II. Introduction au parallélisme multitâche par directives via OpenMP

1) Description de l'approche

Cours et matériel supplémentaires sur internet

Le cours de l'IDRIS est à mon sens le plus complet en français et anglais:
http://www.idris.fr/formations/openmp/

Le site du consortium fournit une documentation en ligne complète :
https://www.openmp.org/resources/refguides/

Ce cours n'est qu'une introduction très superficielle à OpenMP

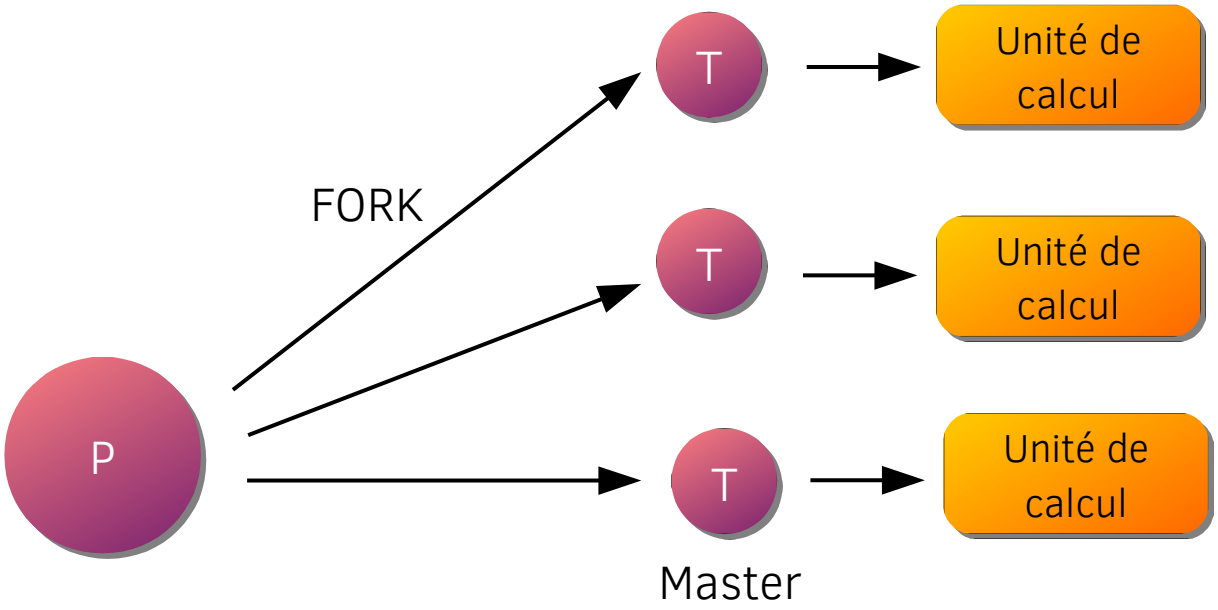
Fonctionnement d'un modèle multitâche *fork-join*

