

REPUBLIQUE DU CAMEROUN
REPUBLIC OF CAMEROON
Peace-Work-Fatherland

UNIVERSITE DE DSCHANG
UNIVERSITY OF DSCHANG
Scholae Thesaurus Dschangensis Ibi Cordum

BP 96, Dschang (Cameroun)

Tél. /Fax (237) 233 45 13 81

Website : <http://www.univ-dschang.org>.

E-mail : udsrectorat@univ-dschang.org



FACULTE DES SCIENCES
FACULTY OF SCIENCE

*Département de Mathématiques et
Informatique*
*Department of mathematics and Computer
Science*

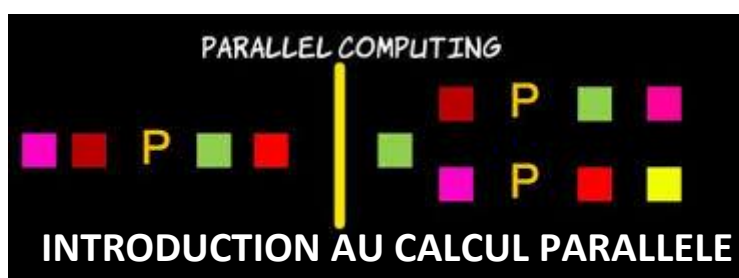
BP 96, DSc hang (Cameroun)

Tél. /Fax (237) 233 45 13 81

Website : <http://fs.univ-dschang.org>.

E-mail : dept.math-info@univ-dschang.org

COMPLEXITE ET ALGORITHMES AVANCES



Noms et prénoms des membres du groupe	Matricules	SPECIALITE
KENGNE WAMBO DARIL RAOUL	CM-UDS-18SCI0131	IA
TEMFACK DERICK	CM-UDS-18SCI0797	IA
FOAM TOUKAM CINDY LENA	CM-UDS-18SCI0092	IA
NGANFANG VICTOIRE CABRELLE	CM-UDS-18SCI1828	IA

Enseignant : Pr. KENGNE TCHENDJI VIANNEY

Année académique
2021/2022

Table de matières

LISTES DES FIGURES	iv
LISTE DES TABLEAUX.....	v
INTRODUCTION.....	vi
I. HISTORIQUE ET PRINCIPE DU PARALLELISME	7
1. La petite histoire du parallélisme.....	7
2. Quelques définitions	8
3. Principe du calcul parallèle	9
II. LES APPLICATIONS DU PARALLELISME.....	13
1. Le parallélisme dans la géologie.....	13
2. Deep Blue et le parallélisme.....	14
3. Le parallélisme dans les marchés boursiers et le Business Intelligence (BI)	15
4. Le parallélisme dans la météorologie.....	16
5. Simulations dans le domaine Aérospatial.....	17
6. Les serveurs et le parallélisme.....	17
a. Nginx	17
b. Unicorn.....	19
7. Le parallélisme et les ordinateurs quantiques	19
8. La blockchain et le parallélisme.....	20
III. LES DIFFÉRENTS TYPES DE PARALLELISME.....	22
1. Classification suivant le flot d'exécution d'instructions	22
2. Classification suivant la topologie du réseau d'interconnexion.....	28
3. Classification suivant la granularité du calcul.....	29
4. Classification selon la connectivité des unités de calcul	30
5. Classification suivant les mémoires des machines parallèles.....	31
IV. Modèles de calcul parallèle	33
1) Le modèle PRAM	33
2) Le modèle systolique	36
3) Le modèle BSP.....	37
4) Le modèle grille à deux dimensions et l'hypercube.....	39
5) Modèle CGM.....	41
V. CALCUL DE LA COMPLEXITÉ DES ALGORITHMES PARALLÈLES	43
VI. Architecture parallèle à mémoire partagé avec OpenMp	45
1. OpenMP, c'est quoi ?	45
2. Le modèle « fork-join » de openMP.....	47

3. Construction d'openMP définissant le partage du travail.....	48
VII. Architecture parallèle à mémoire distribuée avec MPI	48
1. MPI c'est quoi ?	49
2. Pourquoi utiliser MPI ?	50
VIII. Architecture parallèle hybride avec openMP et MPI	50
a. Avantages de la programmation hybride.....	51
b. Inconvénients de la programmation hybride.....	52
IX. Les avantages et limites du parallélisme et comparaison avec le paradigme séquentiel...	52
1. Avantages	53
2. Limites	53
3. Comparaison du paradigme parallèle et du paradigme séquentielle	54
CONCLUSION	56
TRAVAUX DIRIGES.....	57
BIBLIOGRAPHIE	61
WEBOGRAPHIE.....	61

LISTES DES FIGURES

Figure 1: Multi-thread dans un cœur de processeur	10
Figure 2: distributed computing architecture	12
Figure 3: Distributed architecture link to a server	12
Figure 4: World Community group logo.....	12
Figure 6:Interprétation sismique de Dug Software	14
Figure 5:Equipement de Dug Software	14
Figure 7: Garry Kasparov Vs Deep Blue	15
Figure 8: Bourses de Wall Street.....	16
Figure 9: Analyses dans les marches bousiers	16
Figure 10:Gestion des tâches a l'aide des workers dans un serveur	18
Figure 11: Usage de 5 worker en simultané pour traiter les requêtes arrivant sur Unicorn ...	19
Figure 12: Fonctionnement globale de la blockchain	21
Figure 13:Architecture SISD	25
Figure 14: Architecture SIMD	26
Figure 15: Architecture MISD	26
Figure 16: Architecture MIMD	27
Figure 17: Réseau parallèles topologiques	29
Figure 18: Carte processeur de T3E-600.....	31
Figure 19: Schéma explicatif pour la classification à mémoire partagée.....	32
Figure 20: Schéma explicatif pour la classification à mémoire distribuée.....	32
Figure 21 : Structure abstraite du modèle PRAM	35
Figure 22: Synchronisation des barrières	38
Figure 23:Grille a deux dimensions de taille 16	40
Figure 24: hypercubes de dimensions 2, 4, 8, 16 et 32	41
Figure 25: Architecture parallèle à mémoire centralisée.....	45
Figure 26 : Modèle "Fork-Join" de openMP	47
Figure 27: Construction d'openMP définissant le partage du travail.....	48
Figure 28: Représentation visuel des architectures à mémoire distribuées.....	49
Figure 29: Modèle hybride	51

LISTE DES TABLEAUX

Table 1: Résumé de la taxonomie de Flynn	23
Table 2:Recapitulatif de la taxonomie de Kuck	23
Table 3:Recapitulatif de la taxonomie de Treleaven.....	24
Table 4: recapitulatif de la taxonomie de Gajski.....	24
Table 5:Comparaison entre les paradigmes séquentiels et parallèles:	55

INTRODUCTION

De nombreuses décennies après la naissance des ordinateurs qui réalisaient déjà la plupart des tâches pour lesquelles ils étaient construits, au même titre que leur miniaturisation vers laquelle les concepteurs ou alors les grandes firmes se penchaient, l'on désirait réaliser dans une machine plusieurs tâches simultanées et "indépendantes" ce qui la rendra plus rapide et performante. C'est de là qu'est née le parallélisme avec l'apparition dans les années 80, de la fameuse "**connexion machine**" qui était un gros ordinateur composé de plusieurs processeurs, coûtant jusqu'à un million de dollars. Mais, ce n'est que dans les années 2000 avec l'avènement des processeurs multicœurs que l'on intègre réellement le parallélisme dans nos différents équipements avec l'exécution des tâches de manière simultanée.

À ce niveau naît une panoplie d'interrogations à savoir **dans quels cadres le parallélisme pourrait nous être utile ?** Pour être un peu plus pragmatique, **quels sont les différents projets et applications réalisées utilisant le calcul parallèle dans le quotidien ?** L'intérêt étant saisi, **comment fonctionne le calcul parallèle ?** Plus loin, **en quoi est-il avantageux par rapport au paradigme séquentiel traditionnel et quels en sont donc les avantages et les limites ?** Pour ainsi finir, **si l'on voudrait le réaliser, comment procéderons-nous ?**

Tout ce chapitre sera axé sur ces questions. Nous présenterons tout d'abord le principe et les applications du calcul parallèle, puis les différentes classifications et modèles du calcul parallèle, après quoi viendra la présentation de l'interface MPI nécessaire dans l'architecture parallèle ; ensuite, nous réaliserons quelques programmes parallèles avec OpenMP et en fin présenterons les avantages et limites du parallélisme.

I. HISTORIQUE ET PRINCIPE DU PARALLELISME

1. La petite histoire du parallélisme

L'intérêt pour le calcul parallèle remonte à la fin des années 1950, avec des avancées sous la forme des supercalculateurs dans les années 60 et 70. Il s'agissait de multiprocesseurs à mémoire partagée, avec plusieurs processeurs travaillant côte à côte sur des données partagées. Au milieu des années 1980, un nouveau type de parallélisme informatique a été lancé lorsque le **projet Caltech Concurrent Computation** a construit un supercalculateur pour applications scientifiques à partir de 64 processeurs Intel 8086/8087. Ce système a montré que l'extrême des performances pourraient être atteintes avec des microprocesseurs du marché de masse. C'est massivement que les processeurs parallèles (MPP) en sont venus à dominer le haut de gamme de l'informatique, avec **l'ASCI Red** supercalculateur en **1997** franchissant la barrière des mille milliards d'opérations en virgule flottante par seconde. Depuis lors, les MPP n'ont cessé de croître en taille et en puissance.

À partir de la fin des années 80, les clusters sont entrés en concurrence et ont finalement remplacé les MPP pour de nombreuses applications.

2. Quelques définitions

Avant de donner le principe du calcul parallèle, il est important que nous comprenions et fixions les sens de certains termes qui d'ailleurs tout au long de ce chapitre nous seront très utiles.

La Mémoire : c'est un dispositif électronique servant à stocker des informations. Elle est dite partagée lorsque plusieurs processeurs peuvent y accéder simultanément et distribuée lorsqu'elle est répartie en plusieurs nœuds, chaque portion n'étant accessible qu'à certains processeurs.

Un processeur quant à lui est un composant présent dans de nombreux dispositifs électroniques, ayant pour rôle d'exécuter les instructions machines et les programmes informatiques. Il est dit **multicœurs** s'il possède plusieurs cœurs physiques fonctionnant simultanément ; un cœur physique étant tout simplement un ensemble de circuit capable d'exécuter des programmes de façon autonome (réalisant les instructions de Lecture, de décodage, d'écriture et de rangement).

Une architecture en informatique désigne l'organisation des différents éléments d'un système informatique (logiciel, matériel, humain, information...), ainsi que les relations entre eux.

Un processus est l'instance d'un programme en cours d'exécution. Et un **Thread** n'est rien d'autre qu'un processus léger.

Une tâche quant à elle est une liste d'instructions réalisant un certain travail et utilisant des données en entrée dans un ensemble E (ensemble des variables d'entrées) et qui modifient un ensemble de variable S (variables de sortie). Selon que la tâche soit petite ou grosse (en termes de nombre d'instructions), on parlera de granularité fine ou grossière.

Le Parallélisme en informatique consiste à mettre en œuvre des architectures d'électroniques numériques permettant de traiter les informations de manière simultanée ainsi que les algorithmes spécialisés pour celles-ci

Un calcul est une opération ou un ensemble d'opérations réalisées sur des grandeurs (les nombres par exemple). Il est dit **parallèle** lorsque l'on lui applique les principes du parallélisme, et ceux dans l'optique d'obtenir plus rapidement le résultat.

Ordinateur parallèle : c'est un ordinateur composé de plusieurs processeurs qui coopèrent à la solution d'un même problème.

Un système distribué (ou repartit) : est un système composé de plusieurs unités de calculs impliquées dans la résolution d'un ou plusieurs problèmes.

Le partitionnement : C'est le découpage du problème complet en tâches, avec l'espoir que certaines de ces tâches pourront être résolues en parallèle. Souvent, dans les cas SPMD (Small Program Multiple Data), le Partitionnement revient à découper le domaine de calcul (c'est l'ensemble des variables d'entrée qui est fractionné). On parle de « **Domain décomposition** ».

