Introduction aux architectures programmables modernes

EI-SE5 – Calcul haute performance

Adrien Cassagne

Le 13 septembre 2022

Table des matières

Intervenants

1 Introduction

- Deux intervenants, Adrien Cassagne et Quentin Meunier
  - Enseignant-chercheur à Sorbonne Université
  - Recherche : Laboratoire d'Informatique de Sorbonne Université (LIP6)
    - Architecture et Logiciels pour Systèmes Embarqués sur Puce (ALSoC)
    - Sécurité des architectures CPU
    - Chaines de traitement du signal pour des architectures parallèles
  - E-mail : adrien.cassagne [at] lip6 [dot] fr

Contenu et organisation du cours

- 7×2 heures de cours magistraux
  - Cours 1 : Introduction aux architectures programmables modernes
  - Cours 2 : Optimisations séquentielles [CPU]
  - Cours 3 : Modèle *Single Instruction Multiple Data* (SIMD) [CPU]
  - Cours 4 : Modèle *Single Program Multiple Data* (SPMD) [CPU]
  - Cours 5 : Modèle *Single Instruction Multiple Threads* (SIMT) [GPU]
  - Cours 6 : Modèle *multi-threads* [CPU]
  - Cours 7 : Modèle *multi-threads* [CPU]
- 7×2 heures de travaux dirigées et pratiques
  - Utilisation de l'environnement EasyPAP
  - Essentiellement du traitement de l'image
- 4×4 heures de projet
  - **Évaluation sur les projets, soutenances le mardi 29 novembre**
  - Sujet à déterminer

Sondage : systèmes d'exploitation et architectures matérielles

1 Introduction

- Rendez-vous ici : https://app.wooclap.com/HBSLCO

Sondage : systèmes d'exploitation et architectures matérielles

1 Introduction

- Rendez-vous ici : https://app.wooclap.com/HBSLCO
- Que retenir ?
  - Grande diversité des composants dans les ordinateurs modernes
  - Plusieurs composants programmables (le CPU et le GPU)
    - Machines hétérogènes
    - Machines parallèles (multi-cœur mais pas que...)
  - Peu de programmeurs sont réellement capables de programmer ces architectures

Objectifs de ce cours

En général :

- Comprendre les architectures d'aujourd'hui
- Apprendre les différents modèles de parallélisme disponibles
- Initiation à l'optimisation de code
- Initiation à la programmation parallèle

Objectifs de ce cours

En général :

- Comprendre les architectures d'aujourd'hui
- Apprendre les différents modèles de parallélisme disponibles
- Initiation à l'optimisation de code
- Initiation à la programmation parallèle

Aujourd'hui :

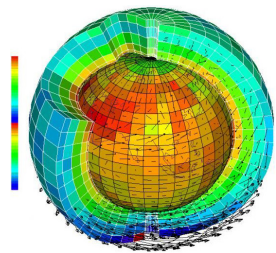
- Qu'est ce que le calcul haute performance ?
- Présentation succincte des architectures programmables
- Quelques modèles de parallélisme

Table des matières

2 Calcul haute performance

Le calcul haute performance, quésaco ?

- *High performance computing* ou HPC en anglais
- Association d'un grand nombre de processeurs (milliers voire millions) au sens large (CPU, GPU, FPGA, ...)
- Construction d'architectures massivement parallèles
- Résolution de problèmes **très très** gourmands en calcul
  - Modélisation des évolutions du climat
  - Optimisation de la production d'énergie
  - Analyse des sous-sols pour la recherche de pétrole
  - Craquage des codes en cryptographie
  - Analyse financière
  - ...



- **Enjeux majeurs pour les entreprises comme pour les états**

Les supercalculateurs

2 Calcul haute performance

- Support physique/matériel du calcul haute performance
- 1 pétaflop/s = 10^{15} opérations flottantes par seconde

Figure: Supercalculateur Adastra au CINES (Montpellier), 70 PFlops/s

Le top 500 : mais qui qui a le plus gros ?

