L‘accès aux données – question critique pour le Calcul HPC

Frank Hülsemann,

EDF R&D, EDF Lab Paris-Saclay

25/10/2019

Plan de l'exposé

- Introduction
- Évolution des moyens de calcul HPC
  - puissance de calcul vs. accès aux données
- Matrices denses, Linpack
- Unités vectorielles
- Matrices creuses, SpMV
- Modèle de prédiction de performance Roofline
- Benchmark HPCG
- Conclusion

Contexte : EDF

- EDF : producteur et fournisseur d‘énergie
- EDF R&D :
  - environ 1900 personnes
  - En principe sur tous les sujets possibles liés à la production et la commercialisation de l‘électricité
  - Développeur de plusieurs logiciels de simulation, parmi d‘autres[1] :
    - Mécanique des structures : code_aster (https://www.code-aster.org)
    - Mécanique des fluides : *Code_Saturne* (https://www.code-saturne.org)

[1] https://www.edf.fr/groupe-edf/premier-electricien-mondial/activites/recherche-et-developpement/communaute-scientifique/codes-de-calcul

Calcul scientifique à EDF

- présent dans la liste Top500 (https://www.top500.org) depuis 2006 (d‘abord EDF R&D, puis EDF)

- Top500 de novembre 2018 :
  - No 97 (41472 cœurs x86_64)
  - No 460 (29568 cœurs x86_64)
  -
- Top500 de juin 2019 :
  - No 127 (toujours 41472 cœurs x86_64)

« Définitions »

- Définition de travail :
  HPC = calcul avec enjeu de performance
- Focus pour aujourd’hui : Calcul des champs physiques, principalement les applications « classiques » :
  - mécanique des fluides,
  - mécanique des structures,
- Hors scope aujourd’hui (mais certainement pas moins importants)
  - La chémie,
  - La biologie
  - L‘analyse des grands volumes de données
  - Les applications en temps réel
  - ...

Vers des simulations toujours plus complexes

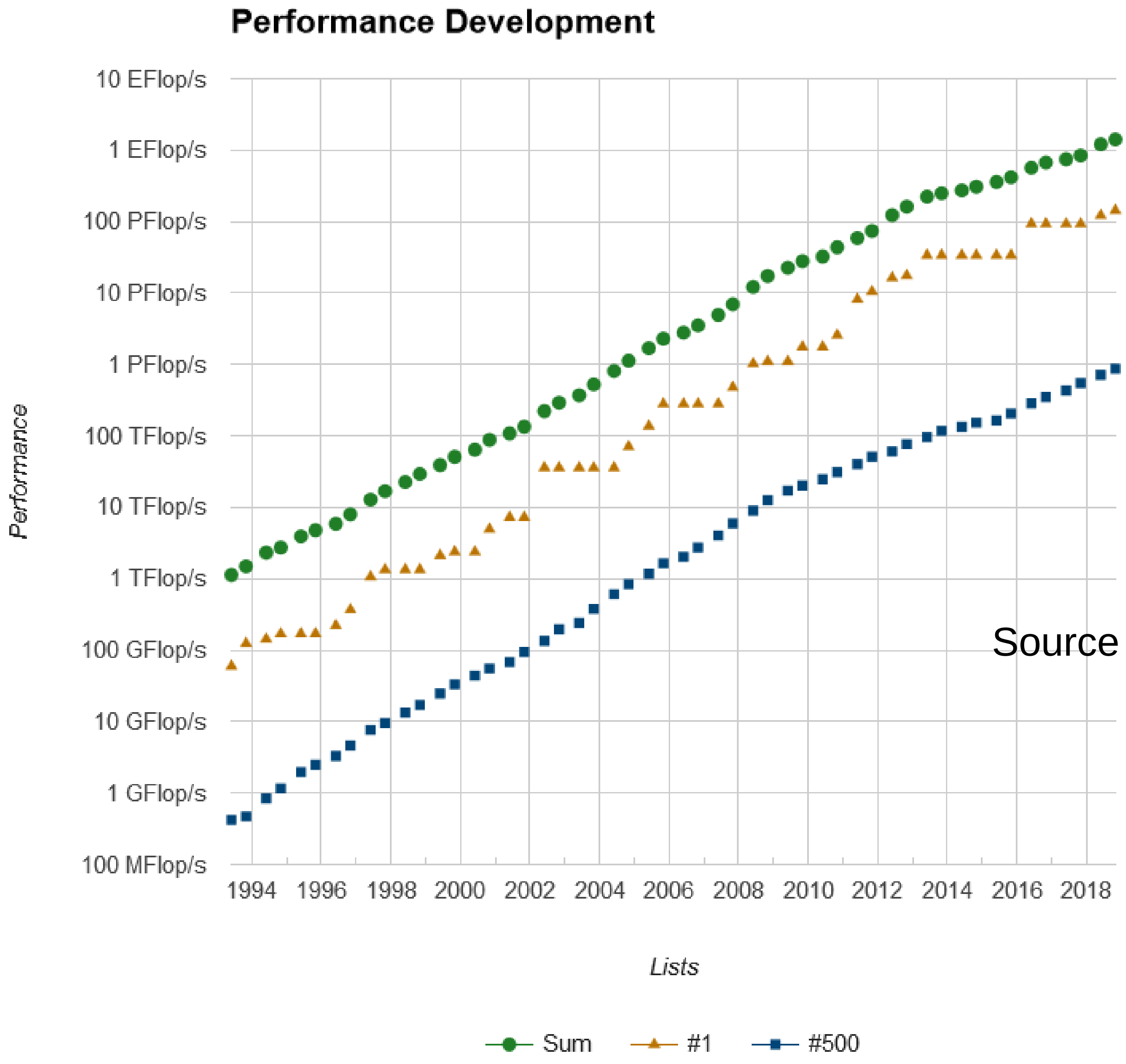
- Plus de phénomènes physiques dans les modèles
  - multi-physiques comme interaction fluide-structure
  - Turbulence, …
- Des Discrétisations plus robustes, plus fiables
  - Maillages complexes, de grande taille (> 10^9 inconnues)
  - Discrétisations basées sur la géométrie différentielle (mimetic, …)
- Et le tout sur des machines
  - De plus en plus grandes (scalabilité?)
  - De plus en plus puissantes
  - De plus en plus hétérogènes, donc compliquées *(NUMA : non-uniform memory access, cartes graphiques, accélérateurs)*