Multithreaded model

Démarrage séquentiel du programme

Fonctionnement d'un modèle multitâche *fork-join*

Multithreaded model

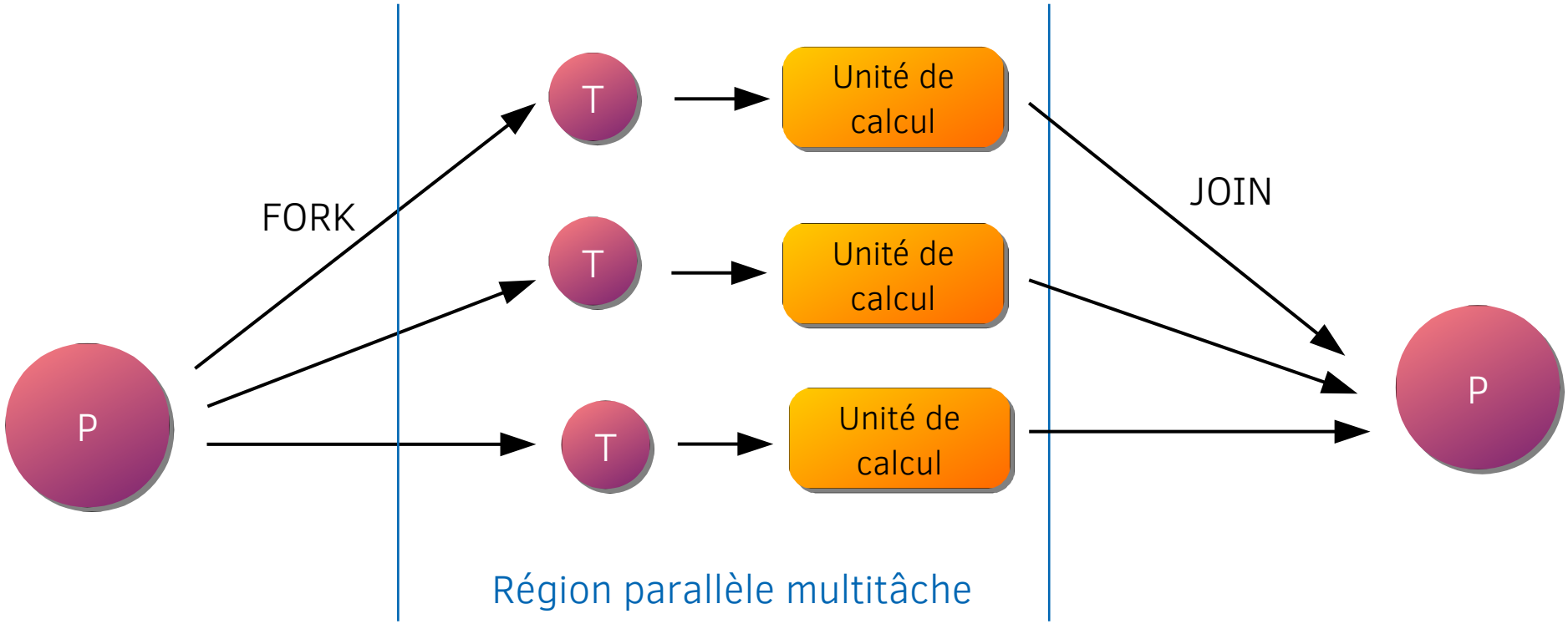


Le processus principal génère des threads secondaires (*slave*) pour résoudre un problème en parallèle (boucle par exemple).

Le processus initial devient le thread maître (*master*).

