INSTITUT DES SCIENCES ET TECHNIQUES DES YVELINES

Programmation Parallèle et Distribuée avec OpenMP

IATIC4, ISTY(2018-2019)

Nahid Emad

Laboratoire
d'**I**nformatique
Parallélisme
Réseaux
Algorithmique
Distribuée

MAISON DE LA SIMULATION

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

Plan

- Introduction
- Principaux modèles de programmation et d'exécution parallèle
- Programmation parallèle avec openMP
- Bibliographie et sites Web

Programmation parallèle avec openMP

- Qu'est-ce OpenMP?
- Quel modèle d'exécution pour OpenMP?
- Caractéristiques d'OpenMP
  - Régions parallèle
  - Partage du travail,
  - Données d'environnement,
- Placement de données,
- Notions avancées

Programmation parallèle avec openMP

- Qu'est-ce OpenMP?
- Quel modèle d'exécution pour OpenMP?
- Caractéristiques d'OpenMP
  - Régions parallèle
  - Partage du travail,
  - Données d'environnement,
- Placement de données,
- Notions avancées

Programmation multitâches sur architecture à mémoire partagée

- Des tâches s'exécutent en parallèle
- La mémoire est commune entre elles et est partagée (physiquement ou virtuellement)
- *Les tâches communiquent par lectures et écritures dans la mémoire partagée*. Par ex. les cœurs d'un processeur se partagent la mémoire du processeur (un processus / tâche peut être attribué à un cœur).

Programmation multitâches sur architecture à mémoire partagée

La librairie de threads POSIX *Pthreads* est adoptée par la plupart des SE. L'écriture d'un code nécessite un nombre important de lignes dédiées aux threads.

Par exemple: paralléliser une boucle implique la déclaration ses structures de thread, la création des threads, Le calcul des bornes de boucles, leur affectation aux threads, ...

OpenMP: Une solution pour remédier à ces problèmes pour la programmation parallèle.

Exemple

Paralléliser le code suivant en utilisant des threads:

```
for (i=0; i<n; i++)
 {
     sum = sum + sqrt(Tab[i]);
 }
```

Pourquoi cela est t-il difficile?

- Nécessite ***mutex*** pour les accès à ***sum***
- Écrire des codes différents pour les versions séquentielle et serial and parallèle
- Pas de réglage intégré (# of processors?)

OpenMP (Open specifications for MultiProcessing)

- Une API portable permettant la programmation à mémoire partagée (existe pour Fortran, C et C ++)
- Parallélisme de tâches et parallélisme de données.
- Combine les versions séquentielle et parallèle d'un programme dans un même code.
- Standardisé ~ 20 ans d'expérience de threading assuré par le compilateur.

http://www.openmp.org

Spécification courente est OpenMP 3.0

318 Pages

(combined C/C++ and Fortran)

Programmation multitâches sur architecture à mémoire partagée

Partant d'un code séquentiel, on choisit, par exemple, une boucle qui peut être exécutée en parallèle et on lui applique la directive OpenMP adhéquate.

La conversion de programme est assez mécanique: il faut simplement dire que la boucle doit être exécutée en parallèle et laisser le compilateur faire le reste.

OpenMP fait exactement ça !!!

