Processeurs

LIBERATE IT

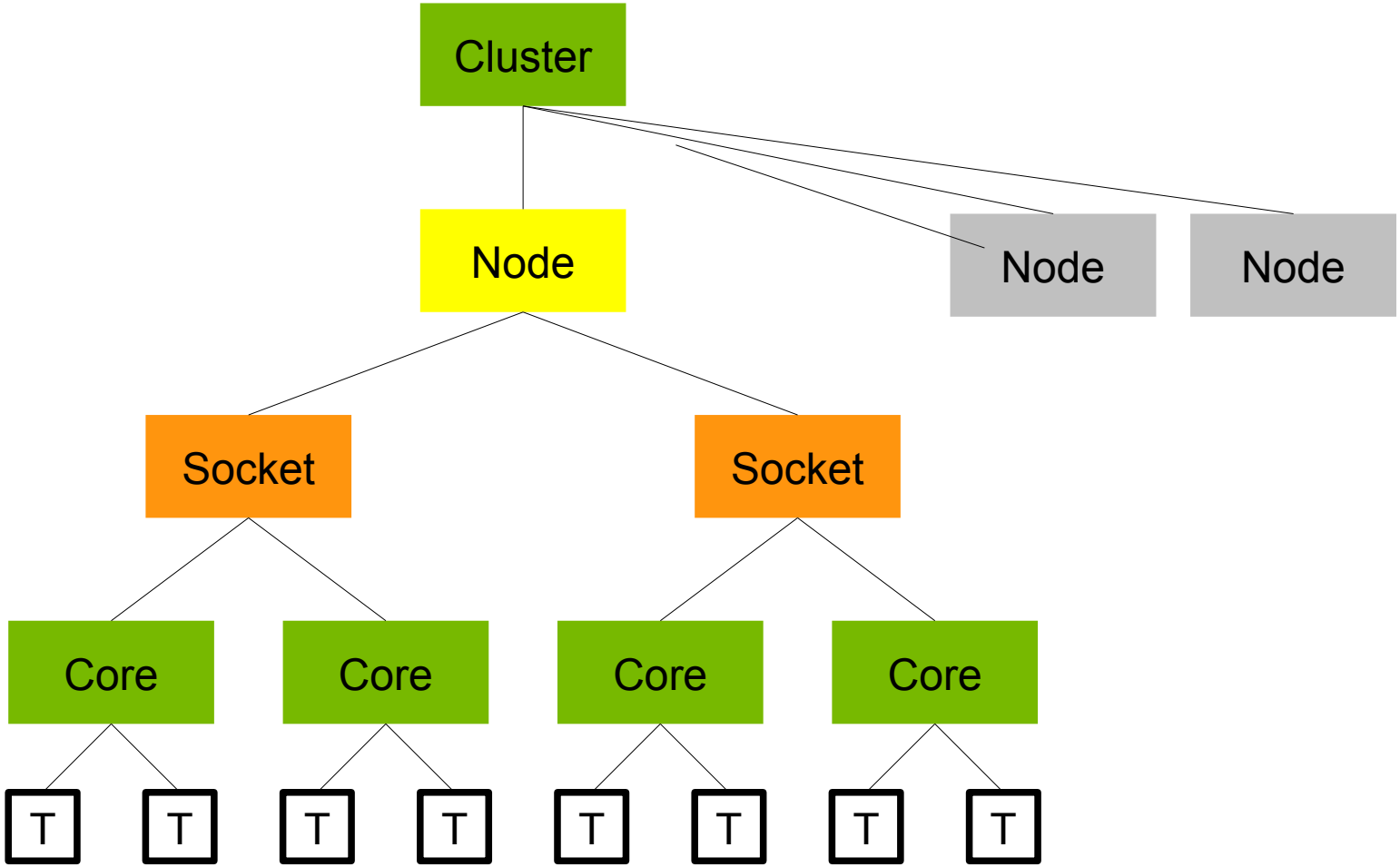
Processeurs

- A l'heure actuelle, dominé par les multi-coeurs
  - 2-coeurs (Intel Woodcrest, Wolfdale, IA-64 Montecito ...)
  - 4-coeurs
    - Les faux : 2x2-coeurs (Intel Clovertown, Harpertown).
    - Les vrais : Intel Nehalem-EP
  - 6-coeurs
    - AMD Opteron Istanbul 24xx
    - Futur Intel Westmere
  - Pas anodin en terme de performances ...
- On évite de parler de CPUs : on distingue clairement le « package » (ou parfois socket) de l'unité de calcul elle-même (le coeur)
- Pour l'instant, le marché des x86_64 est dominé par les serveurs bi-processeurs multi-coeurs.

Processeurs - caractéristiques

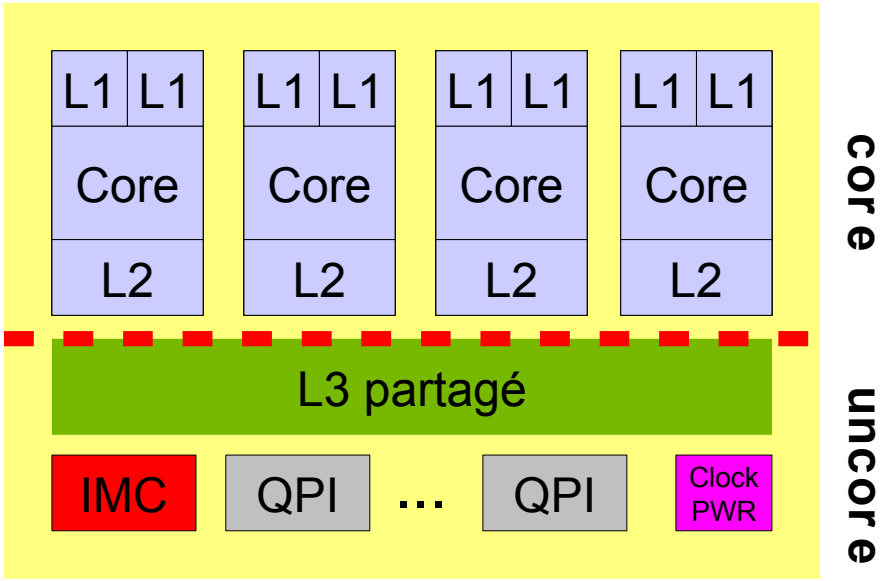
- La largeur des données entières (arithmétique) supportée
- Fréquence (unité : GHz)
- Cache : niveau, taille, organisation, associativité
- Gestion des accès à la mémoire centrale
  - Bus, liens (contrôleur intégrés) → vitesse, débit (GT/sec, GiB/sec)
  - Translation d'adresses, TLB ...
- Micro-architecture
  - Caractéristique du FPU
    - Largeur
    - Nombre d'instruction/cycle (3-4)
  - Les jeux d'instruction supportés (extension SIMD) – parallélisme de données
    - MMX, SSE, 3D-Now!, AVX ...
  - « Ordering » des instructions :
    - IA64 – In order
    - AMD64/EM64T – out of order
  - Pipelinning - parallélisme d'instructions

1 noeud, 2 sockets, 4 coeurs, 8 threads ...



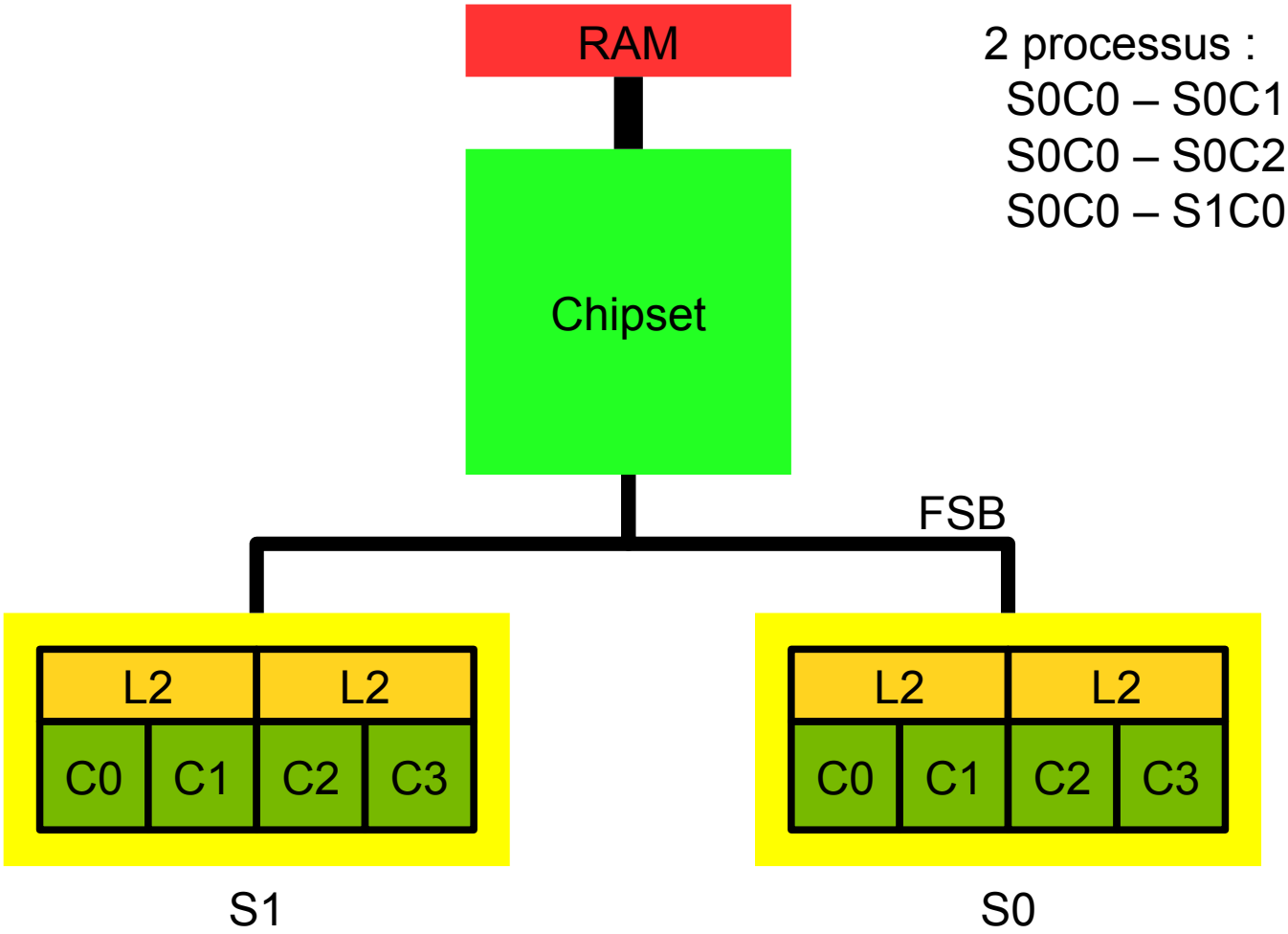
De plus en plus complexes ...