Table: Top 500 en juin 2022 : https://www.top500.org/lists/top500/2022/06/.

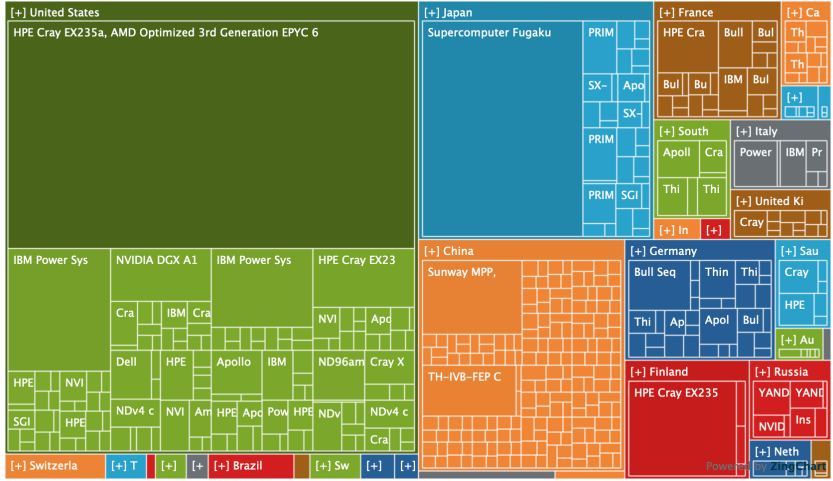
Rang	Nom du système	Pays	# Cœurs	Rmax (PFlop/s)	Rpeak (PFlop/s)	Puissance (kW)
1	Frontier	USA	8,730,112	1,102.00	1,685.65	21,100
2	Fugaku	Japon	7,630,848	442.01	537.21	29,899
3	LUMI	Finlande	1,110,144	151.90	214.35	2,942
4	Summit	USA	2,414,592	148.60	200.79	10,096
5	Sierra	USA	1,572,480	94.64	125.71	7,438
6	Sunway TaihuLight	Chine	10,649,600	93.01	125.44	15,371
7	Perlmutter	USA	761,856	70.87	93.75	2,589
8	Selene	USA	555,520	63.46	79.22	2,646
9	Tianhe-2A	Chine	4,981,760	61.44	100.68	18,482
10	Adastra	France	319,072	46.10	61.61	921

Le top 500 : mais qui qui a le plus gros ?

Table: Top 500 en juin 2022 : https://www.top500.org/lists/top500/2022/06/.

Rang	Nom du système	Pays	# Cœurs	Rmax (PFlop/s)	Rpeak (PFlop/s)	Puissance (kW)
1	Frontier	USA	8,730,112	1,102.00	1,685.65	21,100
2	Fugaku	Japon	7,630,848	442.01	537.21	29,899
3	LUMI	Finlande	1,110,144	151.90	214.35	2,942
4	Summit	USA	2,414,592	148.60	200.79	10,096
5	Sierra	USA	1,572,480	94.64		
6	Sunway TaihuLight	Chine	10,649,600	93.01		
7	Perlmutter	USA	761,856	70.87		
8	Selene	USA	555,520	63.46		
9	Tianhe-2A	Chine	4,981,760	61.44		
10	Adastra	France	319,072	46.10	61.61	921

Le top 500 (juin 2022) : *tree map*



Et alors ?

- Pas d'accès à une machine de ce type malheureusement
- Un assemblage d'un grand nombre (> 1000) d'ordinateurs
  - Un peu dopé aux hormones (plutôt 128 cœurs CPU par ordinateur)
  - Un réseau fibre optique de l'espace (mieux que la fibre de gamer)
  - Un espace de stockage digne d'un cloud mais en plus rapide

Et alors ?