Une implémentation efficace, est-elle plus difficile aujourd'hui qu'il y a 15 ans ?

Un regard sur l'évolution des moyens de calcul

- Top500 : Liste des ordinateurs les plus performants depuis 1993
- Mesure de la performance (benchmark) : Linpack
- Caractéristiques du benchmark Linpack
  - Résolution d'un système linéaire Ax=b, A $\in \mathbb{R}^{N\times N}$
  - Matrice dense avec des valeurs réels en 64bit (souvent appelé « double précision »)
  - La taille N du problème est adaptée à la machine
  - $O(N^3)$ opérations arithmétiques

Top500 : L‘évolution de la performance



Source : http://www.top500.org

Les axes du progrès (en puissance de calcul)

- Initialement : la fréquence *(mais ça, c‘est terminé)*
- Augmentation du nombre des nœuds (mémoire distribuée – MPI)
- Augmentation du nombre des cœurs par nœud (mémoire partagée – MPI/OpenMP/…)
- Introduction des unités vectorielles/dédiées. Exemple X86 :
  - SSE2 (4 x 32 bit, 2 x 64 bit) en 2001
  - AVX (8 x 32 bit, 4 x 64 bit) en 2011
  - AVX-512 (16 x 32 bit, 8 x 64 bit) en 2015
  - FMA : fused multiply-add, d = a•x+y, en 2011
- Introduction des cartes accélératrices
  - Cartes « graphiques »
  - Cartes dédiées
  - *Intel Xeon Phi*
- (amélioration du réseau)

Les axes du progrès (en performance)

- ASCI Red : 2,379 Tflop/s, N=362 880
  - première machine X86 au numéro 1 (juin 1999)
  - 4736 puis 4816 nœuds à 2 cœurs (9472/9632 cœurs au total), 333 MHz
  - Sans unités vectorielles
- Frontera : 23 516,4 Tflop/s, N = 9 262 848
  - No. 5 (juin 2019) - le mieux placé X86_64 sans accélérateur
  - 8008 nœuds à 56 cœurs (448 448 cœurs au total), 2.7 GHz
  - AVX-512

Le Linpack va bien, tout va bien?

- Le benchmark Linpack profite (bien) de l'évolution de la puissance de calcul :
  - 1999 : No. 1 (ASCI Red) 74% de la performance crête (Rpeak)
  - 2019 : No. 1 (Summit) : 74% de Rpeak
  - 2019 : No. 5 (Frontera) : 61 % de Rpeak
- Réponse partielle à la question de la programmabilité :

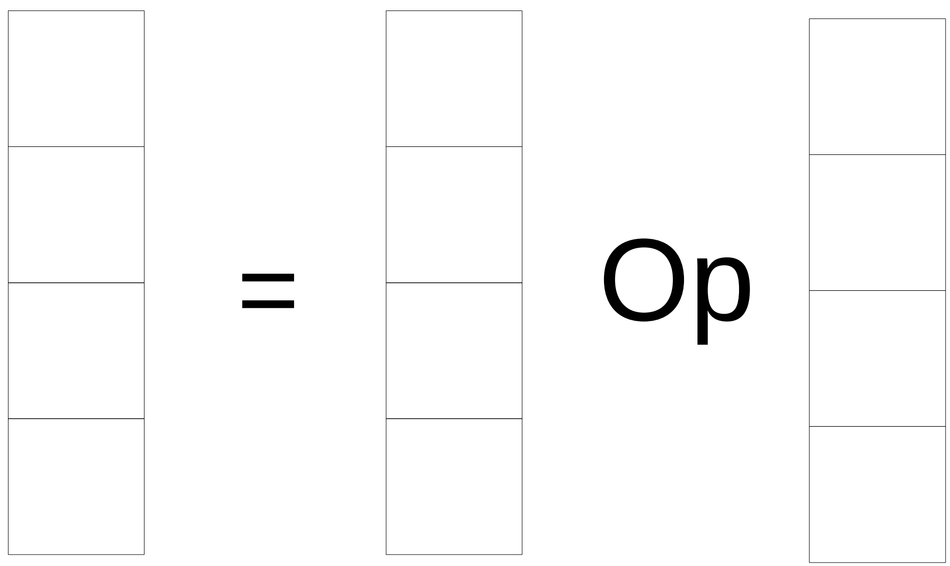
  Si la programmation est devenue plus compliquée, pour le Linpack, on a su surmonter les obstacles.