Variable partagée : c'est une variable dont le nom permet d'accéder au même bloc de stockage au sein d'une région parallèle entre tâches.

Variable privée : c'est une variable dont le nom permet d'accéder à différents blocs de stockage suivant les tâches, au sein d'une région parallèle.

3. Principe du calcul parallèle

Tout d'abord pourquoi le parallélisme ? Nous avons dans l'introduction donné une ébauche des raisons qui poussent à entreprendre des calculs parallèles. En effet, comme dit, ils sont fortement connexes aux architectures :

- Premièrement, le parallélisme est présent au plus profond de la microarchitecture du processeur. Autrefois, les processeurs exécutaient des programmes en répétant ce qu'on appelle le cycle d'instructions, une séquence de quatre étapes :
- (i) **lecture et décodage d'une instruction**
 - (ii) **trouver les données nécessaires pour traiter l'instruction**
 - (iii) **le traitement de l'instruction**
 - Et (iv) **écrire le résultat dans la mémoire.**

Étant donné que l'étape (ii) a introduit de longs retards dus à l'arrivée des données, une grande partie de la recherche s'est concentrée sur des conceptions qui ont réduit ces retards et de cette façon augmenter la vitesse d'exécution effective des programmes. Au fil des années, cependant, l'objectif principal est devenu la conception d'un processeur capable d'exécuter des instructions simultanément. Le fonctionnement d'un tel processeur a permis la détection et l'exploitation du parallélisme inhérent à l'exécution des instructions. Ces processeurs ont permis des vitesses d'exécution des programmes encore plus élevés, quel que soit la fréquence du processeur et de la mémoire.

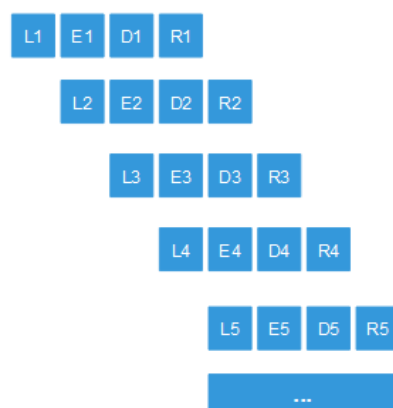


Figure 1: Multi-thread dans un cœur de processeur

- Deuxièmement, tout ordinateur commercial, tablette et smartphone contient un processeur avec plusieurs cœurs, chacun étant capable d'exécuter sa propre instruction flux. Si les flux sont conçus de manière à ce que les cœurs collaborent à l'exécution d'une l'application, cette dernière s'exécute en parallèle et peut être considérablement accélérée.
- Troisièmement, de nombreux serveurs contiennent plusieurs processeurs multicœurs. Un tel serveur est capable d'exécuter un service en parallèle, ainsi que plusieurs services en parallèle.
- Enfin, même les ordinateurs grand public contiennent des processeurs graphiques (GPU) capables d'exécuter des centaines voire des milliers de threads en parallèle. Des processeurs capables de faire face à un si grand parallélisme sont nécessaires pour prendre en charge l'animation graphique.

Il y a de ce fait plusieurs raisons qui ont poussé à ce que l'on parallélise nos machines :

- Tout d'abord, il n'est pas possible d'augmenter indéfiniment les fréquences du processeur et de la mémoire, du moins pas avec la technologie actuelle à base de silicium. Par conséquent, pour augmenter la puissance de calcul des ordinateurs, de nouvelles architectures et organisations des notions sont nécessaires. Nous pouvons en occurrence énumérer projet **Word community grid** de **IBM**, qui au travers des appareils (smart phones, tablettes, ordinateurs) de volontaires exécutent en arrière-plan jeux des expériences et tests dans le cadre de la covid-19.



Figure 2: distributed computing architecture



Figure 3: Distributed architecture link to a server



Figure 4: World Community group logo

- Enfin, le parallélisme est devenu une partie intégrante de tout ordinateur et il est probable qu'il restera inchangé du fait de la simple inertie : le parallélisme peut se faire et il se vend bien.

Le calcul parallèle est donc tout simplement la réalisation des différentes tâches d'une opération de manière simultanée. Ici, l'on découpe le problème initial en plusieurs sous problèmes plus ou moins simple à résoudre confié chacun à différentes unités de traitement (celle-ci peuvent être soit des processeurs, soit des cœurs) pouvant se partager les données ou pas et qui sont d'une manière ou d'une autre liées entre elles afin que le résultat global attendu de l'exécution soit la résolution du problème initial. Pour donc réaliser cette exécution sur différentes

unités de traitement, plusieurs types de parallélisme relatifs aux architectures des composants sont mis sur pied. On en distingue plusieurs types de parallélismes :

- ✚ **Le parallélisme des tâches ou de contrôle** dans lequel un ensemble de tâches indépendantes est réalisé simultanément par différents UC.
- ✚ **Le parallélisme de données** dans lequel un ensemble de tâches est répété sur plusieurs données différentes.

II. LES APPLICATIONS DU PARALLELISME

Tout d’abord, il est nécessaire de prendre connaissance de ce qui a déjà été réalisé en utilisant le parallélisme.

1. Le parallélisme dans la géologie

La géologie est la science dont le principal objet d'étude sont les matériaux qui composent le globe terrestre et l'ordre suivant lequel ces matériaux ont été disposés dans le temps et dans l'espace. Discipline majeure des sciences de la Terre, elle se base en premier lieu sur l'observation, puis établit des hypothèses permettant d'expliquer l'agencement des roches et des structures les affectant afin d'en reconstituer l'histoire et les processus en jeu. Ainsi, la détection sismique par exemple prenant en compte de nombreux paramètres sur un temps relativement long, nécessite de grandes puissances de calculs pour l'analyse et l'interprétation des métadonnées (la stratigraphie, la porosité du sol, les différents minéraux constitutifs de la région étudiée, les pressions, etc.) suivant leur différente projection à l'aide des **machines vectorielles** qui utilisent des algorithmes

parallèles(démontré par un groupe de chercheur de l'université d'Algarve au Portugal).

Nous pouvons donc ici énumérer la plateforme **Dug Software** qui a l'aide de son super ordinateur (d'ailleurs l'un des plus grand su monde avec des équipements déployés sur plus de la moitié d'un stade de football) analyse à partir d'algorithmes parallèles et donne des prédictions sismiques sur des reliefs. Cette étude a été réalisée par les **Docteurs Matthew Lamont** et **Troy Thompson** spécialistes en géophysique de **Curtin University en Australie** et en algorithme d'imagerie respectivement.

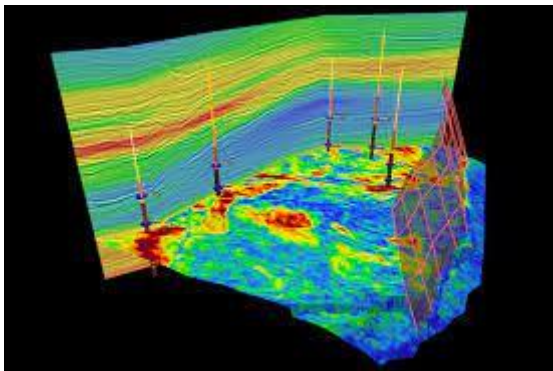


Figure 6:Interprétation sismique de Dug Software



Figure 5:Equipement de Dug Software

2. Deep Blue et le parallélisme

Dans le contexte des jeux avec ordinateur ou celui-ci est appelé à interagir rapidement, il nécessite de grandes puissances de calcul et une rapidité extraordinaire. Ces caractéristiques lors du jeu peuvent être réalisés grâce à l'exécution en parallèle des tâches dans l'optique d'avoir des résultats rapidement.

Pour le visualiser prenons le cas des **jeux d'échecs**.

Dans ce jeu, a tout instant de réactions, l'on doit évaluer un ensemble de combinaisons possibles de déplacements pour gagner en un temps très court.

Alors, à partir de chaque pion, l'on pourrait évaluer **simultanément** et **en parallèle** les combinaisons possibles pour la victoire.

Un super-ordinateur déjà réalisé pour cela est **Deep Blue** d'IBM qui, pouvant calculer plus de 200 millions de combinaisons à un moment, a pu battre en **1997** le champion en titre des jeux d'échecs **Garry Kasparov**.



Figure 7: Garry Kasparov Vs Deep Blue

3. Le parallélisme dans les marchés boursiers et le Business Intelligence (BI)

Dans les différents marchés boursiers, les bourses, et les investisseurs désirent une certaine assurance de bénéfice pour pouvoir acheter des actions. Alors, ils devront analyser des montagnes de données (les résultats et performances boursières des entreprises, informations financières...) pour débusquer le bon coup avant le reste du marché); c'est là qu'intervient les algorithmes parallèles dont les instances analyseront chacun une partie des données pour une grande rapidité.

C'est ainsi qu'est née dans les années 2010, le logiciel **Lexicon** lancé par le **Dow Jones**: un service d'information financière pour les investisseurs

professionnels. Ce service analyse et prend des décisions sur les différentes actions à acheter en quelques microsecondes, d'où le terme « **flash trading** ».

Par ailleurs, pour des entreprises désirant par exemples lancer des gammes de produits, ils doivent étudier les consommations des potentiels clients suivant différents aspects et même les pouvoirs de la concurrence. Ceci revient à analyser des métadonnées en peu de temps ; ainsi intervient des calculs parallèles pour optimiser et gagner en temps.



Figure 9: Analyses dans les marchés boursiers



Figure 8: Bourses de Wall Street

4. Le parallélisme dans la météorologie

La difficulté ici est l'analyse massive des données sur le temps. En effet, pour réaliser des prédictions météorologiques, l'on doit prendre en compte des paramètres tels que les vents, les courants marins, les températures, la position des nuages, leur évolution, etc. et ce suivant la direction, les régions et bien d'autres encore. Comme précédemment, il nécessite beaucoup de puissances de calculs qui sont fournis par des super ordinateurs comme [Fukagu](#) utilisant des algorithmes parallèles.

5. Simulations dans le domaine Aérospatial

Pour faire simuler des évènements dans l'espace ou même le décollage et l'atterrissage, les institutions comme la **SI**, **SpaceX**, et la **NASA** font recours aux algorithmes parallèles qui permettra de prendre en compte la multitude de paramètres entrant en jeux et bien-sûr assez rapidement. Chaque instance du programme de simulation pouvant gérer un aspect du problème à résoudre. C'est ainsi que certaines simulations ont permis bien avant à la **NASA** grâce aux simulations de son superordinateur parallèle **Aitiken** (basé sur le Modular Data center en abrégé **MDC** et le système de grille que nous verrons plus tard) de découvrir qu'un astéroïde devrait heurter la terre. ; ce qui a favorisé la déviation de sa trajectoire avec la **mission DART** de la **NASA** du le **24 Novembre 2021**.

En bref, Ici on retient que ces applications tournent toutes autour **de la vitesse d'analyses de données massives**.

6. Les serveurs et le parallélisme

a. Nginx

[NGINX](https://nginx.org/) est un serveur web open-source qui, depuis son succès initial en tant que serveur web, est maintenant aussi utilisé comme **reverse proxy**, **cache HTTP**, et **load balancer**. Il est largement adopté par les grandes entreprises comme Gitlab, Google, Microsoft, IBM, etc., pour gérer leur fort trafic.

Alors que de nombreux serveurs Web et serveurs d'applications utilisent une architecture simple à threads ou basée sur des processus, NGINX se distingue par une architecture sophistiquée axée sur les événements qui lui permettent de s'adapter à des centaines de milliers de connexions simultanées sur du matériel moderne. Il a un processus maître (qui effectue les opérations privilégiées telles

que la lecture de la configuration et la liaison aux ports) et un certain nombre de processus de travail et d'assistance. La configuration NGINX recommandée dans la plupart des cas est l'exécution d'un processus de travail par cœur de processeur, ce qui permet l'utilisation la plus efficace des ressources matérielles : **c'est le parallélisme**.

Lorsqu'un serveur NGINX est actif, seuls les processus de travail sont occupés. Chaque processus de travail gère plusieurs connexions de manière non bloquante, ce qui réduit le nombre de changements de contexte. Autrement dit, chaque processus de travail est à thread unique et s'exécute indépendamment, en récupérant de nouvelles connexions depuis le thread maître et en les traitant. Les processus peuvent communiquer en utilisant la mémoire partagée pour les données de cache partagées, les données de persistance de session et d'autres ressources partagées.