- Pas d'accès à une machine de ce type malheureusement
- Un assemblage d'un grand nombre (> 1000) d'ordinateurs
  - Un peu dopé aux hormones (plutôt 128 cœurs CPU par ordinateur)
  - Un réseau fibre optique de l'espace (mieux que la fibre de gamer)
  - Un espace de stockage digne d'un cloud mais en plus rapide
- Approximation pour ce cours
  - **Si l'on sait programmer son ordinateur personnel "correctement" alors on n'est pas très loin de pouvoir programmer un supercalculateur**

Et alors ?

- Pas d'accès à une machine de ce type malheureusement
- Un assemblage d'un grand nombre (> 1000) d'ordinateurs
  - Un peu dopé aux hormones (plutôt 128 cœurs CPU par ordinateur)
  - Un réseau fibre optique de l'espace (mieux que la fibre de gamer)
  - Un espace de stockage digne d'un cloud mais en plus rapide
- Approximation pour ce cours
  - **Si l'on sait programmer son ordinateur personnel "correctement" alors on n'est pas très loin de pouvoir programmer un supercalculateur**
  - À relativiser puisque nous ne verrons pas (ou peu) les problèmes liées aux transferts de données entre différents nœuds (nœud = ordinateur)

Et alors ?

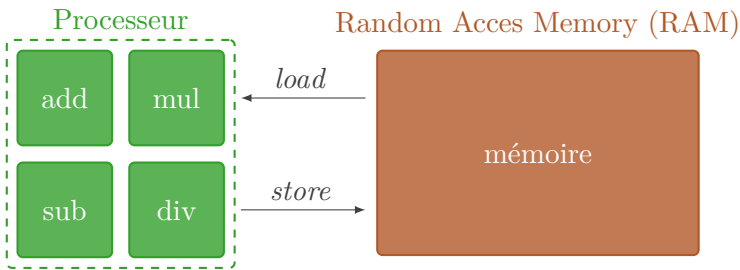
- Pas d'accès à une machine de ce type malheureusement
- Un assemblage d'un grand nombre (> 1000) d'ordinateurs
  - Un peu dopé aux hormones (plutôt 128 cœurs CPU par ordinateur)
  - Un réseau fibre optique de l'espace (mieux que la fibre de gamer)
  - Un espace de stockage digne d'un cloud mais en plus rapide
- Approximation pour ce cours
  - **Si l'on sait programmer son ordinateur personnel "correctement" alors on n'est pas très loin de pouvoir programmer un supercalculateur**
  - À relativiser puisque nous ne verrons pas (ou peu) les problèmes liées aux transferts de données entre différents nœuds (nœud = ordinateur)
- Aussi valable pour
  - La programmation de systèmes embarqués et temps réel
  - Les logiciels gourmands en calcul (jeux-vidéos, traitement de l'image, etc.)

Table des matières

3 Architectures programmables

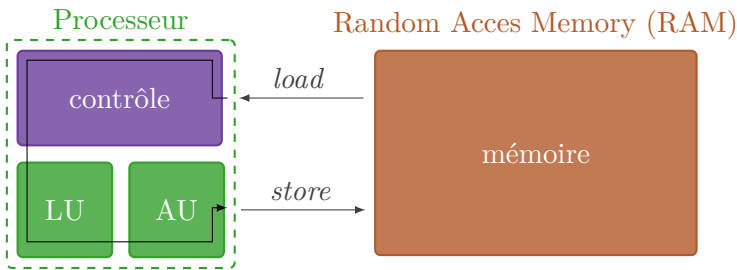
Architecture CPU (beaucoup) simplifiée

- Unités de calcul et mémoire
  - La mémoire permet de charger (*load*) et de stocker (*store*) des données
  - Les unités de calcul permettent de transformer les données



Architecture CPU (un peu moins) simplifiée

- Plus précisément, dans un CPU il y a :
  - Des unités de contrôle (`if`, `goto`, placement d'instructions, ...)
  - Des unités logiques, LU ($==$, $!=$, $>$, $<$, ...)
  - Des unités arithmétiques, AU ($+$, $*$, $-$, $/$, ...)