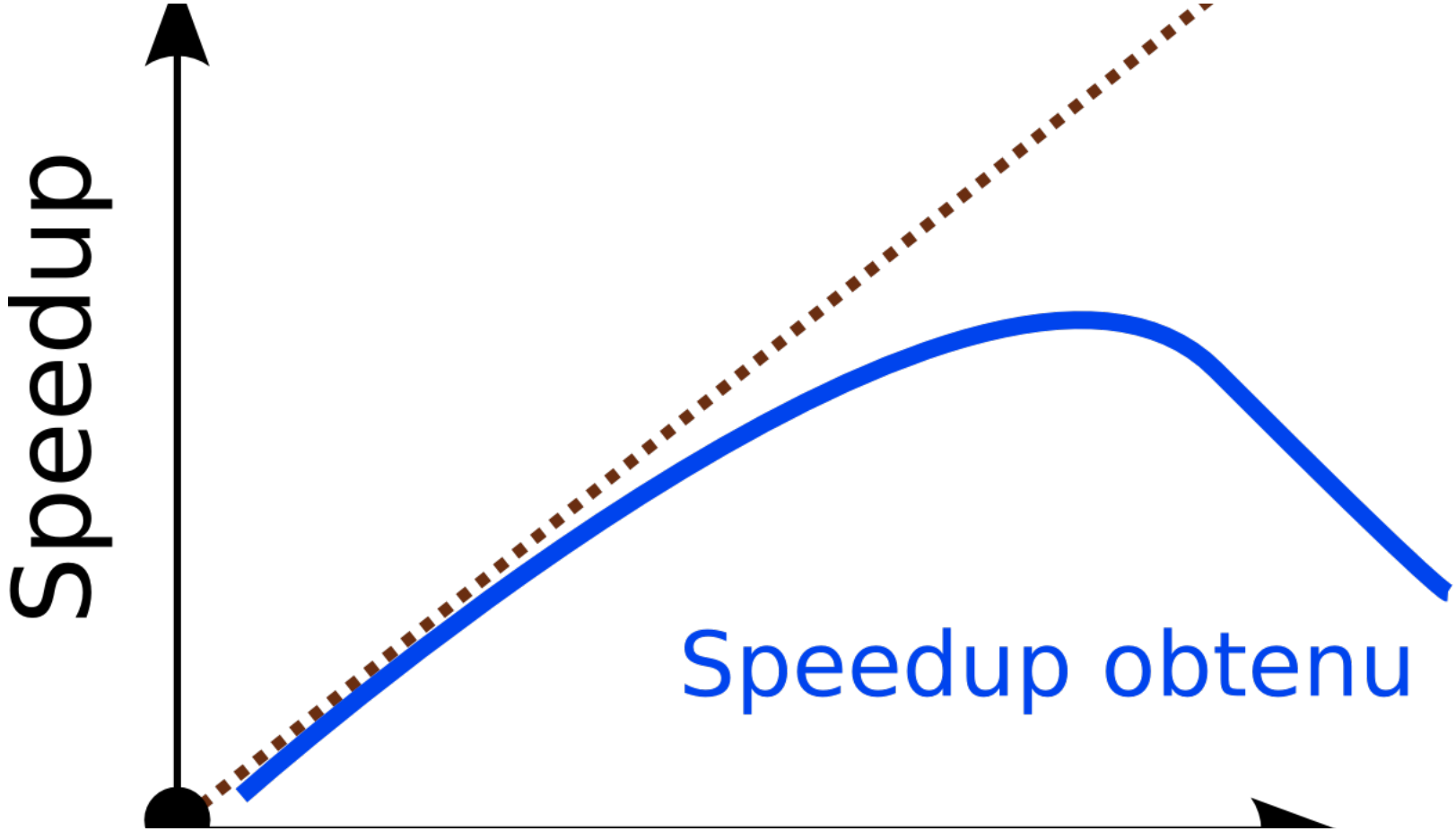
OpenMP

- Avantages
  - Standard mûre (20 ans)
  - Portabilité
  - Langage haut niveau
- Assez simple à utiliser
  - Communications implicites
  - En opposition à MPI

OpenMP

- Désavantages
  - Communications implicites (!)
  - C'est le compilateur qui prend les décisions
  - Difficile à se rendre compte de ce qui se passe
  - Souvent difficile d'avoir un bon speedup
  - Problèmes de scalabilité surtout

