

Introduction à la Programmation Parallèle

Partie 4 - Algorithmes parallèles et modèles de calcul



Dr Babacar Diop

Département d'Informatique

UFR des Sciences Appliquées et de Technologies

Université Gaston Berger de Saint-Louis

Année scolaire: 2019/2020

Analyse d'un algorithme parallèle (1)

Les **algorithmes parallèles** sont conçus pour améliorer la vitesse de calcul et d'exécution sur un ordinateur.

Pour analyser un algorithme parallèle, on considère généralement les paramètres suivants:

- **La complexité temporelle** (temps d'exécution),
- **Nombre total de processeurs utilisés**,
- Et le coût total.

Analyse d'un algorithme parallèle (2)

Analyse temporelle

- ◎ L'**objectif principal** de paralléliser un algorithme c'est de **réduire le temps de calcul de l'algorithme séquentiel**.
- ◎ Évaluer le temps d'exécution d'un algorithme est extrêmement important pour analyser son efficacité.
- ◎ **Temps d'exécution total** = durée à partir du moment où l'algorithme commence à s'exécuter jusqu'au moment où il s'arrête.
- ◎ **Temps d'exécution total** = le moment où le premier processeur a commencé son exécution jusqu'au moment où le dernier processeur arrête son exécution.

Analyse d'un algorithme parallèle (3)

La **complexité temporelle** d'un algorithme peut être classée en trois catégories:

- ◎ **Complexité au pire cas** - lorsque le temps nécessaire à un algorithme pour une entrée donnée est maximal.
- ◎ **Complexité au moyen des cas** - Lorsque la quantité de temps requise par un algorithme pour une entrée donnée est moyenne.
- ◎ **Complexité au meilleur des cas** - Lorsque le temps requis par un algorithme pour une entrée donnée est minimum.

Analyse d'un algorithme parallèle (5)

Nombre de processeurs utilisés

- ◉ **Le nombre de processeurs** utilisés est un facteur important dans l'analyse de l'efficacité d'un algorithme parallèle.
- ◉ Le coût de **déploiement, de maintenance et d'exécution** des ordinateurs est calculé.
- ◉ L'idée consisterait à minimiser le nombre de processeurs/processus utilisé pour résoudre efficacement un problème.

Algorithme parallèle: modèles

Le modèle d'un algorithme parallèle est développé en considérant **une stratégie pour diviser les données** et la méthode de traitement et en appliquant **une stratégie appropriée pour réduire les interactions**.

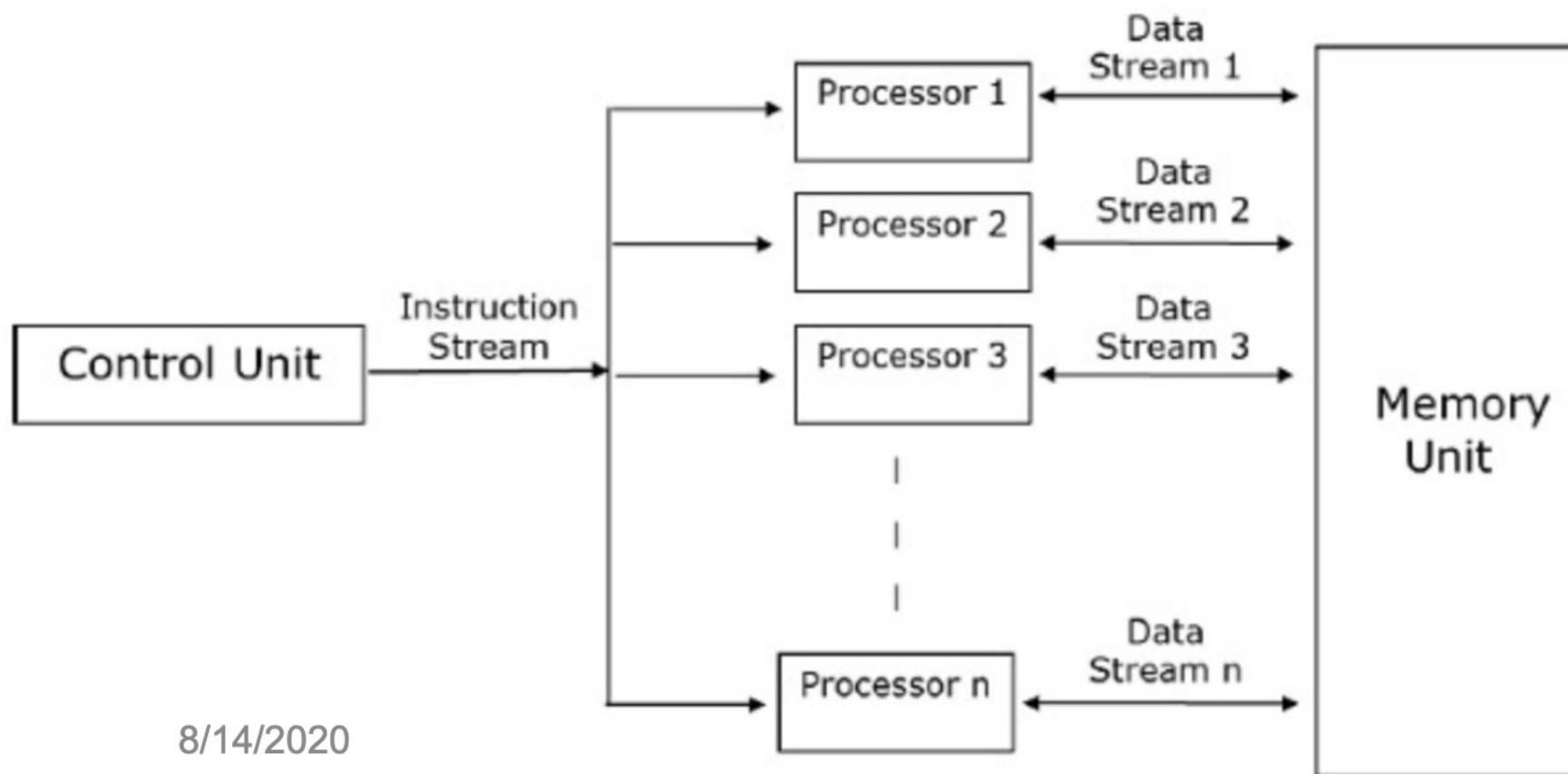
Nous discuterons des modèles d'algorithme parallèle suivants:

- ✓ Modèle **parallèle de données**
- ✓ Modèle de **graphe des tâches**
- ✓ Modèle de **pool de travail**
- ✓ Modèle **Maître-Esclave**
- ✓ Modèle du **Producteur-Consommateur** ou **modèle de pipeline**
- ✓ Modèles **PRAM (Parallel Random Access Machine)**
- ✓ Modèle **hybride**

Modèle de parallélisme de données

Dans ce modèle, les tâches sont attribuées aux processus et chaque tâche effectue des opérations similaires sur différentes données.

Le parallélisme des données est une séquence d'opérations identiques qui appliquées sur plusieurs éléments de données par différents processeurs.



Principale caractéristique:
le parallélisme des données **augmente avec la taille du problème**, ce qui implique d'utiliser plus de processus pour résoudre des problèmes plus grands en termes de données.

Modèle de graphe des tâches

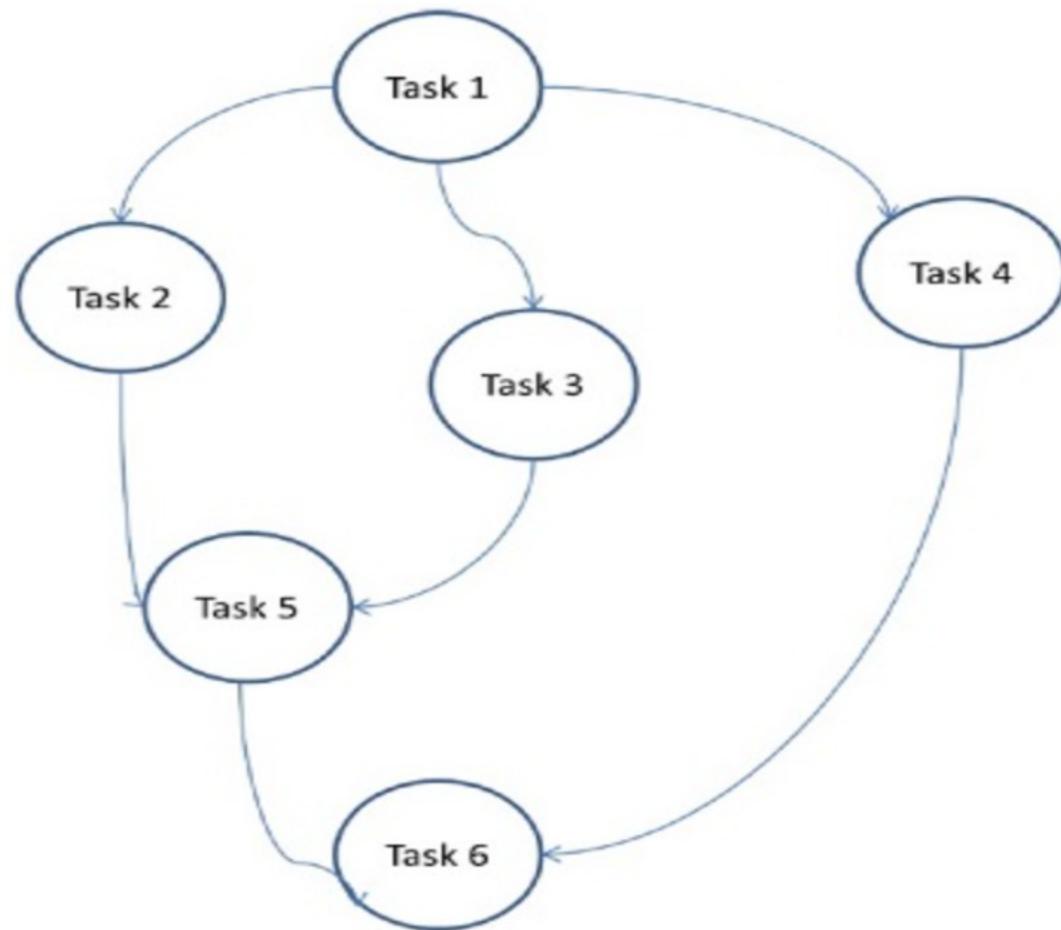
Dans ce modèle, le parallélisme est exprimé par un graphe de tâches.

La corrélation entre les tâches est utilisée pour minimiser les coûts.

Ce modèle est appliqué pour résoudre des **problèmes où la quantité de données associée aux tâches est énorme par rapport au nombre de processeurs** qui leur est associé.

Les tâches sont assignées pour aider à améliorer le coût du mouvement des données parmi les tâches.