Architecture CPU : cycle, fréquence et débit

- Un tick d'horloge = un cycle
- A chaque cycle, le CPU effectue une tâche élémentaire
- La fréquence = le nombre de cycles par seconde (en Hertz)

Architecture CPU : cycle, fréquence et débit

- Un tick d'horloge = un cycle
- A chaque cycle, le CPU effectue une tâche élémentaire
- La fréquence = le nombre de cycles par seconde (en Hertz)
- Fréquence CPU/RAM modernes : entre 1 GHz et 4 GHz (1 GHz = 10^9 Hz)
- Débit de la mémoire vive (RAM) $\approx$ 50 Go/s (DDR5)
- CPU capable consommer/produire des données à $\approx$ 5 To/s (i9-9900K)
  - **Le CPU est beaucoup plus rapide ($\times$100) !**
  - Comment résoudre ce problème ?

Architecture CPU : hiérarchie mémoire

Constat : la plupart des applications réutilisent souvent les mêmes données

Architecture CPU : hiérarchie mémoire

Constat : la plupart des applications réutilisent souvent les mêmes données

- Mémoire plus rapide entre le CPU et la RAM = **mémoire cache**
  - La mémoire plus rapide est plus chère et occupe de l'espace physique
  - Cette mémoire (= cache) est beaucoup plus petite que la RAM
  - 3 niveaux de cache (dans le processeur) :
    - L1, le plus rapide et le plus petit (32 Ko), temps d'accès $\approx$ 2 cycles
    - L2, plus lent que le L1 mais plus grand (1 Mo), temps d'accès $\approx$ 10 cycles
    - L3, plus lent que le L2 mais plus grand (4 Mo), temps d'accès $\approx$ 30 cycles
  - La latence d'accès à la RAM $\approx$ 100 cycles