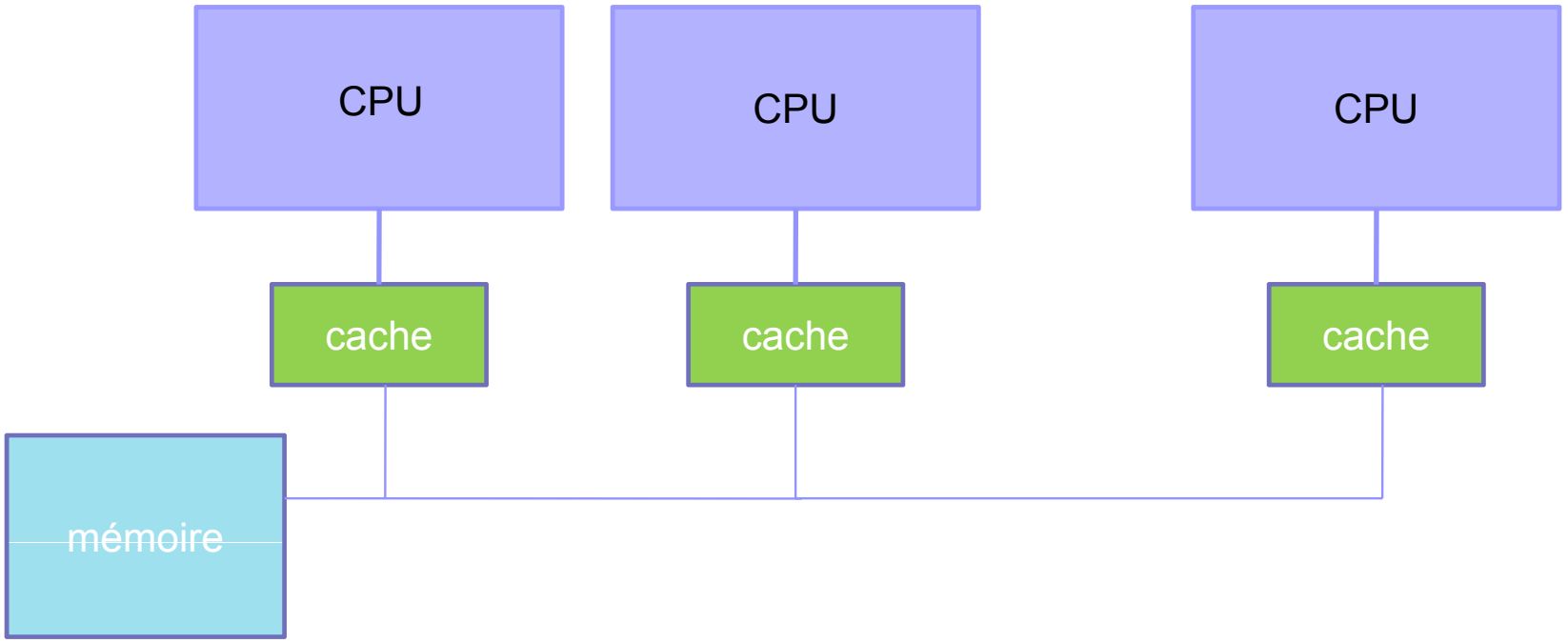
Performances



Programmation parallèle avec openMP

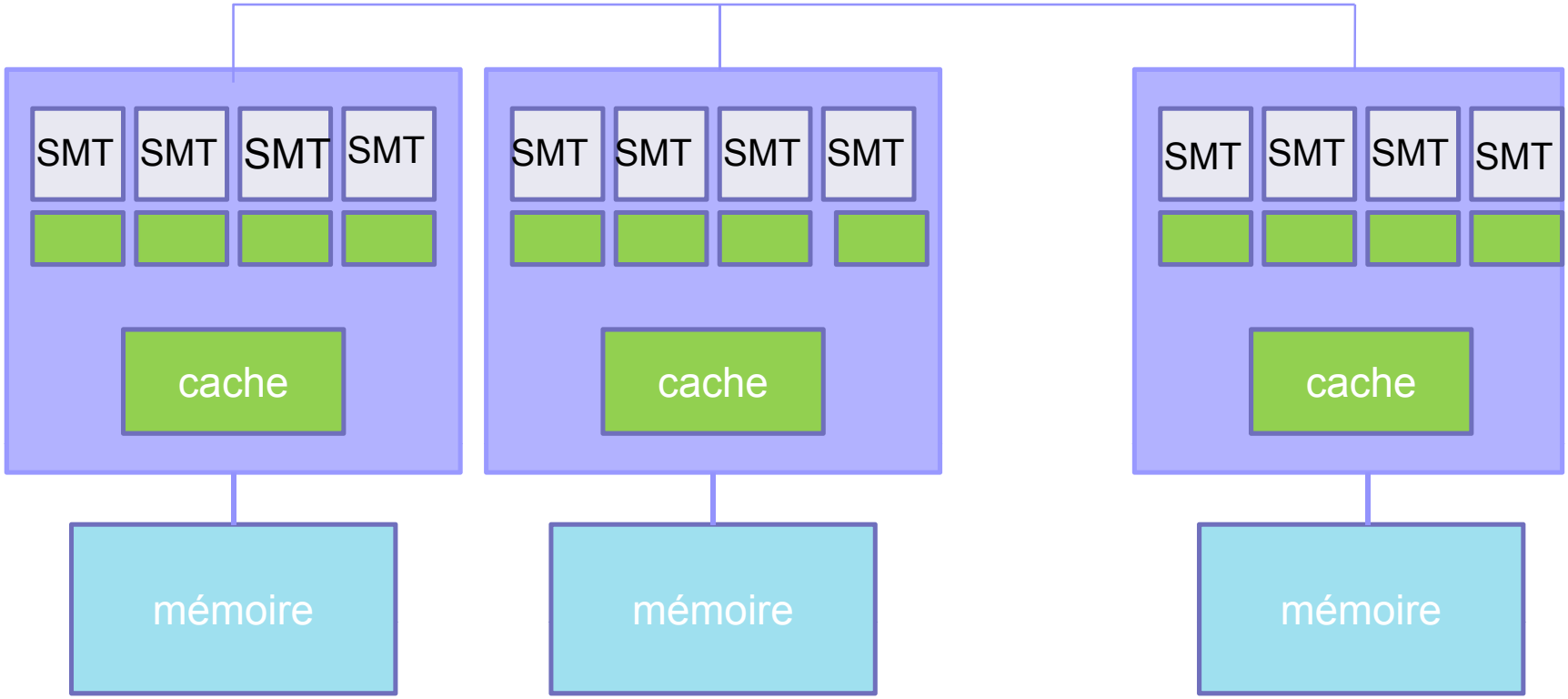
- Qu'est-ce OpenMP?
- Quel modèle d'exécution pour OpenMP?
- Caractéristiques d'OpenMP
  - Régions parallèle
  - Partage du travail,
  - Données d'environnement,
- Placement de données,
- Notions avancées

Architecture UMA (Uniform Memory Access)



