03

OpenMP

Inria

C'est quoi ?

C'est quoi ?

Interface de programmation pour architecture à mémoire partagée

C'est quoi ?

Interface de programmation pour architecture à mémoire partagée

Principe

C'est quoi ?

Interface de programmation pour architecture à mémoire partagée

Principe

On décompose un programme en **Tâches** (ou Tasks)

C'est quoi ?

Interface de programmation pour architecture à mémoire partagée

Principe

On décompose un programme en **Tâches** (ou Tasks)

Ces tâches sont alors exécutées en parallèle par des **Threads**

C'est quoi ?

Interface de programmation pour architecture à mémoire partagée

Principe

On décompose un programme en **Tâches** (ou Tasks)

Ces tâches sont alors exécutées en parallèle par des **Threads**

Statut des variables

C'est quoi ?

Interface de programmation pour architecture à mémoire partagée

Principe

On décompose un programme en **Tâches** (ou Tasks)

Ces tâches sont alors exécutées en parallèle par des **Threads**

Statut des variables

Public (par défaut) : toutes les tâches ont accès à cette variable

C'est quoi ?

Interface de programmation pour architecture à mémoire partagée

Principe

On décompose un programme en **Tâches** (ou Tasks)

Ces tâches sont alors exécutées en parallèle par des **Threads**

Statut des variables

Public (par défaut) : toutes les tâches ont accès à cette variable

Privée : chaque tâche à sa propre version de la variable

C'est quoi ?

Interface de programmation pour architecture à mémoire partagée

Principe

On décompose un programme en **Tâches** (ou Tasks)

Ces tâches sont alors exécutées en parallèle par des **Threads**

Statut des variables

Public (par défaut) : toutes les tâches ont accès à cette variable

Privée : chaque tâche à sa propre version de la variable

Si une variable privée à le même nom qu'un variable en-dehors de tâche, les deux variables sont indépendantes.

Code (hello.F90)

```
program main
  implicit none
!$OMP PARALLEL
  print*, "hello "
!$OMP END PARALLEL
end program main
```

Compilation

```
gfortran -o exec.out hello.F90
```

Sortie

```
hello
```

Code (hello.F90)

```
program main
  implicit none
!$OMP PARALLEL
  print*, "hello "
!$OMP END PARALLEL
end program main
```

Compilation

```
gfortran -fopenmp -o exec.out hello.F90
```

Sortie

```
hello
hello
hello
hello
....
```

Code (hello.F90)

```
program main
  implicit none
!$OMP PARALLEL NUM_THREADS(2)
  print*, "hello "
!$OMP END PARALLEL
end program main
```

Compilation

```
gfortran -fopenmp -o exec.out hello.F90
```

Sortie

```
hello
hello
```

Code (hello.F90)

```
program main
  implicit none
!$OMP PARALLEL
  print*, "hello "
!$OMP END PARALLEL
end program main
```

Compilation

```
export OMP_NUM_THREADS=2
gfortran -fopenmp -o exec.out hello.F90
```

Sortie

```
hello
hello
```

Code (hello.F90)

```
program main
  !$ use OMP_LIB
  implicit none
  integer :: rang
!$OMP PARALLEL
  rang = OMP_GET_THREAD_NUM()
  print*, "hello, mon rang est ", rang
!$OMP END PARALLEL
end program main
```

Sortie (avec OMP_NUM_THREADS=4)

```
 hello, mon rang est           3
 hello, mon rang est           2
 hello, mon rang est           2
 hello, mon rang est           3
```

Code (hello.F90)

```
program main
  !$ use OMP_LIB
  implicit none
  integer :: rang
!$OMP PARALLEL PRIVATE(rang)
  rang = OMP_GET_THREAD_NUM()
  print*, "hello, mon rang est ", rang
!$OMP END PARALLEL
end program main
```

Sortie (avec OMP_NUM_THREADS=4)

```
 hello, mon rang est           3
 hello, mon rang est           2
 hello, mon rang est           0
 hello, mon rang est           1
```

Code (hello.F90)

```
program main
  !$ use OMP_LIB
  implicit none
  integer :: rang
  rang=12
!$OMP PARALLEL PRIVATE(rang)
  print*, "en entrée de la tâche, mon rang est ", rang
  rang = OMP_GET_THREAD_NUM()
  print*, "maintenant, mon rang est ", rang
!$OMP END PARALLEL
  print*, "enfin, mon rang est ", rang
end program main
```