Architecture CPU : hiérarchie mémoire

- Scénario d'un **premier** chargement (*load*) d'une donnée en mémoire

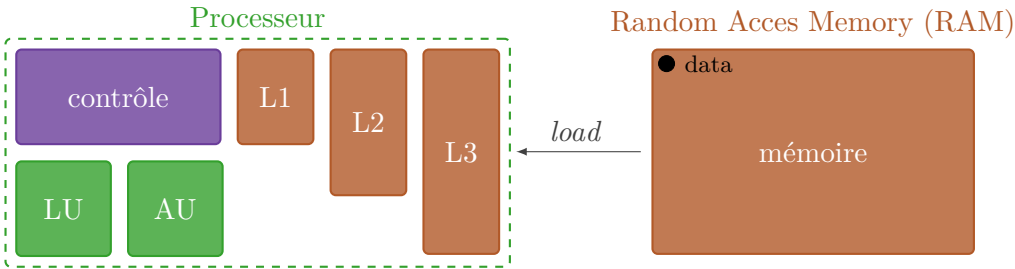


Figure: Chargement depuis la RAM: 100 cycles

Architecture CPU : hiérarchie mémoire

- Scénario d'un **premier** chargement (*load*) d'une donnée en mémoire

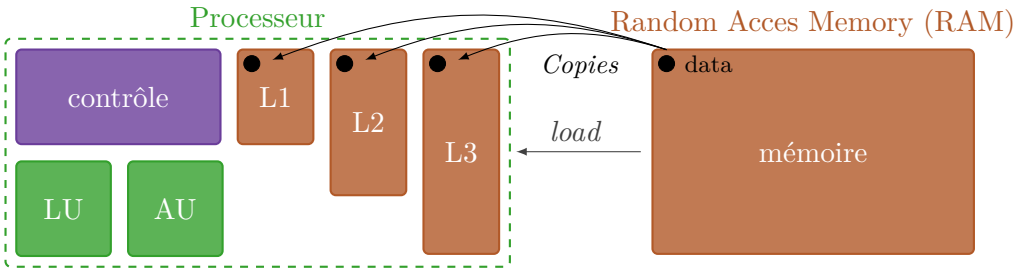


Figure: Chargement depuis la RAM: 100 cycles

Architecture CPU : hiérarchie mémoire

- Scénario d'un **second** chargement (*load*) de la même donnée en mémoire

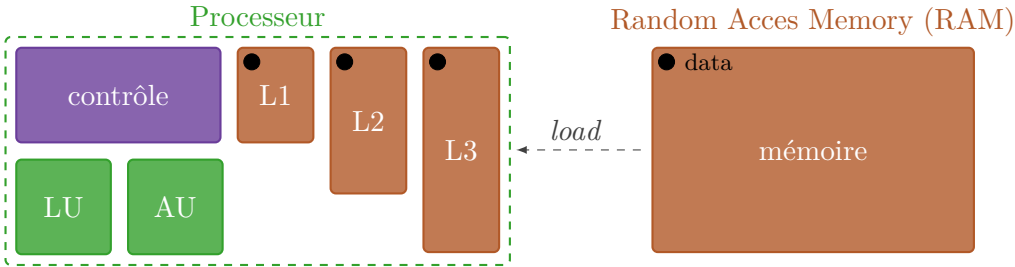


Figure: Chargement depuis le cache L1: 1-2 cycles

Architecture CPU : hiérarchie mémoire

- Scénario d'un **second** chargement (*load*) de la même donnée en mémoire

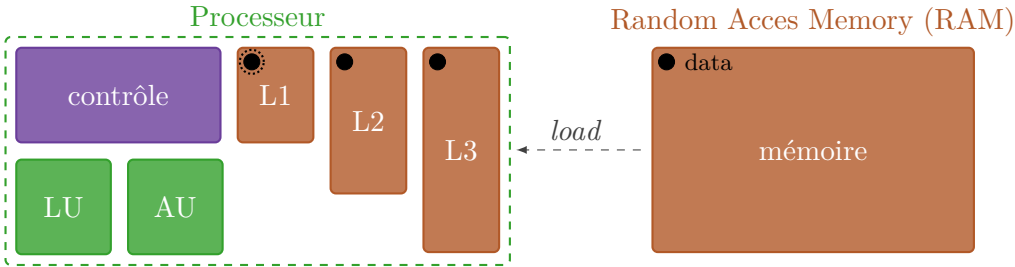


Figure: Chargement depuis le cache L1: 1-2 cycles

- C'est le principe de la **localité temporelle !**

Architecture CPU : hiérarchie mémoire

- Scénario d'une écriture (*store*) en mémoire
- Deux modes possibles
  - ***Write-through***
    - "Éventuellement plus simple" à implanter en matériel
    - Écriture à la fois dans la RAM et dans le cache
    - Choix fait dans certaines architectures embarquées
  - ***Write-back***
    - "Plus compliqué" à implanter en matériel (protocole de cohérence de cache)
    - Écriture dans le cache et écriture dans la RAM uniquement si la donnée est invalidée du cache
    - Écrire moins souvent dans la RAM = consommer moins d'énergie
    - Le choix fait par la majorité des architectures (ordinateurs, HPC, embarquée)

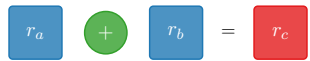
Architecture CPU : localité spatiale

- Une ligne de cache = un certain nombre de données (par ex. 256 bits)
- Le plus petit paquet de données qui transitent entre la RAM et le CPU est une ligne de cache
- **Il est plus intéressant d'accéder aux données de manière contigüe !**
- Sinon une partie des données venant de la RAM sera inutilisée (= perte de bande passante)

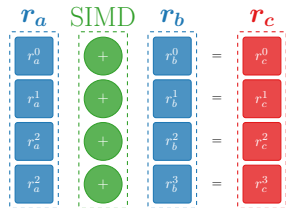
Le fait d'utiliser des données qui sont dans une même ligne de cache est appelé la **localité spatiale** !

