REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mouloud MAMMERI de Tizi-Ouzou
Faculté de Génie Electrique et d'Informatique
Département d'Informatique

Mémoire de Fin d'étude

En vue de l'obtention du diplôme de Master en informatique

Option : Conduite de Projets Informatiques

Thème :

Coopération de plusieurs processus dans un environnement virtuel

Proposé & dirigé par :

Mr: DIB.A

Réalisé par :

Melle : AMMARI Tassadit

Melle : MESSAOUDI Kahina

Promotion 2011 / 2012

REMERCIEMENTS

C'est à « ALLAH », que nous adressons toute notre gratitude en premier lieu.

Nos remerciements, les plus vifs, notre profonde gratitude et nos respects s'adressent à notre promoteur Monsieur Ahmed DIB pour avoir accepté de nous encadrer, pour les conseils et orientations tant précieux qu'il nous a prodigués durant ce mémoire.

Nos remerciements vont également à l'honorable jury qui a consenti à juger notre travail.

Nous ne remercierons jamais assez nos chères familles pour

Nous avoir toujours encouragées et nous avoir inculquées le goût du savoir et de l'ambition.

Nous remercieront aussi l'ensemble de la famille enseignante du département informatique pour nous avoir formé durant toutes ces années et tous ceux qui ont contribué de prés ou de loin pour la réalisation de ce travail modeste.

Dédicaces

Je dédie ce modeste travail à:

Ma très chère mère

Mon très cher père

Mes frères et sœurs et belles sœurs

Ma chère amie Kahina et sa famille

Mes amis, en particulier le groupe GJ

Tassadit

Dédicaces

Je dédie ce modeste travail à:

Mes très chers parents et grands-parents

Mon frère et mes sœurs ainsi qu'à toute ma Famille

Mon cher bien aimé et sa famille

Ma chère amie Tassadit(Kahina) et sa famille

Tous mes amis(es)

Kahina

TABLE DES MATIERES

Table des matières

Liste des figures

Liste des tableaux

INTRODUCTION GENERALE

Introduction générale

Depuis plus d'une décennie, les technologies de l'informatique et de la communication ont fait une percée majeure dans nos sociétés, touchant des domaines de notre vie immuables à tout changement jusqu'à un passé proche. Reste que le grand bénéficiaire de ces technologies est sans conteste les différentes branches scientifiques, la médecine, l'astronomie, la météorologie, l'ingénierie … ont trouvé des outils précieux leurs assurant de grands résultats dans une matrice coût-temps très avantageuse.

Dans une telle quête pour plus de performances et moins de coûts et du temps, les architectures parallèles sont sollicitées pour résoudre une large classe de problèmes, surtout les architectures à mémoire distribuée (ensemble de machines interconnectées par un réseau), qui réalisent un rapport coût/performance à la portée de la majorité des groupes de recherche et autres organismes et même des individus.

Dans un tout autre secteur de l'informatique, La virtualisation est une technologie de plus en plus utilisée lors de la définition de nouvelles architectures informatiques mais aussi afin d'aider à l'évolution de ces dernières. La virtualisation peut aboutir à la mise en place de solutions économiques pour tous les types d'entreprises, quelle que soit leurs tailles. Cette technologie permet de répondre à des problématiques au niveau de la haute disponibilité ou encore de la sécurité.

Dans cette application qu'on se propose de réaliser, on reliera parallélisme et virtualisation, notre présent travail consiste à paralléliser la compression de fichiers sur un réseau de machines virtuelles. Pour arriver à terme de ce travail, nous avons organisé notre mémoire en trois chapitres :

- Dans le premier chapitre, nous parlerons sur la virtualisation, ses différents types et techniques, des réseaux virtuels et leurs modes.
- Dans le deuxième chapitre, nous parlerons sur le parallélisme, ses différentes architectures, ses différents types ainsi que quelques notions sur les environnements parallèles, bibliothèques et outils pour le programmeur.
- Dans le troisième et dernier chapitre**,** ça sera le tour de la conception et de la mise en œuvre de notre application où l'on expliquera en détail toutes les démarches qu'on a entreprises pour converger à notre application finale.

Chapitre I

LA VIRTUALISATION

Introduction

La virtualisation est une technologie de plus en plus utilisée lors de la définition de nouvelles architectures informatiques mais aussi afin d'aider à l'évolution de ces dernières.

Encore peu connue et peut être complexe au départ, la virtualisation peut aboutir à la mise en place de solutions économiques pour tous les types d'entreprises, quelle que soit leurs tailles. Cette technologie permet de répondre à des problématiques au niveau de la haute disponibilité ou encore de la sécurité.

La virtualisation peut donc être envisagée dans le cadre du déploiement de nouveaux services des Systèmes d'Information ou pour la consolidation de ces derniers.

I.1-La virtualisation

I.1.1. Définition de la virtualisation

La virtualisation recouvre l'ensemble des techniques matérielles et/ou logicielles qui permettent de faire fonctionner sur une seule machine plusieurs systèmes d'exploitation, plusieurs instances différentes et cloisonnées d'un même système ou plusieurs applications, séparément les uns des autres, comme s'ils fonctionnaient sur des machines physiques distinctes. Il s'agit donc d'utiliser une seule machine physique en remplacement de plusieurs et d'utiliser les possibilités offertes par la virtualisation pour démultiplier le nombre de machines virtuelles.

En bref, la virtualisation est une méthode faisant fonctionner un ou plusieurs systèmes d'exploitation invités dans des machines virtuelles, au-dessus d'un système d'exploitation hôte **[01]**.

I.1.2. Types de virtualisation

Plusieurs types de virtualisation existent, mais toutes fonctionnent selon un même principe :

1. un système d'exploitation principal (appelé *système d'exploitation hôte*) est installé dans l'ordinateur et sert de système d'accueil à d'autres systèmes d'exploitation.
2. dans le système d'exploitation hôte, un logiciel de virtualisation (appelé *hyperviseur*) est installé. Celui-ci crée des environnements clos, isolés, avec des ressources bien précises : ces environnements clos sont appelées des *machines virtuelles*.
3. d'autres systèmes d'exploitation (appelés *systèmes d'exploitation invités*) peuvent alors être installés dans des machines virtuelles. Leur instance est totalement isolée des autres systèmes hôtes et systèmes invités **[01]**.

I.1.3. Les techniques de virtualisation

I.1.3.1. La virtualisation au niveau du noyau

- ✓ Elle fait croire à la présence de plusieurs machines.
- ✓ Il n'est pas possible d'utiliser des noyaux différents en même temps.
- ✓ Les performances sont excellentes. **[02]**

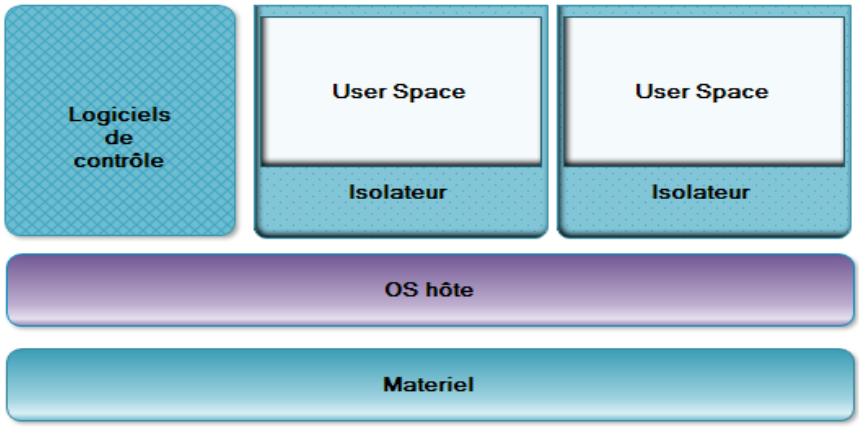

Figure I.1 : la virtualisation au niveau du noyau [02]

I.1.3.2. La virtualisation par l'emploi d'un émulateur

- ✓ Elle simule un processeur ou une machine complète. (mémoire, disque, réseau, périphériques, …etc.).
- ✓ Il est ainsi possible d'exécuter des applications prévues pour d'autres architectures.
- ✓ Les performances sont médiocres. **[02]**

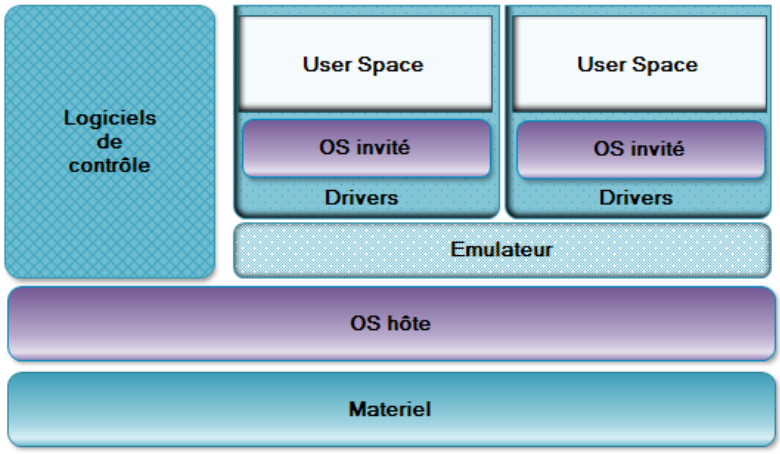

Figure I.2 : La virtualisation par l'emploi d'un émulateur [02]

I.1.3.3. La virtualisation par l'utilisation d'un hyperviseur de machines virtuelles

- ✓ Les machines virtuelles fonctionnent directement avec le matériel sans interception des instructions.
- ✓ Il est possible de faire cohabiter des noyaux et des systèmes d'exploitation différents.
- ✓ Les performances sont excellentes. **[02]**

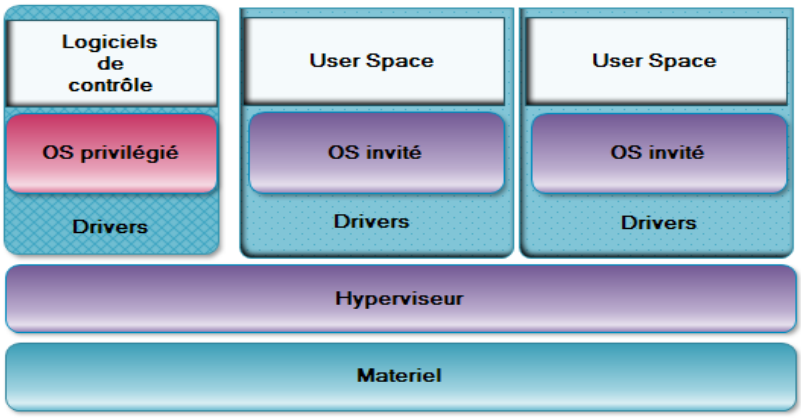

Figure I.3 : la virtualisation par emploi d'un hyperviseur [02]

I.1.3.4. La virtualisation au niveau du processeur

- ✓ Il n'est plus nécessaire de surveiller et de traduire les instructions des machines virtuelles.
- ✓ Les performances de la virtualisation logicielle sont nettement améliorées. **[02]**

I.1 .4. Avantages et inconvénients de la virtualisation

La virtualisation présente de nombreux intérêts :

I.1.4.1. Avantages

- ➢ Utilisation optimisée des ressources des serveurs récents (architectures processeurs multi-cœurs). répartition des machines virtuelles sur les machines physiques en fonction des charges respectives.
- ➢ Installation, déploiement et migration facile des machines virtuelles d'une machine physique à une autre.
- ➢ Economie sur le matériel par mutualisation (consommation électrique, entretien physique, monitoring, support, compatibilité matérielle, etc.).
- ➢ Sécurisation et/ou isolation d'un réseau (arrêt des systèmes d'exploitation virtuels, mais pas des systèmes d'exploitation hôtes qui sont invisibles pour l'attaquant).
- ➢ Isolation des différents utilisateurs simultanés d'une même machine (utilisation de type site central).
- ➢ Allocation dynamique de la puissance de calcul en fonction des besoins de chaque application à un instant donné.
- ➢ La multiplication des services.
- ➢ Limiter le nombre de machines physiques.
- ➢ Support de systèmes d'exploitation hétérogènes.

- Mécanismes avancés de clonage, de sauvegarde et de restauration des machines virtuelles. **[02]**

I.1.4.2. Inconvénients

La virtualisation pose quelques problématiques :

- Concentration des services → augmentation de la criticité de l'infrastructure**.**
- Concentration des compétences techniques.
- Gestion des mises à jour des multiples serveurs virtualisés
- Choix entre un **stockage local** ou **centralisé**.
- Utilisation des mécanismes de **snapshots**.
- Mise en œuvre d'une politique de sauvegarde (les machines virtuelles ou les données essentielles).
- Mise en place de **clusters** pour gérer des grappes de serveurs et assurer la disponibilité des services **[02]**.

I.2.Machine virtuelle

Une machine virtuelle est un ordinateur logiciel qui exécute un système d'exploitation et des applications. La machine virtuelle se compose d'un ensemble de fichiers de spécification et de configuration, elle est secondée par les ressources physiques d'un hôte. Chaque machine virtuelle a des périphériques virtuels qui fournissent la même fonction que le matériel physique et présentent un intérêt supplémentaire en termes de portabilité, maniabilité et sécurité.

Une machine virtuelle se compose de plusieurs types de fichiers stockés sur un périphérique de stockage compatible.

Il en existe beaucoup de logiciels simples permettant aujourd'hui de créer facilement des machines virtuelles, tels que VirtualPC, VirtualBox et Vmware.

Chaque outil de virtualisation implémente une ou plusieurs de ces notions :

- couche d'abstraction matérielle et/ou logicielle.

- système d'exploitation hôte (installé directement sur le matériel).

- systèmes d'exploitations (ou applications, ou encore ensemble d'applications) « virtualisés » ou « invités ».
- partitionnement, isolation et/ou partage des ressources physiques et/ou logicielles.
- images manipulables : démarrage, arrêt, gel, clonage, sauvegarde et restauration, sauvegarde de contexte, migration d'une machine physique à un autre réseau virtuel : réseau purement logiciel, interne à la machine hôte, entre hôte et invités. [**01**]

I.2.1. Vmware

I.2.1.1.Définition de Vmware

Est un logiciel de virtualisation qui permet la création d'une ou plusieurs machines virtuelles au sein d'un même système d'exploitation (généralement Windows ou Linux), ceux-ci pouvant être reliés au réseau local avec une adresse IP différente, tout en étant sur la même machine physique (machine existant réellement). Il est possible de faire fonctionner plusieurs machines virtuelles en même temps, la limite correspondant aux performances de l'ordinateur hôte. La version Linux présente l'avantage de pouvoir sauvegarder les fichiers de la machine virtuelle (*.vmsd) pendant son fonctionnement.

Le logiciel Vmware permet de faire fonctionner un système d'exploitation virtuel sur une machine (en plus du système présent) mais non de le créer.
Il a été développé par VMware Inc et il a trois lignes de produits, VMware Workstation, GSX et ESX. **[03]**

- **VMware Workstation:** permet

- La construction de réseaux complexes, développement, test et déploiement des nouvelles applications, sur une seule et même machine.
- La configuration et test des applications multi-tiers, les mises à jour applicatives sur un seul PC.

- La restauration, l'archivage et partage des environnements de test, et ainsi réduire les temps de préparation des machines.
- Utilisation de la portabilité des machines virtuelles pour faciliter le partagedes environnements de développement ou de test.
- Ajout ou modification des systèmes d'exploitation sans repartitionner les disques.
- Utilisation des systèmes d'exploitation les plus récents tout en conservant les applications les plus anciennes sur le même PC.
- Disposition de machines rapidement "propres" pour les formations, avec plusieurs environnements complets rapidement disponibles.
- Accélération de la résolution des problèmes de support avec une "bibliothèque" de machines virtuelles préconfigurées. **[03]**

❖ **Fonctionnement de Vmware Workstation**

VMware Workstation permet de faire fonctionner plusieurs systèmes d'exploitation en même temps ainsi que les applications associées sur une même machine physique. Ces différents systèmes d'exploitation et applications sont isolés dans des machines virtuelles sécurisées qui coexistent sur la même machine physique. La couche de virtualisation de VMware va mapper les ressources matérielles de la machine vers les ressources virtuelles de l'environnement virtuel. Ainsi, chaque machine virtuelle a son propre CPU, sa propre mémoire, ses propres disques, ses propres périphériques d'entrées-sorties. **[03]**

❖ **VMware GSX Server**

C'est une application serveur qui est conçue pour faire tourner des serveurs virtuels. La connexion à un serveur virtuel se fait par l'intermédiaire d'une console VMware depuis un poste Linux ou Windows. Lorsque la console VMware est fermée, le serveur continue de fonctionner en tâche de fond. **[03]**

❖ **VMware ESX**

La grande différence de ce produit par rapport aux autres est qu'il s'installe directement sur la couche matérielle. Il n'est pas nécessaire d'installer un système d'exploitation « hôte » pour installer VMware ESX.

VMware ESX est donc un hyperviseur qui permet une gestion plus précise des ressources pour chaque machine virtuelle et de meilleures performances. La solution VMware ESX est la solution la plus industrielle de la gamme. Vmware ESX est un système d'exploitation ou hyperviseur basé sur une distribution RHEL5 (RedHatLinux 5) modifiée. Cet hyperviseur est composé de plusieurs modules :

- **VMKERNEL** : Ce module « noyau » est le "cerveau" de VMware ESX ; il permet de gérer et de hiérarchiser l'ensemble des ressources matérielles (mémoire, CPU, disques, réseaux) en fonction de chaque serveur. De plus c'est le noyau qui est chargé de toute la gestion des ressources physiques pour ESX.
- **SERVICE CONSOLE** : Ce module permet la gestion de l'hyperviseur en mode commande. Cette console sert à lancer certaines commandes inaccessibles depuis l'interface graphique ou encore de parcourir les dossiers dans lesquels sont stockées les machines virtuelles. Enfin elle peut permettre de collecter des informations de débogage sur les machines virtuelles ou sur le serveur ESX. **[03]**

I.2.1.2. Caractéristique de Vmware

- **Isolation :** Les machines sont indépendantes les unes des autres, si une meurt, les autres continuent de fonctionner.
- **Encapsulation :** Chaque machine est physiquement représentée par un ensemble de fichiers, y compris le disque dur et la définition matérielle (cartes Réseau, Vidéo etc…).
- **Compatibilité :** Chaque machine virtuelle, voit le même PC standard.
- **Indépendance du matériel :** Chaque machine virtuelle ne voit pas le matériel physique de la station host, vu qu'elle ne voit qu'un PC standard.

- **Mutualisation et optimisation des ressources :** Chaque machine virtuelle se voit dotée au départ d'un certain nombre de ressources : mémoire vive, nombre de CPU, espace disque, cartes réseaux. **[03]**

I.2.1.3. Configuration de VMware

La configuration de **VMware** est très simple. Une fois installé, avant de créer le nouveau virtuel Host, il est nécessaire de mettre en place des réseaux virtuels.
Il y a plusieurs manières pour configurer les réseaux d'une VMWARE : Bridged, NAT et Host Only. **[03]**

- **Mode Bridged:**

Par défaut le réseau virtuel est configuré en bridge, dans ce mode de configuration les deux machines (Physique et virtuelle) apparaissent comme étant deux machines indépendantes, les cartes réseau de la machine physique et de la machine virtuelle ont des adresses IP différentes. La machine virtuelle a l'accès direct au LAN exactement comme n'importe quelle autre machine sur le réseau et on peut soit fixer l'@ IP manuellement ou utiliser une @IP attribuée par le DHCP. **[03]**

- **Mode Nat:**

Permet de partager l'@IP de la machine hôte(Physique). Via la carte réseau, le système Nat transmet les paquets de la machine virtuelle en utilisant l'@IP de la machine hôte. **[03]**

- **Mode Host only**

Les deux modes précédents nécessitent un LAN opérationnel et connecté. Dans le cas où on n'a pas ce réseau ou on veut garder la machine invisible dans le réseau on utilise le mode Host only. Via la carte réseau, la machine virtuelle et la machine hôte (physique) communiquent entre elles. **[03]**

I.2.2. Virtual PC

I.2.2.1. Définition de Virtuel PC

Virtual PC est un logiciel propriétaire gratuit d'émulation et de virtualisation qui permet de créer des machines virtuelles sur lesquelles on peut installer un système d'exploitation et ceci peut être utile dans plusieurs cas. Idéal pour tester un système d'exploitation sans être obligé de repartitionner le disque dur, l'inconvénient de ce logiciel, c'est qu'il lance un nouvel système d'exploitation en même temps que l'ancien, donc toutes les ressources du PC seront partagées par les deux systèmes d'exploitation. Par conséquent, on ne peut pas avoir le maximum des performances du PC sur le nouveau système d'exploitation **[Url 01].**

I.2.2.2. Les extensions (compléments) de machines virtuelles

Pour faciliter l'échange et le partage de fichiers, de dossiers et de données entre l'hôte et le système hébergé, Virtual PC supporte l'installation des *Additions de machine virtuelle*. Celles-ci sont installées dans le système hébergé et fournissent les fonctionnalités suivantes:

- de meilleures performances sur le système hébergé.
- l'utilisation intégrée et améliorée de la souris.
- des drivers vidéo optimisés.
- la synchronisation horaire avec l'hôte.
- Le partage du presse-papier.
- des possibilités de "Copier&coller". **[Url 01]**

I.2.3. VirtualBox

I.2.3.1. Définition de VirtualBox

VirtualBox est un logiciel de virtualisation de systèmes d'exploitation. En utilisant les ressources matérielles de l'ordinateur (système hôte), VirtualBox permet la création d'un ou de plusieurs ordinateurs virtuels dans lesquels s'installent d'autres systèmes d'exploitation (systèmes invités).