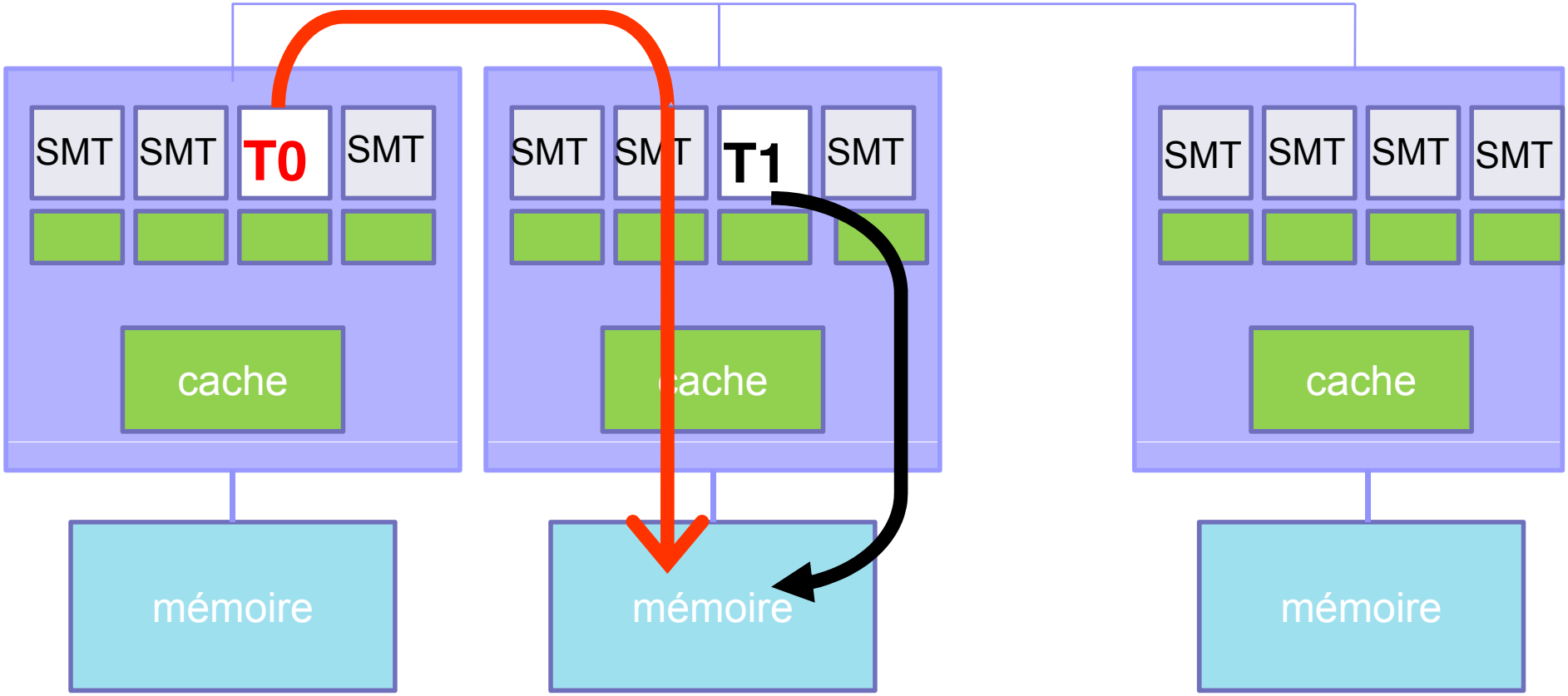
Problème inhérent : les contentions mémoire

Architecture NUMA (Non-Uniform Memory Access)



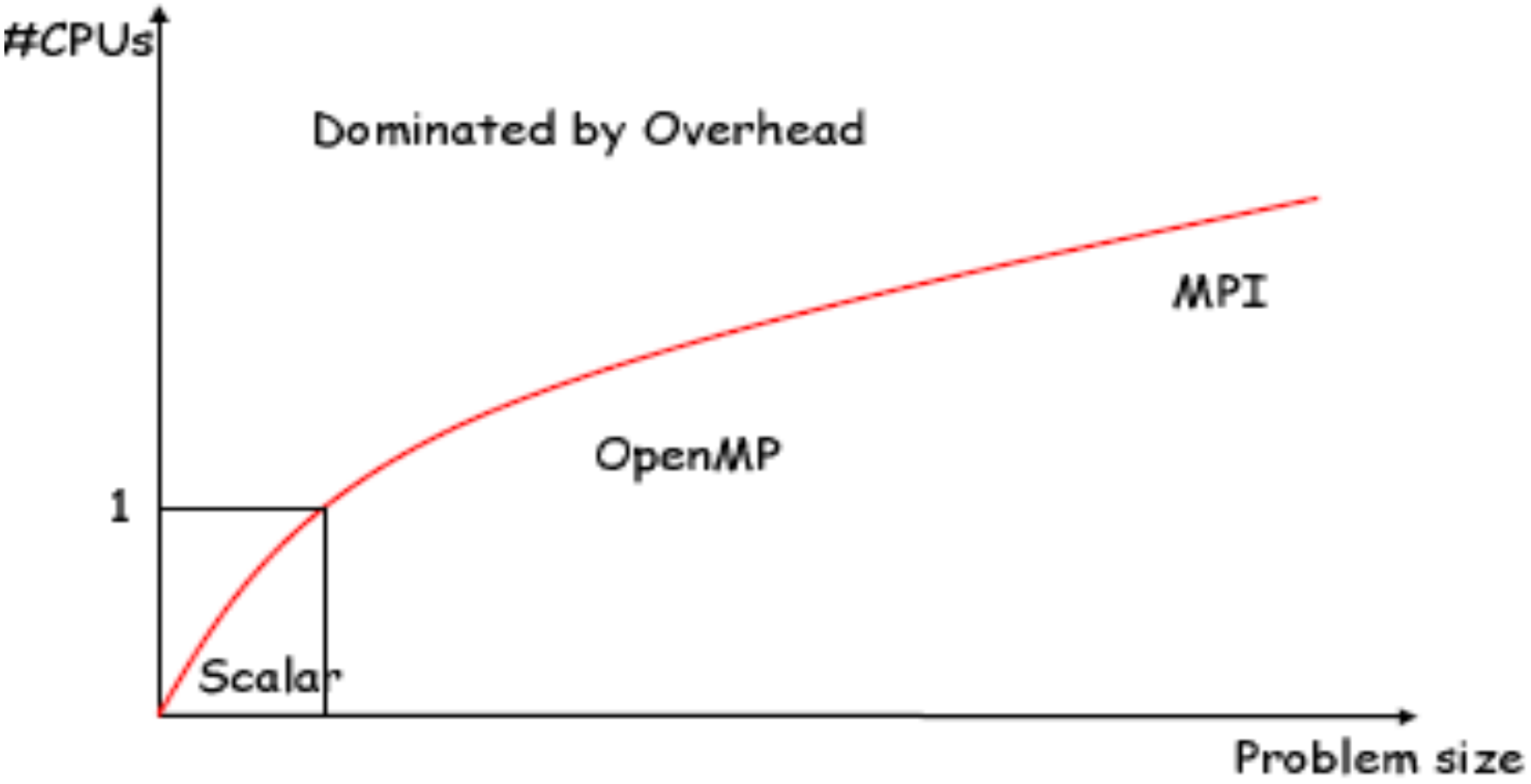
Inconvénient: programmation plus compliquée