Linpack II

- Caractéristiques du benchmark Linpack
  - *Résolution d'un système linéaire Ax=b,* $A \in \mathbb{R}^{N \times N}$
  - *Matrice dense avec des valeurs réels en 64bit (souvent appelé « double précision »)*
  - *La taille N du problème est adaptée à la machine*
  - *$O(N^3)$ opérations arithmétiques*
  - **pour $O(N^2)$ données** (structurées, d'accès régulier)

Focus sur les unités vectorielles

- AVX (8 x 32 bit, **4 x 64 bit**)



$Op \in \{+,-,\times\}$

- SIMD : Single Instruction, Multiple Data

  Appliquer **la même opération** à **tous les éléments** de deux vecteurs => 4 opérations arithmétiques (« double précision ») **par cycle**

Focus sur les unités vectorielles (II)

- AVX (8 x 32 bit, 4 x 64 bit), 4 opérations arithmétiques 64bit par cycle
- Autrement dit : Un logiciel « scalaire » ignore
  - 3/4 de la puissance de calcul installée sur une machine AVX
  - 7/8 de la puissance disponible sur une machine AVX-512

OK, pour le calcul performant, les unités vectorielles sont essentielles. Commet peut-on les programmer ?

- Faire confiance au compilateur
- OpenMP (#pragma omp simd) – à partir d'OpenMP 4.0
- Pragma non-standadisés (Intel)
- Compiler intrinsics (non-standardisé)
- Assembler

Focus sur les unités vectorielles (III)

- BLAS : Basic Linear Algebra Subroutines
  - BLAS 1 : Opérations vecteur/vecteur
  - BLAS 2 : Opérations entre matrices et vecteurs
  - BLAS 3 : Opérations matrice/matrice
- Supposons qu'un algorithme utilise beaucoup d'opérations BLAS1 entre des longs vecteurs de réels :
  - R = X + Y $\alpha \in \mathbb{R}$ $R, X, Y \in \mathbb{R}^N$
  - Y = α × X + Y « [s,d]axpy »

Les instructions AVX vont-elles accélérer le logiciel ?

Exemple : Station de travail

- Intel Xeon E3-1280v3 (mono-processeur) :
  - Fréquence 3,6 GHz
  - 4 cœurs
  - AVX
  - Bande passante (selon constructeur) : 25,6 GB/s
- Puissance de calcul
  - Scalaire, mono-cœur : 3,6 GFlop/s
  - AVX, mono-cœur : 28,8 GFlop/s
  - AVX, 4 cœurs : 115,2 GFlop/s

Exercice :

Pour un processeur Xeon E3-1280v3, estimez le temps de calcul pour la sommation de deux vecteurs de réels 64 bits de longueur $1,6 * 10^9$ entrées.

Pour simplification,

- supposez que 10^9 Byte $\approx$ 1 GB,
- ignorez le temps pour l'écriture des résultats vers la mémoire.

- Pour un calcul mono-cœur, scalaire
- Pour un calcul mono-cœur, AVX
- Pour un calcul 4 cœurs, AVX

Exercise : Conclusion

- 2 x 1,6 x 10^9 entrées à 8 Bytes chacune =>

  8 x 3,2 x 10^9 Bytes à lire => 25,6 GB à lire au total
- Nombre d'opérations arithmétiques : 1,6 x 10^9 additions
- Tmin : tout le calcul se fait pendant le transfert
- Tmax : tout le calcul se fait après le transfert
- Tmin = 1s
- 1C, scalaire : Tmax = Tmin + 1,6/3,6 s $\approx$ 1s + 0,45s
- 1C, AVX : Tmax = Tmin + 1,6/28,8s $\approx$ 1s + 0,05s
- 4C, AVX : Tmax = Tmin + 1,6/115,2s $\approx$ 1s + 0,014s

Exemple : Nœud de cluster, carte GPGPU

- Pour rappel : Station de travail
  - 115 GFlop/s pour 25,6 GB/s

- Pour comparaison : un nœud de calcul d’un cluster récent :
  - 2 processeurs Intel Xeon E5-2680v4
  - 2x14 = 28 cœurs
  - 1 075,2 GFlop/s pour 153,6 GB/s   (x9 / x6)

- Carte Nvidia Tesla V100
  - 7 000 GFlop/s pour 900 GB/s   (x61 / x35)

Déséquilibre généralisé entre la puissance de calcul et la capacité d‘alimenter le calcul en données à partir de la mémoire principale.

Parenthèse : Le retour du vecteur ?

- Annoncé à Supercomputing 2017 :

  NEC SX-Aurora TSUBASA
  - Carte PCIe avec processeurs vectoriels
  - 2 450 GFlop/s pour 1 200 GB/s (x21 / x47)

- Modèle à notre disposition :
  - 2 150 GFlop/s pour 1 200 GB/s (x19 / x47)