Figure 10: Gestion des tâches à l'aide des workers dans un serveur

b. Gunicorn

Gunicorn (Green Unicorn) est un serveur HTTP Python **WSGI** (Web Server Gateway Interface) pour UNIX. Il est conçu pour que de nombreux serveurs Web différents puissent interagir avec elle (Ex : Nginx, Apache). Il ne se soucie pas de ce que vous avez utilisé pour créer votre application Web tant que celui-ci peut être utilisé avec l'interface WSGI. Gunicorn réside généralement entre un reverse proxy (par exemple, Nginx) ou un équilibreur de charge (par exemple, **AWS ELB**) et une application Web telle qu'une application Django ou Flask.

Gunicorn est conçu pour permettre lors du déploiement de choisir le nombre de « **worker** » qui vont pouvoir traiter nos requêtes. Il est aussi à noter que chaque worker s'exécute indépendamment sur chaque CPU. Pour connaître le nombre de workers à utiliser en production, il faut utiliser la formule (**2*CPU**) +1. Ainsi, si notre application est déployée sur une machine avec **2 CPUs**, il faudra utiliser **5 workers**.

```
$gunicorn --workers=5 runserver:app
[2018-07-15 21:23:17 +0200] [2288] [INFO] Starting gunicorn 19.9.0
[2018-07-15 21:23:17 +0200] [2288] [INFO] Listening at: http://127.0.0.1:8000 (2288)
[2018-07-15 21:23:17 +0200] [2288] [INFO] Using worker: sync
[2018-07-15 21:23:17 +0200] [2306] [INFO] Booting worker with pid: 2306
[2018-07-15 21:23:17 +0200] [2307] [INFO] Booting worker with pid: 2307
[2018-07-15 21:23:17 +0200] [2308] [INFO] Booting worker with pid: 2308
[2018-07-15 21:23:17 +0200] [2309] [INFO] Booting worker with pid: 2309
[2018-07-15 21:23:17 +0200] [2310] [INFO] Booting worker with pid: 2310
```

Figure 11: Usage de 5 worker en simultané pour traiter les requêtes arrivant sur Gunicorn

7. Le parallélisme et les ordinateurs quantiques

Dans les exemples précédents, nous avons travaillé avec des machines à plusieurs processeurs. Ces dernières années, on entend beaucoup parler d'ordinateurs quantiques. Cela nous pousse donc à nous intéresser au parallélisme

au niveau de ces derniers. Pour rappel, un ordinateur quantique est l'équivalent d'un ordinateur classique, sauf que ses calculs sont effectués à l'échelle atomique et se base sur les lois de la physique quantique qui s'intéresse au comportement de la matière et de la lumière au niveau microscopique. Concernant donc ceux-ci, on parle de **parallélisme quantique**.

Le parallélisme quantique est une méthode selon laquelle un ordinateur peut effectuer des calculs de façon simultanée à l'aide d'un seul processeur quantique en exploitant les propriétés de superposition d'états quantiques des **qubits** (superposition d'états à la fois 0 et 1).

8. La blockchain et le parallélisme

Le problème de performance et de rapidité intervient dans presque, sinon tous les domaines de l'informatique. La blockchain qui gagne de plus en plus en popularité ces dix dernières années n'en fait pas exception. Par définition, la blockchain est une technologie qui permet de stocker et transmettre des informations de manière transparente, sécurisée et sans organe central de contrôle. Elle ressemble à une grande base de données qui contient l'historique de tous les échanges réalisés entre ses utilisateurs depuis sa création. La blockchain peut être utilisée de trois façons :

- Pour du transfert d'actifs (monnaie, titres, actions...).
- Pour une meilleure traçabilité d'actifs et produits.
- pour exécuter automatiquement des contrats (des "[smart contracts](#)").

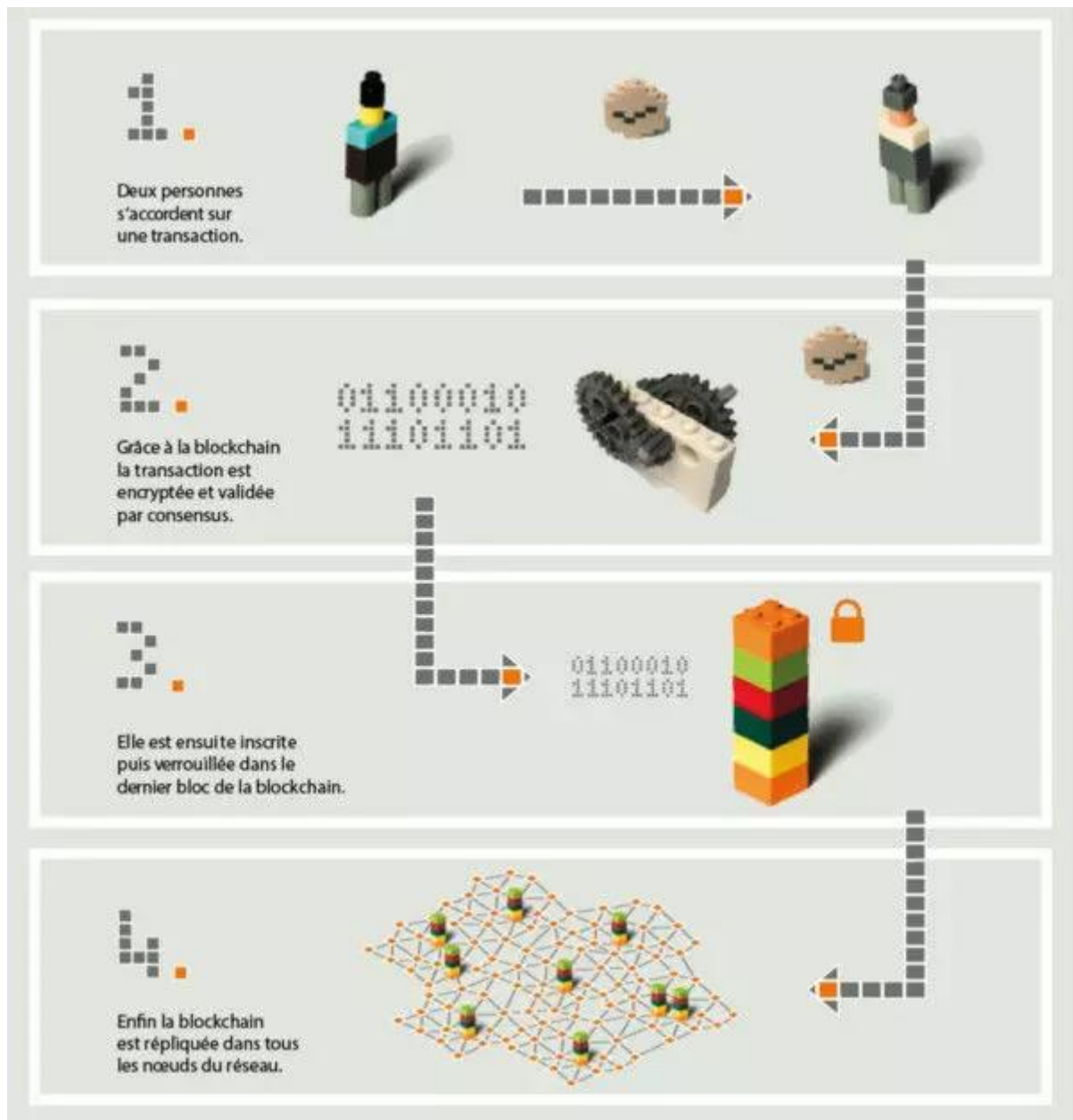


Figure 12: Fonctionnement globale de la blockchain

La blockchain est un paradigme nouveau et complexe. Ce que nous pouvons dire globalement est que dans le monde de la blockchain, la plupart des

traitements se font l'un après l'autre. C'est la principale raison pour laquelle beaucoup de recherche sont menés pour introduire le parallélisme dans le réseau blockchain. Nous avons par exemple la publication « **Transform Blockchain into Distributed Parallel Computing Architecture for Precision Medicine** » publié à la “**2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)**” ». Nous avons également le travail de **Nam-Yong Lee** sur le thème “**Hierarchical Multi-Blockchain System for Parallel Computation Cryptocurrency Transfers and Smart Contracts**” qui traitent également du parallélisme dans le monde de la blockchain.

Nous n'avons pas donné beaucoup de détails sur la blockchain car c'est un domaine trop vaste qui sort du cadre de ce cours. Nous avons essayé d'illustrer ce cas pour montrer que le parallélisme est un sujet d'actualité en informatique ayant pour principal objectif de permettre aux applications d'exploiter complètement les capacités du matériel informatique.

III. LES DIFFÉRENTS TYPES DE PARALLELISME

Comme dit à l'introduction, il existe plusieurs types de parallélisme.

1. Classification suivant le flot d'exécution d'instructions

Dans cette classe, on s'intéresse à l'organisation des flots d'instructions et leur accès données. Nous en distinguons plusieurs.

Taxonomie de Flynn

		Unique	Multiple
Flux des instructions	Unique	SISD	SIMD
	Multiple	MISD	MIMD

Table 1: Résumé de la taxonomie de Flynn

Taxonomie de kuck

		Execution Stream			
		Simple Scalaire	Simple Vectoriel	Multiple Scalaire	Multiple Vectoriel
Instruction Stream	Simple Scalaire	SESSES	SESSEA		
	Simple Vectoriel		SIASEA		
	Multiple Scalaire			MISMES	MISMEA
	Multiple Vectoriel				

Table 2:Recapitulatif de la taxonomie de Kuck

✚ Taxonomie Treleaven

	Mémoire Partagée	Mémoire Locale
Control Driven	Von Neumann	CSP
Pattern Driven	Logique	Acteurs
Demand Driven	Réduction (graphe)	Réduction (chaînes)
Data Driven	Data Flow	Data Flow (tokens)

Contrôle Explicite ↑

↓ Contrôle Implicite

Table 3: Recapitulatif de la taxonomie de Treleaven

✚ Taxonomie de Gajski

S : série P : Parallèle	Tâches	Processus	Instructions
Cray X-MP	S	P	P
NYU	P	-	S
Cedar	P	P	S

Table 4: Récapitulatif de la taxonomie de Gajski

Cependant nous nous focaliserons sur la Classification de Flynn :

CLASSIFICATION DE FLYNN

La taxonomie de Flynn, proposée par l'américain **Michael J. Flynn en 1966**, est l'un des premiers systèmes de classification de machines parallèles. Les programmes et les architectures sont classés selon le type d'organisation du flux de données et du flux d'instructions. Ainsi on a :

- ❖ **SISD** (Single Instruction, Single Data) : Ce sont les machines les plus simples traitant une donnée à la fois ; Il s'agit d'un ordinateur séquentiel qui n'exploite aucun parallélisme, tant au niveau des instructions qu'au niveau de la mémoire. Cette catégorie correspond à l'architecture de Von Neumann.

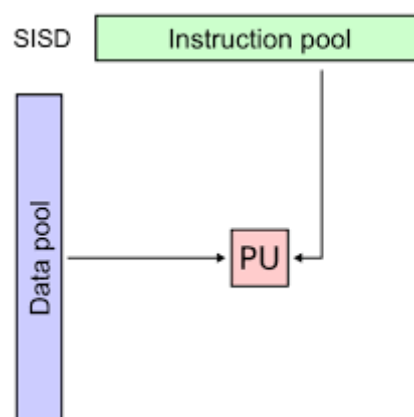


Figure 13:Architecture SISD

- ❖ **SIMD** (Single Instruction, Multiple Data) : Ce sont des systèmes traitant de grandes quantités de données d'une manière uniforme ; c'est typiquement le cas des processeurs vectoriels ou des unités de calcul gérant le traitement du signal comme la vidéo ou le son. Un **processeur vectoriel** est un processeur possédant diverses fonctionnalités architecturales lui permettant d'améliorer l'exécution de programmes utilisant massivement des tableaux,

des matrices, et qui permet de profiter du parallélisme inhérent à l'usage de ces derniers.