Sortie (avec OMP_NUM_THREADS=4)

```
en entrée de la tâche, mon rang est      32706
maintenant, mon rang est                     0
en entrée de la tâche, mon rang est          0
maintenant, mon rang est                     2
en entrée de la tâche, mon rang est          0
maintenant, mon rang est                     3
en entrée de la tâche, mon rang est          0
maintenant, mon rang est                     1
enfin, mon rang est                         12
```

Code (hello.F90)

```
program main
!$ use OMP_LIB
  implicit none
  integer :: rang
  rang=12
!$OMP PARALLEL FIRSTPRIVATE(rang)
  print*, "en entrée de la tâche, mon rang est ", rang
  rang = OMP_GET_THREAD_NUM()
  print*, "maintenant, mon rang est ", rang
!$OMP END PARALLEL
  print*, "enfin, mon rang est ", rang
end program main
```

Sortie (avec OMP_NUM_THREADS=4)

```
en entrée de la tâche, mon rang est          12
maintenant, mon rang est                      0
en entrée de la tâche, mon rang est                  0
maintenant, mon rang est                      2
en entrée de la tâche, mon rang est                  0
maintenant, mon rang est                      3
en entrée de la tâche, mon rang est                  0
maintenant, mon rang est                      1
enfin, mon rang est                  12
```

Utilisation des variables privées (5/5)

PRIVATE

PRIVATE

La variable privée est non initialisée au début de la tâche, même si elle existe en dehors la tâche

PRIVATE

La variable privée est non initialisée au début de la tâche, même si elle existe en dehors la tâche

En sortie de la tâche, la variable privée est perdue

PRIVATE

La variable privée est non initialisée au début de la tâche, même si elle existe en dehors la tâche

En sortie de la tâche, la variable privée est perdue

FIRSTPRIVATE

PRIVATE

La variable privée est non initialisée au début de la tâche, même si elle existe en dehors la tâche

En sortie de la tâche, la variable privée est perdue

FIRSTPRIVATE

La variable privée est initialisée au début de la tâche à la valeur actuelle de la variable en dehors de la tâche

PRIVATE

La variable privée est non initialisée au début de la tâche, même si elle existe en dehors la tâche

En sortie de la tâche, la variable privée est perdue

FIRSTPRIVATE

La variable privée est initialisée au début de la tâche à la valeur actuelle de la variable en dehors de la tâche

PRIVATE

La variable privée est non initialisée au début de la tâche, même si elle existe en dehors la tâche

En sortie de la tâche, la variable privée est perdue

FIRSTPRIVATE

La variable privée est initialisée au début de la tâche à la valeur actuelle de la variable en dehors de la tâche

LASTPRIVATE

PRIVATE

La variable privée est non initialisée au début de la tâche, même si elle existe en dehors la tâche

En sortie de la tâche, la variable privée est perdue

FIRSTPRIVATE

La variable privée est initialisée au début de la tâche à la valeur actuelle de la variable en dehors de la tâche

LASTPRIVATE

Définie uniquement pour les boucles

PRIVATE

La variable privée est non initialisée au début de la tâche, même si elle existe en dehors la tâche

En sortie de la tâche, la variable privée est perdue

FIRSTPRIVATE

La variable privée est initialisée au début de la tâche à la valeur actuelle de la variable en dehors de la tâche

LASTPRIVATE

Définie uniquement pour les boucles

Assigne la valeur de la variable à la dernière itération de la boucle (voir la partie "boucle" plus loin)

Clause DEFAULT

DEFAULT (PRIVATE)

DEFAULT (PRIVATE)

Définit par défaut toutes les variables comme privées.

DEFAULT (PRIVATE)

Définit par défaut toutes les variables comme privées.

Exemple : !$OMP PARALLEL DEFAULT(PRIVATE) FIRSTPRIVATE(rang) SHARED(I)

DEFAULT (PRIVATE)

Définit par défaut toutes les variables comme privées.

Exemple : !$OMP PARALLEL DEFAULT(PRIVATE) FIRSTPRIVATE(rang) SHARED(I)

On peut utiliser DEFAULT (LASTPRIVATE) ou DEFAULT (SHAREd) de la même manière

DEFAULT (PRIVATE)

Définit par défaut toutes les variables comme privées.

Exemple : !$OMP PARALLEL DEFAULT(PRIVATE) FIRSTPRIVATE(rang) SHARED(I)

On peut utiliser DEFAULT (LASTPRIVATE) ou DEFAULT (SHAREd) de la même manière

DEFAULT(NONE)

DEFAULT (PRIVATE)

Définit par défaut toutes les variables comme privées.

