Calcul Parallèle

Semestre 4

A. BOURJIJ

bourjij5@univ-lorraine.fr

Université de Lorraine

Table des Matières

Qu'est ce que le Calcul Parallèle

Exécution séquentielle:

- Tradionnellement, les logiciels ont été écrits pour une exécution instruction après instruction,
- l'exécution est faite par un unique processeur,
- sur un jeu de données

Qu'est ce que le Calcul Parallèle

Exécution séquentielle:

- Tradionnellement, les logiciels ont été écrits pour une exécution instruction après instruction,
- l'exécution est faite par un unique processeur,
- sur un jeu de données

Qu'est ce que le Calcul Parallèle

Exécution séquentielle:

- Tradionnellement, les logiciels ont été écrits pour une exécution instruction après instruction,
- l'exécution est faite par un unique processeur,
- sur un jeu de données

Qu'est ce que le Calcul Parallèle

Exécution parallèle:

- un problème est décomposé en un ensemble fini de parties pouvant être résolues simultanément
- chaque partie est constituée d'une suite d'intructions,
- les instructions de chaque partie sont exécutées en parallèle sur des processeurs différents
- un mécanisme de contrôle/coordination gère l'ensemble.

Qu'est ce que le Calcul Parallèle

Exécution parallèle:

- un problème est décomposé en un ensemble fini de parties pouvant être résolues simultanément
- chaque partie est constituée d'une suite d'intructions,
- les instructions de chaque partie sont exécutées en parallèle sur des processeurs différents
- un mécanisme de contrôle/coordination gère l'ensemble.

Qu'est ce que le Calcul Parallèle

Exécution parallèle:

- un problème est décomposé en un ensemble fini de parties pouvant être résolues simultanément
- chaque partie est constituée d'une suite d'intructions,
- les instructions de chaque partie sont exécutées en parallèle sur des processeurs différents
- un mécanisme de contrôle/coordination gère l'ensemble.

Qu'est ce que le Calcul Parallèle

Exécution parallèle:

- un problème est décomposé en un ensemble fini de parties pouvant être résolues simultanément
- chaque partie est constituée d'une suite d'intructions,
- les instructions de chaque partie sont exécutées en parallèle sur des processeurs différents
- un mécanisme de contrôle/coordination gère l'ensemble.

Qu'est ce que le Calcul Parallèle

La classification de M.J. Flynn (1966) propose 4 catégories d'architectures d'ordinateurs selon l'organisation des flux de données et d'instructions:

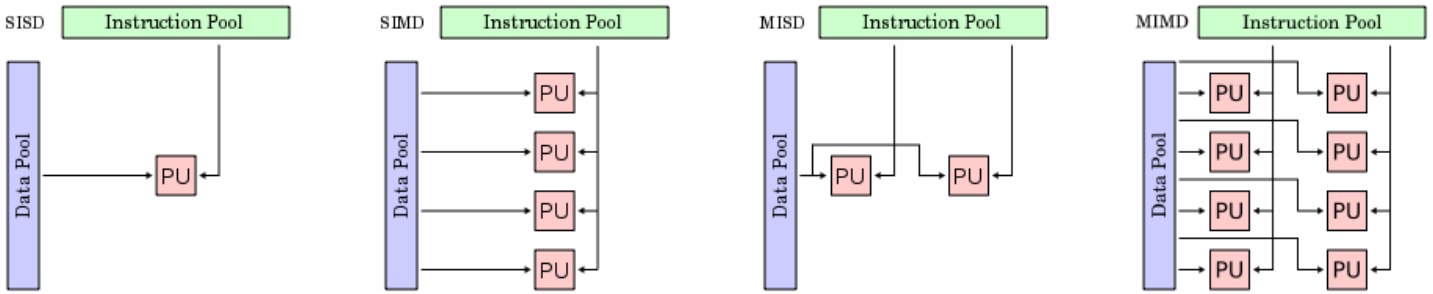
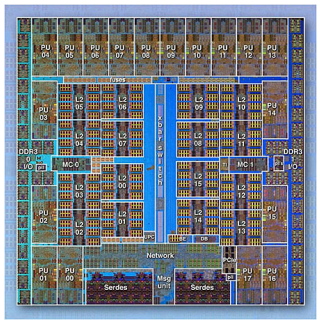


Table : De Von Neumann aux Supercalculateurs

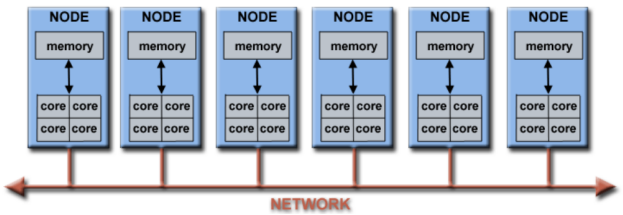
Qu'est ce que le Calcul Parallèle

Un calculateur parallèle:

- est un ordinateur composé de multiples processeurs ou coeurs, comme la carte IBM BG/Q avec 18 coeurs (PU) et 16 unités de cache (L2):



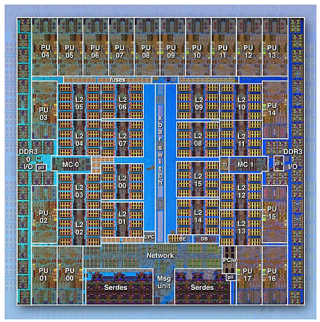
- ou un ensemble d'ordinateurs connectés par un réseau:



Qu'est ce que le Calcul Parallèle

Un calculateur parallèle:

- est un ordinateur composé de multiples processeurs ou coeurs, comme la carte IBM BG/Q avec 18 coeurs (PU) et 16 unités de cache (L2):



- ou un ensemble d'ordinateurs connectés par un réseau:

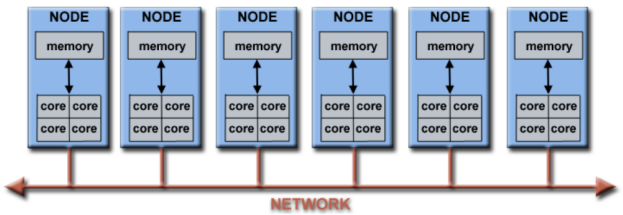
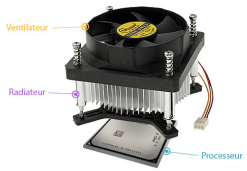


Table des Matières

Intérêts du parallèlisme

Loi de moore

- Entre 1971 et 2001, la densité des transistors a doublé chaque 1,96 année. Les machines électroniques sont devenues de moins en moins honéreuses et de plus en plus puissantes.
- Depuis 2004, la fréquence des processeurs tend à stagner (difficultés de dissipation thermique), malgré la taille de plus en plus faible des composants.

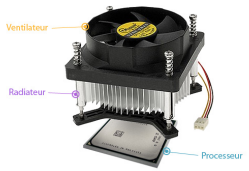


- Les fondeurs ont désormais recours aux architectures multi-cœurs.
- Le logiciel doit donc s'adapté pour tirer avantage du matériel.

Intérêts du parallèlisme

Loi de moore

- Entre 1971 et 2001, la densité des transistors a doublé chaque 1,96 année. Les machines électroniques sont devenues de moins en moins honéreuses et de plus en plus puissantes.
- Depuis 2004, la fréquence des processeurs tend à stagner (difficultés de dissipation thermique), malgré la taille de plus en plus faible des composants.

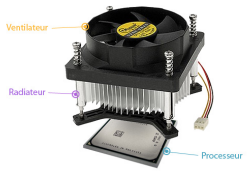


- Les fondeurs ont désormais recours aux architectures multi-cœurs.
- Le logiciel doit donc s'adapté pour tirer avantage du matériel.

Intérêts du parallèlisme

Loi de moore

- Entre 1971 et 2001, la densité des transistors a doublé chaque 1,96 année. Les machines électroniques sont devenues de moins en moins honéreuses et de plus en plus puissantes.
- Depuis 2004, la fréquence des processeurs tend à stagner (difficultés de dissipation thermique), malgré la taille de plus en plus faible des composants.

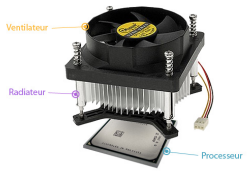


- Les fondeurs ont désormais recours aux architectures multi-cœurs.
- Le logiciel doit donc s'adapté pour tirer avantage du matériel.

Intérêts du parallèlisme

Loi de moore

- Entre 1971 et 2001, la densité des transistors a doublé chaque 1,96 année. Les machines électroniques sont devenues de moins en moins honéreuses et de plus en plus puissantes.
- Depuis 2004, la fréquence des processeurs tend à stagner (difficultés de dissipation thermique), malgré la taille de plus en plus faible des composants.



- Les fondeurs ont désormais recours aux architectures multi-cœurs.
- Le logiciel doit donc s'adapté pour tirer avantage du matériel.

Table des Matières

Evaluation du parallélisme

- Mesure de la vitesse: FLOPS
- Classification des calculateurs: TOP500
- Limites et coût du calcul parallèle

Evaluation du parallélisme

- Mesure de la vitesse: FLOPS
- Classification des calculateurs: TOP500
- Limites et coût du calcul parallèle

Evaluation du parallélisme

- Mesure de la vitesse: FLOPS
- Classification des calculateurs: TOP500
- Limites et coût du calcul parallèle

Flops

- FLOPS ou Flops ou encore Flop/s: FLoating-point Operations Per Second, est l'unité de mesure de la vitesse d'un système informatique.
- Les opérations sur les réels (virgule flottante) prennent beaucoup plus de temps de calcul que des opérations sur les nombres entiers.

Flops

- FLOPS ou Flops ou encore Flop/s: FLoating-point Operations Per Second, est l'unité de mesure de la vitesse d'un système informatique.
- Les opérations sur les réels (virgule flottante) prennent beaucoup plus de temps de calcul que des opérations sur les nombres entiers.

Flops

La plupart des microprocesseurs modernes incluent une unité de calcul en virgule flottante (floating-point unit, FPU), Le FLOP est couramment utilisé dans le domaine scientifique:

Unité	FLOPS
yottaFLOPS	10^{24}
zettaFLOPS	10^{21}
exaFLOPS	10^{18}
pétaFLOPS	10^{15}
téraFLOPS	10^{12}
gigaFLOPS	10^{9}
mégaFLOPS	10^{6}
kiloFLOPS	10^{3}

Table : Unité de performance des processeurs

Flops

- En 1964, la barre du mégaFLOPS a été franchie par le superordinateur américain Control Data 6600.
- En 1985, la barre du gigaFLOPS a été franchie par le superordinateur américain Cray-2.
- En 1997, la barre du téraFLOPS a été franchie par le superordinateur américain ASCI Red.
- En 2008, la barre du pétaFLOPS a été franchie par le superordinateur américain Roadrunner.