- Les processeurs sont des entités de plus en plus complexes
  - Multithreading (SMT)
  - Tuboboost
  - Gestion de l'énergie
  - Domaine core/uncore
  - Compteurs hardwares
    - Performances
    - Renseignement diverses
  - Gestion de la mémoire ...



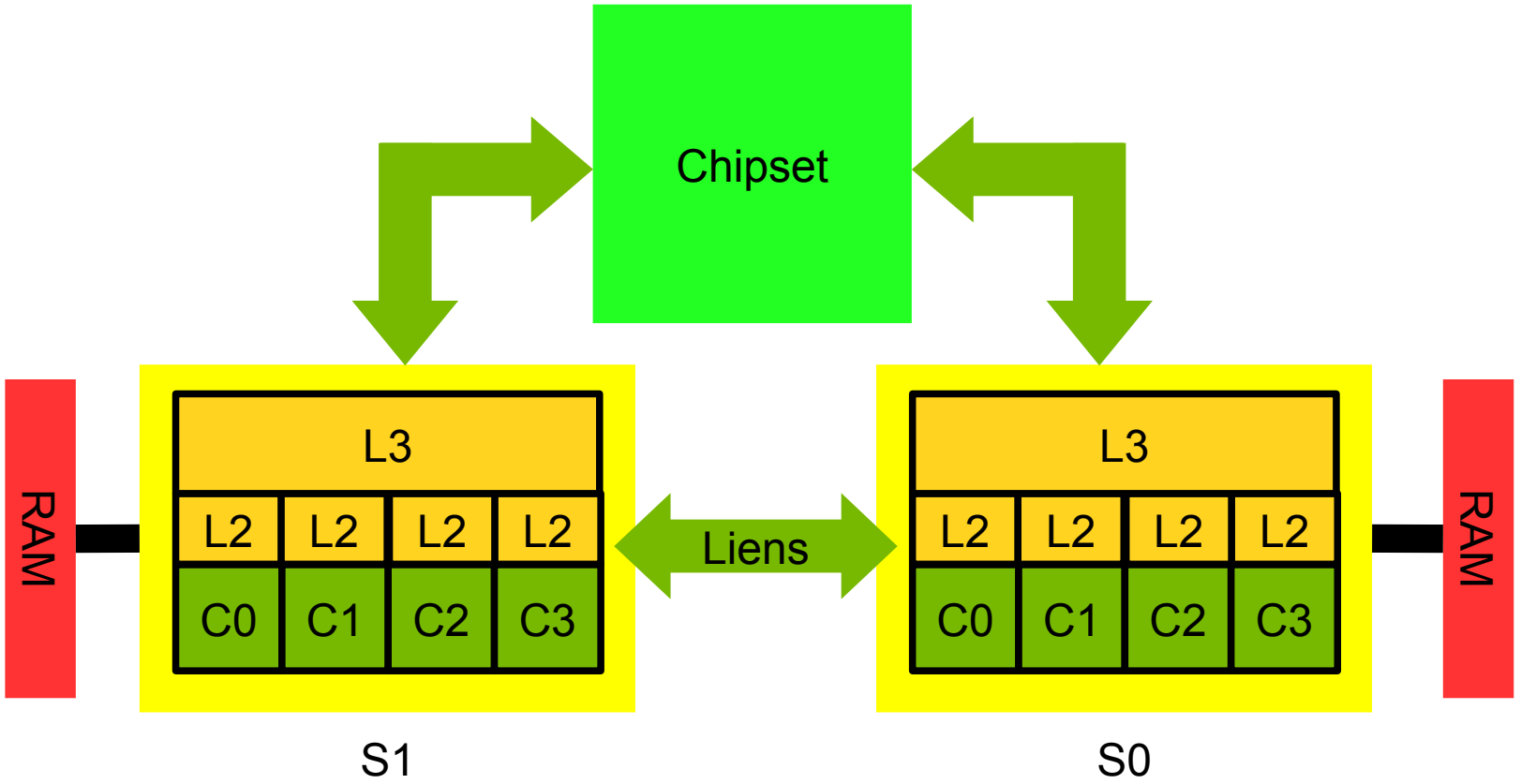
- Un processeur ne se réduit pas à sa seule fréquence !

Configuration non-NUMA avec BUS



2 processus :
S0C0 – S0C1
S0C0 – S0C2
S0C0 – S1C0

Configuration NUMA

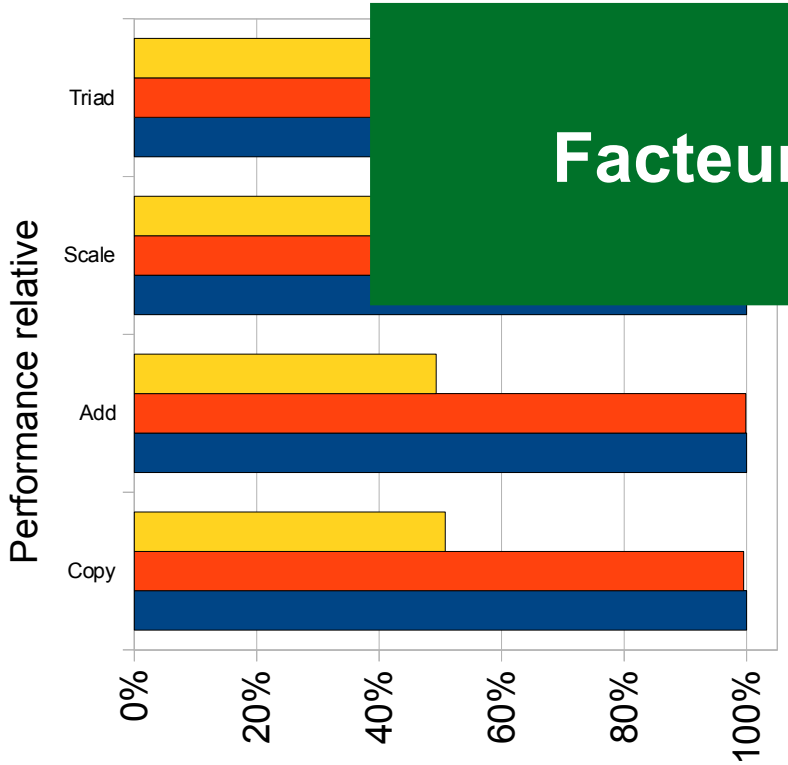


NUMA : bande passante mémoire

- Linux possède tout ce qu'il faut pour générer le NUMA

conf. n/m :

numactl –membind=n –cpubind=m ./cmd arg1, arg2, ...



