Complexité, intensité arithmétique, Roofline model, Numa, etc...

« Mais pourquoi donc est-ce que mon code ne va pas si vite que ça ? »

Roland Denis

CNRS / Institut Camille Jordan

21 mars 2019

Complexité algorithmique

Complexité des algorithmes

Lors de l'écriture de codes, il est important d'analyser leur complexité qui permet d'en estimer les performances et de le comparer à d'autres algorithmes.

Complexité

La complexité correspond à une comparaison asymptotique d'une mesure de performance (le temps d'exécution ou la mémoire utilisée) en fonction de paramètres caractériques de l'algo (la dimension de l'espace, la taille du vecteur, ...).

Complexité des algorithmes

Lors de l'écriture de codes, il est important d'analyser leur complexité qui permet d'en estimer les performances et de le comparer à d'autres algorithmes.

Complexité

La complexité correspond à une comparaison asymptotique d'une mesure de performance (le temps d'exécution ou la mémoire utilisée) en fonction de paramètres caractériques de l'algo (la dimension de l'espace, la taille du vecteur, ...).

Types de complexité

On parle de complexité :

- dans le meilleur des cas quand les données fournies en entrée sont les plus favorables à l'algorithme,
- dans le pire des cas quand les données fournies en entrée sont les plus défavorables à l'algorithme,
- moyenne pour la performance moyenne étant donnée une distribution sur l'ensemble des données en entrée possibles.

Exemples de complexités en temps

- accès au n-ième élément d'un vecteur : complexité constante, en $\mathcal{O}(1)$,
- recherche par dichotomie dans un vecteur de taille N : complexité logarithmique, en $\mathcal{O}(\log(N))$,
- calcul de la norme d'un vecteur de taille N : complexité linéaire, en $\mathcal{O}(N)$,
- tri d'un vecteur de taille N par la méthode du tri fusion : complexité linéarithmétique, en $\mathcal{O}(N \log(N))$,
- multiplication matrice $\times$ vecteur de taille N : complexité quadratique (polynomiale), en $\mathcal{O}(N^2)$,
- méthode du pivot de Gauss en dimension N : complexité cubique (polynomiale), en $\mathcal{O}(N^3)$,
- problème du rangement de N objets dans un sac à dos, par force brute : complexité exponentielle en $\mathcal{O}(N2^N)$.

Voir `fr.wikipedia.org/wiki/Analyse_de_la_complexité_des_algorithmes`

Examples

Produit scalaire

Pour le produit scalaire de deux vecteurs de tailles n, on réalise n produits et $n-1$ sommes, pour obtenir un scalaire.
On a donc une complexité $\mathcal{O}(n)$ en temps et $\mathcal{O}(1)$ en espace.

Examples

Produit scalaire

Pour le produit scalaire de deux vecteurs de tailles n, on réalise n produits et $n-1$ sommes, pour obtenir un scalaire.
On a donc une complexité $\mathcal{O}(n)$ en temps et $\mathcal{O}(1)$ en espace.

Produit matrice $\times$ vecteur

Pour le produit d'une matrice de taille $m \times n$ par un vecteur de taille n, on réalise n produits et $n-1$ sommes pour chaque élément du vecteur final de taille m.
On a donc une complexité $\mathcal{O}(n \times m)$ en temps et $\mathcal{O}(m)$ en espace.

Examples

Produit scalaire

Pour le produit scalaire de deux vecteurs de tailles n, on réalise n produits et $n-1$ sommes, pour obtenir un scalaire.
On a donc une complexité $\mathcal{O}(n)$ en temps et $\mathcal{O}(1)$ en espace.

Produit matrice $\times$ vecteur

Pour le produit d'une matrice de taille $m \times n$ par un vecteur de taille n, on réalise n produits et $n-1$ sommes pour chaque élément du vecteur final de taille m.
On a donc une complexité $\mathcal{O}(n \times m)$ en temps et $\mathcal{O}(m)$ en espace.

Produit matrice $\times$ matrice

Pour le produit de deux matrices de taille $m \times n$ et $n \times p$, on réalise n produits + n sommes pour chaque élément de la matrice finale de taille $m \times p$.
On a donc une complexité $\mathcal{O}(n \times m \times p)$ en temps et $\mathcal{O}(m \times p)$ en espace.
Pour le cas où $m = p = n$, on a $\mathcal{O}(n^3)$ en temps et $\mathcal{O}(n^2)$ en espace.
C'est l'approche naïve, la meilleure complexité à ce jour étant en $\mathcal{O}(n^{2.3728639})$ (see https://en.wikipedia.org/wiki/Matrix_multiplication).

Parallélisme en mémoire partagée

Le parallélisme en mémoire partagée expliqué en 2 minutes.

Machines actuelles :

- $1, 2, 4...$ processeurs,
- chaque processeur a n cœurs, avec $n = 2, 4, 8, 10, 12, \ldots$.

Bientôt : une centaine de cœurs par machine.

Le parallélisme en mémoire partagée expliqué en 2 minutes.

Machines actuelles :

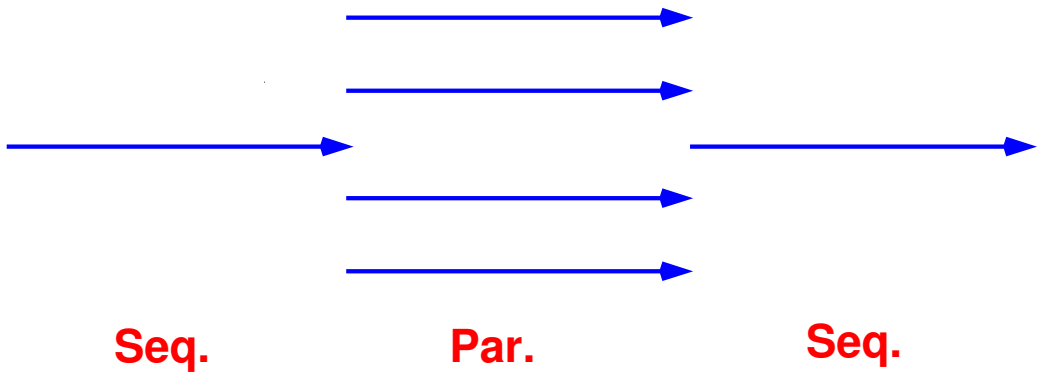
- $1, 2, 4...$ processeurs,
- chaque processeur a n cœurs, avec $n = 2, 4, 8, 10, 12, \ldots$.

Bientôt : une centaine de cœurs par machine.

Parallélisme obligatoire !

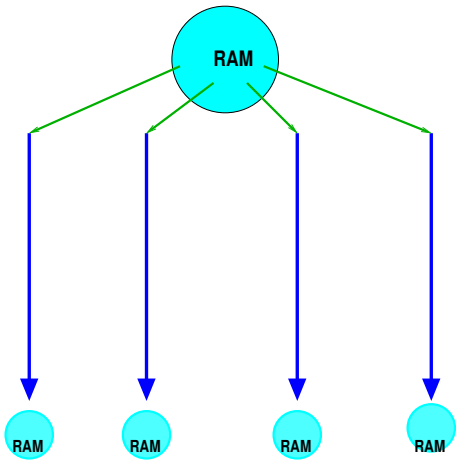
Le parallélisme en mémoire partagée expliqué en 2 minutes.

Processus légers (threads) :



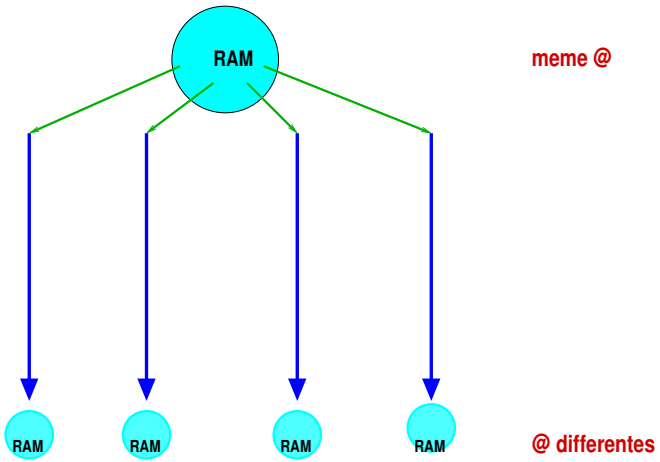
Le parallélisme en mémoire partagée expliqué en 2 minutes.

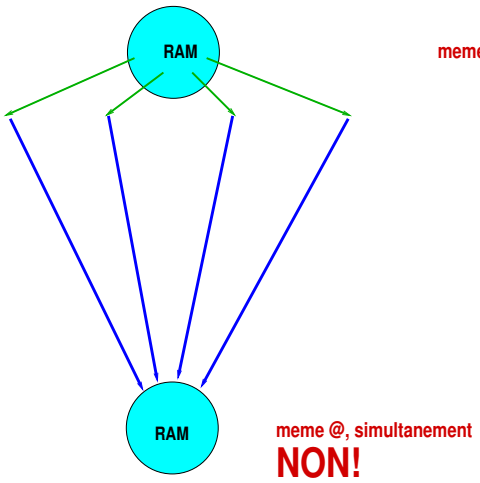
Partage de la mémoire (sans sécurité).