Flops

- En 1964, la barre du mégaFLOPS a été franchie par le superordinateur américain Control Data 6600.
- En 1985, la barre du gigaFLOPS a été franchie par le superordinateur américain Cray-2.
- En 1997, la barre du téraFLOPS a été franchie par le superordinateur américain ASCI Red.
- En 2008, la barre du pétaFLOPS a été franchie par le superordinateur américain Roadrunner.

Flops

- En 1964, la barre du mégaFLOPS a été franchie par le superordinateur américain Control Data 6600.
- En 1985, la barre du gigaFLOPS a été franchie par le superordinateur américain Cray-2.
- En 1997, la barre du téraFLOPS a été franchie par le superordinateur américain ASCI Red.
- En 2008, la barre du pétaFLOPS a été franchie par le superordinateur américain Roadrunner.

Flops

- En 1964, la barre du mégaFLOPS a été franchie par le superordinateur américain Control Data 6600.
- En 1985, la barre du gigaFLOPS a été franchie par le superordinateur américain Cray-2.
- En 1997, la barre du téraFLOPS a été franchie par le superordinateur américain ASCI Red.
- En 2008, la barre du pétaFLOPS a été franchie par le superordinateur américain Roadrunner.

Flops

- En 2013 un Ordinateur personnel peut disposer d'une puissance d'environ 200 gigaFLOPS avec un microprocesseur de puissance comparable aux superordinateurs de 1995 comme l'Intel Core i7-3770
- 5 621 gigaFLOPS avec un processeur graphique comme le NVidia GTX 690
- soit une puissance comparable aux superordinateurs de 2001.

Flops

- En 2013 un Ordinateur personnel peut disposer d'une puissance d'environ 200 gigaFLOPS avec un microprocesseur de puissance comparable aux superordinateurs de 1995 comme l'Intel Core i7-3770
- 5 621 gigaFLOPS avec un processeur graphique comme le NVidia GTX 690
- soit une puissance comparable aux superordinateurs de 2001.

Flops

- En 2013 un Ordinateur personnel peut disposer d'une puissance d'environ 200 gigaFLOPS avec un microprocesseur de puissance comparable aux superordinateurs de 1995 comme l'Intel Core i7-3770
- 5 621 gigaFLOPS avec un processeur graphique comme le NVidia GTX 690
- soit une puissance comparable aux superordinateurs de 2001.

TOP500

Le TOP500 est un projet de classification des 500 supercalculateurs les plus puissants au monde. Le logiciel de test est Linpack, Il effectue ses calculs sur des nombres d'une longueur de 64 bits.

Nom	Processeurs	Vendeur	Site	Coeurs	Système
Tianhe-2	Xeon + Xeon phi	NUDT	National Supercomputing chine 2013	3 120 000	Linux (Kylin)
Titan	Opteron +Nvidia	Cray Inc	Oak Ridge Nat. Lab. USA 2012	560 000	Linux (CLES, SLES)
Sequoia	PowerPC A2, Custom	Cray Inc	Lawrence Livermore Nat. Lab. USA 2013	1 572 864	Linux (RHEL et CNK)
K computer	Sparc64, Tofu	Fujitsu	Riken Japon 2011	705 024	Linux
Mira	PowerPC A2, Custom	IBM	Argonne Nat. Lab. USA 2013	786 432	Linux (RHEL et CNK)
Piz Daint	Xeon +Nvidia	Cray Inc	EPF Zurich SUISSE 2013	115 984	Linux (CLES)
Stampede	Xeon + Xeon phi	Dell	Texas Advanced Computing Center USA 2013	462 462	Linux
Juqueen	PowerPC A2, Custom	IBM	Centre Recherche de Jülich Allemagne 2013	458 752	Linux (RHEL et CNK)
Vulcan	PowerPC A2, Custom	IBM	Lawrence Livermore Nat. Lab. USA 2013	393 216	Linux (RHEL et CNK)
Cray CS-Storm	Xeon +Nvidia	Cray Inc	Gouv. Fédéral USA 2014	72 800	Linux (CLES)

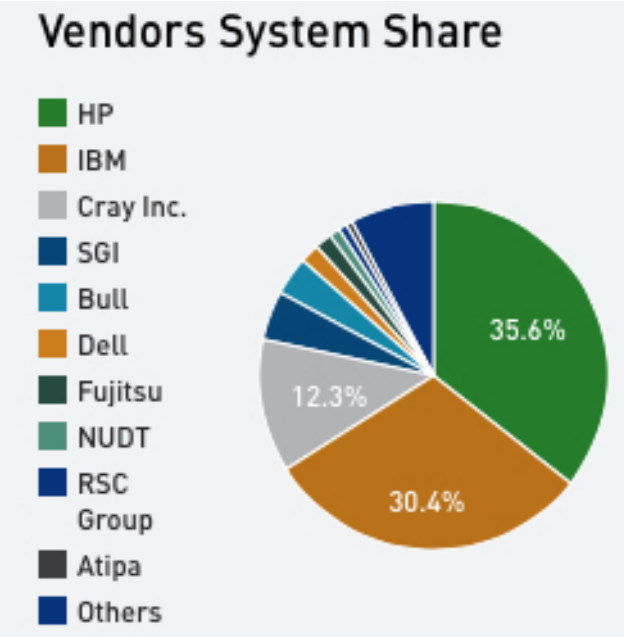
Table : Top 10 du 43ième TOP500 nov 2014

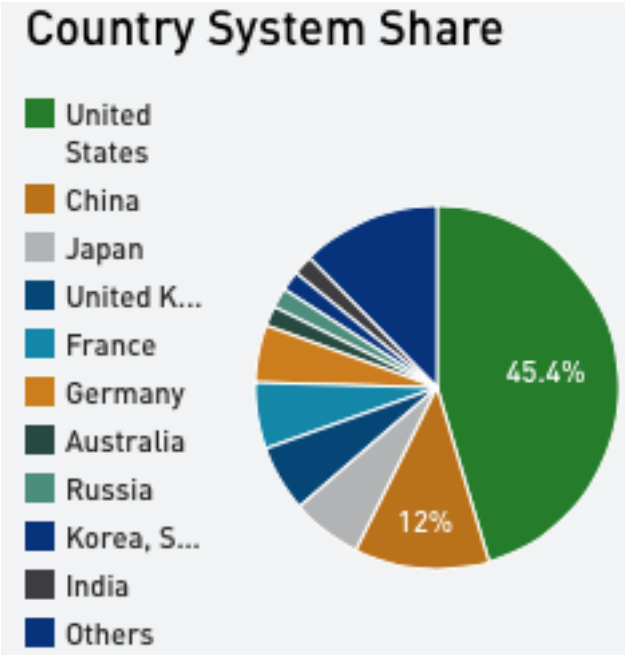
TOP500

Les géants de 2014: Tianhe-2, Titan et Sequoia

TOP500

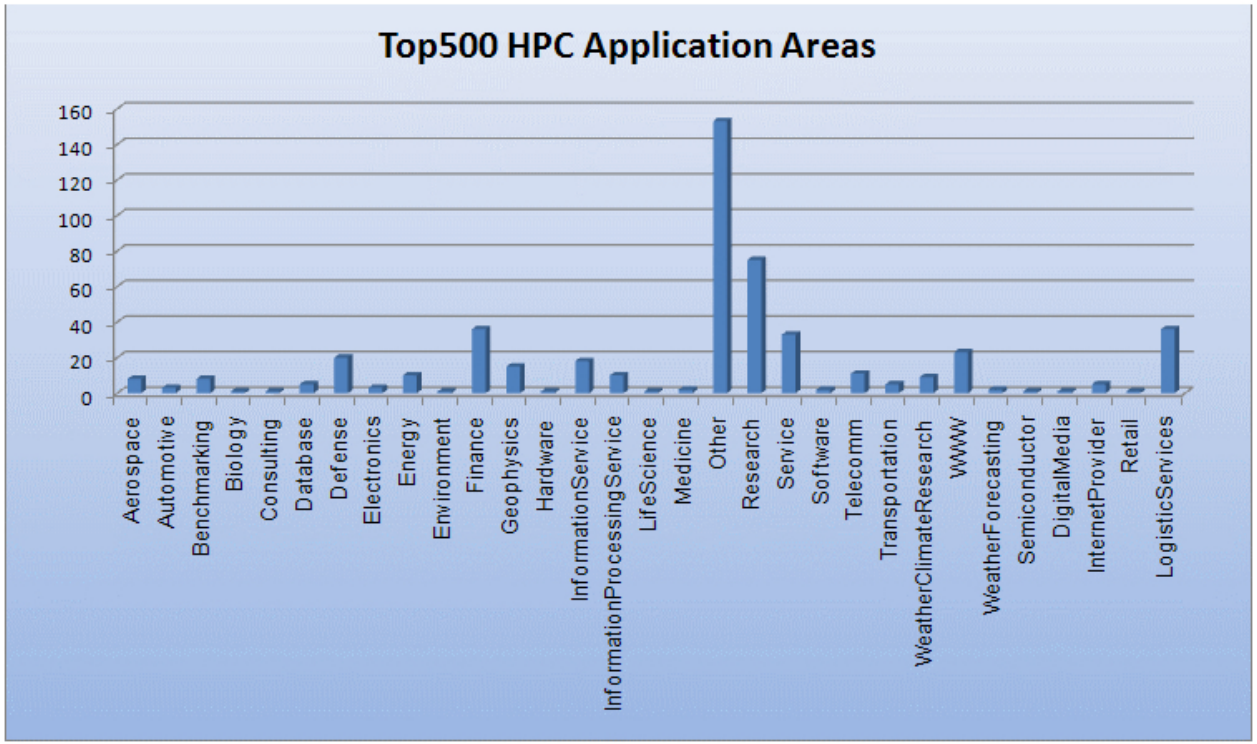
La majorité des superordinateurs sont des clusters produits par les grandes firmes d'informatique:





TOP500

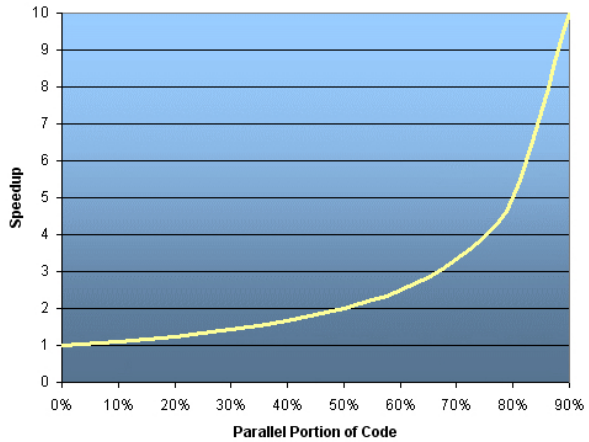
Domaines d'utilisation des superordinateurs:



Limites et coût du calcul parallèle

La loi d'Amdahl dit que la possibilité d'accération d'un programme est définie par la fraction de code (P) pouvant être parallélisée: $\boxed{Acc = \frac{1}{1 - P}}$

(si P=0 ==>Acc=1 pas d'accélération , si 50% du code est // Acc=2, si tout le code est // Acc est infinie)



Limites et coût du calcul parallèle

La prise en compte du nombre de processeurs exécutant la portion de code parallèle, donne la relation suivante:

$$Acc = \frac{1}{\frac{P}{N} + S}$$

où P=fraction parallèle, N= nombre de processeurs et S=fraction série.