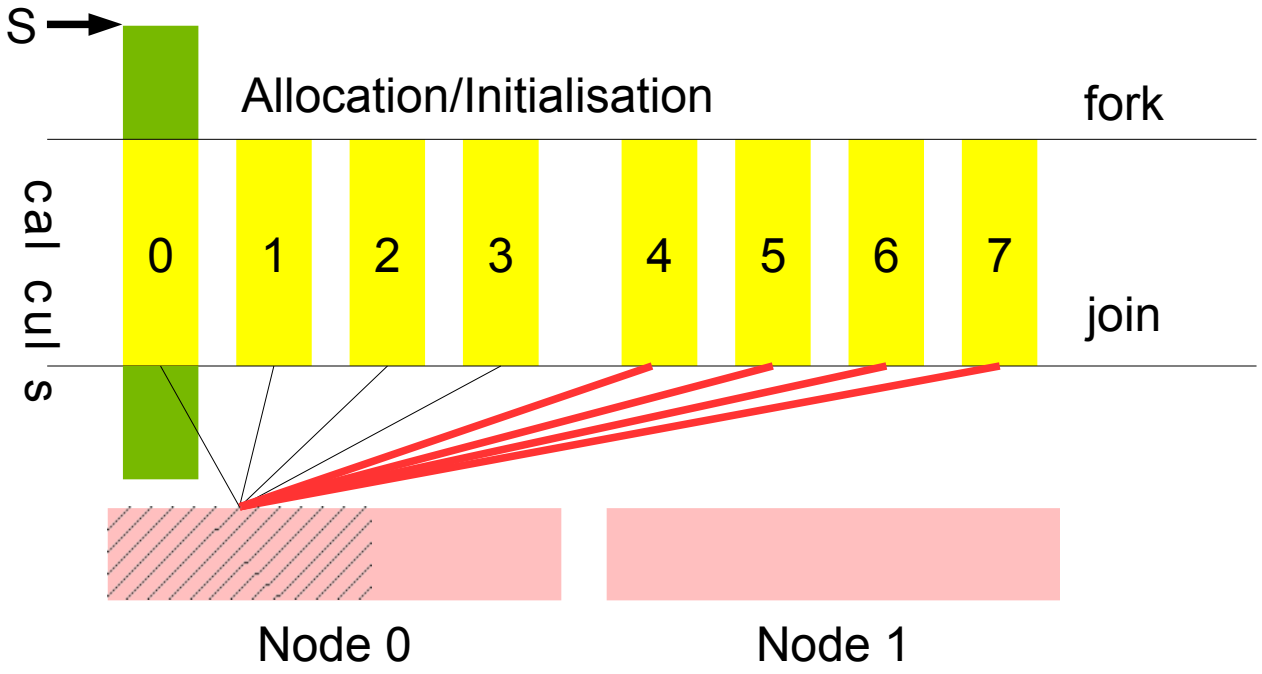
Facteur NUMA de 1:2

r Intel X5570
lz.

Conf.	Copy	Add	Scale	Triad
0/0	17828	18015	16786	17168
1/1	17739	17988	16747	17121
0/1	9058	8883	9223	9202

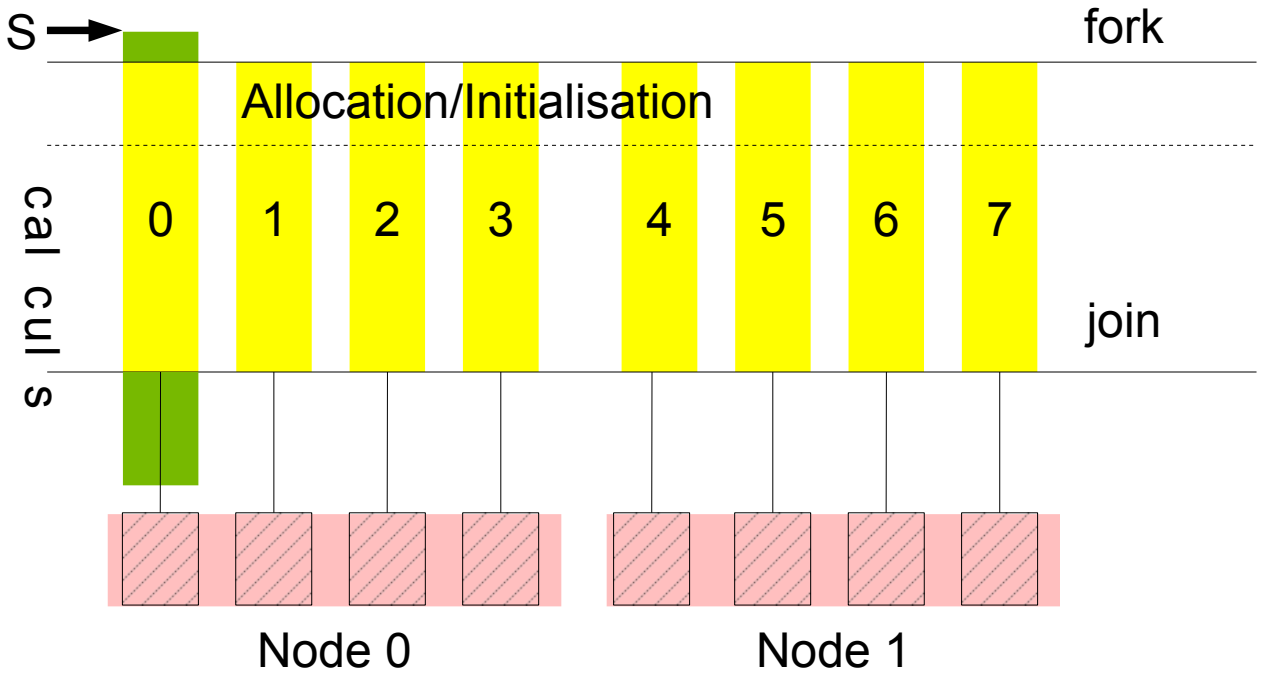
Effet du NUMA

- Processus OpenMP 8 threads, placement « compact »



Effet NUMA

- Processus OpenMP 8 threads, placement « compact »



Processeurs

- Capacité à effectuer des calculs
  - Nombre d'instructions traités par cycles
    - Généralement accessible par un compteur du processeur
    - Certains noyaux permettent via des drivers particuliers de remonter cette information dans l'espace utilisateur
  - **Dans le cadre d'un calcul scientifique, c'est le nombre d'opérations flottantes effectuées par seconde qui est intéressant : FLOPS/sec** plus généralement de nos jours GFLOPS/sec
    - Le calcul du maximum est simple :

      Pmax = (nbre d'op/cycle) x Frequence

      Ex: Intel X5570 : Pmax = 4 op/cycles x 2.933 GHz = 11 GFLOPS/sec par coeur (dans le Top500 en 1997!)
    - Accès via compteurs « hardware » et bibliothèques de performance tel que PAPI
    - Plus généralement : comptage « empirique »

Comptage empirique

- On isole la partie du code à etudier
- A partir de l'algorithme on détermine effectivement le nombre d'opérations flottantes à effectuer dans cette portion de code.
- On « time » (avant/après)
- On déduit simplement le
- Attention :
  - On entend généralement par op. Flottante les additions et les multiplication ; les divisions font partis des opérations complexes pour un processeur.
  - Le timing est parfois une opération compliquée, surtout sur les très petites durées (moteur ooo).

Exemple

- Multiplication matrice en double précision : operation BLAS3 (dgemm)
  - Plusieurs implementation : netlib, GotoBLAS, MKL ...

```
T1=time()
do i=1,n
  do j=1,n
    do k=1,n
      C(i,j) = alpha*A(i,k)*B(k,j) + beta*C(i,j)
    enddo
  enddo
enddo
T2=time()
```

FLOPS = $4N^3$
FLOPS/sec = FLOPS/(T2-T1)

Exemple

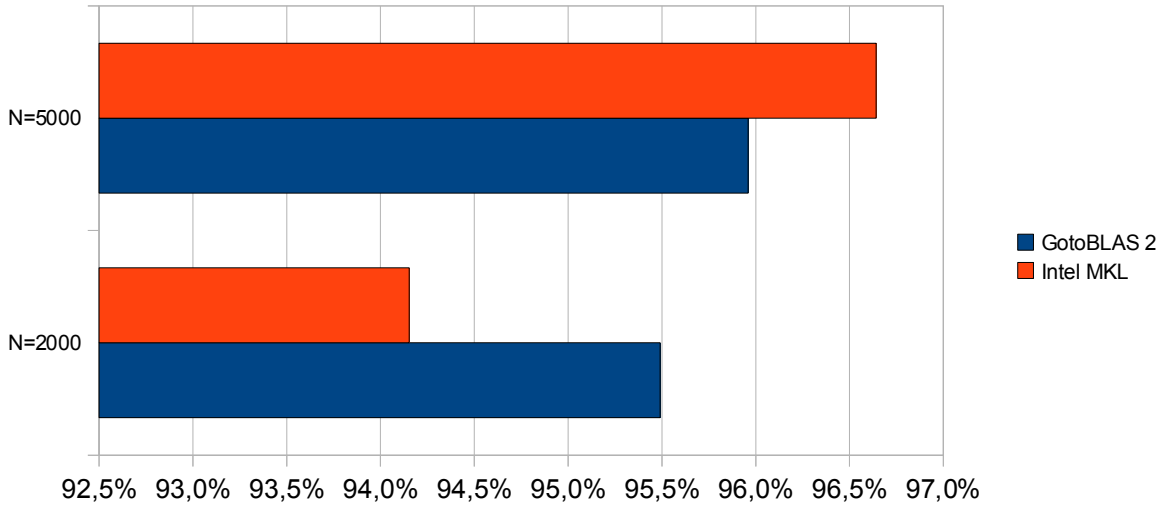
- Multiplication matrice en double précision : operation BLAS3 (dgemm)
  - Plusieurs implementation : netlib, GotoBLAS, MKL ...
  - Version '*optimisée*'

```
T1=time()
do j=1,n
  do i=1,n
    C(i,j) = beta*C(i,j)
  enddo
  do k=1,n
    R0 = alpha*B(k,j)
    do i=1,n
      C(i,j) = C(i,j) + r0*A(i,k)
    enddo
  enddo
enddo
T2=time()
```