Exemple : !$OMP PARALLEL DEFAULT(PRIVATE) FIRSTPRIVATE(rang) SHARED(I)

On peut utiliser DEFAULT (LASTPRIVATE) ou DEFAULT (SHAREd) de la même manière

DEFAULT(NONE)

Oblige à déclarer le statut de toutes les variables

DEFAULT (PRIVATE)

Définit par défaut toutes les variables comme privées.

Exemple : !$OMP PARALLEL DEFAULT(PRIVATE) FIRSTPRIVATE(rang) SHARED(I)

On peut utiliser DEFAULT (LASTPRIVATE) ou DEFAULT (SHAREd) de la même manière

DEFAULT(NONE)

Oblige à déclarer le statut de toutes les variables

Permet d'éviter d'oublier des variables

DEFAULT (PRIVATE)

Définit par défaut toutes les variables comme privées.

Exemple : !$OMP PARALLEL DEFAULT(PRIVATE) FIRSTPRIVATE(rang) SHARED(I)

On peut utiliser DEFAULT (LASTPRIVATE) ou DEFAULT (SHAREd) de la même manière

DEFAULT(NONE)

Oblige à déclarer le statut de toutes les variables

Permet d'éviter d'oublier des variables

Exemple : !$OMP PARALLEL DEFAULT(NONE) FIRSTPRIVATE(rang) SHARED(I)

Code (boucle.F90)

```
do i=1,n
   a(i)=b(i)*c(i)
end do
```

Code (boucle.F90)

```
do i=1,n
   a(i)=b(i)*c(i)
end do
```

Version OpenMP, solution 1 (tirée du cours de l'Idris)

```
ntasks = OMP_GET_NUM_THREADS()
!$OMP PARALLEL PRIVATE(i,beg,end,rang)
rang = OMP_GET_THREAD_NUM()
print*, "hello, mon rang est ", rang
beg =1+( rang * n ) / ntasks
end =(( rang +1) * n ) / ntasks
do i=beg,end
   a(i)=b(i)*c(i)
end do
!$OMP END PARALLEL
```

Code (boucle.F90)

```
do i=1,n
   a(i)=b(i)*c(i)
end do
```

Version OpenMP, solution 2 (à éviter)

```
ntasks = OMP_GET_NUM_THREADS()
!$OMP PARALLEL PRIVATE(i,beg,end,rang)
rang = OMP_GET_THREAD_NUM()
print*, "hello, mon rang est ", rang
do i=1+rang,n,ntasks
   a(i)=b(i)*c(i)
end do
!$OMP END PARALLEL
```

Code (boucle.F90)

```
do i=1,n
   a(i)=b(i)*c(i)
end do
```

Version OpenMP, solution 3

On crée des paquets de plus en petits : solution utile quand la charge de travail varie d'une itération à l'autre, et qu'on utilise plus de threads que de processeurs. Permet d'améliorer l'équilibrage de charge.

Code (boucle.F90)

```
do i=1,n
   a(i)=b(i)*c(i)
end do
```

Version OpenMP

```
!$OMP PARALLEL
!$OMP DO
do i=1,n
   a(i)=b(i)*c(i)
end do
!$OMP END DO
!$OMP END PARALLEL
```

Version OpenMP

```
!$OMP PARALLEL
!$OMP DO
do i=1,n
   a(i)=b(i)*c(i)
end do
!$OMP END DO
!$OMP END PARALLEL
```

!$OMP DO :

Dans ce cas, la variable de boucle est automatiquement de type PRIVATE.

La boucle est coupée en NTHREADS paquets de taille équivalente (solution 1 du cas “à la main”)

Version OpenMP

```
!$OMP PARALLEL
!$OMP DO SCHEDULE(TYPE,size_paquet)
do i=1,n
   a(i)=b(i)*c(i)
end do
!$OMP END DO
!$OMP END PARALLEL
```

Version OpenMP

```
!$OMP PARALLEL
!$OMP DO SCHEDULE(TYPE,size_paquet)
do i=1,n
   a(i)=b(i)*c(i)
end do
!$OMP END DO
!$OMP END PARALLEL
```

CLAUSE SCHEDULE

Version OpenMP

```
!$OMP PARALLEL
!$OMP DO SCHEDULE(TYPE,size_paquet)
do i=1,n
   a(i)=b(i)*c(i)
end do
!$OMP END DO
!$OMP END PARALLEL
```

CLAUSE SCHEDULE

TYPE=STATIC : la boucle est découpée en paquets de taille size_paquet, qui sont répartis à l'avance sur les threads.