Le parallélisme en mémoire partagée expliqué en 2 minutes.

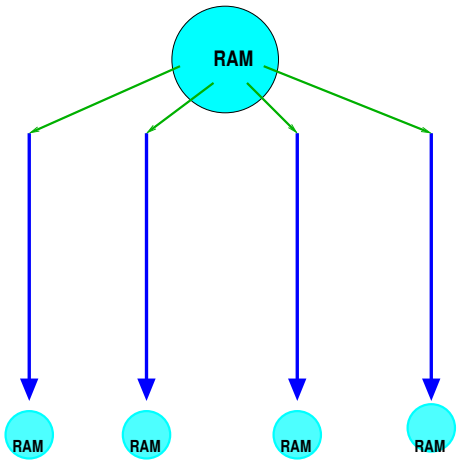
Partage de la mémoire (sans sécurité).

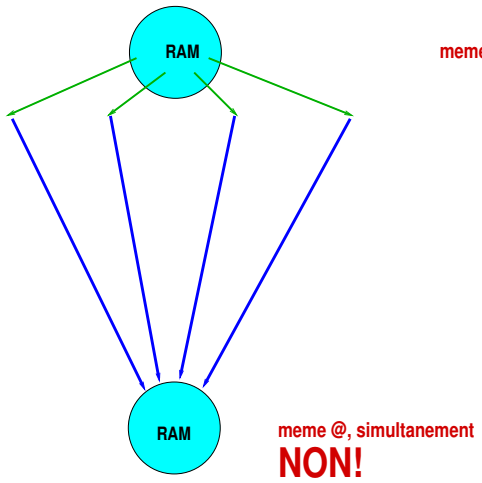




Le parallélisme en mémoire partagée expliqué en 2 minutes.

Partage de la mémoire (sans sécurité).

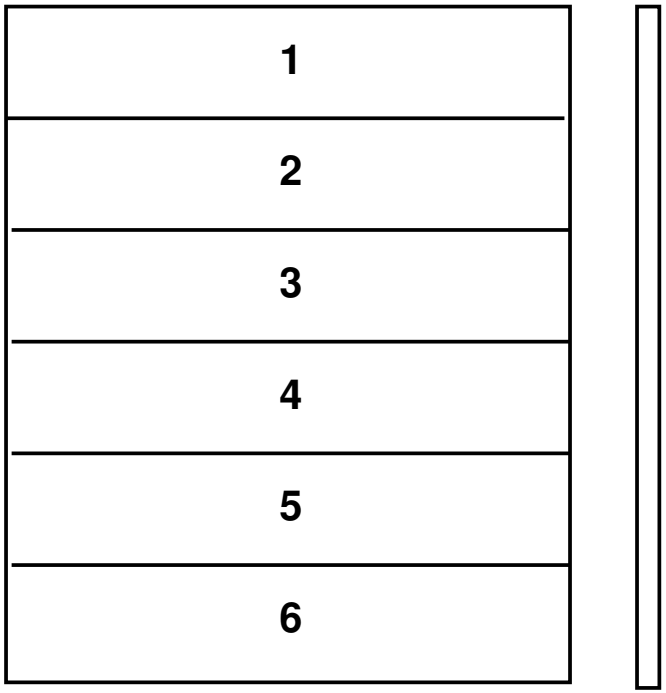




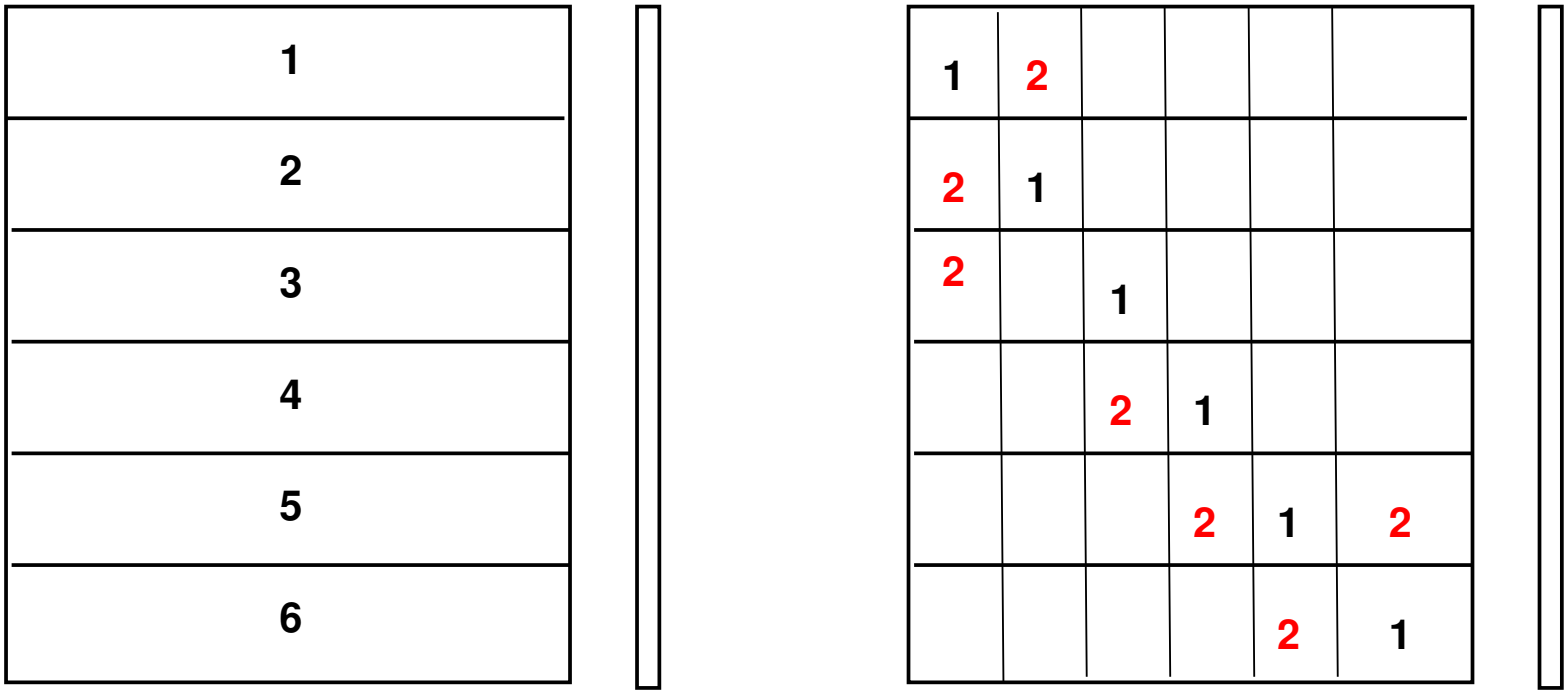
Standard : Open MP, mais aussi TBB (C++).

Le parallélisme en mémoire partagée expliqué en 2 minutes.

Exemple : le produit matrice x vecteur.

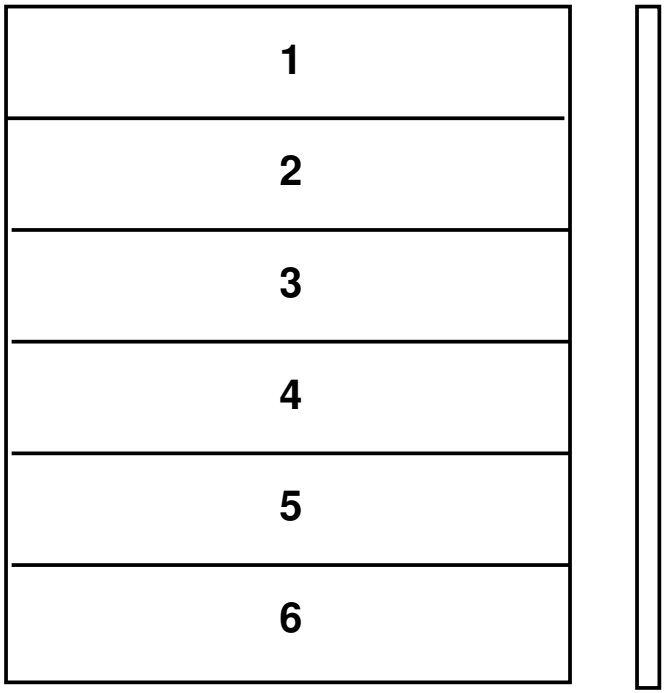


Exemple : le produit matrice x vecteur.

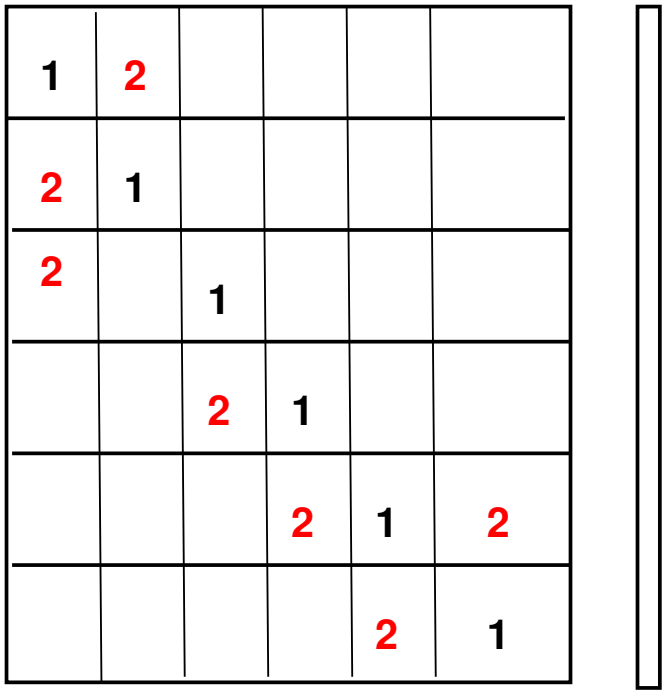


Le parallélisme en mémoire partagée expliqué en 2 minutes.