$FLOPS = 2N^2(1+N)$
$FLOPS/sec = FLOPS/(T2-T1)$

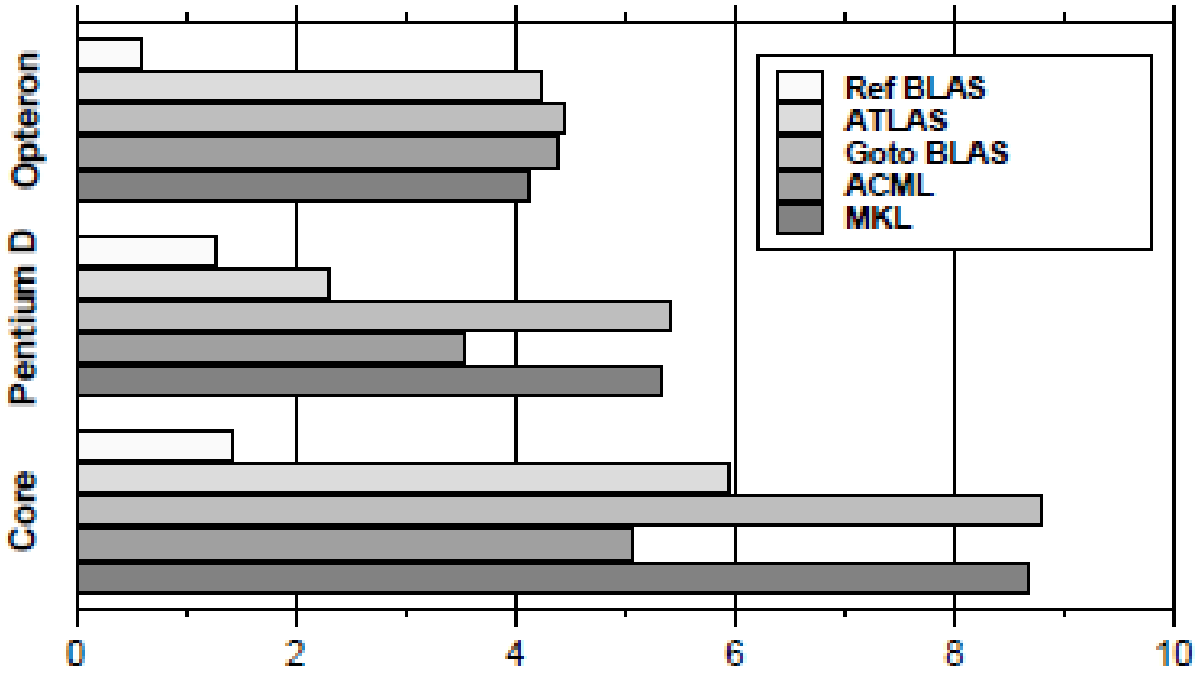
Exemple

Implementation		N=2000	N=5000
Algorithme standard	-00	200	
	-O2 -ip -xSSE4.2	2187	2002
GotoBLAS 2	1.04	11203	11258
Intel MKL	11.1.046	11046	11338



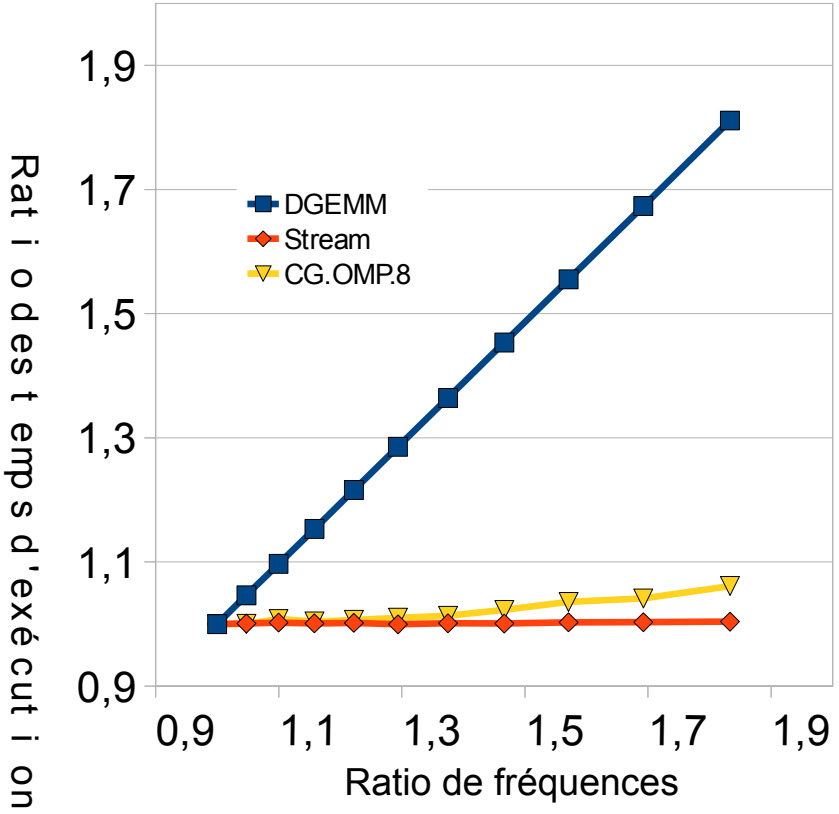
DGEMM

- Les performances des DGEMM dépendent
  - de la librairie (optimisée) de BLAS que l'on utilise
  - De la plateforme sur lequel on travaille



Code CPU bound ?

- Comment savoir si mon code est sensible à la fréquence du CPU ?
  - Faire varier la fréquence (!) et mesuré l'impact.
  - Les processeurs x86_64 supportent ces propriétés pour (Ex. Intel SpeedStep Technology).



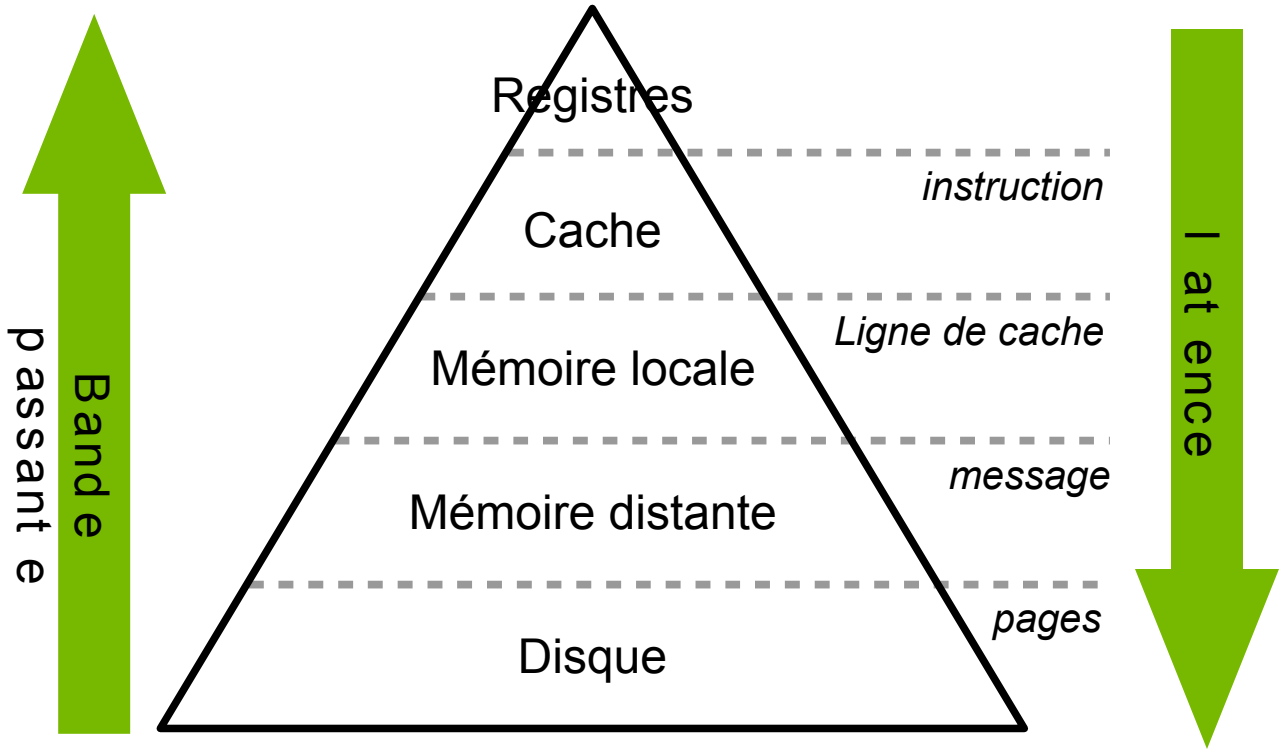
Contenu de la boîte à outils «CPU»

- Il est difficile de trouver un benchmark CPU standard
  - C'est le domaine des benchmarks utilisateurs (applicatifs)
  - C'est fondamentalement ce que voudrons tester les utilisateurs : la capacité du CPU à résoudre leur problème ...
    - Éviter les interférences avec d'autres éléments de la solution
      - → Se restreindre à des benchmarks mono-cœur en mode dépeuplé (1 processus par nœud maximum)
      - → Au plus OpenMP
- Un benchmark d'une « DGEMM optimisée » ou tout autre opération de BLAS n'est pas un benchmark CPU à proprement dit.
  - Peut avoir un sens si ce sont des noyaux couramment utilisés dans les applications utilisateurs.
  - Ça performance dépend du CPU mais surtout de celle de librairie mathématique et de ses optimisations.