Architecture CPU : instructions SIMD

- Instruction scalaire: produit une donnée pendant 1 cycle



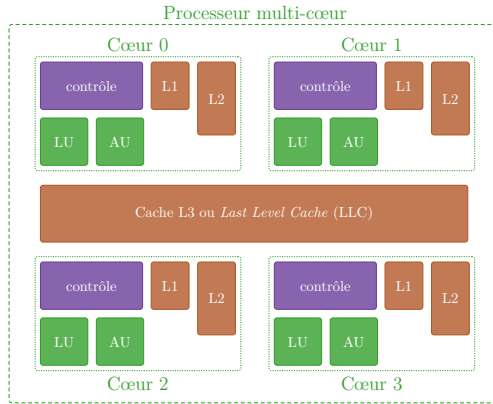
- Une instruction SIMD produit n données pendant 1 cycle



- SIMD = *Single Instruction Multiple Data*
- Les instructions SIMD opèrent sur des registres dit “vectoriels”

Architecture CPU : multi-cœur

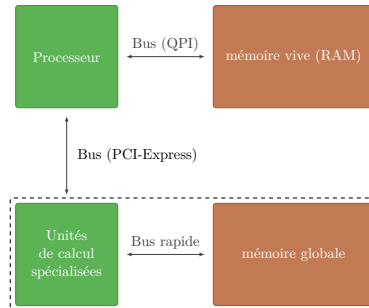
- Les CPU modernes sont maintenant tous multi-cœur
- Généralement les caches L1 et L2 sont dédiés à un cœur
- Les cœurs partagent le *Last Level Cache* (LLC) ou le L3



Les accélérateurs de calcul

3 Architectures programmables

- Matériel physiquement séparé du CPU (ou pas avec les SoC aujourd'hui...)
- Souvent connecté au CPU via le bus PCI-Express
- Possède sa propre mémoire vive (mémoire globale)
- Exemples d'accélérateurs
  - *Graphics Processing Unit* (GPU)
  - *Field-Programmable Gate Array* (FPGA)
  - *Many Integrated Cores* (MIC)

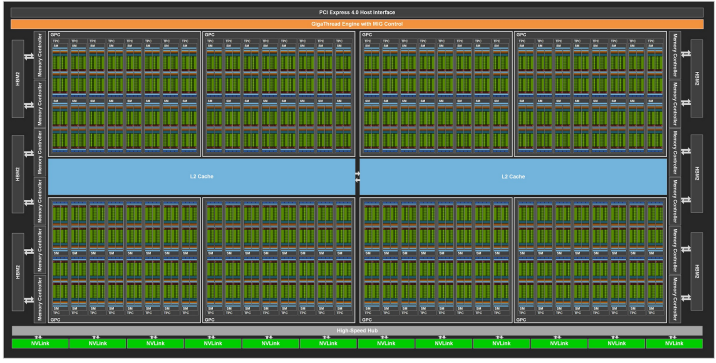


Architecture GPU

- D'abord conçu pour le traitement d'images
- Architecture massivement parallèle
- Moins d'unités de contrôle que pour les CPU mais plus d'unités de calcul
- Mémoire globale plus rapide que la RAM du CPU ($\approx$ 500 Go/s)
- Ratio performance/consommation d'énergie généralement plus intéressant que sur CPU
- Souvent adapté au calcul scientifique
- Des unités d'accélération matérielle pour l'IA (*tensor cores*)