Exemple : le produit matrice x vecteur.



OK !



NON !

Performances théoriques

Performances d'une machine : rêves et réalité

On compte les flops : $\times, +, -, /$.
Attention aux divisions (plus lentes) !

Performances d'une machine : rêves et réalité

On compte les flops : $\times, +, -, /$.
Attention aux divisions (plus lentes) !

Exemple :
machine Intel « Sandy bridge » 2 processeurs, avec chacun 8 cœurs, tournant à 2.6 Ghz ($2.6\ 10^9$ cycles/seconde).

Performances d'une machine : rêves et réalité

On compte les flops : $\times, +, -, /$.
Attention aux divisions (plus lentes) !

Exemple :
machine Intel « Sandy bridge » 2 processeurs, avec chacun 8 cœurs, tournant à 2.6 Ghz ($2.6\ 10^9$ cycles/seconde).

Aspect SIMD (Single Instruction, Multiple Data) :

Instructions AVX :

En un cycle d'horloge, faire :
$y_i = a_i\ x_i\ +\ b_i,\ i = 1, \ldots, 4$ (8 flops).

Performances d'une machine : rêves et réalité

On compte les flops : $\times, +, -, /$.
Attention aux divisions (plus lentes) !

Exemple :
machine Intel « Sandy bridge » 2 processeurs, avec chacun 8 cœurs, tournant à 2.6 Ghz ($2.6\ 10^9$ cycles/seconde).

Aspect SIMD (Single Instruction, Multiple Data) :

Instructions AVX :

En un cycle d'horloge, faire :
$y_i = a_i\ x_i\ +\ b_i,\ i = 1, \ldots, 4$ (8 flops).

La performance *peak* est donc de :

2×8

Performances d'une machine : rêves et réalité

On compte les flops : $\times, +, -, /$.
Attention aux divisions (plus lentes) !

Exemple :
machine Intel « Sandy bridge » 2 processeurs, avec chacun 8 cœurs, tournant à 2.6 Ghz ($2.6\ 10^9$ cycles/seconde).

Aspect SIMD (Single Instruction, Multiple Data) :

Instructions AVX :

En un cycle d'horloge, faire :
$y_i = a_i\ x_i\ +\ b_i,\ i = 1, \ldots, 4$ (8 flops).

La performance *peak* est donc de :

$2 \times 8 \times 2.6\ 10^9$

Performances d'une machine : rêves et réalité

On compte les flops : $\times, +, -, /$.
Attention aux divisions (plus lentes) !

Exemple :
machine Intel « Sandy bridge » 2 processeurs, avec chacun 8 cœurs, tournant à 2.6 Ghz ($2.6\ 10^9$ cycles/seconde).

Aspect SIMD (Single Instruction, Multiple Data) :

Instructions AVX :

En un cycle d'horloge, faire :
$y_i = a_i\ x_i\ +\ b_i,\ i = 1, \ldots, 4$ (8 flops).

La performance *peak* est donc de :

$2 \times 8 \times 2.6\ 10^9 \times 8$

Performances d'une machine : rêves et réalité

On compte les flops : $\times, +, -, /$.
Attention aux divisions (plus lentes) !

Exemple :
machine Intel « Sandy bridge » 2 processeurs, avec chacun 8 cœurs, tournant à 2.6 Ghz ($2.6\ 10^9$ cycles/seconde).

Aspect SIMD (Single Instruction, Multiple Data) :

Instructions AVX :

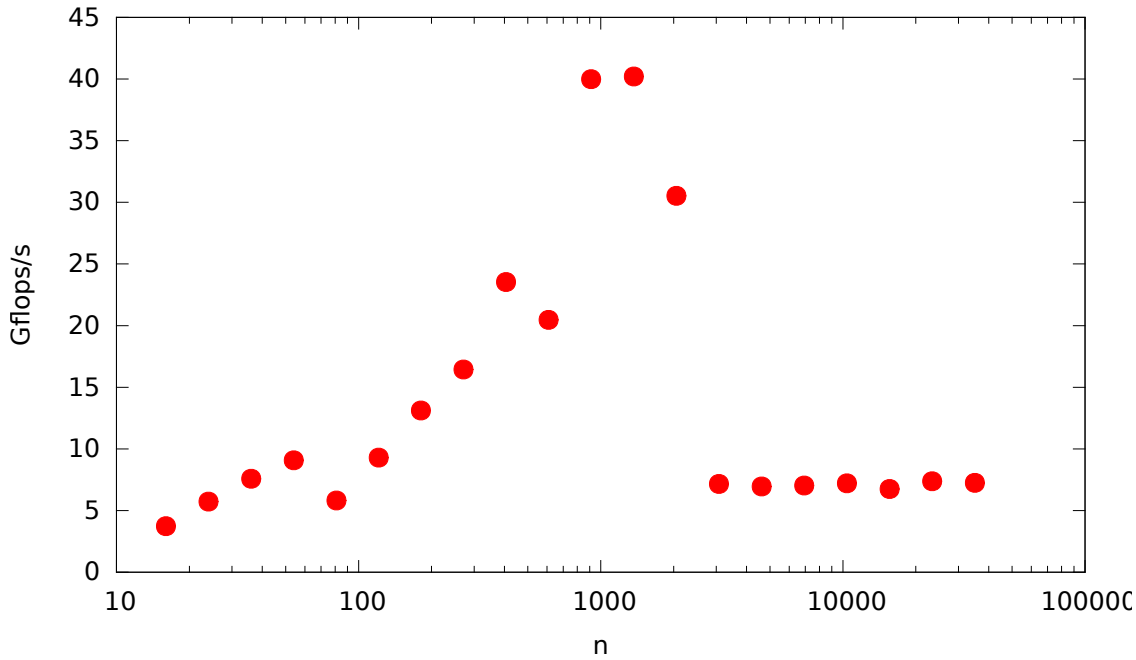
En un cycle d'horloge, faire :
$y_i = a_i\ x_i\ +\ b_i,\ i = 1, \ldots, 4$ (8 flops).

La performance *peak* est donc de :

$2 \times 8 \times 2.6\ 10^9 \times 8 = 332$ Gflops/seconde.

Essai : produit matrice x vecteur

Blas Intel, parallèle.



Nombre d'opérations :

- produit scalaire de 2 vecteurs de taille n : $2n$ flops.
- produit matrice$(n \times n)$ x vecteur n : n produits scalaires $=> 2n^2$ flops.

Intensité arithmétique

Bande passante

Pourquoi je n'obtiens pas les performances attendues ?

Un calcul ne peut être effectué que si :

1. les opérandes sont disponibles,
2. on peut écrire le résultat.

Bande passante

Pourquoi je n'obtiens pas les performances attendues ?

Un calcul ne peut être effectué que si :

1. les opérandes sont disponibles,
2. on peut écrire le résultat.

$=>$ la bande passante entre (processeur / cache) et la mémoire limite les performances.

Intensité arithmétique

L'intensité arithmétique prend en compte ce facteur limitant et permet de mieux comprendre le comportement d'un noyau de calcul par rapport à une architecture cible :

