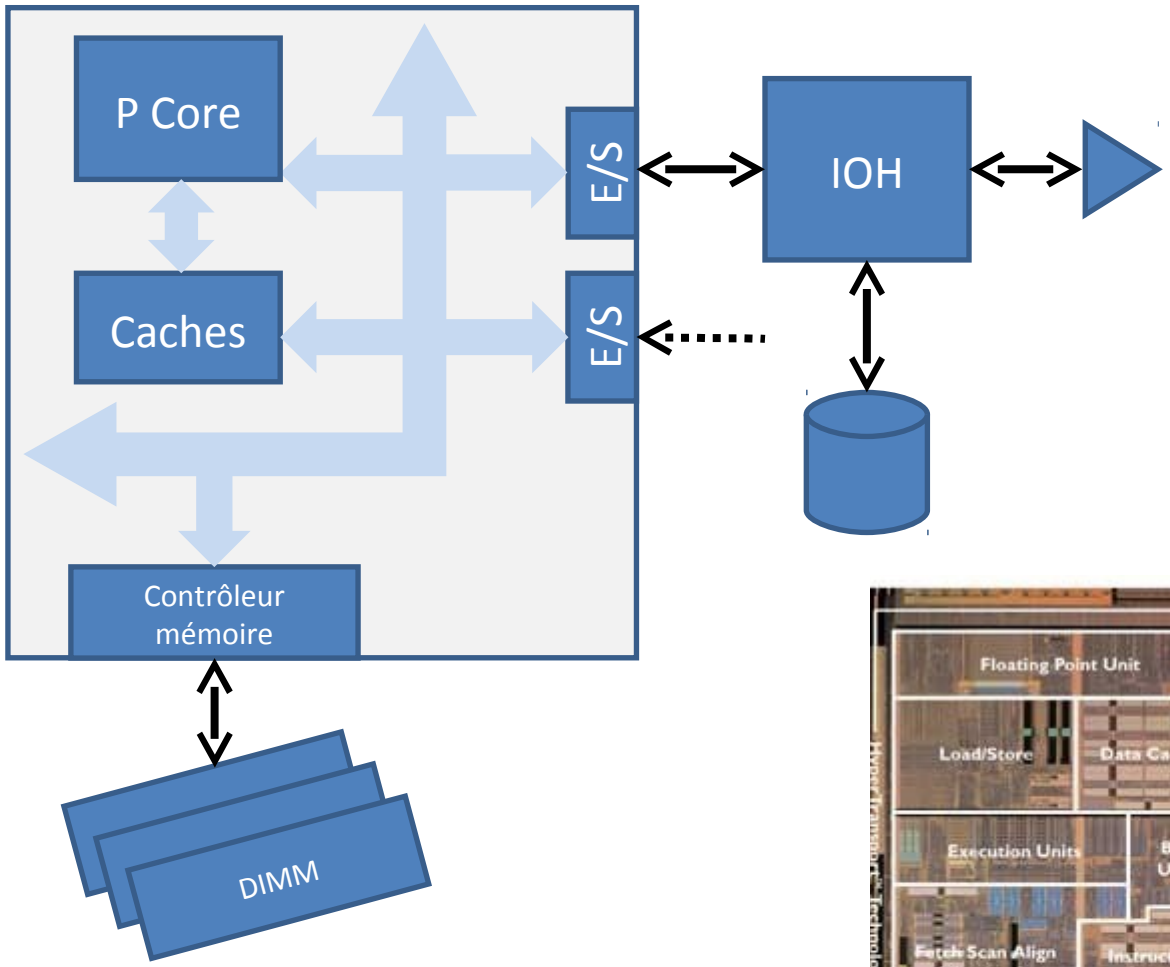
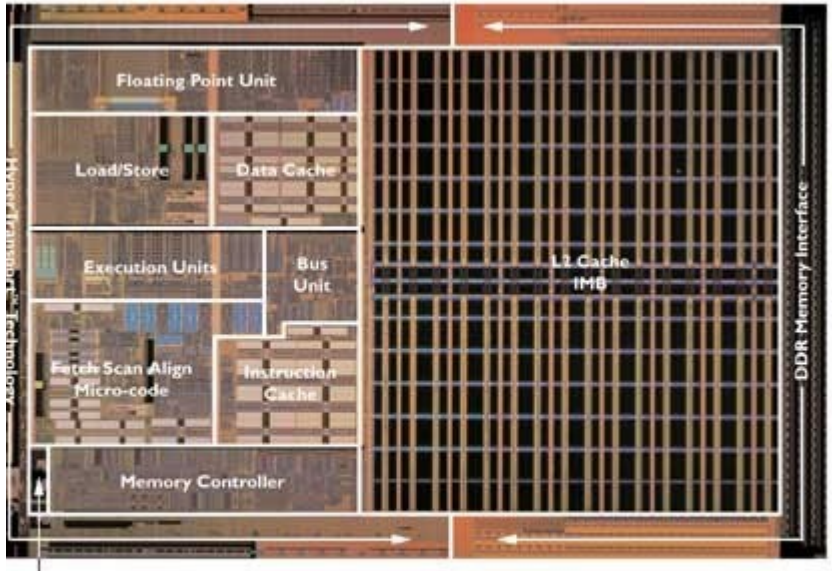
Elements de microarchitecture