Les *systèmes invités* fonctionnent en même temps que le *système hôte*, mais seul ce dernier a l'accès directement au véritable matériel de l'ordinateur. Les *systèmes invités*

exploitent du matériel générique, simulé par un « faux ordinateur » (machine virtuelle) créé par VirtualBox.

VirtualBox permet de faire fonctionner plus d'un système d'exploitation en même temps en toute sécurité. En effet, les *systèmes invités* n'interagissent pas directement avec le *système hôte*, et n'interagissent pas entre eux. Le champ d'action des *systèmes invités* est confiné, limité à leur propre machine virtuelle **[04]**.

I.2.3.2. Configuration requise

Afin de pouvoir utiliser VirtualBox sur une machine, il faut posséder au minimum la configuration suivante.

- **Processeur** : 1 GHz (simple cœur)
- **Mémoire vive** : 513 Mio
- **Disque dur** : 116 Mio d'espace disque libre
- **Carte graphique** : peu importe

Ceci étant la configuration minimale, elle n'est pas forcément adaptée à toutes les utilisations.
En effet, pour pouvoir utiliser certains systèmes d'exploitation ou logiciels *via* VirtualBox, il faudra une configuration plus solide. **[04]**

I.2.3.3.Caractéristiques de VirtualBox

- ✓ VirtualBox fonctionne sur un grand nombre de 32-bit et 64-bit des systèmes d'exploitation hôtes.

- ✓ VirtualBox nécessite un système d'exploitation existant (doit être installé). On peut donc courir aux côtés des applications existantes sur cet hôte.
- ✓ Pour une très large mesure, VirtualBox est fonctionnellement identique sur toutes les plates-formes d'accueil, et le même fichier et les formats d'image sont utilisés. Cela permet d'exécuter des machines virtuelles créées sur un hôte sur un autre hôte avec un système d'exploitation hôte différent, par exemple, on peut créer une machine virtuelle sur Windows, puis l'exécuter sous Linux.

- ✓ En outre, les machines virtuelles peuvent facilement être importés et exportés en utilisant le format OVF (Open Virtualisation ou Importation et exportation de machines virtuelles).
- ✓ Pas de virtualisation matérielle requise. Pour de nombreux scénarios, VirtualBox n'exige pas que le processeur disposé intègre dans les nouveaux matériels. Contrairement à beaucoup d'autres solutions de virtualisation, on peut utiliser VirtualBox, même sur du matériel ancien, où ces caractéristiques ne sont pas présentes. **[04]**

I.2.3.4. Les versions de Virtualbox

Il existe deux versions de VirtualBox :

I.2.3.4.1. Version PUEL (Personal Use and Evaluation License)

Cette version, sous licence PUEL, est complète et gratuite dans le cadre d'une utilisation privée, académique ou pour essayer le logiciel. Les entreprises doivent acquérir des licences commerciales pour pouvoir utiliser cette version.

I.2.3.4.2. Version OSE (Open Source Edition)

Cette version, gratuite et sous licence GNU GPL V2, est équivalente à la version complète, à l'exception de certaines fonctionnalités qui ciblent plutôt les entreprises, afin d'encourager ces dernières à acquérir la version PUEL. A noter tout de même que cette version ne gère à priori pas les périphériques USB. Actuellement, VirtualBox fonctionne sur Windows**,** Linux, Macintosh et Solaris, *il* accueille et soutient un grand nombre de systèmes d'exploitation invités**,** *y compris* mais non limité à Windows.

VirtualBox est activement développé avec mises à jour fréquentes et dispose d'une liste toujours croissante de fonctionnalités. **[04]**

I.2.3.5. Evaluation de VirtualBox

- ➢ **Installation :**

Il est possible d'utiliser VirtualBox pour virtualiser tout système d'un PC. VirtualBox est relativement gourmand en RAM, bien plus qu'en CPU (Il faudra penser à

dimensionner généreusement les machines pour démarrer simultanément deux machines virtuelles)

En ce qui concerne son ergonomie, elle est fortement inspirée des aspects agréables des habitudes "à la Windows", avec un assistant de création de machines virtuelles utilisable avec un recours minimum aux aides en ligne. Démarrer, arrêter ou gérer plus généralement des machines virtuelles est aussi d'une simplicité bien pensée. Cependant, cette ergonomie intéressante gagnerait à s'équiper aussi de fonctionnalités de contrôle à distance, qui manqueront aux administrateurs. **[04]**

I.3 Les réseaux virtuels

I.3.1. Définition de réseaux virtuels

Les réseaux virtuels sont apparus comme une nouvelle fonctionnalité dans l'administration réseau avec le développement des commutateurs.

La notion de réseau virtuel est un concept qui permet de réaliser des réseaux de façon indépendante du système de câblage. Ces réseaux permettent de définir des domaines de diffusion restreints, cela signifie qu'un message émis par une station du réseau virtuel ne pourra être reçu que par les stations de ce même réseau virtuel.

Un réseau virtuel, est donc, un regroupement logique, et non physique, de plusieurs stations. Pour réaliser ce regroupement, on intervient directement, par voie logicielle, sur le ou les éléments actifs que sont les commutateurs.

En bref, un réseau virtuel est un ensemble de ressources interconnectées et regroupées logiquement indépendamment de leur localisation géographique **[05].**

I.3.2. Les objectifs des réseaux virtuels

- Ils permettent de supporter les organisations virtuelles, en rendant l'appartenance à un groupe indépendante de sa position géographique.
- Ils optimisent la bande passante, en réalisant des réseaux disjoints, donc en réalisant des domaines de collision disjoints.
- Ils simplifient l'administration, en utilisant des commandes centralisées pour gérer un réseau plutôt que des interventions dans les armoires de brassage.

- Ils améliorent la sécurité, en créant des règles de communication inter-réseaux.

[05]

I.4. Les modes de réseaux dans virtualBox

Il possède 5 modes réseau:

I.4.1. Network Address Translation (NAT)

NAT est le moyen le plus simple d'accéder à un réseau externe à partir d'une machine virtuelle. Habituellement, il ne nécessite aucune configuration sur le réseau hôte et le système invité. Pour cette raison, il est le mode par défaut en réseau dans VirtualBox.

Une machine virtuelle activée avec NAT agit comme un véritable ordinateur qui se connecte à Internet via un routeur. Le "routeur", dans ce cas, est le moteur de mise en réseau de VirtualBox, qui mappe le trafic vers la machine virtuelle de façon transparente. L'inconvénient du mode NAT est que, tout comme un réseau privé derrière un routeur, la machine virtuelle est invisible et inaccessible depuis Internet à l'extérieur.

Les trames réseau envoyées par le système d'exploitation invité sont reçues par le moteur de VirtualBox NAT, qui extrait les données TCP / IP et les renvoie à l'aide du système d'exploitation hôte. Pour une application sur l'hôte ou à un autre ordinateur sur le même réseau que l'hôte, ces données ont été envoyées par l'application VirtualBox sur l'hôte, en utilisant une adresse IP appartenant à l'hôte. VirtualBox à l'écoute des réponses aux paquets envoyés, les reconditionne et les renvoie à la machine invitée sur son réseau privé.

La machine virtuelle reçoit son adresse réseau et la configuration sur le réseau privé à partir d'un serveur DHCP intégré dans VirtualBox. L'adresse IP ainsi attribuée à la machine virtuelle est habituellement sur un réseau complètement différent de celui de l'hôte **[Url 02]**.

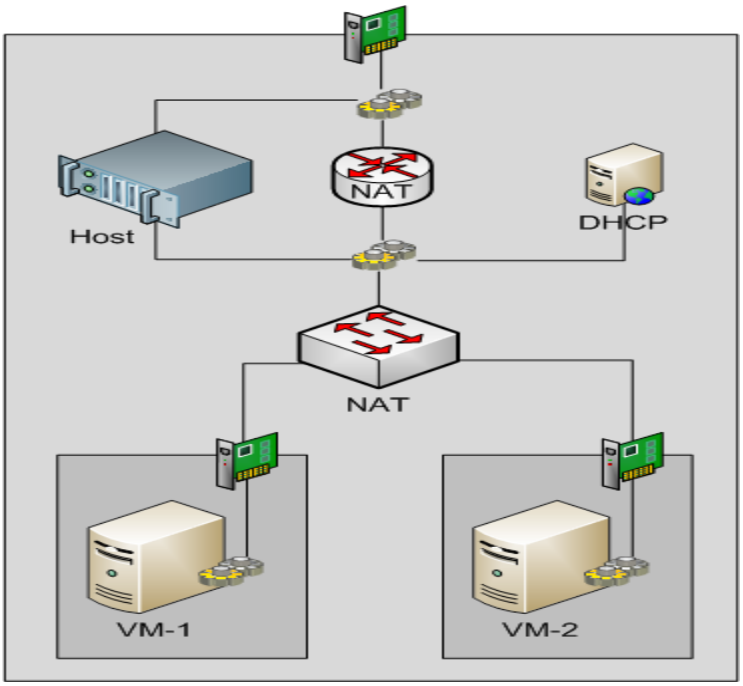

Figure I.4 : Mode Nat [Url 03]

I.4.2. Réseau Bridged

Avec un réseau ponté, VirtualBox utilise un pilote de périphérique sur le système *hôte* qui filtre les données de la carte réseau physique. Ce pilote est donc appelé un «filet de filtre" du pilote. Ceci permet à VirtualBox d'intercepter des données du réseau physique et injecter des données en elles, créant une nouvelle interface réseau dans le logiciel. Quand un client utilise une telle nouvelle interface logicielle, il ressemble au système hôte comme si les invités étaient physiquement connectés à l'interface en utilisant un câble réseau: l'hôte peut envoyer des données à l'invité par le biais de cette interface et recevoir des données qu'il contient. Cela signifie qu'on peut configurer le routage entre l'invité et le reste du réseau **[Url 02]**.

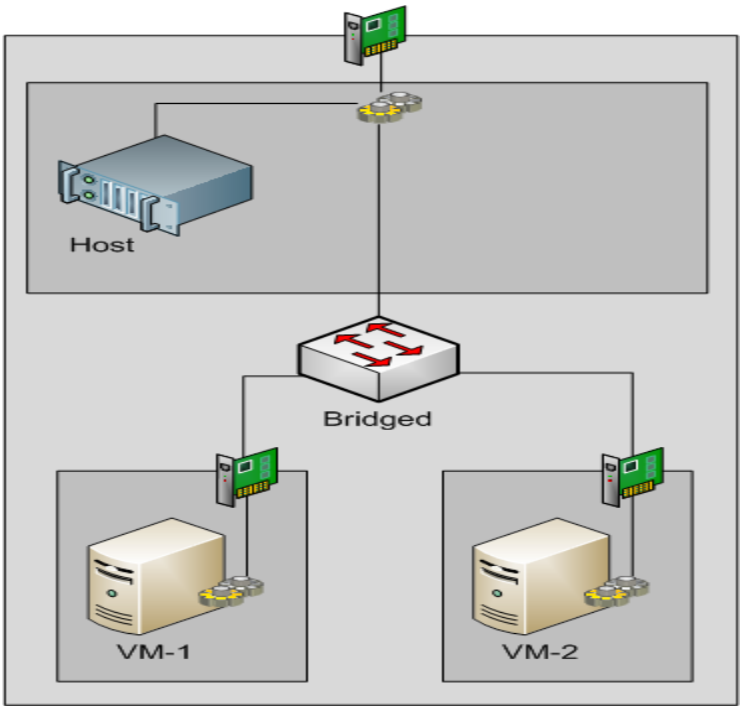

Figure I.5 : Mode Bridged [Url 03]

I.4.3. Mise en réseau interne

Le réseau interne est similaire au réseau ponté dont les machines virtuelles peuvent communiquer directement avec le monde extérieur. Cependant, le «monde extérieur» est limité à d'autres machines virtuelles sur le même hôte qui se connectent au même réseau interne.

Même si, techniquement, tout ce qui peut être fait en utilisant des réseaux internes peut également être fait en utilisant le réseau ponté, il ya des avantages de sécurité avec le réseau interne. En mode réseau ponté, tout le trafic passe par une interface physique du système hôte **[Url 02]**.

I.4.4. Réseau Host-only

Le réseau Host-only est un autre mode de mise en réseau qui a été ajouté avec VirtualBox. Il peut être considéré comme un hybride entre les modes de mise en réseau ponté et interne: comme avec le réseau ponté, les machines virtuelles peuvent parler les unes aux autres comme si elles étaient connectées par un commutateur Ethernet physique. De même, comme avec le réseau interne, Cependant, une interface réseau physique ne doit pas être présentée, et les machines virtuelles ne peuvent pas

parler au monde extérieur de l'hôte, car elles ne sont pas raccordées à une interface réseau physique. **[Url 02]**

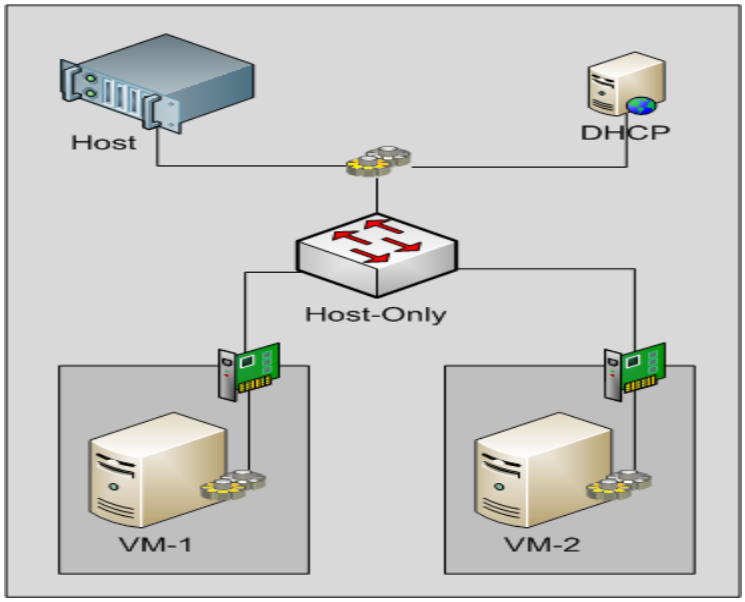

Figure I.6: Mode Host only [Url 03]

I.4.5. Mise en réseau tunnel UDP

Ce mode permet d'interconnecter des réseaux de machines virtuelles fonctionnant sur des hôtes différents.

Techniquement, cela se fait en encapsulant les trames Ethernet envoyées ou reçues par la carte réseau hôte dans un datagramme UDP / IP, puis les envoyer sur n'importe quel réseau disponible à l'hôte.

Le mode tunnel UDP a trois paramètres:

- **Source du port UDP**

Le port sur lequel l'hôte est à l'écoute. Les datagrammes arrivant sur ce port à partir de n'importe quelle adresse source seront transmis à la partie réception de la carte réseau invité.

- **Adresse de destination**

Adresse IP de l'hôte cible des données transmises.

- **Destination du port UDP**

Numéro de port auquel les données transmises sont envoyées.

Lorsqu'on relie deux machines virtuelles sur deux hôtes différents, leurs adresses IP doivent être inversées. Lorsqu'un seul hôte, les ports UDP source et destination doivent être inversés. **[Url 02]**

Conclusion

Dans ce chapitre, nous avons fait une étude d'ensemble sur la virtualisation, qui n'est pas une idée nouvelle. En fait, elle remonte aux débuts de l'informatique.

Pour rappel, la virtualisation consiste à exécuter un système d'exploitation dans une machine virtuelle, ce qui permet d'utiliser plusieurs systèmes sur la même machine, soit pour utiliser un système d'exploitation différent de l'hôte soit pour compartimenter de façon efficace les différents services d'une machine. Malgré ses avantages, la virtualisation reste une technologie assez mystérieuse pour le grand public. Il nous est donc apparu utile de faire un tour d'horizon des solutions de virtualisation disponibles actuellement et de leurs développements futurs.

Voilà que conclu notre rapide tour d'horizon de la virtualisation. Grâce à la multiplication du nombre des cœurs au sein des processeurs, on peut même imaginer que la virtualisation devienne une des tendances principales de l'informatique personnelle dans le futur proche. Sécurité, stabilité, polyvalence, interopérabilité, les avantages que chacun pourrait en retirer sont nombreux. Mais uniquement quand les performances offertes par les optimisations matérielles implantées par les fondeurs seront vraiment efficaces, suffisamment pour rendre l'utilisation des machines virtuelles aussi confortables que celles de machines réelles.

Chapitre II

LE PARALLÉLISME

Introduction

Depuis l'avènement de l'informatique dans le milieu du 20ème siècle, tous les domaines scientifiques sont quasiment touchés. Et de plus en plus, la puissance du calcul se fait ressentir au sein de la communauté scientifique. En effet, des applications telles que les simulations météorologiques ou biologiques nécessitent la résolution de systèmes des plus complexes et la quantité de données exploitées et celles générées est très importante. Ainsi, un ordinateur, aussi puissant soit-il, ne peut réaliser de telles tâches dans les normes exigées, c'est pourquoi le parallélisme s'est imposé comme une solution fiable et bon marché. De nos jours, et avec la technologie des cores (processeurs multi-cœurs), les processeurs graphiques (GPUs), les machines SMP (SymetricMultiProcessors), les grappes ou les grilles de calcul, le parallélisme gagnent encore du terrain, et malgré que beaucoup d'informaticiens ne soient pas familiarisés avec, l'avenir reste très promoteur.

II.1 Programmation parallèle

II.1.1 Définition

La programmation parallèle consiste à exécuter simultanément plusieurs programmes, qui coopèrent pour réaliser un but commun et/ou qui sont en compétition pour la possession de ressources. L'exécution parallèle peut se faire :

- sur une machine ne disposant que d'un seul processeur partagé.
- sur une machine ayant plusieurs processeurs, chacun prenant en charge l'un des programmes.
- sur plusieurs machines différentes, reliées par un réseau de communication. Chacune des machines pourra comprendre un ou plusieurs processeurs. La synchronisation et la communication de données entre les machines seront effectuées par des messages échangés**[Url 04].**

II.1.2 Intérêt de la programmation parallèle

Il existe plusieurs raisons qui rendent le parallélisme très avantageux, on peut citer :

- **Meilleure utilisation du processeur**, grâce à la multiprogrammation.
- **Animation des pages Web** : Dans les pages web, il y a souvent des petits dessins qui bougent, pour attirer l'attention sur un point particulier, ou bien des animations qui apportent des explications... là encore, il faut pouvoir interrompre l'animation pour cliquer où on veut. Donc l'animation doit être un programme parallèle à celui qui gère la page principale.
- **Les coûts réduits**, puisque au lieu d'utiliser un supercalculateur assez onéreux, il suffit d'opter pour l'utilisation de plusieurs ressources informatiques beaucoup moins chères.
- **Rapidité accrue d'exécution**sur un réseau et sur des machines possédant plusieurs processeurs : Certains programmes de calcul sont très longs à l'exécution. Mais s'ils peuvent se décomposer en plusieurs parties relativement indépendantes les unes des autres, alors il est envisageable de faire calculer chacune de ces parties (processus coopérants) par un processeur différent**[06]**.

II.1.3 Terminologie des systèmes concurrents

La caractéristique d'un système concurrent est qu'un ensemble d'actions se déroulant en même temps (sur des processeurs, machines, ou sites différents) sont liées entre elles au niveau logique en un seul système coordonné. Pour définir précisément de tels systèmes, nous définissons d'abord la notion de *fil d'exécution***[06].**

II.1.3.1 Fil d'exécution (tâche)

Un fil d'exécution est une suite logique d'actions résultant de l'exécution d'un programme. Un programme peut être constitué de plusieurs fils d'exécution ; on parle alors dans certains cas de *multiprogrammation* ou de *multithreading*.**[06]**

II.1.3.2 systèmes concurrents

Nous définissons un système concurrent comme un ensemble (pouvant varier dans le temps) de fils d'exécution qui interagissent pour parvenir à un but commun.

Notons qu'un système concurrent peut contenir plusieurs programmes. Suivant ces définitions, des exemples usuels de systèmes concurrents sont les machines multiprocesseurs et/ou parallèles, les grappes de PC, ou encore le système constitué d'un serveur *web* et d'un ou plusieurs clients.**[06]**

II.1.3.3 Système à mémoire distribuée

Un système à mémoire distribuée met en jeu plusieurs ressources de calcul qui n'ont pas de mémoire partagée, que ce soit de manière physique ou logicielle.**[06]**

II.1.3.4 Système à mémoire partagée

Un système à mémoire partagée met en jeu plusieurs ressources de calcul qui ont de la mémoire partagée physiquement ou de manière logicielle.**[06]**

II.1.3.5 paradigme

Au sens large, un *paradigme* est un modèle auquel on apparente une famille de modèles de programmation, de modèles de communication, ou d'autres types de modèles.**[06]**

II.1.3.5.1paradigme de programmation (modèle de programmation)

Est le point de vue du programmeur sur un système concurrent. Les deux principauxparadigmes de programmation sont *réparti* et *parallèle*. Le paradigme parallèle peut êtresubdivisé en *parallèle à mémoire distribuée* et *parallèle à mémoire partagée*. **[06]**

II.1.3.5.2 paradigme de communication

Le paradigme de communication modélise la façon dont les différents fils d'exécutionéchangent des informations. **[06]**

II.1.3.6Machines mono ou multiprocesseurs

Si la machine ne possède qu'un seul processeur, il n'y aura pas de vrai parallélisme, maisseulement une illusion pour l'utilisateur. Dans un environnement multi-utilisateurs, leprocesseur (unique) est alloué à son tour à chaque utilisateur, pour une durée limitée. Sil'utilisateur termine son travail dans ce temps, c'est parfait. Sinon, une horloge l'interrompt, etle système donne le processeur à un autre utilisateur. Lorsque tous les utilisateurs auront étéservis, le premier reprendra le processeur pour une nouvelle tranche de temps.