Modèle de graphe des tâches



- Le problème est divisé en **tâches atomiques** et exécutées en graphe.
- **Chaque tâche** est une **unité de travail indépendante** qui a des **dépendances sur une ou plusieurs tâches antérieures**.
- Une fois la tâche terminée, **la sortie d'une tâche antécédente passe à la tâche dépendante**.
- Une tâche avec une tâche antécédente **ne commence l'exécution que lorsque sa tâche antécédente est terminée**.
- Le **résultat final** du graphe est reçu **lorsque la dernière tâche dépendante est terminée** (Exemple: Tâche 6 de la figure ci-dessus).

Modèle de pool de travail (1)

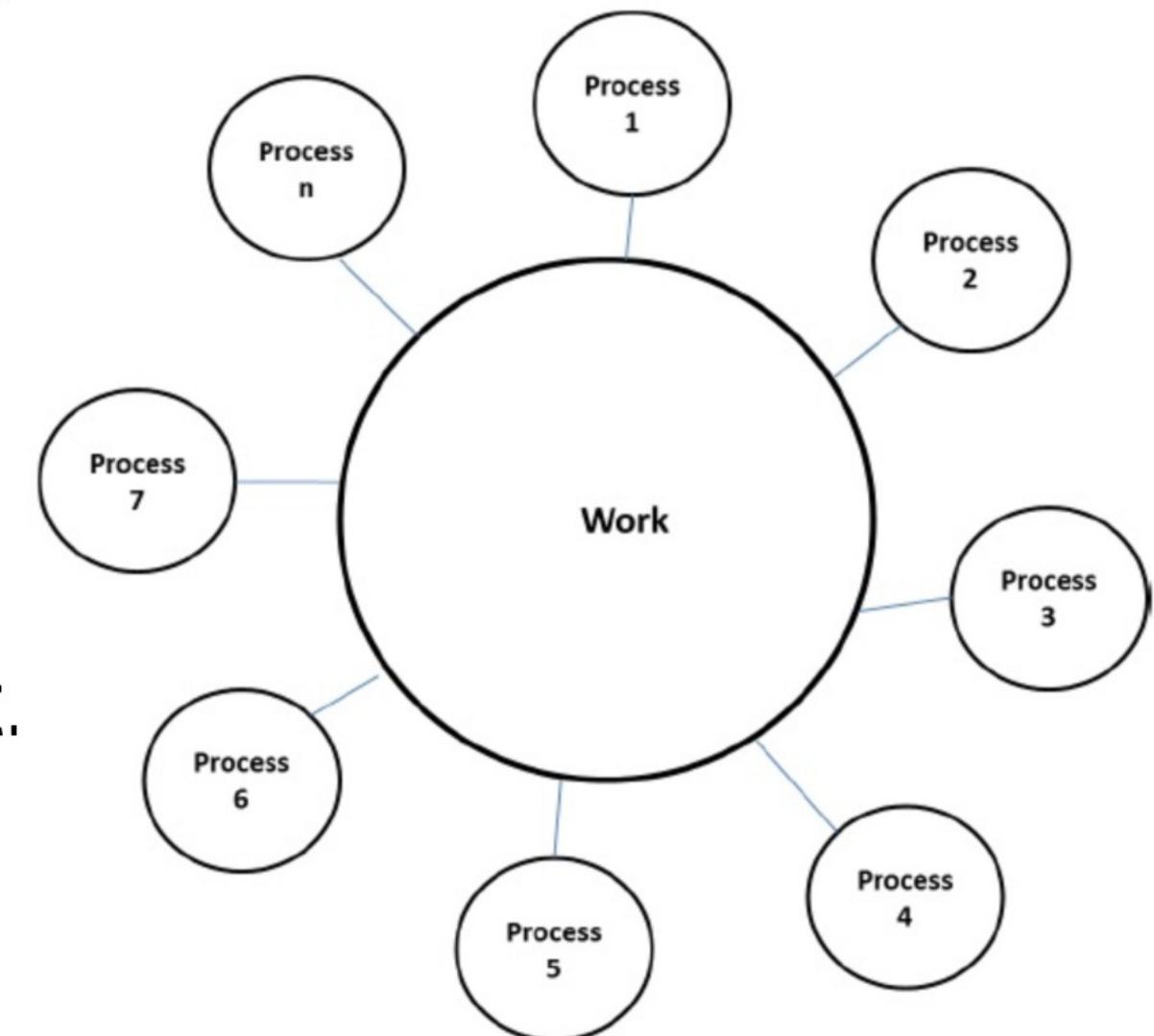
- Dans ce modèle, les tâches sont dynamiquement affectées aux processus d'équilibrage de la charge.
- *Tout processus peut éventuellement exécuter une tâche quelconque.*
- Modèle utilisé lorsque la quantité de données associée aux tâches est comparativement inférieure au calcul associé aux tâches.
- Il n'y a pas de pré-assignation des tâches sur les processus.
- L'attribution des tâches est centralisée ou décentralisée.

Modèle de pool de travail (2)

Les pointeurs sur les tâches sont enregistrés sur

- ✓ une **liste partagée** physiquement ou,
- ✓ dans une **file d'attente** de priorité ou,
- ✓ dans une **table** ou
- ✓ dans un **arbre de hashage**,
- ✓ ou dans une **structure de données spécifique** ...

... dans tous les cas répartie physiquement.