Fonctionnement d'un modèle multitâche *fork-join*

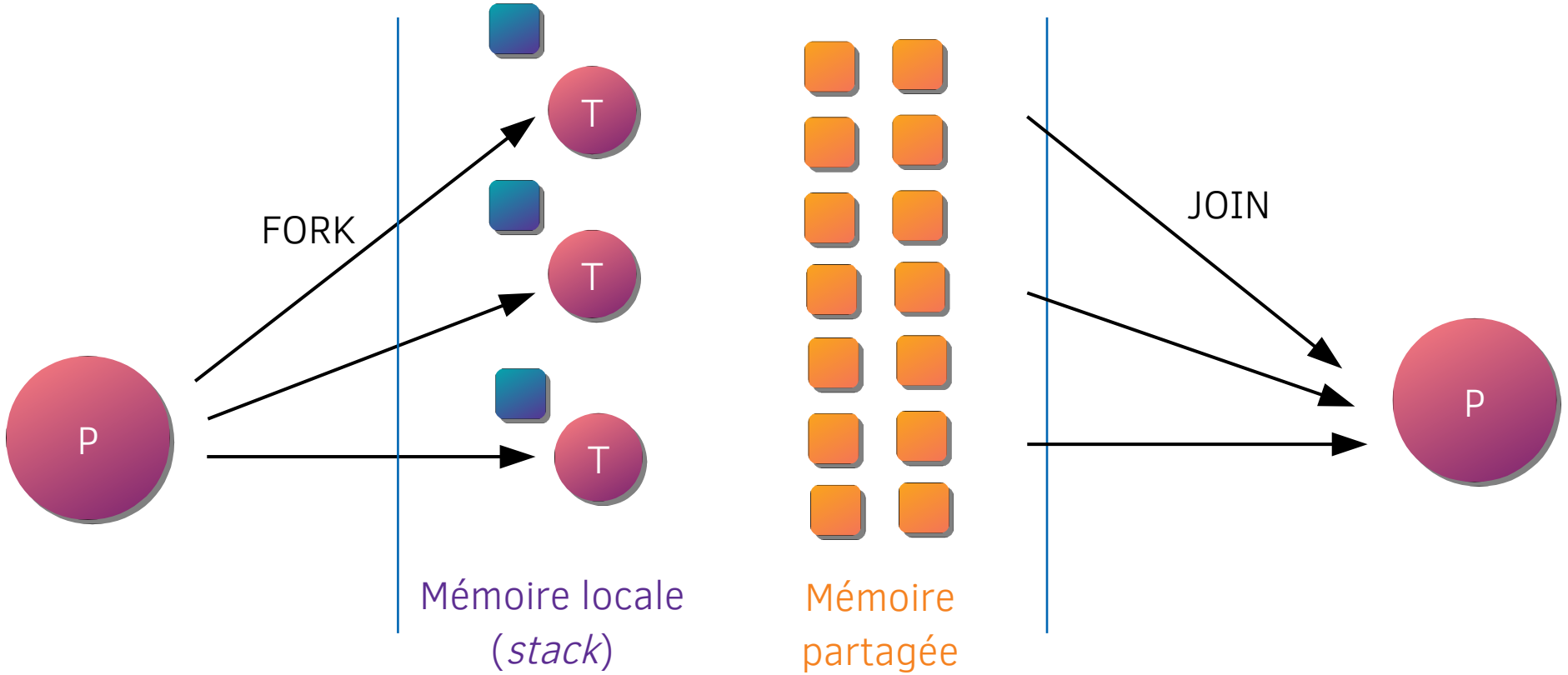
Multithreaded model



Fin de la région parallèle, les threads secondaires sont détruits et le programme redevient séquentiel.

Fonctionnement d'un modèle multitâche *fork-join*

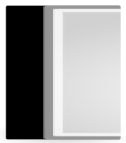
Multithreaded model



La mémoire est partagée par défaut entre tous les threads mais il existe une mémoire privée (déclaration locales de certaines variables) appelée *stack*.

Principe de la programmation par directives

Une directive est une ligne de code compréhensible par le compilateur qui va aiguiller la compilation pour un bloc de code dans le langage utilisé.


.f90

```
…
Mon code fortran…
…

!$omp … (directive OpenMP en Fortran)

…
La suite de mon code…
…
```

Principe de la programmation par directives

OpenMP fournit également une bibliothèque avec des fonctions appelables directement depuis le code

.f90

```
Use omp_lib

…
Mon code fortran…
…

Call fonction_openmp()...

…
La suite de mon code…
```

Où trouver OpenMP ?

La plupart des compilateurs récents (GNU, INTEL, LLVM...) possèdent une implémentation d'OpenMP installé par défaut.

Attention tout de même, les implémentations Fortran sont souvent en retard sur les implémentations C/C++.

II. Introduction au parallélisme multitâche par directives via OpenMP

2) Les directives de base d'OpenMP

Compilation d'un programme OpenMP (Fortran)

La compilation nécessite simplement l'ajout d'un paramètre à la ligne de compilation classique.

Pour compiler en activant OpenMP:

```
> gfortran -fopenmp -O3 program.f90 -o executable
```

Attention, la syntaxe du paramètre dépend du compilateur

Execution d'un programme OpenMP (Fortran)

L'exécution se fait comme un programme classique mais des paramètres d'environnement sont à spécifier pour contraindre son déroulement (si non spécifié dans le code).

Pour spécifier le nombre de threads dans les régions parallèles

```
> export OMP_NUM_THREADS=4
> ./executable
```