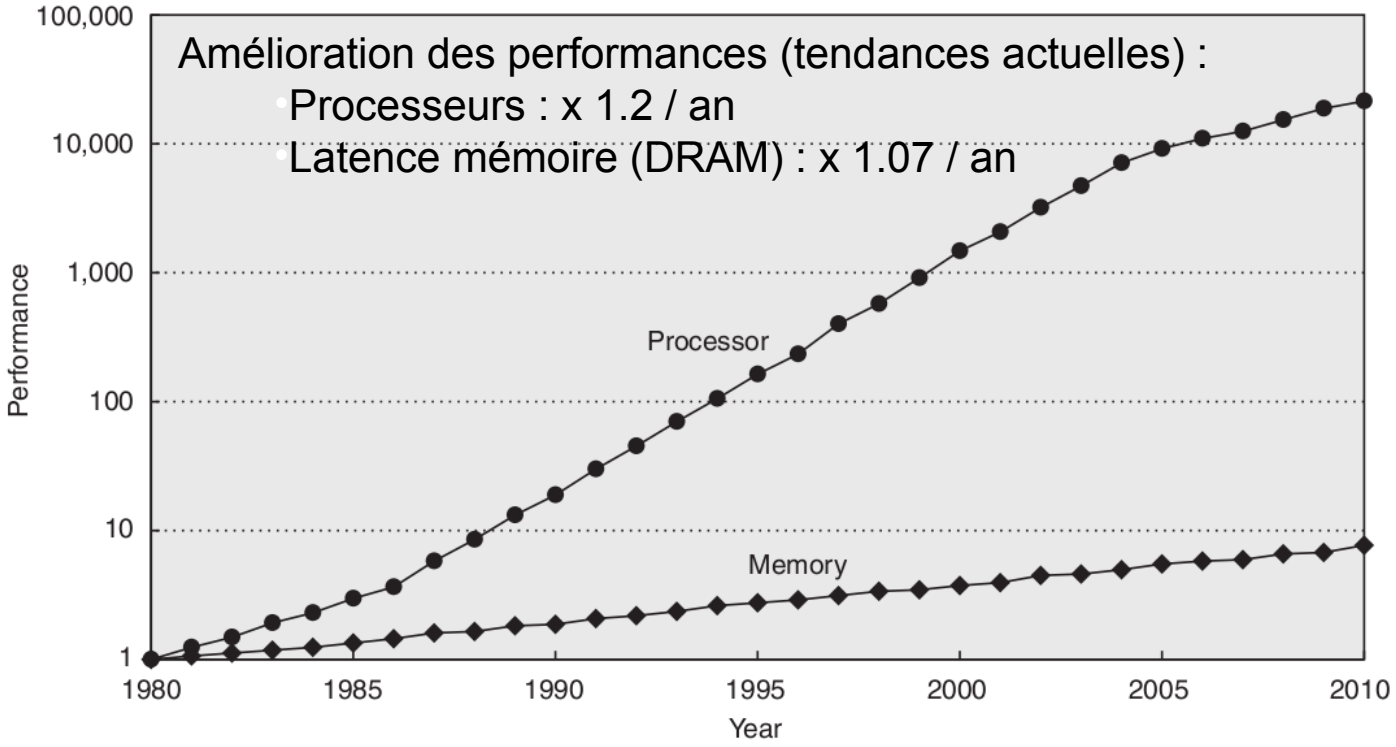
Mémoire

LIBERATE IT

Hiérarchie mémoire



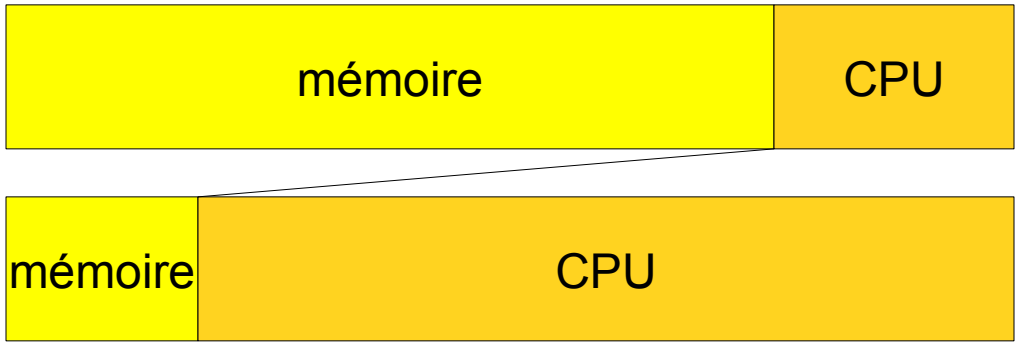
« Hitting the memory wall »



- Bill Wulf and Sally McKee (1994) : « *Hitting the Memory Wall: Implications of the Obvious* »
  - La croissance exponentielle de l'écart entre la performance des mémoires et processeurs va conduire rapidement à ce que les performances des codes ne soient uniquement dominées par les performances des mémoires.

Motivations

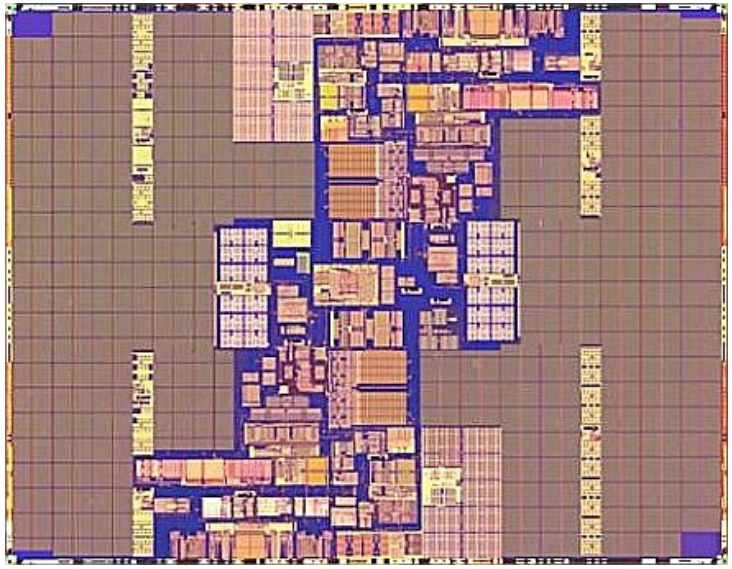
- On cherche un équilibre entre accès mémoires et calculs



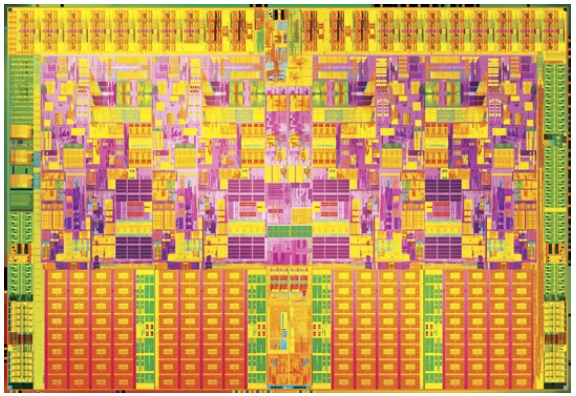
On ne fait que déplacer un bottleneck

- Les codes actuels sont à tendance « memory-bound » plutôt que « CPU-bound »
  - Les bandes passantes ont tendance à augmenter
    - Mais le nombre de cœurs aussi !
  - Augmenter la taille des caches : cher, limité
  - On doit optimiser son code : « *cache-reuse* »
  - Les fréquences à diminuer ou stagner (conso.)

