Cours SIM203

Initiation au calcul haute performance

Introduction — Architecture CPU — Calcul mono-cœur

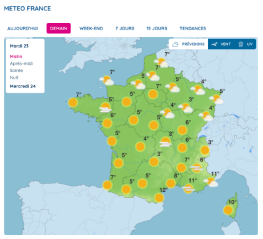
Axel Modave — 18 mars 2024

Contexte, motivation et généralités

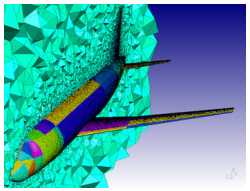
Architecture du CPU

Stratégies pour calcul mono-cœur

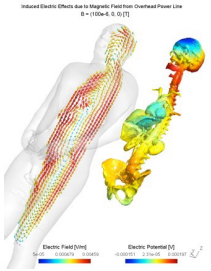
Applications utilisant le calcul haute performance ...



Météo

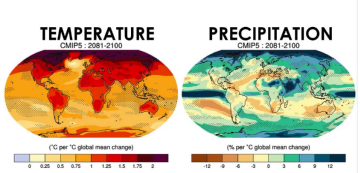


Industrie aérospatiale

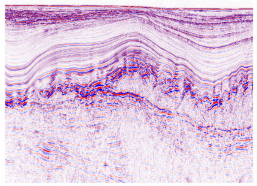


Applications biomédicales

(clone virtuel)



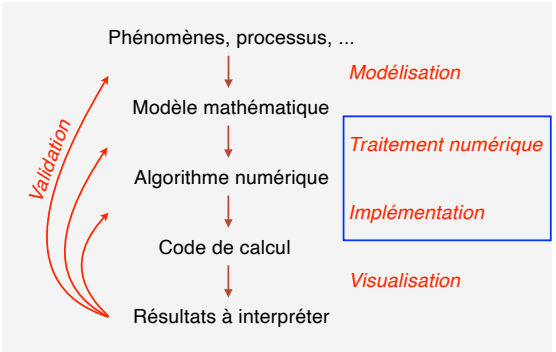
Évolution du climat (GIEC)



Industrie pétrolière

Calcul haute performance !?!

Conception d'un outil de simulation numérique ...



On cherche à concevoir des codes de simulation rapides pour des problèmes coûteux :

→ **Optimisation des codes** pour tirer parti des **ressources de calcul** et de la **mémoire disponible**.

→ En amont, **modifications les algorithmes** pour permettre des implémentations plus efficaces.

Quelques machines de calcul

de l'ordre de 100 GFLOP/s 10^{11}

Stations de travail des salles informatiques (ENSTA)

Chaque station est équipée avec un **processeur quad-core** :
Intel(R) Core(TM) i5-6500 CPU @ **3.20GHz**
Architecture du processeur de type 'Haswell'

Taille réelle :
3,75cm x 3,75cm

Débit arithmétique : ~ **80 GFLOP/s** = 80 · 10^9 FLOP/s
(estimation en utilisant des benchmarks)

FLOP/s = Floating point operations per second
Opérations à virgule flottante par seconde

Mémoire vive : 8 GB dans la DRAM

B = Byte (8 bits)

DRAM = Dynamic random-access memory
Mémoire principale où sont stockées les données

≠ Disque dur (HDD)

Sources : https://ranker.sisoftware.co.uk/show_run.php?q=c2ffc9f1d7b6d7eadbecd5e1d6eec8ba87b791f491ac9cbac9f4cc&l=en

Quelques machines de calcul

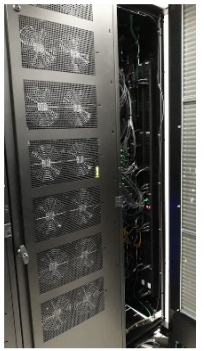
de l'ordre de 100 TFLOP/s 10^{14}

Calculateur *cholesky* du mésocentre de l'IP Paris

General view

Front

Back



Sources : https://meso-ipp.gitlab.labos.polytechnique.fr/user_doc/

Quelques machines de calcul

de l'ordre de 100 TFLOP/s 10^{14}

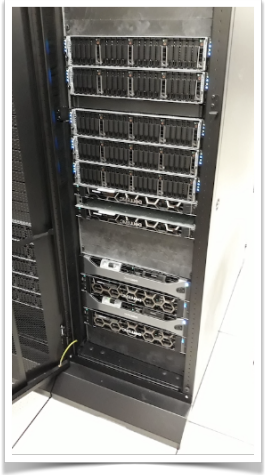
Calculateur *cholesky* du mésocentre de l'IP Paris