Pour spécifier le type ordonnanceur lors de la parallélisation des boucles

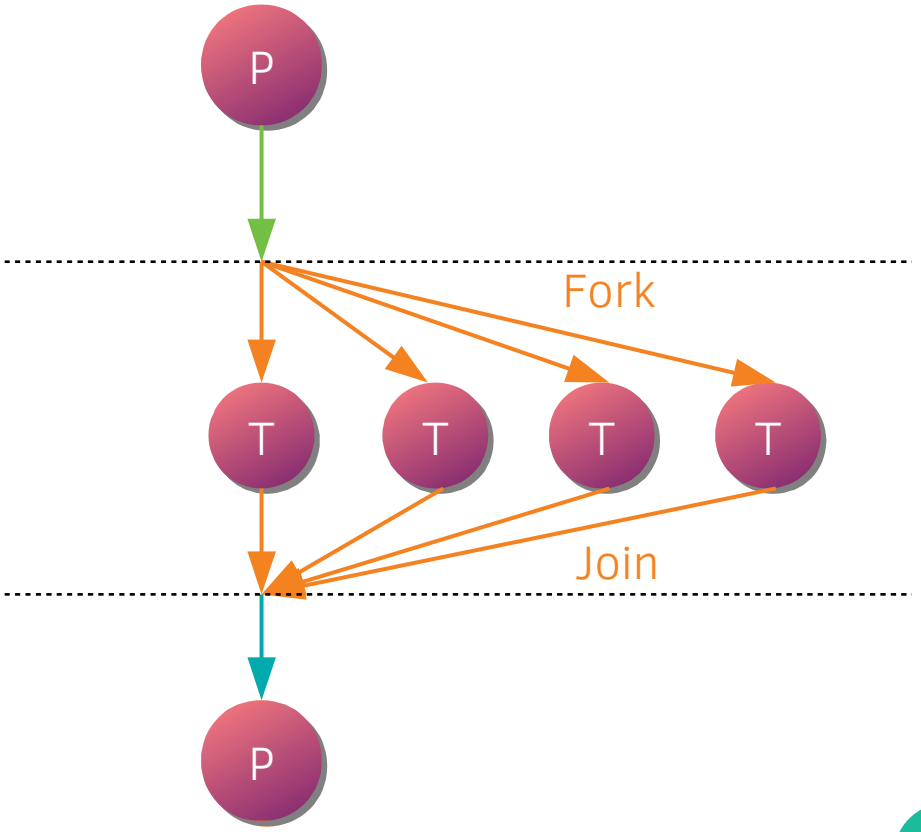
```
> export OMP_SCHEDULER=DYNAMIC
> ./executable
```

Création d'une région parallèle `!$ omp parallel`

Cette directive permet de spécifier la création d'une région parallèle : Le code présent dans la région parallèle est exécuté par chaque thread.

.f90

```
Program openmp

Implicit none

… partie séquentielle

!$omp parallel

… partie parallèle

!$omp end parallel

… partie séquentielle

End program
```

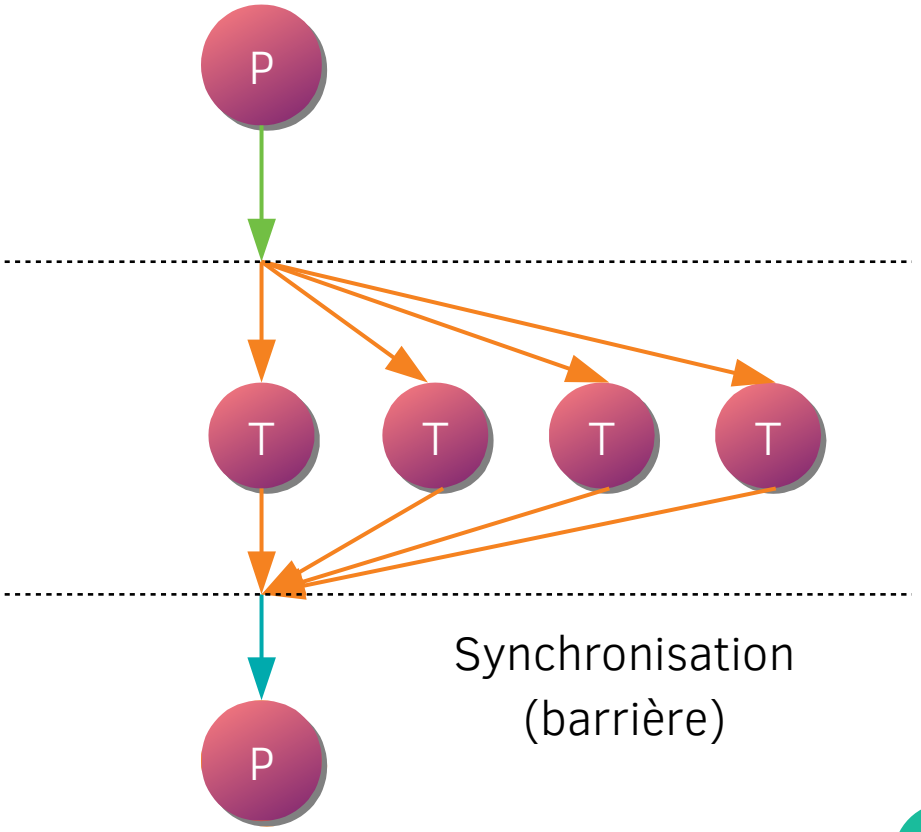


Création d'une région parallèle `!$ omp parallel`

Au moment de la fermeture de la région parallèle (**`!$omp end parallel`**), il y a une synchronisation de tous les threads (attente que chacun a fini son travail)