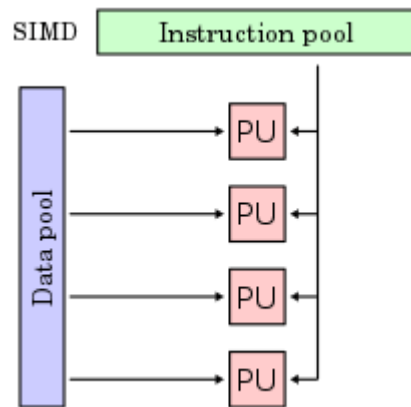


Figure 14: Architecture SIMD

- ❖ **MISD** (Multiple Instruction, Single Data) : Ce sont des systèmes traitant de grandes quantités de données d'une manière uniforme ; Autrement dit, il s'agit d'une architecture dans laquelle une même donnée est traitée par plusieurs unités de calcul en parallèle ; Il existe peu d'implémentations en pratique. Cette catégorie peut être utilisée dans le filtrage numérique et la vérification de redondance dans les systèmes critique (Exemple de **Shazam** qui utilise les fonctions mathématiques pour le filtrage du sons).

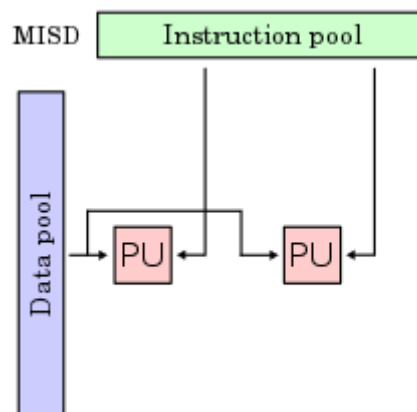


Figure 15: Architecture MISD

- ❖ **MIMD (Multiple Instruction, Multiple Data)** : Dans ce cas, plusieurs unités de calcul traitent des données différentes, car chacune d'elles possède une mémoire distincte. Il s'agit de l'architecture parallèle la plus utilisée, dont les deux principales variantes rencontrées sont les suivantes:

- **MIMD à mémoire partagée**

Les unités de calcul ont accès à la mémoire comme un espace d'adressage global. Tout changement dans une case mémoire est vu par les autres unités de calcul. La communication entre les unités de calcul est effectuée via la mémoire globale.

- **MIMD à mémoire distribuée**

Chaque unité de calcul possède sa propre mémoire et son propre système d'exploitation. Ce second cas de figure nécessite un middleware pour la synchronisation et la communication.

Les processeurs accèdent à une mémoire commune : la synchronisation peut se faire au moyen de :

- [Sémaphores](#)
- Verrous, ou [Mutex](#) (exclusion mutuelle)
- [Barrières de synchronisation](#)

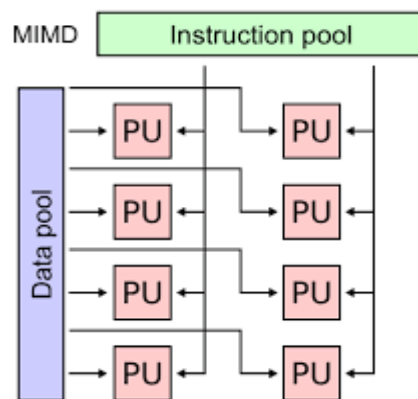




Figure 16: Architecture MIMD

Un système MIMD hybride est l'architecture la plus utilisée par les superordinateurs. Ces systèmes hybrides possèdent l'avantage d'être très extensibles, performants et à faible coût.

2. Classification suivant la topologie du réseau d'interconnexion

Ce qui distingue les machines parallèles dans cette classification est la topologie du réseau d'interconnexion utilisée pour relier les différents nœuds de calculs avec les unités de mémoires. On distingue principalement deux classes de réseaux d'interconnexion :

-  **Les réseaux à topologie statique :** Ce sont des réseaux dont la topologie est définie une fois pour toute lors de la construction de la machine parallèle. Cette topologie représente un type de graphe qui caractérise le réseau. Les réseaux statiques sont utilisés essentiellement dans les machines à passage de messages (**MPI**), les machines cellulaires, etc. Dans ces machines, on demande moins de performances au réseau d'interconnexion. Ce sont des machines à processeurs faiblement couplés. Par exemple, dans la topologie en anneau, un nœud n'est relié qu'à deux voisins, et dans l'hyper cube à quatre dimensions, chacun est relié à quatre voisins. Ainsi, si un nœud veut communiquer avec un nœud non-voisin, il doit obligatoirement passer par tous les nœuds les séparant suivant un chemin donné.
-  **Les réseaux à topologies dynamiques :** Ce sont des réseaux dont la topologie (topologie logique ou comportement du réseau) peut varier (via des « switches ») au cours de l'exécution d'un programme parallèle ou

entre deux exécutions de programmes. Les réseaux dynamiques sont souvent utilisés dans les multiprocesseurs à mémoire partagée.

Représentation de quelques topologies statiques.

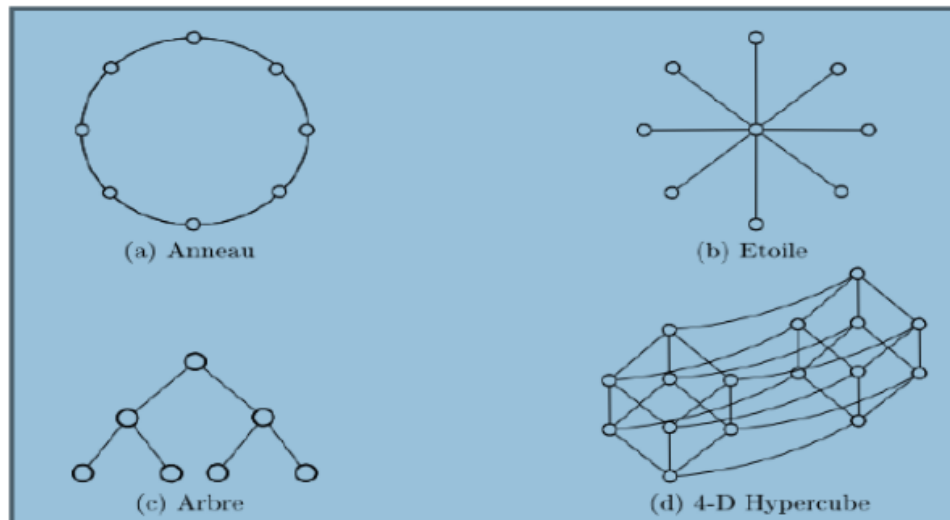


Figure 17: Réseau parallèles topologiques

3. Classification suivant la granularité du calcul

La taille du grain d'une machine parallèle peut se définir par la longueur typique d'un morceau de code exécutable sans interruption par l'unité de calcul. Une interruption peut être externe, provoquée par exemple par l'ordonnanceur du système d'exploitation, ou bien interne, provoquée par le processus en cours lui-même, suite à un besoin de communication avec d'autres processus (ou processeurs). Beaucoup de machines parallèles ne permettent que des interruptions internes.

Selon la taille du grain de calcul, on rencontre trois classes de machines parallèles :

- ✚ **Les machines à grains fins** : pour ces modèles, on suppose que le nombre de processeurs est sensiblement égal au nombre de données en entrée. Les processeurs exécutent de très petits morceaux de code entre deux communications successives. Le coût des communications est négligé au profit des calculs. Cette classe correspond aux **machines systoliques**.
- ✚ **Les machines à gros grains** : pour ces modèles, la taille de chaque mémoire locale est beaucoup plus grande que la taille d'une donnée. Et donc, chaque processeur est capable d'exécuter sans interruption des procédures voir des programmes entiers. Ces modèles reconnaissent et prennent en compte le coût des communications sans pour autant spécifier la topologie du support de communication. Les **clusters** de stations de travail sont un exemple typique de cette classe.
- ✚ **Les machines à grains moyens** : ce sont les machines qui sont entre les deux classes précédentes.

4. Classification selon la connectivité des unités de calcul

La classification peut être aussi faite selon la vitesse du réseau d'interconnexion. Dans ce cas, deux classes se distinguent :

- ❖ **Les machines à processeurs fortement couplés** : il s'agit des machines dont les processeurs sont étroitement connectés par un réseau rapide. Ce sont par exemple les machines massivement parallèles telle que la **T3E de SCI/CRAY**.

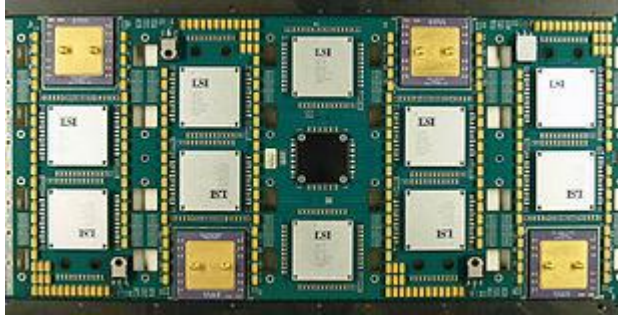


Figure 18: Carte processeur de T3E-600

- ❖ **Les machines à processeurs faiblement couplés** : Ce sont des machines dont les processeurs sont reliés par un réseau plutôt lent. C'est le cas des **clusters de stations** de travail qui utilisent des réseaux locaux **standard 2**.

Les machines parallèles sont aussi classifiées selon leurs performances, selon leur nombre de processeurs, selon leur taille mémoire et selon leur extensibilité (possibilité d'augmentation du nombre de processeurs sans perte significative de performance). La connaissance des ordinateurs parallèles ne suffit pas pour obtenir un bon algorithme parallèle, il faut en plus une bonne gestion de tout le matériel informatique utilisé.

5. Classification suivant les mémoires des machines parallèles

Cette classe se base sur le type de la mémoire. On trouve généralement deux classes de machines parallèles : **les machines à mémoire partagée** et **les machines à mémoire distribuée**.

- **Mémoire partagée**

Tous les processeurs ont la capacité à adresser l'espace mémoire en tant qu'espace d'adresses globale. La communication entre les tâches est réalisée par les opérations de lecture et écriture sur la mémoire partagée.

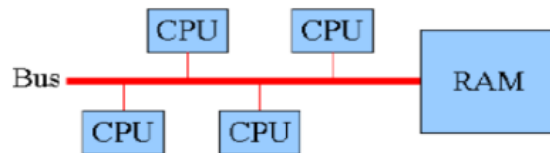


Figure 19: Schéma explicatif pour la classification à mémoire partagée

- **Mémoire distribuée**

Chaque processeur a sa propre mémoire locale. Chaque processeur a le droit et la capacité d'adresser uniquement sa mémoire locale, la communication entre les processus s'exécutant sur différents processeurs est effectuée par des messages passés par la communication réseau.

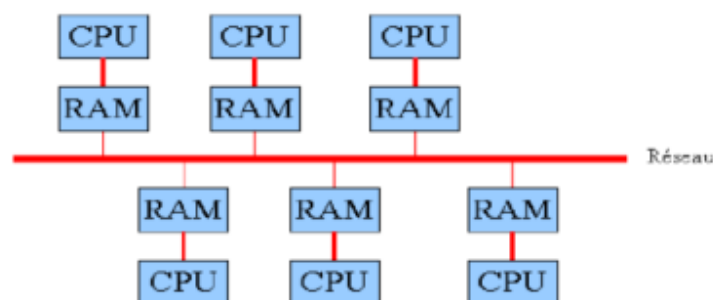


Figure 20: Schéma explicatif pour la classification à mémoire distribuée

IV. Modèles de calcul parallèle

Un **modèle de calcul parallèle** est en fait la description d'une machine permettant d'implémenter les algorithmes parallèles suivant une architecture bien précise. Ci-dessous nous allons présenter quelques modèles de machines parallèles.

1) Le modèle PRAM

Le modèle **PRAM** (Parallel Random Access Machine) est le modèle de calcul parallèle le plus couramment utilisé. Il a été développé avec comme objectif de définir un modèle de calcul qui jouerait, pour les architectures et algorithmes parallèles, le même rôle unificateur et, surtout, simplificateur que celui joué par le modèle RAM pour les algorithmes et machines séquentielles.

Une machine PRAM modélise le fonctionnement d'une architecture **MIMD** à mémoire partagée.