Cache



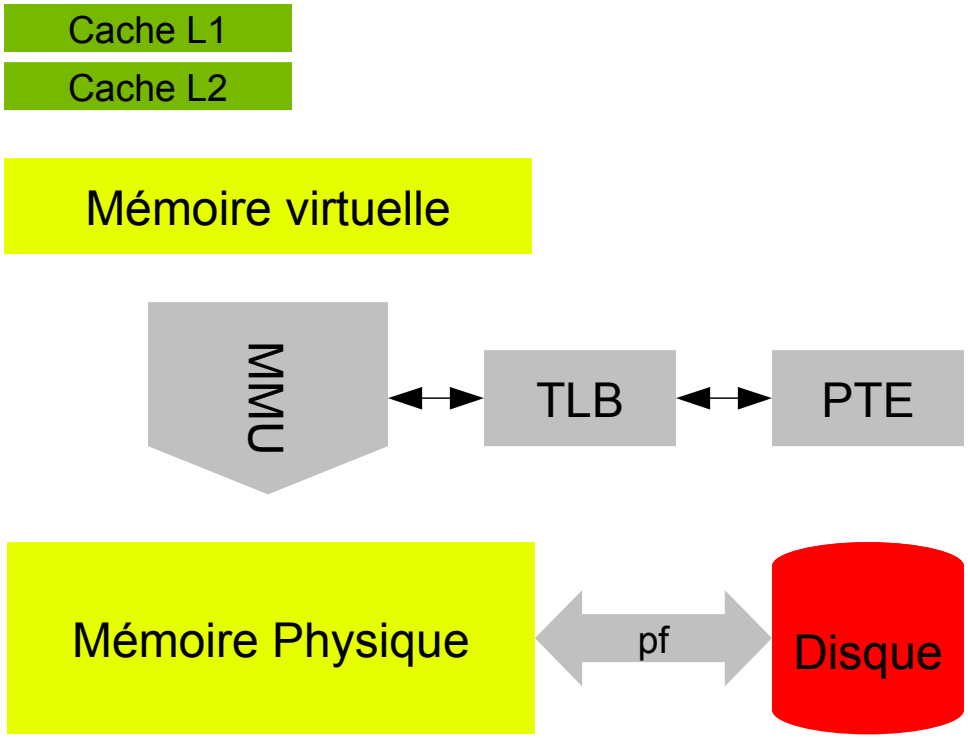
Intel IA64 Montecito (1.6 GHz)



Intel Xeon Nehalem (3.2 GHz)

- Ex. Accès cache : 15 cycles – Accès Mémoire : 200 cycles
  - Accès à 100 éléments, 100 fois
    - Sans cache : 2 000 000 cycles
    - Avec cache : 168 500 cycles
  - > 91 % de gain !

Mémoire virtuelle



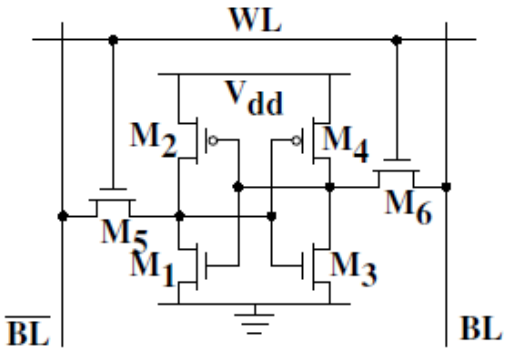
- Clairement, accéder à une donnée en mémoire est une chose complexe. Remonter aux débits et aux latences à partir des seules caractéristiques des barrettes est mission (quasi-)impossible.
  → **Démarche empirique : benchmarks !**

Accès mémoires

- Il est question des **accès mémoires** (les données)
  - Pour l'utilisateur, les métriques pertinentes sont :
    - Le temps d'accès à une donnée : la latence (temps ou nombre de cycles)
    - Le débit (GiB/secondes)
- Quels facteurs influent sur les performances des accès mémoire
  - Localisation : cache ? Mémoire centrale ? Pis, disques !
  - Si les données sont en RAM
  - La manière dont elles sont accédées :
    - Au travers d'un chipset, directement par le processeur ? Via le processeur voisin ?
      - → Les caractéristiques propres de chacun de ces éléments et de leur agencement (population des bancs mémoire par ex.)

SRAM ou DRAM ?

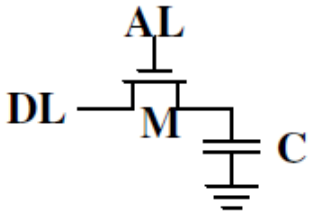
- **SRAM**
-



Static RAM

- Avantage
  - Rapidité
  - Stabilité (pas de rafraîchissement)
- Désavantage
  - Coût
  - Densité

- **DRAM**
-



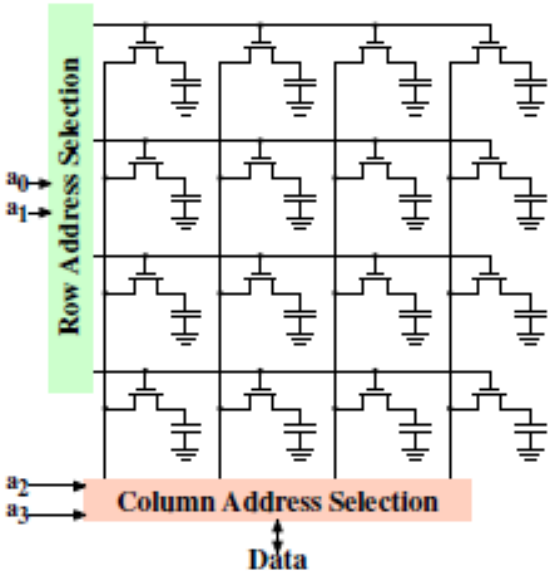
Dynamic RAM

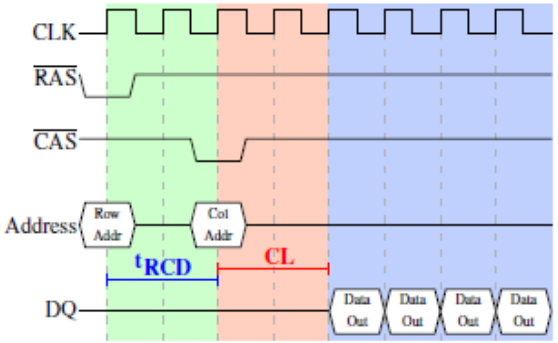
- Avantage
  - Côut
  - Densité
- Désavantage
  - Latence/Débit
  - Besoin d'un rafraichissement périodique

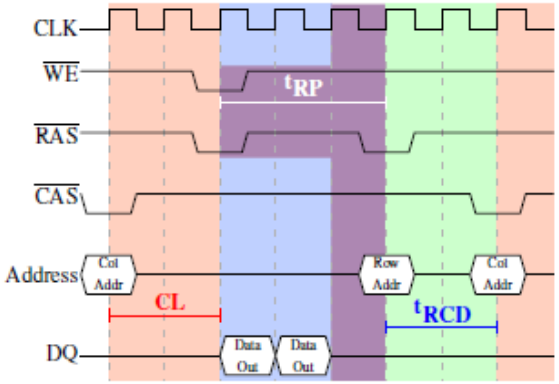
Type de mémoire et agencement

- DIMM : « dual inline memory modules »
  - Assemblage de DRAM
    - SDR, DDR, DDR2, DDR3
- Ses caractéristiques sont dépendantes de celles des DRAM et de leur agencement.
  - Type :
    - Parallèle : DDR2, DDR3
    - Série : FB-DIMM
  - Vitesse : Transferts par secondes (ou débits)
  - Timings : tCL, tRCD, tRP, tRAS
  - Autres
    - Bufferisation (registered, unbuffered)
    - Correction d'erreurs
    - Consommation électrique

Timings



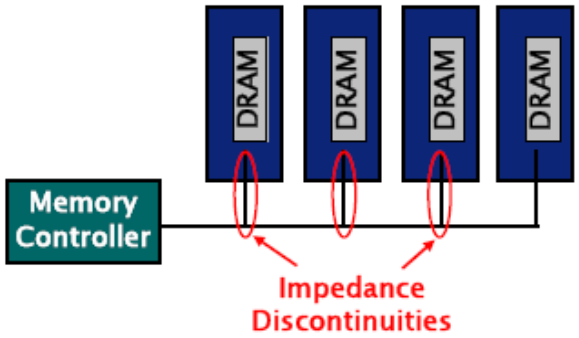


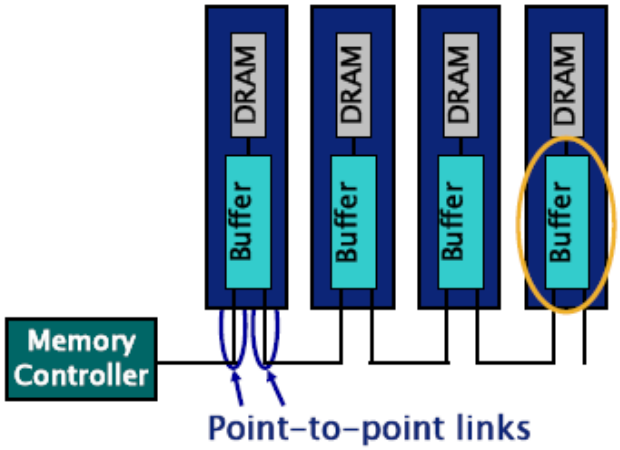


FB-DIMM

- Bus parallèle : « stub-bus topology »
  - Rupture d'impédance → intégrité des données.
    - plus les signaux d'horloge augmentent plus le problème se pose
    - Conduit à réduire le nombre de barrette par canal → diminution de la capacité globale des systèmes !
    - C'est l'inverse de la demande !