Le sujet « mémoire » à Supercomputing 2016

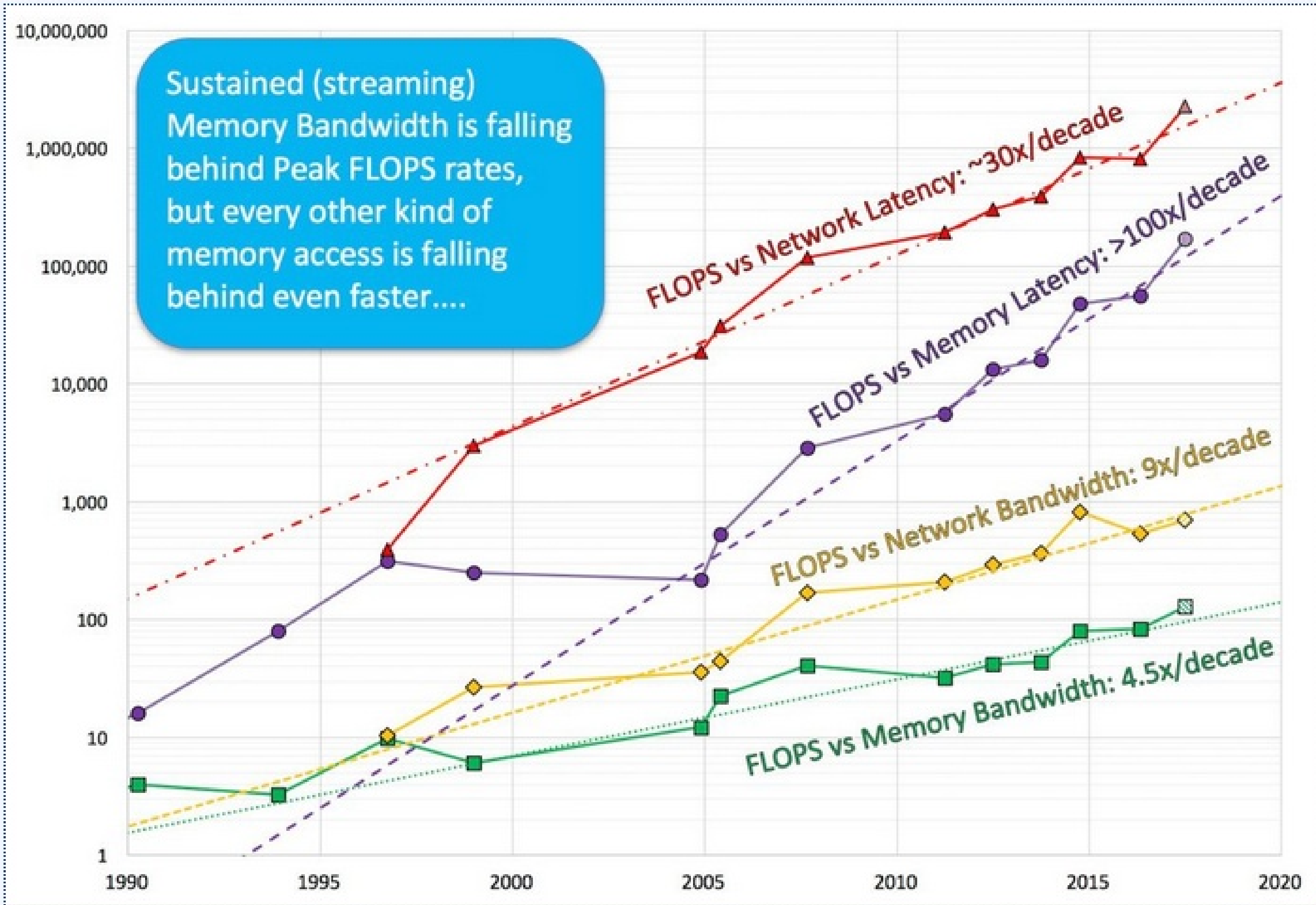


Image caption: Trends in the relative performance of floating-point arithmetic and several classes of data access for select HPC servers over the past 25 years.

Présentation de John D. McCalpin, développeur du stream benchmark

http://sc16.supercomputing.org/2016/10/07/sc16-invited-talk-spotlight-dr-john-d-mccalpin-presents-memory-bandwidth-system-balance-hpc-systems/

Stream : LE cas test pour la bande passante

- Comme dans le cas de la puissance de calcul, les valeurs pour la bande passante fournies par les constructeurs sont souvent difficiles à atteindre.
- LE cas test « Stream » de J. McCalpin cherche à évaluer le débit RAM -> CPU pour des opérations sur des grands vecteurs (hors caches). Site web :

  http://www.cs.virginia.edu/stream/ref.html

Changement d‘application : Matrices creuses au lieu du Linpack (matrices denses)

- « Définition de travail » :

  Une matrice est creuse, quand il est préférable (pour une raison ou une autre) de stocker seulement les valeurs différentes de zéro.

- Grand nombre d’applications :

  Des matrices creuses servent à décrire

  - des réseaux (sociaux, routiers, …)
  - des graphes (potentiellement avec des poids attachés aux arêtes)
  - des modèles discrétisés des phénomènes physiques

Matrices creuses

Les techniques de discrétisation comme

- Différences finies
- Éléments finis
- Volumes finis

impliquent des matrices, souvent carrées,

- de grandes tailles (tendance $\uparrow$)
  - 10^7 à 10^8 lignes en mécanique des structures
  - 10^9 à 10^{12} lignes en mécanique des fluides
- très creuses (de 1 à 300 non-zéros par ligne)
- en général non-symétriques
- et sans motif particulier

Matrices creuses II

- Les systèmes linéaires creux de grandes tailles sont, pour l’instant encore, en dehors de la portée des méthodes derrières Linpack.
- Les opérations de base des méthodes de résolution dites itératives sont
  - des opérations BLAS1 entre vecteurs,
  - le produit entre la matrice creuse et un vecteur (SpMV pour Sparse Matrix Vector product)
- Quelle performance à attendre sur du matériel récent ?

Structure de données pour matrices creuses

Propositions ?