Plus précisément, l'objectif du modèle PRAM est de définir un modèle de calcul pour lequel on peut concevoir des algorithmes théoriques et abstraits ainsi que des algorithmes dont les analyses asymptotiques permettent de « prédire » le comportement sur les machines parallèles et de classer les divers algorithmes parallèles tout en effectuant des analyses de complexité. Le modèle PRAM repose donc sur un ensemble de simplifications semblables à celle du modèle RAM. Dans le cas du modèle PRAM, ces simplifications vont nous conduire à extraire et à déterminer le maximum de parallélisme possible contenu dans un algorithme. Les composantes d'une machine PRAM sont donc les suivantes :

- Une collection de p processeurs P_1, \dots, P_p , chaque processeur étant un processeur RAM standard.

- **Une unité de mémoire globale**, partagée entre les différents processeurs, qui permet à chaque processeur d'accéder à une case mémoire en temps constant ($\Theta(1)$). Il est important de souligner que les **p processeurs** ne sont pas reliés directement entre eux, mais communiquent strictement par l'intermédiaire de la mémoire.

L'exécution d'un algorithme passe alors par trois phases (comme dans le modèle RAM) exécutées conjointement (c'est-à-dire de façon synchrone, sous le contrôle de l'horloge) par les divers processeurs.

- ✚ **Lecture** d'une ou plusieurs cases mémoires et transfert dans les registres locaux aux processeurs.
- ✚ **Calcul** sur les registres locaux, chacun des calculs se fait en parallèle mais tous les processeurs exécutent la même instruction de façon synchrones.
- ✚ **Écritures** de résultats (registres) dans la mémoire.

La synchronisation dans l'exécution des instructions est une caractéristique importante et cruciale du modèle PRAM. Plus précisément, dans le modèle PRAM, les diverses phases d'un cycle d'exécution doivent être effectuées de façon synchrone par l'ensemble des processeurs. Ainsi tous les accès en lecture sont tout d'abord faits de façon concurrente et synchrone. Ensuite tous les processeurs exécutent de façons concurrentes et synchrones les calculs sur les valeurs et registres locaux. Finalement, tous les processeurs écrivent leurs résultats en mémoires.

Bien que pour certains algorithmes ce synchronisme ne soit pas tout à fait nécessaire, d'autres algorithmes ont absolument besoin de cette propriété d'exécution concurrente et synchrone pour assurer leur bon fonctionnement. Une autre caractéristique du modèle est que tous les calculs se font strictement dans les registres locaux des processeurs. Les accès à la mémoire ne se font que pour

charger des valeurs de la mémoire vers les registres, ou bien transférer le contenu des registres vers la mémoire. En d'autres mots, une machine PRAM est donc une machine de style **load-store architecture**, tout comme le sont les machines RISC modernes.

Il existe différents modèles PRAM qui se distinguent par la façon dont les conflits d'accès à des adresses mémoires sont traités. Les plus couramment utilisés sont les suivants :

- ❖ **EREW (Exclusif Read Exclusif Write)** : modèle, le plus restrictif, permettant strictement la lecture et l'écriture en mode exclusif.
- ❖ **CREW (Concurrent Read Exclusif Write)** : modèle, intermédiaire, permettant la lecture concurrente mais ne supportant que l'écriture exclusive.
- ❖ **CRCW (Concurrent Read Concurrent Write)** : modèle, le moins restrictif, permettant la lecture et l'écriture concurrente.

La structure abstraite d'une machine parallèle PRAM est donc telle que décrit à là-ci-dessous.

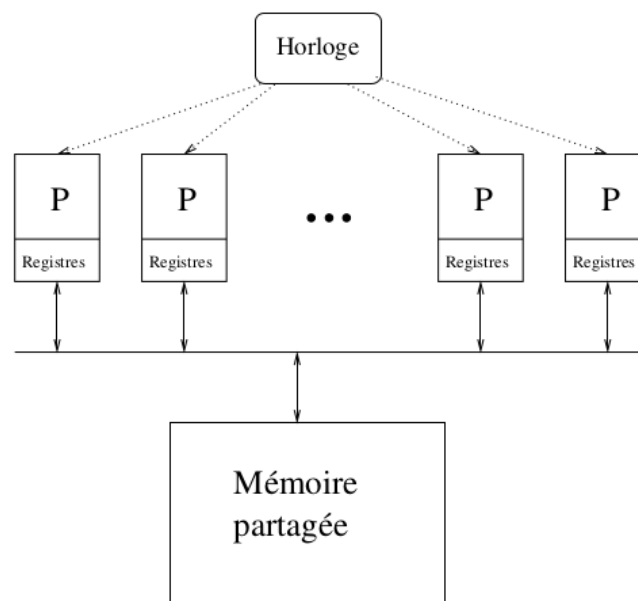


Figure 21 : Structure abstraite du modèle PRAM

2) Le modèle systolique

On peut définir un **réseau systolique** comme un réseau de processeurs qui calculent et échangent des données régulièrement. L'analogie est souvent faite avec la régularité de la contraction cardiaque qui propage le sang dans le système circulatoire du corps. Chaque processeur d'un réseau systolique peut être vu comme un cœur jouant le rôle de pompe sur plusieurs flots le traversant. Le rythme régulier de ces processeurs maintient un flot de données constant à travers tout le réseau.

Une donnée introduite une seule fois dans le réseau est propagée d'un processeur à un processeur voisin et peut ainsi être utilisée un grand nombre de fois. Cette propriété autorise de gros débits de calculs même si la cadence des entrées-sorties reste faible. En d'autres termes, on évite les engorgements des buffers d'entrées-sorties. Ceci rend le modèle systolique adéquat pour beaucoup de problèmes dont le nombre de calculs sur une même donnée est largement supérieur à son nombre d'entrées-sorties. Les caractéristiques dominantes d'un réseau systolique peuvent être définies par un parallélisme massif et décentralisé, par des communications locales et régulières, et par un mode opératoire synchrone. Pour décrire un réseau systolique, il est donc nécessaire de spécifier :

- La topologie du réseau d'interconnexion des processeurs ;
- L'architecture d'un processeur (description des registres et canaux : nom, type, sémantique, etc.) ;
- Le programme d'un processeur ;
- Le flot de données consommées par le réseau pour produire une solution.

Ce modèle, introduit en **1978** par **Kung** et **Leiserson**, s'est révélé être un outil puissant pour la conception de processeurs intégrés spécialisés. En un mot, une

architecture systolique est agencée en forme de réseau. Ces réseaux se composent d'un grand nombre de cellules élémentaires identiques et localement interconnectées. Chaque cellule reçoit des données en provenance des cellules voisines, effectue un calcul simple, puis transmet les résultats, toujours aux cellules voisines, un temps de cycle plus tard. Pour fixer un ordre de grandeur, disons que chaque cellule a la complexité au plus d'un petit microprocesseur. Les cellules évoluent en parallèle, en principe sous le contrôle d'une horloge globale (synchronisme total) : plusieurs calculs sont effectués simultanément sur le réseau, et on peut "pipeliner" la résolution de plusieurs instances du même problème sur le réseau.

Le "**pipelining**" est une technique de parallélisation des tâches dont on peut donner un exemple dans le domaine de la construction automobile avec les chaînes de fabrication : chaque tâche est réalisée sur un poste différent, et ainsi un grand nombre d'automobiles sont simultanément en cours de fabrication sur la chaîne. La dénomination "systolique" provient d'une analogie entre la circulation des flux de données dans le réseau et celle du sang humain, l'horloge qui assure la synchronisation constituant le "cœur" du système.

3) Le modèle BSP

Le modèle BSP (**Bulk Synchronous Parallel**) est un modèle de pontage pour la conception d'algorithmes parallèles. Il est similaire au modèle de machine parallèles à accès aléatoire parallèle (PRAM), mais contrairement à la PRAM, BSP ne considère pas la communication et la synchronisation comme acquises. En fait, la quantification de la synchronisation et de la communication requises est une partie importante de l'analyse d'un algorithme.

Ce modèle formalise les caractéristiques architecturales des machines existantes à l'aide de quatre paramètres. Un algorithme écrit dans le modèle BSP

est constitué d'une séquence de super-étapes. Lors d'une super-étape, un processeur peut faire des calculs locaux et un certain nombre de communications (composées d'envois et de réceptions). Entre deux super-étapes consécutives, une barrière de synchronisation, permet comme son nom l'indique, de synchroniser l'exécution sur les processeurs. Le modèle BSP est spécifié à l'aide des paramètres suivants :

- ✚ L: période de synchronisation qui correspond à l'unité de temps nécessaires pour synchroniser tous les processeurs.
- ✚ G : coût pour envoyer un mot à travers le réseau.
- ✚ H : nombre maximal de messages que peut envoyer ou recevoir chaque processeur.
- ✚ S : surcoût fixe dû à la mise en place d'une communication, aussi petit que soit le message à envoyer.

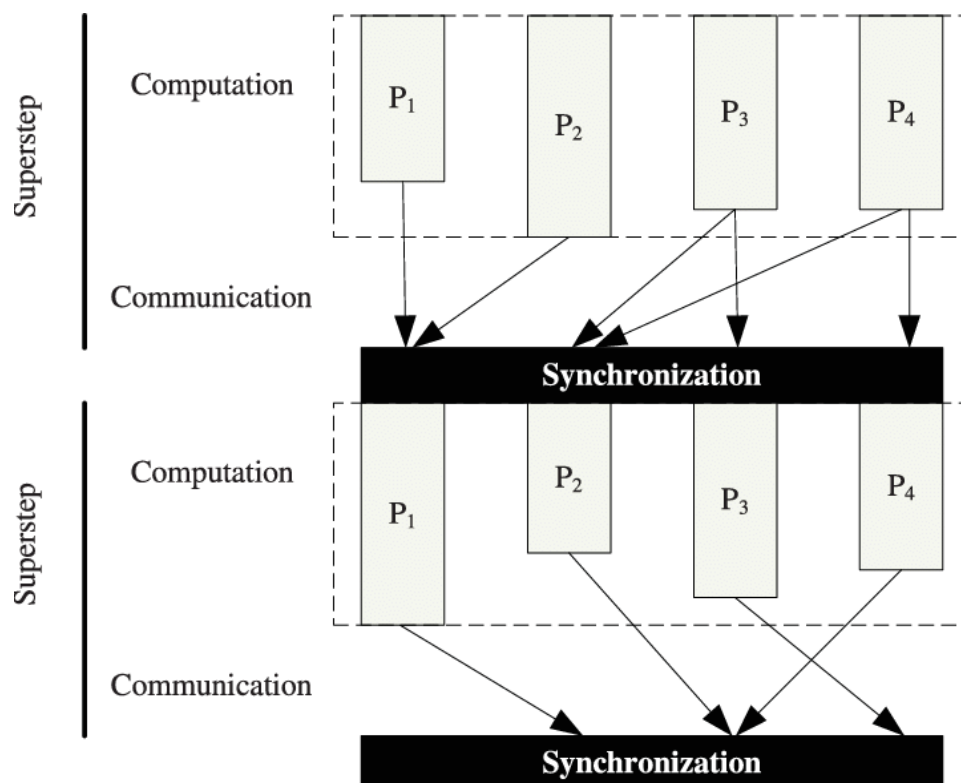


Figure 22: Synchronisation des barrières

4) Le modèle grille à deux dimensions et l'hypercube

Parmi les modèles de machines à mémoires distribués, la grille à deux dimensions et l'hypercube ont été beaucoup utilisés. Dans ces modèles chaque processeur a sa propre mémoire locale de taille constante, il n'existe pas de mémoire partagée. Les processeurs peuvent communiquer uniquement grâce à un réseau d'interconnexion. Comme dans le cas de PRAM et du modèle systolique, les processeurs travaillent de manière synchrone. À chaque étape, chaque processeur peut envoyer un mot de données à un de ses voisins, recevoir un mot de données d'un de ses voisins, et effectuer un traitement local sur ces données. La complexité d'un algorithme est définie comme étant le nombre d'étapes de son exécution.