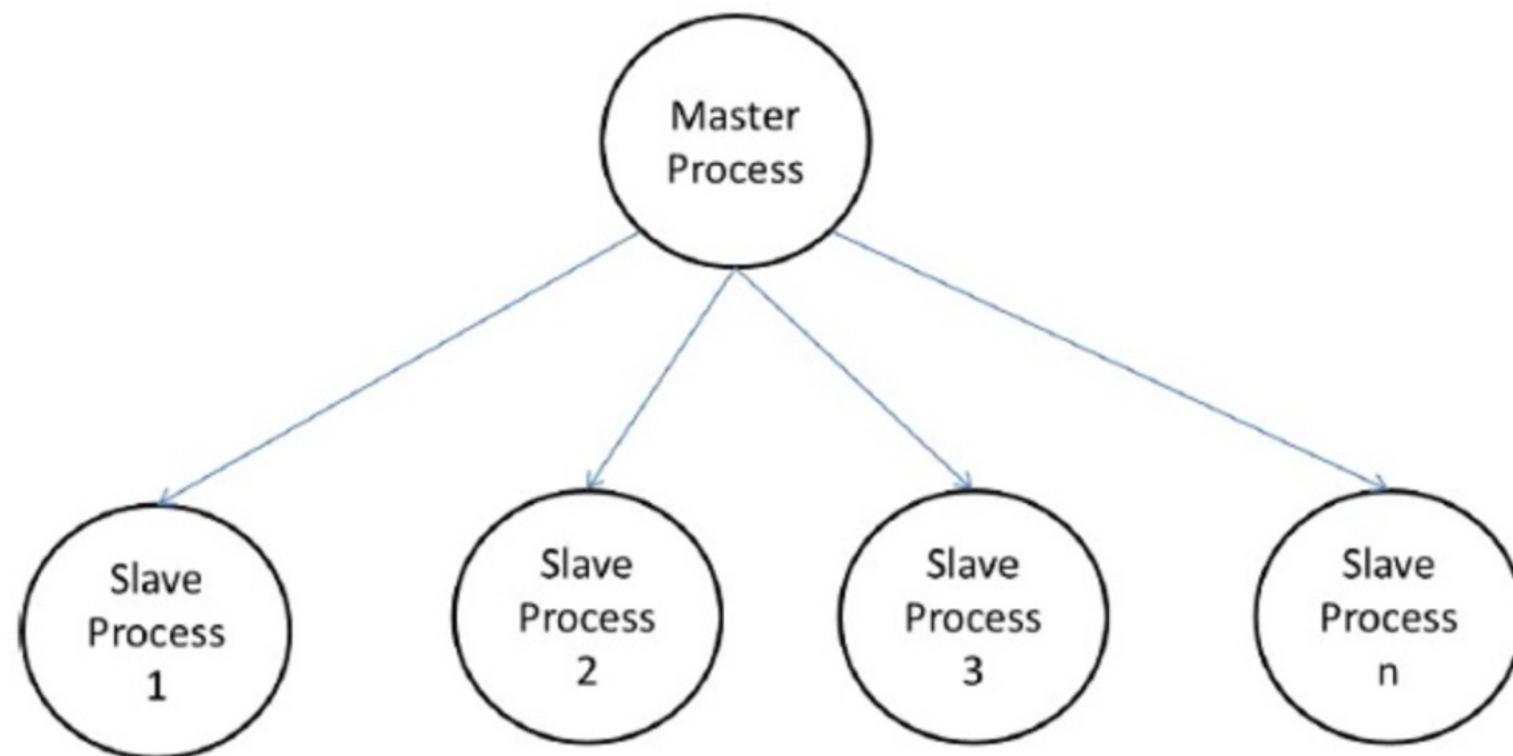


Modèle de pool de travail (3)

- La tâche peut être disponible au début, ou peut être générée dynamiquement.
- Si la tâche est générée dynamiquement et qu'une assignation décentralisée de la tâche est effectuée, un algorithme de détection de terminaison est requis pour que tous les processus puissent détecter l'achèvement de l'ensemble du programme et arrêter de chercher plus de tâches.

Modèle maître-esclave (1)

Dans ce modèle, un ou plusieurs processus **Maîtres (ou Master)** génèrent une tâche et l'attribuent à d'autres processus **Esclaves (ou Slave)**.



Modèle maître-esclave (2)

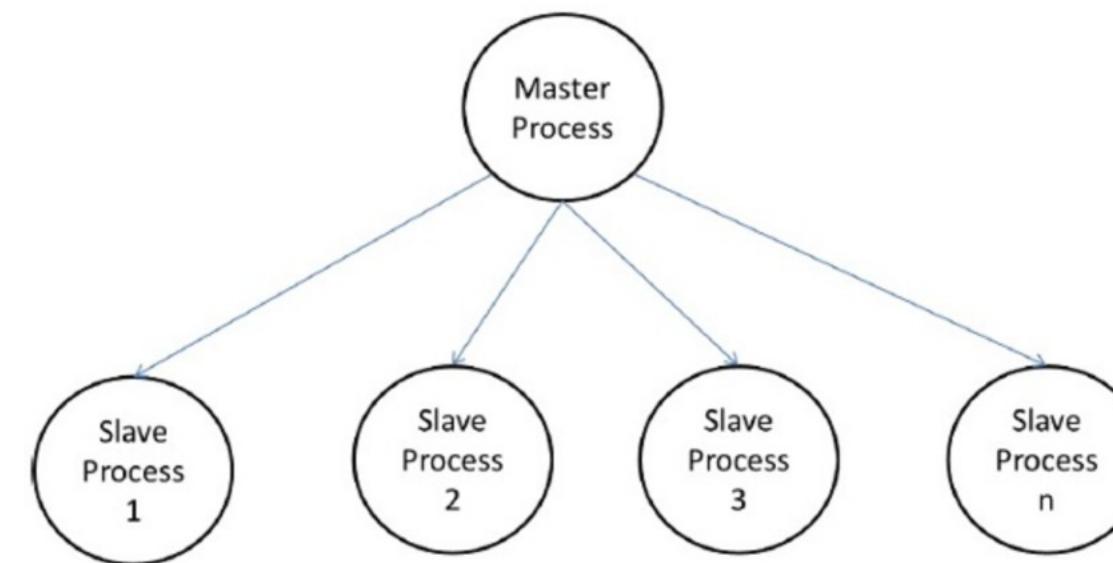
- Dans certains cas, une tâche doit être complétée par étapes et la tâche dans chaque phase doit être terminée avant que la tâche des phases suivantes ne puisse être générée.
- Le modèle **Maître-Esclave** peut être généralisé à un modèle **Maître-Esclave hiérarchique ou multi-niveau** dans lequel le **Maître** de niveau supérieur alimente la grande partie des tâches au(x) **Maître(s)** de second niveau, lesquels **subdivisent les tâches entre leurs propres esclaves**, et peuvent aussi effectuer une partie des tâches eux-même.