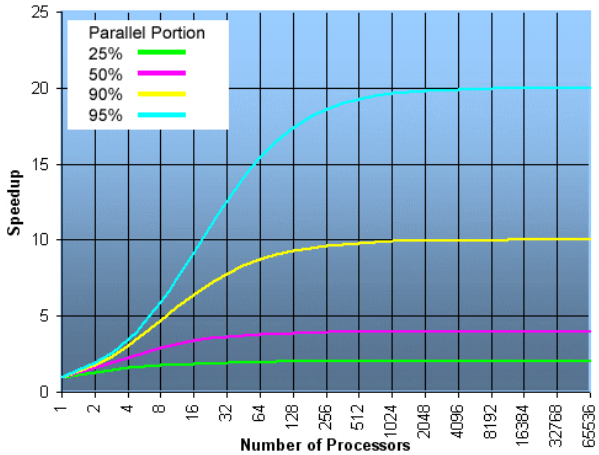


Table des Matières

Modèles de programmation parallèle

- Les Threads POSIX
- OpenMP
- OpenMPI

Modèles de programmation parallèle

- Les Threads POSIX
- OpenMP
- OpenMPI

Modèles de programmation parallèle

- Les Threads POSIX
- OpenMP
- OpenMPI

Les Threads POSIX

Un thread est un processus leger ou fil d'exécution:

- C'est une procédure qui s'exécute indépendamment de son programme principal,
- Il dispose de moyens de communication plus simples,
- La création d'un thread est de 10 à 100 fois plus rapide que celle d'un processus,
- Tous les threads d'un même processus partagent le même espace d'adressage et sont ordonnancés par l'OS pour s'exécuter de façon concurrente.

Les Threads POSIX

Un thread est un processus leger ou fil d'exécution:

- C'est une procédure qui s'exécute indépendamment de son programme principal,
- Il dispose de moyens de communication plus simples,
- La création d'un thread est de 10 à 100 fois plus rapide que celle d'un processus,
- Tous les threads d'un même processus partagent le même espace d'adressage et sont ordonnancés par l'OS pour s'exécuter de façon concurrente.

Les Threads POSIX

Un thread est un processus leger ou fil d'exécution:

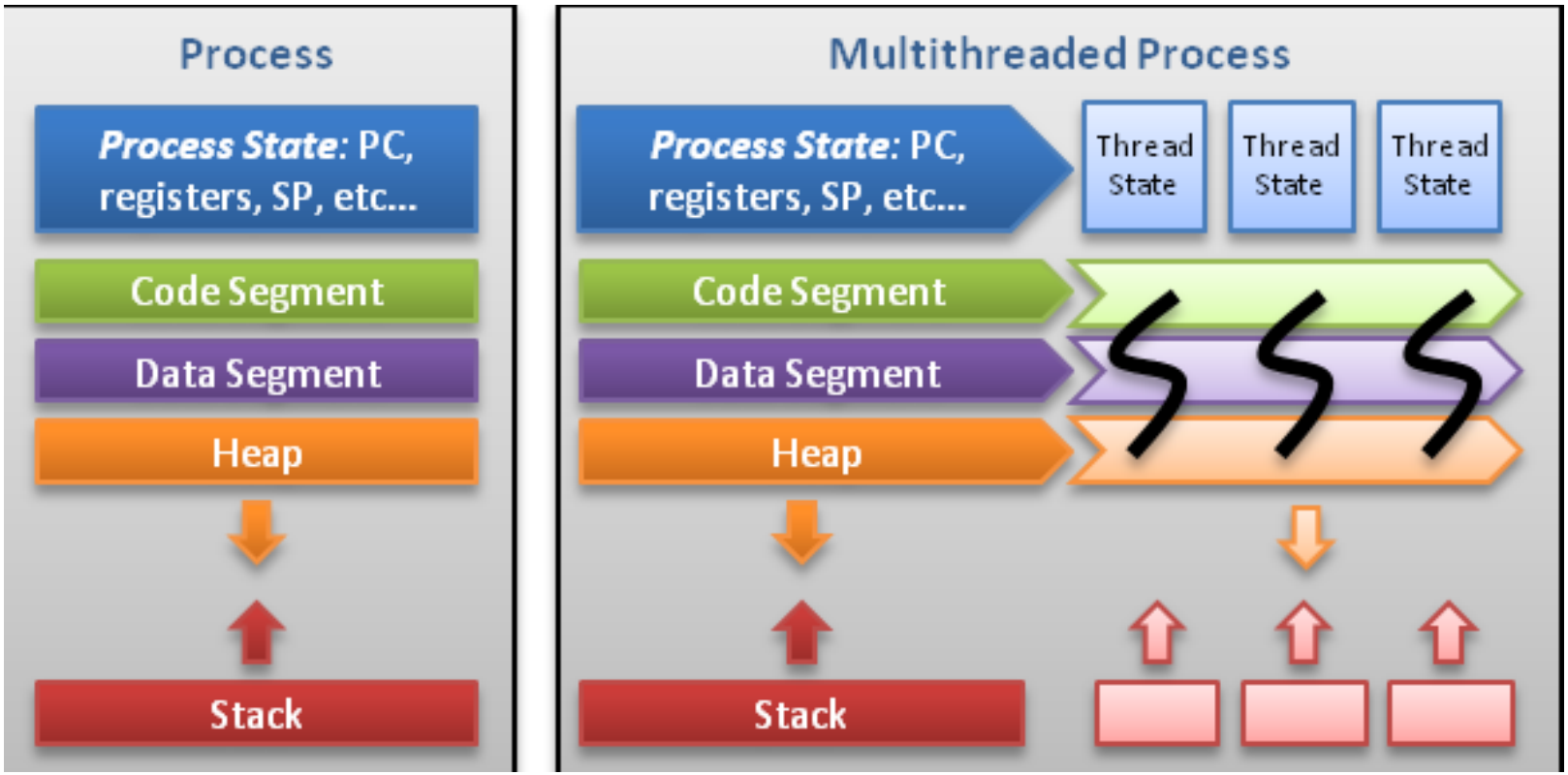
- C'est une procédure qui s'exécute indépendamment de son programme principal,
- Il dispose de moyens de communication plus simples,
- La création d'un thread est de 10 à 100 fois plus rapide que celle d'un processus,
- Tous les threads d'un même processus partagent le même espace d'adressage et sont ordonnancés par l'OS pour s'exécuter de façon concurrente.

Les Threads POSIX

Un thread est un processus leger ou fil d'exécution:

- C'est une procédure qui s'exécute indépendamment de son programme principal,
- Il dispose de moyens de communication plus simples,
- La création d'un thread est de 10 à 100 fois plus rapide que celle d'un processus,
- Tous les threads d'un même processus partagent le même espace d'adressage et sont ordonnancés par l'OS pour s'exécuter de façon concurrente.

Les Threads POSIX



Création et Terminaison de Threads

Chaque appel à la routine pthread_create(...), crée un thread et le rend exécutable, le nombre maximum de threads pouvant être créé par un processus dépend de l'implémentation.
Une fois créés les threads sont égaux, il n'y a pas de hiérarchie entre threads.
Routines:

pthread_create(thread, attr, fonction, arg)
pthread_exit(status)
pthread_cancel(thread)
pthread_attr_init(attr)
pthread_attr_destroy(attr)

(linux: ulimit -a)

Passage d'arguments aux Threads

- La routine pthread_create(...) permet le passage d'un argument à la routine lancée.
- Pour pouvoir passer plusieurs arguments, il suffit de créer une structure contenant tous les arguments.
- Tous les arguments doivent être passés par référence et castés à (void *).

Passage d'arguments aux Threads

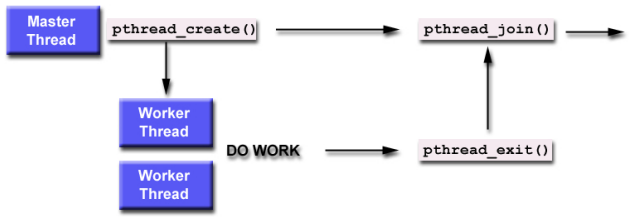
- La routine pthread_create(...) permet le passage d'un argument à la routine lancée.
- Pour pouvoir passer plusieurs arguments, il suffit de créer une structure contenant tous les arguments.
- Tous les arguments doivent être passés par référence et castés à (void *).

Passage d'arguments aux Threads

- La routine pthread_create(...) permet le passage d'un argument à la routine lancée.
- Pour pouvoir passer plusieurs arguments, il suffit de créer une structure contenant tous les arguments.
- Tous les arguments doivent être passés par référence et castés à (void *).

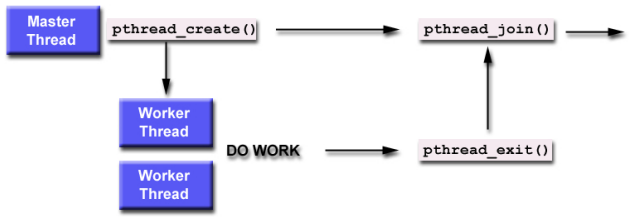
Synchronisation des Threads

- La routine pthread_join(threadid, status) bloque le thread appelant jusqu'à ce que le thread spécifié par threadid se termine.
- Le programmeur peut récupérer l'état (status) du thread cible, grâce à l'appel à pthread_exit().
- Une terminaison de thread ne peut correspondre qu'à un seul appel à pthread_join().
- Il existe deux autres moyens de synchroniser les threads: les mutex et les variables de condition.



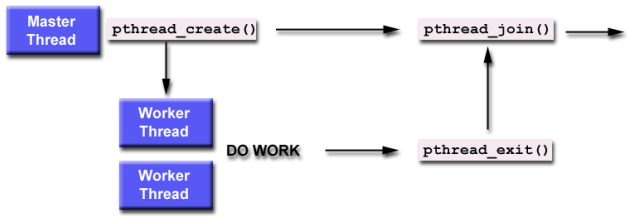
Synchronisation des Threads

- La routine pthread_join(threadid, status) bloque le thread appelant jusqu'à ce que le thread spécifié par threadid se termine.
- Le programmeur peut récupérer l'état (status) du thread cible, grâce à l'appel à pthread_exit().
- Une terminaison de thread ne peut correspondre qu'à un seul appel à pthread_join().
- Il existe deux autres moyens de synchroniser les threads: les mutex et les variables de condition.