- Toujours un sujet de recherche
- Chaque année, des nouvelles variantes apparaissent

- Idée de base : ne stocker que les entrées non-zéro

Exemple : CSR

CSR : Compressed Sparse Row format

$$A = \left\{ \begin{matrix} 2 & -1 & \\ -2 & 4 & -3 \\ & -4 & 6 \end{matrix} \right\}$$

NNZ = 7;   NLigne = 3

Mat :

2	-1	4	-3	-2	-4	6

Col :

1	2	2	3	1	2	3

Row_start

1	3	6	8

Exemple : CSR

- Mat :
  - vecteur de réels de longueur NNZ
  - Contient les entrées de la matrice
- Col :
  - Vecteur d‘entiers de longueur NNZ
  - Contient les indices des colonnes des entrées d‘une ligne, dans le même ordre que les entrées de Mat
- Row_start
  - Vecteurs d‘entiers de longueur Nligne+1
  - Contient pour chaque ligne le premier indice dans Mat et Col

Exemple : SpMV, version CSR

```
/*  res = A*Vec      */
/*  A in CSR format */

for (int r=1; r<=A->NLigne; ++r){
  double tmp; tmp=0.0;
  int rs = A->Row_start[r];
  int re = A->Row_start[r+1];

  for (int lc=rs; lc<re; ++lc){
      tmp += A->Mat[lc] * Vec[ A->Col[lc] ];
   }

  res[r] = tmp;
}
```

The Roofline Model

- Représentation graphique des facteurs déterminants (limitants?) pour la performance
- Concept central : « operational intensity » (intensité des opérations) défini comme suite :
  - OI = Nombre des opérations par octet lu ou écrit (Flops per Byte)
- Une borne supérieure de la performance en Flop/s est en suite déterminée par

  min **{** (operational intensity [Flop/B] x bande passante [Byte/s]) ;
  performance crête [Flop/s] ;   **}**

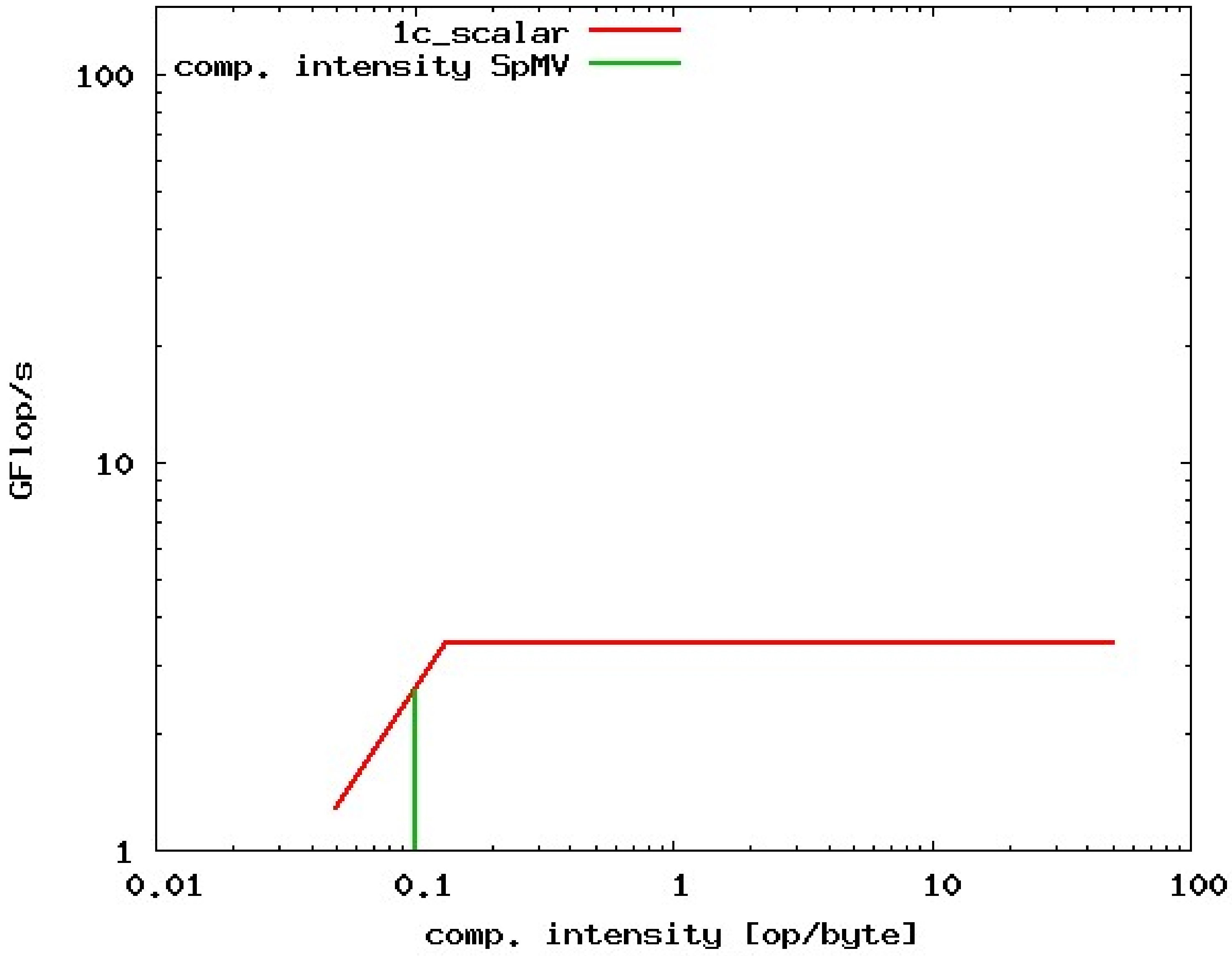
  S. Williams, A. Waterman, D. Patterson,
  "Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures",
  Communications of the ACM (CACM), April 2009,
  doi: 10.1145/1498765.1498785