Modèle maître-esclave (3)

Les tâches peuvent être attribuées au préalable si :

- ✓ le Maître peut estimer le volume des tâches,
- ✓ ou par une **assignation aléatoire** pouvant satisfaire l'équilibrage de la charge.

Ce modèle est généralement adapté aux paradigmes de mémoire partagée ou de passage de message, du fait de l'interaction dans les deux sens.



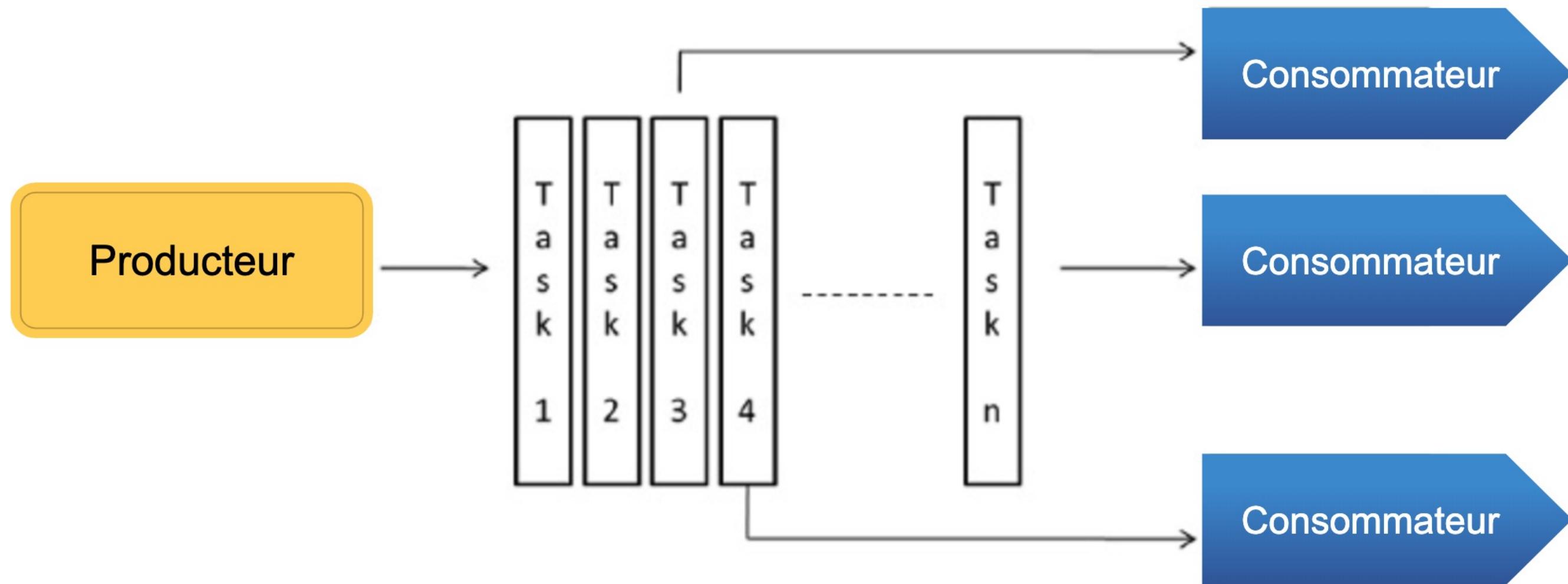
Modèle Pipeline (1)

- Un **ensemble de données** est transmis à travers une **série de processus**, dont **chacun réalise une tâche**.
- L'arrivée de **nouvelles données** génère l'exécution d'une **nouvelle tâche** par un processus en file d'attente.
- Les **processus pourraient constituer une file d'attente** sous la forme de **tableaux linéaires** ou multidimensionnels, d'arbres ou de graphiques généraux avec ou sans cycles.

Modèle Pipeline (2)

- **Exemple:** Chaîne de producteurs et de consommateurs.
- **Chaque processus en file d'attente** peut être considéré comme **un consommateur de données** pour le processus qui l'a précédé dans la file d'attente et en tant que **producteur de données** pour le processus qui le suit dans la file d'attente.
- **La file d'attente** peut être une **chaîne linéaire** ou un **graphe dirigé**.