Équipé de 72 nœuds de calcul :
- 68 noeuds "CPU" avec 2 CPUs
- 4 noeuds "GPU" avec 2 CPUs et 2 GPUs

Équipé d'un système de stockage
Équipé d'un système communication entre les nœuds

Débit arithmétique :
1.715 GFLOP/s par CPU
14.000 GFLOP/s par GPU
~ 360 TFLOP/s pour le calculateur
(estimations en utilisant des benchmarks)

Mémoire vive :
entre 160 GB et 384 GB par nœud de calcul
~ 16 TB pour le calculateur



Sources : https://meso-ipp.gitlab.labos.polytechnique.fr/user_doc/

Quelques machines de calcul

de l'ordre de 10 PFLOP/s 10^{16}

Supercalculateur *Jean Zay* du CNRS/IDRIS/CENCI
(Institut du développement et des ressources en informatique scientifique)

Équipé de 2.227 nœuds de calcul :
- 1.580 noeuds "CPU" avec 2 CPUs
- 647 noeuds "GPU" avec 2 CPUs et 4 GPUs

Équipé d'un système de stockage
Équipé d'un système communication

Débit arithmétique *"de crête"* :
~ 36,2 PFLOP/s

Performance de crête
("peak performance") calculée
sur base des capacités physiques

Théoriquement impossible
à dépasser!

Mémoire vive :
~ 50 TB

Sources : *http://www.genci.fr/fr/content/calculateurs-et-centres-de-calcul*

Classement des supercalculateurs

vers 1 EFLOP/s 10^{18}

https://top500.org/

top500.org/lists/top500/2022/11/

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, **HPE** DOE/SC/Oak Ridge National Laboratory United States	8,730,112	1,102.00	1,685.65	21,100
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, **Fujitsu** RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, **HPE** EuroHPC/CSC Finland	2,220,288	309.10	428.70	6,016
4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, **Atos** EuroHPC/CINECA Italy	1,463,616	174.70	255.75	5,610
5	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, **IBM** DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096

Rmax ***Performance effective*** (débit arithmétique évalué avec la librairie LINPACK — librairie d'algèbre linéaire)
Rpeak ***Performance de crête*** (débit arithmétique de crête — max. théorique sur base des caractéristiques physiques)
Power Consommation électrique de l'installation

Rmax < Rpeak

Objectif des programmeurs :
Rmax aussi proche de **Rpeak**

Cours SIM203

Objectifs du cours

- Mise en œuvre informatique efficace pour du calcul scientifique *sur un processeur multi-coeur*
- Améliorer la maitrise des environnements/outils de programmation *(compilateurs, machines...)*
- Sens critique sur l'évaluation de la performance d'un code de calcul
- Culture scientifique autour du calcul haute performance *(High Performance Computing – HPC)*

Cours destiné à des étudiants des filières utilisant la simulation numérique
(mathématique appliquée, mécanique numérique, ...)

Organisation

- Cours magistraux et TP sur ordis persos *(exercices de programmation en C++ ; projets)*
- Projet : *"Éléments finis discontinus" (en C++)*

Pré-requis : Programmation en C ou C++ ; ANN201 pour le projet

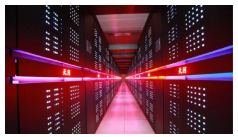
Évaluation

- 1 devoir (mini-rapport/mini-codes) en solo
- 1 projet (mini-rapport/codes intermédiaires + rapport/codes finaux) en solo ou binôme

Pas de soutenance !

https://sim203.pages.math.cnrs.fr/

Vision globale de calcul haute performance ...

