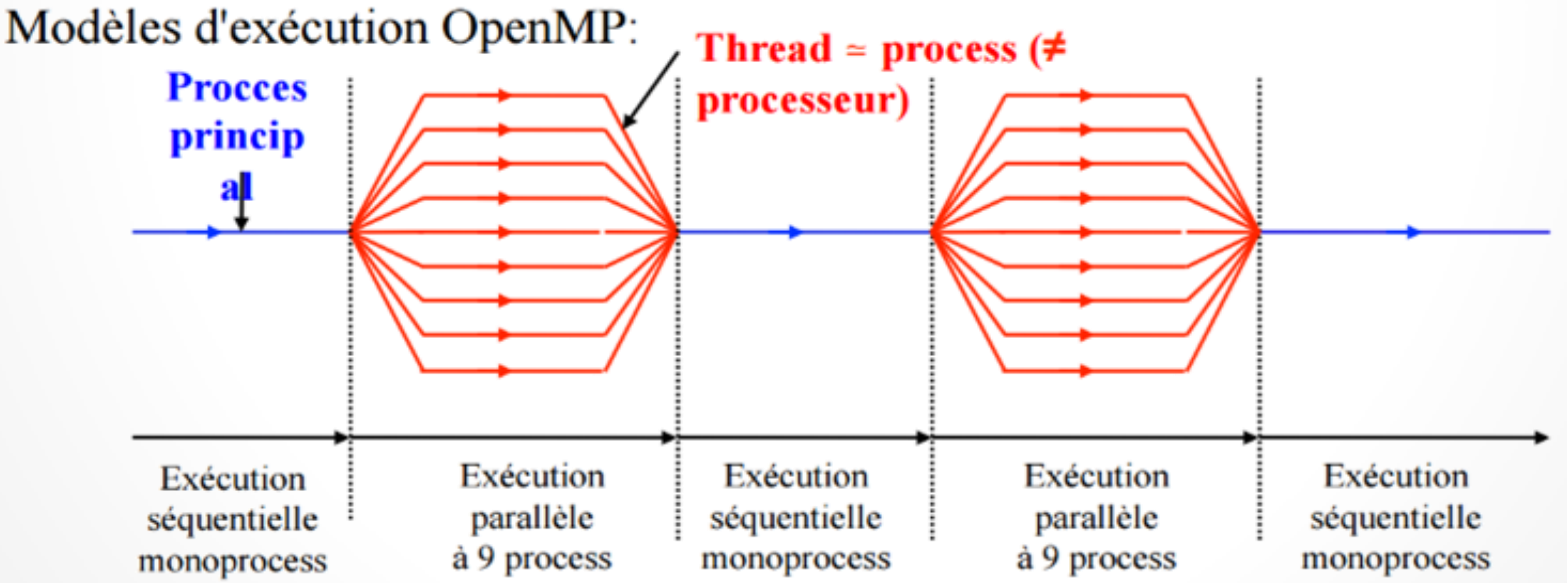
La parallélisation facile OpenMP

- **Open MP (Open Multi-Processing)** est une interface de programmation pour le calcul parallèle sur architecture à mémoire partagée

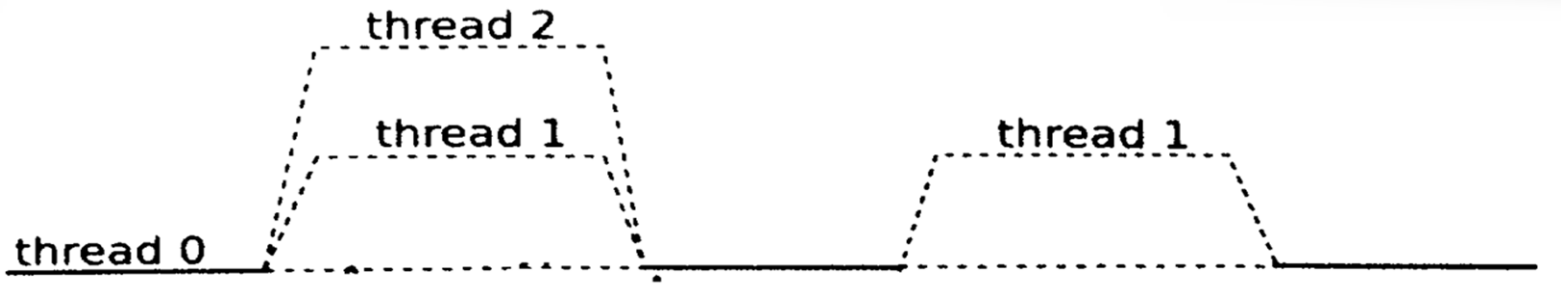
- Elle est supportée sur de nombres plateformes incluant Linux et Windows pour les langages C/C++ et Fortran