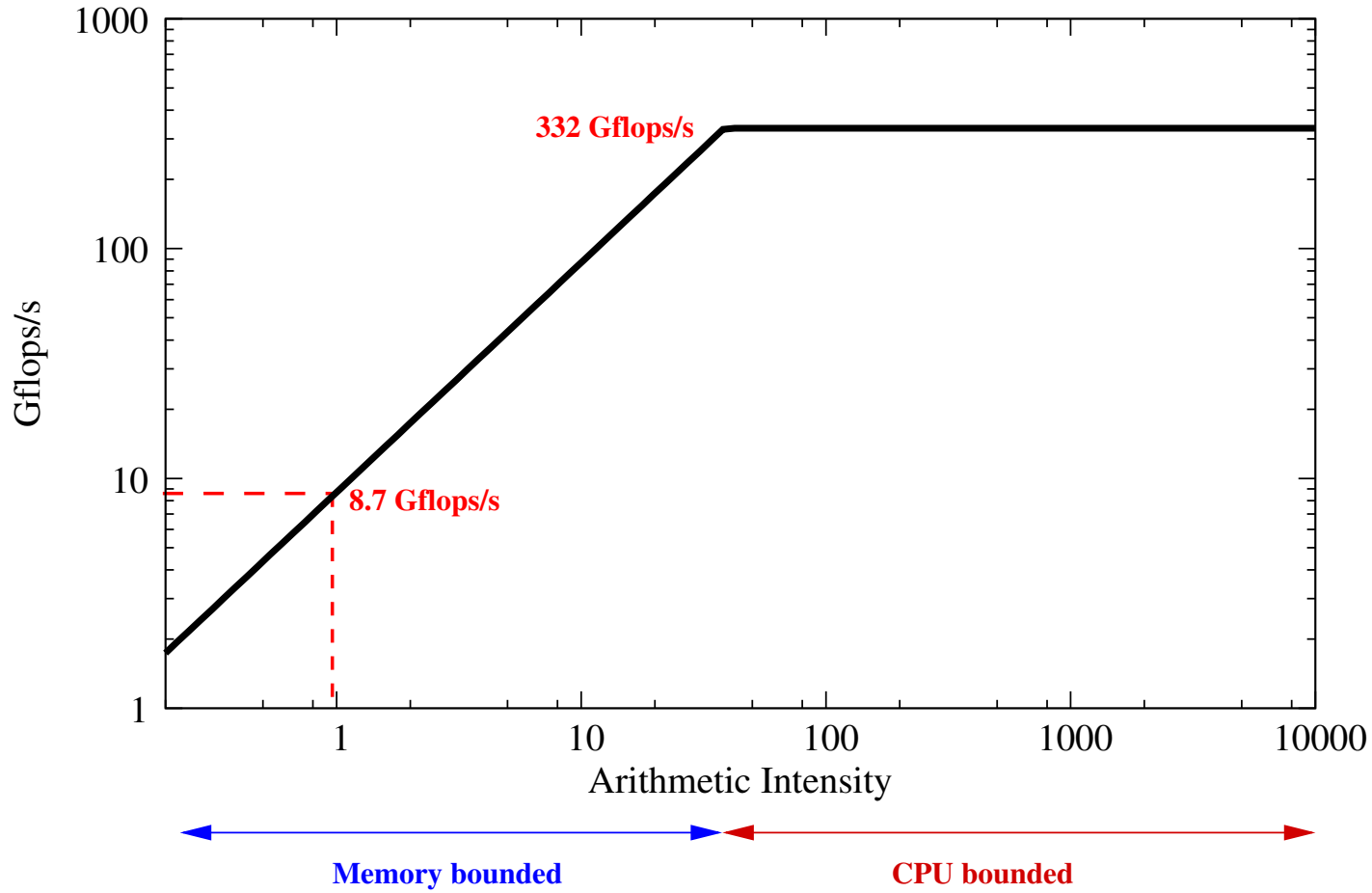
$$I_a = \frac{\text{nombre d'opérations}}{\text{quantité de mémoire échangée}}.$$

On peut donc l'augmenter en favorisant le réemploi des données, via par exemple une utilisation efficace des différents niveaux de cache du processeur.

Le « Roofline Model »

Performance [GFlops/sec] atteignable (*Peak performance*)

$$\min\left(\text{Performance crête}, \text{Bande passante mémoire} \cdot I_a\right)$$

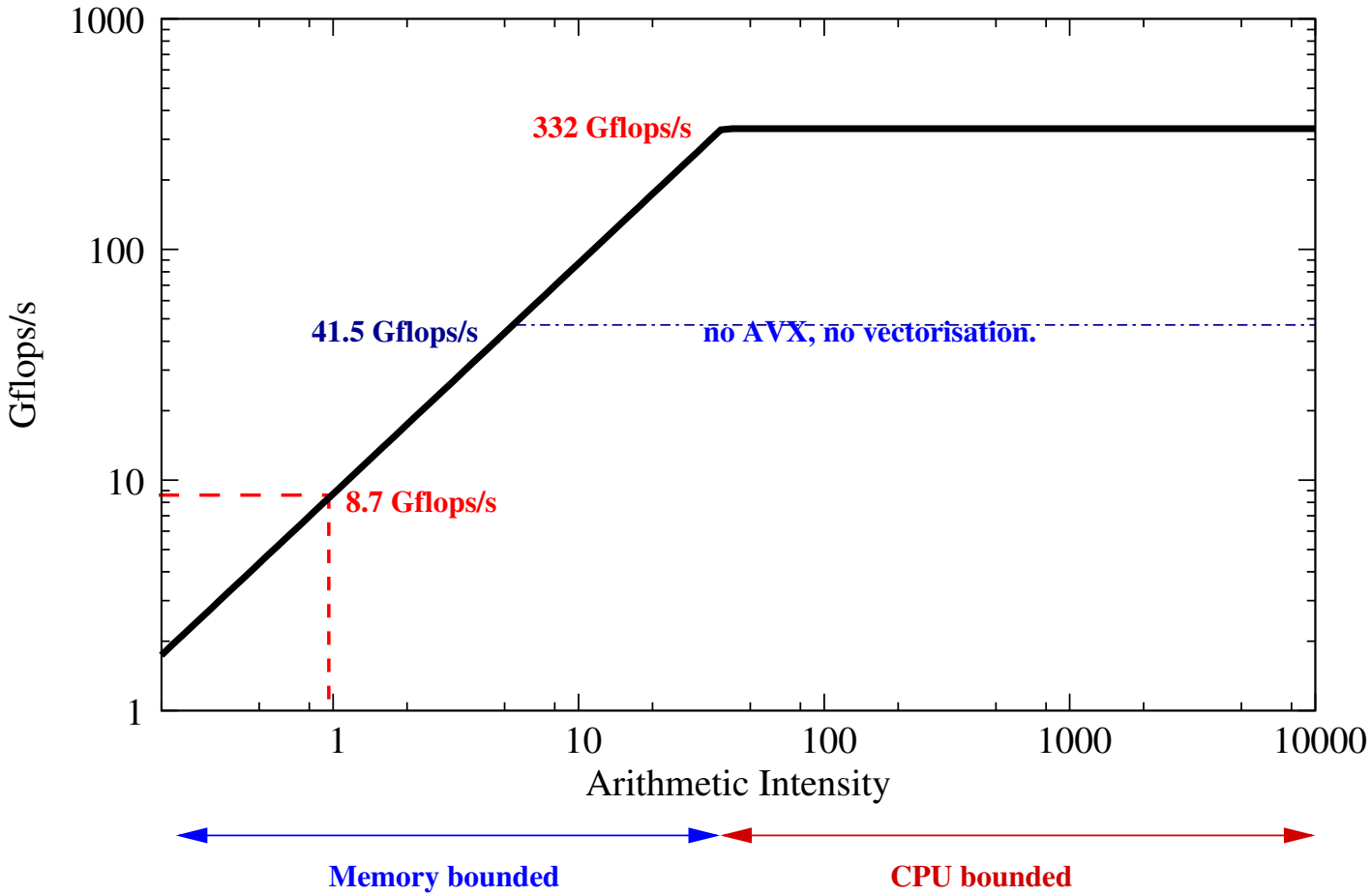


Williams S. et al: *Roofline : An Insightful Visual Performance Model for Multicore Architectures* – Commun. ACM, 2009.

Le « Roofline Model »

Performance [GFlops/sec] atteignable (*Peak performance*)

$$\min\left(\text{Performance crête}, \text{Bande passante mémoire} \cdot I_a\right)$$