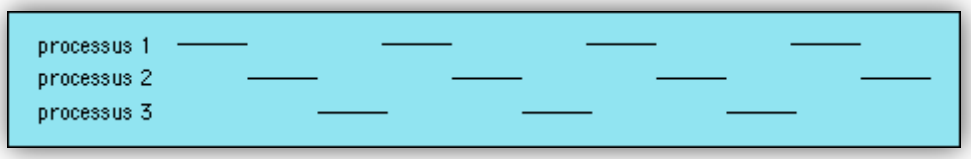

Figure II.1 : mécanisme du pseudo parallélisme

Dans le schéma ci-dessus, chaque trait représente l'allocation du processeur au processus correspondant.

Ce découpage étant très rapide, si le programme écrit des données à l'écran à chaque tranche de temps, l'utilisateur aura l'impression que la machine travaille en permanence pour lui. Simplement, elle lui paraîtra peut-être un peu plus lente.

Lorsqu'on écrit un programme, il n'est pas nécessaire de penser à ce découpage. En effet, c'est une horloge qui interrompt, et donc on ne peut absolument pas savoir où et quand le programme sera interrompu. Mais le système d'exploitation gère les différents programmes de telle manière qu'ils soient, après avoir été interrompus, exactement dans le même état qu'avant.

Enfin, du point de vue du programmeur, on peut toujours imaginer que le système aura à sa disposition autant de processeurs que nécessaire. Si ce n'est pas le cas, le système partagera le temps des processeurs existants de manière transparente pour l'utilisateur.**[Url 05]**

II.1.3.7 Ressources et Processus

II.1.3.7.1 Ressources

Une *ressource* est une entité pouvant servir à l'exécution d'un travail :

- les organes de la machine sont des ressources : mémoire, imprimante, disques (tous périphériques), processeurs.
- les entités logiques telles que les fichiers sont aussi des ressources.

Accès aux ressources :

- Une ressource est ***locale*** à un processus s'il est seul à pouvoir l'utiliser.

- Une ressource est ***commune*** si elle n'est locale à aucun processus.
- Une ressource commune est ***partageable avec n points d'entrée***, n ≥ 1, si elle peut êtreattribuée au même instant à n processus au plus.**[06]**

II.1.3.7.2 Processus

Un *processus* est un ensemble d'instructions (programme) ayant une unité d'algorithme et un ensemble de ressources utilisées bien précis.

Autrement,un processus est l'entité dynamique représentant l'exécution d'un programme sur un processeur.

- ***différence entre processus et programme***

Le programme est une description statique; le processus est une activité dynamique (il a un début, un déroulement et une fin, il a un état qui évolue au cours du temps).

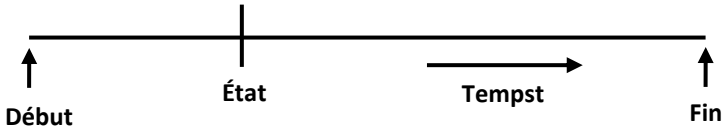

Figure II.2 : Evolution de l'état d'un processus dans le temps

Intérêt de la notion de processus

- ✓ abstraction de la notion d'exécution séquentielle, qui la rend indépendante de la disponibilité effective d'un processeur physique
- ✓ représentation des activités parallèles et de leurs interactions

- ***Etat des processus***

On ne peut attribuer globalement les ressources, car elles sont en nombre limité.

Un processus est ***actif*** à un instant donné si toutes les ressources dont il a besoin pour évoluer sont disponibles.

Le schéma ci-dessous montre l'évolution de deux processus qui se partagent une ressource R. Cette ressource ne peut pas être utilisée en même temps par les deux processus.

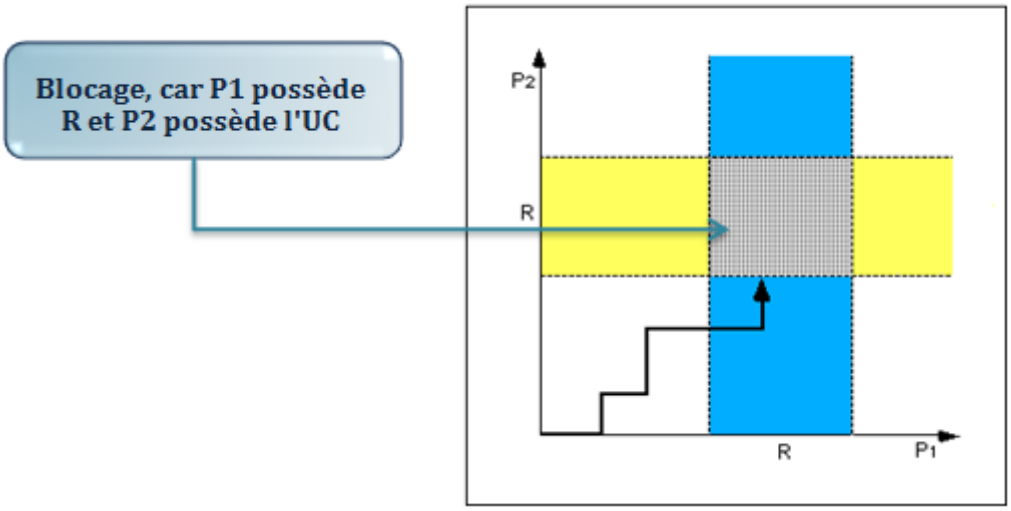

Figure II.3 : Evolution de deux processus qui se partagent une ressource R

Les axes P_1 et P_2 représentent le temps pour les processus P_1 et P_2. Les bandes colorées indiquent les parties pendant lesquelles les deux programmes utilisent la ressource R. Le chemin en escalier (trait gras) représente l'évolution alternée des deux processus :

- la partie horizontale correspond à l'évolution de P_1, lorsqu'il possède le processeur ;
- la partie verticale correspond de même à P_2.

Lorsque le trait gras coupe la bande bleue, la ressource R est allouée à P_1. La dernière partie verticale correspond à l'évolution de P_2 qui possède le processeur. Au bout de la flèche, P_2 tente de s'approprier la ressource R, qui est déjà allouée à P_1 et non partageable. P_2 ne doit donc pas accéder à la ressource, tant que P_1 ne l'aura pas libérée.

Donc, **P_2 doit libérer** le processeur à cet endroit; le trait en escalier continuera donc horizontalement jusqu'à ressortir de la bande bleue. A ce moment-là, la ressource sera libérée, et P_1 pourra reprendre son évolution**[Url 05]**.

II.1.3.8 Exclusion mutuelle

L'exclusion mutuelle sert lorsque deux processus ne doivent pas faire la même chose au même moment, (deux utilisateurs par exemple ne peuvent pas imprimer leurs fichiers simultanément sur la même imprimante). **[Url 05]**

II.1.3.8.1 Programmation de l'exclusion mutuelle

Une ressource non partageable (simultanément) est dite ***Ressource Critique***. Toute séquencequi l'utilise est dite ***Section Critique***.

II.1.3.8.2 Conditions de réalisation de l'exclusion mutuelle

Exclusion : A tout instant, un processus au plus est en section critique.

Accès : Si des processus demandent la section critique, et si la section critique est libre, l'un de ceux qui demandent doit y entrer au bout d'un temps fini.

Indépendance : Le blocage par cette section critique doit être indépendant des autres types de blocage : par exemple, si un processus est terminé, ou bloqué en attente d'une imprimante, il ne doit pas empêcher un autre processus d'utiliser un CD ROM.

Uniformité : Aucun processus ne doit jouer de rôle privilégié.

Ces règles doivent être satisfaites par les solutions envisagées pour l'exclusion mutuelle. La dernière n'est pas strictement indispensable, mais elle permet une programmation plus claire et systématique.

La mention pendant un temps fini est importante : elle indique que l'attente infinie est évitée, sans donner la moindre précision sur la durée de l'attente. Que le processus utilise la ressource pendant un dixième de seconde ou pendant trois jours importe peu ; les processus doivent fonctionner sans faire d'hypothèses sur les durées**[Url 05]**.

II.1.4 Modèles de programmation parallèle

Le modèle de programmation parallèle doit permettre au programmeur d'exprimer lasémantique parallèle du programme, c'est-à-dire :

- ✓ La concurrence : le fait que des activités s'exécutent en parallèle,
- ✓ La synchronisation : la coordination des activités concurrentes,
- ✓ La distribution (ou répartition) : c'est-à-dire la répartition des données et descalculs entrant en jeu dans les activités concurrentes,
- ✓ Les communications : les échanges de données entre les activités concurrentes.

Ces quatre aspects caractérisent le parallélisme. Ils sont liés entre eux, la concurrencenécessite des moyens de synchronisations, la distribution des données implique descommunications, etc.**[07]**

II.1.5 Types de parallélisme

II.1.5.1 Parallélisme explicite

Un modèle de programmation parallèle sera qualifié deparallélisme explicite lorsque le programmeur a la maîtrise totale de tous les aspects duparallélisme (La concurrence, la synchronisation, la distribution, les communications).**[08]**

II.1.5.2 Parallélisme implicite

Le programmeur écrit son application dans un langageséquentiel conventionnel et le parallélisme est extrait par le système sous-jacent. Celapeut être à différents niveaux : un compilateur parallélisant, un exécutif, au niveau dumatériel.**[08]**

II.1.5.3 Parallélisme des données

On l'utilise dans des structures de données homogènes. Il consiste à découper les données puis les affecter aux mémoires des processeurs donc chaque processeur effectue les mêmes calculs à des données différentes. Cette méthode est très utilisée dans la résolution de problèmes d'algèbre linéaire [FigureII.4]

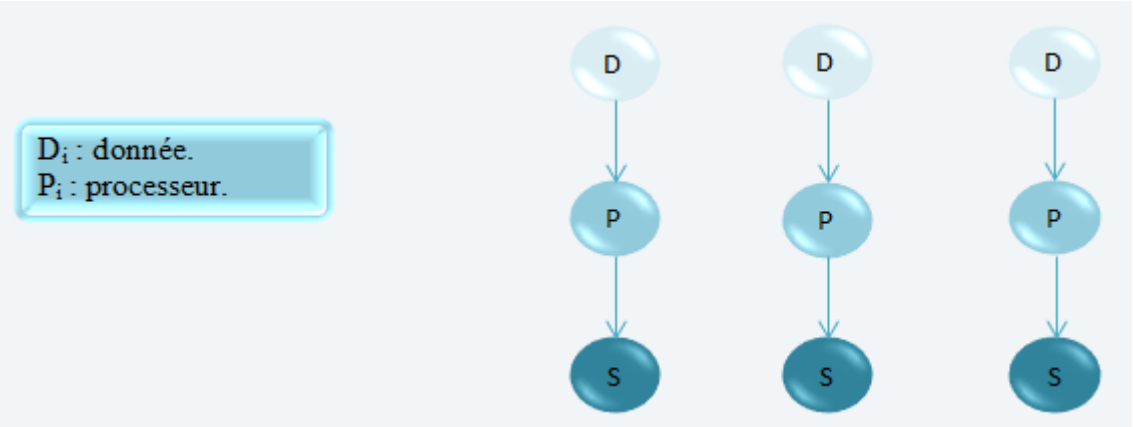

Figure II.4: Parallélisme de données

Dans ce type de parallélisme, la distribution de données se fait sous trois formes :

Distribution par bloc : les données sont divisées par le nombre de processeurs, par conséquence, on aura pour les P processeurs, P blocs de données.

Distribution cyclique : les données sont distribuées une à une en anneau sur l'ensemble des processeurs.

Distribution par blocs cycliques : les données sont mises en blocs de taille prédéfinie (un diviseur du nombre de processeurs) puis on procède à la distribution cyclique.**[08]**

II.1.5.4 Parallélisme de contrôle

Dans ce type de parallélisme, qui est plus général que le parallélisme de données, un programme séquentiel est découpé en un ensemble de tâches, puis une étude sera faite sur la taille et les interdépendances de ces tâches afin de distinguer celles qui vont s'exécuter en parallèle sur plusieurs processeurs et celles qui vont s'exécuter séquentiellement. [Figure II.5]

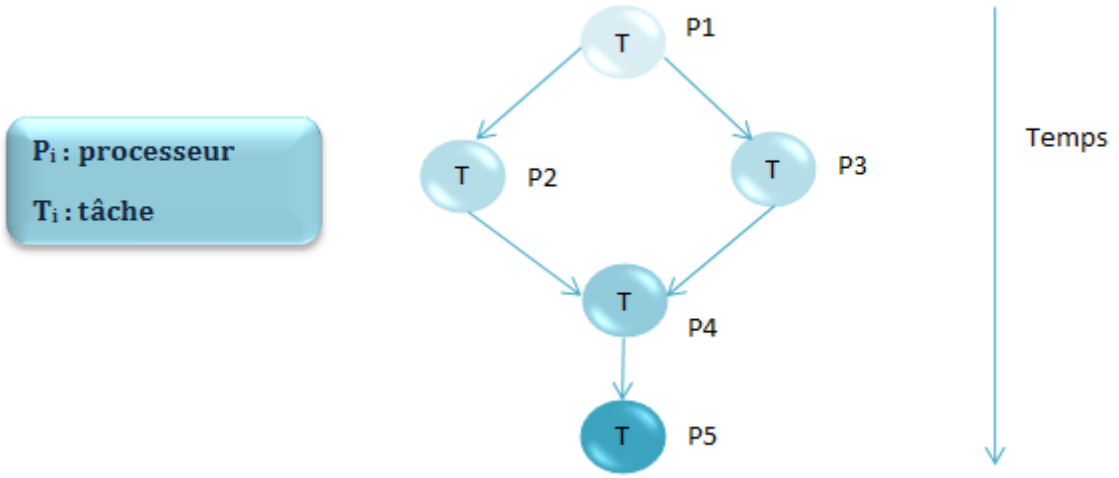

Figure II.5 : Parallélisme de contrôle

Le découpage et l'exécution des tâches en parallèle est sensible à plusieurs critères :

- ✓ Le choix des tâches influence le degré du parallélisme.
- ✓ La taille des tâches doit être suffisamment grande pour que le temps de lancement d'une tâche soit négligeable.
- ✓ La taille des tâches exécutées en parallèle doit être presque la même.
- ✓ Le placement des tâches sur les processeurs influence le parallélisme. **[08]**

II.1.6 Degré de parallélisme

C'est le nombre d'opérations exécutées simultanément, l'idée véhiculée est de savoir lenombre de processeurs nécessaires pour l'exécution des données en parallèle. Le degré peut varier au cours de cette exécution. **[09]**

II.1.7 Quelques modèles de programmation parallèle

Aujourd'hui, Il existe plusieurs modèles de programmation. Les quatre les plus répandussont :

II.1.7.1 La programmation par mémoire partagée

C'est une excellente option pour utiliserles architectures multiprocesseurs. La communication est faite naturellement pardes variables partagées et par sémaphores (synchronisation). La plupart dessystèmes d'exploitation mettent à disposition une interface de programmationimplantant ce modèle. Un des avantages de ce modèle est

que la décomposition et leplacement sont automatiquement traités par le système d'exploitation.**[10]**

II.1.7.2 L'échange de messages

C'est la technologie de base qui vise à réaliser descommunications entre les processeurs d'une architecture MIMD. Ces paradigmes sontbasés sur deux primitives très simples, *send*et *receive*, dont les paramètres sontl'identification de l'autre processeur impliqué à la communication, l'adresse dumessage à envoyer (ou l'adresse pour stocker le message à la réception).

Afin d'assurer la portabilité et la facilité d'utilisation (et de réutilisation) desprogrammes, un standard pour l'interface de programmation par échange demessages a été élaboré (MPI).**[10]**

II.1.7.3 Communication avec rendez-vous

L'émetteur et le récepteur doiventdemander à communiquer pour que la transmission ait lieu.

Dans le modèle à canal non borné, l'émetteur peutémettre, le message sera mémorisé et délivré au récepteur quand celui-ci ledemandera. La première solution (synchrone) ne permet pas d'utilisercommodément toute la bande passante du réseau. La seconde (asynchrone) supposeune capacité non bornée du réseau. La plupart des bibliothèques de communicationoffrent un modèle à canal borné et c'est à la charge du programmeur de calculer la taille destampons nécessaires à son application pour éviter les blocages.**[10]**

II.1.7.4 L'appel de procédures à distance (*Remote Procédure Call* - RPC)

Etend lemécanisme « conventionnel » d'appel de procédures des langages comme C. Un RPCest un appel de procédures entre deux différents processeurs, l'appelant et l'appelé.

Quand un processus du processeur A réalise un appel vers une procédure à distancesur un autre processeur B, le processus de B reçoit les paramètres d'entrée, exécutela procédure, et puis renvoie les résultats vers l'appelant (A). Comme le rendez-vous,le RPC est *synchrone*. Quand le processus de A réalise un appel à distance d'uneprocédure P, il reste bloqué jusqu'au moment où la procédure P est finie et le résultatretourné. Ceci limite le degré de parallélisme. Un processus dit serveur peut

exécuter ces appelsde procédures l'un après l'autre. Une autre solution consiste à créer un tel processuspour chaque appel**[10]**.

II.2 Architectures Parallèles

Aujourd'hui, la plupart des serveurs sont des machines parallèles (des multiprocesseurs). C'est ainsi qu'un ordinateur peut exécuter des instructions aussi rapidement que la vitesse de la lumière. En réalité, il s'agit d'une structure dite de "pipe-line" qui consisteà décomposer l'exécution d'une instruction en étapes élémentaires traitées en parallèle parplusieurs processeurs. De la sorte, le temps moyen d'exécution d'une instruction estconsidérablement accéléré**[11]**.

II.2.1 Classification d'architectures parallèles

II.2.1.1 Classification de Flynn

C'est la classification laplus ancienne et la plus connue. Les machines sont classées selon deux flots : le flot d'instructions et le flot des données. Ces flots peuvent être simples ou multiples, définissant ainsi 4 classes de Machines. **[11]**

II.2.1.1.1 Machine SISD

C'est l'ordinateur séquentiel standard. Une machine *SISD* (*S*ingle *I*nstruction *S*ingle *D*ata) est ce que l'on appelle d'habitude une machine de Von Neumann. Une seule instruction est exécutée et une seule donnée (simple, non-structurée) est traitée à tout instant. **[11]**

Figure II.6. Machine SISD

II.2.1.1.2 Machine MISD

N processeurs, chacun avec sa propre unité de contrôle, se partagent une mémoire commune. Une machine ayant une architecture *MISD* (*M*ultiple *I*nstruction *S*ingle *D*ata) de calcul parallèle peut exécuter plusieurs instructions en même temps par plusieurs unités fonctionnelles sur la même donnée. Cela peut paraître paradoxal mais recouvre un type de micro-parallélisme dans les micro-processeurs modernes (les architectures pipelines). La réalité est que la donnée traitée durant chaque étape

detraitement dans le pipeline est différente de la précédente. Il n'existe aucune machine construite sur ce modèle. Ce n'est qu'un modèle théorique. **[11]**

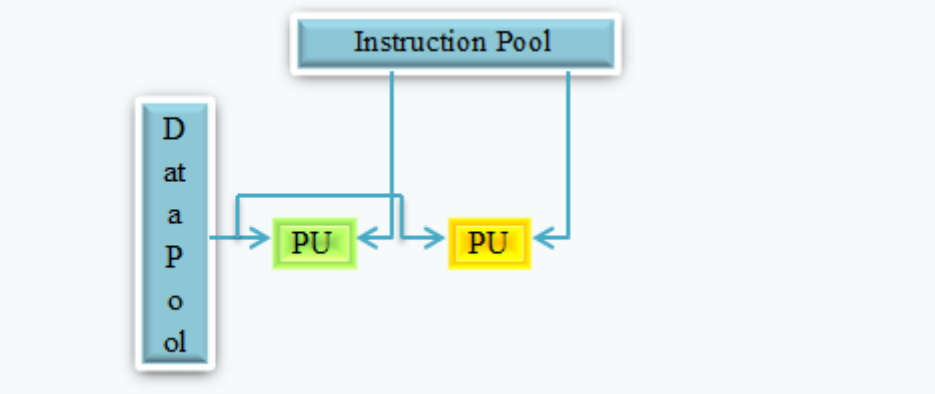

Figure II.7. Machine MISD

II.2.1.1.3 Machine SIMD

Dans le modèle SIMD (Single Instruction Stream Multiple Data Stream) de Flynn, on réplique les unités de traitement (Ut) pour effectuer exactement la même opération,exactement au même moment sur des données différentes : toutes les opérations parallèles d'une instruction sont exécutées de manière parfaitement synchrone sous le contrôle d'une unité de contrôle centrale (UC) qui possède l'horloge de la machine SIMD. Pour le programmeur, tout se passe comme si son programme s'exécutait sur une machine séquentielle classique, à la seule différence que les instructions exécutent des opérations multiples au lieu d'opérations simples. Du point de vue de la programmation, les machines SIMD sont des machines séquentielles. **[11]**