→ **FB-DIMM !**



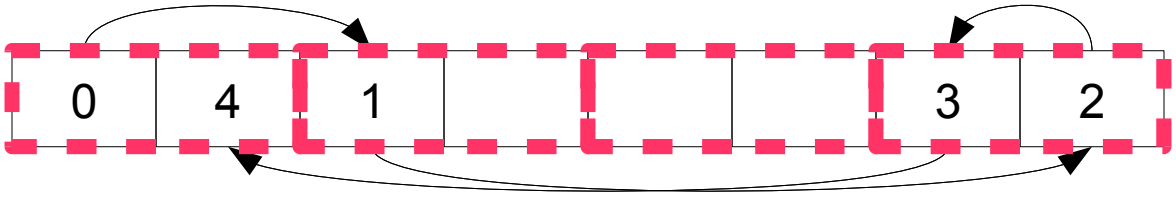


Débit ou latence ?

- Quel est la métrique la plus pertinente ?
  - Ca dépend !
    - Accès aléatoires : latence
      - → Accès au cache
      - → Accès à la mémoire centrale
    - Accès séquentiels : débit

Latence Mémoire

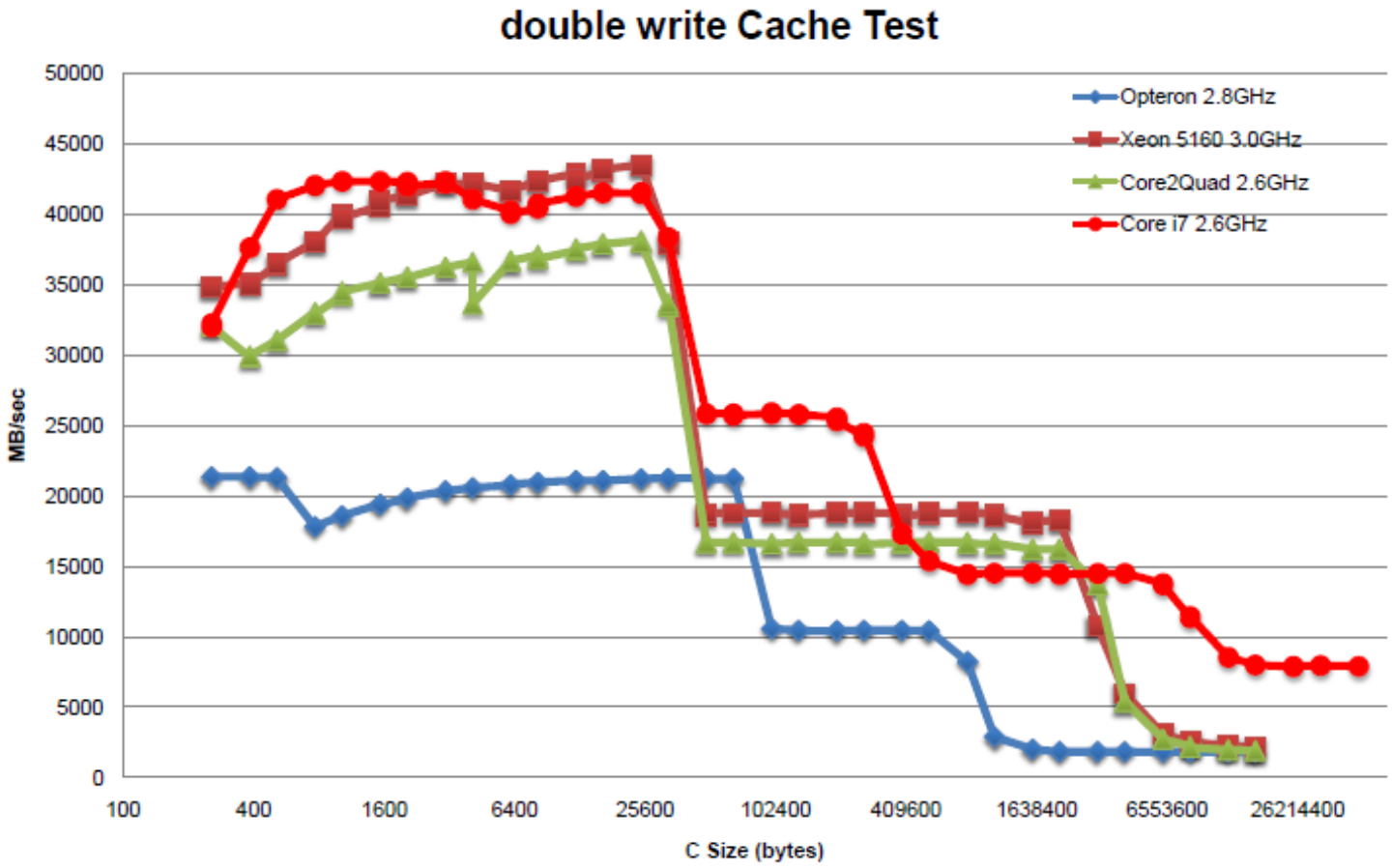
- Accès aléatoire de la mémoire
  - C'est le cas de certains codes applicatifs : multiples niveaux d'indirection (maillage, CFD).
  - L'accès à la mémoire coûte cher : 200 cycles est un bon ordre de grandeur ...
  - Pas forcément évident : il faut « flouer » le « prefetch » (software et hardware)
    - « *chase-pointer strategy* » : naviguer dans une liste chaînée de pointeur.
    - Attention à certaine options d'optim. des compilateurs (prefetch, code mort).
  - Unité : nombre de cycle, nanoseconde, parfois le Gups/sec (« *giga updates per second* »)



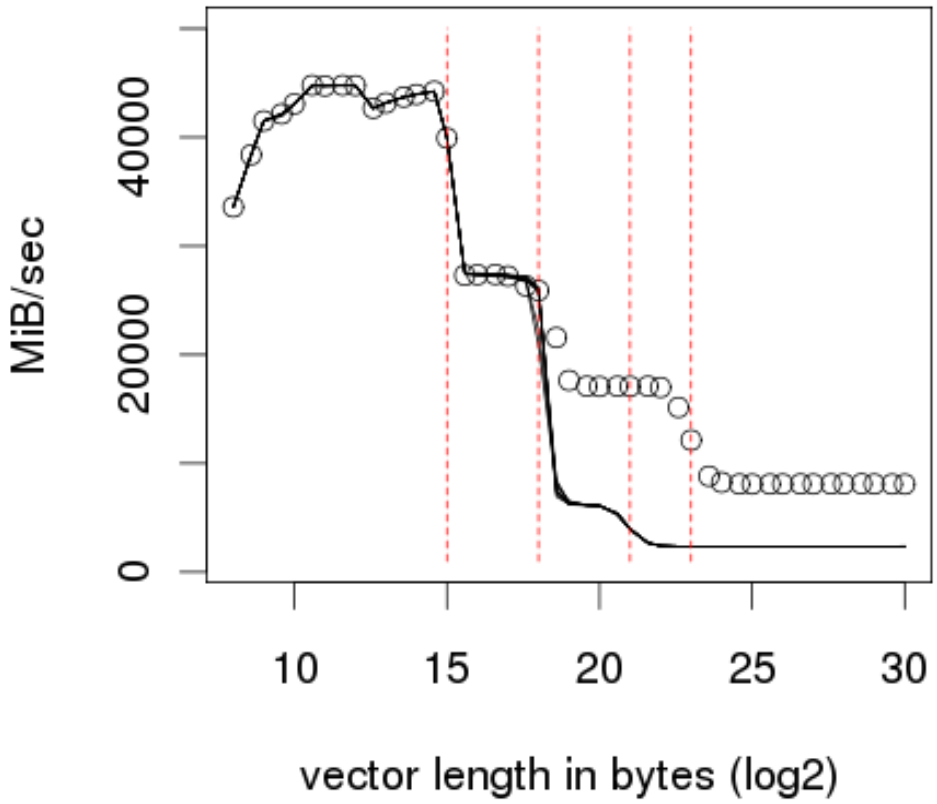
Cachebench (llcbench)

- http://icl.cs.utk.edu/projects/llcbench/
- Mesure de la bande passante pour différente taille de tableau (256 bytes à plusieurs dizaine de MiBytes) et différents « pattern »
  - Découverte de la hiérarchie mémoire
- Portage :
  - Immédiat
  - C
- Execution :
  - Immédiat

Hiérarchie de cache



Hiérarchie de cache



Intel X5570 et mémoire 1066 MHz

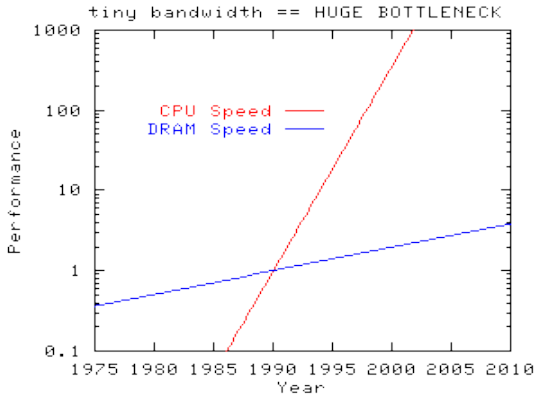
- L1 32KB, L2 256 KB, L3 8M (partagée)