- C'est une ensemble de directives, d'une bibliothèque logicielle et de variables d'environnement



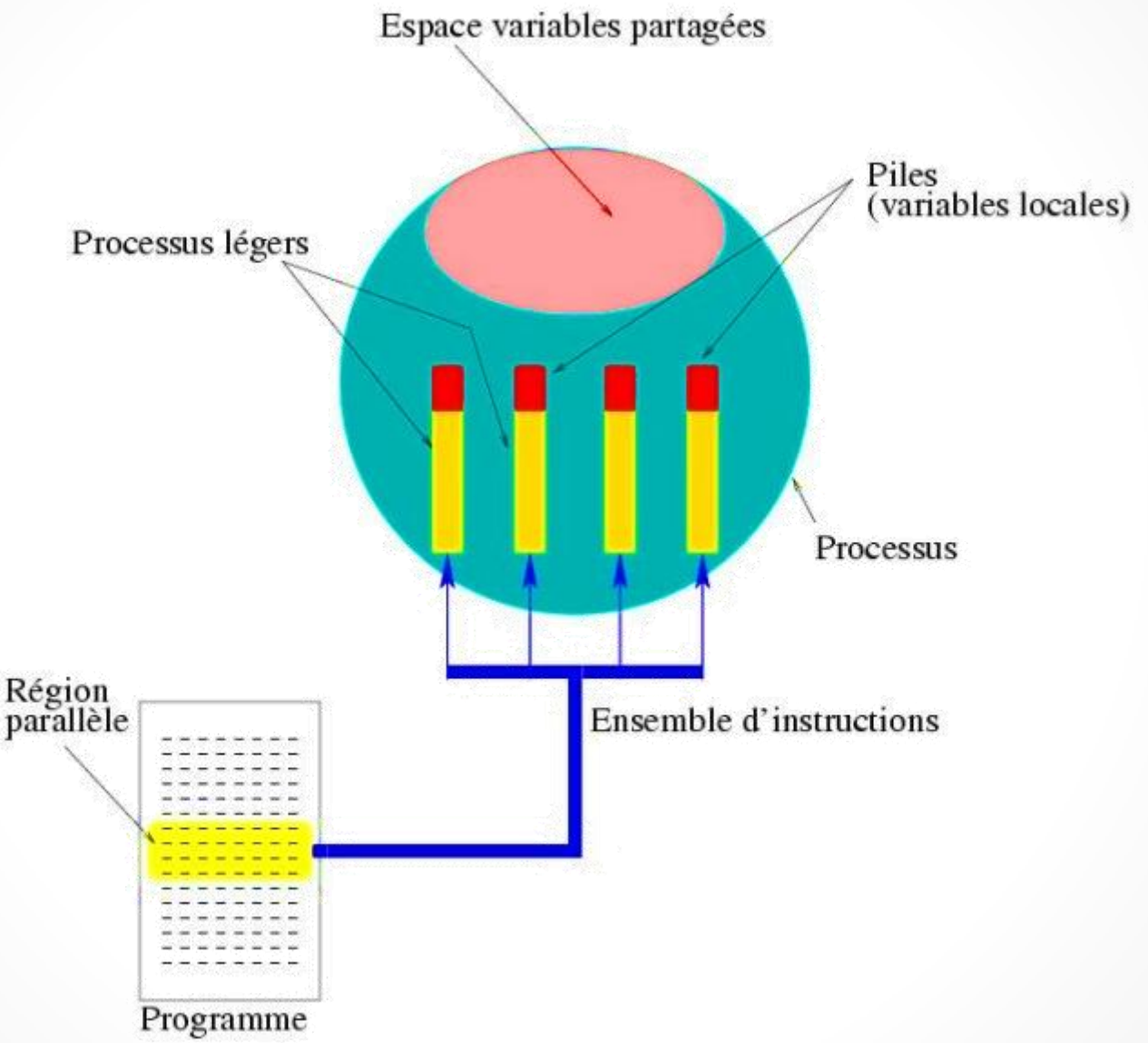
Principes de base

- Un programme OpenMP est exécuté par un processus unique
- Ce processus active des processus légers (threads) à l'entrée d'une région parallèle

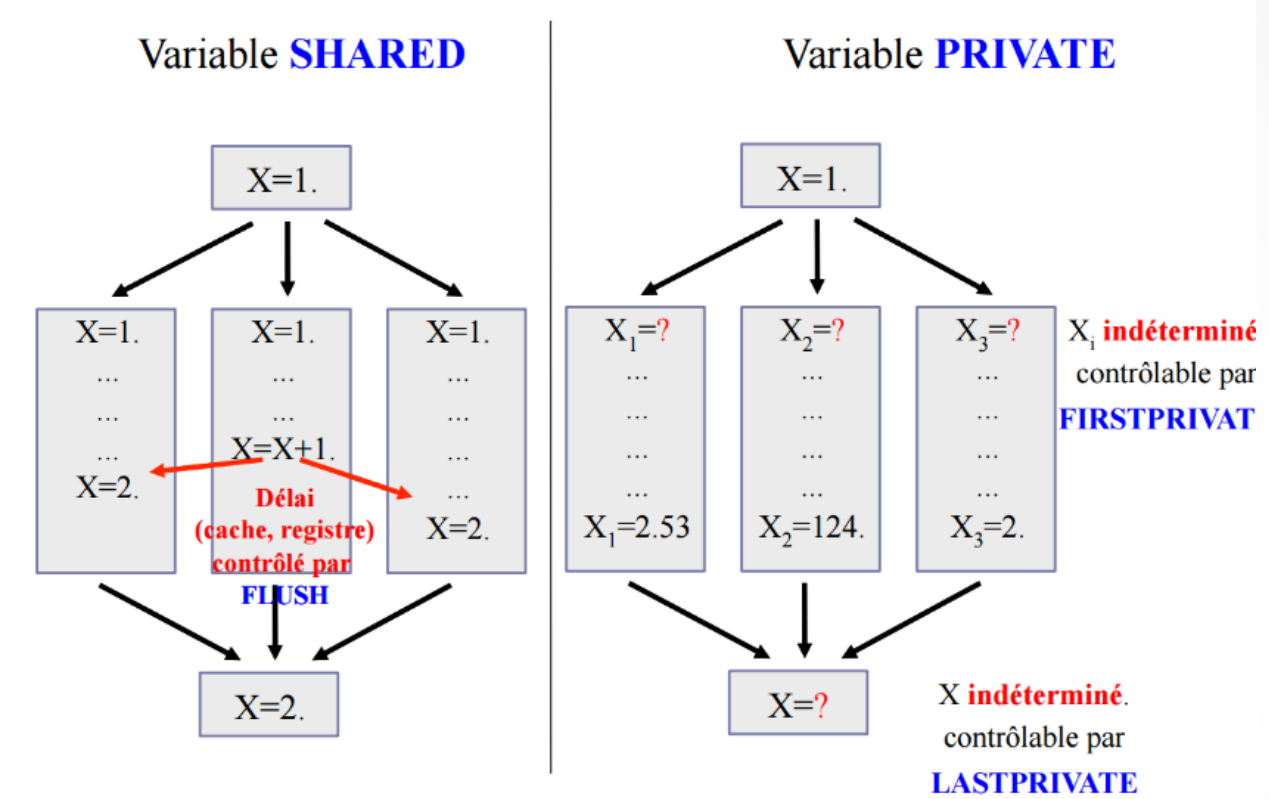


- Chaque processus léger exécute une tache composée d'un ensemble d'instructions
- Pendant l'exécution d'une tache, une variable peut être lue et/ou modifiée en mémoire
  - Elle peut être défini dans la pile(stack)(espace mémoire local) d'un processus léger; on parle ici de variable privée
  - Elle peut être définie dans un espace mémoire partagé
- Le programmeur peut choisir si une variable est privée ou partagée
- La déclaration des zones parallèles se fait à l'aide des directives OpenMP