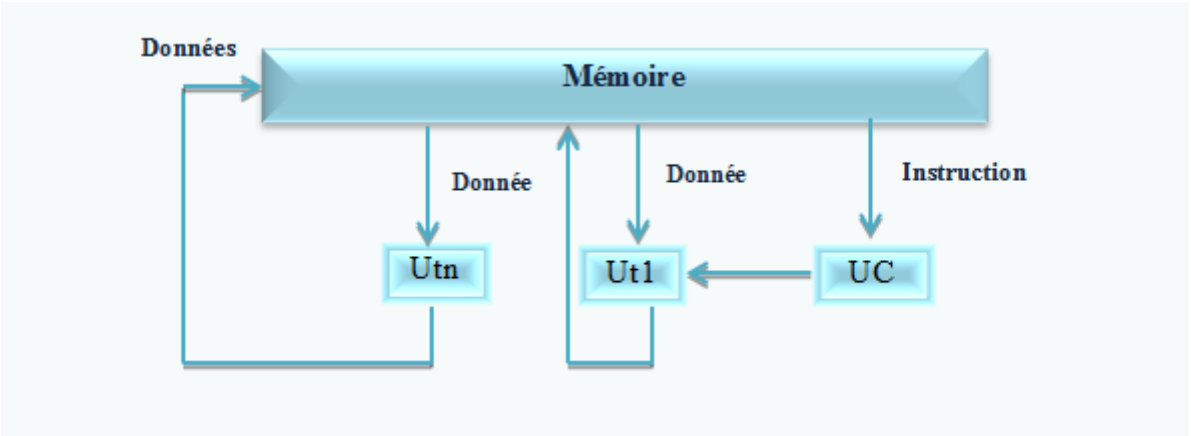

Figure II.8: Modèle de fonctionnement de SIMD

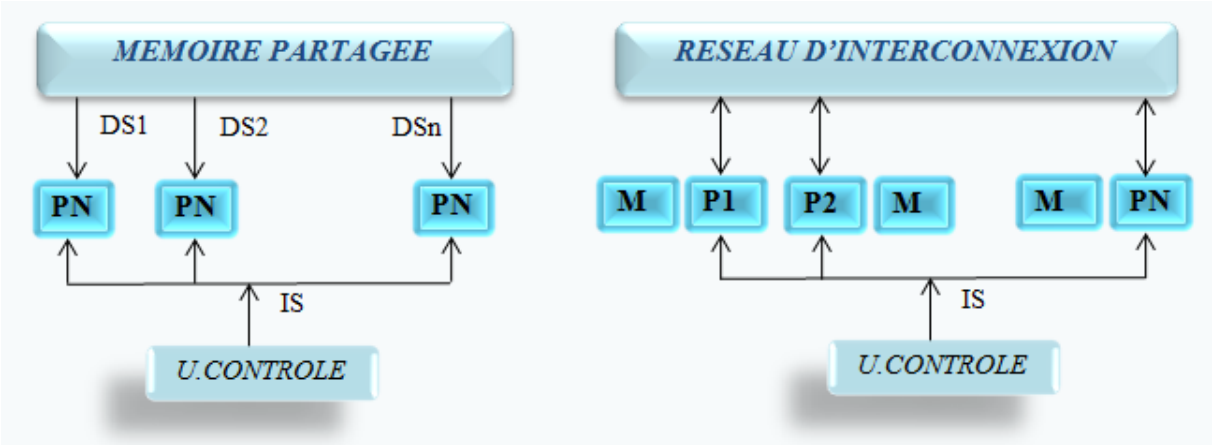

Figure II.9: Machine SIMD

Le modèle Pipeline conserve la structure monoprocesseur du modèle de Von Neumann. Cependant les unités de traitement et de contrôle sont découpées en étages chargés d'une partie de l'opération à effectuer. Ainsi, des données peuvent être traitées à l'étage 2 pendant que d'autres sont traitées à l'étage 1. Le flot d'information est continu et la vitesse de traitement s'accroît avec le nombre d'étages. Dans la classification de Flynn, ces machines sont de type MISD, mais on peut considérer que l'effet pipe-line est semblable à un effet Multiple Data Stream et, en pratique, on classe les machines pipe-line avec les machines SIMD.**[11]**

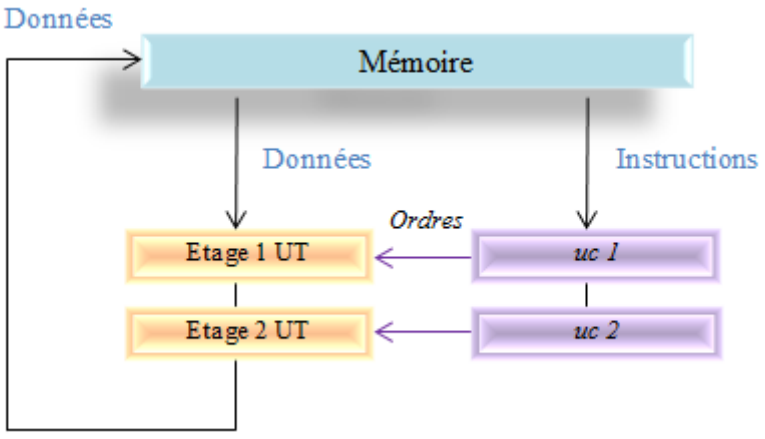

Figure II.10: Modèle de fonctionnement vectoriel

II.2.1.1.4 Machine MIMD

Le cas des machines *MIMD* (*M*ultiple *I*nstruction *M*ultiple *D*ata) est la classe la plus générale et la plus intuitive. Il y a *N* processeurs, *N* flots d'instructions et *N* flots de données. Ces machines ont un nombre de processeurs qui fonctionnent de manière asynchrone et indépendante. A tout moment, différents processeurs peuvent exécuter différentes instructions sur différentes données. Les machines MIMD ainsi que SIMD sont appropriées aux techniques de parallélisme de données. Elles sont utilisées dans différents domaines d'applications tels que la conception assistée, la simulation et la modélisation.

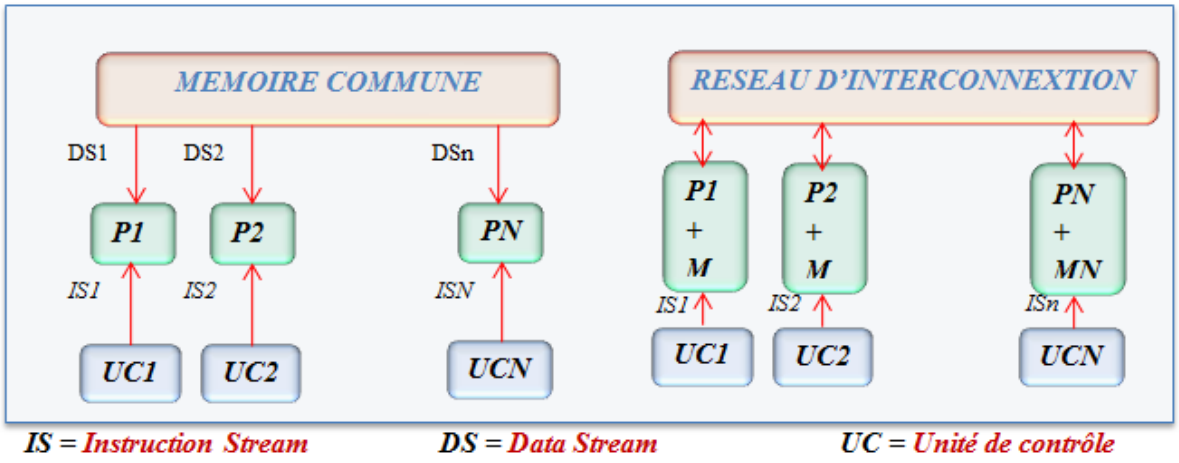

Figure II.11: Machine MIMD

La communication de données ou de résultats entre les processeurs peut se faire à travers une mémoire commune ou un réseau d'interconnexion. On distingue deux catégories d'architectures possibles en fonction de l'accès des processeurs MIMD à la mémoire:

(1) Mémoires partagées: basées sur le concept de bus. Elles sont dites machines multiprocesseurs ou fortement couplées.

(2) Mémoires locales avec réseau de communication (Transputer), ou système réparti. Elles sont appelées multi-computers ou machines faiblement couplées. **[11]**

II.2.1.2 Classification par Mémoire

La classification de Flynn ne tient pas compte d'un élément important qui s'est produit dans les années soixante-dix : les progrès de la technologie ont rendu possible la réalisation de mémoires rapides de grande taille.

Cette classification classe les machines parallèles en fonction de l'organisation de contrôle (MIMD ou SIMD comme Flynn) et de l'organisation des données en mémoire. On constate alors qu'il existe deux grands types d'organisation de la mémoire.**[10]**

II.2.1.2.1 Classification selon l'organisation de mémoire

II.2.1.2.1.1 Les machines à mémoire commune

Mémoire partagée entre tous les processeurs, les données d'une application sont toutes placées dans le même médium (la mémoire commune). Ainsi, un processus **p** placé sur un processeur **P** a la possibilité physique d'accéder aux données d'un autre processus **q** placé sur un autre processeur **Q**. Cette propriété est très utile car elle permet aux processus d'échanger des informations à travers des variables partagées. **[10]**

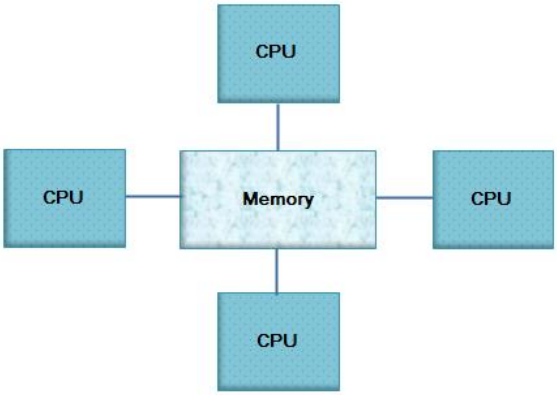

Figure II.12: Les machines à mémoire commune

II.2.1.2.1.2 Les machines à mémoire distribuée

Chaque processeur possède sa propre mémoire locale privée,ce n'est pas possible d'accéder aux données d'un autre processus, c'est pourquoi il est obligatoire d'ajouter un réseau externe d'intercommunication entre les processeurs pour qu'ils puissent échanger des informations.**[10]**

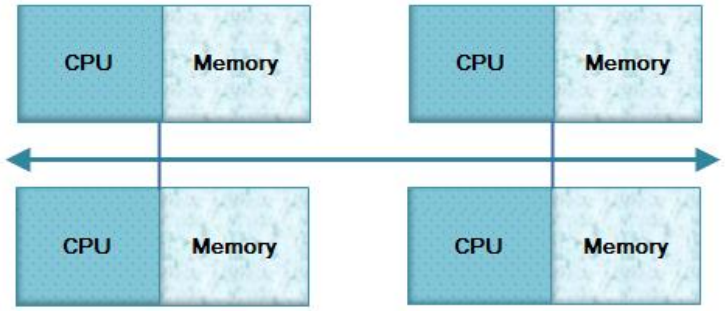

Figure II.13: Les machines à mémoire distribuée

II.2.1.2.1.3 Architectures à mémoires Hybrides

- Les ordinateurs les plus puissants au monde sont aujourd'hui un mixte de mémoire partagée et mémoire distribuée.

- La brique de base (nœud) est un multiprocesseur à mémoire partagée.

- Ces briques sont interconnectées par un réseau (type ethernet, myrinet, Infiniband, …). **[10]**

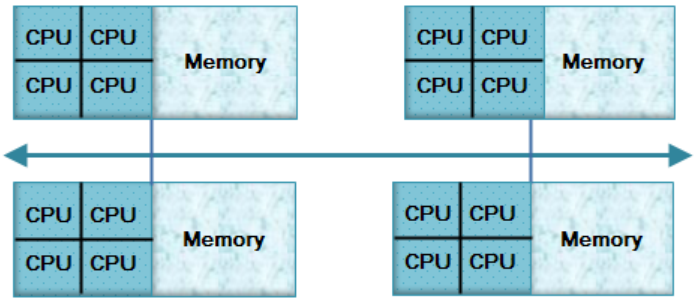

Figure II.14: Les machines à mémoire Hybride

II.2.1.2.2 selon les méthodes d'accès à la mémoire

Cette seconde classification organise les machines MIMD en trois catégories :

II.2.1.2.2.1UMA (Uniform Memory Access)

Dans ce type de machines, tous les processeurs ont accès à toute la mémoire de la machine de façon uniforme. C'est le cas des machines à mémoire centralisée et partagée (CentralizedShared Memory) entre tous les processeurs au moyen d'un bus. Ce bus devient rapidement le goulot d'étranglement lorsque les requêtes des processeurs à la mémoire sont nombreuses et simultanées, il engendre donc un phénomène de contention des processeurs qui est proportionnel à leur nombre dans l'architecture. **[Url 08]**

II.2.1.2.2.2 NUMA (Non Uniform Memory Access)

Dans ces machines, tous les processeurs ont accès à tout l'espace mémoire de la machine (comme dans les machines UMA), mais le temps d'accès n'est plus uniforme, il dépend de la localisation géographique des données dans la machine. C'est le cas des machines à mémoire distribuée-partagée (DistributedShared Memory - DSM). Chaque processeur est connecté directement à une mémoire locale et a un bus qui lui permet d'accéder à chaque mémoire locale de chaque processeur de la machine. **[Url 08]**.

II.2.1.2.2.3 NORMA (N0 Remote Memory Access)

Dans cette architecture les processeurs n'ont plus accès à toute la mémoire de la machine. Chaque processeur est connecté à une mémoire locale (qui lui est privée), et ne peut accéder à la mémoire locale d'un autre processeur.

Chaque processeur dans ce cas peut fonctionner quasi-indépendamment des autres, ce sont des machines faiblement couplées et appartiennent à la catégorie des multi-computers**[Url 08]**.

II.3 Concept d'échange de messages

Comme son nom l'indique, dans un modèle à échange de messages, un message est envoyé par un émetteur, il doit être intercepté (reçu) par un ou plusieurs destinataires. [Fig.II.15]

Les messages sont des paquets de données qui se déplacent entre les sous programmes, en plus de ces données, chaque message doit contenir un ensemble d'informations

- ✓ Processeur émetteur.
- ✓ Emplacement de la source.
- ✓ Type de données à transmettre.
- ✓ Longueur de données à transmettre.
- ✓ Processeur(s) récepteur.
- ✓ Emplacement de la destination.
- ✓ Taille de la destination**[12]**

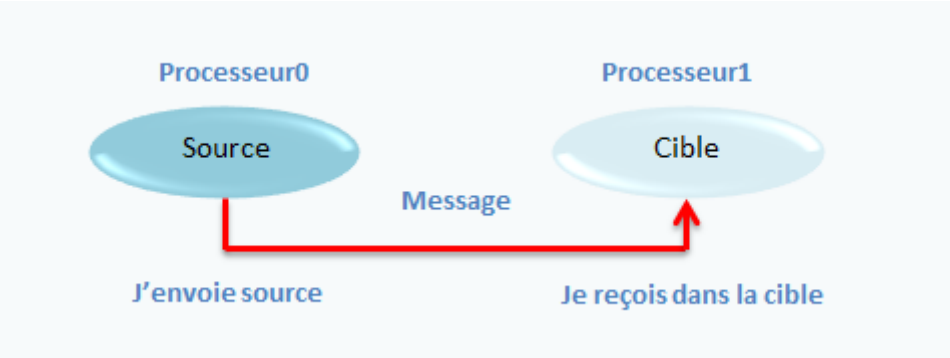

Figure II.15: Echange de messages

II.3.1 Caractéristiques du modèle d'échange de messages

- **Difficulté de programmation :** la principale difficulté du modèle de passage par messages réside dans la gestion explicite de la distribution des données, dans l'ordonnancement des tâches et dans la communication entre les tâches. Ce qui a pour conséquences entre autre : la perte du temps, la multiplication des erreurs et donc des coûts additionnels dans le développement.
- **Efficacité :** le programmeur dispose d'une grande liberté et de beaucoup de choix d'approches de résolution de problèmes, ainsi que d'une grande possibilité d'optimisation des solutions. Si les ressources sont disponibles, une grande efficacité peut être obtenue.
- **Portabilité :** Le modèle de passage par messages est très populaire ce qui a conduit à sa normalisation et standardisation, actuellement, les standards sont installés sur une grande variété de machines, donc un programme écrit en respectant les standards peut être facilement transposé sur un autre ordinateur.**[12]**

II.4Environnements parallèles, bibliothèques et outils pour le programmeur

Actuellement, il n'existe pas de langage parallèle standard. En général, tous lesconstructeurs proposent une programmation basée sur un langage standard comme C,C++ ou Fortran, augmenté d'un ensemble de fonctions de communication.

Plusieurs bibliothèques ont été mises à disposition soit par les constructeurs demachines soit par des laboratoires de recherche, Initialement les plus répandues sont:

- ✓ MPI: MPICH
- ✓ PVM standard
- ✓ Langages pour mémoire partagée: OpenMP**[13]**

II.4.1 Linda

Linda utilise la communication par mémoire partagée, contrairement à ceuxqu'on va présenter, qui utilisent la communication par échange de message. Le schémaemployé par Linda est de type maître-esclave. Le travail à réaliser est divisé en uncertain nombre de tâches, lesquelles sont mémorisées dans l'espace global desdonnées. Un processus, le maître, crée des tâches, récupère et élabore les résultats.

Chaque esclave exécute au moins une tâche. Cette approche est relativement facile àmettre en œuvre. En outre, Linda assure des fonctions d'équilibrage de charges.**[13]**

II.4.2. PVM

II.4.2.1. Définition

PVM est un intergiciel pour le calcul parallèle distribué. Il est basé sur le concept de *machine parallèle virtuelle* (PVM = *Parallel Virtual Machine*). Une machinevirtuelle parallèle est un ensemble hétérogène et dynamique d'hôtes reliés par des réseaux, ensemble sur lequel la vue fournie par PVM est l'illusion d'un unique calculateur parallèle. Les communications entre les différentes tâches se font par échange de messages, en point à point ou diffusion. Actuellement PVM supporte de nombreuses architectures, du simple PC jusqu'aux supercalculateurs, sous les variantes d'Unix ou sous Windows, pour la programmation en C, C++ et FORTRAN.

PVM a été source d'inspiration pour de nombreux projets par la suite, et tout particulièrement MPI.

PVM comporte les fonctionnalités suivantes :

- ✓ gestion des nœuds,
- ✓ gestion des processus,
- ✓ gestion des messages entre processus,
- ✓ gestion des groupes de processus,

- ✓ détection des processus instables.

PVM utilise un environnement dynamique instanciable à l'exécution. Un démon (du terme*daemon* au sens Unix) est alors lancé avant toute exécution d'application.

PVM est donc une surcouche du système d'exploitation et s'exécute dans l'espace mémoire de l'utilisateur. Tout accès aux primitives de la bibliothèque de PVM est intercepté par le démon, qui gère les accès distants (concurrents ou non), les communications, les synchronisations et la cohérence de l'état de la machine virtuelle.

PVM permet à un ensemble de machines connectées sur un réseau local de se conduire comme une seule et même ressource appelée machine virtuelle**[14].**

II.4.2.2.Composition de PVM

➢Un démon,

➢Une console,

➢Des bibliothèques (C et Fortran) pour la programmation par échange de messages.

II.4.2.3. Caractéristiques

➢ Simple (cache les problèmes à l'utilisateur),

➢ Portable,

➢Hétérogène: machines, architecture, OS, réseaux.

- ➢ peut être composée de machines séquentielles et parallèles.**[14]**

II.4.2.4. Organisation logique des processus au sein de la machine virtuelle PVM

Dans une machine virtuelle, les processus s'organisent sous la forme d'un arbre. Il y a un processus qu'on appelle «processus racine». Cette organisation est due à la spécification de PVM. En effet, l'utilisateur ne peut créer qu'un seul processus et seuls ces mêmes processus peuvent en créer d'autres. Ainsi, les processus ont tous un père sauf le processus «racine». Lorsqu'un fils est créé, il hérite de l'environnement de son père. De plus, la machine virtuelle donne un identifiant unique à chaque processus.**[14]**

II.4.2.5. Fonctionnement de la machine virtuelle PVM

PVM a un démon appelé «pvmd»qui gère les messages qui transitent entre processus et contrôle la présence des processus dont il a la charge. Lorsqu'il est lancé sur l'un des nœuds par un utilisateur, il devient le «pvmd maître». Il lance les démons, sur les

autres nœuds, qui deviennent les «pvmd esclaves». Seul le maître peut créer des esclaves et peut détecter leur instabilité. Les démons communiquent entre eux à l'aide des protocoles UDP/IP. Mais, pour amorcer la communication avec un nœud, le «démon maître» utilise les protocoles TCP/IP.

Le «démon maître» est composé d'un gestionnaire de messages et d'un gestionnaire de ressources (ressources manager RM). Le manager des ressources est un processus PVM qui a pour fonction de gérer l'ensemble des processus ainsi que de synchroniser les différents nœuds.**[14]**

II.4.3OpenMP

II.4.3.1Définition

Il s'agit d'un ensemble de procédures et de directives de compilation identifiées par un mot clef initial **!$OMP** visant à réduire le temps de restitution lors de l'exécution d'un programme sans en changer la sémantique.

- Standard "industriel" basé sur :
  - ✓ une API (*Application Programming Interface*) pour la portabilité d'applications *multithreadées*,
  - ✓ des machines à mémoire partagée ou à mémoire virtuellement partagée**[15]**.

II.4.3.2 Structure d'OpenMP

1) Directives et clauses de compilation :

- ✓ elles servent à définir le partage du travail, la synchronisation et le statut privé ou partagé des données ;
- ✓ elles sont considérées par le compilateur comme des lignes de commentaires à moins de spécifier une option adéquate de compilation pour qu'elles soient interprétées.

2) Fonctions et sous-programmes : ils font partie d'une bibliothèque chargée à l'édition de liens du programme.