Programmation multitâches sur architecture multi-cœurs NUMA

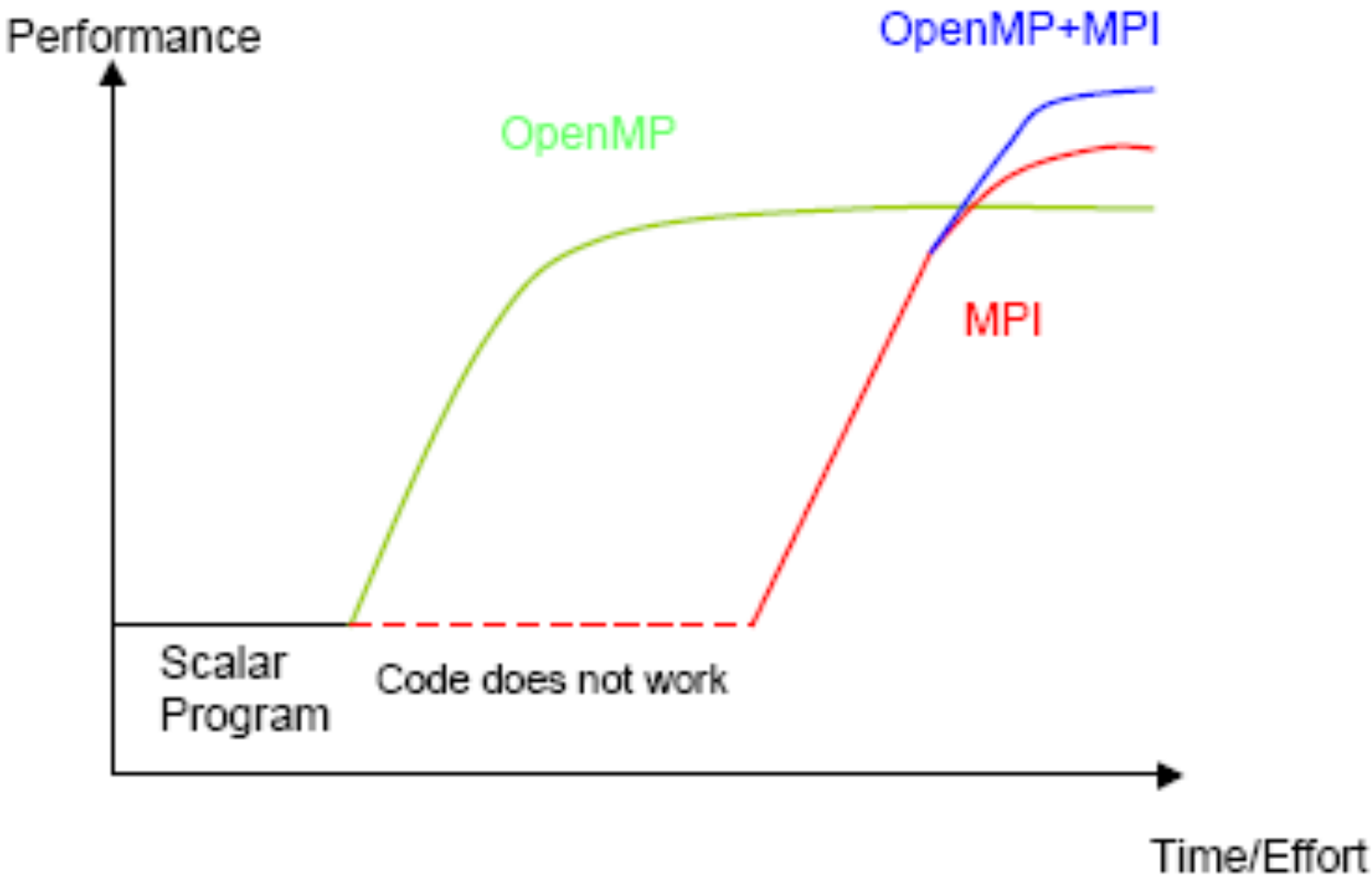


Communication local versus Communication distante

Pourquoi utiliser OpenMP?



Pourquoi utiliser OpenMP?