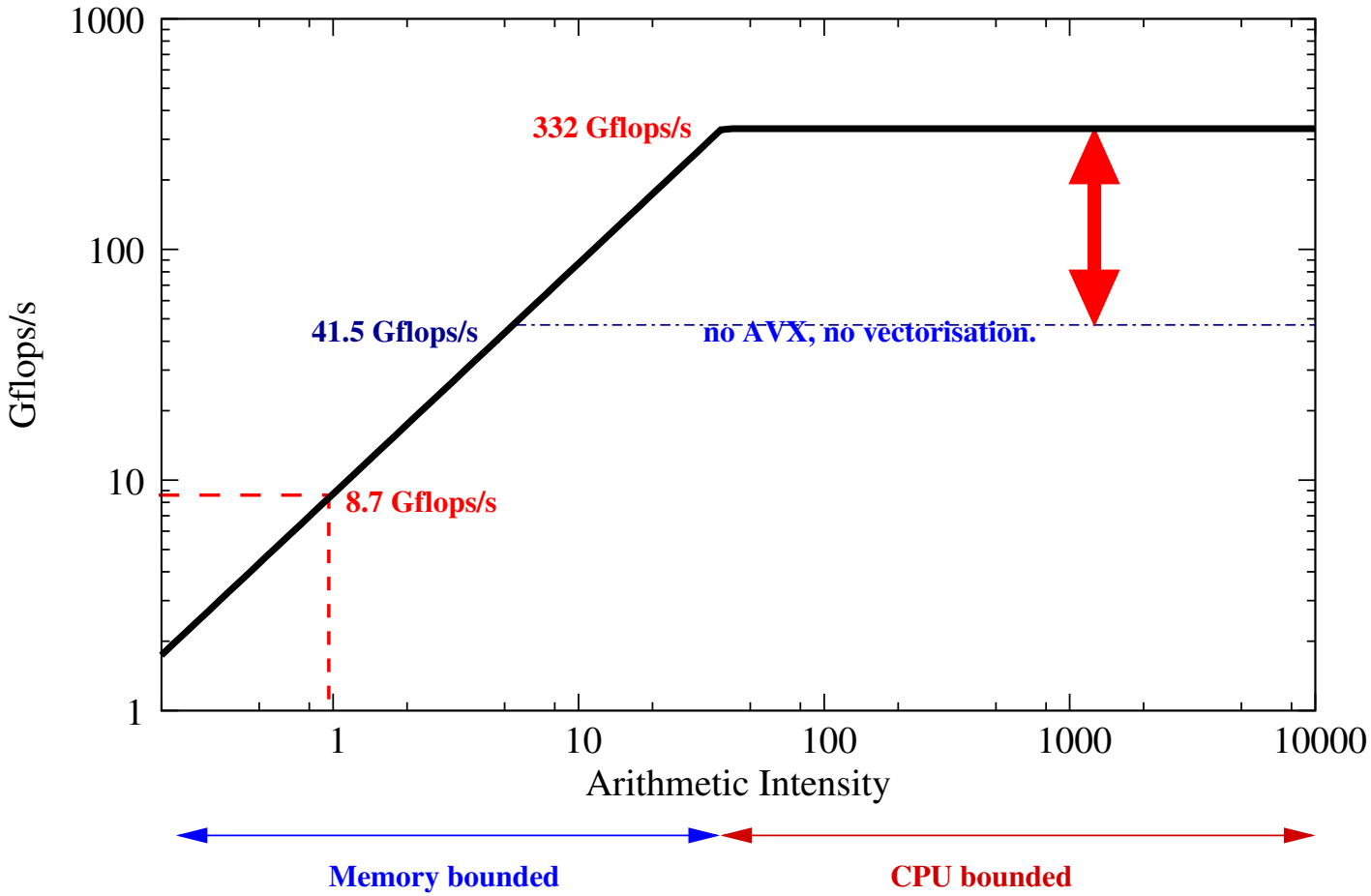


Williams S. et al: *Roofline : An Insightful Visual Performance Model for Multicore Architectures* – Commun. ACM, 2009.

Le « Roofline Model »

Performance [GFlops/sec] atteignable (*Peak performance*)

$$\min\left(\text{Performance crête}, \text{Bande passante mémoire} \cdot I_a\right)$$



Williams S. et al: *Roofline : An Insightful Visual Performance Model for Multicore Architectures* – Commun. ACM, 2009.

Intensité arithmétiques : quelques exemples

Produit scalaire

Pour le produit scalaire de deux vecteurs de tailles n, on réalise n produits et $n-1$ sommes, en lisant/écrivant $2n+1$ valeurs.
On a donc $I_a = \mathcal{O}(1)$.

Intensité arithmétiques : quelques exemples

Produit scalaire

Pour le produit scalaire de deux vecteurs de tailles n, on réalise n produits et $n-1$ sommes, en lisant/écrivant $2n+1$ valeurs.
On a donc $I_a = \mathcal{O}(1)$.

Produit matrice $\times$ vecteur

Pour le produit d'une matrice de taille $n \times n$ par un vecteur de taille n, on réalise n produits et $n-1$ sommes pour chaque élément du vecteur final de taille n, en lisant/écrivant n^2+2n valeurs.

On a donc une intensité arithmétique de $I_a = \frac{n(2n-1)}{n^2+2n} = \mathcal{O}(1)$.

Intensité arithmétiques : quelques exemples

Produit scalaire

Pour le produit scalaire de deux vecteurs de tailles n, on réalise n produits et $n-1$ sommes, en lisant/écrivant $2n+1$ valeurs.
On a donc $I_a = \mathcal{O}(1)$.

Produit matrice $\times$ vecteur

Pour le produit d'une matrice de taille $n \times n$ par un vecteur de taille n, on réalise n produits et $n-1$ sommes pour chaque élément du vecteur final de taille n, en lisant/écrivant n^2+2n valeurs.

On a donc une intensité arithmétique de $I_a = \frac{n(2n-1)}{n^2+2n} = \mathcal{O}(1)$.

Produit matrice $\times$ matrice

Pour le produit de deux matrices de taille $n \times n$, on réalise n produits $+$ $n-1$ sommes pour chaque élément de la matrice finale de taille $n \times n$, en lisant/écrivant $3n^2$ valeurs.

On a donc une intensité arithmétique de $I_a = \frac{n^2(2n-1)}{3n^2} = \mathcal{O}(n)$.

Intensité arithmétiques : quelques exemples

Laplacien 3D en différences finies

Appliquer le stencil du Laplacien à 7 points en dimension 3 :

$$\Delta u(x,y,z) \approx \frac{u(x-h,y,z)+u(x+h,y,z)+\cdots-6u(x,y,z)}{h^2}$$

d'où une intensité arithmétique en $I_a = 8/8 = 1$.

Intensité arithmétiques : quelques exemples

Laplacien 3D en différences finies

Appliquer le stencil du Laplacien à 7 points en dimension 3 :

$$\Delta u(x,y,z) \approx \frac{u(x-h,y,z)+u(x+h,y,z)+\cdots-6u(x,y,z)}{h^2}$$

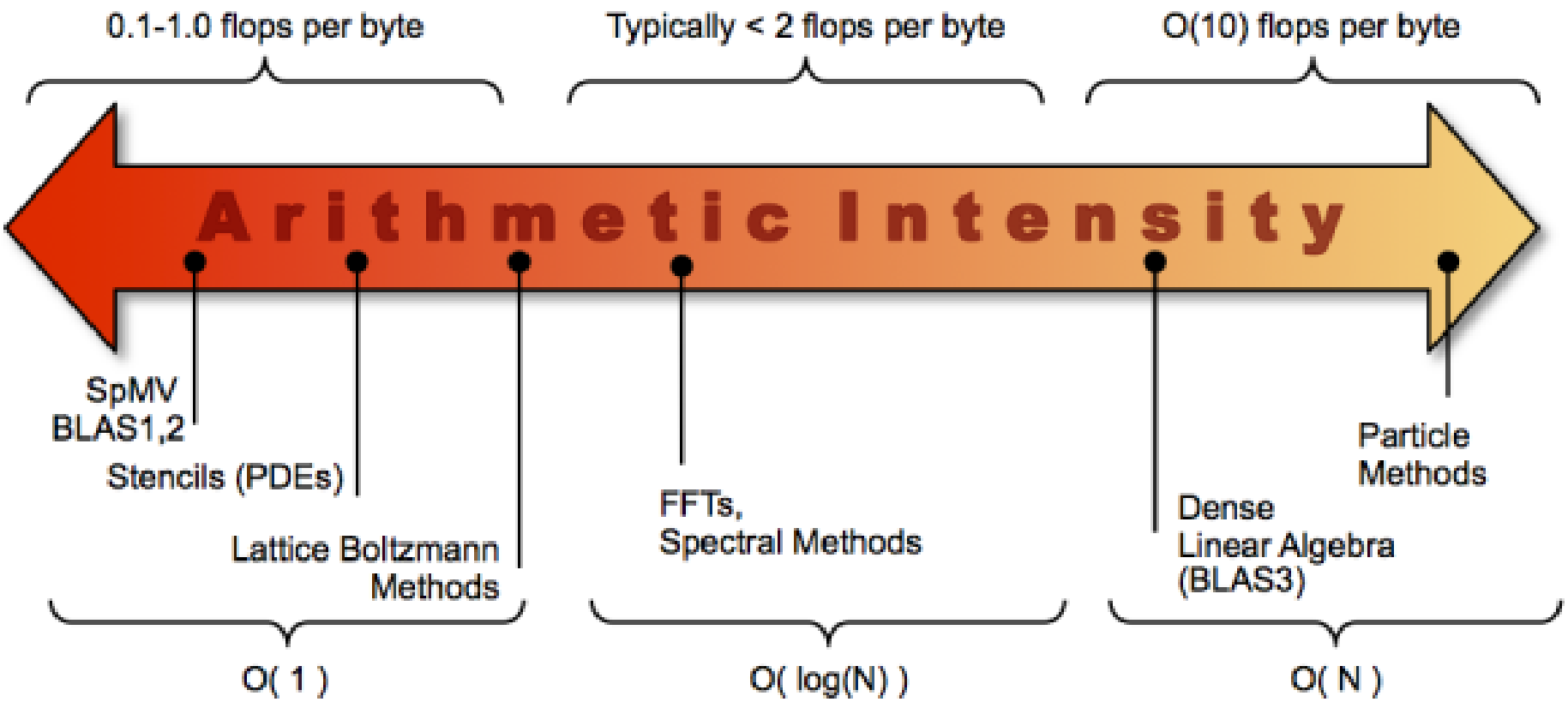
d'où une intensité arithmétique en $I_a = 8/8 = 1$.

Cache mémoire

Attention : un mauvaise utilisation du cache mémoire peut dégrader l'intensité arithmétique (matrice $\times$ matrice en $\mathcal{O}(1)$).