Principes de base



Principes de base



Les variables du code source séquentiel original peuvent être partagées(*shared*) ou privées (*private*) en OpenMP.

- **Variable partagée :** chaque thread accède à la même et unique variable originale : C'est le status par défaut
- **Variable privée :** chaque thread a sa propre copie locale de la variante originale C'est le status de toute variable déclarée à l'intérieur d'une zone parallèle

Compilation et exécution d'un programme OpenMP

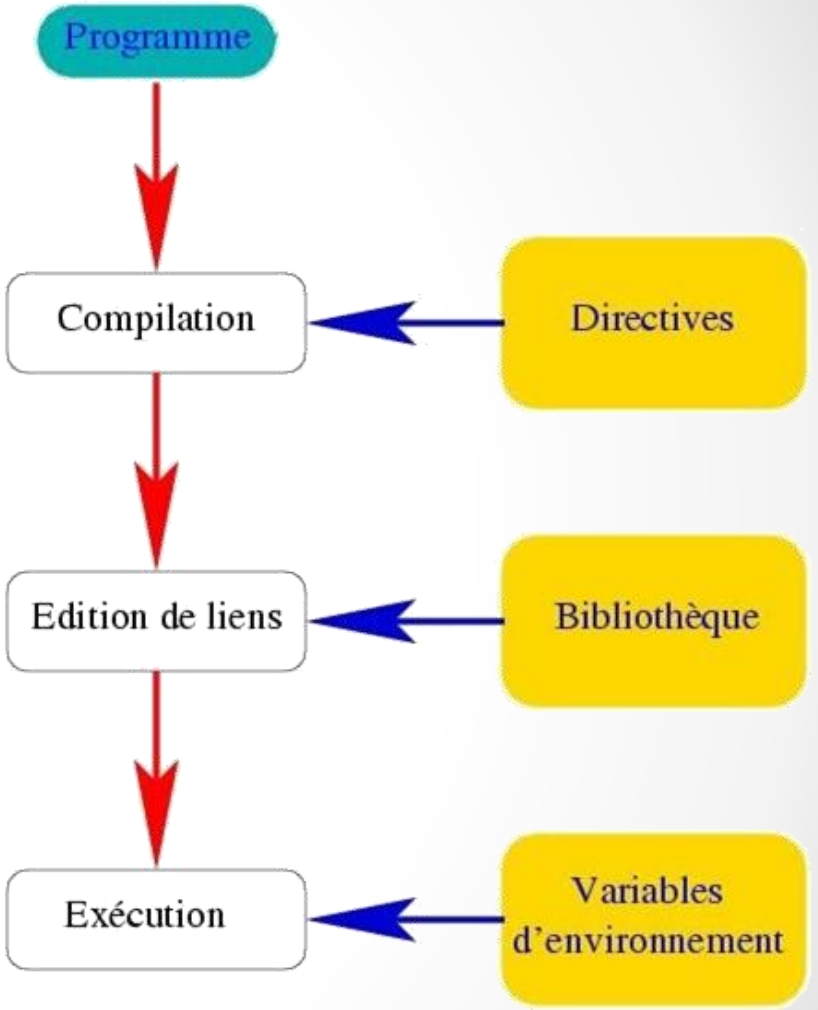
- Compilation : les directives de compilation (#pragma en c et c++, commentaires en Fortran) sont interprétées si le compilateur les reconnait. Dans le cas contraire, elles sont assimilées à des commentaires. Les directives indiquent au compilateur comment paralléliser le code

- Edition de liens : bibliothèques particulières OpenMP

- Variables d'environnement : une fois positionnées, leurs valeurs sont prises en compte à l'exécution

- La compilation se réalise avec :
  `gcc -fopenmp prog.c` ....