Cœur	**Processeur** *(plusieurs cœurs)*	**Cluster** *(plusieurs processeurs)*
		
Calcul séquentiel Calcul vectoriel	Calcul parallèle à ***mémoire partagée***	Calcul parallèle à ***mémoire distribuée***
Séances 1 et 2	Les cœurs peuvent travailler en parallèle sur des tâches différentes. Ils partagent des mémoires rapides (RAM + L2 ou L3) **Séances 3 et 4**	Les processeurs peuvent travailler en parallèle sur des tâches différentes Les données sont distribuées entre les RAM des processeurs, qui doivent communiquer (avec par ex. **MPI**). **Séance 5** *(intro)* **Cours AMS301 et AMS-I03** ***(3A ModSim et M2 AMS)***

Questions importantes:
Comment **gérer les opérations** ?
Comment **gérer les données** nécessaires pour les opérations ?

Contexte, motivation et généralités
Architecture du CPU
Stratégies pour calcul mono-cœur

Dans les salles d'ordi de l'ENSTA ...

Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz

Famille	Intel Core i5
Numéro du modèle	i5-6500
Fréquence	**3.20 GHz**
Vitesse du bus	8 GT/s
Type d'architecture	Haswell
Largeur de donnée	64 bit
Nombre de cœurs CPU	**4**
Nombre de threads	4

Taille réelle : 3.75cm x 3.75cm
Prix: environ $190 — $240

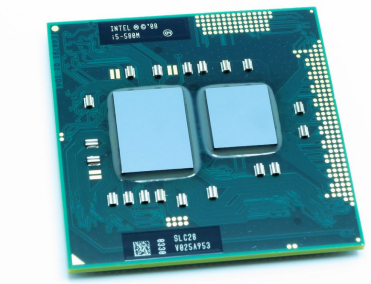
Cache L1	4 x **32KB** 8-way set associative instruction caches 4 x **32KB** 8-way set associative data caches
Cache L2	4 x **246KB** 8-way set associative caches
Cache L3	**6MB** 8-way set associative shared cache

Lignes de commande à tester :

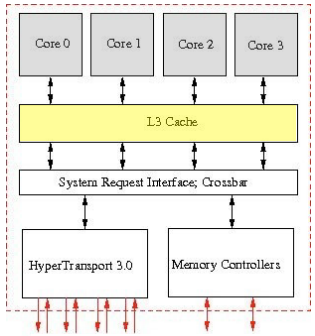
```
less /proc/cpuinfo
lscpu
```

Sources : http://www.cpu-world.com/CPUs/Core_i5/Intel-Core%20i5-4430.html

À quoi ressemble un processeur ?



Exemple de processeur quad-core :



Similaire aux processeurs des stations de l'ENSTA
Similaire au processeur de mon MacBook Pro

À quoi ressemble un cœur CPU ?

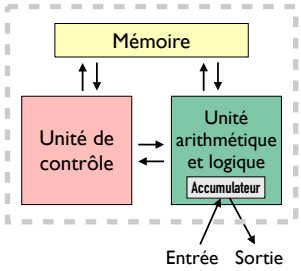
Architecture de Von Neumann ***(modèle très simplifié)***

Composition du cœur

Mémoire qui stocke le programme et les données
Unité de contrôle qui pilote les opérations et les données
Unité de de calcul qui exécute les instructions
Unité de arithmétique et logique (ALU)

Mode de fonctionnement pour une instruction

1. Recuperation de l'instruction ***[fetch]***
2. Décodage de l'instruction ***[decode]***
   (détermine les opérations et les opérandes)
3. Récupération des opérandes
4. Réalisation l'opération ***[execute]***
5. Écriture du résultat ***[writeback]***
6. Passage à l'instruction suivante



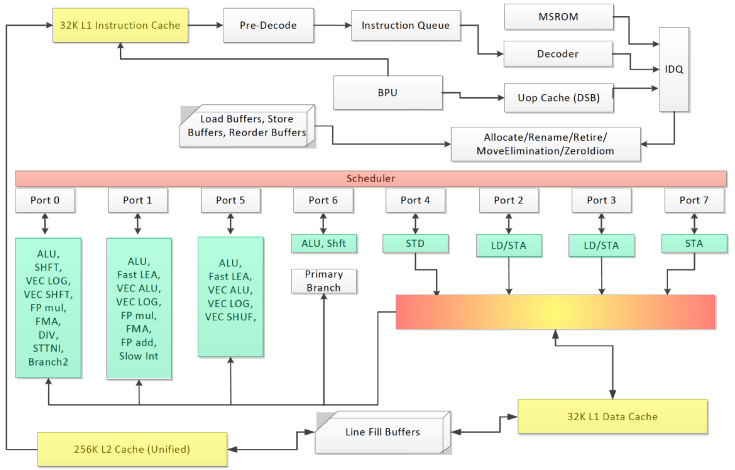
En vert : unités de calcul
En rouge : gestion/contrôle
En jaune : zones de mémoire