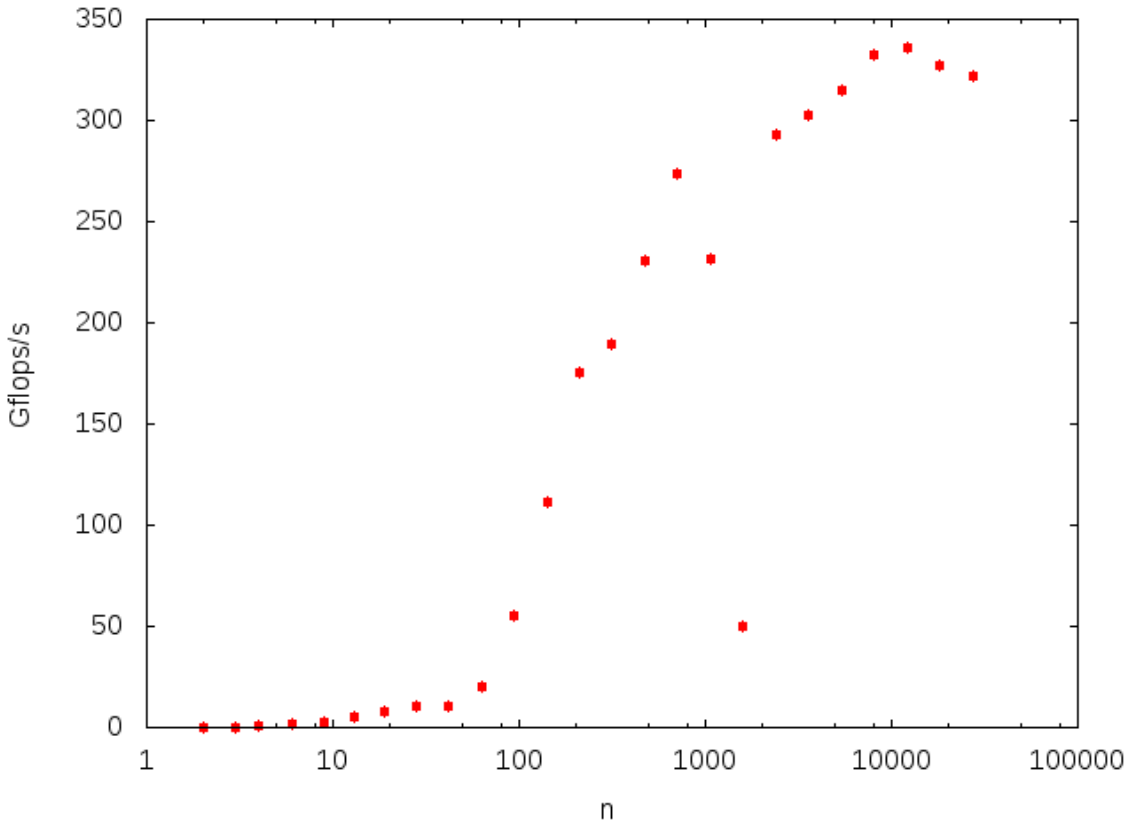
Intensité arithmétiques : quelques exemples

Quelques méthodes numériques courantes



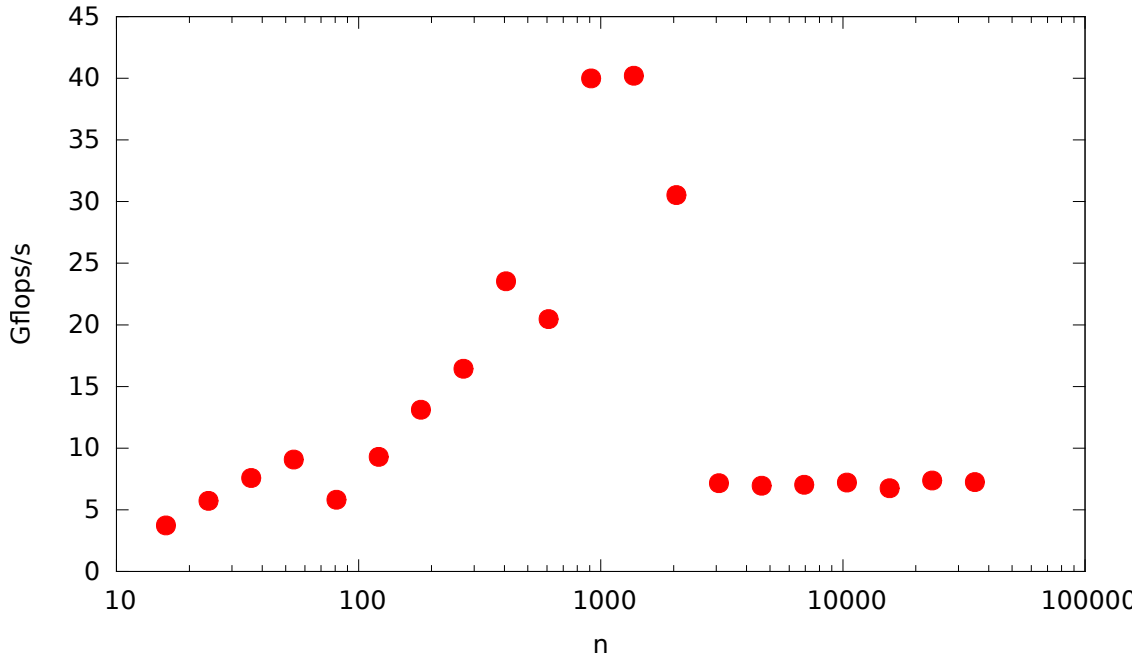
Intensité arithmétique : experiences

Matrix $\times$ Matrix product (DGEMM, Intel mkl parallel version) :



Intensité arithmétique : experiences

Produit Matrice $\times$ Vecteur (DGEMV, Intel mkl parallel version) :



Intensité arithmétique : expériences

Appliquer le stencil du laplacien en 3-d (7 points), stocké en matrice CSR :

Intensité arithmétique : expériences

Appliquer le stencil du laplacien en 3-d (7 points), stocké en matrice CSR :

	0	1	2	3	4
0	2.0		3.5		6.7
1		8.2		9.2	
2		1.1	2.8		
3	3.0		1.5	4.5	
4		2.5		8.9	

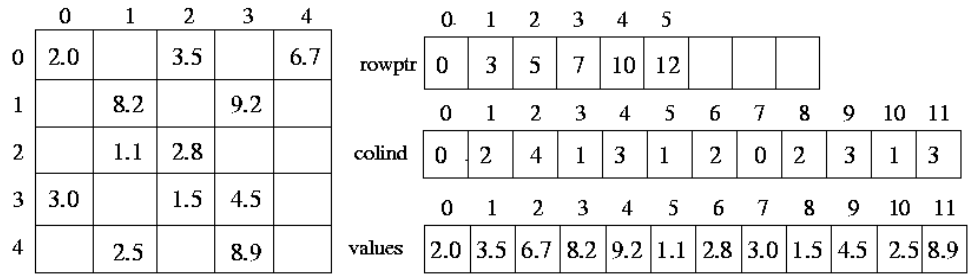
	0	1	2	3	4	5
rowptr	0	3	5	7	10	12

	0	1	2	3	4	5	6	7	8	9	10	11
colind	0	2	4	1	3	1	2	0	2	3	1	3

	0	1	2	3	4	5	6	7	8	9	10	11
values	2.0	3.5	6.7	8.2	9.2	1.1	2.8	3.0	1.5	4.5	2.5	8.9

Intensité arithmétique : expériences

Appliquer le stencil du laplacien en 3-d (7 points), stocké en matrice CSR :



	0	1	2	3	4
0	2.0		3.5		6.7
1		8.2		9.2	
2		1.1	2.8		
3	3.0		1.5	4.5	
4		2.5		8.9	

	0	1	2	3	4	5
rowptr	0	3	5	7	10	12

	0	1	2	3	4	5	6	7	8	9	10	11
colind	0	2	4	1	3	1	2	0	2	3	1	3

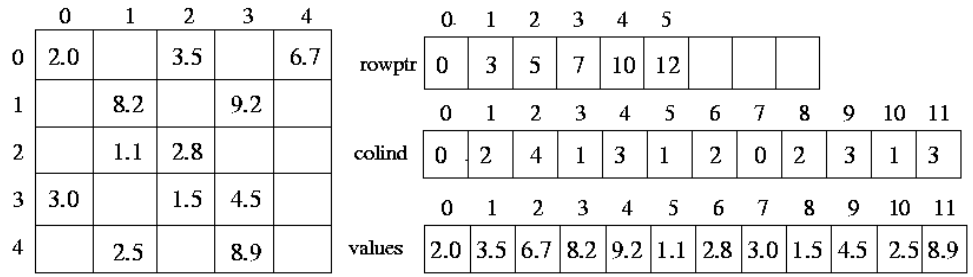
	0	1	2	3	4	5	6	7	8	9	10	11
values	2.0	3.5	6.7	8.2	9.2	1.1	2.8	3.0	1.5	4.5	2.5	8.9

On fait l'hypothèse (raisonnable) que :
`sizeof(double) = 2 sizeof(int)`.

- Algorithme : Mémoire $= 36/2$ doubles ; Flops $= 13 \Longrightarrow I_a \simeq 0.722$.

Intensité arithmétique : expériences

Appliquer le stencil du laplacien en 3-d (7 points), stocké en matrice CSR :



	0	1	2	3	4
0	2.0		3.5		6.7
1		8.2		9.2	
2		1.1	2.8		
3	3.0		1.5	4.5	
4		2.5		8.9	

	0	1	2	3	4	5
rowptr	0	3	5	7	10	12

	0	1	2	3	4	5	6	7	8	9	10	11
colind	0	2	4	1	3	1	2	0	2	3	1	3

	0	1	2	3	4	5	6	7	8	9	10	11
values	2.0	3.5	6.7	8.2	9.2	1.1	2.8	3.0	1.5	4.5	2.5	8.9

On fait l'hypothèse (raisonnable) que :
`sizeof(double) = 2 sizeof(int)`.