3) Variables d'environnement : une fois positionnées, leurs valeurs sont prises en compte à l'exécution.**[15]**

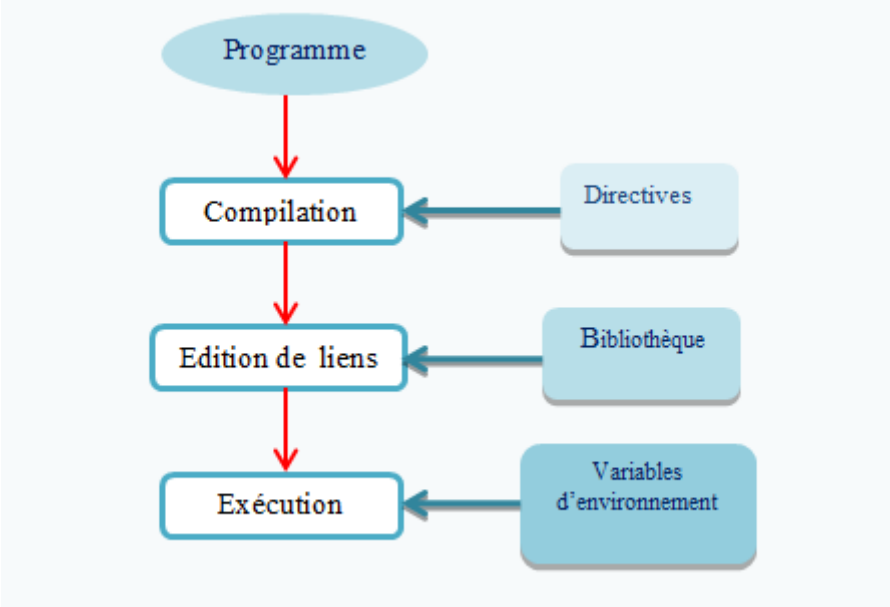

Figure II.16: Structure d'OpenMP[15]

II.4.3.3 L'architecture générale d'OpenMP

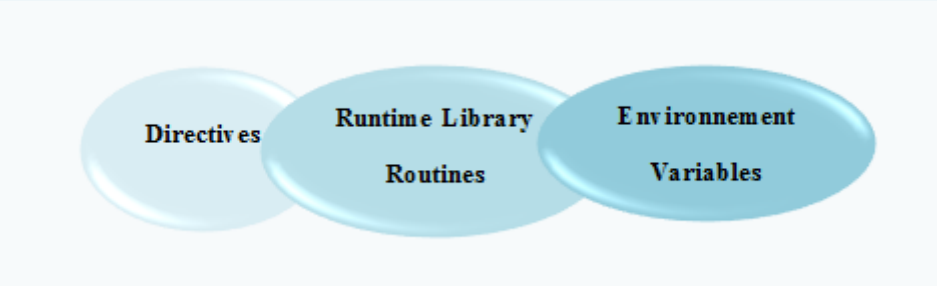

Figure II.17:Architecture générale d'OpenMP

- Les directives et leurs clauses sont de 3 types :
  - ✓ partage du travail (***work-sharing)***,
  - ✓ partage des données,
  - ✓ synchronisation.
- Les directives ne sont que des commentaires activables par le compilateur grâce à une option adéquate. Ainsi, un compilateur ne supportant pas OpenMP les ignorera. C'est ce qui permet de garantir :
  - ✓ la portabilité des codes OpenMP,
  - ✓ la maintenance d'une version unique, séquentielle et parallèle du code.

- La ***Run-time Library*** et les variables d'environnement permettent de contrôler l'environnement d'exécution comme par exemple le nombre de *threads.*

OpenMP :

- ✓ Est une API adaptée aux architectures multi cœurs.
- ✓ Elle est conçue pour une utilisation en multithread et avec de la mémoire partagée.
- ✓ Tous les threads peuvent utiliser la même mémoire partagée.
- ✓ Les données peuvent être privées ou partagées.
- ✓ Les données partagées sont utilisables par tous les threads.
- ✓ Les données privées sont accessibles que par le thread qui les possède.
- ✓ Les transferts de données sont transparents pour le programmeur.
- ✓ La synchronisation est généralement implicite.
- ✓ Les données possèdent des étiquettes**[16]**.

II.4.4 MPI (Message Passing Interface)

II.4.4.1 Qu'est-ce que le standard de communication MPI

MPI est une bibliothèque d'échange de messages pour machines parallèles hétérogènes, c'est un standard créé pour le développement d'applications parallèles portables. Une application MPI est un ensemble de processus exécutant chacun son propre code et communiquant via des appels à des sous-programmes de la bibliothèque MPI. Le modèle d'exécution d'une application MPI est le SPMD (*Single Program Multiple Data*), soit l'exécution du même programme pour tous les processus.

Les paramètres et les données sont privés à chaque processus, que l'on soit sur une architecture à mémoire distribuée ou partagée. Le partitionnement, l'échange des données, le contrôle de la synchronisation des processus MPI sont de la responsabilité de l'utilisateur**[17]**.

Remarque : la librairie MPI supporte divers langages comme le fortran 90, le C, le C++,C# ou encore Java.

II.4.4.2 Concepts de base de la programmation MPI

Il s'agit d'un modèle de programmation parallèle qui repose sur l'échange de messages.

Une donnée est échangée entre deux processus par l'appel à des primitives de communication prédéfinies.

Un message contient les données à échanger entre le producteur et le(s) consommateur(s).En plus des données, un message contient les informations suivantes :

-l'identificateur du processus émetteur

- le type de données

- la taille du paquet de données

-l'identificateur du processus récepteur

L'exécution d'un algorithme parallèle implémenté avec MPI correspond à l'exécution d'un ensemble de processus ayant en commun un programme séquentiel unique et communiquant à l'aide des fonctions de la bibliothèque MPI.

MPI ne spécifie aucun opérateur pour la création dynamique de processus. Les processus sont statiquement créés à l'initialisation.

Le standard MPI offre la possibilité aux processus d'une application d'échanger des messages dans plusieurs contextes de communication indépendants, appelés communicateurs.

Chaque communicateur est associé à un sous-ensemble de processus.

Les processus sont toujours identifiés par des entiers naturels allant de 0 à n-1, n étant le nombre de processus appartenant au communicateur. Toute application MPI possède un communicateur par défaut associé à l'ensemble de tous les processus. Ces concepts ont pour but de fournir des structures de données standards facilitant la réutilisation de codes parallèles**[10]**.

II.4.4.3 Processus/Groupe

Une application MPI est donc un ensemble statique de processus appelé *world*. A chaque processus est associé un identificateur (*Rank*), numéroté de 0 à n-1 où n est la cardinalité de l'ensemble *world*. Ce groupe initial peut être subdivisé en plusieurs sous groupes pendant l'exécution du programme.

Un processus peut appartenir donc simultanément à plusieurs sous groupes. Des primitives pour déterminer le *Rank* et le nombre de processus (*size*), appartenant à un groupe (sous groupe) sont disponibles.

Les groupes sont toujours des sous ensembles de l'ensemble initial *world*. Un groupe est identifié par un communicateur. Un processus sera toujours désigné par un rang (*rank*) relativement au groupe représenté par le communicateur.

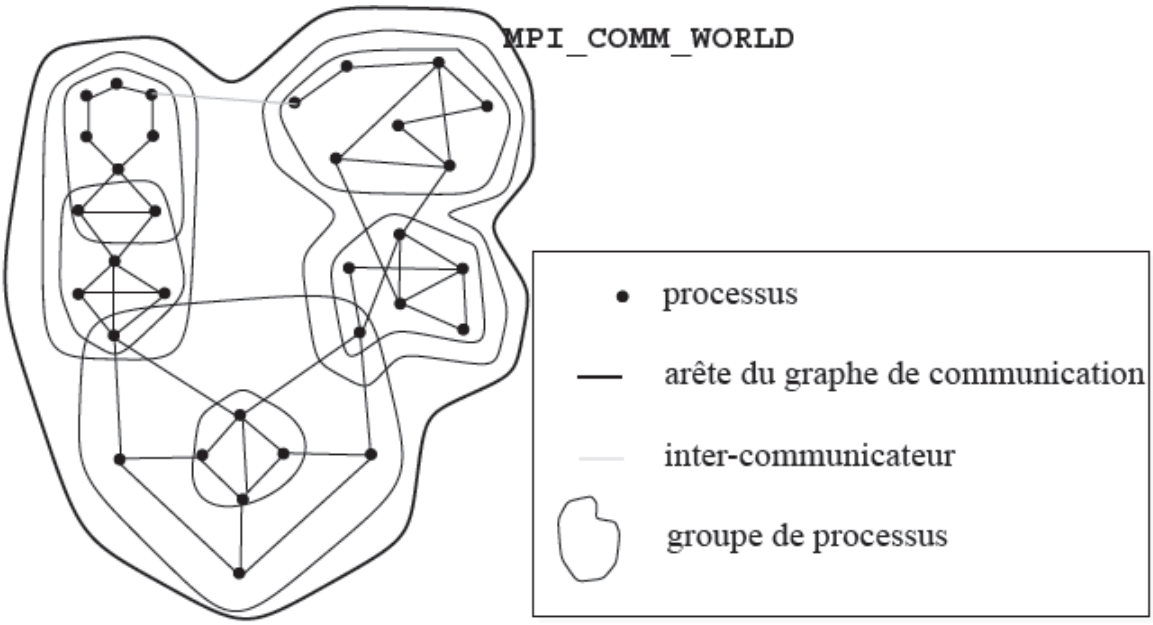

Figure II.18 : communication entre processus avec MPI

Lors de la création d'un programme MPI, un communicateur par défaut est créé, il s'agit du communicateur MPI_COMM_WORLD qui regroupe toutes les processus actifs.

Dans le modèle MPI, les communications sont gérées par un environnement qui se charge de router chaque message vers son destinataire.

Les sous-programmes de la bibliothèque MPI appartiennent à plusieurs familles :

- ✓ Communications point à point
- ✓ Communications multi-points
- ✓ Groupes et communicateurs **[10]**

II.4.4.4 Modes de communication avec MPI

On distingue quatre modes d'envoi:

- **Bufferisé :** ce mode présente une sémantique de terminaison dite locale c'est à dire laterminaison de l'opérateur *send*est indépendante du moment où le *recv*correspondantest fait. Le message à émettre est stocké localement chez l'émetteur. L'émission terminealors.

- **Standard :** dans ce mode, l'implantation est libre de choisir si l'opérateur *send*réaliseou non le stockage des messages à émettre. Si le stockage est fait, un appel *send*peutterminer avant que le *recv*associé soit posté. S'il n'y a plus de ressources pour lestockage (tampons), ou que l'implantation ne le réalise pas, le *send*n'est terminéqu'après que les données soient transférées chez le récepteur. La sémantique determinaison est dite non locale. Le standard n'impose pas de contrôle de flux. Ainsicertaines implantations (MPICH,MPI-IBM) n'implantent pas de contrôle de flux et arrêtent l'exécution en cas desaturation des tampons du destinataire.

- **Ready :** pour éviter ce problème, le mode *ready*n'émet pas le message tant que le *recv*n'a pas été fait. C'est une synchronisation de type rendez-vous.Le *send*termine quand le rendez-vous est terminé.

- **Synchrone :** ce mode a une sémantique de complétion non locale. L'émission a lieu aussitôt que possible et considère que le *recv* correspondant a été posté préalablement. Dans le cas contraire, le comportement n'est pas défini (perte, erreur, etc.).

Chacun de ces quatre modes de communication existe en version bloquante et en version non bloquante. Cependant, dans tous les cas, le succès de la communication est garanti : aucun message ne peut se perdre entre l'émetteur et le récepteur. Tous ces modes de fonctionnement servent à assurer plus ou moins strictement la synchronisation entre l'émission d'un message et sa réception**[10]**.

II.4.4.5 Gestion de l'environnement

Un processus commence toujours par définir un monde commun qui contient l'ensemble des processus du programme parallèle. Ceci se fait par la commande **MPI_Init**.Il est ensuite alors possible d'accéder au nombre total de processus (**MPI_Comm_size**), au numéro du processus courant par **MPI_Comm_rank**, au nom de la machine qui gère le processus courant (**MPI_Get_processor_name**),…

Le programme se termine toujours par **MPI_Finalize**.

MPI_Init: Initialise l'environnement d'exécution de MPI. Cette fonction doit s'appeler dans chaque programme MPI, avant toutes les autres fonctions de MPI et doit s'appeler seulement une fois dans un programme MPI. Sa syntaxe : MPI_Init (&argc, &argv) ou MPI_INIT (ierr).

MPI_Comm_size: Détermine le nombre de processus dans le groupe lié à un communicateur.Généralement utilisé dans le communicateur MPI_COMM_WORLD pour déterminer le nombre de processus employés par une application.

*Sasyntaxe:*MPI_Comm_size (MPI_COMM_WORLD, &size)

MPI_Comm_rank: Détermine le grade (rang) du processus appelant dans le communicateur. Au commencement, à chaque processus sera assigné un rang unique de nombre entier entre 0 et le nombre de processeurs - 1 dans le communicateur MPI_COMM_WORLD.

Sa syntaxe :MPI_Comm_rank (MPI_COMM_WORLD, &rank).

MPI_Abort : Termine tous les processus de MPI liés au communicateur. Dans la plupart des réalisations de MPI, il termine tous les processus indépendamment du communicateur indiqué.

Sa syntaxe : MPI_Abort (MPI_COMM_WORLD, code erreur).

MPI_Finalize: Termine l'environnement d'exécution de MPI. Cette fonction devrait être la dernière routine de MPI appelée dans chaque programme de MPI - aucune autre routine de MPI ne peut s'appeler après.

Sasyntaxe: MPI_Finalize () ouMPI_FINALIZE (ierr).

D'où la structure générale d'un programme MPI, est la suivante :

```
#include<mpi.h>
Main (intargc, char**argv)
{
MPI_Init (&argc, &argv);
< Reste de programme>
<Appel des procédures MPI>
MPI_Finalize ( ) ;
}
```

[10].

II.4.4.6. Types de communication

a) Communication point à point

Une communication point à point est effectuée entre deux processus, un émetteur et un récepteur, identifiés par leur rang dans le communicateur au sein duquel se fait la communication. L'enveloppe du message est constituée :

- ✓ du rang de processus émetteur ;
- ✓ du rang de processus récepteur ;
- ✓ du nom du communicateur ;
- ✓ du type de données ;
- ✓ de la longueur du message ;
- ✓ de l'étiquette du message.

Dans le modèle MPI, les communications standards sont synchrones, ce qui veut dire qu'un producteur de messages attend l'accusé de réception d'un consommateur avant de continuer.Il existe plusieurs possibilités d'effectuer une communication point à point avec MPI.**[17]**

- **Communications bloquantes**

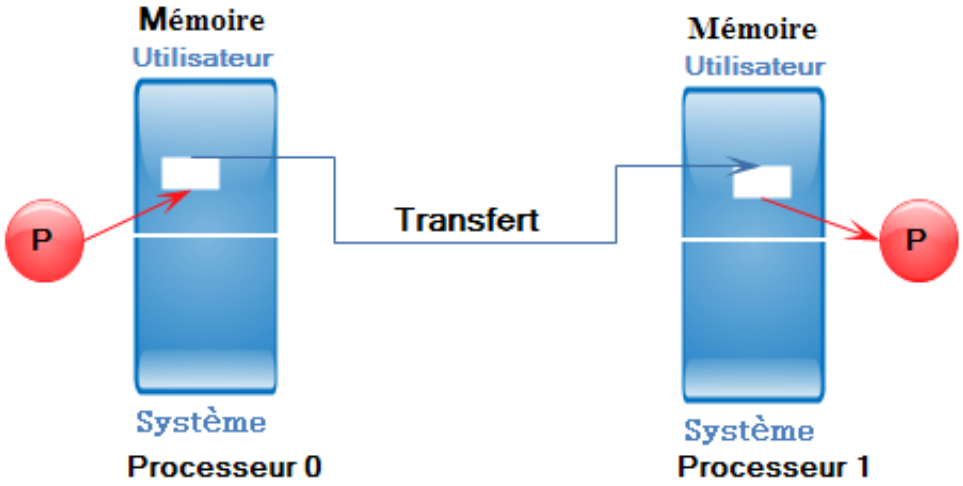

Figure II.19 : Communication bloquante

Les fonctions qui implémentent les envois et les réceptions de données (respectivementMPI_SEND et MPI_RECV) sont fondamentalement bloquantes : elles ne retournent à la tâche appelante qu'une fois la transaction achevée.

On dénombre les quatre primitives suivantes pour L'envoi bloquant

_ MPI_Send : pour l'envoi bloquant standard ;

_ MPI_Ssend : pour l'envoi bloquant synchronisé ;

_ MPI_Bsend : pour l'envoi bloquant bufferisé ;

_ MPI_Rsend : pour l'envoi bloquant Ready.**[17]**

- **Communication non bloquante :**

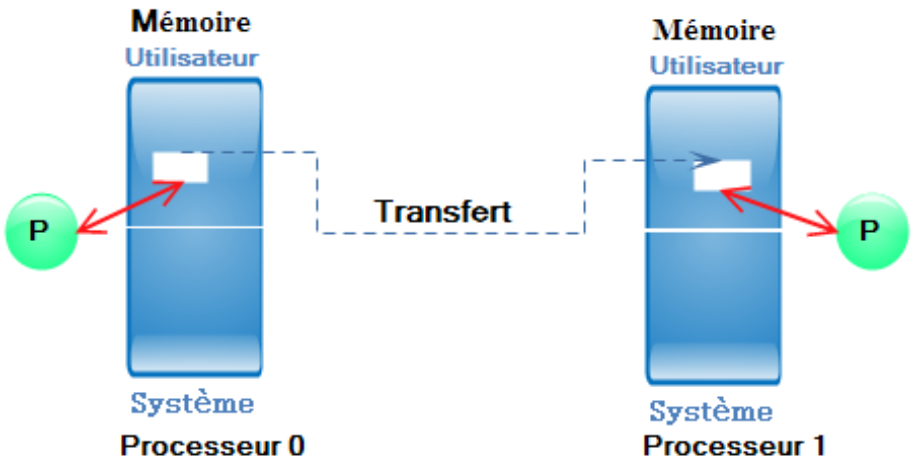

Figure II.20 : Communication non bloquante

Est une communication désynchronisée de la tâche appelante. La communication non bloquante est exécutée par un autre processus, créé pour l'occasion. Cela ne présume pas de la synchronisation entre les processus qui effectuent le transfert des données.

L'usage de fonctions non bloquantes est conseillé pour masquer la latence des communications. Cela permet notamment d'effectuer un calcul *indépendant des donnéesimpliquées dans la communication* juste après l'appel à une fonction non bloquante. Ces primitives non bloquantes fournissent des mécanismes de synchronisation qu'il convient de tester pour s'assurer que le transfert a effectivement eu lieu. Ces primitives ont un argument additionnel appelé **requête** (*request*), lequel est utilisé par d'autres primitives pour vérifier la terminaison d'une opération non bloquante précédemment démarrée. L'utilisation de telles primitives permet, si c'est possible, un recouvrement des calculs et des communications.

On dénombre les quatre primitives suivantes pour l'envoi non bloquant :

_ MPI_Isend : pour l'envoi non bloquant standard ;

_ MPI_Issend : pour l'envoi non bloquant synchronisé ;

_ MPI_Ibsend : pour l'envoi non bloquant bufférisé ;

_ MPI_IRsend : pour l'envoi non bloquant Ready.**[17]**

b) Communications persistantes

La communication persistante est utilisée quand on effectue plusieurs fois une même communication avec les mêmes arguments.On associe un objet request persistant. Ainsi, nous aurons les quatre primitives d'envoi persistant suivantes:

_ MPI_Send_init : pour l'envoi persistant standard ;

_ MPI_Ssend_init : pour l'envoi persistant synchronisé ;

_ MPI_Bsend_init : pour l'envoi persistant bufférisé ;

_ MPI_Rsend _init : pour l'envoi persistant Ready.**[17]**

c) Les communications globales ou collectives

Les communications globales doivent être exécutées par chacun des processus du groupe concerné. Elles sont considérées comme une des caractéristiques les plus importantes de MPI pour l'implémentation d'applications portables et efficaces.

Liée aux communications globales, on retrouve notamment la notion de barrières de synchronisation avec MPI_BARRIER. Cette fonction bloque le processus appelant jusqu'à ce que tous les processus du groupe l'aient appelé.
Les opérations de communications globales fournies par MPI sont les suivantes:

- ✓ **Broadcast :**

La diffusion d'un membre vers tous les autres (OTA: One-To-All) est réalisée par la fonction MPI_BCAST (buffer, count, datatype, root, comm). La fonction est appelée avec les mêmes valeurs pour les paramètres root et commmais elle aura un comportement différent selon que le rang du processus est égal à root (émetteur) ou non (récepteur).

- ✓ **Gather :**

Le regroupement de tous les membres du groupe vers un élément particulier (ATO: All-To-One) est obtenu grâce à la fonction MPI_GATHER.

- ✓ **Scatter :** la diffusion personnalisée (POTA: Personalized-One-To-All) est possiblepar la fonction MPI_SCATTER.
- ✓ **Reduce, Scan :**

Les opérations globales sur tous les membres du groupe commemax, min, somme, etc. ... sont activées par la fonction MPI_REDUCE.

- ✓ **All broadcast :**

L'échange total ou diffusion depuis tous les processeurs (ATA: All- To-All) est fournie par MPI_ALLTOALL

- ✓ **All gather:**

L'échange total personnalisé (PATA: Personalized-All-To-All) est possible avec la fonction MPI_ALLGATHER.**[17]**

II.4.4.7Les différents types de messages

Pour le type de données transmises dans le contenu du message, MPI spécifie des types standards MPI_type. Par exemple pour les entiers le type MPI s'appelle**MPI_INT.**

La librairie MPI met en œuvre six types de messages qui permettent aux différents processus de communiquer entre eux. Les informations échangées entre processus sont des tableaux d'éléments qui peuvent être de type :

§ MPI_CHAR.
§ MPI_SHORT.
§ MPI_INT.
§ MPI_LONG.
§ MPI_UNSIGNED_CHAR.
§ MPI_UNSIGNED_SHORT.
§ MPI_UNSIGNED.
§ MPI_INSIGNED_LONG.
§ MPI_FLOAT.
§ MPI_DOUBLE.
§ MPI_LONG_DOUBLE.
§ MPI_BYTE (groupe de 8 bits).
§ MPI_PACKED (pour gérer d'autres types).**[17]**

II.4.4.8 Les environnements de programmation MPI

II.4.4.8.1 LAM/MPI

(Local Area Multicomputer MPI), est un environnement de programmation et de développement respectant le standard MPI pour réseaux d'ordinateurs hétérogènes. Avec LAM, une grappe (de calcul) dédiée ou une infrastructure de calcul en réseau existante peut agir comme un seul ordinateur parallèle résolvant un problème.**[17]**

II.4.4.8.2 MPICH

- **Présentation**

MPICH propose une librairie MPI ainsi que l'environnement nécessaire pour respecter les autres exigences de la norme. Les fonctions MPI implémentées dans MPICH sont écrites au-dessus d'une ADI (*Abstract Device Interface*). Cette ADI est une bibliothèque de macros définissant une interface avec une bibliothèque de communications appelée *device*. De fait, à la compilation de MPICH il est nécessaire de choisir un *device*. Les deux principaux *devices* sont ch_p4 et ch_schmem.