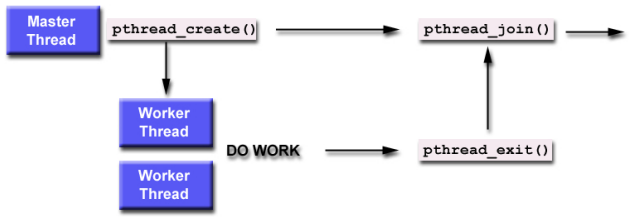
Synchronisation des Threads

- La routine pthread_join(threadid, status) bloque le thread appelant jusqu'à ce que le thread spécifié par threadid se termine.
- Le programmeur peut récupérer l'état (status) du thread cible, grâce à l'appel à pthread_exit().
- Une terminaison de thread ne peut correspondre qu'à un seul appel à pthread_join().
- Il existe deux autres moyens de synchroniser les threads: les mutex et les variables de condition.



Synchronisation des Threads

- La routine pthread_join(threadid, status) bloque le thread appelant jusqu'à ce que le thread spécifié par threadid se termine.
- Le programmeur peut récupérer l'état (status) du thread cible, grâce à l'appel à pthread_exit().
- Une terminaison de thread ne peut correspondre qu'à un seul appel à pthread_join().
- Il existe deux autres moyens de synchroniser les threads: les mutex et les variables de condition.



Les Mutex: généralités

- Abréviation de "exclusion mutuelle", les mutex sont le principal outil de synchronisation de threads et de protection des données lors d'écritures multiples.
- Une variable mutex agit comme un verrou permettant de protéger l'accés à une ressource partagée.
- Lorsque plusieurs threads essaient de verrouiller une variable mutex, seul l'un d'entre eux peut le faire. Obligeant les autres à attendre leur tour.

Les Mutex: généralités

- Abréviation de "exclusion mutuelle", les mutex sont le principal outil de synchronisation de threads et de protection des données lors d'écritures multiples.
- Une variable mutex agit comme un verrou permettant de protéger l'accés à une ressource partagée.
- Lorsque plusieurs threads essaient de verrouiller une variable mutex, seul l'un d'entre eux peut le faire. Obligeant les autres à attendre leur tour.

Les Mutex: généralités

- Abréviation de "exclusion mutuelle", les mutex sont le principal outil de synchronisation de threads et de protection des données lors d'écritures multiples.
- Une variable mutex agit comme un verrou permettant de protéger l'accés à une ressource partagée.
- Lorsque plusieurs threads essaient de verrouiller une variable mutex, seul l'un d'entre eux peut le faire. Obligeant les autres à attendre leur tour.

Les Mutex: exemple

Malgré des tests adaptés le parallélisme peut engendrer des erreurs.
Soit deux threads exécutant une même procédure de réservation de billets.
Au départ resa=5 et NPT=10

code	**Thread1**: dem=3	**Thread2**: dem=5
si (resa+dem <= NPT) Alors	vrai	vrai
dem <– dem+resa	8	13
Fsi		

Les Mutex: Séquence typique

- Création et initialisation d'une variable mutex,
- Plusieurs threads cherchent à verrouiller le mutex,
- Seul l'un d'entre eux y arrive: il en est propriétaire,
- Le propriétaire peut exécuter des actions,
- Le propriétaire déverrouille le mutex,
- Un autre thread peut verrouiller à son tour le mutex et recommencer le processus,
- Finalement le mutex est détruit.

Les Mutex: Séquence typique

- Création et initialisation d'une variable mutex,
- Plusieurs threads cherchent à verrouiller le mutex,
- Seul l'un d'entre eux y arrive: il en est propriétaire,
- Le propriétaire peut exécuter des actions,
- Le propriétaire déverrouille le mutex,
- Un autre thread peut verrouiller à son tour le mutex et recommencer le processus,
- Finalement le mutex est détruit.

Les Mutex: Séquence typique

- Création et initialisation d'une variable mutex,
- Plusieurs threads cherchent à verrouiller le mutex,
- Seul l'un d'entre eux y arrive: il en est propriétaire,
- Le propriétaire peut exécuter des actions,
- Le propriétaire déverrouille le mutex,
- Un autre thread peut verrouiller à son tour le mutex et recommencer le processus,
- Finalement le mutex est détruit.

Les Mutex: Séquence typique

- Création et initialisation d'une variable mutex,
- Plusieurs threads cherchent à verrouiller le mutex,
- Seul l'un d'entre eux y arrive: il en est propriétaire,
- Le propriétaire peut exécuter des actions,
- Le propriétaire déverrouille le mutex,
- Un autre thread peut verrouiller à son tour le mutex et recommencer le processus,
- Finalement le mutex est détruit.

Les Mutex: Séquence typique

- Création et initialisation d'une variable mutex,
- Plusieurs threads cherchent à verrouiller le mutex,
- Seul l'un d'entre eux y arrive: il en est propriétaire,
- Le propriétaire peut exécuter des actions,
- Le propriétaire déverrouille le mutex,
- Un autre thread peut verrouiller à son tour le mutex et recommencer le processus,
- Finalement le mutex est détruit.

Les Mutex: Séquence typique

- Création et initialisation d'une variable mutex,
- Plusieurs threads cherchent à verrouiller le mutex,
- Seul l'un d'entre eux y arrive: il en est propriétaire,
- Le propriétaire peut exécuter des actions,
- Le propriétaire déverrouille le mutex,
- Un autre thread peut verrouiller à son tour le mutex et recommencer le processus,
- Finalement le mutex est détruit.

Les Mutex: Séquence typique

- Création et initialisation d'une variable mutex,
- Plusieurs threads cherchent à verrouiller le mutex,
- Seul l'un d'entre eux y arrive: il en est propriétaire,
- Le propriétaire peut exécuter des actions,
- Le propriétaire déverrouille le mutex,
- Un autre thread peut verrouiller à son tour le mutex et recommencer le processus,
- Finalement le mutex est détruit.

Les Mutex: création et destruction

Les routines:

pthread_mutex_init (mutex,attr)
pthread_mutex_destroy (mutex)
pthread_mutexattr_init (attr)
pthread_mutexattr_destroy (attr)

Utilisation:

Les variables mutex doivent être déclarées de type *pthread_mutex_t* et initialisées avant utilisation.

Soit statiquement, exemple: pthread_mutex_t monmutex = PTHREAD_MUTEX_INITIALIZER;

Soit dynamiquement avec la routine pthread_mutex_init (mutex,attr) qui permet de fixer les attributs.

Par défaut le mutex est initialement déverrouillé.

Les Mutex: vérrouillage/déverrouillage

Les routines:

pthread_mutex_lock (mutex)

permet à un thread de verrouiller une variable mutex. Si le mutex est déjà verrouillé par un autre thread, l'appel bloque le thread appelant jusqu'au deverrouillage du mutex.

pthread_mutex_trylock (mutex)

(permet d'éviter les bloquages)

pthread_mutexattr_unlock (attr)

une erreur est retournée si le mutex est déjà déverrouillé ou si le mutex est propriété d'un autre thread.

OpenMP: Introduction

OpenMP est une API de programmation parallèle (multithreading et mémoire partagée) définie par un consortium de constructeurs et éditeurs informatique.
Son but est de fournir un standard, simple d'utilisation et portable grâce aux langages d'implémentation (C/C++ et Fortran).
Elle est composée de :

- Directives de compilation,
- Bibliothèque de fonctions
- Variables d'environnements

OpenMP: Introduction

OpenMP est une API de programmation parallèle (multithreading et mémoire partagée) définie par un consortium de constructeurs et éditeurs informatique.
Son but est de fournir un standard, simple d'utilisation et portable grâce aux langages d'implémentation (C/C++ et Fortran).
Elle est composée de :

- Directives de compilation,
- Bibliothèque de fonctions
- Variables d'environnements

OpenMP: Introduction

OpenMP est une API de programmation parallèle (multithreading et mémoire partagée) définie par un consortium de constructeurs et éditeurs informatique.
Son but est de fournir un standard, simple d'utilisation et portable grâce aux langages d'implémentation (C/C++ et Fortran).
Elle est composée de :

- Directives de compilation,
- Bibliothèque de fonctions
- Variables d'environnements

Directives de compilation

Elles permettent de:

- Définir une région parallèle,
- Répartir des blocs de code entre threads,
- Distribuer les différentes itérations d'une boucle entre threads
- Sérialiser des sections de code,
- Synchroniser les tâches des threads

Les directives respectent la syntaxe suivante:
sentinelle **directive** **[clause, ...]**
Exemple C/C++: #pragma omp parallel default(shared) private(beta,pi)

Directives de compilation

Elles permettent de:

- Définir une région parallèle,
- Répartir des blocs de code entre threads,
- Distribuer les différentes itérations d'une boucle entre threads
- Sérialiser des sections de code,
- Synchroniser les tâches des threads

Les directives respectent la syntaxe suivante:
sentinelle **directive** **[clause, ...]**
Exemple C/C++: #pragma omp parallel default(shared) private(beta,pi)

Directives de compilation

Elles permettent de:

- Définir une région parallèle,
- Répartir des blocs de code entre threads,
- Distribuer les différentes itérations d'une boucle entre threads
- Sérialiser des sections de code,
- Synchroniser les tâches des threads

Les directives respectent la syntaxe suivante:
sentinelle **directive** **[clause, ...]**
Exemple C/C++: #pragma omp parallel default(shared) private(beta,pi)

Directives de compilation

Elles permettent de:

- Définir une région parallèle,
- Répartir des blocs de code entre threads,
- Distribuer les différentes itérations d'une boucle entre threads
- Sérialiser des sections de code,
- Synchroniser les tâches des threads

Les directives respectent la syntaxe suivante:
sentinelle **directive** **[clause, ...]**
Exemple C/C++: #pragma omp parallel default(shared) private(beta,pi)

Directives de compilation

Elles permettent de:

- Définir une région parallèle,
- Répartir des blocs de code entre threads,
- Distribuer les différentes itérations d'une boucle entre threads
- Sérialiser des sections de code,
- Synchroniser les tâches des threads

Les directives respectent la syntaxe suivante:
sentinelle **directive** **[clause, ...]**
Exemple C/C++: #pragma omp parallel default(shared) private(beta,pi)

Bibliothèque de fonctions

L'API OpenMP comprend un nombre toujours croissant de bibliothèque de fonctions. Elles permettent de :

- Fixer ou obtenir le nombre de threads,
- Obtenir l'identificateur d'un thread (Thread ID) ou sa taille,
- Fixer ou obtenir les caractéristiques des threads
- Savoir si on est dans une région parallèle et à quel niveau,
- Fixer, initialiser et terminer les verrous,
- Obtenir l'heure.