- Algorithme : Mémoire = 36/2 doubles ; Flops = 13 $\Longrightarrow$ $I_a \simeq 0.722$.
- Machine : Mémoire : 8.73 Giga doubles/s

  $=>$ Atteignable $= 0.722 \times 8.73 = 6.30$ Gflops.

Intensité arithmétique : expériences

Appliquer le stencil du laplacien en 3-d (7 points), stocké en matrice CSR :

	0	1	2	3	4
0	2.0		3.5		6.7
1		8.2		9.2	
2		1.1	2.8		
3	3.0		1.5	4.5	
4		2.5		8.9	

	0	1	2	3	4	5
rowptr	0	3	5	7	10	12

	0	1	2	3	4	5	6	7	8	9	10	11
colind	0	2	4	1	3	1	2	0	2	3	1	3

	0	1	2	3	4	5	6	7	8	9	10	11
values	2.0	3.5	6.7	8.2	9.2	1.1	2.8	3.0	1.5	4.5	2.5	8.9

On fait l'hypothèse (raisonnable) que :
`sizeof(double) = 2 sizeof(int)`.

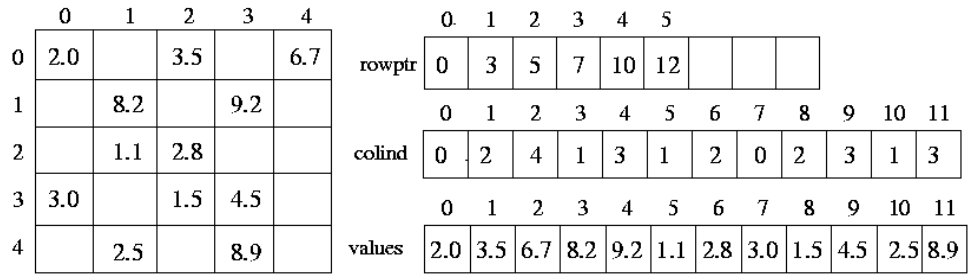
- Algorithme : Mémoire $= 36/2$ doubles ; Flops $= 13 \Longrightarrow I_a \simeq 0.722$.
- Machine : Mémoire : 8.73 Giga doubles/s

$$=> \text{Atteignable} = 0.722 \times 8.73 = 6.30 \text{ Gflops}.$$

$$\text{Mesuré} = 6.42 \text{ Gflops}.$$

Intensité arithmétique : expériences

Appliquer le stencil du laplacien en 3-d (7 points), stocké en matrice CSR :



	0	1	2	3	4
0	2.0		3.5		6.7
1		8.2		9.2	
2		1.1	2.8		
3	3.0		1.5	4.5	
4		2.5		8.9	

	0	1	2	3	4	5
rowptr	0	3	5	7	10	12

	0	1	2	3	4	5	6	7	8	9	10	11
colind	0	2	4	1	3	1	2	0	2	3	1	3

	0	1	2	3	4	5	6	7	8	9	10	11
values	2.0	3.5	6.7	8.2	9.2	1.1	2.8	3.0	1.5	4.5	2.5	8.9

On fait l'hypothèse (raisonnable) que :
`sizeof(double) = 2 sizeof(int)`.

- Algorithme : Mémoire $= 36/2$ doubles ; Flops $= 13 \Longrightarrow I_a \simeq 0.722$.
- Machine : Mémoire : 8.73 Giga doubles/s

$$=> \text{Atteignable} = 0.722 \times 8.73 = 6.30 \text{ Gflops}.$$

$$\text{Mesuré} = 6.42 \text{ Gflops}.$$

Note : borné à $0.87 \times 8.73 \simeq 7.6$ Gflops/s quelque soit la structure de données.

Comment connaître la bande passante d'une machine ?

Sur le web

En allant sur le site du constructeur : ark.intel.com, www.amd.com, ...

En la mesurant

- Divers programmes, dont `stream` https://www.cs.virginia.edu/stream/
- Programme en langage bas niveau (C/C++, ...).

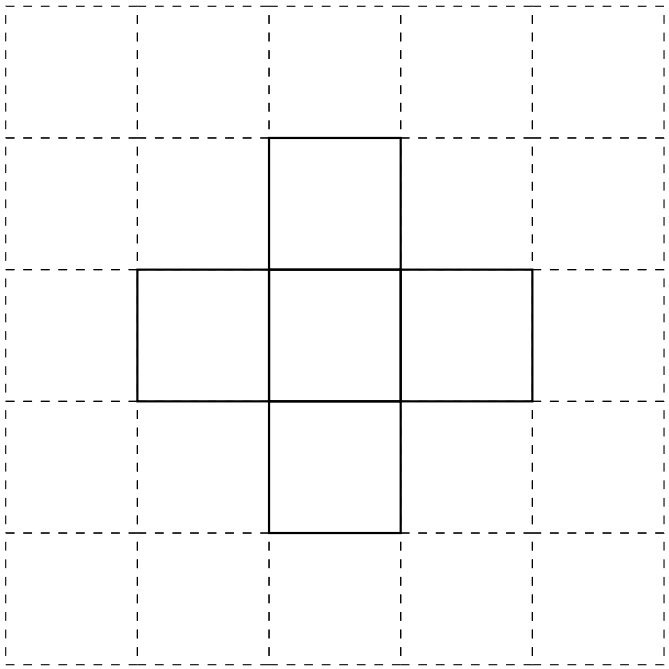
NB : augmenter le plus possible la taille des tableaux !
NB : la bande passante n'est parfois pas atteignable sur un seul cœur $\Rightarrow$ parallélisme.

Est-ce sans espoir ?

On peut (il faut !) favoriser la réutilisation des données stockées dans le cache L1.

Est-ce sans espoir ?

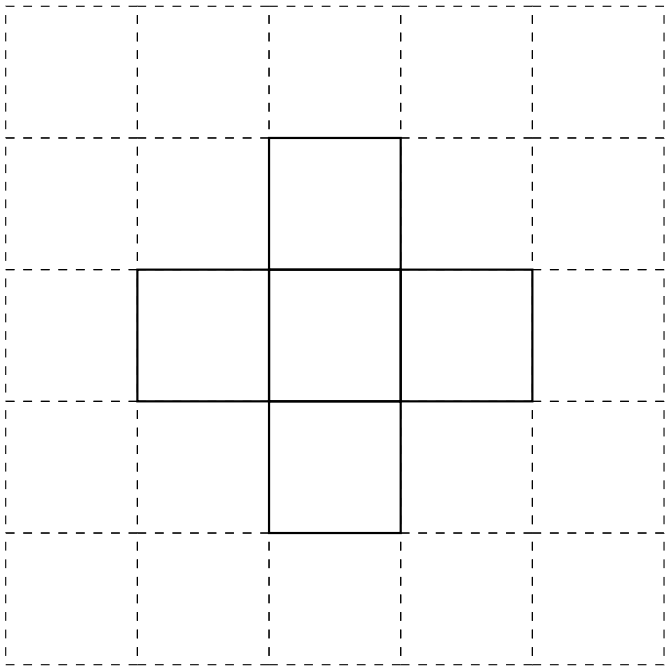
On peut (il faut !) favoriser la réutilisation des données stockées dans le cache L1.

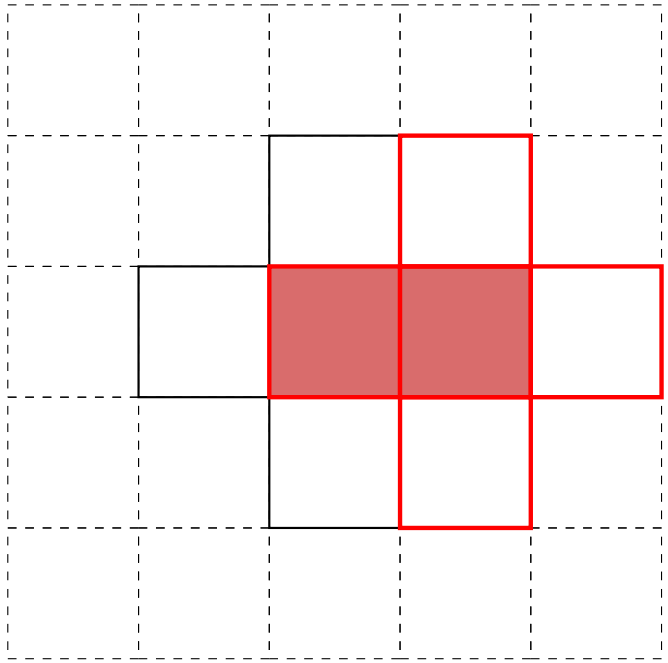


$$u_{ij} = 0.25\ (u_{i+1,j} + u_{i-1,j} - 4u_{ij} + u_{i,j+1} + u_{i,j-1}).$$

Est-ce sans espoir ?