Operational Intensity

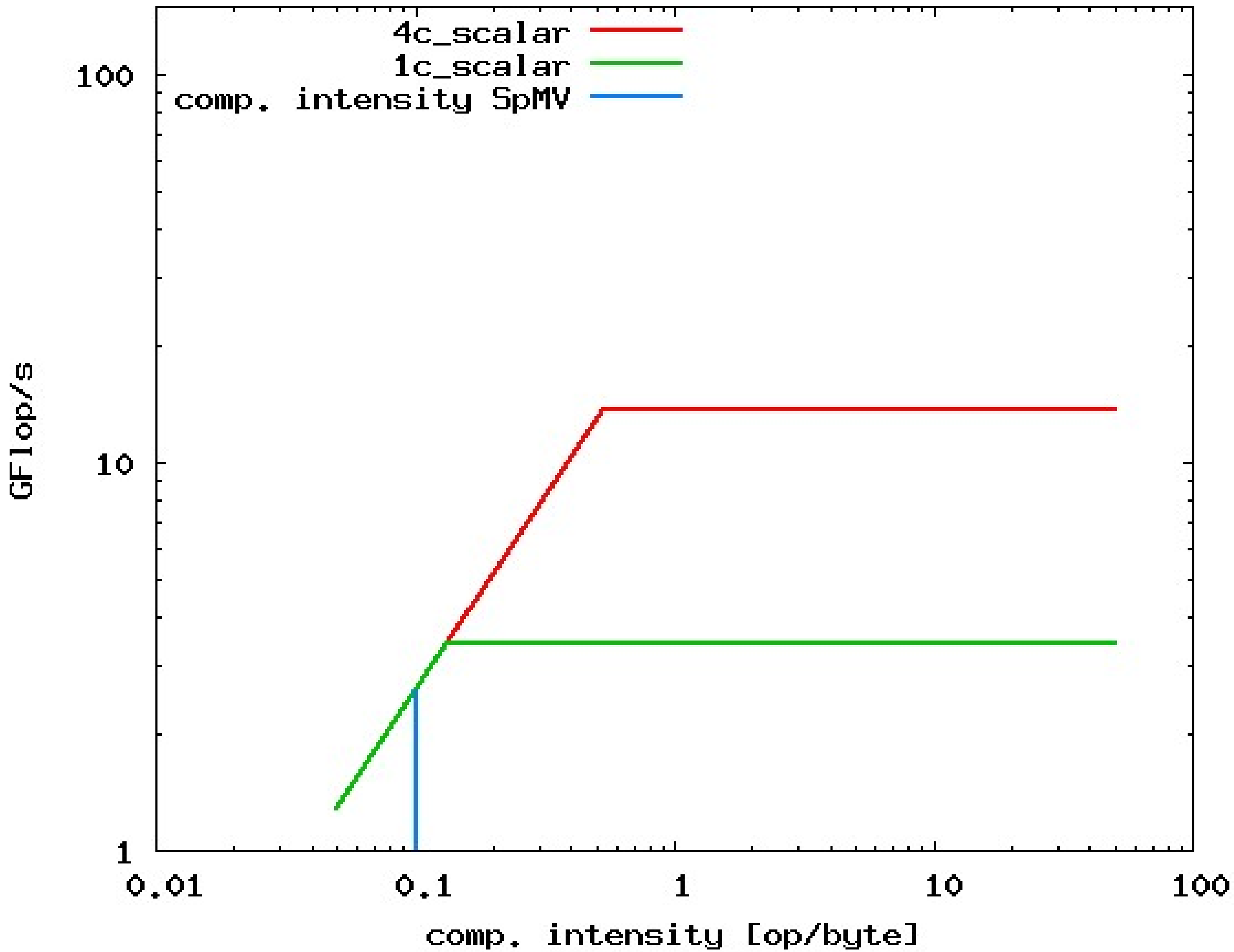
- **Somme de deux vecteurs de réels 64 bit :**
  - 1 opération pour 16 Bytes lus + 8 Bytes écrits
    => OI(VecSum) = 1/24 ≈ 0.042
- **SpMV :**
  - Supposons des indices entiers de 32 bit (4 Bytes)
  - Supposons des réels de 64bit (8 Bytes) pour les entrées de la matrice et du vecteur
- **Accès mémoire :**
  - NNZ termes de la matrice et du vecteur => NNZ x 2 x 8 Bytes
  - NNZ indices => 4 x NNZ Bytes, au total 20 x NNZ Bytes
  - NLigne valeurs => NLigne x 8 Bytes
- **Opérations :**
  - NNZ mult. + (NNZ – NLigne) add. ≈ 2 x NNZ opérations
- **OI(SpMV) ≈ 1/10 [F/B]**