Ces modèles prennent explicitement en compte la topologie du réseau d'interconnexion. Ce dernier présente différentes caractéristiques importantes comme :

- ❖ **Le degré**, qui est le nombre maximal de voisins d'un processeur, il correspond à une sorte de limitation architecturale donnée par le nombre maximum de liens physiques associés à chaque processeur ;
- ❖ **Le diamètre**, qui est la distance maximale entre deux processeurs (cette distance est donnée par le plus court chemin dans le réseau d'interconnexion entre les deux processeurs les plus éloignés). Il donne une borne inférieure sur la complexité des algorithmes dans lesquels deux processeurs arbitraires doivent communiquer ;
- ❖ **La largeur**, qui est le nombre minimum de liens à enlever afin de diviser le réseau en deux réseaux de même taille (plus ou moins un). Elle présente une borne inférieure sur le temps d'exécution des algorithmes où il existe une phase qui fait communiquer une moitié des processeurs avec une autre moitié.

La grille à deux dimensions

La grille à deux dimensions de taille p est composée de p processeurs $P_{i,j}$, $1 \leq i, j \leq \sqrt{p}$, tels que le processeur $P_{i,j}$ est relié aux processeurs $P_{i-1,j}$, $P_{i+1,j}$, $P_{i,j-1}$ et $P_{i,j+1}$ pour $2 \leq i, j \leq \sqrt{p} - 1$ dans le cas du carré mais on n'omet pas le cas où on pourrait avoir une grille rectangulaire. La grille a un degré de 4, un diamètre de $O(\sqrt{p})$ et une largeur de bande de \sqrt{p} . Cette structure ainsi que ses dérivés, comme la grille à trois dimensions, le **tore** à 2 ou 3 dimensions (les processeurs à la périphérie sont connectés avec ceux se trouvant à l'opposé) ont l'avantage d'être simple et flexible.

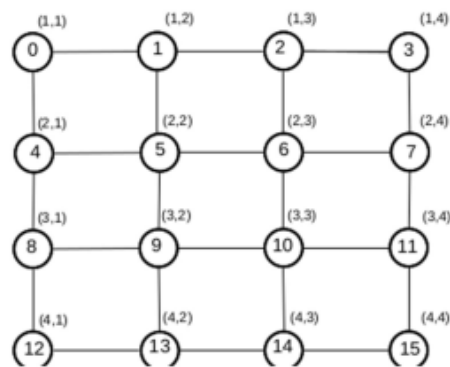


Figure 23: Grille à deux dimensions de taille 16

L'hypercube

L'hypercube de dimension d est composé de $p = 2^d$ processeurs, numérotés de 0 à $p-1$. Deux processeurs P_i et P_j sont reliés si et seulement si la représentation binaire de i et celle de j diffèrent seulement d'un bit. Si ce bit est égal à k , on dit que P_i et P_j sont voisins suivant la dimension k et que $j = i \oplus 2^k$. Le degré et le diamètre sont égaux à $\log_2(p)$, et la largeur de bande est égale à $p/2$.

L'hypercube est attrayant de par sa régularité et son petit diamètre, mais est difficilement réalisable en pratique (à cause du degré qui augmente avec le nombre de processeurs). Le schéma ci-dessus présente des hypercubes de dimensions 2, 4, 8, 16 et 32.

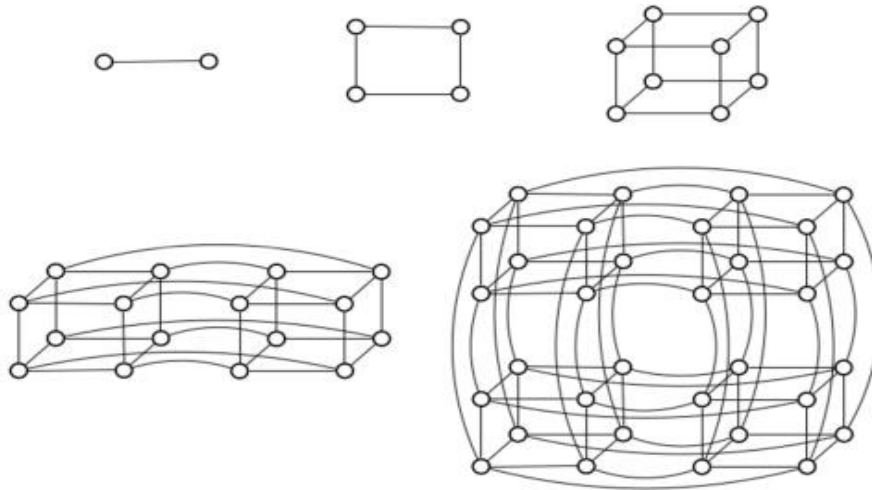


Figure 24: hypercubes de dimensions 2, 4, 8, 16 et 32

Pour écrire un algorithme dans un de ces modèles à mémoire distribuée, l'on peut soit de simuler un algorithme PRAM soit d'écrire un algorithme spécifique pour l'architecture du modèle choisi. La première solution bien qu'élégante conduit souvent à des algorithmes non efficaces. En revanche, la deuxième solution permet d'obtenir des résultats efficaces mais reste très souvent compliquée. De plus, la plupart du temps, elle ne permet pas d'écrire des algorithmes portables, puisque chaque modèle à mémoire distribuée dépend fortement de la topologie du réseau choisi. Par conséquent, lorsqu'on change la topologie du réseau d'interconnexion, il faut souvent changer d'algorithme.

5) Modèle CGM

Le modèle **CGM (Coarse Grained Multicomputers)** a été introduit en **1993**. Par rapport au modèle BSP, il s'affranchit des paramètres L , g , et s , ainsi que de

l'étape de synchronisation. En effet, ce modèle n'utilise que deux paramètres : P qui est le nombre de processeurs utilisés et N qui représente le nombre de données d'entrée du problème. Dans le modèle CGM, P doit être nettement inférieur à N ($P \ll N$). Ce modèle représente beaucoup mieux les architectures existantes composées de plusieurs milliers de processeurs et qui peuvent traiter des millions ou des milliards de données. De plus, le modèle donne explicitement le nombre de données par processeur. Par la suite, nous considérerons le nombre de données fournies aux processeurs dans nos problèmes, de taille \sqrt{P} .

Les algorithmes écrits dans ce modèle sont composés d'une succession de deux phases :

- **La première phase** où chaque processeur effectue un calcul sur ses données locales.
- **Et la Deuxième** où les processeurs échangent des données afin de les redistribuer. Pendant cette seconde phase, chaque processeur peut envoyer les données et en recevoir en $O\left(\frac{N}{P}\right)$. Ainsi, dans un modèle CGM à P processeurs, chacun des processeurs a une mémoire locale de taille

$$\text{en } O\left(\frac{N}{P}\right).$$

La communication est globale entre les processeurs et n'importe quel réseau d'interconnexion peut être utilisé. Le modèle cherche à obtenir un nombre de rondes de communications qui soit le plus petit possible (**$\log P$**). En aucun cas ce nombre ne doit dépendre de la taille du problème car les performances s'en trouveraient dégradées.

En application du modèle PRAM, on considère que le nombre de super-étapes devrait être poly logarithmique en P , mais cela semble être loin de la réalité. En fait, les algorithmes qui assurent simplement un nombre de super-étapes en fonction de P (et non de N) fonctionnent bien dans la pratique. En effet, comme le nombre de super-étapes dépend du nombre de rondes de communication et que

chaque processeur doit, dans le pire des cas, travailler sur l'ensemble des données, ce nombre de super-étapes doit être au plus poly logarithmique en P ou fonction de P.

Après avoir étudié, les différents modèles, une combinaison des modèles BSP et GCM (mis sur pied pour diminuer l'écart entre le matériel et le logiciel) nous permet de résoudre de manière efficace des problèmes de programmation dynamique ou l'on a une forte dépendance entre les différents calculs tels que le problème d'ordonnement de produit de chaîne de matrices, le problème de triangulation optimale d'un polygone convexe, le problème de recherche de l'arbre binaire de recherche optimal, etc.[Ken14]

V. CALCUL DE LA COMPLEXITÉ DES ALGORITHMES PARALLÈLES

Pour étudier les performances d'un algorithme parallèle, plusieurs mesures sont utilisées. La plus naturelle est le temps d'exécution de l'algorithme. Il correspond au **temps écoulé entre le moment où le premier processeur commence l'exécution de l'algorithme et le moment où le dernier processeur finit cette exécution**. Dans la suite de cette section, le temps d'exécution de l'algorithme parallèle sur p processeurs sera noté T_p . Ce temps peut être aussi défini comme étant la somme du **temps de calculs, du temps de communication et du temps d'attente d'un processeur arbitraire j** ($T_p = T_j \text{ calculs} + T_j \text{ com} + T_j \text{ attentes}$), ou comme étant **la somme de ces trois temps sur tous les processeurs divisés par le nombre de processeurs**

$$T_p = \frac{\sum_{i=1}^p T_i \text{ calculs} + \sum_{i=1}^p T_i \text{ coms} + \sum_{i=1}^p T_i \text{ attente}}{p}.$$

Une manière simple d'évaluer les communications est de donner le nombre de messages transmise et leur taille en octets. Ainsi, envoyer un message ayant **k Octets** prendra un temps $T_{\text{mesg}} = T_{j \text{ init}} + k \times T_{j \text{ trans}}$, où $T_{j \text{ init}}$ est le temps d'initialisation de la communication, et $T_{j \text{ trans}}$ est le temps pour transférer un octet entre deux processeurs. La réalité est souvent plus complexe car il faut prendre en compte la nature du réseau.

La latence indique le temps nécessaire pour communiquer un message unitaire entre deux processeurs. Tout envoi d'un message prendra donc un temps supérieur à celui de la latence.

Le facteur d'accélération (« **speedup** ») correspond au rapport entre le temps d'exécution de l'algorithme séquentiel optimal T_s et le temps d'exécution de l'algorithme parallèle sur p processeurs : $A_p = \frac{T_s}{T_p}$.

Aussi, p est le meilleur facteur d'accélération que l'on puisse obtenir.

L'efficacité est le facteur d'accélération divisé par le nombre de processeurs :

$$E_p = \frac{A_p}{p}$$

Il représente la fraction de temps où les processeurs travaillent. Il caractérise aussi l'efficacité avec laquelle un algorithme utilise les ressources de calcul d'une machine.

On s'intéresse aussi à l'extensibilité (« **scalability** ») lorsqu'on regarde l'évolution du temps d'exécution et du facteur d'accélération en fonction du nombre de processeurs.

VI. Architecture parallèle à mémoire partagé avec OpenMp

Pour rappel, dans les architectures parallèles à mémoire partagée, il existe un espace mémoire globale pour tous les processeurs et chaque processeur a sa propre mémoire locale (Cache) dans laquelle est copiée une partie de la mémoire globale. Comme avantage de ce type d'architecture, on peut citer le fait que l'espace d'adressage globale est facile à utiliser par les processeurs et dont l'échange données est rapide. Le principal inconvénient ici est la gestion difficile des accès conflictuels en mode écriture à la mémoire. Une telle architecture est représentée par :

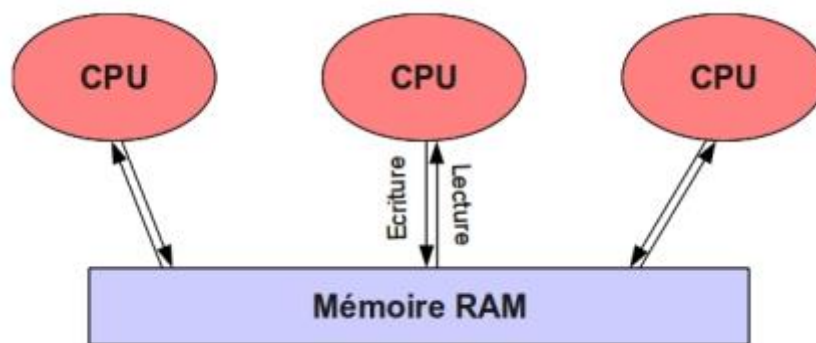


Figure 25: Architecture parallèle à mémoire centralisée

1. OpenMP, c'est quoi ?

OpenMP (**Open Multi-Processing**) est une interface de programmation à mémoire partagée basé sur les directives à insérer dans le code source (**C**, **C++**, **Fortran**). En d'autres termes, c'est une API permettant la parallélisation en mémoire partagée multi-threadée et basée sur une combinaison de directives de parallélisation, de routines et de variables d'environnement. C'est un modèle de

programmation explicite où le programmeur a le contrôle sur la parallélisation. OpenMP utilise un modèle d'exécution « **fork-join** » et tout programme open MP peut être compilé par un compilateur non openMP ou sans prendre en compte les directives openMP. OpenMp est composé de trois types de composants : **les directives de parallélisation, les routines et les variables d'environnement.**

a. Les directives de parallélisation

Les directives de parallélisations que dispose openMP sont :