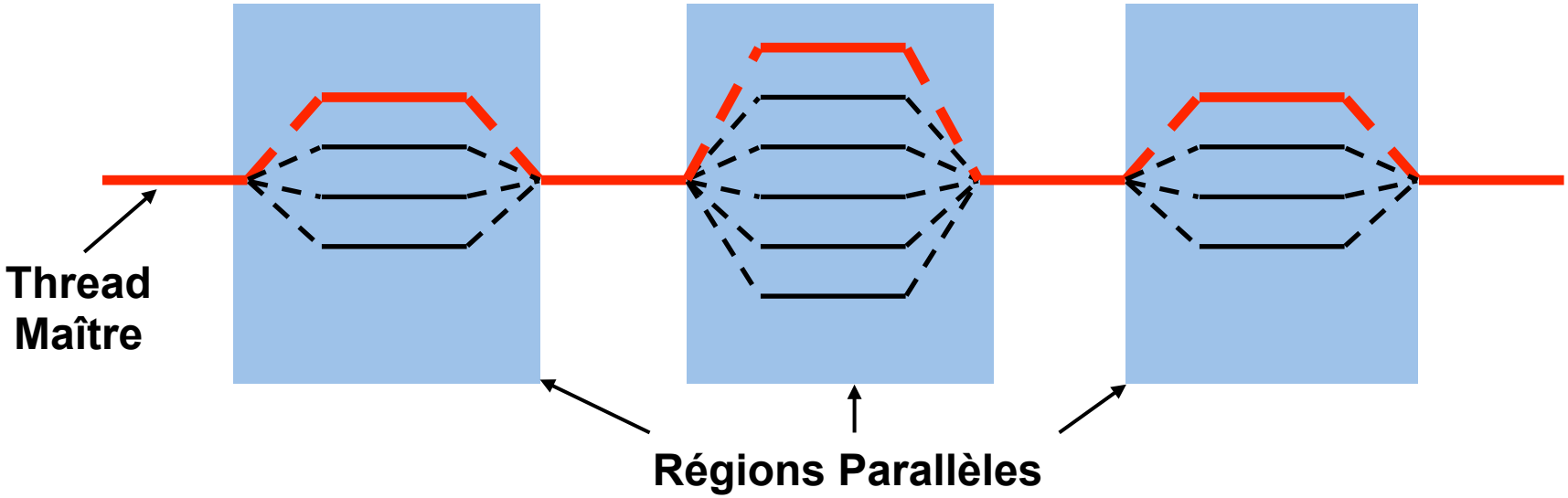


Programmation parallèle avec openMP

- Qu'est-ce OpenMP?
- Quel modèle d'exécution pour OpenMP?
- Caractéristiques d'OpenMP
  - Régions parallèle
  - Partage du travail,
  - Données d'environnement,
- Placement de données,
- Notions avancées

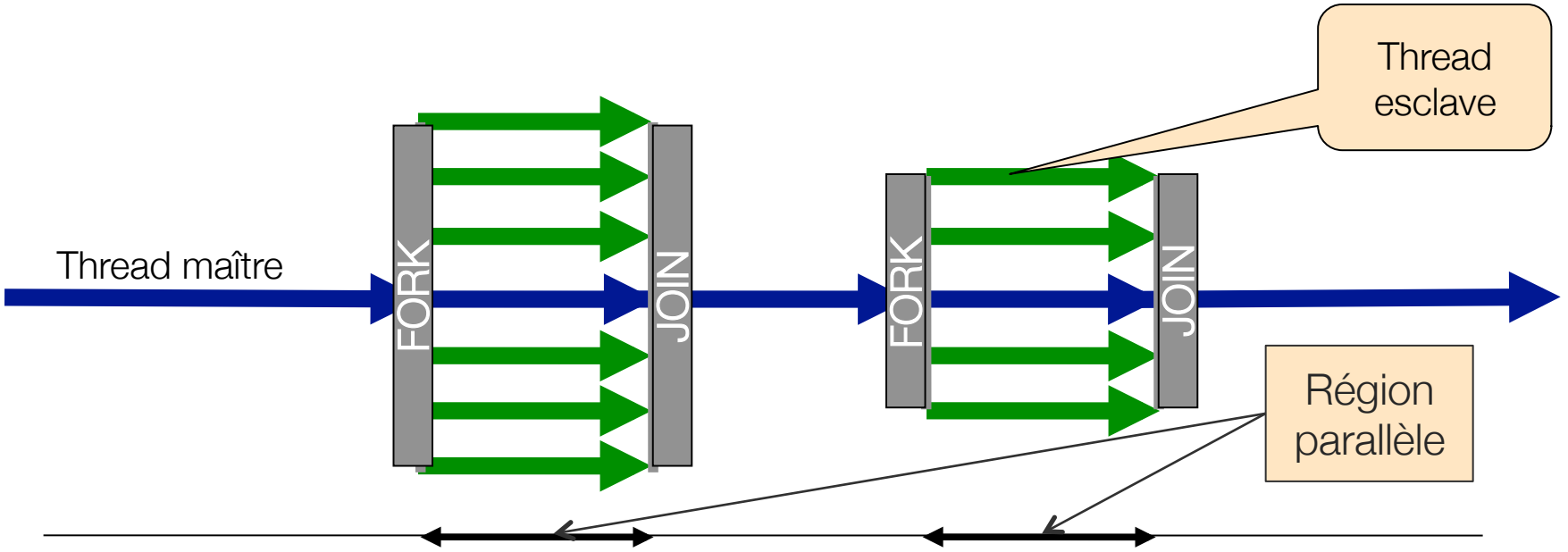
Exécution de programme avec OpenMP

- Un programme OpenMP est exécuté par un unique Thread.
- Le thread principal génère une équipe de threads au besoin.
- Le programme séquentiel évolue en programme parallèle



Exécution de programme avec OpenMP

Fork & Join: le thread principal génère une équipe de threads si nécessaire



OpenMP

- Calcul parallèle sur machine à mémoire partagée (MP)
- Portable sur des architectures séquentielle et parallèles à MP
- Basé sur le compilateur
- Peut être étendu aux langages de programmation existants:
  - Essentiellement par des directives
  - Et quelques librairies
- Disponible pour Fortran et C/C++
- Permet aussi le parallélisme de données.