En C/C++ toutes les fonctions s'utilisent ainsi:

```
#include <omp.h>
int omp_get_num_threads(void)
```

Bibliothèque de fonctions

L'API OpenMP comprend un nombre toujours croissant de bibliothèque de fonctions. Elles permettent de :

- Fixer ou obtenir le nombre de threads,
- Obtenir l'identificateur d'un thread (Thread ID) ou sa taille,
- Fixer ou obtenir les caractéristiques des threads
- Savoir si on est dans une région parallèle et à quel niveau,
- Fixer, initialiser et terminer les verrous,
- Obtenir l'heure.

En C/C++ toutes les fonctions s'utilisent ainsi:

```
#include <omp.h>
int omp_get_num_threads(void)
```

Bibliothèque de fonctions

L'API OpenMP comprend un nombre toujours croissant de bibliothèque de fonctions. Elles permettent de :

- Fixer ou obtenir le nombre de threads,
- Obtenir l'identificateur d'un thread (Thread ID) ou sa taille,
- Fixer ou obtenir les caractéristiques des threads
- Savoir si on est dans une région parallèle et à quel niveau,
- Fixer, initialiser et terminer les verrous,
- Obtenir l'heure.

En C/C++ toutes les fonctions s'utilisent ainsi:

```
#include <omp.h>
int omp_get_num_threads(void)
```

Bibliothèque de fonctions

L'API OpenMP comprend un nombre toujours croissant de bibliothèque de fonctions. Elles permettent de :

- Fixer ou obtenir le nombre de threads,
- Obtenir l'identificateur d'un thread (Thread ID) ou sa taille,
- Fixer ou obtenir les caractéristiques des threads
- Savoir si on est dans une région parallèle et à quel niveau,
- Fixer, initialiser et terminer les verrous,
- Obtenir l'heure.

En C/C++ toutes les fonctions s'utilisent ainsi:

```
#include <omp.h>
int omp_get_num_threads(void)
```

Bibliothèque de fonctions

L'API OpenMP comprend un nombre toujours croissant de bibliothèque de fonctions. Elles permettent de :

- Fixer ou obtenir le nombre de threads,
- Obtenir l'identificateur d'un thread (Thread ID) ou sa taille,
- Fixer ou obtenir les caractéristiques des threads
- Savoir si on est dans une région parallèle et à quel niveau,
- Fixer, initialiser et terminer les verrous,
- Obtenir l'heure.

En C/C++ toutes les fonctions s'utilisent ainsi:

```
#include <omp.h>
int omp_get_num_threads(void)
```

Bibliothèque de fonctions

L'API OpenMP comprend un nombre toujours croissant de bibliothèque de fonctions. Elles permettent de :

- Fixer ou obtenir le nombre de threads,
- Obtenir l'identificateur d'un thread (Thread ID) ou sa taille,
- Fixer ou obtenir les caractéristiques des threads
- Savoir si on est dans une région parallèle et à quel niveau,
- Fixer, initialiser et terminer les verrous,
- Obtenir l'heure.

En C/C++ toutes les fonctions s'utilisent ainsi:

```
#include <omp.h>
int omp_get_num_threads(void)
```

Variables d'environnements

OpenMP fournit plusieurs variables d'environnement pour contrôler l'exécution du code parallèle . Elles permettent de :

- Fixer le nombre de threads,
- Définir la division des itérations,
- Lier les threads aux processeurs,
- Activer/désativer le parallélisme imbriqué, fixer le niveau maximum d'imbrication,
- Activer/désactiver les threads dynamiques,
- Fixer la taille de la pile des threads,
- Régler la politique d'attente des threads.

Exemple sh/bash:
export OMP_NUM_THREADS=8

Variables d'environnements

OpenMP fournit plusieurs variables d'environnement pour contrôler l'exécution du code parallèle . Elles permettent de :

- Fixer le nombre de threads,
- Définir la division des itérations,
- Lier les threads aux processeurs,
- Activer/désativer le parallélisme imbriqué, fixer le niveau maximum d'imbrication,
- Activer/désactiver les threads dynamiques,
- Fixer la taille de la pile des threads,
- Régler la politique d'attente des threads.

Exemple sh/bash:
export OMP_NUM_THREADS=8

Variables d'environnements

OpenMP fournit plusieurs variables d'environnement pour contrôler l'exécution du code parallèle . Elles permettent de :

- Fixer le nombre de threads,
- Définir la division des itérations,
- Lier les threads aux processeurs,
- Activer/désativer le parallélisme imbriqué, fixer le niveau maximum d'imbrication,
- Activer/désactiver les threads dynamiques,
- Fixer la taille de la pile des threads,
- Régler la politique d'attente des threads.

Exemple sh/bash:
export OMP_NUM_THREADS=8

Variables d'environnements

OpenMP fournit plusieurs variables d'environnement pour contrôler l'exécution du code parallèle . Elles permettent de :

- Fixer le nombre de threads,
- Définir la division des itérations,
- Lier les threads aux processeurs,
- Activer/désativer le parallélisme imbriqué, fixer le niveau maximum d'imbrication,
- Activer/désactiver les threads dynamiques,
- Fixer la taille de la pile des threads,
- Régler la politique d'attente des threads.

Exemple sh/bash:
export OMP_NUM_THREADS=8

Variables d'environnements

OpenMP fournit plusieurs variables d'environnement pour contrôler l'exécution du code parallèle . Elles permettent de :

- Fixer le nombre de threads,
- Définir la division des itérations,
- Lier les threads aux processeurs,
- Activer/désativer le parallélisme imbriqué, fixer le niveau maximum d'imbrication,
- Activer/désactiver les threads dynamiques,
- Fixer la taille de la pile des threads,
- Régler la politique d'attente des threads.

Exemple sh/bash:
export OMP_NUM_THREADS=8

Variables d'environnements

OpenMP fournit plusieurs variables d'environnement pour contrôler l'exécution du code parallèle . Elles permettent de :

- Fixer le nombre de threads,
- Définir la division des itérations,
- Lier les threads aux processeurs,
- Activer/désativer le parallélisme imbriqué, fixer le niveau maximum d'imbrication,
- Activer/désactiver les threads dynamiques,
- Fixer la taille de la pile des threads,
- Régler la politique d'attente des threads.

Exemple sh/bash:
export OMP_NUM_THREADS=8

Variables d'environnements

OpenMP fournit plusieurs variables d'environnement pour contrôler l'exécution du code parallèle . Elles permettent de :

- Fixer le nombre de threads,
- Définir la division des itérations,
- Lier les threads aux processeurs,
- Activer/désativer le parallélisme imbriqué, fixer le niveau maximum d'imbrication,
- Activer/désactiver les threads dynamiques,
- Fixer la taille de la pile des threads,
- Régler la politique d'attente des threads.

Exemple sh/bash:
export OMP_NUM_THREADS=8

C/C++ - Structure générale du code

```
#include <omp.h>
main() {
     int var1, var2, var3;
     Code séquentiel ...
   Début de la section parallèle. Lancement des threads.
   Spécification de la portée des variables.
#pragma omp parallel private(var1, var2) shared(var3)
{          Section parallèle exécutée par tous les threads
         Autres directives OpenMP
         Appel de fonctions de la bibliothèque OpenMP
         Tous les threads rejoignent le thread principal et finissent
}
     Reprise du code séqentiel
     ...
```

Format des directives C/C++

#pragma omp	**directive**	**[clause, ...]**	**retour à la ligne**
Obligatoire avant toutes les directives C/C++	Directive OpenMP valide.	clauses optionnelles	Obligatoire avant le bloc de la directive

Exemple:
#pragma omp parallel default(shared) private(phi, teta)

Règles générales:

- Sensible à la casse,
- Les directives doivent respecter les conventions des compilateurs C/C++,
- Un seul nom de directive est autorisé par directive,
- Les directives longues peuvent être écrites sur plusieurs lignes finissant par un backslash "\".

Format des directives C/C++

#pragma omp	**directive**	**[clause, ...]**	**retour à la ligne**
Obligatoire avant toutes les directives C/C++	Directive OpenMP valide.	clauses optionnelles	Obligatoire avant le bloc de la directive

Exemple:
#pragma omp parallel default(shared) private(phi, teta)

Règles générales:

- Sensible à la casse,
- Les directives doivent respecter les conventions des compilateurs C/C++,
- Un seul nom de directive est autorisé par directive,
- Les directives longues peuvent être écrites sur plusieurs lignes finissant par un backslash "\".

Format des directives C/C++

#pragma omp	**directive**	**[clause, ...]**	**retour à la ligne**
Obligatoire avant toutes les directives C/C++	Directive OpenMP valide.	clauses optionnelles	Obligatoire avant le bloc de la directive

Exemple:
#pragma omp parallel default(shared) private(phi, teta)

Règles générales:

- Sensible à la casse,
- Les directives doivent respecter les conventions des compilateurs C/C++,
- Un seul nom de directive est autorisé par directive,
- Les directives longues peuvent être écrites sur plusieurs lignes finissant par un backslash "\".

Format des directives C/C++

#pragma omp	**directive**	**[clause, ...]**	**retour à la ligne**
Obligatoire avant toutes les directives C/C++	Directive OpenMP valide.	clauses optionnelles	Obligatoire avant le bloc de la directive

Exemple:
#pragma omp parallel default(shared) private(phi, teta)

Règles générales:

- Sensible à la casse,
- Les directives doivent respecter les conventions des compilateurs C/C++,
- Un seul nom de directive est autorisé par directive,
- Les directives longues peuvent être écrites sur plusieurs lignes finissant par un backslash "\".

Construction d'une section de code parallèle

Une section de code parallèle est un bloc d'instructions qui sera exécuté par plusieurs threads.

Format:

```
#pragma omp parallel [clause ...]
                     if (expression_logique)
                     private(liste)
                     shared (liste)
                     default (shared | none)
                     firstprivate (liste)
                     reduction (opérateur: liste)
                     copyin (liste)
                     num_threads (expression_entière)
    bloc_structuré
```

Construction d'une section de code parallèle

- Lorsqu'un thread atteint une directive parallèle, il crée un ensemble de threads dont il devient le maître, il a le numéro 0.
- À partir du début de cette section parallèle le code est dupliqué et tous les threads exécutent ce code.
- À la fin de ce bloc seul le thread maître continue l'exécution.
- Si un thread se termine dans cette section, tous les threads se terminent aussi.

Construction d'une section de code parallèle

- Lorsqu'un thread atteint une directive parallèle, il crée un ensemble de threads dont il devient le maître, il a le numéro 0.
- À partir du début de cette section parallèle le code est dupliqué et tous les threads exécutent ce code.
- À la fin de ce bloc seul le thread maître continue l'exécution.
- Si un thread se termine dans cette section, tous les threads se terminent aussi.