.f90

```
Program openmp

Implicit none

… partie séquentielle


!$omp parallel

… partie parallèle

!$omp end parallel


… partie séquentielle

End program
```



Notion de variable partagée
shared

Les variables déclarées avant la région parallèle sont par défaut partagées par les threads.

Une variable partagée ne peut pas être modifiée par tous les threads en même temps au risque d'avoir de la concurrence sur l'écriture en mémoire et d'obtenir une valeur aléatoire.


.f90

```
Program openmp

Implicit none

Integer :: A

!$omp parallel

… partie parallèle
… la même variable A est partagée par les threads

!$omp end parallel

End program
```

Notion de variable partagée
shared

Il est tout de même conseillé de spécifier le comportement par défaut grâce à la clause **`default`**.
Pour ne plus avoir de comportement par défaut : **`default(none)`**


.f90

```
Program openmp

Implicit none

Integer :: A

!$omp parallel default(shared)

… partie parallèle
… la même variable A est partagée par les threads

!$omp end parallel

End program
```

Notion de variable privée
private

Lorsqu'une variable prendra des valeurs différentes pour chaque thread (redéclaration locale). Il est préférable de la déclarer comme une valeur privée en utilisant la clause `private`.


.f90

```
Program openmp

Implicit none

Integer :: A, B

!$omp parallel default(none) shared(A) private(B)

… partie parallèle
… la même variable A est partagée par les threads
… la variable B est dupliquée pour chaque thread (dans la stack)

!$omp end parallel

End program
```

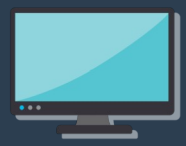
Notion de variable privée
private

La valeur d'une variable privée définie avant une région parallèle n'est pas copiée dans les versions locales aux threads

.f90

```
Program openmp

Implicit none

Integer :: A, B

B = 5

!$omp parallel default(none) shared(A) private(B)

… B n'est pas initialisé

!$omp end parallel

End program
```

Exercice n°1 : votre premier programme OpenMP



- Rendez vous sur le Gitlab des exercices : https://gitlab.maisondelasimulation.fr/mlobet/cours-hpc-m2-dfe
- Télécharger les exercices sur votre session de travail
- Décompressez l'archive en ligne de commande

```
> tar xvf archivedossier.tar
```

- Rendez vous dans le dossier de l'exercice n°1 OpenMP appelé `1_omp_parallel`

```
> cd exercises/openmp/1_omp_parallel
```

- Ouvrez les instructions contenues dans le fichier README.md avec votre éditeur de fichier favori (vim, emacs, atom, gedit...)

Vous pouvez lire le README directement depuis le Gitlab et c'est plus confortable comme ça.

Parallélisation d'une boucle DO

La directive **!$OMP DO** permet de distribuer le travail d'une boucle dans une région parallèle entre tous les threads.

.f90

```
Program openmp

Implicit none

Real, dimension(N) :: A, B, C, D
Integer            :: i

!$omp parallel default(shared) private(i)

DO i=1,N
  A(i) = B(i) + C(i)*D(i)
ENDDO

!$omp end parallel

End program
```

La boucle est ici exécuté par tous les threads en même temps

Parallélisation d'une boucle DO

La directive **!$OMP DO** permet de distribuer le travail d'une boucle dans une région parallèle entre tous les threads.