Programmation avec OpenMP

- **Modèle de mémoire partagée:**
  - Les threads communiquent en accédant à des variables partagées
    - Le partage est défini syntaxiquement
    - Toute variable vue par deux ou plusieurs threads est partagée
    - Toute variable vue par un seul thread est privée
- **Conseils de programmation:**
    - Utilisez la synchronisation pour éviter les conflits
- Changez la façon dont les données sont stockées pour minimiser la synchronisation

Construction OpenMP

Pour pouvoir utiliser les fonctions de librairies OpenMP, il faut l'inclure dans son fichier d'entêtre programme:

```
#include <omp.h>
```

La plupart des constructions OpenMP sont des directives ou pragmas de compilation

Pour C et C++, les pragmas sont sous la forme:

```
#pragma omp construct [clause [clause]…]
```

Un programme simple OpenMP

- OpenMP basé sur directives/pragmas de compilation
- OpenMP est concentré surtout sur le parallélisme de boucles

Serial Program:

```
void main()
{
   double Res[1000];

   for(int i=0;i<1000;i++) {
         do_huge_comp(Res[i]);
   }
}
```

Parallel Program:

```
void main()
{
   double Res[1000];
#pragma omp parallel for
   for(int i=0;i<1000;i++) {
         do_huge_comp(Res[i]);
   }
}
```

Partage de travail

Le partage de travail est le terme général utilisé dans OpenMP pour décrire la répartition du travail entre les threads.

Voici trois exemples de partage de travail sous OpenMP:

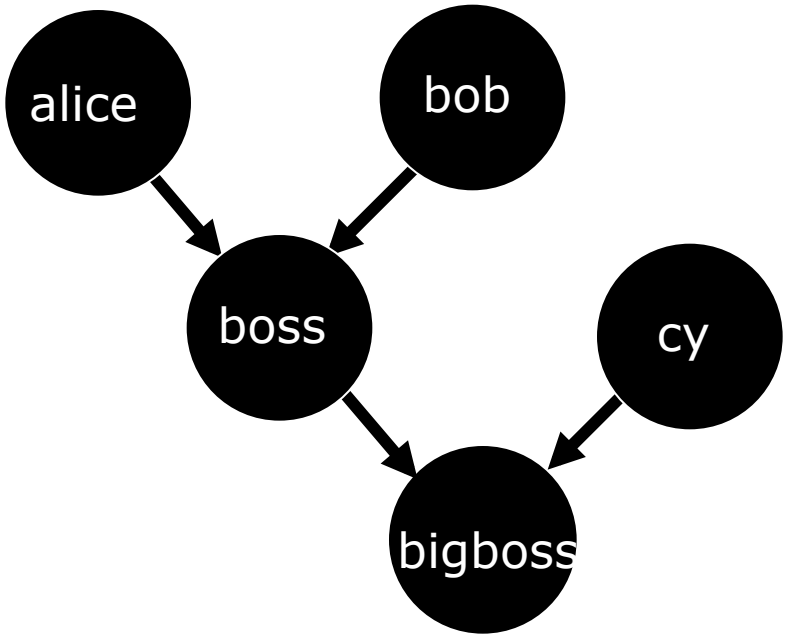
omp sections: construction de sections omp

omp for: construction de boucle omp

omp task: construction de tâche omp

Partage de travail

```
a = alice();
b = bob();
s = boss(a, b);
c = cy();
printf ("%6.2f\n", bigboss(s,c));
```



alice, bob, et cy peuvent être calculés en parallèle

Sections OpenMP

#pragma omp sections

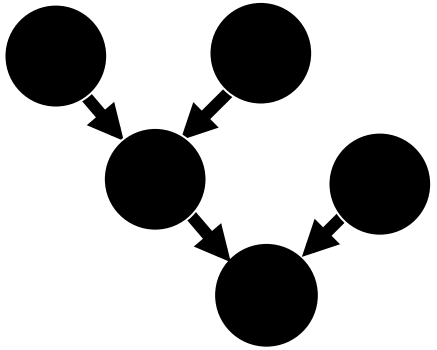
- Doit être dans une région parallèle
- Précède un bloc de code contenant N blocs de code pouvant être exécutés simultanément par N threads
  Englobe chaque section omp

#pragma omp section

- Précède chaque bloc de code dans le bloc englobant décrit ci-dessus
- Peut être omis pour la première section parallèle après le *pragma* des sections parallèles
- Les segments de programme inclus sont distribués pour une exécution parallèle entre les threads disponibles

Ex1: Partage de travail avec sections OpenMP

```
#pragma omp parallel sections
{
    #pragma omp section  /* optionnel */
      a = alice();
    #pragma omp section
      b = bob();
    #pragma omp section
       c = cy();
}
s = boss(a, b);
printf ("%6.2f\n",bigboss(s,c));
```



Ex2: Partage de travail avec Sections OpenMP

```
reponse1 = long_calcul_1();
reponse2 = long_calcul_2();
if (reponse1 != reponse2) { … }
```

Comment paralléliser?