Stream benchmark

- http://www.cs.virginia.edu/stream/

« The STREAM benchmark is a simple synthetic benchmark program that measures sustainable memory bandwidth (in MB/s) and the corresponding computation rate for simple vector kernels. »

- Benchmark accidentel ! Développer pour comprendre les différences de performance entre deux architectures pour un code météo.
- Indispensable dans sa trousse-à-outil
- Accès mémoires séquentiels
  - Portage :
    - → Aucune difficulté
    - → Fortran ou C (OpenMP)
  - Exécution : aucune difficulté
  - Auto-vérifiant (petite taille)

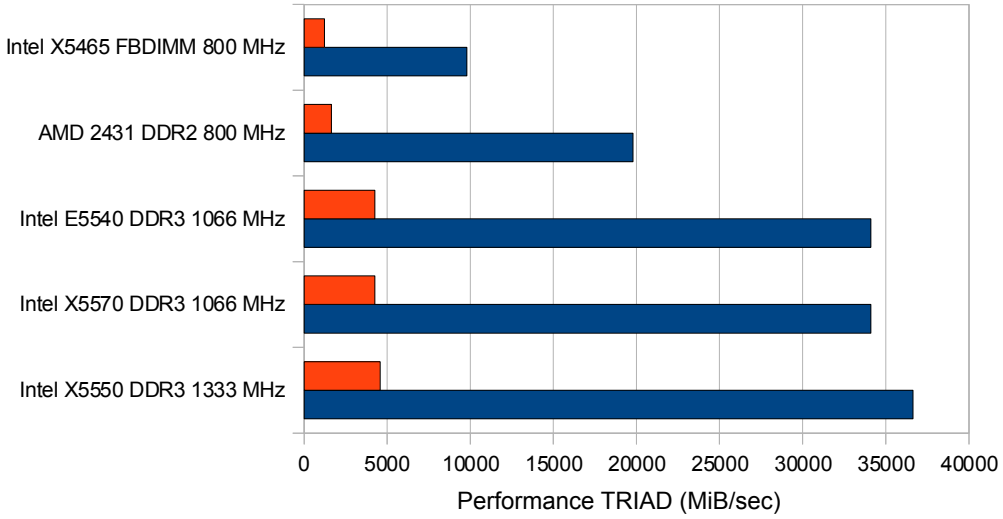


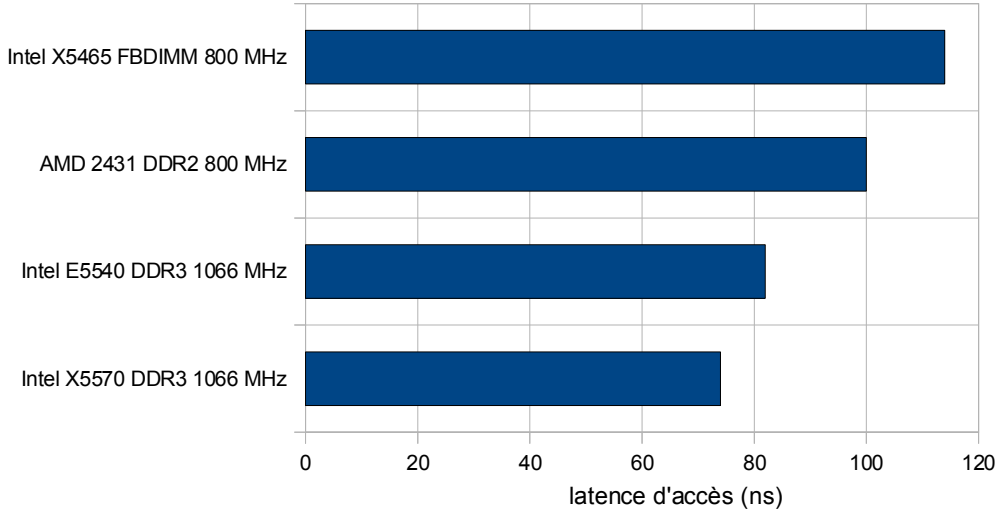
Stream benchmark

nom	noyau	bytes/iter	FLOPS/iter
COPY	a(i) = b(i)	16	0
SCALE	a(i) = q*b(i)	16	1
SUM	a(i) = b(i) + c(i)	24	1
TRIAD	a(i) = b(i) + q*c(i)	24	2

```
-------------------------------------------------------------
STREAM version $Revision: 5.9 $
-------------------------------------------------------------
Array size = 20000000, Offset = 0
Total memory required = 457.8 MB.
Each test is run 10 times, but only
the *best* time for each is used.
-------------------------------------------------------------
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:         3718.0250      0.0862       0.0861       0.0864
Scale:        3772.0257      0.0850       0.0848       0.0852
Add:          4138.4692      0.1162       0.1160       0.1167
Triad:        4149.5237      0.1160       0.1157       0.1164
-------------------------------------------------------------
Solution Validates
-------------------------------------------------------------
```

Comparaison entre système

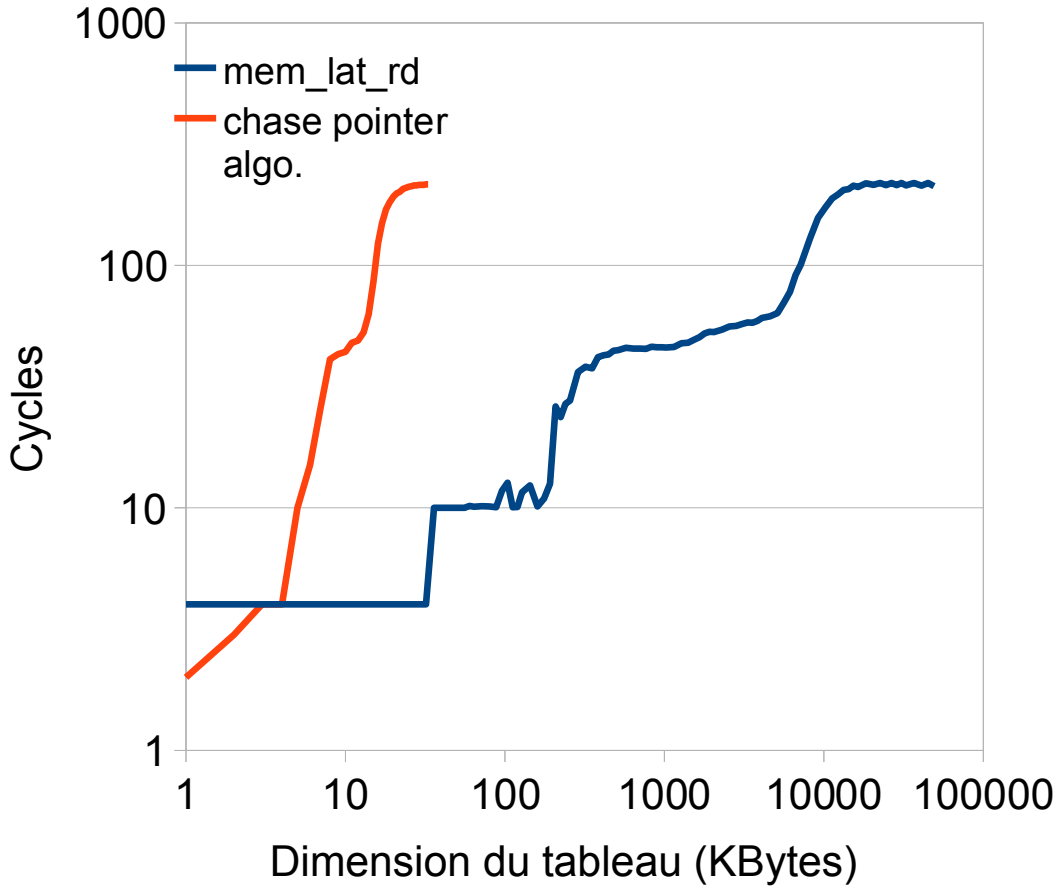




lat_mem_rd (LMBench)

- http://www.bitmover.com/lmbench/get_lmbench.html
- Latence d'accès (en nanosecondes) aux données au travers de la hiérarchie de cache
- Portage
  - C
  - Makefile, immédiat
- Execution
  - Simple

Latence mémoire



- « Chase-pointer algorithm » sur Intel X5570 et mémoire 1066 MHz
  - L1 32KB, L2 256 KB, L3 8M (partagée)

Contenu de la boîte à outils « mémoire »

- Hiérarchie :
  - Cachebench (llcbench)
  - mem_lat_rd (LMBench)
- Latence d'accès :
  - mem_lat_rd
  - HPCC randomAccess (random)
- Débit :
  - Stream

    Permet de détecter d'enventuels problèmes de mémoire
- Autres :
  - Memtest
  - Surpport de mcelog (kernel)