Version OpenMP

```
!$OMP PARALLEL
!$OMP DO SCHEDULE(TYPE,size_paquet)
do i=1,n
   a(i)=b(i)*c(i)
end do
!$OMP END DO
!$OMP END PARALLEL
```

CLAUSE SCHEDULE

TYPE=STATIC : la boucle est découpée en paquets de taille size_paquet, qui sont répartis à l'avance sur les threads.

TYPE=DYNAMIC : la boucle est découpée en paquets de taille size_paquet, qui sont répartis au fur et à mesure sur les threads. Dès qu'un thread a terminé, il reçoit un nouveau paquet.

Version OpenMP

```
!$OMP PARALLEL
!$OMP DO SCHEDULE(TYPE,size_paquet)
do i=1,n
   a(i)=b(i)*c(i)
end do
!$OMP END DO
!$OMP END PARALLEL
```

CLAUSE SCHEDULE

TYPE=GUIDED : la boucle est découpée en paquets de taille décroissante et supérieure à size_paquet, qui sont répartis au fur et à mesure sur les threads.

Version OpenMP

```
!$OMP PARALLEL
!$OMP DO SCHEDULE(TYPE,size_paquet)
do i=1,n
   a(i)=b(i)*c(i)
end do
!$OMP END DO
!$OMP END PARALLEL
```

CLAUSE SCHEDULE

TYPE=AUTO : choisi par le compilateur.

Version OpenMP

```
!$OMP PARALLEL
!$OMP DO SCHEDULE(TYPE,size_paquet)
do i=1,n
   a(i)=b(i)*c(i)
end do
!$OMP END DO
!$OMP END PARALLEL
```

CLAUSE SCHEDULE

TYPE=AUTO : choisi par le compilateur.

TYPE=RUNTIME : défini par la variable d'environnement.

Fusion des clauses

```
!$OMP PARALLEL DO SCHEDULE(TYPE,size_paquet)
do i=1,n
   a(i)=b(i)*c(i)
end do
!$OMP END PARALLEL DO
```

Définition

La réduction consiste à effectuer une opération sur plusieurs threads

Example : le produit scalaire

```
res=0
!$OMP PARALLEL DO REDUCTION(+:res)
do i=1,n
   res=res+b(i)*c(i)
end do
$OMP END PARALLEL DO
```

autres opérateurs

*,-,MAX,MIN,AND,OR,etc.

Définition

La fusion consiste à regrouper des boucles pour augmenter la taille des paquets.

COLLAPSE(m) permet de fusionner m boucles

Example : opération matricielle

```
res=0
!$OMP PARALLEL DO COLLAPSE(2) PRIVATE(k)
do i=1,n
  do j=1,n
    do k=1,100
   mat_a(i,j)=mat_a(i,j)+mat_b(i,j)*mat_c(i,j)+k
  end do
end do
$OMP END PARALLEL DO
```


autres opérateurs

*,-,MAX,MIN,AND,OR,etc.

Temps d'execution : cpu_time versus omp_get_wtime

Temps d'execution : cpu_time versus omp_get_wtime

cpu_time(t1) : calcule la somme des temps de tous les threads.

Temps d'execution : cpu_time versus omp_get_wtime

cpu_time(t1) : calcule la somme des temps de tous les threads.

t1=omp_get_wtime() : calcule le temps du thread le plus lent.

Temps d'execution : cpu_time versus omp_get_wtime

cpu_time(t1) : calcule la somme des temps de tous les threads.

t1=omp_get_wtime() : calcule le temps du thread le plus lent.

Passage à l'échelle : scalabilité forte versus scalabilité faible

Temps d'execution : cpu_time versus omp_get_wtime

cpu_time(t1) : calcule la somme des temps de tous les threads.

t1=omp_get_wtime() : calcule le temps du thread le plus lent.

Passage à l'échelle : scalabilité forte versus scalabilité faible

scalabilité forte : on augmente le nombre de threads à charge de calcul fixée.

Temps d'execution : cpu_time versus omp_get_wtime

cpu_time(t1) : calcule la somme des temps de tous les threads.

t1=omp_get_wtime() : calcule le temps du thread le plus lent.

Passage à l'échelle : scalabilité forte versus scalabilité faible

scalabilité forte : on augmente le nombre de threads à charge de calcul fixée.

scalabilité faible : on augmente le nombre de threads proportionnellement à la charge de calcul.

Le cours de l'Idris

http://www.idris.fr/formations/simd/