- Les régions paretles (**PARALLEL**)
- Les boucles parallélisées (**PARALLEL DO**)
- Sections parallèles (**PARALLEL SECTION**)
- Section exécutée par un seul processeur (**SINGLE**)
- Synchronisation (**BARRIER, CRITICAL, ATOMIC, ...**)
- Les propriétés des données (**PRIVATE, SHARED, REDUCTION, ...**)

b. Les routines

On distingue plusieurs parmi lesquelles :

- **OMP_SET_NUM_THREADS**
- **OMP_GET_NUM_THREADS**

c. Les variables d'environnements

Il en existe une pléthore mais nous allons juste en énumérer quelques-unes :

- [omp_get_active_level](#)
- [omp_get_ancestor_thread_num](#)
- [omp_get_dynamic](#)
- [Omp_get_level](#)

2. Le modèle « fork-join » de openMP

Dans le modèle « **fork-join** » de openMP, le **thread maître** s'exécute séquentiellement jusqu'à la première région parallèle. À ce niveau, le thread maître crée un ensemble de threads qui s'exécutent en parallèle : c'est l'opération **FORK**. Lorsque tous les threads ont terminé leur travail, ils se synchronisent et le thread maître continue séquentiellement : c'est l'opération **JOIN**. Il est à noter que le nombre de threads est indépendant du nombre de processeurs.

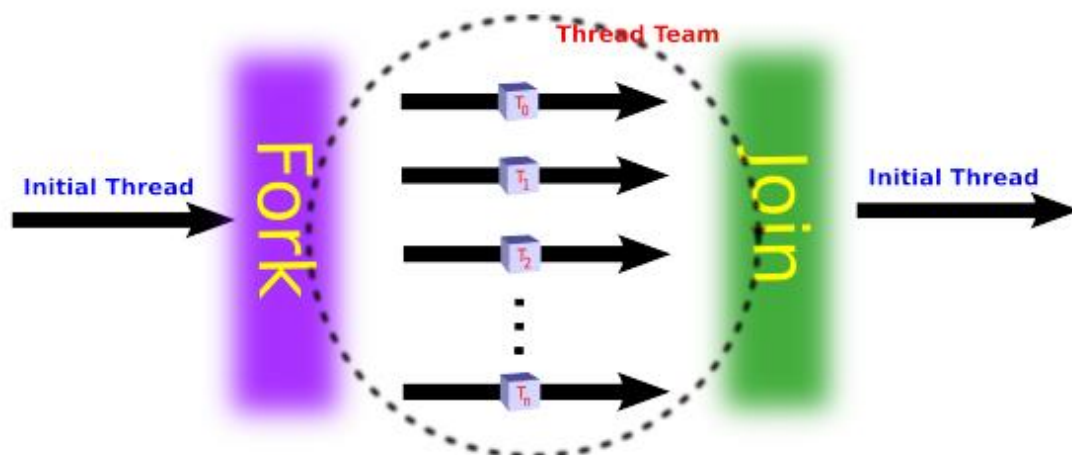


Figure 26 : Modèle "Fork-Join" de openMP

3. Construction d'openMP définissant le partage du travail

Les constructions openMp pour le partage du travail sont :

- **DO/for loops** : Il est utilisé pour faire la parallélisation des données
- **SECTION** : Cette construction découpe le travail global en section indépendantes qui sont exécutées en parallèle.
- **SINGLE** : Celui-ci est utilisé pour sérialiser une partie du code. Il est utilisé pour des sections qui ne sont pas **threadsafe**. (EX : E/S). Un morceau de code est **thread-safe** s'il fonctionne correctement pendant l'exécution simultanée par plusieurs threads.

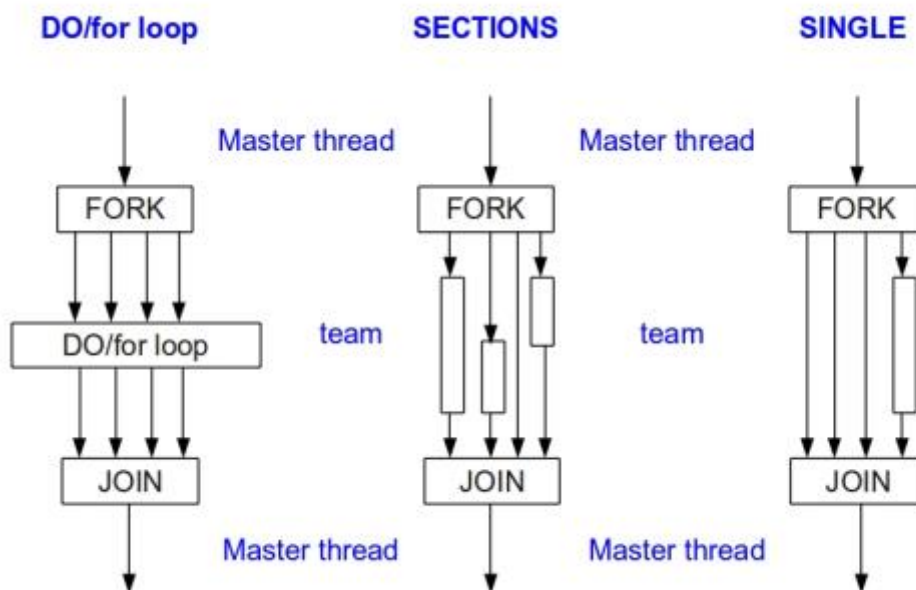


Figure 27: Construction d'openMP définissant le partage du travail

VII. Architecture parallèle à mémoire distribuée avec MPI

Dans ce type de modèle, une espace mémoire est associé à chaque processus/processeur et l'accès à la mémoire du processus/processeur voisin se fait par échange de messages entres les processus. Aussi, chaque processus travaille sur des variables privées qui résident dans sa propre mémoire locale. Une

des choses intéressants dans ce type d'architecture est que **les algorithmes sont conçus pour minimiser les communications entre les processus** (distribution des données, minimisation des envois,). Visuellement, cette architecture se présente comme suit :

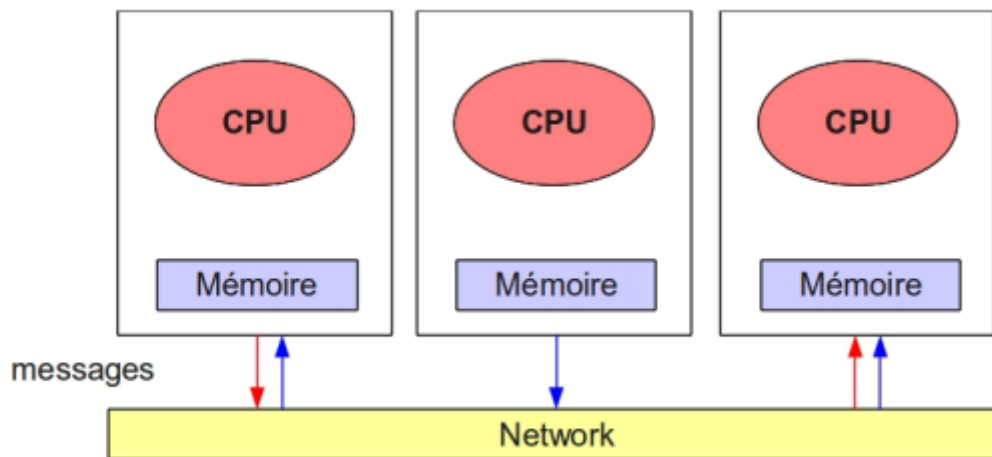


Figure 28: Représentation visuel des architectures à mémoire distribuées

1. MPI c'est quoi ?

MPI (Message Passing Interface) est une API de haut niveau pour la programmation parallèle obéissant au paradigme de l'échange de message. Il est utilisable en **C, C++ et fortran**. MPI est adaptée pour la programmation parallèle distribuée et est très répandu dans le monde du calcul intensif.

La bibliothèque MPI permet de gérer :

- L'environnement d'exécution
- Les communications point à point
- Les communications collectives
- Les groupes de processus

Il permet aussi de gérer :

- La communication unidirectionnelle
- La création dynamique de processus
- Le multithreading
- Les entrées/sorties parallèles

2. Pourquoi utiliser MPI ?

MPI étant adaptée pour la programmation parallèle distribuée, quelques raisons qui poussent à l'utiliser sont :

- MPI est avant tout une interface
- Il est présent sur tout type d'architecture parallèle
- Les constructeurs de machines et/ou de réseaux rapides fournissent des bibliothèques MPI optimisées pour leurs plateformes.

VIII. Architecture parallèle hybride avec openMP et MPI

La programmation hybride parallèle consiste à mélanger plusieurs paradigmes de programmation parallèle dans le but de tirer parti des avantages des différentes approches. De nos jours, les architectures de machines parallèles qu'on rencontre sont souvent à plusieurs niveaux de parallélisme. Généralement, **MPI est utilisé au niveau des processus et openMP est utilisé à l'intérieur de chaque processus**. À la place de openMP, on peut utiliser **OpenACC, pthreads, Cuda, etc.**

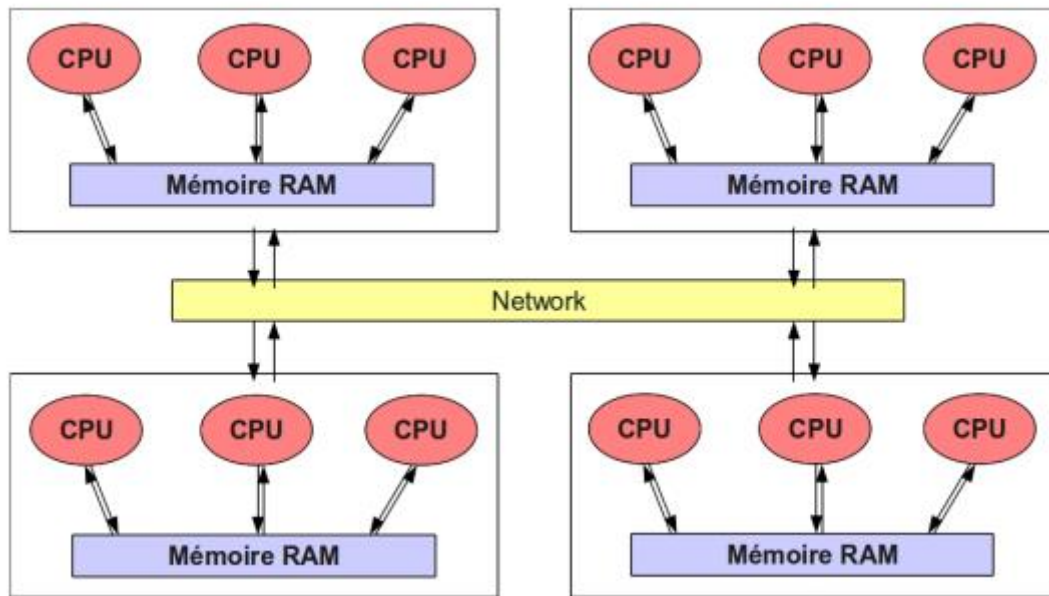


Figure 29: Modèle hybride

La programmation hybride apporte son lot d'avantages et d'inconvénients.

a. Avantages de la programmation hybride

Quelques avantages visibles de la programmation hybride sont :

- Optimisation de la consommation de la mémoire totale (grâce à l'approche mémoire partagée OpenMP, gain au niveau des données répliquées dans les processus MPI et de la mémoire utilisée par la bibliothèque MPI elle-même).
- Amélioration des performances de certains algorithmes en réduisant le nombre de processus MPI (**moins de domaines=meilleur pré conditionneur si on laisse tomber les contributions des autres domaines**)
- Moins d'accès simultanés en entrées-sorties et taille moyenne des accès plus grands. Cela entraîne moins de charge sur les serveurs de métadonnées avec des requêtes de tailles plus adaptées. Les gains potentiels sur une application massivement parallèle peuvent être importants