On peut (il faut !) favoriser la réutilisation des données stockées dans le cache L1.





$$u_{ij} = 0.25\ (u_{i+1,j} + u_{i-1,j} - 4u_{ij} + u_{i,j+1} + u_{i,j-1}).$$

À éviter

- produits scalaires,
- matrices creuses (préconditionnement LU incomplet),
- combinaisons linéaires de grands vecteurs,
- méthodes avec un parallélisme limité,
- ... méthodes avec une faible intensité.

À éviter

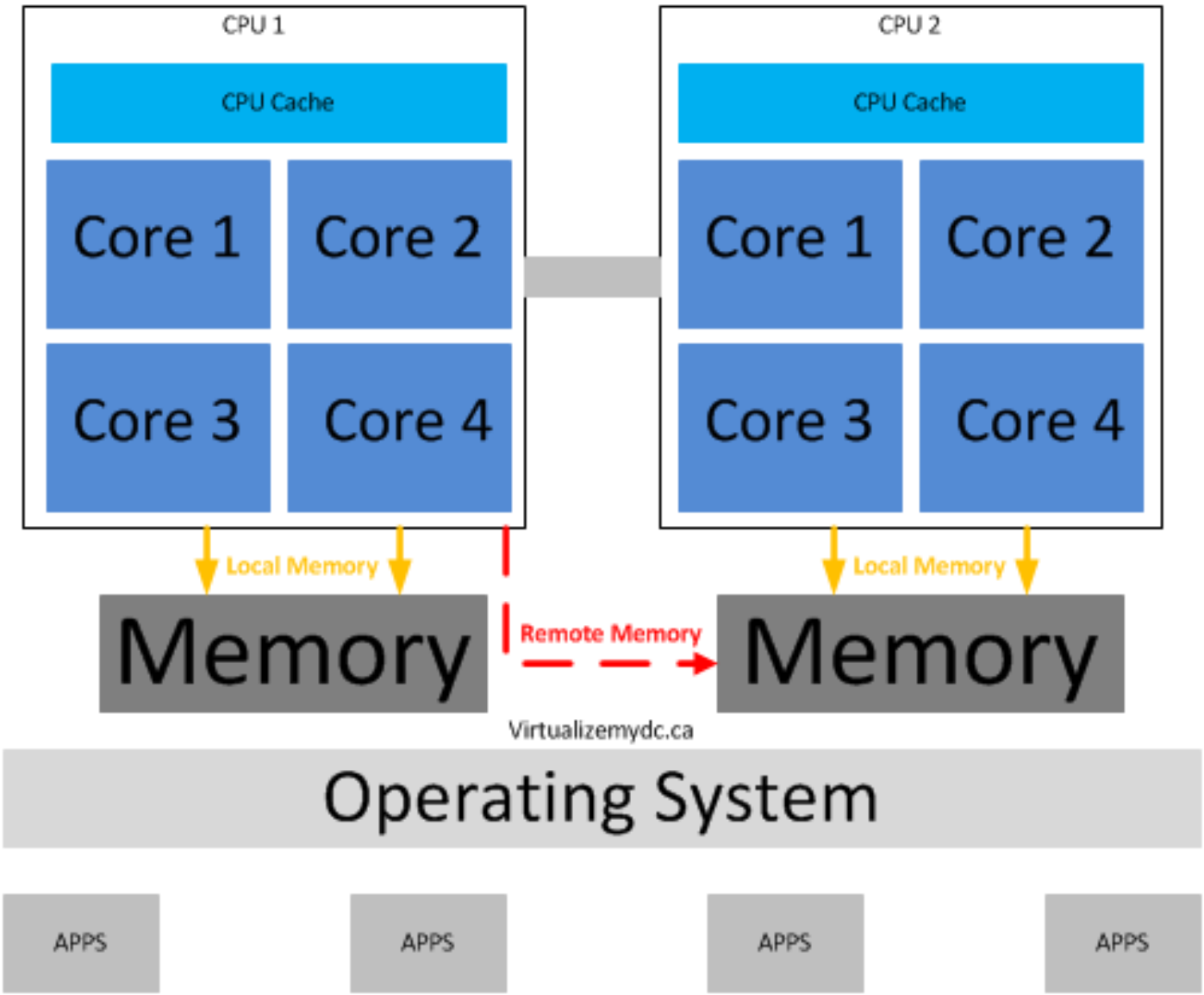
- produits scalaires,
- matrices creuses (préconditionnement LU incomplet),
- combinaisons linéaires de grands vecteurs,
- méthodes avec un parallélisme limité,
- ... méthodes avec une faible intensité.

À préférer

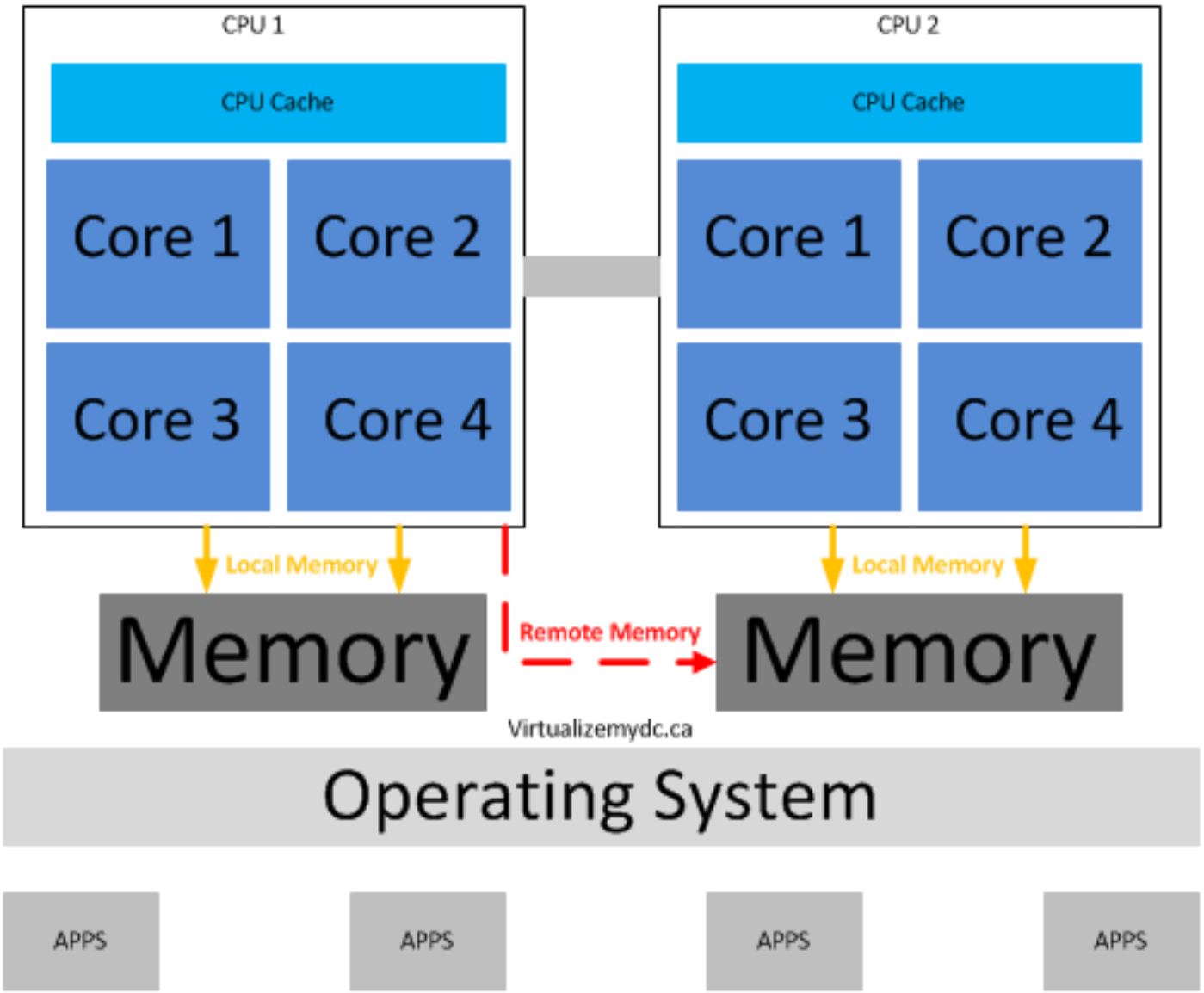
- ... méthodes de grande intensité,
- méthodes "embarrassingly" parallèles.

NUMA

Non Uniform Memory Access (NUMA)



Non Uniform Memory Access (NUMA)



Les accès distants sont très lents !

NUMA : remèdes

Fixer les threads sur les cœurs : `tasket` ou `numactl`

On évite qu'un thread ne migre vers un autre CPU et se retrouve alors à faire des accès distants :

```
$ numactl --cpunodebind=0 --membind=0 mysimulation
```

NUMA : remèdes

Fixer les threads sur les cœurs : `tasket` ou `numactl`

On évite qu'un thread ne migre vers un autre CPU et se retrouve alors à faire des accès distants :

```
$ numactl --cpunodebind=0 --membind=0 mysimulation
```

"Toucher" les données

La mémoire n'est réellement assignée, que lors de la première écriture, par page et en fonction du CPU.

1. allouer la matrice ;
2. créer n threads ;
3. initialiser la matrice par bloc et en // (1 thread = 1 bloc) ;
4. faire les opérations par blocs (1 thread = le même bloc).