Modèle Pipeline (3)

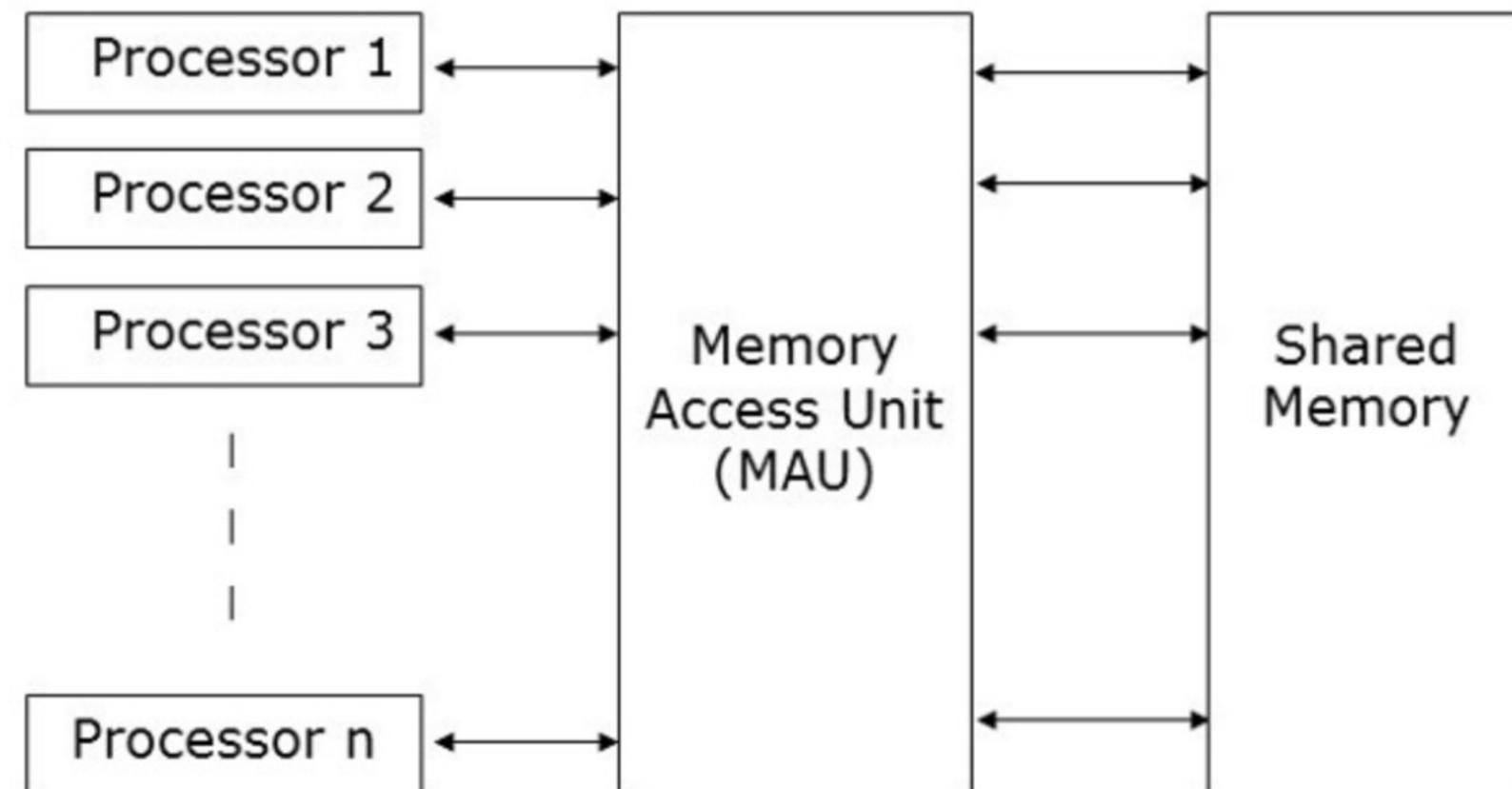


Modèles hybrides

- Un **modèle d'algorithme hybride** est nécessaire lorsqu'un **seul modèle est insuffisant** pour représenter un problème.
- Un **modèle hybride** peut être composé de plusieurs modèles appliqués soit **hiérarchiquement** ou **séquentiellement** à différentes phases d'un algorithme parallèle.

Parallel Random Access Machines (PRAM)

- **Parallel Random Access Machines (PRAM)** est un modèle, où plusieurs processeurs similaires sont attachés à un seul bloc de mémoire.
- Les **processeurs** peuvent **communiquer entre eux** par la seule **mémoire partagée**.



Parallel Random Access Machines (PRAM)

- Plusieurs processeurs peuvent souhaiter effectuer en même temps des **opérations indépendantes** sur un certain jeu de données.
- Cela peut entraîner un **accès simultané au même emplacement de mémoire par différents processeurs**.

Comment contrôler l'accès concurrent aux données de la mémoire partagée de manière cohérente et sans blocage ?

Parallel Random Access Machines (PRAM)

Pour résoudre ce problème,
les contraintes suivantes ont été appliquées sur le modèle :

- ✓ **Lecture exclusive** et **Écriture exclusive** (EREW) - Les processeurs **ne sont pas autorisés à lire ou à écrire dans le même emplacement de mémoire en même temps.**
- ✓ **Lecture exclusive** et **Écriture simultanée** (ERCW) - Les processeurs **ne sont pas autorisés à lire simultanément le même emplacement de mémoire, mais sont autorisés à écrire sur le même emplacement de mémoire en même temps.**
- ✓ **Lecture simultanée** et **Écriture exclusive**(CREW) - Tous les processeurs sont **autorisés à lire à partir du même emplacement de mémoire en même temps, mais ne sont pas autorisés à écrire sur le même emplacement de mémoire en même temps.**
- ✓ **Lecture simultanée** et **Écriture simultanée** (CRCW) - Tous les processeurs sont **autorisés à lire ou à écrire dans le même emplacement de mémoire en même temps.**

Implémentation du modèle PRAM

Il existe de nombreuses méthodes pour implémenter le modèle **PRAM**, mais les plus connues sont:

- Modèle de mémoire partagée
- Modèle de passage de message
- Modèle parallèle de données

Mémoire partagée (1)

- Met l'accent sur le **parallélisme de contrôle** et non sur le parallélisme des données.
- **Plusieurs processus s'exécutent sur différents processeurs indépendamment, mais ils partagent un espace mémoire commun.**
- En raison de toute activité du processeur, s'il y a un changement dans n'importe quel emplacement de la mémoire, il est visible pour le reste des processeurs.