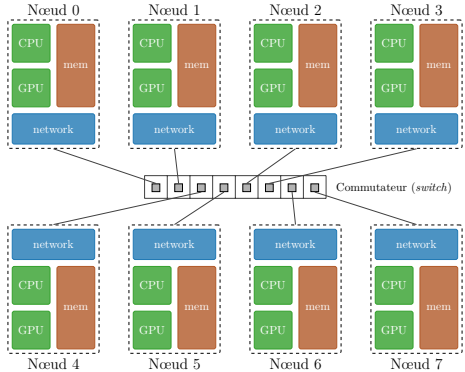
Architecture GPU : Nvidia Ampere

3 Architectures programmables



Architecture d'un supercalculateur

3 Architectures programmables



- Un supercalculateur très simpliste
- Interconnexion d'ordinateurs avec un réseau en étoile (commutateur)
- La performance théorique maximale est 8 fois celle d'un nœud

Table des matières

4 Modèles d'exécution parallèles

Constat

Nous avons vu précédemment que toutes les architectures modernes exploitent le parallélisme.

- Impossible de continuer d'augmenter les fréquences (consomme trop d'énergie)
- Solution : augmenter le nombre de cœurs à fréquence constante
- Le parallélisme est partout : supercalculateurs, ordinateurs, smartphones, montres connectées, consoles de jeu, télévisions, ...
- **Peut-on encore programmer sans tenir compte du parallélisme ?**

Modèle d'exécution Single Instruction Multiple Data

- Vient directement de la taxonomie de Flynn (1966)
- Une même instruction est appliquée à plusieurs données
- Modèle transposable à un cœur CPU et parfois à un cœur GPU
- Se programme généralement en assembleur
  - Ou plutôt avec des fonctions intrinsèques pour ne pas avoir à gérer l'allocation des registres
- Quelques jeux d'instructions SIMD : SSE, AVX, NEON, SVE

Modèle d'exécution multi-threads

- Plusieurs fils d'exécution au sein d'un même processus (IBM OS/360, 1967)
- **La mémoire est partagée** entre les différents fils d'exécution
- Modèle utilisé pour programmer plusieurs cœurs CPU d'un même ordinateur
- Se programme généralement avec des bibliothèques
- Parfois directement avec des langages dédiés (ex. Cilk)
- Quelques bibliothèques *multi-threads* : threads POSIX, OpenMP, Threading Building Blocks (TBB)

Modèle d'exécution Single Program Multiple Data

- Un exécutable unique mais exécuté sur des données différentes
- Plusieurs instances du même programme (processus) lancées en parallèle
- **La mémoire est distribuée** entre les différents processus
- Modèle utilisé pour programmer plusieurs cœurs CPU d'un même ordinateur ou pour communiquer entre différents nœuds d'un supercalculateur
  - **Modèle de prédilection en calcul haute performance**
- Se programme généralement avec des bibliothèques
- Un standard bien établi : *Message Passing Interface* (MPI, 1991)
- Peut être vu comme une surcouche aux *sockets* (réseau)
- Beaucoup d'implémentations : OpenMPI, MPICH, Intel MPI, IBM MPI, HP MPI, ...

Modèle d'exécution Single Instruction Multiple Threads

- Un mix entre le modèle SIMD et le modèle *multi-threads* (Nvidia, 2007)
- En SIMT, il y a des *Work-items* qui font tous la même opération (comme en SIMD)
- Plusieurs *Work-items* sont rassemblés dans des *Wavefronts* qui correspondent plus à des threads dans le modèle *multi-threads*
- **Modèle utilisé pour programmer les GPU** et parfois les CPU/FPGA
- Se programme généralement avec des langages dédiés (CUDA, OpenCL, Metal, ...)