Au final, 3 types de tâches : { Operations sur les données / Transferts de données / Gestion de séquencement

À quoi ressemble un cœur CPU ?

En vert : unités de calcul
En rouge : gestion/contrôle
En jaune : zones de mémoire

Architecture de type *"Intel Haswell"*



Source : http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf

Unités de calcul

Quelques composants

ALU	Unité arithmétique et logique (*Arithmetic Logic Unit*)
FMA	Unité qui multiplie-accumule (*Fused Multiply–Add*)
FP ADD, FD MUL	Addition et multiplication de nombre à virgule flottante
DIV	Division
VEC ALU, VEC LOG	Unités vectorielles ...

Commentaires

- Les opérations peuvent se faire en demi, simple ou double précision.
  Si l'application le permets, diminuer la précision permets d'accélérer les calculs.

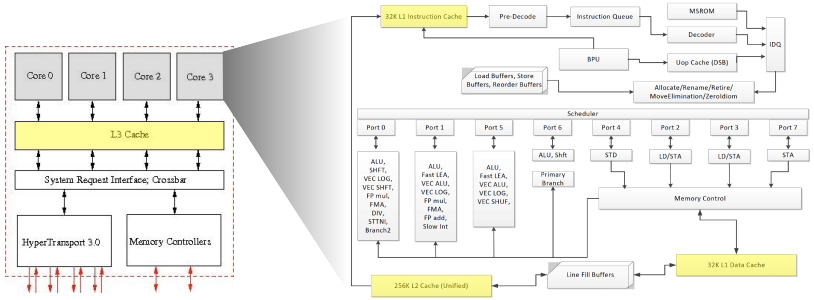
	16 bits (2 bytes)	32 bits (4 bytes)	64 bits (8 bytes)
Nombre entier	`short`	`int`	`long long`
Nombre à virgule flottante		`float`	`double`
	Demi précision	***Simple précision***	***Double précision***

- Sur les processeurs modernes, la combinaison d'opérations *"multiplie-accumule"* (FMA) est réalisé en une seule fois : `x ← ax + b`
  Cette combinaison d'opérations a le même coût que chaque opération séparée.
- Les divisions (DIV) sont très lentes en comparaison avec les multiplications (MUL).

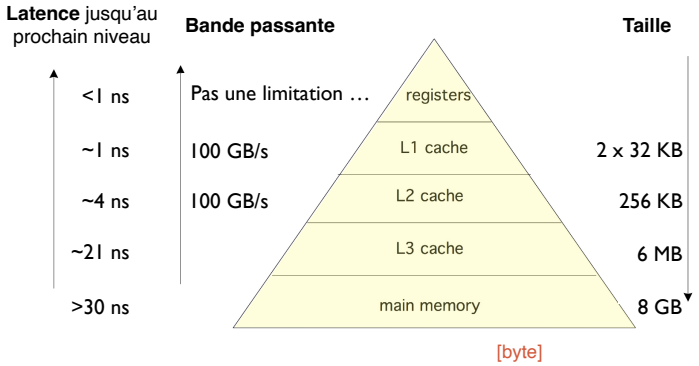
Hiérarchie des mémoires (1/6)

Dans beaucoup de cas, les transferts de données sont trop lents pour maintenir le processeur en activité (c'est le "mur de la mémoire").

- Dans le processeur, la **mémoire est divisée en niveaux**, du plus lent au plus rapide :
  Mémoire principale (DRAM) — cache L3 — cache L2 — cache L1 — registres
- **Les données traversent les différents niveaux** pour être dans la mémoire la plus rapide (les registres) avant d'être utilisées.