Construction d'une section de code parallèle

- Lorsqu'un thread atteint une directive parallèle, il crée un ensemble de threads dont il devient le maître, il a le numéro 0.
- À partir du début de cette section parallèle le code est dupliqué et tous les threads exécutent ce code.
- À la fin de ce bloc seul le thread maître continue l'exécution.
- Si un thread se termine dans cette section, tous les threads se terminent aussi.

Construction d'une section de code parallèle

- Lorsqu'un thread atteint une directive parallèle, il crée un ensemble de threads dont il devient le maître, il a le numéro 0.
- À partir du début de cette section parallèle le code est dupliqué et tous les threads exécutent ce code.
- À la fin de ce bloc seul le thread maître continue l'exécution.
- Si un thread se termine dans cette section, tous les threads se terminent aussi.

Nombre de threads

Le nombre de threads d'une section parallèle est déterminé dans l'ordre par les facteurs suivants:

- Évaluation de la clause **if**,
- Affectation de la valeur de la clause **num_threads**,
- Utlisation de la fonction **omp_set_num_threads()**,
- Affectation de la valeur de la variable d'environnement **OMP_NUM_THREADS**,
- Implémentation par défault (généralement le nombre de processeur/noeud).

Les threads sont numérotés de 0 (le maître) à N-1.

Nombre de threads

Le nombre de threads d'une section parallèle est déterminé dans l'ordre par les facteurs suivants:

- Évaluation de la clause **if**,
- Affectation de la valeur de la clause **num_threads**,
- Utlisation de la fonction **omp_set_num_threads()**,
- Affectation de la valeur de la variable d'environnement **OMP_NUM_THREADS**,
- Implémentation par défault (généralement le nombre de processeur/noeud).

Les threads sont numérotés de 0 (le maître) à N-1.

Nombre de threads

Le nombre de threads d'une section parallèle est déterminé dans l'ordre par les facteurs suivants:

- Évaluation de la clause **if**,
- Affectation de la valeur de la clause **num_threads**,
- Utlisation de la fonction **omp_set_num_threads()**,
- Affectation de la valeur de la variable d'environnement **OMP_NUM_THREADS**,
- Implémentation par défault (généralement le nombre de processeur/noeud).

Les threads sont numérotés de 0 (le maître) à N-1.

Nombre de threads

Le nombre de threads d'une section parallèle est déterminé dans l'ordre par les facteurs suivants:

- Évaluation de la clause **if**,
- Affectation de la valeur de la clause **num_threads**,
- Utlisation de la fonction **omp_set_num_threads()**,
- Affectation de la valeur de la variable d'environnement **OMP_NUM_THREADS**,
- Implémentation par défault (généralement le nombre de processeur/noeud).

Les threads sont numérotés de 0 (le maître) à N-1.

Nombre de threads

Le nombre de threads d'une section parallèle est déterminé dans l'ordre par les facteurs suivants:

- Évaluation de la clause **if**,
- Affectation de la valeur de la clause **num_threads**,
- Utlisation de la fonction **omp_set_num_threads()**,
- Affectation de la valeur de la variable d'environnement **OMP_NUM_THREADS**,
- Implémentation par défault (généralement le nombre de processeur/noeud).

Les threads sont numérotés de 0 (le maître) à N-1.

Restrictions

- Lorsqu'une clause **if** est présente, un ensemble de threads n'est créé que si la condition est vraie sinon le bloc est exécuté séquentiellement par le thread maître,
- Une section parallèle doit être un bloc structuré qui ne couvre pas plusieurs routines ou fichiers de code,
- L'utilisation du **goto** est interdite depuis ou vers une section parallèle,
- Une seule clause **if** est autorisée,
- Une seule clause **num_threads** est autorisée,
- Un programme ne doit pas dépendre de l'ordre des clauses.

Restrictions

- Lorsqu'une clause **if** est présente, un ensemble de threads n'est créé que si la condition est vraie sinon le bloc est exécuté séquentiellement par le thread maître,
- Une section parallèle doit être un bloc structuré qui ne couvre pas plusieurs routines ou fichiers de code,
- L'utilisation du **goto** est interdite depuis ou vers une section parallèle,
- Une seule clause **if** est autorisée,
- Une seule clause **num_threads** est autorisée,
- Un programme ne doit pas dépendre de l'ordre des clauses.

Restrictions

- Lorsqu'une clause **if** est présente, un ensemble de threads n'est créé que si la condition est vraie sinon le bloc est exécuté séquentiellement par le thread maître,
- Une section parallèle doit être un bloc structuré qui ne couvre pas plusieurs routines ou fichiers de code,
- L'utilisation du **goto** est interdite depuis ou vers une section parallèle,
- Une seule clause **if** est autorisée,
- Une seule clause **num_threads** est autorisée,
- Un programme ne doit pas dépendre de l'ordre des clauses.

Restrictions

- Lorsqu'une clause **if** est présente, un ensemble de threads n'est créé que si la condition est vraie sinon le bloc est exécuté séquentiellement par le thread maître,
- Une section parallèle doit être un bloc structuré qui ne couvre pas plusieurs routines ou fichiers de code,
- L'utilisation du **goto** est interdite depuis ou vers une section parallèle,
- Une seule clause **if** est autorisée,
- Une seule clause **num_threads** est autorisée,
- Un programme ne doit pas dépendre de l'ordre des clauses.

Restrictions

- Lorsqu'une clause **if** est présente, un ensemble de threads n'est créé que si la condition est vraie sinon le bloc est exécuté séquentiellement par le thread maître,
- Une section parallèle doit être un bloc structuré qui ne couvre pas plusieurs routines ou fichiers de code,
- L'utilisation du **goto** est interdite depuis ou vers une section parallèle,
- Une seule clause **if** est autorisée,
- Une seule clause **num_threads** est autorisée,
- Un programme ne doit pas dépendre de l'ordre des clauses.

Restrictions

- Lorsqu'une clause **if** est présente, un ensemble de threads n'est créé que si la condition est vraie sinon le bloc est exécuté séquentiellement par le thread maître,
- Une section parallèle doit être un bloc structuré qui ne couvre pas plusieurs routines ou fichiers de code,
- L'utilisation du **goto** est interdite depuis ou vers une section parallèle,
- Une seule clause **if** est autorisée,
- Une seule clause **num_threads** est autorisée,
- Un programme ne doit pas dépendre de l'ordre des clauses.

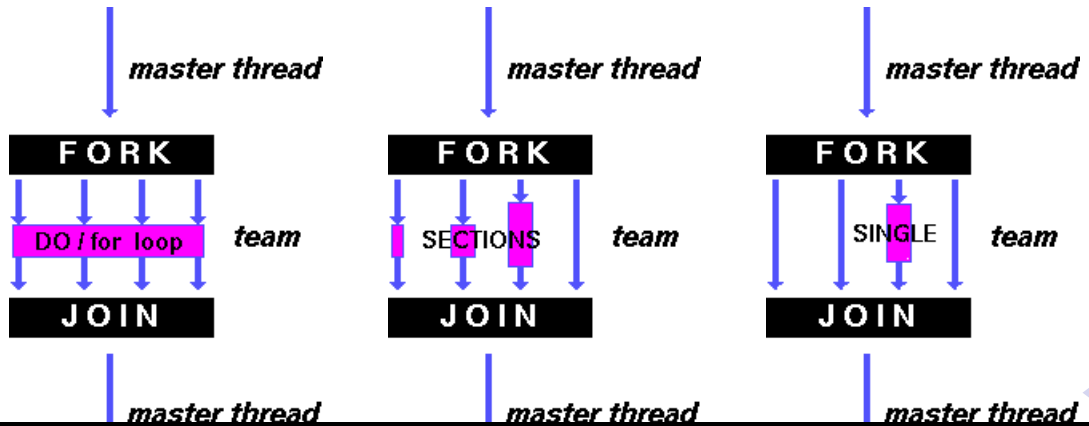
Exemple

```
#include <omp.h>
int main() {
int nthreads, tid;
/* création d'un ensemble de threads disposant chacun d'une
variable tid privée */
#pragma omp parallel private(tid)
{
     tid=omp_get_thread_num();
     printf(" Je suis le thread Numéro: %d \n",tid);
    if (tid==0) { /* Exécuté seulement par le maître */
         nthreads=omp_get_num_threads();
        printf(" Nombre de threads: %d \n",nthreads);
     }
}
```

Construction d'une zone de travail partagé

Elle permet de diviser l'exécution d'une zone de code entre les membres d'un ensemble de threads. Elle ne lance pas de nouveau thread.

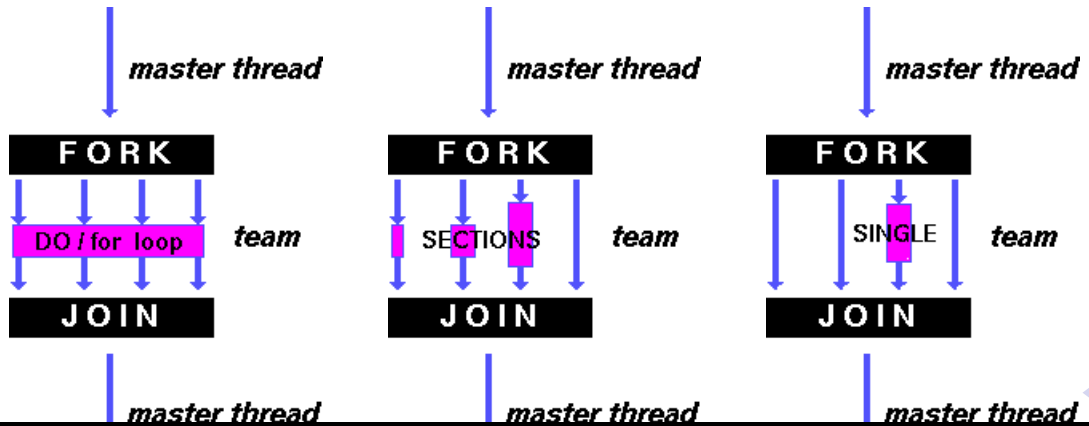
- La directive **for** partage les itérations d'une boucle entre les membres de l'ensemble (parallélisme de données),
- La directive **sections** permet de faire exécuter chaque section par un thread (parallélisme fonctionnel),
- La directive **single** sérialise une section de code.



Construction d'une zone de travail partagé

Elle permet de diviser l'exécution d'une zone de code entre les membres d'un ensemble de threads. Elle ne lance pas de nouveau thread.

