Notamment :

- Expliquer le code généré pour chaque instruction "data-parallèle".
- Décrire les communications générées (schéma de communication, nombre de communications, données communiquées). Illustrer avec $n = 10001$ et 5 processeurs.
- Décrire les zones de recouvrement utilisées et expliquer leur utilisation et leur intérêt.
- Commenter (critiquer) le code SPMD obtenu.

4) Compléter le code HPF fourni afin de vérifier que le résultat calculé dans F est correct.

Décrire le code SPMD généré par le compilateur *Adaptor* pour ces lignes de code supplémentaires : comment les calculs sont-ils distribués sur les processeurs et quelles communications sont nécessaires ?

II - Code MPI

1) Ecrire un programme MPI permettant de résoudre ce même problème et de telle sorte que chaque interaction symétrique ne soit calculée qu'une seule fois (soit par le processus qui possède X_i, soit par celui qui possède X_j, dans le cas où X_i et X_j ne sont pas sur le même processeur).

Si besoin, pour simplifier l'écriture, on pourra supposer que le nombre de processus est impair et que la taille du tableau est un multiple du nombre de processus.

2) Expliquer votre implémentation : algorithme, données locales, calculs locaux, communications (préciser le schéma de communication, le nombre de communications, les données communiquées).

Comparer ce code avec celui généré par *Adaptor* notamment du point de vue des communications.

3) Décrire les modifications à apporter à votre programme pour qu'il fonctionne avec un nombre pair de processus.

III - Code OpenMP

1) Modifier le programme Fortran *interactions.f* fourni en annexe en y insérant des directives OpenMP afin de permettre une parallélisation du code obtenu sur machine parallèle à mémoire partagée.

2) Expliquer vos choix (portée des variables, directives utilisées, ...) et commenter le comportement de votre programme (rôle de chaque tâche, équilibrage de charge entre tâches, ...).

Exercice B

Ecrire un programme Fortran (ou éventuellement C) intégrant des directives OpenMP et qui calcule une approximation de Π :

$$\Pi = \int_0^1 \frac{4}{1+x^2} dx$$

$$\Pi \approx \frac{1}{N} \times \sum_{i=1}^{N} \frac{4}{1+x_i^2}, x_i = \frac{i-0.5}{N}$$

N étant le nombre de sous-intervalles de l'intervalle [0,1].

Annexes

Programme *interactions.hpf*

```
        program interactions
        integer i, n
        integer, allocatable :: X(:), F(:), CX(:), CA(:)

!hpf$   PROCESSORS pr(NUMBER_OF_PROCESSORS())
!hpf$   ALIGN F(:)  WITH X(:)
!hpf$   ALIGN CX(:) WITH X(:)
!hpf$   ALIGN CA(:) WITH X(:)
!hpf$   DISTRIBUTE X(BLOCK) ONTO pr

        print *, 'Taille du tableau : '
        read *, n
        allocate (X(n), F(n), CX(n), CA(n))

        FORALL (i=1:n) X(i) = i
        F = 0
        CX = X
        CA = 0

        DO I = 1, n/2
          CX = CSHIFT(CX,1,1)
          CA = CSHIFT(CA,1,1)
          F = F + X * CX
          CA = CA + X * CX
        ENDDO

        F = F + CSHIFT(CA, -n/2, 1)

        deallocate (CA, CX, F, X)
        END
```