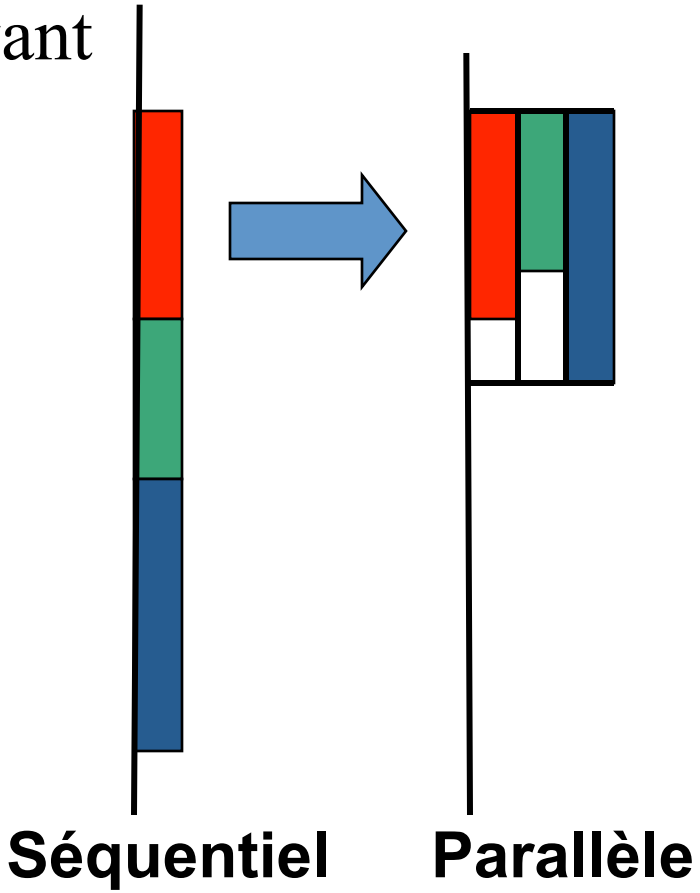
```
#pragma omp sections
{ #pragma omp section
  reponse1 = long_calcul_1();
  #pragma omp section
  reponse2= long_calcul_2(); }
if (reponse1 != reponse2) { … }
```

Avantage des sections parallèles

Sections indépendantes de code pouvant être exécutées en parallèle:

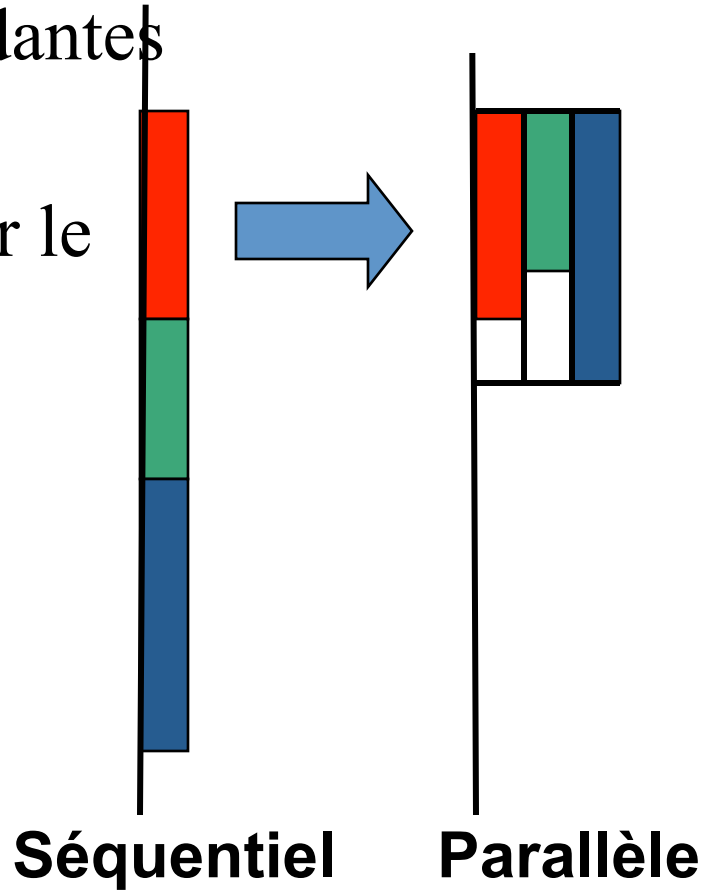
réduction du temps d'exécution

```
#pragma omp parallel sections
{
   #pragma omp section
   phase1();
   #pragma omp section
   phase2();
   #pragma omp section
   phase3();
}
```



Qu'est-ce une tâche?

- Les tâches sont des unités indépendantes de travail.
- Un thread est désigné pour exécuter le travail d'une tâche
- Le « SE » qui décide quelle tâche s'exécutera en premier



Une tâche est composée de

- **Code** à exécuter
- **Données** d'environnement
- **Variables** de contrôle interne