- ✓ **ch_p4** :

Permet une liaison inter-machines (inter-nœuds) par utilisation du protocole TCP/IP. Son utilisation pour réaliser des communications au sein d'une machine conduit à une dégradation importante des performances.

- ✓ **ch_shmem**:

Permet une liaison rapide entre tâches s'exécutant sur une même machine par utilisation de la mémoire partagée pour effectuer les échanges de messages.

- ➢ **Pourquoi MPICH ?**

- MPICH est un support efficace pour différents calculs et plates-formes de communication.
- Il permet une recherche de premier plan dans MPI à travers une source ouverte, structure modulaire facile à étendre.

Nous reviendrons dans le prochain chapitre sur l'installation, la configuration et l'utilisation du MPICH.

- ➢ **MPICH2**

MPICH2 est une implémentation de la norme MPI-2. Propose une librairie MPI ainsi que l'environnement nécessaire pour respecter les autres exigences de la norme. D'une grande portabilité, elle est disponible pour Linux, Unix et Windows. Plusieurs types de réseaux sont supportés, comme les réseaux TCP, infiniband ou Myrinet.**[17]**

II.4.8.3. Extensions MPI-2

Le standard MPI-2 présente certaines nouveautés afin de combler certains vides constatés, ces nouveautés concernent notamment:

- ➢ **Gestion dynamique des processus**

Contrairement à PVM, les modèles de programmes parallèles offerts par MPI 1.0 reposent entièrement sur un nombre statique de processus, MPI 2.0 a remédié à cet état de cause en permettant la gestion dynamique des processus, ce qui a pour effet :

- ✓ La Possibilité de développer des codes MPMD.
- ✓ Un support multi plates-formes.
- ✓ Le démarrage et l'arrêt dynamique des sous-tâches.
- ✓ La gestion des signaux système.

- **E/S parallèles**

Permet la réalisation des E/S asynchrones, des accès à des fichiers par tranche et le contrôle de la répartition des fichiers sur les supports mémoires.

- **Communication de mémoire à mémoire (RMA)**

Accès direct à la mémoire d'un processus distant (*Remote Memory Access*), trois sous programmes sont prévus pour cela :

- ✓ **Get**: Lecture à distance.
- ✓ **Put** : Ecriture à distance.
- ✓ **Remote accumulation** : Une variante d'écriture à distance où la valeur écrite et combinée avec la valeur déjà existante. **[17]**

II.4.4.9Fonctionnement de MPICH2 en réseau

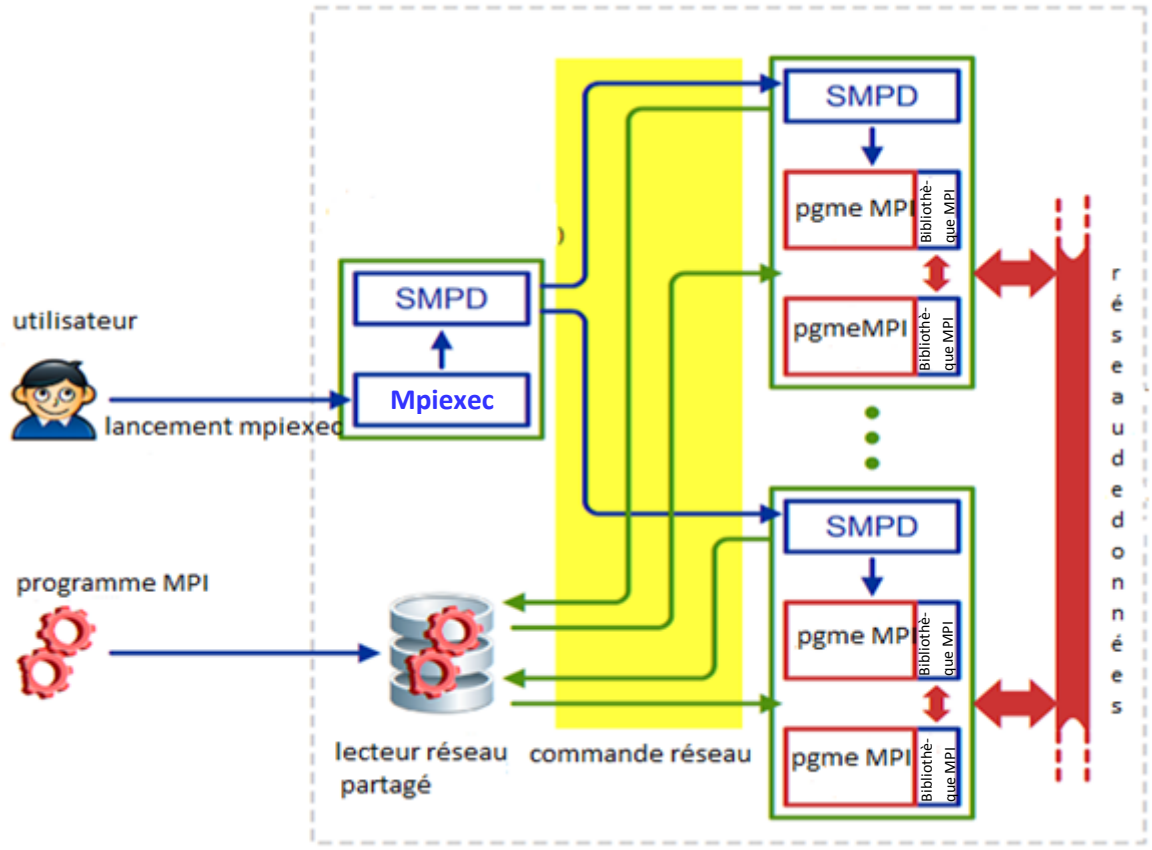

Figure II.21: Schéma de MPICH2 en réseau

La figure ci-dessus (figure II.21) illustre l'exécution d'un programme MPI dans un réseau de machines windows, MPICH2 fournit le gestionnaire des processus SMPD (super multipurpose daemon) pour les applications MPICH2 qui est un service système qui doit être installé et configuré sur toutes les machines du réseau et qui gère la liste des nœuds de calcul dans le système.

1. L'utilisateur lance mpiexec, spécifie le nom de l'exécutable du programme MPI et le nombre de processus requis.
2. Mpiexec donne des informations sur les processus en cours, au gestionnaire local, qui a une liste de nœuds de calcul disponibles.
3. Le gestionnaire de processus se réfère à des nœuds de calcul dans la liste, en passant les orientations d'exécution du programme MPI en cours.
4. Les processus de gestion sur les nœuds de calcul exécutent plusieurs copies du programme MPI (copies multiples éventuellement sur chaque nœud), en passant le programme d'information nécessaire pour communiquer les uns avec les autres. Cela signifie que si vous exécutez le programme `C:\progme.exe` par exemple, tous les gestionnaires des processus sur les nœuds de calcul vont essayer de l'exécuter. Si au moins un nœud ne répond pas, une erreur se produit en cours d'exécution**[Url 08]**.

Le fonctionnement du programme MPI se présente comme suit:

1. Le programme démarre et initialise la bibliothèque run-time de MPICH2 en appelant `MPI_Init`.
2. La bibliothèque reçoit l'emplacement du programme et le nombre des autres processus, du gestionnaire de processus, et établit une communication avec eux.
3. Après l'exécution, cette copie du programme peut partager des informations les uns avec les autres à travers la bibliothèque de MPICH2. Du point de vue système d'exploitation, la bibliothèque fait partie du programme (elle fonctionne dans le même processus), on peut donc supposer que l'exécution d'une copie de l'échange de données MPI-programme directement les uns avec

les autres comme toutes les autres applications qui transmettent des données sur le réseau.

4. La console d'entrée-sortie de tous les processus MPI-programme est redirigée vers la console, qui a lancé mpiexec.
5. Avant l'achèvement, tous les processus appellent la fonction `MPI_Finalize`, qui complète correctement la transmission et la réception de messages, et déconnecte MPICH2.

Remarque: Tous les principes ci-dessus s'appliquent, même si on exécute le programme MPI sur un ordinateur. **[Url 08]**

Conclusion

Dans ce chapitre, nous avons donné un aperçu sur quelques notions liées au parallélisme : définitions, types de parallélisme, les architectures et les environnements parallèle….etc.

MPI permet de compiler et d'exécuter un programme parallèle ou distribué.Cette librairie a pour but principal de définir un ensemble de primitives de communication portables sur un grand nombre de machines. Il faut noter qu'avec MPI, les processus sont naturellement et statiquement liés à un processeur.

Chapitre III

CONCEPTION ET MISE EN ŒUVRE

Introduction

Depuis le début de l'informatique, les besoins en puissance de calcul ne cessent d'augmenter. Beaucoup d'applications comme dans le domaine de traitement d'images en 3D ont besoin de plus de puissance de calcul. Par ailleurs, les applications ont besoin d'être exécutées en un temps toujours plus court. C'est pourquoi la nécessité d'accroître les performances matérielles et logicielles constitue un problème primordial.

Pour fournir la puissance de calcul nécessaire, on a eu recours à l'assemblage de processeurs à l'aide de réseau de machines virtuelles. C'est après ce dernier fait qu'on a eu recours à l'exécution des processus parallèles.

L'exécution des processus parallèles sur les machines virtuelles pour étudier l'intérêt du parallélisme en temps et en coût nécessite une approche bien étudiée qu'on décrira tout au long de ce chapitre.

Objectif de notre travail :

Notre application consiste en la compression des fichiers par coopération entre plusieurs processus sur plusieurs unités de calcul de réseau virtuel. La compression se fait en deux étapes : la compression des fichiers en fichiers.tar puis la compression de ces derniers en fichiers.tar.gz. Pour atteindre cet objectif, nous avons utilisé le parallélisme sur les machines virtuelles afin d'optimiser l'usage des ressources physiques. En effet, le parallélisme virtuel permet des baisses de coûts importantes par la réduction du nombre de machines physiques, et par toutes les autres économies induites : énergie, temps de mise en œuvre et d'exécution. **[18]**

III.1 Schéma de l'application

Dans la figure III.1, on donne le schéma suivi par l'application.

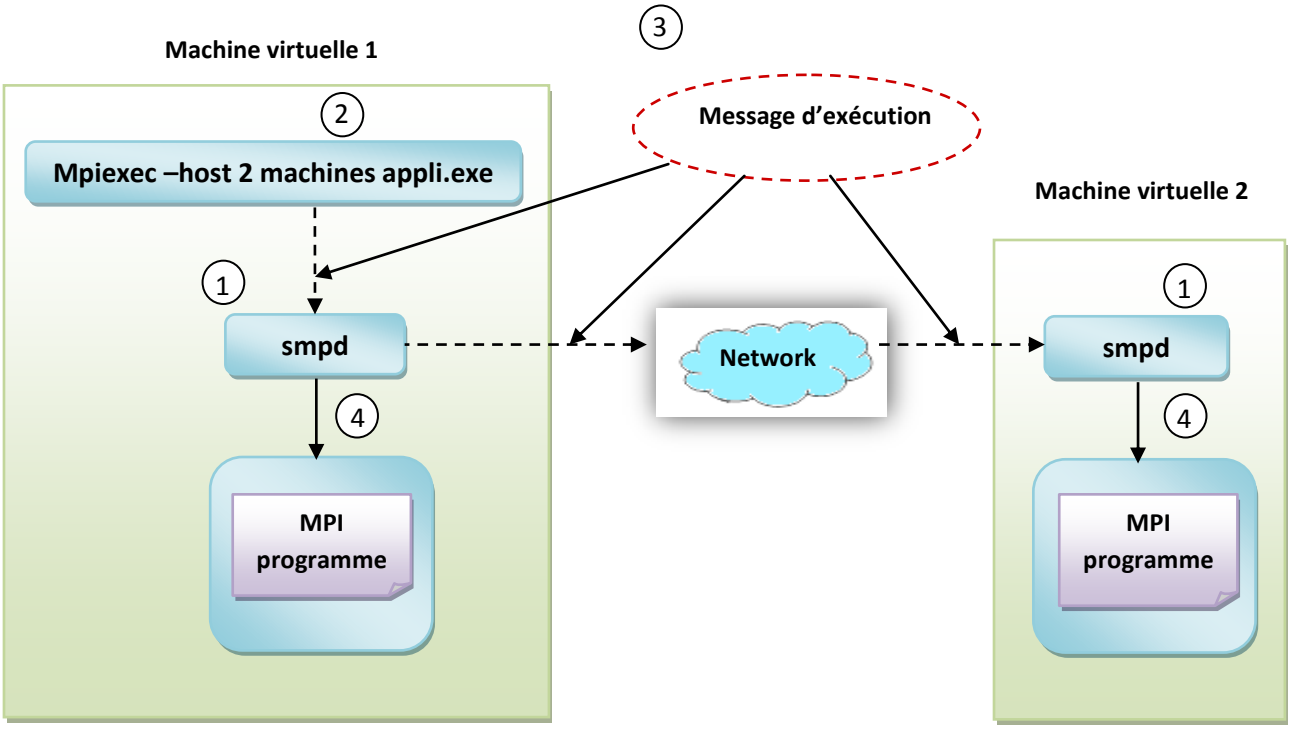

Figure III.1 : Schéma de l'application

Avant toute utilisation de l'application, on doit vérifier la connexion entre les hôtes à utiliser, dans notre cas, l'interface réseau utilisée est celle de virtualBox qui offre un appui de connexion virtuel avec lequel on peut connecter des machines hétérogènes avec des interfaces de communication de différents constructeurs. La vérification de la connexion se fait en général avec la commande 'ping' qui envoie des paquets vers un destinataire X et attend la réponse de ce dernier, si la connexion n'est pas établie, on aura alors une fin de délai d'attente ou hôte injoignable.
La connexion réseau seule n'est pas une condition suffisante pour avoir le bon environnement d'exécution de l'application, on doit vérifier un autre type de connexion, c'est celui demandé par le programme MPICH2. Afin d'assurer cette connexion, on doit vérifier la présence du démon smpd qui représente le lien entre les différents hôtes, et il s'interface avec le programme MPI pour lancer l'application

cible.

L'utilisation de l'application nécessite que les machines virtuelles spécifiées par la commande Mpiexec soient démarrées et en mesure de répondre aux besoins attendus, cette commande, Mpiexec, permet de lancer le programme MPI et accepte comme paramètres le nombre des hôtes à utiliser, leurs noms et le nombre de processus à lancer sur chaque hôte, et comme dernier paramètre, le nom de l'application cible. De plus, pour le bon fonctionnement de l'application, on doit avoir le fichier exécutable (cible) installé sur toutes les machines virtuelles citées dans les paramètres de la commande.

A partir de la figure ci-dessus, on peut citer les grands points suivants:

1. L'utilisateur lance les smpd sur les deux machines virtuelles (MV).
2. L'utilisateur exécute le programme mpiexec pour envoyer les messages d'exécution (arguments sur ligne de commande) au gestionnaire local.
3. Le gestionnaire local de la MV1 distribue le message pour le smpd de la MV2.
4. Les processus de gestion sur les nœuds de calcul exécutent le programme MPI.

III.2 Fonctionnement de l'application

Notre application fonctionne selon le modèle présenté sur la figure ci-dessous (figure III.2) :

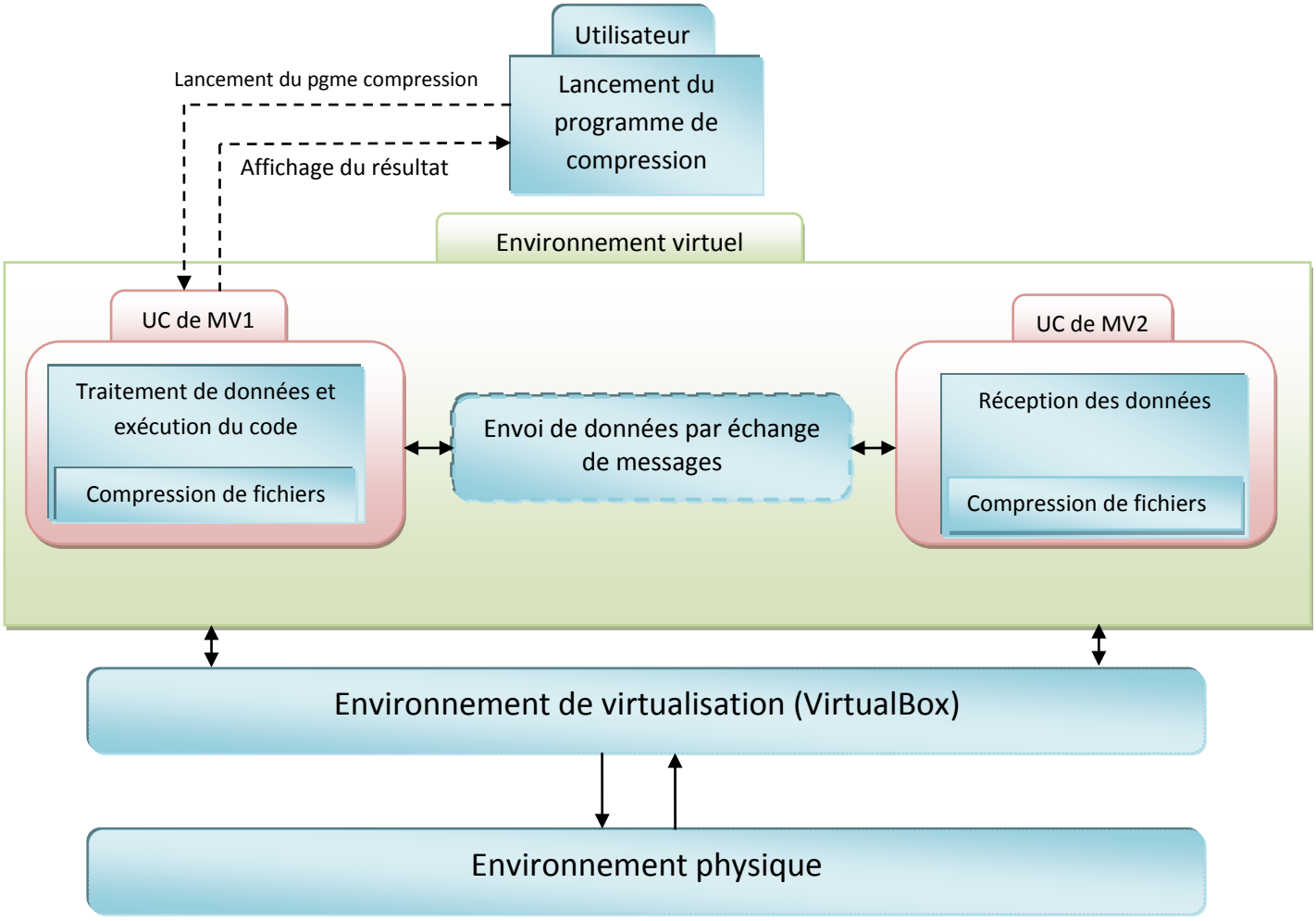

Figure III.2: fonctionnement général de l'application

Avant d'entamer la description du fonctionnement, on doit citer quelques notions importantes sur l'environnement de l'application :

- **Environnement physique** : Machine physique sur laquelle résident les environnements de machines virtuelles. Le nombre de machines virtuelles qui peuvent être prises en charge sur une machine physique varie selon la configuration et les caractéristiques du matériel. Il héberge le **Système d'exploitation hôte** qui est le système d'exploitation principal.

- **Environnement de virtualisation** : Logiciel de virtualisation qui coordonne avec le système d'exploitation hôte les demandes émanant des machines virtuelles concernant le temps UC, la mémoire physique, les opérations de lecture et d'écriture sur les disques, les entrées/sorties (E/S) sur les réseaux, etc. Dans notre cas, ce logiciel est appelé VirtualBox qui est un composant primordial de la technologie de virtualisation. Il intercepte les demandes de ressources matérielles émanant des machines virtuelles qu'il héberge et les convertit dans un format lisible par le matériel physique. De même, les demandes émanant du matériel physique sont converties par VirtualBox de sorte qu'elles puissent être interprétées par les machines virtuelles.
- **Environnement virtuel :** Environnement des machines virtuelles créées par le VirtualBox. Les machines virtuelles ont leur propre matériel virtuel (par exemple UC, mémoire RAM, disques, cartes réseau) sur lequel sont chargés les systèmes d'exploitation invités et les applications.

  L'idées est de réaliser une application de compression de fichiers en parallèle en utilisant un réseau virtuel, composé de deux machines virtuelles installées avec Virtualbox. La communication entre les machines ce réseau virtuel est réalisée par échange des messages grace à la bibliothèque d'échange de messages pour machines parallèles « MPI » installée sur les machines du réseau.