Code SPMP généré par *Adaptor* pour le programme *interactions.hpf cube.f*

```
      PROGRAM INTERACTIONS
      INTEGER*4 I
      INTEGER*4 N
      INTEGER*4 PR_TOPID
      INTEGER*4 X_DSP
      INTEGER*4 X (1:2)
      INTEGER*4 X_DIM1
      INTEGER*4 X_ZERO
      INTEGER*4 CA_DSP
      INTEGER*4 CA (1:2)
      INTEGER*4 CA_DIM1
      INTEGER*4 CA_ZERO
      INTEGER*4 CX_DSP
      INTEGER*4 CX (1:2)
      INTEGER*4 CX_DIM1
      INTEGER*4 CX_ZERO
      INTEGER*4 F_DSP
      INTEGER*4 F (1:2)
      INTEGER*4 F_DIM1
      INTEGER*4 F_ZERO
      INTEGER*4 I16
      INTEGER*4 I15
      INTEGER*4 I14
      INTEGER*4 I13
      INTEGER*4 I12
      INTEGER*4 I11
      INTEGER*4 I10
      INTEGER*4 I9
      INTEGER*4 I8
      INTEGER*4 I7
      INTEGER*4 TMP5_DSP
      INTEGER*4 TMP5 (1:2)
      INTEGER*4 TMP5_DIM1
      INTEGER*4 TMP5_ZERO
      INTEGER*4 TMP3_DSP
      INTEGER*4 TMP3 (1:2)
      INTEGER*4 TMP3_DIM1
      INTEGER*4 TMP3_ZERO
      INTEGER*4 TMP1_DSP
      INTEGER*4 TMP1 (1:2)
      INTEGER*4 TMP1_DIM1
      INTEGER*4 TMP1_ZERO
      INTEGER*4 ISEC_DSP1
```

```
INTEGER*4 X_STOP1
INTEGER*4 X_START1
EXTERNAL dalib_pid
INTEGER*4 dalib_pid
INTEGER*4 dalib_0
COMMON /dalib_data0/ dalib_0
EXTERNAL dalib_nproc
INTEGER*4 dalib_nproc
call dalib_init (4,4,4)
call dalib_set_present (dalib_0)
call dalib_start_subroutine ('INTERACTIONS',12)
call dalib_top_create (PR_TOPID,1,1,dalib_nproc())
IF (dalib_pid() .eq. 1) THEN
   PRINT *,'Taille du tableau : '
   READ *, N
END IF
call dalib_broadcast (N,4,1)
call dalib_array_make_dsp (X_DSP,1,4)
call dalib_distribute (X_DSP,PR_TOPID,1,0)
call dalib_array_define (X_DSP,1,N)
call dalib_array_allocate (X_DSP,X,X_ZERO,X_DIM1)
call dalib_array_make_dsp (F_DSP,1,4)
call dalib_align_source (F_DSP,X_DSP,4,1,0,1)
call dalib_align_target (F_DSP,X_DSP,8,1)
call dalib_array_define (F_DSP,1,N)
call dalib_array_allocate (F_DSP,F,F_ZERO,F_DIM1)
call dalib_array_make_dsp (CX_DSP,1,4)
call dalib_align_source (CX_DSP,X_DSP,4,1,0,1)
call dalib_align_target (CX_DSP,X_DSP,8,1)
call dalib_array_define (CX_DSP,1,N)
call dalib_array_overlap (CX_DSP,0,1)
call dalib_array_allocate (CX_DSP,CX,CX_ZERO,CX_DIM1)
call dalib_array_make_dsp (CA_DSP,1,4)
call dalib_align_source (CA_DSP,X_DSP,4,1,0,1)
call dalib_align_target (CA_DSP,X_DSP,8,1)
call dalib_array_define (CA_DSP,1,N)
call dalib_array_overlap (CA_DSP,0,1)
call dalib_array_allocate (CA_DSP,CA,CA_ZERO,CA_DIM1)
call dalib_array_lslice (X_DSP,1,1,N,X_START1,X_STOP1)
DO I=X_START1,X_STOP1
   X(X_ZERO+I) = I
   F(F_ZERO+I) = 0
   CX(CX_ZERO+I) = X(X_ZERO+I)
   CA(CA_ZERO+I) = 0
```

```
 END DO
 DO I=1,N/2
    call dalib_array_make_dsp (TMP1_DSP,1,4)
    call dalib_align_source (TMP1_DSP,X_DSP,4,1,0,1)