Roofline model pour SpMV

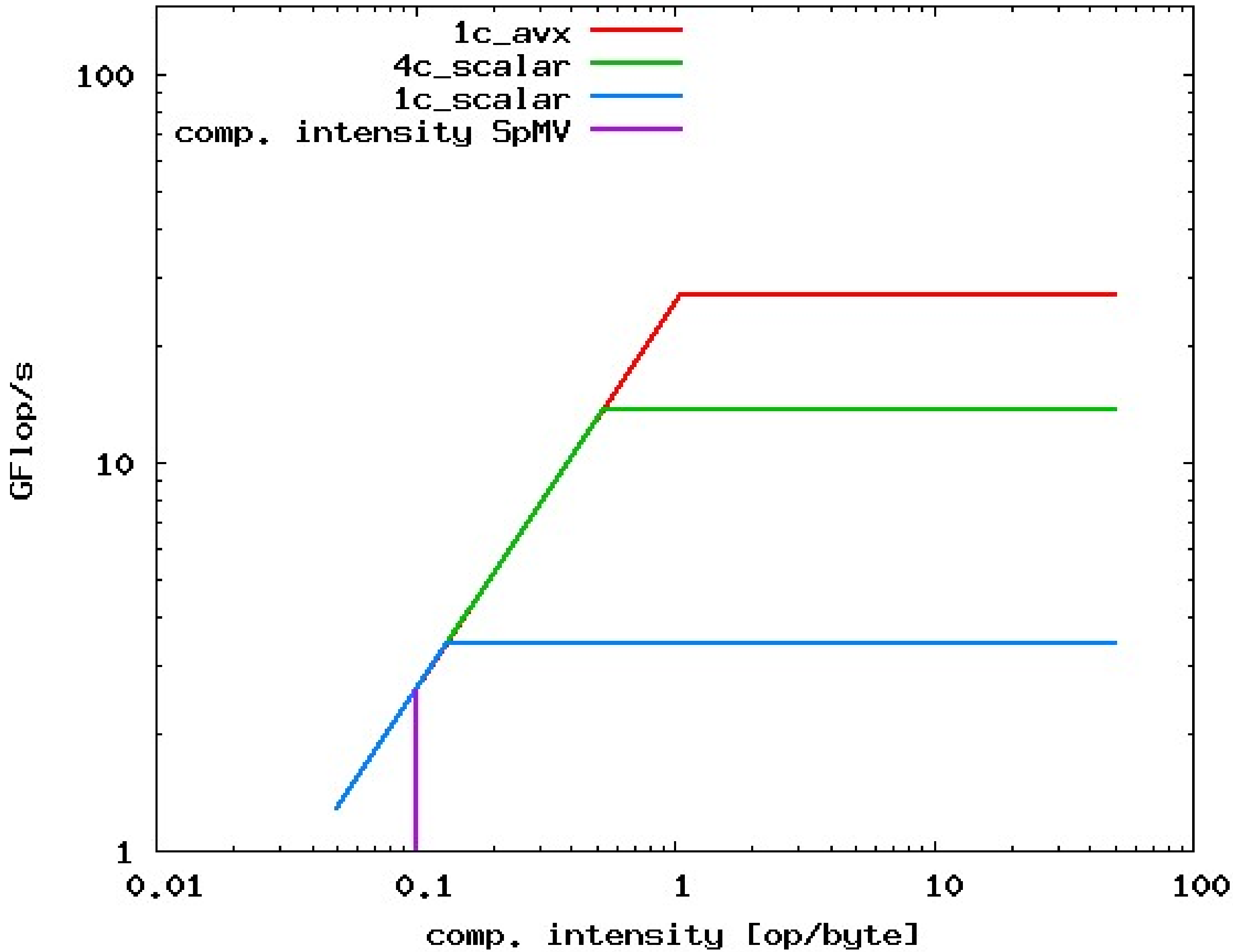
Processeur : Intel Xeon E3-1240v3, 4 Cœurs, 3,4 Ghz, AVX