  La compression se déroule en deux étapes :

- Compression des fichiers en fichiers .tar
- Compression des fichiers.tar en fichiers .tar.gz

Après avoir lancé les deux machines virtuelles MV1 et MV2, l'utilisateur lance mpiexec, le lanceur de processus pour les applications MPICH2, sur MV1où il spécifie le programme à exécuter, le nombre de processus ainsi que les machines du réseau concernées puis lance l'exécution. L'unité de calcul de machine virtuelle1(UC de MV1) traite les données et exécute le code de compression des fichiers en fichiers.tar.

En parallèle, elle envoie les données par échange de messages à la machine virtuelle MV2 qui est sur son réseau pour qu'elle lance la compression en coopération avec elle. Une fois qu'une partie de la première compression est réalisée, les fichiers.tar résultants sont envoyés au processus libres du réseau pour qu'ils lancent la deuxiéme compression en fichiers .tar.gz. le déroulement de la procedure de compression et son résultat sont affichés sur la machine MV1.

Remarque

Les fichiers compressés peuvent être visualisés par l'utilisateur à l'emplacement spécifié dans le programme.

III.3. Scénario de déroulement du programme de compression

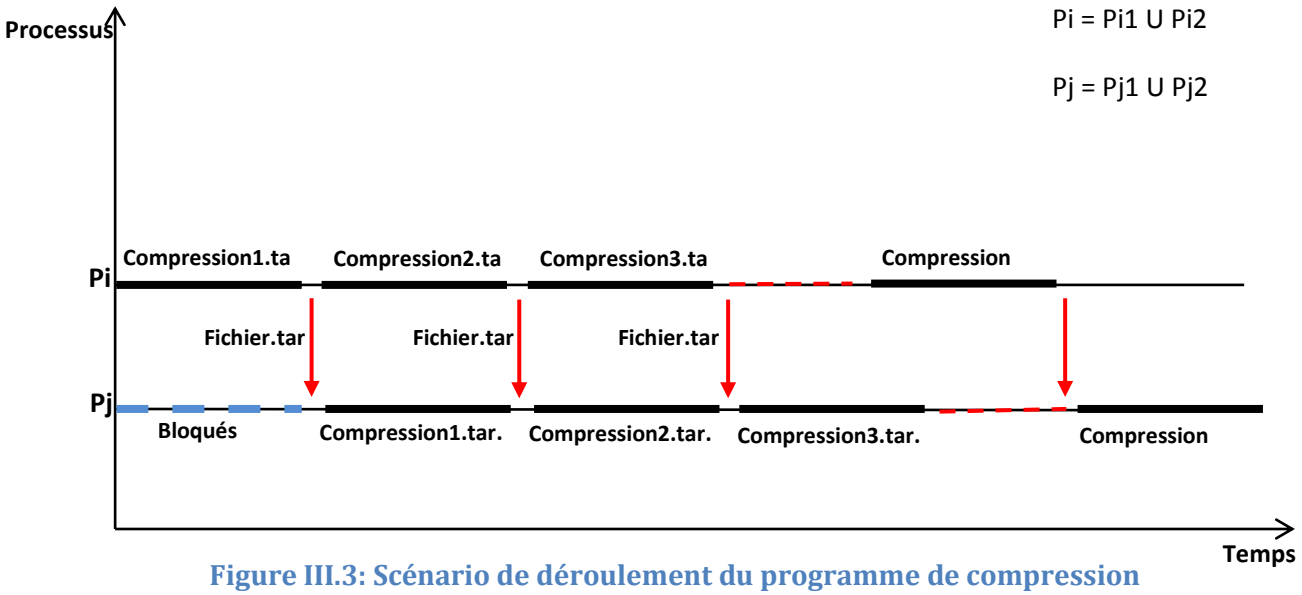

Figure III.3: Scénario de déroulement du programme de compression

Le programme de compression de fichiers se déroule en deux étapes et par la coopération des processus des deux machines MV1 et MV2.

En effet, il y a deux groupes de processus qui se synchronisent pour accomplir cette tâche : les processus **Pi** et les processus **Pj**.

Les processus **Pi1** de l'UC de MV1 initient la compression des fichiers en fichiers.tar et travaillent en coopération avec les processus **Pi2** de l'UC de MV2 pendant que les processus **Pj1** et **Pj2** restent bloqués.

Une fois les processus **Pi1** et **Pi2** produisent des fichiers.tar, ils envoient leurs résultats (fichiers.tar) au deuxième groupe de processus de rang **j (Pj1** de l'UC de MV1 et **Pj2** de l'UC de MV2) et continuent la compression des autres fichiers en fichiers.tar.

Quand les processus **Pj1** et **Pj2** reçoivent les fichiers.tar compressés par **Pi1** et **Pi2**, ils lancent la deuxième étape de la compression (compression.tar.gz) en fichiers.tar.gz (consomment les fichiers.tar et produisent des fichiers.tar.gz) en parallèle avec les processus **Pi1** et **Pi2**.

A la fin de compression, un message s'affiche sur la machine MV1 confirmant le bon déroulement du programme de compression.

III.4 Outils utilisés

Dans le développement de cette application, nous avons utilisé les outils suivants :

III.4.1 Logiciels

- ✓ Windows XP professionnel.
- ✓ Microsoft.NET Framework 3.5.
- ✓ L'environnement MPICH2-1.4.1p1.
- ✓ Devcpp-4.9.9.2.
- ✓ Logiciel de compression, 7-zip dans notre cas.
- ✓ Oracle VM VirtualBox : pour créer les machines virtuelles.

III.4.2 Matériel

Nous avons utilisé une machine hôte de modèle Lenovo G550 win7 PC ayant un système d'exploitation 32 bits, processeur pentium(R) Dual-Core CPU 2,20 GHz, de capacité de mémoire RAM évaluée à 4 Go.

III.5 DEVC++

Dev-C++ est un environnement de développement intégré (IDE) permettant de programmer en C et en C++. Développé avec Borland Delphi 6, Dev-C++ était disponible uniquement sous Microsoft Windows. Longtemps à l'abandon, le projet a été repris par un autre développeur en 2011 et est régulièrement mis à jour.

Il utilise la version MinGW du compilateur GCC (GNU Compiler Collection) (venu du monde du logiciel libre) et permet d'exporter ses projets sous fichiers *.dev*. Dev-C++ peut aussi être utilisé en combinaison avec Cygwin ou tout autre compilateur basé sur GCC.

Cet IDE complet comprend entre autre un « répertoire de classes », servant à localiser facilement les fonctions, classes et membres du code source, un « répertoire de fonctions incluses », fonctionnant comme le répertoire de classes mais pour chercher dans les fichiers inclus (header), et un débogueur qui permet de surveiller l'état des variables pendant l'exécution du programme. Il souffre en revanche de l'absence d'un éditeur de ressources, ce qui rend la conception d'applications délicate si on ne fait pas appel à un outil externe. **[Url 06]**

Différentes versions ont été élaborées, dont voici celle qu'on a utilisé pour développer notre application en lui intégrant la bibliothèque MPI, il s'agit de la version 4.9.9.2 **[Url 06]**.

III.6 Le langage utilisé

Le langage C

Parmi les langages qui supportent le MPI, le langage C qui est un langage de programmation impératif conçu pour la programmation système. Inventé au début des années 1970 avec UNIX, C est devenu un des langages les plus utilisés. De nombreux langages plus modernes comme C++, Java et PHP reprennent des aspects de C. C est qualifié de langage de bas niveau dans le sens où chaque instruction du langage est conçue pour pouvoir être compilée en un nombre d'instructions machine assez prévisible en termes d'occupation mémoire et de charge de calcul. Il propose un éventail de types entiers et flottants conçus pour pouvoir correspondre directement aux types supportés par le processeur. Le langage C a été inventé pour écrire le système d'exploitation UNIX. Il a conservé de cela une très grande efficacité pour tout ce qui

concerne le développement système. Ainsi le noyau de grands systèmes d'exploitation comme Windows et Linux sont développés en grande partie en C.

En contrepartie, la mise au point de programmes en C, surtout s'ils utilisent des structures de données complexes, est plus difficile qu'avec des langages de plus haut niveau. En effet, dans un souci de performance, le langage C impose à l'utilisateur de programmer certains traitements (libération de la mémoire, vérification de la validité des indices sur les tableaux…) qui sont pris en charge automatiquement dans les langages de haut niveau **[19]**.

III.7 Configuration et installation du réseau

III.7.1 configuration des cartes réseau

Après l'installation de virtualBox, on crée les machines virtuelles puis on les configure en réseau, chacune des machines doit être configurée de sorte que :

- sa premiére carte est en mode NAT(Network Adress Translation) comme mode d'accés au réseau réel de la machine hôte.(figure III.1)

- sa deuxiéme carte est en mode privé hôte pour l'accés au réseau réel de la machine hôte.(figure III.2)

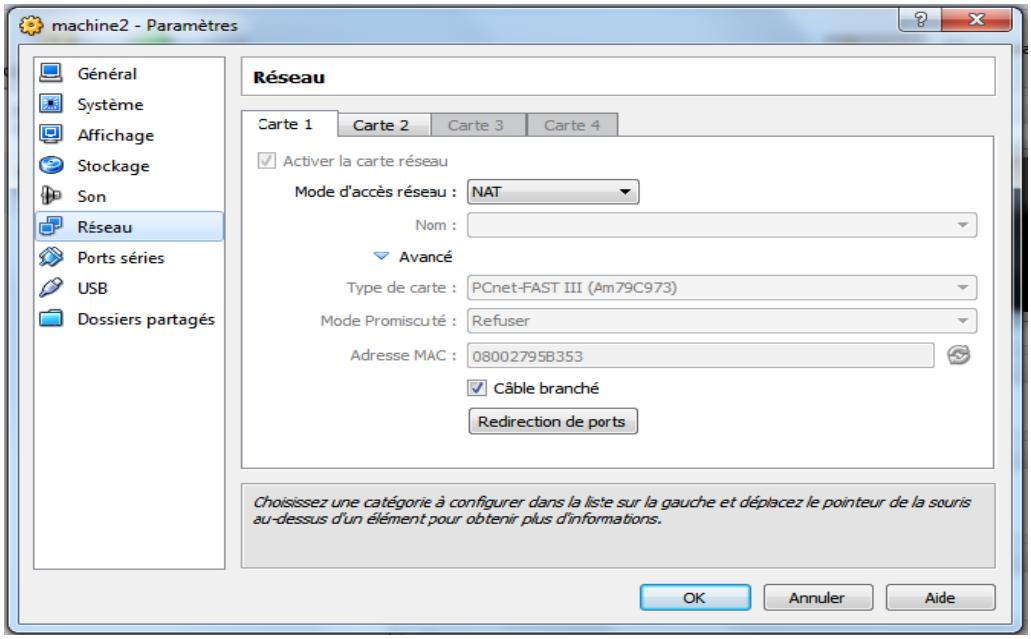

Figure III.4: configuration de la carte 1 en mode NAT

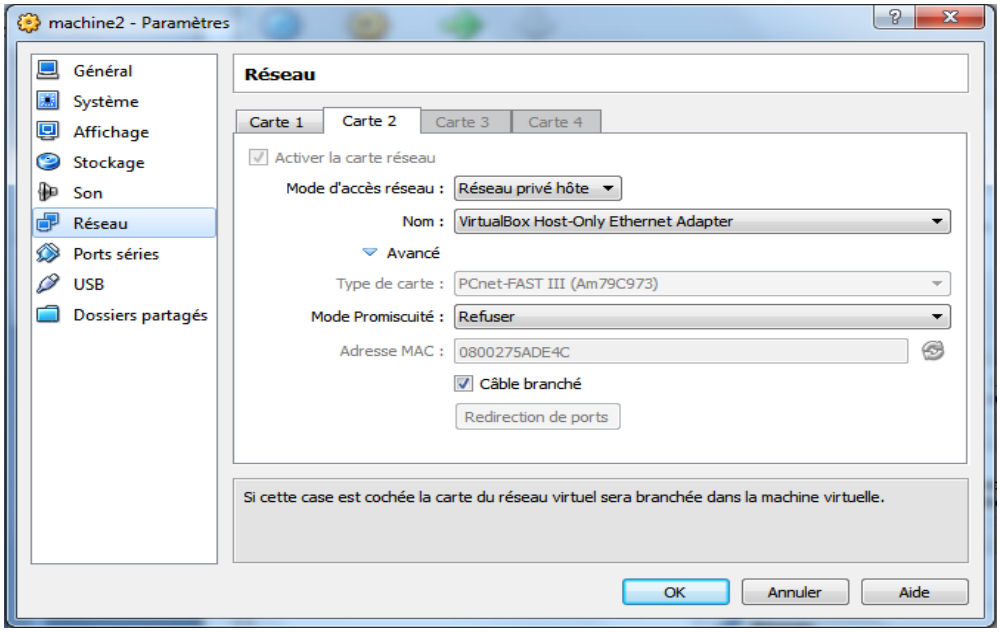

Figure III.5: configuration de la carte 2 en mode privé hôte

De cette façon, les adresses IP des machines virtuelles seront attribuées automatiquement par la machine hôte.

III.7.2 Installation et configuration du MPICH2

Pré-requis :

1. Installer Microsoft Visual C++ 2005 SP1 runtime librairies.
2. Installer Microsoft.NET Framework 3.5.
3. Avoir un compte avec les privilèges d'administration

III.7.2.1 L'installation

Le MPICH2 est disponible gratuitement sur le site officiel **[Url 08],** avant de l'installer sur toutes les machines, on doit d'abord installer la Framework 3.5, et créer une session de travail portant le même nom et mot de passe sur chaque machine virtuelle. L'installation se fait d'une manière classique, il faut juste faire attention à mettre le même mot de passe que celui de la session de travail, et créer un s'il n'existe pas. [Figure III.3]. Enfin choisir de l'installer pour chaque utilisateur de la machine. [Figure III.4]

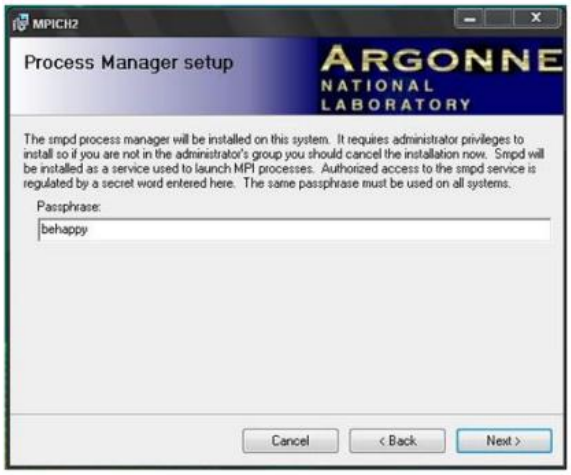

Figure III.6: Installation de MPICH2 (1).

Figure III.7: Installation de MPICH2 (2).

III.7.2.2 La configuration

Pour commencer, il faut d'abord que le nom et le mot de passe des sessions de travail soient identiques pour toutes les machines composant le réseau ainsi que le Workgroup.

- Allez dans le dossier «bin» qui se trouve dans le répertoire «MPICH2» et exécutez le fichier «wmpiregister.exe». Dans la fenêtre qui va apparaitre, saisissez le nom de la machine et de la session de travail, puis le mot de passe. [Figure III.5]. Cliquez sur «Register», ensuite sur «OK».

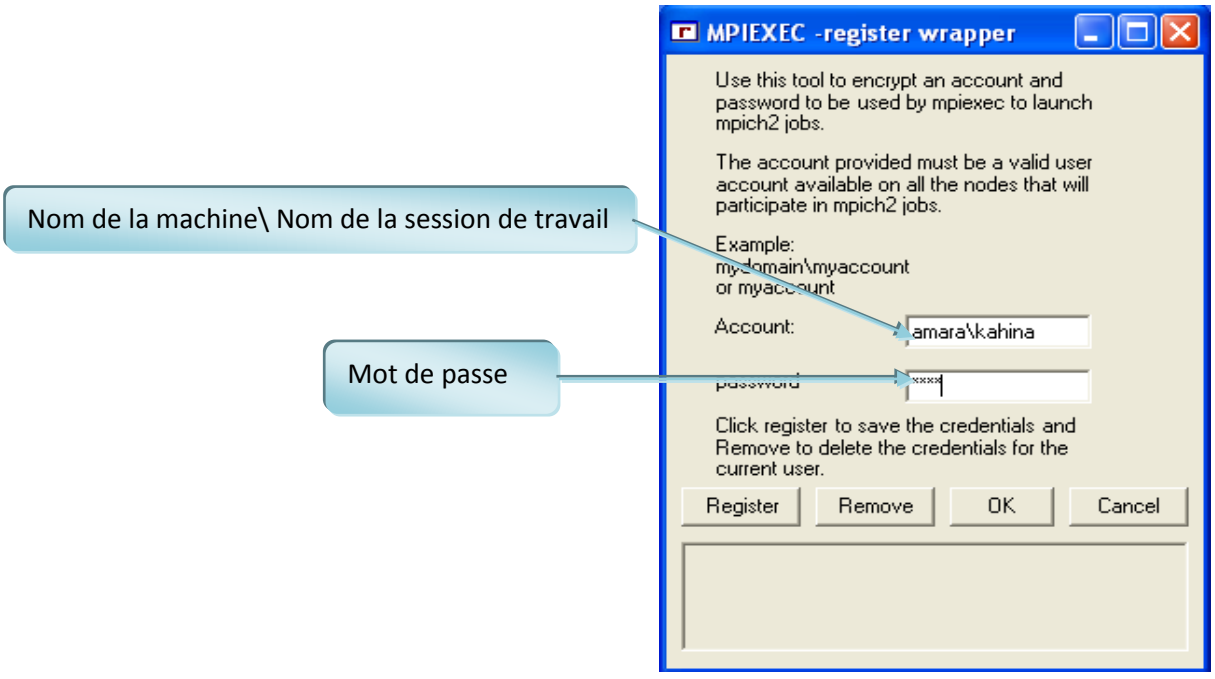

Figure III.8: Authentification MPICH2

- Sur la machine qui va lancer l'exécution du programme MPI, revenez dans le dossier «bin» et exécutez le fichier «wmpiconfig.exe».

Dans le champ «Domain», sélectionnez votre groupe de travail (WORKGROUP dans notre cas), cliquez sur «Get Hosts» puis sur «Scan Hosts». [Figure III.6]

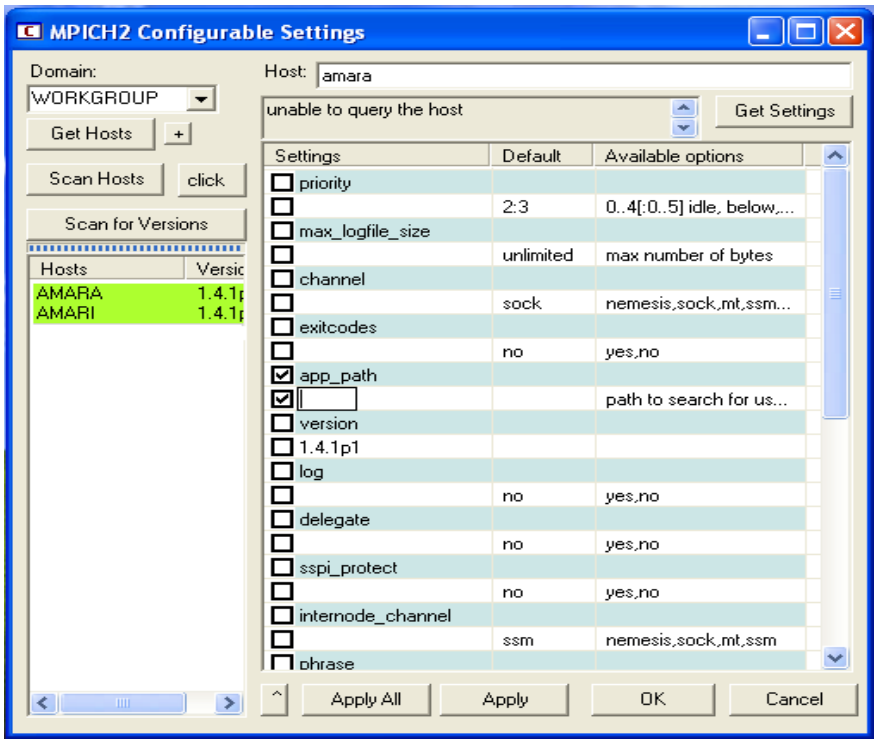

Figure III.9: Configuration de MPICH2

Terminez la configuration de MPICH2 en cliquant sur «Apply All» puis «OK». Pour que le programme MPI s'exécute sur toutes les machines, tous les noms de ces dernières doivent être colorés en vert comme cela est indiqué dans la figure précédente.

Remarques

- Le répertoire «bin» de MPICH2 doit être ajouté aux variables d'environnement du système d'exploitation.
- Comme le modèle utilisé par MPICH2 est de type SPMD, donc le programme à exécuter doit être présent dans un même répertoire sur toutes les machines.
- toutes les machines doivent appartenir au même workgroup.

III.7.2.3 Intégration de la bibliothèque MPI dans le Dev C++

A ce stade, nous allons aborder la méthode d'intégration de la bibliothèque MPI au sein du Visual Dev C++ 2005, et on finira par donner un exemple d'exécution d'un simple programme MPI. Installer Dev-Cpp 4.9.9.2, l'installation se fait d'une manière classique.