    call dalib_align_target (TMP1_DSP,X_DSP,8,1)
    call dalib_array_define (TMP1_DSP,1,N)
    call dalib_array_allocate (TMP1_DSP,TMP1,TMP1_ZERO,TMP1_DIM1)
    call dalib_section_create (ISEC_DSP1,CX_DSP,1,0,N,1)
    call dalib_overlap_update (ISEC_DSP1,0,1)
    call dalib_section_free (ISEC_DSP1)
    call dalib_array_lslice (X_DSP,1,1,N,X_START1,X_STOP1)
    DO I10=X_START1,X_STOP1
       TMP1(TMP1_ZERO+I10) = CX(CX_ZERO+(I10+1))
    END DO
    call dalib_array_lslice (X_DSP,1,1,N,X_START1,X_STOP1)
    DO I11=X_START1,X_STOP1
       CX(CX_ZERO+I11) = TMP1(TMP1_ZERO+I11)
    END DO
    call dalib_array_free (TMP1_DSP)
    call dalib_array_make_dsp (TMP3_DSP,1,4)
    call dalib_align_source (TMP3_DSP,X_DSP,4,1,0,1)
    call dalib_align_target (TMP3_DSP,X_DSP,8,1)
    call dalib_array_define (TMP3_DSP,1,N)
    call dalib_array_allocate (TMP3_DSP,TMP3,TMP3_ZERO,TMP3_DIM1)
    call dalib_section_create (ISEC_DSP1,CA_DSP,1,0,N,1)
    call dalib_overlap_update (ISEC_DSP1,0,1)
    call dalib_section_free (ISEC_DSP1)
    call dalib_array_lslice (X_DSP,1,1,N,X_START1,X_STOP1)
    DO I12=X_START1,X_STOP1
       TMP3(TMP3_ZERO+I12) = CA(CA_ZERO+(I12+1))
    END DO
    call dalib_array_lslice (X_DSP,1,1,N,X_START1,X_STOP1)
    DO I13=X_START1,X_STOP1
       CA(CA_ZERO+I13) = TMP3(TMP3_ZERO+I13)
    END DO
    call dalib_array_free (TMP3_DSP)
    call dalib_array_lslice (X_DSP,1,1,N,X_START1,X_STOP1)
    DO I14=X_START1,X_STOP1
       F(F_ZERO+I14) = F(F_ZERO+I14)+X(X_ZERO+I14)*CX(CX_ZERO+I14)
       CA(CA_ZERO+I14) = CA(CA_ZERO+I14)+X(X_ZERO+I14)*CX(CX_ZERO+
&I14)
    END DO
 END DO
 call dalib_array_make_dsp (TMP5_DSP,1,4)
```

```
 call dalib_align_source (TMP5_DSP,X_DSP,4,1,0,1)
 call dalib_align_target (TMP5_DSP,X_DSP,8,1)
 call dalib_array_define (TMP5_DSP,1,N)
 call dalib_array_allocate (TMP5_DSP,TMP5,TMP5_ZERO,TMP5_DIM1)
 call dalib_CSHIFT (TMP5(TMP5_ZERO+1),
$  CA(CA_ZERO+1),-N/2,1,TMP5_DSP,CA_DSP,dalib_0,dalib_0)
 call dalib_array_lslice (X_DSP,1,1,N,X_START1,X_STOP1)
 DO I16=X_START1,X_STOP1
    F(F_ZERO+I16) = F(F_ZERO+I16)+TMP5(TMP5_ZERO+I16)
 END DO
 call dalib_array_free (TMP5_DSP)
 call dalib_array_free (CA_DSP)
 call dalib_array_free (CX_DSP)
 call dalib_array_free (F_DSP)
 call dalib_array_free (X_DSP)
 call dalib_end_subroutine ()
 call dalib_exit ()
 END PROGRAM INTERACTIONS
```

Programme *interactions.f*

```
program interactions
integer i, k, r, n
integer, allocatable :: X(:), F(:), CA(:)

print *, 'Taille du tableau : '
read *, n

allocate (X(n), F(n), CA(n))


do i=1,n
   X(i)  = i
   F(i)  = 0
   CA(i) = 0
enddo


do i=1,n-1
   do k=i+1,n
      r = x(i)*x(k)
      F(i)  = F(i) + r
      CA(k) = CA(k) + r
   enddo
enddo


do i=1,n
   F(i)  = F(i) + CA(i)
enddo

end
```