.f90

```
Program openmp

Implicit none

Real, dimension(N) :: A, B, C, D
Integer            :: i

!$omp parallel default(shared) private(i)

!$omp do
DO i=1,N
  A(i) = B(i) + C(i)*D(i)
ENDDO
!$omp end do

!$omp end parallel

End program
```

La charge est maintenant partagée entre les threads disponibles

Partage du travail avec la clause SCHEDULE(type,chunk)

La clause **schedule** permet d'expliciter la manière de partager le travail


.f90

```
Program openmp

Implicit none

Real, dimension(N) :: A, B, C, D
Integer            :: i

!$omp parallel default(shared) private(i)

!$omp do schedule(static, 10)
DO i=1,N
  A(i) = B(i) + C(i)*D(i)
ENDDO
!$omp end do

!$omp end parallel

End program
```

Partage du travail avec la clause `SCHEDULE(type,chunk)`

La clause **`schedule`** permet d'expliciter la manière de partager le travail :

- **`Static`** : partage des itérations en paquet de taille **`chunk.`** Les paquets sont attribués de façon cyclique.
- **`Dynamic`** : partage des itérations en paquets de taille **`chunk`**. Les paquets sont distribués dynamiquement dès qu'un thread est disponible.
- **`Guided`** : partage des itérations suivant une taille de paquet qui décroît. La taille est supérieure ou égale à **`chunk`**.
- **`Auto`** : le compilateur décide
- **`Runtime`** : la décision est prise en utilisant la variable d'environnement **`OMP_SCHEDULE`** avant l'exécution

```
> export OMP_SCHEDULE= « DYNAMIC, 100 »
```

Retour au séquentiel au sein des régions parallèles

Ouvrir et fermer régulièrement une région parallèle (`!$omp parallel`) possède un coût (overhead). Il est recommandé de le faire le moins possible au cours d'un programme, en particulier dans les régions intensives.

Il est pourtant parfois nécessaire d'effectuer des calculs sur un seul thread (affichage à l'écran par exemple, lecture ou sortie de fichier, appel de fonctions non multitâches…). Pour cela, on peut utiliser :

- `!$omp master` : seul le thread master exécute ce qui est dans la région qui suit. Les autres threads passent leur chemin et exécutent la suite.

- `!$omp single` : le premier thread qui arrive au niveau de cette directive se charge de la suite, les autres attendent qu'il finisse.

- `!$omp critical` : Tous les threads exécutent cette partie mais à tour de rôle dans un ordre non déterminé. Cette région est protégée par le thread en cours.

Retour au séquentiel au sein des régions parallèles : `!$omp master`

`!$omp master` : seul le thread master exécute ce qui est dans la région qui suit. Les autres threads passent leur chemin et exécutent la suite.

.f90

```
Program openmp

Implicit none

Real, dimension(N) :: A, B, C, D
Integer            :: i

!$omp parallel default(shared) private(i)

… partie parallèle

!$omp master

Print*, « je suis le master »

!$omp end master

!$omp end parallel

End program
```

Seul le thread master affichera l'information à l'écran

Retour au séquentiel au sein des régions parallèles : `!$omp single`

- `!$omp single` : le premier thread qui arrive au niveau de cette directive se charge de la suite, les autres attendent qu'il finisse.

.f90

```
Program openmp
Implicit none
Integer             :: id
!$omp parallel default(shared) private(id)
… partie parallèle
!$omp single
Id = omp_get_thread_num()
Print*, « je suis le thread », id
!$omp end single
!$omp end parallel
End program
```

Le premier thread qui arrive ici affichera son rang dans le terminal

Où trouver OpenMP ?

A partir d'OpenMP 5, de nombreux changements apparaissent dans la nomenclature mais les compilateurs ne sont pas encore à jour.

Exercice n°2 : Parallélisation d'une boucle OpenMP

- Rendez vous sur le Gitlab des exercices : https://gitlab.maisondelasimulation.fr/mlobet/cours-hpc-m2-dfe
- Télécharger les exercices sur votre session de travail
- Décompressez l'archive en ligne de commande

```
> tar xvf archivedossier.tar
```

- Rendez vous dans le dossier de l'exercice n°2 OpenMP appelé `2_omp_do`

```
> cd exercises/openmp/2_omp_do
```

- Ouvrez les instructions contenues dans le fichier README.md avec votre éditeur de fichier favori (vim, emacs, atom, gedit...)

Vous pouvez lire le README directement depuis le Gitlab et c'est plus confortable comme ça.

Fin de la partie sur OpenMP

A ce stade du cours, vous savez maintenant :

- Utiliser des directives
- Paralléliser un programme composé de boucles simples en OpenMP