➤ **Problème: Accès concurrent à la mémoire**

➤ Exemple: plusieurs processeurs écrivent et y lisent en même temps

Mémoire partagée (2)

- **Problème:** Accès concurrent à la mémoire

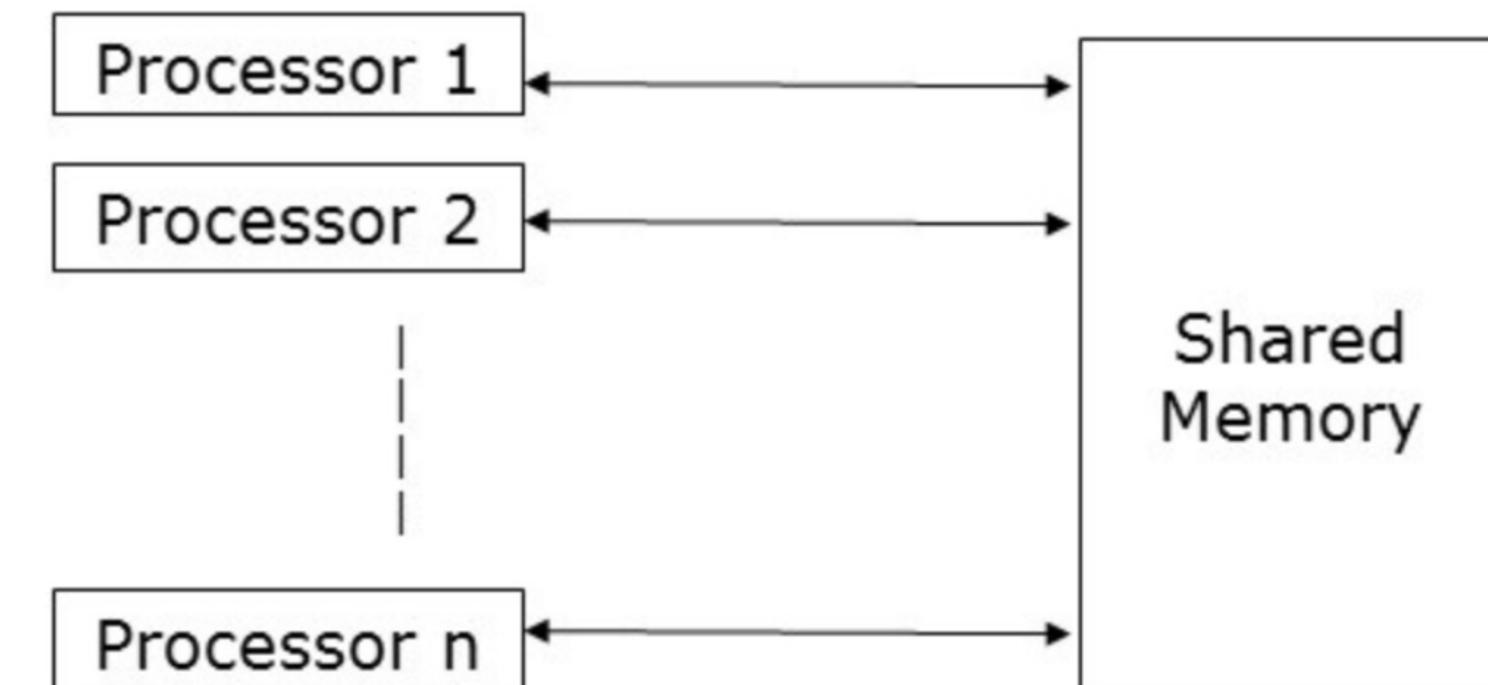
- **Exemple:** plusieurs processeurs écrivent et y lisent en même temps. Peut provoquer des incohérences sur les données.

- **Solution:** Mettre en place des *mécanismes de contrôle*

- Verrouillage
- Séaphore

- **Exemples:**

- SolarisTM threads [Solaris]
- POSIX threads [Linux]
- Win32 threads [Windows]
- JavaTM threads



Mémoire partagée (3)

■ Avantages

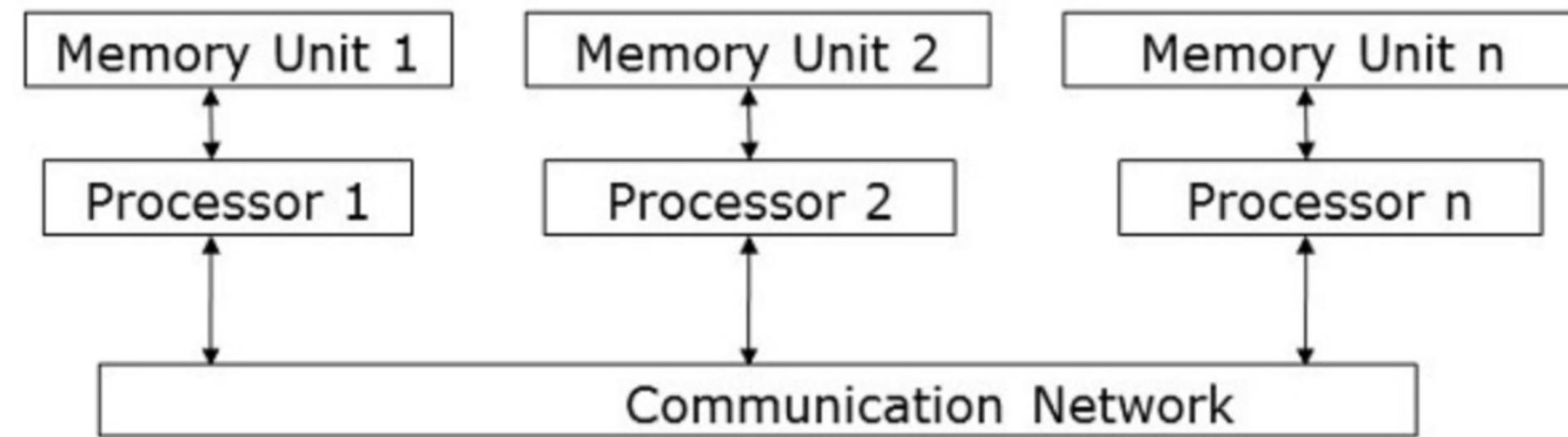
- Adressage globale
- Accès rapide aux données partagées
- Pas besoin de spécifier la communication
- Facile à implémenter