Hiérarchie des mémoires (2/6)

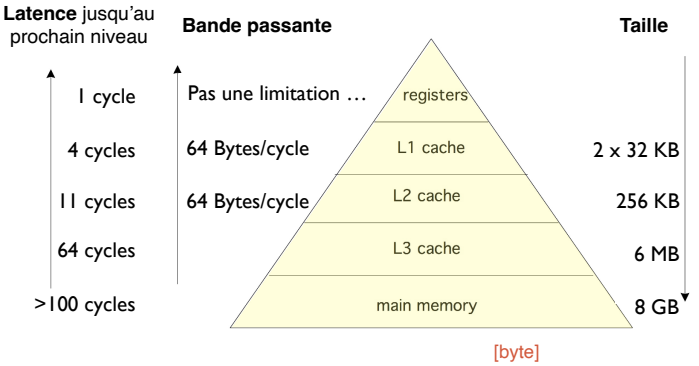


$$\text{Temps de transfert} = \text{Latence} + \frac{\text{Taille des données transférées}}{\text{Bande passante}}$$

[cycle] ou [s] [cycle] ou [s] [byte/cycle] ou [byte/s]

Latence : Temps minimum nécessaire pour transférer des données, quel que soit la quantité.
Bande passante : Débit de donnée pouvant être maintenu lorsque le transfert a commencé.
Un cycle : Inverse de la fréquence du processeur (de l'ordre de la nanoseconde … 1 ns = 10^{-9} s)

Hiérarchie des mémoires (3/6)



$$\text{Temps de transfert} = \text{Latence} + \frac{\text{Taille des données transférées}}{\text{Bande passante}}$$

[cycle] ou [s] [cycle] ou [s] [byte/cycle] ou [byte/s]

Latence : Temps minimum nécessaire pour transférer des données, quel que soit la quantité.
Bande passante : Débit de donnée pouvant être maintenu lorsque le transfert a commencé.
Un cycle : Inverse de la fréquence du processeur (de l'ordre de la nanoseconde … 1 ns = 10^{-9} s)

Hiérarchie des mémoires (4/6)

Dans le processeur, la **mémoire est divisée en niveaux** …

Registres

- Mémoire avec la plus rapide, mais la plus chère
- Construits sur la puce du processeur, proche des unités de calcul
- Séparent les instructions et les données

Caches L1, L2 et L3

- L1 — Caches les plus proches des registres, séparent données et instructions
- L2 — Cache privée, stocke à la fois données et instructions
- L3 — Cache partagée entre différents coeur

Mémoire principale (DRAM)

- Forme de RAM la moins chère, mais la plus lente
- La plupart des données vivent dans cette mémoire

Disque dur (HDD)

- Memoire permanente … très lente

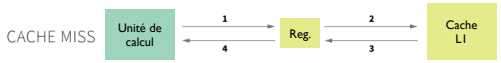
Hiérarchie des mémoires (5/6)

Les données traversent les différents niveaux pour être
dans la mémoire la plus rapide *(les registres)* avant d'être utilisées.

- **Les accès sont transparents pour le programmeur**

  Les données sont dans un registre, dans une mémoire cache ou dans la mémoire globale.
  Chargement depuis la mémoire la plus proche de l'unité de calcul, si les données y sont.

  Si les données se trouvent dans une mémoire cache → ***"cache hit"***
  Si les données ne se trouvent pas dans la mémoire cache → ***"cache miss"***
  … et on va chercher dans la mémoire du niveau suivant.



En dehors des transfers entre le disque dur (HDD) et la mémoire principale (RAM),
le programmeur n'a pas de contrôle direct sur la façon dont ces transferts se font,
mais il peut les influencer !

Hiérarchie des mémoires (6/6)

Les données traversent les différents niveaux pour être
dans la mémoire la plus rapide *(les registres)* avant d'être utilisé.

- **Les données sont transférées par blocs de taille minimum (la ligne de cache).**

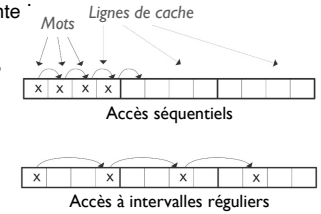
  La **ligne de cache** est la plus petite unité de donné transférée
  entre la mémoire principale et les caches *(ou entre les différentes caches)*.