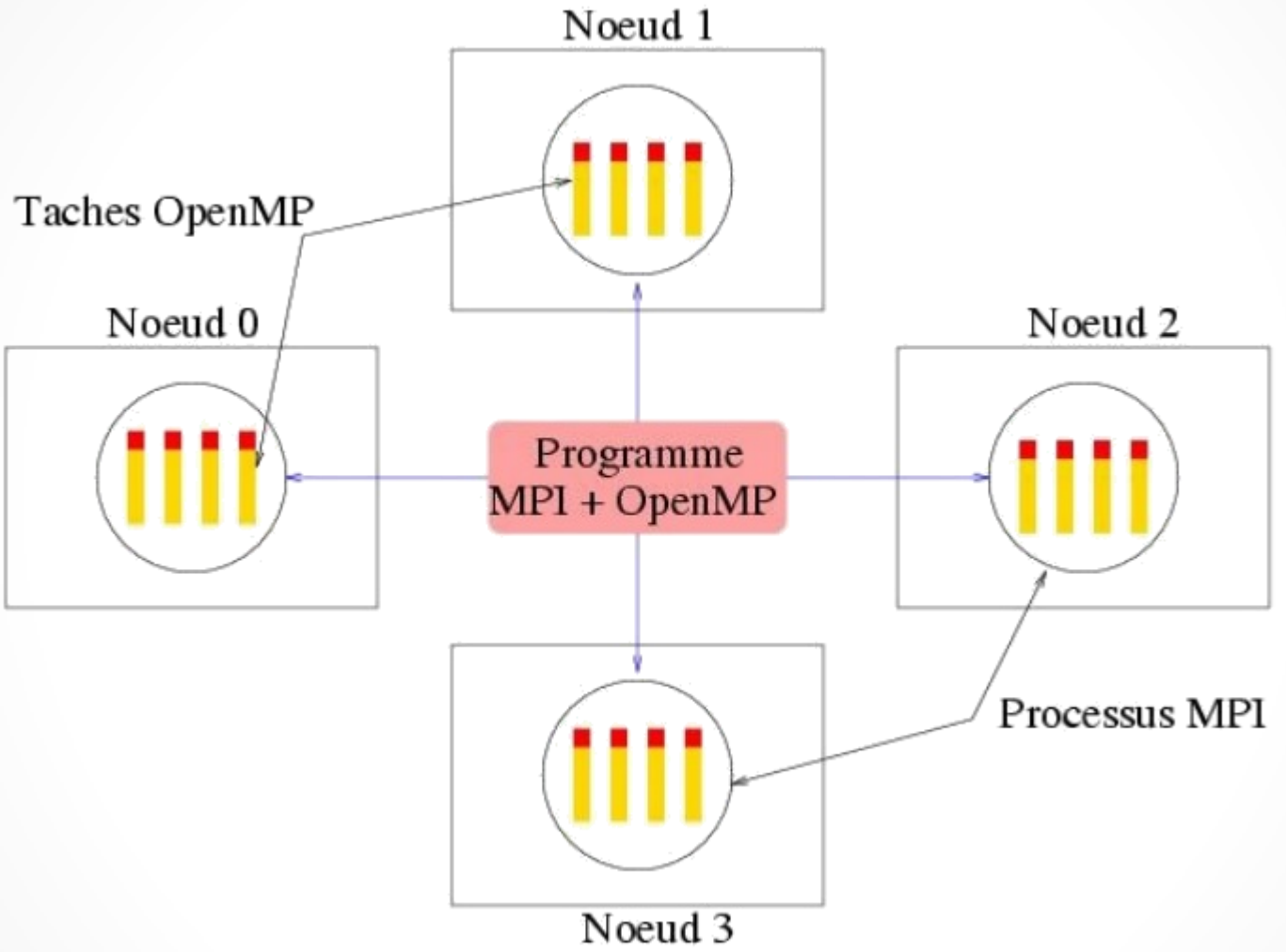
- L'exécutable s'exécute comme tout exécutable sous Unix/Linux : `./executable`