1. Télécharger MPI project template et l'extraire à C:\Dev-Cpp\Templates
2. Créer un nouveau projet, *File - New - Project* et choisir *MPI Application*
3. Choisir *C Project* et donner un nom au projet, cliquer sur OK. (figure III.7)

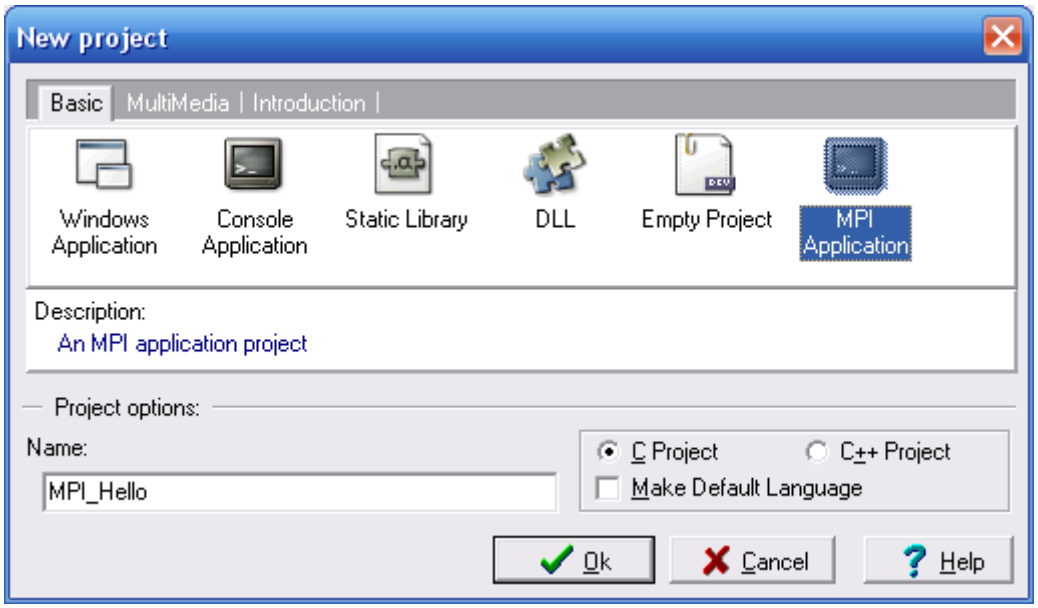

Figure III.10: Création d'un projet MPI.

Exemple d'un programme MPI: MPI_Hello

```
#include <stdio.h>
#include <mpi.h>

#define ROOT 0

int main(int argc, char **argv)
{

  int myrank, nprocs;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

  // TODO: Parallel code here
  printf("Hello from processor %d of %d\n", myrank, nprocs);

  MPI_Finalize();

  return 0;
}
```

Figure III.11 : Exemple d'un programme MPI: MPI_Hello.

Exécution

1. Aller dans le menu Démarrer-*tous les programmes* - *MPICH2* - *wmpiexec.exe*.
2. Entrer le chemin de l'exécutable du programme MPI et le nombre de processus, cliquer sur le bouton d'exécution.
3. Le résultat du programme va s'afficher au dessous de la fenêtre (figure III.8)

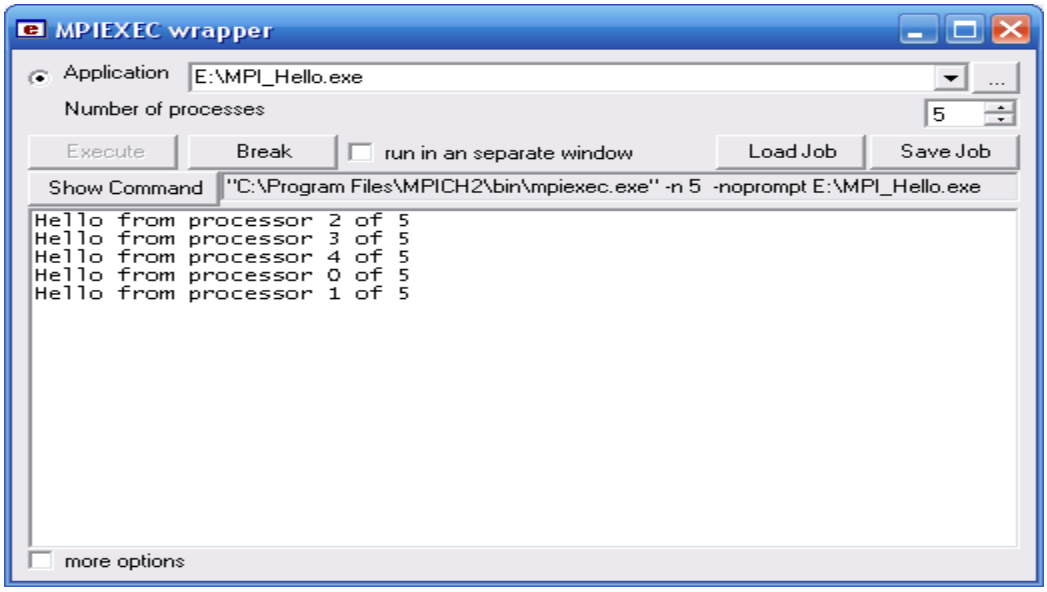

Figure III.12: Exécution d'un programme MPI

Ainsi se termine l'installation et la configuration des outils nécessaires au bon déroulement de notre application.

III.8 Les interfaces

Dans le but de faciliter l'interaction de l'utilisateur avec l'application, on a conçu une interface pour gérer l'application

Figure III.13 : Interface exécution de l'application

- «**choisir la machine** » : c'est une liste déroulante qui permet de choisir la machine d'exécution.
- « **exécuter le programme** » : permet d'exécuter le programme.
- **« Quitter » :** permet de quitter l'interface de l'application

Remarque

L'exécution d'un programme en parallèles peut se faire en local ou en réseau.

III.9 Résultats et discussion

Temps d'exécution de notre programme

Le tableau suivant montre les différentes mesures du temps d'exécution obtenues pour un nombre de processus variant de 2 à 20 processus utilisant un mode d'envoi bloquant standard (MPI_Send()):

Nombre de processus	2	3	4	5	6	7	8	9	10	15	20
Temps d'exécution(seconde)	0.48	0.29	0.32	0.44	0.48	0.49	0.58	0.64	0.65	1.13	1.7

Tableau III.1 : Mesures du temps d'exécution pour 20 processus (envoi standard bloquant)

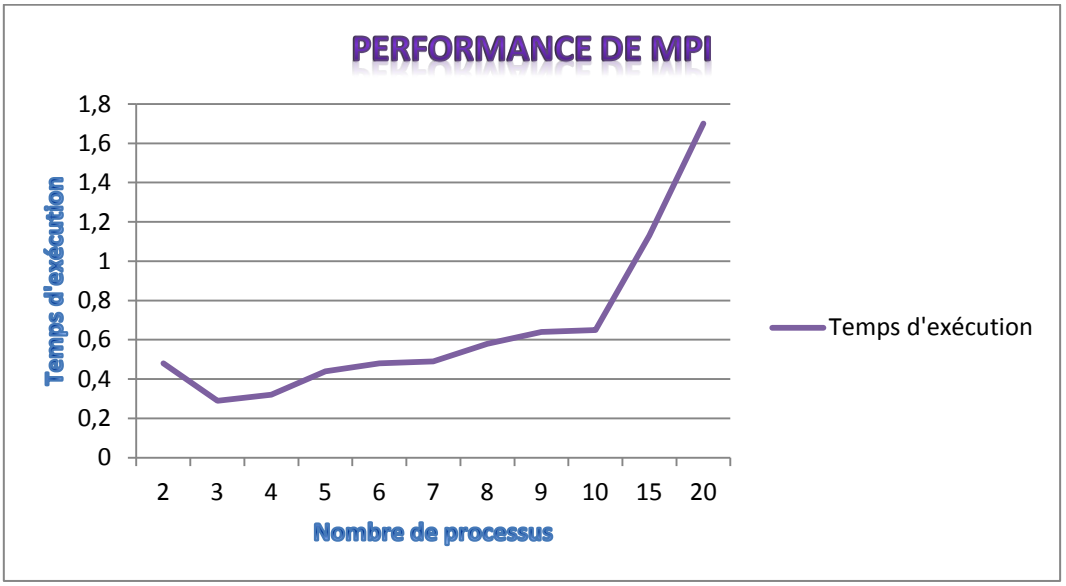

Figure III.14 : Résultats du temps d'exécution (envoi standard bloquant)

La figure ci-dessus montre les différentes mesures du temps des processus obtenues lors de l'exécution de notre programme, celles-ci doivent être mesurées via la fonction MPI_WTIME () qui calcule le temps d'exécution des processus parallèles en secondes.

Comme indiqué dans la **figure III.14**, le nombre de processus utilisés varie de 2 à 20 processus et la meilleure performance a été obtenue pour un nombre de processus N= 3 qui correspond à un temps de performance de 0.29 secondes.
Quand on augmente le nombre de processus (N> 3), le temps d'exécution augmente.
L'une des principales raisons de cette augmentation dans le temps d'exécution pourrait être le type de transmission utilisée qui est l'envoi standard bloquant, réception des commandes utilisées dans notre application, le blocage et mise en mémoire tampon.

Une autre raison pourrait être liée à l'utilisation d'un réseau virtuel au lieu d'un réseau physique, ce qui peut être expliqué par la compétition entre les processus sur les ressources de la machine.

Essayons maintenant de réaliser le même travail mais en utilisant un mode d'envoi standard non bloquant (MPI_Isend()) :

Nous avons obtenu le tableau suivant :

Nombre de processus (N)	2	3	4	5	6	7	8	9	10	15	20
Temps d'exécution(seconde)	0.32	0.4	0.4	0.52	0.6	0.69	0.86	0.9	1.13	1.72	2.27

Tableau III.2 : mesures du temps d'exécution pour 20 processus (envoi standard non bloquant)

Le graphe correspondant est :

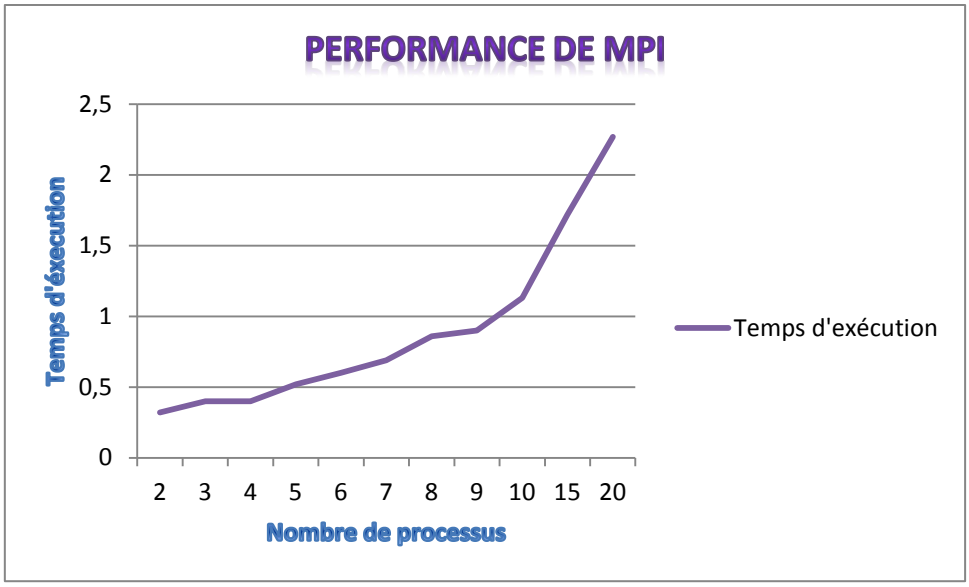

Figure III.15 : Résultats du temps d'exécution (envoi standard non bloquant)

Comme indiqué dans la **figure III.15**, la meilleure performance a été obtenue pour un nombre de processus N= 2 qui correspond à un temps de 0.32 secondes.

Quand on augmente le nombre de processus (N> 2), le temps d'exécution augmente.

Cette augmentation peut être liée au mode d'envoi utilisé qui est l'envoi standard non bloquant qui fait la recopie temporaire des messages dans un espace mémoire temporaire (Buffer), comme elle peut être liée à l'utilisation d'un réseau virtuel au lieu d'un réseau physique.

Discussion

Les résultats de performance obtenus dans les graphes précédents montrent que le temps d'exécution augmente lorsqu'on augmente le nombre de processus dans les deux tests, la performance maximale obtenue correspond à un nombre de processus N appartenant à l'intervalle de [2-3]. Le deuxième test devait donner des résultats plus performants vue l'utilisation du mode d'envoi non bloquant. Ceci peut être interprété par l'utilisation d'un réseau virtuel où il pourrait y avoir compétition entre les processus sur les ressources de la machine hôte.

III.10 Optimisation d'un programme parallèle

Lorsqu'on parallélise un code, un autre temps s'ajoute au temps de calcul, c'est le temps de communication entre les processus i.e. temps total = temps calcul + temps communication.

L'optimisation d'un code parallèle consiste à minimiser le temps de communication entre les processus. Celui-ci contient un temps de préparation du message et un temps de transfert. Le temps de préparation contient un temps de latence pendant lequel les paramètres réseaux sont initialisés. Le reste du temps de préparation des messages (appelé aussi temps de surcoût) est lié à l'implémentation MPI et au mode de transfert utilisé (voir figure III.16 pour plus de détails).

Il existe plusieurs possibilités pour optimiser le temps de communication, parmi elles :

– limiter les modes de transfert qui utilisent la recopie du message dans un espace mémoire temporaire (buffering).

– limiter les appels répétitifs aux fonctions de communication MPI (qui coûtent cher en temps). **[20]**

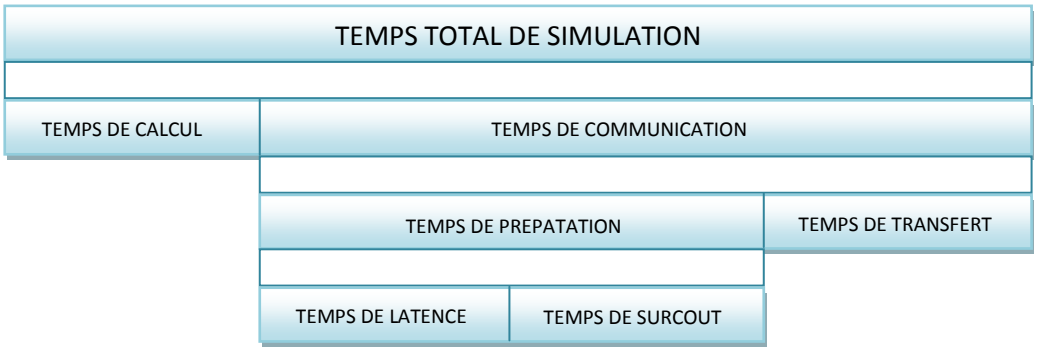

Figure III.16 : composition du temps total de simulation d'un programme [20]

Conclusion

Ce chapitre a été un espace pour illustrer l'installation et la configuration des outils nécessaires au bon déroulement de notre application, les différentes étapes de l'exécution de cette application de compression de fichiers en parallèles, nous avons vu notamment les résultats d'exécution de cette application.

CONCLUSION & PERSPECTIVES

Conclusion et perspectives

Le travail que nous avons réalisé a pour but l'installation et la configuration d'un environnement virtuel puis la mise en place d'une application de compression des fichiers en utilisant le parallélisme virtuel, nous avons effectué une parallélisation du programme de compression des fichiers par 7-zip en deux étapes : la compression en fichiers.tar puis la compression de ces derniers en fichiers.tar.gz.

Pour y arriver, nous sommes passées par un certain nombre d'étapes. La première phase consiste essentiellement à présenter la virtualisation ensuite nous avons poursuivi par des généralités sur les architectures parallèles et le modèle de programmation par passage de messages. Enfin, nous avons abordé la conception de notre application ; nous sommes ensuite passées à la mise en œuvre durant laquelle nous avons présenté les différents outils et technologies utilisés pour l'implémentation de notre application.

Outre les aspects liés directement à notre application, nous avons eu l'occasion de nous approfondir dans la programmation parallèle en utilisant la bibliothèque de passage de messages MPI sous l'environnement MPICH2, des connaissances supplémentaires dans le langage C utilisé pour écrire les programmes MPI. On note aussi notre initiation aux réseaux virtuels.

Difficultés rencontrées

Notre première difficulté résidait dans la configuration de l'environnement MPICH2, en effet, la moindre erreur de connexion ou de configuration provoque son échec, il fallait donc réessayer à plusieurs reprises avant de pouvoir exécuter un programme parallèle sous MPICH2. Une autre difficulté était la différence de versions de la Framework ou de sa mise à jour entre les différentes machines virtuelles constituant le réseau, ceci provoquerait aussi des fois un arrêt complet de l'environnement.

Perspectives

Espérant que les années à venir vont apporter plus d'intérêts à la programmation parallèle en utilisant la MPI vu que cette technologie est encore inexploitée, malgré son plein essor à travers le monde, et que les prochaines études portant sur ce thème prendront en considération les remarques citées ci-dessus.

REFERENCES

Références

Ouvrages et Articles

Traces	Ouvrages et articles
[01]	groupe LINAGORA, Livre blanc *« la virtualisation »*, Paris.
[02]	Dostes, *« La virtualisation »*, Strasbourg, 10 Juin 2011.
[03]	OSSIR, « Retour d'expérience sur un déploiement VMWARE à grande échelle », Septembre 2007.
[04]	Arthur Delorme, *« Utilisez plusieurs systèmes d'exploitation avec VirtualBox »*, IPGP, février 2011.
[05]	Djillali Seba, « *Interconnexion des réseaux à l'aide des routeurs et commutateurs »*, ENI, novembre 2003
[06]	Alexandre DENIS, *« Contribution à la conception d'une plate-forme haute performance d'intégration d'exécutifs communicants pour la programmation des grilles de calcul »*, 2003.
[07]	Mohamed JEMNI, **«** Introduction aux algorithmes et architectures parallèles », 2004.
[08]	SEBTI SAID et TAGUEMOUNT ADEL, **«** parallélisation de l'algorithme de Deriche », 2004.
[09]	Pierre-Eric Bernard, « Parallélisation et multiprogrammation pour une application irrégulière de dynamique moléculaire opérationnelle », 1997.
[10]	Alexandre da Silva, «Exploitation de la multiprogrammation légère sur grappes de Multiprocesseurs », CARISSIMI, 1999.
[11]	Jean-Paul Sansonnet et franck cappello, « *Architecture des Machines Parallèles* ».
[12]	Rajkumar Buyya, «*Parallel Programming with Message-Passing Interface (MPI) »*, Melbourne (Australia), 2008.

[13]	Michael J. Quinn, «Parallel programming in C with MPI and OpenMP», 2004.
[14]	NICAISSE Sébastien, « *Analyse et mise en œuvre d'un cluster dans le domaine astronomique* », Strasbourg, 2002-2003.
[15]	Etienne Gondet & Pierre-François Lavallée, « Cours OpenMP », 2000.
[16]	Aurélia Marchand, « Les architectures parallèles : les machines parallèles de l'observatoire », 2005.
[17]	Laurent COLOMBET, « Parallélisassions d'applications pour des réseaux de processeurs homogènes ou Hétérogènes », 1992.
[18]	Nahla Techini Ep Ben Salem, « Configuration et mise en place d'un Datacenter sécurisé dans un environnement virtuel », 2010/2011.
[19]	Brian Kernighan & Dennis Ritchie, «The C Programming Language », 1990.
[20]	Stéphanie DELAGE SANTACREU, «Introduction au calcul parallèle avec la bibliothèque MPI », Novembre 2008.

Liens Internet

[Url 01]	www.commentçamarche.net
[Url 02]	http://www.virtualbox.org/manual/ch06.html
[Url 03]	http://root-lab.fr/2012/01/31/theorie-des-reseaux-virtuels/
[Url 04]	http://www.oamp.fr, m.à.j 02/02/2010
[Url 05]	http://www.dil.univ-mrs.fr
[Url 06]	http://www.wikipedia.org
[Url 07]	https://computing.llnl.gov/tutorials/mpi/
[Url 08]	http://iproc.ru/programming/mpich-windows/
[Url 09]	http://www.mcs.anl.gov/research/projects/

ANNEXE

On donne ci-dessous un fragment du code C de la compression des fichiers en parallèle :

```
 #include<stdio.h>
#include "mpi.h"
main (int argc, char* argv[])
{
        int        procID, nprocs, i, etiquette=99;
        double  temps_debut, temps_fin, temps_fin_max;
        MPI_Status  statut;
        const int  maximum_message_length = 100;
        char      command[maximum_message_length];  /* Message string        */
        MPI_Init(&argc, &argv);  // Initialisation de  MPI
        MPI_Comm_rank(MPI_COMM_WORLD,&procID);  // Récupèrer le rang du processus courant
        MPI_Comm_size(MPI_COMM_WORLD,&nprocs); // Récupèrer le nombre de processus

       char message1 [] = "C:\\systeme1.tar";
       char message2 [] = "C:\\systeme2.zip";
      temps_debut = MPI_Wtime();

      system("C:\\7-Zip\\7z.exe  a  -t7z  C:\\systeme1.tar  C:\\test");

       // Communications
        if (procID == 0)
        {
        MPI_Send(message1, strlen(message1) + 1, MPI_CHAR, 1, etiquette, MPI_COMM_WORLD);
        Printf (" moi processus %d, j'envoie %s au processus 1\n\n", procID, message1);
        }
        else if (procID == 1)
        {
        MPI_Recv (message1, maximum_message_length, MPI_CHAR, 0, etiquette,
MPI_COMM_WORLD, &statut);
        Printf ("Moi, processus 1, j'ai reçu %s du processus 0 \n\n", message1);
        System ("C:\\7-Zip\\7z.exe  a  C:\\systeme2.zip  C:\\systeme1.tar");
       MPI_Send(message2, strlen(message2) + 1, MPI_CHAR, 0, etiquette, MPI_COMM_WORLD);
       printf( " moi processus %d, j'envoie %s au processus 0\n\n", procID, message2) ;
        }
        temps_fin = MPI_Wtime();
        // calcul du temps d'exécution
        temps_fin_max = temps_fin-temps_debut;
        // affichage du temps d'exécution
 printf( "Temps écoule = %f.\n\n", temps_fin_max);
 // Finalisation du programme
        MPI_Finalize();
}
```