- **Moins de fichiers à gérer** si on utilise une approche où le nombre de fichiers est proportionnel au nombre de processus MPI (approche fortement déconseillée dans un cadre de parallélisme massif).
- Un code MPI est une succession de phase de calcul et de communication. Plus la granularité d'un code est importante, plus il est extensible. Comparée à l'approche pure MPI, l'approche hybride augmente significativement la granularité et donc l'extensibilité des codes.

b. Inconvénients de la programmation hybride

En programmation hybride, on peut citer quelques inconvénients :

- ❖ Complexité et niveau d'expertise accrus
- ❖ Nécessité d'avoir de bonnes performances MPI et OpenMP
- ❖ Gains en performances non garantis (surcoûts supplémentaires, ...)
- ❖ Le code produit peut devenir plus difficile à déboguer, à analyser et à maintenir

IX. Les avantages et limites du parallélisme et comparaison avec le paradigme séquentiel

1. Avantages

- ❖ L'informatique parallèle permet de gagner du temps, permettant l'exécution d'applications en un temps record.
- ❖ Le parallélisme permet de résoudre des problèmes plus importants dans un court laps de temps.
- ❖ Par rapport à la séquentialité, l'informatique parallèle est bien mieux adaptée pour modéliser, simuler et comprendre des phénomènes complexes du monde réel.
- ❖ Allouer plus de ressources à une tâche raccourcira son temps d'exécution, avec des économies de coûts potentielles. Les ordinateurs parallèles peuvent être construits à partir de composants de base bon marché.
- ❖ De nombreux problèmes sont si importants et/ou complexes qu'il est peu pratique ou impossible de les résoudre sur un seul ordinateur, en particulier compte tenu de la mémoire limitée de l'ordinateur.
- ❖ Vous pouvez faire beaucoup de choses simultanément en utilisant plusieurs ressources informatiques.
- ❖ Le parallélisme permet d'utiliser des ressources informatiques sur le réseau étendu (WAN) ou même sur Internet.
- ❖ Il dispose d'un stockage de données massif et de calculs de données rapides.

2. Limites

- ❖ La programmation pour cibler l'architecture parallèle est un peu difficile, mais avec une bonne compréhension et de la pratique, vous êtes prêt à partir.

- ❖ L'utilisation du calcul parallèle vous permet de résoudre des problèmes de calcul et de données à l'aide de processeurs multicœurs, mais, parfois, cet effet sur certains de nos algorithmes de contrôle et ne donne pas de bons résultats et cela peut également affecter la convergence du système en raison de l'option parallèle.
- ❖ Les surcoûts (c'est-à-dire l'augmentation du temps d'exécution) induits sont dus aux transferts de données, à la synchronisation, à la communication, à la création/destruction de threads, etc. Ces coûts peuvent parfois être assez importants, et peuvent même dépasser les gains dus à la parallélisation.
- ❖ Divers ajustements de code doivent être effectués pour différentes architectures cibles afin d'améliorer les performances.
- ❖ De meilleures technologies de refroidissement sont nécessaires en cas de clusters.
- ❖ La consommation d'énergie est énorme par les architectures multicœurs.
- ❖ Les solutions parallèles sont plus difficiles à mettre en œuvre, elles sont plus difficiles à déboguer ou à prouver qu'elles sont correctes, et elles fonctionnent souvent moins bien que leurs homologues en série en raison de la surcharge de communication et de coordination.

3. Comparaison du paradigme parallèle et du paradigme séquentielle

CRITERES	ALGORITHMES PARALLELS	ALGORITHMES SEQUENTIELS
Type de problèmes	Ils sont adaptés aux problèmes dont les sous	Ils sont adaptés aux problèmes dont les sous

	problèmes sont indépendants	problèmes sont itératifs ou doivent être réalisés l'un à la suite de l'autre
Rapidité d'exécution	Ils sont plus rapides pour des problèmes bien conçues pour ce paradigme	Ils sont moins rapides en général
Utilisation des ressources	Toutes les ressources peuvent être utilisées et il y a optimisation de l'utilisation du matériel	Toutes les ressources ne sont pas forcément utilisées
Exécution des tâches	Exécution en parallèles sur des processeurs ou cœurs différents	Exécution séquentielle sur un cœur ou processeur

Table 5: Comparaison entre les paradigmes séquentiels et parallèles :

CONCLUSION

Rendu au terme de ce cours, où il était question d'implanter les bases du paradigme parallèle en présentant son principe et ses applications, ses différentes variantes suivant le flot d'instruction, de la mémoire, du types de calcul ,la communication entre les processus dans ce paradigme grâce à MPI, une plateforme permettant d'écrire des programmes parallèles(nous parlons de **OpenMp**),l'évaluation de la complexité d'un algorithme parallèle ainsi que ses avantages et limites et enfin une comparaison entre le paradigme séquentiel et le paradigme parallèle .Il en ressort que le but principal du parallélisme est l'exécution rapide des tâches. Une fois ceci fait, il est donc question de passer à la pratique avec OPenMp en occurrence et réaliser (ou alors de bien comprendre) même de la programmation GPU comme le fait le géant **NVidia** et/ou avec **Javascript** grâce à ses requêtes asynchrones qui font intervenir le parallélisme au même titre que **Django** avec les **workers**, **C**, **C++**, **Java** les **threads**.

TRAVAUX DIRIGES

Exercice 1

1. C'est quoi le parallélisme et quel est son but ?
2. Présenter la classification de **Flynn** pour les architectures de machines parallèle.
3. A quoi sert un modèle de machine parallèle ? citez deux exemples.
4. Pour éviter l'incohérence des données lors de l'adressage de la mémoire par les processeurs, la synchronisation des exécutions des processeurs est nécessaire. Citez en expliquant, trois mécanismes pouvant être mis en place pour assurer cette synchronisation.
5. La résolution d'un problème à l'aide d'un algorithme parallèle est toujours plus efficace que sa résolution avec un algorithme séquentiel ? Justifiez

Exercice 2

1. Citez quelques APIs permettant d'implémenter le parallélisme.
2. Sur quel type d'architecture de machine parallèle doit-on utiliser : OpenMP, MPI ?
3. Donnez la structure d'un programme OpenMP écrit en langage C.
4. OpenMP repose sur trois concepts fondamentaux. Citez en donnant le rôle de chaque concept.
5. Quel est le rôle de la directive **parallèle** ? Donnez sa syntaxe et sa description.
6. même question pour la directive **barrière**.
7. Comment implémenter les barrières de synchronisations en environnement partagée ?

Exercice 3

1. Rappeler les différentes catégories de systèmes distribués et donner pour chacune d'elle un exemple.
2. Rappeler les différents modes de fonctionnement d'une PRAM. Montrer que certains d'entre eux peuvent également s'interpréter en termes d'architecture distribuée.

Exercice 4

```
#include <stdio.h>
#include <stdlib.h>
#include "omp.h"
#include "header.h"

Int main () {
    Int x = 3 ;
    Int y = 4 ;
    Int z = 0 ;
    Printf (" valeurs de x y et z : %i %i %i \n », x, y, z) ;
    #pragma omp parallel private(x,y) firstprivate(z)
    {
        tid = omp_get_thread_num();
        x = x + tid;
        y = 2 + tid;
        z = 1 + z;
        printf (" parallel x y z: %i %i %i \n",x, y, z);
    }
    printf ("Non parallel x y z: %i %i %i \n",x, y, z);
    return 0;
}
```

- a- Identifiez et corrigez les erreurs sur le programme ci-dessus.
- b- Donnez un éventuel résultat de l'exécution du programme corrigé.
- c- Quel est le nombre de threads par défaut ?

Exercice 5

Considérons le bout de code ci-dessous :

```
#include <stdio.h>

int main() {
    #pragma omp parallel
    printf("Hello,\n");
    printf("World!\n");
    return 0;
}
```

1. Compiler ce programme avec et sans l'option **-fopenmp**.
2. Exécuter ce programme dans les deux cas.
3. Quel est le nombre de threads par défaut ? Est-ce raisonnable ?
4. Changer le nombre de threads utilisé pour exécuter votre programme.

Exercice 5

Soit une boucle que l'on désire paralléliser à l'aide de OpenMP. Soit **p** le nombre de fils d'exécution disponibles et **n** le nombre total d'itérations de la boucle.

1. Pour la clause **schedule(static,)** de la directive **for** de OpenMP, où **k** est un nombre entier positif, dites à quel fil d'exécution sera confiée quelle itération de la boucle ? Numérotez vos fils de **0** à **p - 1** et vos itérations de **0** à **n - 1**. Formulez votre réponse en fonction de **p**, **n** et **k**.
2. En supposant que chacune des **n** itérations demande le même temps d'exécution, que changerait la clause **schedule(dynamic, k)** ?
3. Parmi les stratégies **static** et **dynamic**, laquelle offrirait la meilleure performance et pourquoi ?

Exercice 6 calcul de π

On souhaite paralléliser un programme séquentiel pour le calcul de π par la méthode des Rectangles sur **n** intervalles avec un pas **h**

$$\pi = \int_0^1 \frac{4}{1+x^2} dx.$$

- a- Écrire le programme séquentiel permettant de calculer la valeur de π

b- Paralléliser le programme séquentiel précédent

Exercice 7 : Tri par insertion

Soit un tableau T de n éléments entiers, chaque élément ne pouvant apparaître qu'une seule fois dans le tableau. On souhaite trier ce tableau en utilisant un algorithme PRAM de tri par insertion sur p processeurs organisés linéairement, c'est-à-dire que chaque processeur dispose d'une zone de mémoire propre qui ne peut être écrite que par lui-même.

- 1- Rappeler la version séquentielle de l'algorithme de tri par insertion. Quelle est sa complexité ?
- 2- Proposer un algorithme PRAM pour $p = n^2$. Quelle est sa complexité ? Quel type d'accès mémoire est nécessaire d'utiliser ?
- 3- Déterminer son facteur d'accélération, son efficacité et son travail ?

BIBLIOGRAPHIE

- [1] *THE MANY CORE SHIFT: Microsoft Parallel Computing Initiative Ushers Computing into the Next Era*
- [2] *Les aspects pratiques du parallélisme*, Violaine Louvet, Institut Camille Jordan
L'essentiel de MPI-1, **Stepahen Vialle**
- [3] *Processus concurrents et parallélisme Chapitre 6 - Notions de calcul parallèle et distribué*,
Gabriel Girard
- [4] *Initiation au calcul parallèle*, **Khodor Khadra**
- [TSBR18] *Introduction to Parallel Computing: From Algorithms to Programming on State-of-the-Art*, **Roman Trobec, Boštjan Slivnik, Patricio Bulic, Borut Robi** ´ c
Platforms – Monograph – July 2, 2018
- [Ken14] *Solutions parallèles efficaces sur le modèle CGM d'une classe de problèmes issus de la programmation dynamique*, **Vianney Tchendji Kengne**

WEBOGRAPHIE

- [1] <https://www.geekboots.com/story/parallel-computing-and-its-advantage-and-disadvantage>
- [2] <https://www.lesechos.fr/tech-medias/hightech/ordinateur-quantique-cinq-questions-pour-enfin-tout-comprendre-1323707>
- [3] http://perso-laris.univ-angers.fr/~chapeau/papers/congres/resume_JIONC_2018_Chapeau.pdf
- [4] <https://www.nginx.com/blog/inside-nginx-how-we-designed-for-performance-scale/>
- [5] <https://kinsta.com/fr/base-de-connaissances/qu-est-ce-que-nginx/>
- [6] <https://www.journaldunet.com/economie/finance/1195520-blockchain-definition-et-application-de-la-techno-derriere-le-bitcoin-juin-2021/>
- [7] https://e6.ijs.si/~roman/files/Book_jul2018/book/book.pdf

- [8] <https://www.capital.fr/economie-politique/comment-l-informatique-a-pris-le-controle-de-wall-street-604813>
- [9] <https://www.jumpstartmag.com/top-10-most-powerful-supercomputers/>