  Chaque cache a sa propre taille de ligne *(architecture Haswell : 64 bytes (?) pour toutes les caches).*

  Si un élément est demandé dans une ligne de cache, la ligne complète est transférée.

  Commentaires:

  - Utiliser au maximum tous les éléments de la ligne,
    vous avez *"payé"* pour eux dans la bande passante
  - Des accès séquentiels sont très bons,
    … des accès à intervalles réguliers sont moyens,
    … des accès aléatoires sont très mauvais !



Contexte, motivation et généralités
Architecture du CPU
Stratégies pour calcul mono-cœur

Stratégie : Localité des données

Localité temporelle

- Si un élément doit être utilisé plusieurs fois, faire en sorte que les utilisations soient rapprochées *(maximise l'utilisation des registres/caches)* ***"data reuse"***

```
for (loop=0; loop<10; loop++){
  for (i=0; i<n; i++){
    (beaucoup d'opérations)
    x[i]
    (beaucoup d'opérations)
  }
}
```


Stratégie : Localité des données

Localité temporelle

- Si un élément doit être utilisé plusieurs fois, faire en sorte que les utilisations soient rapprochées *(maximise l'utilisation des registres/caches)* ***"data reuse"***

```
for (loop=0; loop<10; loop++){
  for (i=0; i<n; i++){
    (beaucoup d'opérations)
    x[i]
    (beaucoup d'opérations)
  }
}
```

Relecture multiple de `x[i]` *dans la mémoire RAM …*

```
for (i=0; i<n; i++){
  s = x[i];
  for (loop=0; loop<10; loop++){
    (beaucoup d'opérations)
    s
    (beaucoup d'opérations)
  }
}
```

Une seule lecture de `x[i]` *et stockage dans* `s` *(pourrait rester en registre)*

Stratégie : Localité des données

Localité temporelle

- Si un élément doit être utilisé plusieurs fois, faire en sorte que les utilisations soient rapprochées *(maximise l'utilisation des registres/caches)* ***"data reuse"***

Localité spatiale

- Utilisation d'éléments qui sont proches en mémoire.
  Si la **ligne de cache** a déjà été chargée, les autres éléments sont … gratuits.

```
tot = 0;
for (j=0; j<n; j++){
  for (i=0; i<m; i++){
    tot += A[i][j];
  }
}
```


Stratégie : Localité des données

Localité temporelle

- Si un élément doit être utilisé plusieurs fois, faire en sorte que les utilisations soient rapprochées *(maximise l'utilisation des registres/caches)* ***"data reuse"***

Localité spatiale

- Utilisation d'éléments qui sont proches en mémoire.
  Si la **ligne de cache** a déjà été chargée, les autres éléments sont … gratuits.

```
tot = 0;
for (j=0; j<n; j++){
  for (i=0; i<m; i++){
    tot += A[i][j];
  }
}
```

On utilise des éléments non-consécutifs …

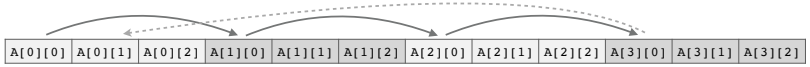


Tableau A : m x n

Stratégie : Localité des données

Localité temporelle

- Si un élément doit être utilisé plusieurs fois, faire en sorte que les utilisations soient rapprochées *(maximise l'utilisation des registres/caches)* ***"data reuse"***

Localité spatiale

- Utilisation d'éléments qui sont proches en mémoire.
  Si la **ligne de cache** a déjà été chargée, les autres éléments sont … gratuits.

```
tot = 0;
for (j=0; j<n; j++){
  for (i=0; i<m; i++){
    tot += A[i][j];
  }
}
```

On peut changer l'ordre des opérations …

```
tot = 0;
for (i=0; i<m; i++){
  for (j=0; j<n; j++){
    tot += A[i][j];
  }
}
```

Stratégie 1

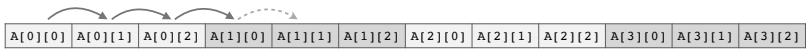


Tableau A : m x n

Stratégie : Localité des données

Localité temporelle

- Si un élément doit être utilisé plusieurs fois, faire en sorte que les utilisations soient rapprochées *(maximise l'utilisation des registres/caches)* ***"data reuse"***