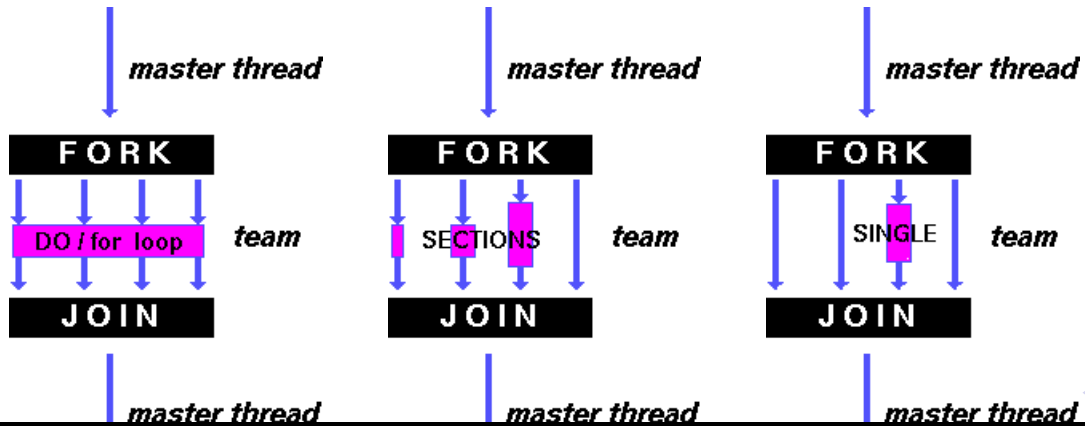
- La directive **for** partage les itérations d'une boucle entre les membres de l'ensemble (parallélisme de données),
- La directive **sections** permet de faire exécuter chaque section par un thread (parallélisme fonctionnel),
- La directive **single** sérialise une section de code.



Construction d'une zone de travail partagé

Elle permet de diviser l'exécution d'une zone de code entre les membres d'un ensemble de threads. Elle ne lance pas de nouveau thread.

- La directive **for** partage les itérations d'une boucle entre les membres de l'ensemble (parallélisme de données),
- La directive **sections** permet de faire exécuter chaque section par un thread (parallélisme fonctionnel),
- La directive **single** sérialise une section de code.



Résumé des directives et clauses

Clause	Directive					
	parallel	for	sections	single	parallel for	parallel sections
if	●				●	●
private	●	●	●	●	●	●
shared	●	●			●	●
default	●				●	●
firstprivate	●	●	●	●	●	●
lastprivate		●	●		●	●
reduction	●	●	●		●	●
copyin				●		
copyprivate					●	●
schedule		●			●	
ordered		●			●	
nowait		●	●	●		

Open MPI: Introduction

Message Passing Interface est une bibliothèque de passage de messages créée par un consortiums de plus 40 organisations (constructeurs, éditeurs de logiciels, chercheurs et utilisateurs).

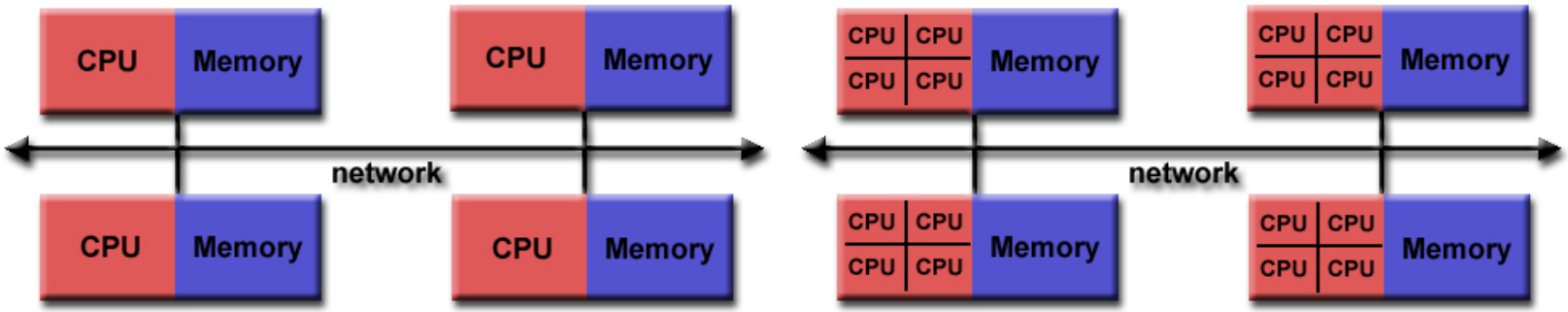
Son but est de fournir un outil portable, standard et flexible pour l'écriture de programmes de passage de messages.

Ce standard n'est pas porté par IEEE ou ISO, mais est devenu Le standard industriel pour toutes les plateformes de calcul parallèle de haut niveau.

Les routines de gestion de l'environnement, de communications point à point ou de communications collectives existent en C/C++ et Fortran mais seul l'aspect programmation en C sera abordé dans ce cours.

OpenMPI

A l'origine MPI a été conçu pour des architectures distribuées. Comme les tendances de l'architecture on changé, l'implémentation de MPI a évolué pour prendre en compte des systèmes à mémoire partagée connectés par un réseau.

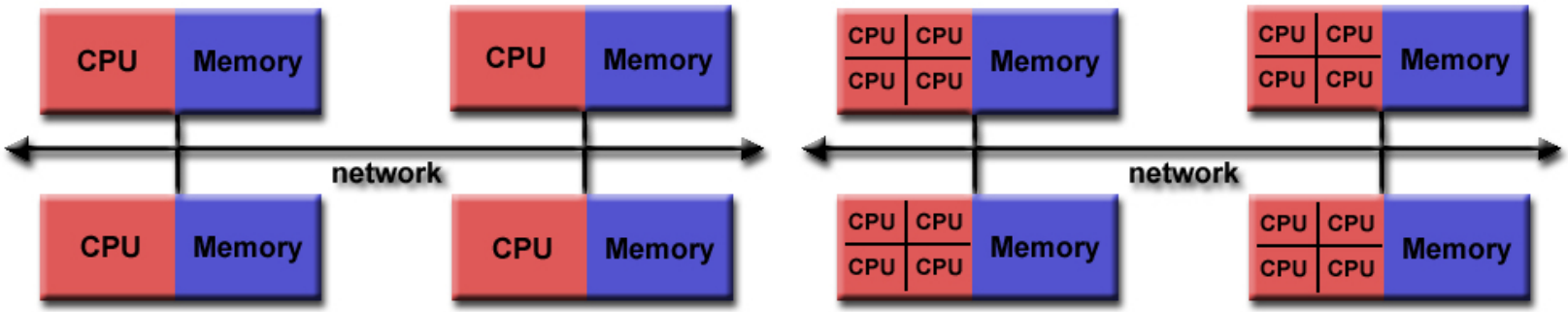


Aujourd'hui MPI tourne sur:

- des systèmes distribués,
- des systèmes à mémoire partagée,
- des systèmes hybrides.

OpenMPI

A l'origine MPI a été conçu pour des architectures distribuées. Comme les tendances de l'architecture on changé, l'implémentation de MPI a évolué pour prendre en compte des systèmes à mémoire partagée connectés par un réseau.

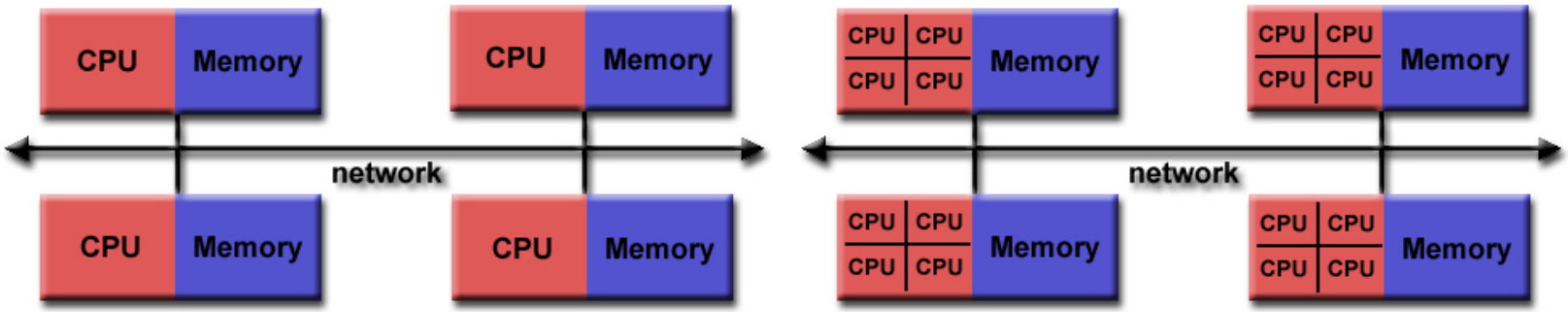


Aujourd'hui MPI tourne sur:

- des systèmes distribués,
- des systèmes à mémoire partagée,
- des systèmes hybrides.

OpenMPI

A l'origine MPI a été conçu pour des architectures distribuées. Comme les tendances de l'architecture on changé, l'implémentation de MPI a évolué pour prendre en compte des systèmes à mémoire partagée connectés par un réseau.



Aujourd'hui MPI tourne sur:

- des systèmes distribués,
- des systèmes à mémoire partagée,
- des systèmes hybrides.

Open MPI: Pourquoi utiliser MPI?

- **Standardisation** - MPI est la seule bibliothèque pouvant être considére comme standard. Elle est supportée par toutes les plateformes de calcul parallèle (HPC).
- **Portabilité** - Peu de modification du code source lors du portage d'un plateforme à l'autre.
- **Performance** - Les Implémentations des fournisseurs devraient être en mesure d'exploiter les fonctionnalités matérielles natives pour optimiser les performances.
- **Fonctionnalité** - Plus de 400 routines définies dans MPI-3.
- **Disponibilité** - Une variété d'implémentations existent tant dans le domaine privé que publique.

Open MPI: Pourquoi utiliser MPI?

- **Standardisation** - MPI est la seule bibliothèque pouvant être considére comme standard. Elle est supportée par toutes les plateformes de calcul parallèle (HPC).
- **Portabilité** - Peu de modification du code source lors du portage d'un plateforme à l'autre.
- **Performance** - Les Implémentations des fournisseurs devraient être en mesure d'exploiter les fonctionnalités matérielles natives pour optimiser les performances.
- **Fonctionnalité** - Plus de 400 routines définies dans MPI-3.
- **Disponibilité** - Une variété d'implémentations existent tant dans le domaine privé que publique.

Open MPI: Pourquoi utiliser MPI?

- **Standardisation** - MPI est la seule bibliothèque pouvant être considére comme standard. Elle est supportée par toutes les plateformes de calcul parallèle (HPC).
- **Portabilité** - Peu de modification du code source lors du portage d'un plateforme à l'autre.
- **Performance** - Les Implémentations des fournisseurs devraient être en mesure d'exploiter les fonctionnalités matérielles natives pour optimiser les performances.
- **Fonctionnalité** - Plus de 400 routines définies dans MPI-3.
- **Disponibilité** - Une variété d'implémentations existent tant dans le domaine privé que publique.

Open MPI: Pourquoi utiliser MPI?