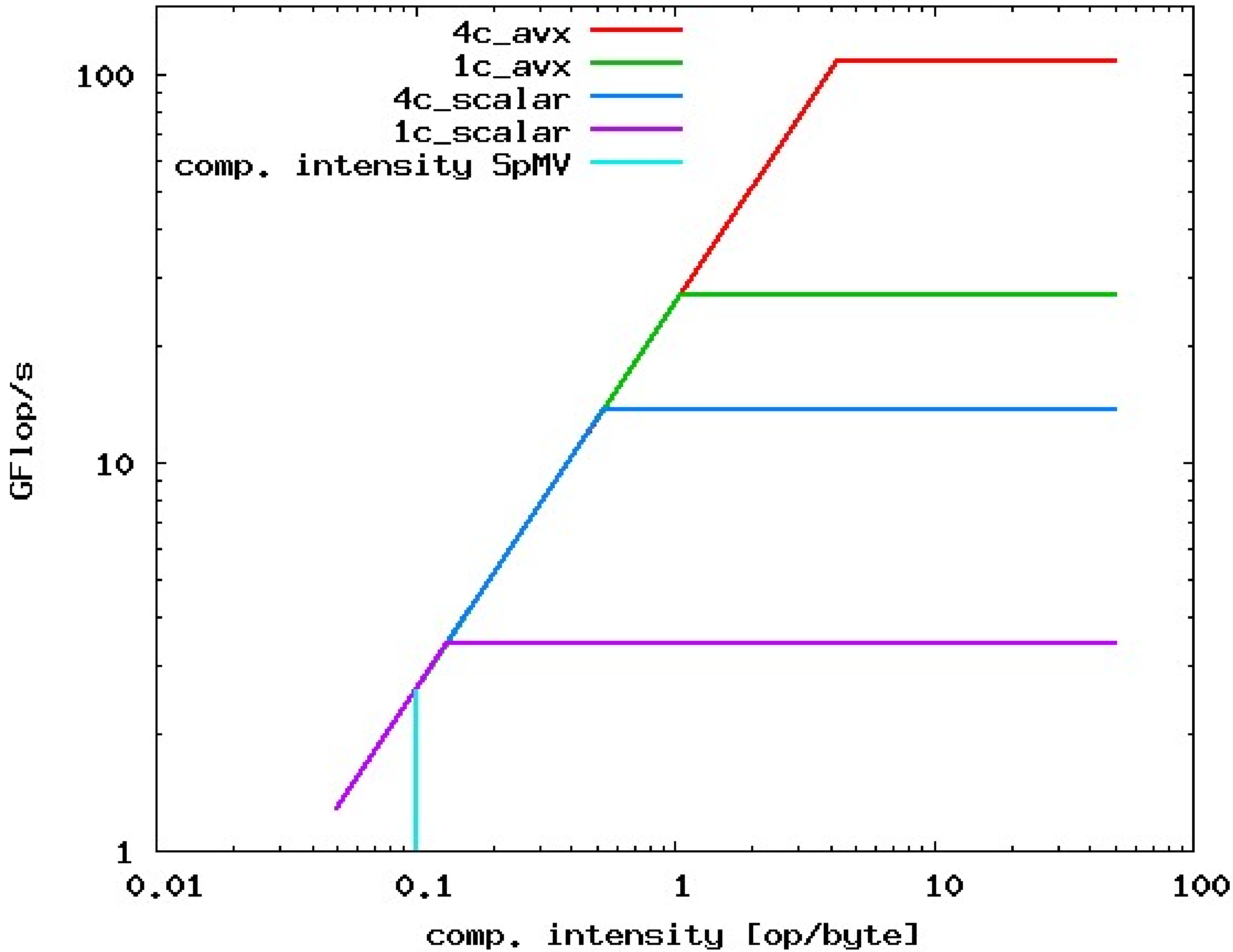
Roofline model pour SpMV



Roofline model pour SpMV



Roofline model pour SpMV



Interpretation du modèle Roofline

- **Un algorithme limité par la bande passante ne profite pas d'une augmentation de la puissance de calcul seulement.**
- **Options pour les algorithmes limités par la bande passante :**
  - Changement de type de variable (double - > float)
  - Changement de l'algorithme dans le but de réutiliser des données déjà chargées (« localité des données »).
  - Introduction des calculs redondants pour réduire les transferts mémoire-processeur.
- **Dans le contexte des matrices creuses, on constate des faibles gains en augmentant la puissance de calcul (vectorisation, parallélisation) malgré la conclusion du modèle roofline.**

  Exemple : 1,1 % de Rpeak au lieu de 1 % => 10 % de gain, par contre, toujours 98,9 % non-exploités.

Un benchmark sur des matrices creuses

- **Conclusion partagée par la communauté HPC :**
  - les résultats du Linpack ne sont que très peu pertinents pour certaines autres applications, en particulier celles sur des matrices creuses
- **Depuis 2014 : Benchmark HPCG comme complément au Linpack**
  - Solveur linéaire itératif sur une matrice creuse, issue d’une discrétisation spatiale 3D.
  - http://www.hpcg-benchmark.org
  - O(N) données pour O(N log N) d‘opérations
- **L’ordre des machines dans les TOP500 selon le HPCG ≠ l’ordre selon Linpack**
- **Encore : moins de résultats pour le HPCG que pour le Linpack**

Les ingrédients du gradient conjugué

- Gradient conjugué sans préconditionnement :

$$\alpha \in \mathbb{R}, \quad x, y \in \mathbb{R}^N, \quad A \in \mathbb{R}^{N \times N}$$

daxpy $\qquad y \leftarrow \alpha x + y$

produit scalaire $\qquad \langle x, y \rangle = \sum_i x_i y_i$

norme $\qquad \|x\| = \sqrt{\langle x, x \rangle}$

daypx $\qquad y \leftarrow \alpha y + x$

produit matrice-vecteur $\quad y = A x$

L‘intensité arithmétique d‘un gradient conjugué

- « réels » : 8 Bytes, entiers : 4 Bytes, matrice en format Ellpack, 7 entrées par ligne

Opération	Intensité arithmétique (i) [F/B]
daxpy	2/24 = 1/12
Produit scalaire	1/16 < i < 1/8 (selon l‘accumulation)
norme	?? (selon l‘algorithme de la racine carrée)
daypx	1/12
Produit matrice-vecteur	13/148 ≈ 0.088

Prédiction de la performance par roofline

- GC sur Nec SX-Aurora T10B (1c, 268 GF/s, 350 GB/s Stream)

Opération	Intensité arithmétique [F/B]	Prédiction [GF/s]
daxpy	2/24 = 1/12	(1/12)*350 = 29,2
produit scalaire	1/16 < i < 1/8	350/16 = 21,9, 350/8 = 43,8
norme	??	??
daypx	1/12	29,2
spmv	13/148 ≈ 0.088	30,7

Performance observée

- GC sur Nec SX-Aurora T10B (1c, 268 GF/s, 350 GB/s Stream)

Opération	Prédiction [GF/s]	Observation [GF/s]
daxpy	29,2	28,9
produit scalaire	21,9 – 43,8	34,8
norme	??	59,4
daypx	29,2	30,1
spmv	30,7	**17,3**

Interprétation des résultats

- Sur l'architecture vectorielle, le modèle « roofline » est précis pour les opérations BLAS1.
- Dans ce cas précis, la différence pour le produit matrice-vecteur s'explique par la bande passante disponible pour des accès irréguliers à la mémoire.
  - Stream ne contient « que » des tests à accès régulier (linéaire même).
- Sur cette architecture, l'accès irrégulier divise la bande passante disponible par un facteur 2.

Conclusion

- **Tous les algorithmes ne profitent pas de l'évolution des moyens de calcul :**
  - Le Linpack en profite encore, au contraire des opérations sur des grands vecteurs et du SpMV.
- **Certaines classes de méthodes** (matrices creuses) **sont tellement mal-adaptées aux moyens de calcul actuels que d'autres méthodes, plus coûteuses en opérations mais aussi avec des meilleures intensités d'opérations, deviennent compétitives** (exemple : Fast Multipole Methods).
- **Un modèle de performance comme le Roofline model aide à prédire si un certain algorithme est bien adapté à une architecture ou pas.**