Localité spatiale

- Utilisation d'éléments qui sont proches en mémoire.
  Si la **ligne de cache** a déjà été chargée, les autres éléments sont … gratuits.

```
tot = 0;
for (j=0; j<n; j++){
  for (i=0; i<m; i++){
    tot += A[i][j];
  }
}
```

On peut changer le stockage …

```
tot = 0;
for (j=0; j<n; j++){
  for (i=0; i<m; i++){
    tot += A[j][i];
  }
}
```

Stratégie 2

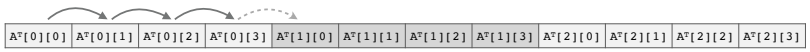


Tableau A^T : n x m

Stratégie : Compilateur et options d'optimisation

Le programme final qui sera exécuté est généré par un **compilateur**.
Tout compilateur propose des **options pour optimiser** les performances.
En terme d'espace mémoire utilisé et/ou vitesse d'exécution …

Compilateurs sur les ordinateurs de l'ENSTA

gcc	Version (Ubuntu 18.04) 7.5.0, 2017 — *Version par défaut*
gcc	Version (GCC) 8.1.0, 2018 — *Accessible après utilisation de la commande* **`useuma gcc`**
clang	Version 6.0.0-1ubuntu2

icpc	Compilateur propriétaire d'Intel, version 19.1.0
	Ajouter à la fin du fichier **`.profile`** *(créer le fichier si il n'existe pas):*
	`source envintel`
	`export PATH=/auto/appy/ensta/pack/parallel2020.4/bin/:$PATH`
	`export LD_LIBRARY_PATH=/auto/appy/ensta/pack/parallel2020.4/mkl/lib/`

Quelques options de compilation

Les compilateurs sont TRÈS malins !!!

`gcc|clang|icc`	**`--version`**	donne la version du compilateur
	`--help`	donne la liste des options de compilation
	`-O0` …	pas d'optimisation
	`-O1` …	premier niveau d'optimisation
	`-O2` …	deuxième niveau d'optimisation *(par défault)*
	`-O3` …	troisième niveau d'optimisation
	`-Ofast`	combine **`-O3`** avec d'autres options

Stratégie : Librairies — Exemple : BLAS

BLAS = Basic Linear Algebra Subprograms

- Les **opérations algébriques de base** avec des vecteurs et des matrices *(produit, addition, …)* sont fréquentes en programmation scientifique.
- Les opérations BLAS ont été classées en fonction du nombre d'opérations à virgule flottante (FLOP) nécessaires.

BLAS 1	Opérations scalaires, vectorielles et vecteur-vecteur	Ex: **x· y**	$\approx 2n$ *FLOP*
BLAS 2	Opérations matrice-vecteur	Ex: **A x**	$\approx 2n^2$ *FLOP*
BLAS 3	Opérations matrice-matrice	Ex: **A B**	$\approx 2n^3$ *FLOP*

- Des **routines optimisées** pour ces opérations ont été développées et standardisées pour permettre des performances élevées et des codes clairs.

Quelques librairies :
- LINPACK (librairie *open source*, années 70' et 80') *http://www.netlib.org/linpack/*
- LAPACK (librairie *open source*, depuis 1992) *http://www.netlib.org/lapack/*
- ATLAS (auto-optimisation sur chaque architecture) *http://math-atlas.sourceforge.net/*
- Intel MKL (librairie propriétaire) *https://software.intel.com/en-us/intel-mkl*

La plupart de ces librairies sont en FORTRAN, mais il est possible de les utiliser en C/C++ !

RÉSUMÉ de quelques stratégies

Localité des données

- Réutilisation des mêmes données dans des temps courts
- Utilisation de données stockées proches en mémoire
- Utilisation maximale de la ligne de cache

Utilisation des options du **compilateur**
Utilisation de **librairies**

Travailler sur l'ordre des opérations
Travailler sur la façon de stocker les données en mémoire *(en particulier les tableaux)*

Le HPC mono-coeur, c'est faire la psychanalyse du CPU et du compilateur …

Tester des idées et des stratégies de programmation pour les comprendre …