- **Standardisation** - MPI est la seule bibliothèque pouvant être considére comme standard. Elle est supportée par toutes les plateformes de calcul parallèle (HPC).
- **Portabilité** - Peu de modification du code source lors du portage d'un plateforme à l'autre.
- **Performance** - Les Implémentations des fournisseurs devraient être en mesure d'exploiter les fonctionnalités matérielles natives pour optimiser les performances.
- **Fonctionnalité** - Plus de 400 routines définies dans MPI-3.
- **Disponibilité** - Une variété d'implémentations existent tant dans le domaine privé que publique.

Open MPI: Pourquoi utiliser MPI?

- **Standardisation** - MPI est la seule bibliothèque pouvant être considére comme standard. Elle est supportée par toutes les plateformes de calcul parallèle (HPC).
- **Portabilité** - Peu de modification du code source lors du portage d'un plateforme à l'autre.
- **Performance** - Les Implémentations des fournisseurs devraient être en mesure d'exploiter les fonctionnalités matérielles natives pour optimiser les performances.
- **Fonctionnalité** - Plus de 400 routines définies dans MPI-3.
- **Disponibilité** - Une variété d'implémentations existent tant dans le domaine privé que publique.

Niveau de support des Threads

Les bibliothèques MPI varient dans leur niveau de support des threads:

- MPI_THREAD_SINGLE - niveau 0: Un seul thread s'exécute.
- MPI_THREAD_FUNNELED - niveau 1: Le processus peut être multithread, mais seul le thread maître peut faire des appels MPI. Les appels MPI sont canalisés vers le thread maître.
- MPI_THREAD_SERIALIZED - niveau 2: Le processus peut être multithread, les threads peuvent effectués des appels MPI, mais un seul à la fois.
- MPI_THREAD_MULTIPLE - niveau 3: Plusieurs threads peuvent effectuer des appels MPI sans restrictions.

Niveau de support des Threads

Les bibliothèques MPI varient dans leur niveau de support des threads:

- MPI_THREAD_SINGLE - niveau 0: Un seul thread s'exécute.
- MPI_THREAD_FUNNELED - niveau 1: Le processus peut être multithread, mais seul le thread maître peut faire des appels MPI. Les appels MPI sont canalisés vers le thread maître.
- MPI_THREAD_SERIALIZED - niveau 2: Le processus peut être multithread, les threads peuvent effectués des appels MPI, mais un seul à la fois.
- MPI_THREAD_MULTIPLE - niveau 3: Plusieurs threads peuvent effectuer des appels MPI sans restrictions.

Niveau de support des Threads

Les bibliothèques MPI varient dans leur niveau de support des threads:

- MPI_THREAD_SINGLE - niveau 0: Un seul thread s'exécute.
- MPI_THREAD_FUNNELED - niveau 1: Le processus peut être multithread, mais seul le thread maître peut faire des appels MPI. Les appels MPI sont canalisés vers le thread maître.
- MPI_THREAD_SERIALIZED - niveau 2: Le processus peut être multithread, les threads peuvent effectués des appels MPI, mais un seul à la fois.
- MPI_THREAD_MULTIPLE - niveau 3: Plusieurs threads peuvent effectuer des appels MPI sans restrictions.

Niveau de support des Threads

Les bibliothèques MPI varient dans leur niveau de support des threads:

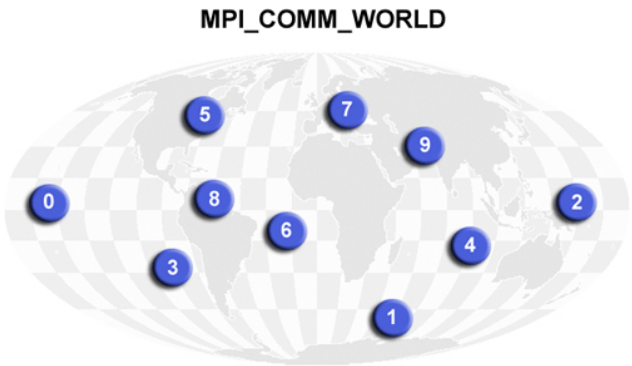
- MPI_THREAD_SINGLE - niveau 0: Un seul thread s'exécute.
- MPI_THREAD_FUNNELED - niveau 1: Le processus peut être multithread, mais seul le thread maître peut faire des appels MPI. Les appels MPI sont canalisés vers le thread maître.
- MPI_THREAD_SERIALIZED - niveau 2: Le processus peut être multithread, les threads peuvent effectués des appels MPI, mais un seul à la fois.
- MPI_THREAD_MULTIPLE - niveau 3: Plusieurs threads peuvent effectuer des appels MPI sans restrictions.

Exemple: choix du niveau de support des threads

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[]) {
    int recu, demand;
/*** choisir une fonction parmi:
    MPI_Init_thread( 0, 0, MPI_THREAD_SINGLE, &recu );
    MPI_Init_thread( 0, 0, MPI_THREAD_FUNNELED, &recu );
    MPI_Init_thread( 0, 0, MPI_THREAD_SERIALIZED, &recu );
    MPI_Init_thread( 0, 0, MPI_THREAD_MULTIPLE, &recu ); ***/
    MPI_Init_thread( 0, 0, MPI_THREAD_MULTIPLE, &recu );
    MPI_Query_thread( &demand );
    printf( "Query thread level= %d Init_thread level= %d", demand,
recu );
    MPI_Finalize();
}
```

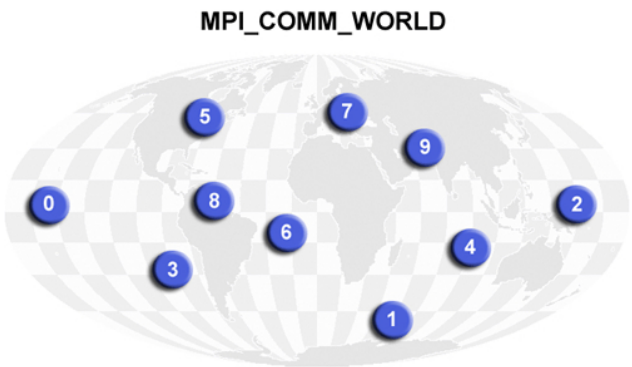
Communicators et Groups

- MPI utilise des objets appelés Communicators et Groups pour définir quels processus communiquent entre eux.
- La plupart des routines MPI necessitent la spécification d'un communicator comme argument.
- MPI_COMM_WORLD sera utilisé dès qu'une communication est nécessaire.



Communicators et Groups

- MPI utilise des objets appelés Communicators et Groups pour définir quels processus communiquent entre eux.
- La plupart des routines MPI necessitent la spécification d'un communicator comme argument.
- MPI_COMM_WORLD sera utilisé dès qu'une communication est nécessaire.



Communicators et Groups

- MPI utilise des objets appelés Communicators et Groups pour définir quels processus communiquent entre eux.
- La plupart des routines MPI necessitent la spécification d'un communicator comme argument.
- MPI_COMM_WORLD sera utilisé dès qu'une communication est nécessaire.

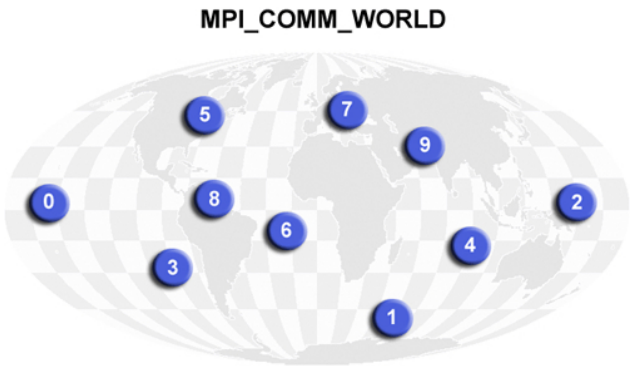


Table des Matières

Bibliographie

- TOP 500
  http://top500.org
- Norme POSIX
  http://get.posixcertified.ieee.org/docs/POSIX_Certification_Guide.
- Threads
  http://randu.org/tutorials/threads/
- OpenMP
  http://openmp.org/wp/
  http://openmp.org/mp-documents/Intro_To_OpenMP_Mattson.pdf
- OpenMPI
  http://www.open-mpi.org

Bibliographie

- TOP 500
  http://top500.org
- Norme POSIX
  http://get.posixcertified.ieee.org/docs/POSIX_Certification_Guide.
- Threads
  http://randu.org/tutorials/threads/
- OpenMP
  http://openmp.org/wp/
  http://openmp.org/mp-documents/Intro_To_OpenMP_Mattson.pdf
- OpenMPI
  http://www.open-mpi.org

Bibliographie

- TOP 500
  http://top500.org
- Norme POSIX
  http://get.posixcertified.ieee.org/docs/POSIX_Certification_Guide.
- Threads
  http://randu.org/tutorials/threads/
- OpenMP
  http://openmp.org/wp/
  http://openmp.org/mp-documents/Intro_To_OpenMP_Mattson.pdf
- OpenMPI
  http://www.open-mpi.org

Bibliographie

- TOP 500
  http://top500.org
- Norme POSIX
  http://get.posixcertified.ieee.org/docs/POSIX_Certification_Guide.
- Threads
  http://randu.org/tutorials/threads/
- OpenMP
  http://openmp.org/wp/
  http://openmp.org/mp-documents/Intro_To_OpenMP_Mattson.pdf
- OpenMPI
  http://www.open-mpi.org

Bibliographie

- TOP 500
  http://top500.org
- Norme POSIX
  http://get.posixcertified.ieee.org/docs/POSIX_Certification_Guide.
- Threads
  http://randu.org/tutorials/threads/
- OpenMP
  http://openmp.org/wp/
  http://openmp.org/mp-documents/Intro_To_OpenMP_Mattson.pdf
- OpenMPI
  http://www.open-mpi.org

Bibliographie (suite)


Calcul parallèle
https://computing.llnl.gov
http://calcul.math.cnrs.fr/Documents/Ecoles/LEM2I/Mod3/paral.pd
http://fr.wikipedia.org/wiki/Parallélisme_(informatique)
http://www.metz.supelec.fr/metz/personnel/vialle/course/IIC-CPDG/
https://www.lri.fr/ de/Polytech-Architectures-parallèles.pdf


GPU Nividia
http://www.nvidia.fr/object/gpu-computing-fr.html

Bibliographie (suite)

Calcul parallèle
https://computing.llnl.gov
http://calcul.math.cnrs.fr/Documents/Ecoles/LEM2I/Mod3/paral.pd
http://fr.wikipedia.org/wiki/Parallélisme_(informatique)
http://www.metz.supelec.fr/metz/personnel/vialle/course/IIC-CPDG/
https://www.lri.fr/ de/Polytech-Architectures-parallèles.pdf

GPU Nividia
http://www.nvidia.fr/object/gpu-computing-fr.html