OpenMP VS MPI

- OpenMP comme MPI possède une interface Fortan, C et C++
- MPI est un modèle multiprocessus dont le mode de communication entre les processus est explicite (la gestion des communications est prise à la charge de l'utilisateur)
- OpenMP est un modèle multitâches dont le mode de communication entre les processus est implicite (la gestion des communications est prise à la charge de compilateur)
- MPI est utilisé en général sur des machines multiprocesseurs à mémoire distribuée
- OpenMP est utilisé sur des machines multiprocesseurs à mémoire partagée
- Sur une matrice de machines indépendantes (nœuds) multiprocesseurs à mémoire partagée, la mise en œuvre d'une parallélisation à deux niveaux(OpenMP et MPI) dans un même programme peut être un atout majeur pour des performances parallèles du code

OpenMP VS MPI



Les directives OpenMP

Elles sont délimitées par une sentinelle :

```
#pragma omp directive [clause]*[clause]
```

Par défaut, il y a une barrière de synchronisation à la fin

Utilisation des directives OpenMP :

- Débranchement externe interdit !
- Une seule directive par sentinelle
- Majuscule/minuscule importante
- Les directives sont : `parallel, for, sections, section, single, master, critical, barrier, atomic, flush, ordered, threadprivate`

Quelques directives OpenMP

#**pragma omp parallel** : section parallèle

#**pragma omp parallel for** : boucle **for** parallèle

#**pragma omp master** : section qui ne sera exécutée que par le thread principal

#**pragma omp critical** : région "critique" qui ne doit être exécutée que par un thread à la fois (doit par exemple être utilisé lorsque des threads doivent écrire dans une variable partagée ou dans un fichier)

#**pragma omp ordered :** section qui doit être exécutée dans l'ordre

#**pragma omp barrier :** barrière qui permet d'attendre que tous les threads soient arrivés à ce point avant d'exécuter la suite

Certains nombre d'options après les directives

private(var1) : **var1** sera une variable privée de la section parallèle (à noter que la valeur originale déclarée dans la partie non parallèle du programme n'est pas recopiée)

shared(var2) : **var2** sera une variable partagée de la section parallèle (ce qui est le cas par défaut de toute variable qui n'est pas déclarée private)

firstprivate(var3) : semblable à **private** sauf que cette fois la valeur privée de **var3** est initialisée à la dernière valeur de **var3** dans la partie non parallélisée qui précède

lastprivate(var4) : à la fin de la section parallèle, la valeur que prend **var4** pour le dernier thread sera affectée à **var4** dans la section non parallèle qui suit

Certains nombre d’options après les directives

ordered : la section parallèle comporte un bloc **#pragma omp ordered**

schedule(static) : option par défaut qui réparti les tâches de façon statique entre les threads

schedule(dynamic) : option qui permet de répartir les tâches de façon dynamique, particulièrement utile lorsque les temps d'exécution des tâches sont différents

reduction(op:val) : permet de "réduire" la variable **val** par l'opérateur **op** en sortie de la section parallélisée : par exemple **reduction(+:val)** permet de sommer toutes les variables privées **val** à la fin de la parallélisation.

Avantages et inconvénients respectifs d'OpenMP/MPI

- Avantages d'OpenMP :
  - plus facile à programmer / mettre au point que MPI
  - préserve le code séquentiel original
  - code plus facile à comprendre/maintenir
  - permet une parallélisation progressive
- Inconvénients d'OpenMP :
  - uniquement pour machines à mémoire (virtuellement) partagée
  - actuellement principalement adapté aux boucles parallèles
- Avantages de MPI :
  - s'exécute sur machines à mémoire partagée ou distribuée
  - peut s'appliquer à une gamme de problèmes plus larges qu'OpenMP
  - chaque processus a ses propres variables (pas de conflits)
- Inconvénients de MPI :
  - modifications algorithmiques importantes souvent nécessaires (envoi de messages, recouvrement communications/calcul)
  - peut être plus dur à mettre au point
  - la performance peut dépendre du réseau de communication utilisé