Modèles d'exécution parallèles : récapitulatif

- *Single Instruction Multiple Data* (SIMD)
  - Permet de programmer un cœur CPU ou un/des *Work-item(s)* GPU
  - Plus souvent utilisé pour programmer un cœur CPU
  - Assembleur, fonctions intrinsèques
- *Multi-threads*
  - Permet de programmer plusieurs cœurs CPU
  - Bibliothèques : threads POSIX, OpenMP
- *Single Program Multiple Data* (SPMD)
  - Permet de programmer plusieurs cœurs CPU et/ou plusieurs nœuds d'un supercalculateur
  - Bibliothèques : MPI
- *Single Instruction Multiple Threads* (SIMT)
  - Permet de programmer un GPU
  - Langages : OpenCL, CUDA

Notion d'accélération (*speedup*)

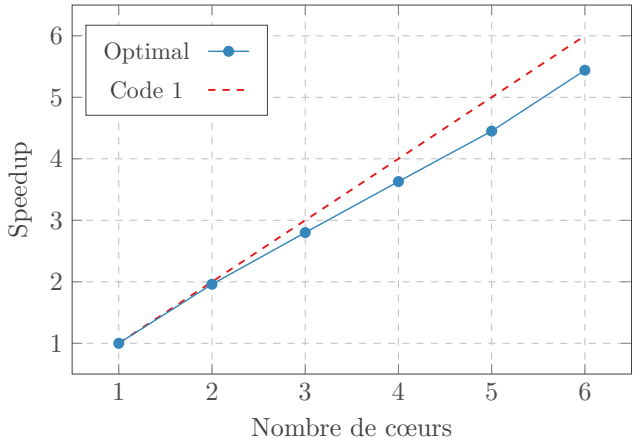
$$\mathcal{S} = \mathcal{D}_{seq}/\mathcal{D}_{par},$$

avec $\mathcal{D}_{seq}$ le temps mesuré pour la version 1 cœur du code et $\mathcal{D}_{par}$ le temps mesuré pour la version parallèle du code.

# cœurs	Durée	Speedup
1	98 ms	1.00
2	50 ms	1.96
3	35 ms	2.80
4	27 ms	3.63
5	22 ms	4.45
6	18 ms	5.44

- Le temps séquentiel est utilisé commet temps de référence
- Une accélération optimale est égale au nombre de cœurs utilisés (pas plus !)

Notion d'accélération sur un graphe



Loi d'Amdahl

- Peut-on indéfiniment augmenter le parallélisme pour accélérer nos codes ?
  - Amdahl a dit non !
  - Pour être plus précis, cela dépend des caractéristiques du code...
  - Si le code est entièrement parallélisable : l'accélération est infinie
  - Si le code n'est PAS entièrement parallélisable : il y a une limite

$$\mathcal{S}_{max} = \frac{1}{1 - ft_p},$$

avec $\mathcal{S}_{max}$ l'accélération (*speedup*) maximale et ft_p la fraction de temps parallèle dans le code ($0 \leq ft_p \leq 1$).

Loi d'Amdahl

- Si on prend un code composé de deux parties :
  - 20 % est intrinsèquement séquentielle
  - 80 % est parallèle
- Quel est l'accélération maximale que l'on peut atteindre ?

$$\mathcal{S}_{max} = \frac{1}{1 - ft_p} = ...$$

Loi d'Amdahl

- Si on prend un code composé de deux parties :
  - 20 % est intrinsèquement séquentielle
  - 80 % est parallèle
- Quel est l'accélération maximale que l'on peut atteindre ?

$$\mathcal{S}_{max} = \frac{1}{1 - ft_p} = \frac{1}{1 - 0.8} = \frac{1}{0.2} = 5.$$

Loi d'Amdahl

- Si on prend un code composé de deux parties :
  - 20 % est intrinsèquement séquentielle
  - 80 % est parallèle
- Quel est l'accélération maximale que l'on peut atteindre ?

$$\mathcal{S}_{max} = \frac{1}{1 - ft_p} = \frac{1}{1 - 0.8} = \frac{1}{0.2} = 5.$$

C'est peu si l'on considère des architectures qui ont des dizaines de cœurs CPU !

Q&R

Merci pour votre écoute !
Avez-vous des questions ?