■ Inconvénients

- Absence de portabilité
- Gestion locale des données difficile

Modèles de messages (1)

- Modèles le plus utilisé en **programmation parallèle**
- Chaque processeur a une **mémoire locale**
- Les processeurs échangent des données via un **réseau de communication**
- Les processeurs utilisent des **bibliothèques** pour la communication entre eux



Modèles de messages (2)

Trois grandes méthodes de communication par message :

- Communication point à point
- Communication collective
- Interface de passage de message

Modèles de messages (3)

Communication point à point

- Forme la plus simple de **passage de message**.
- Un message peut être envoyé du processeur **émetteur** à celui **récepteur** via deux modes :
 - **Mode synchrone** - Le message suivant est envoyé uniquement après avoir *reçu une confirmation que son message précédent a été envoyé*, pour maintenir la séquence du message.
 - **Mode asynchrone** - Pour envoyer le message suivant, il *n'est pas nécessaire de recevoir la confirmation de la réception du message précédent*.

Modèles de messages (4)

Modèles de communication collective

Plus de deux processeurs sont concernés pour le passage des messages.

- **Mode barrière** - Possible si tous les processeurs inclus dans la communication exécutent un bock de code particulier (connu sous le nom de bloc de barrière) pour le passage du message.
- **Broadcasting** – Le **broadcasting (ou diffusion)** est de deux types
 - **One-to-All** - un processeur envoie le même message à tous les autres processeurs.
 - **All-to-All** - tous les processeurs envoient un message à tous les autres processeurs.

Modèles de messages (5)

▪ Avantages

- Fournit un faible contrôle du parallélisme
- Portabilité
- Moins susceptible aux erreurs
- Moins de coût dans la synchronisation parallèle et la distribution de données

▪ Inconvénients

- Nécessite généralement un coût logiciel plus élevé

Interface de passage de message

Il existe de nombreuses **bibliothèques** pour la **transmission de messages** entre processus.

Ici, nous discuterons des deux bibliothèques de messagerie les plus utilisées

- **Interface de passage de message (MPI)**
- **Machine virtuelle parallèle (PVM)**

Interface de passage de messages (1)

- **MPI:** Norme universelle pour fournir une communication entre tous les processus simultanés dans un système de mémoire distribuée.
- La plupart des plateformes informatiques couramment utilisées fournissent au moins une implémentation d'interface de passage de message (MPI).
- Implémenté comme une **collection de fonctions prédéfinies** appelées bibliothèque et peuvent être invoquées à partir de langages telles que C, C++, Fortran, etc.

Interface de passage de messages (2)

▪ Avantages

- Fonctionne uniquement sur des architectures de mémoire partagée ou des architectures de mémoire distribuée
- Chaque processeur possède ses propres variables locales
- Par rapport aux gros ordinateurs de mémoire partagée, les ordinateurs de mémoire distribuée sont moins chers

▪ Inconvénients

- D'autres modifications en termes de programmation sont nécessaires pour exprimer l'algorithme parallèle
- Parfois difficile à concevoir et à contrôler
- Ne fonctionne pas bien dans un réseau de communication avec un grand nombre de nœuds

Machine virtuelle parallèle (PVM)

- Une **PVM** est un **système de transmission de message**, conçu pour **connecter plusieurs machines hôtes hétérogènes** pour former **une seule machine virtuelle**.
- Elle gère les taches suivantes:
 - le **routage des messages**,
 - la **conversion** des données,
 - la **planification des tâches** dans le réseau d'architectures informatiques incompatibles.

Machine virtuelle parallèle (PVM)

Domaines d'Applications

- Grands problèmes de calcul tels que
 - Les études de supraconductivité,
 - Les simulations de dynamique moléculaire
 - Les algorithmes matriciels

Exemples

- Grilles de calcul
- Data centers

Caractéristiques d'une PVM

- Très facile à installer et à configurer;
- Plusieurs utilisateurs peuvent utiliser une **PVM** en même temps;
- Un utilisateur peut exécuter plusieurs applications;
- Supporte C, C++, Fortran;
- Pour une série donnée d'un programme PVM, les **utilisateurs peuvent sélectionner le groupe de machines**;
- Supporte une architecture hétérogène.

Modèles parallèle de données

- Consiste à effectuer des **opérations simultanément sur un ensemble de données**
- L'ensemble de **données** est **organisé en une structure** comme un tableau, un hypercube, etc.
- Les processeurs effectuent des **opérations collectivement sur la même structure de données**.
- **Chaque tâche est effectuée sur une partition différente** de la même structure de données.
- Applicables qu'à certains algorithmes