1. Processeurs

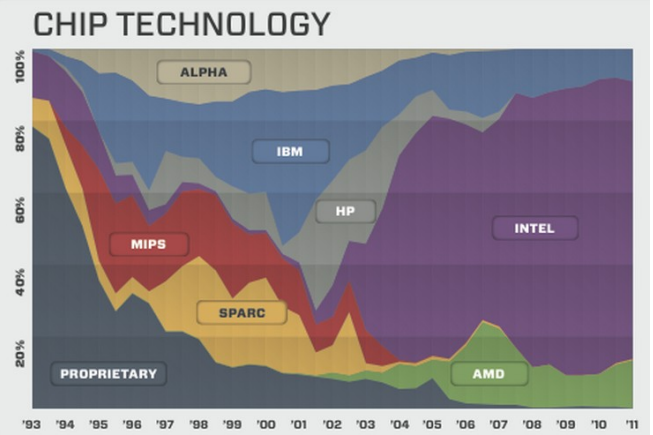
Architecture simplifiée (10.000 feet view)





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 et suivants,
  - 6-coeurs
    - AMD Opteron Istanbul 24xx
    - Intel Westmere-EP,
  - 8-cœurs : Intel Sandy Bridge-EP
  - 10-cœurs : Intel Westmere-EX
  - 16-coeurs : AMD Opterons 6200 (Interlagos)
  - 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)
  - L'hyperthreading vient compliquer les choses !
- 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

Processeurs - caractéristiques

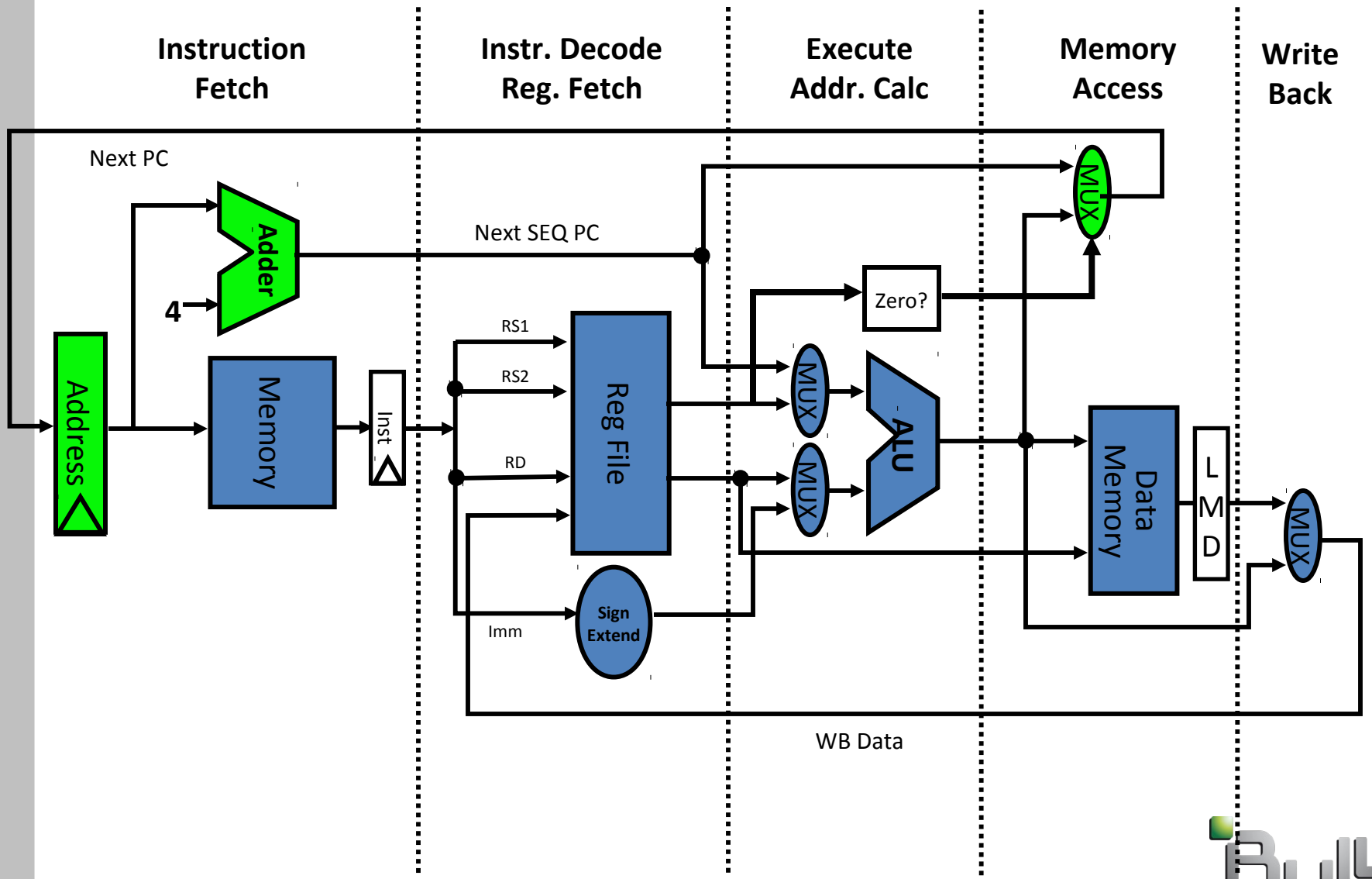
- La largeur des données entières (arithmétique) supportée
- Fréquence (unité : GHz)
- Cache : niveau, taille, organisation, associativité
- Gestion des accès à la mémoire centrale
  - Bus, liens (contrôleur intégrés) → vitesse, débit (GT/sec, GiB/sec)
  - Translation d'adresses, TLB ...
- Micro-architecture
  - Caractéris
    - Largeu
    - Nombre
  - 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

Un processeur ne se réduit pas à sa seule fréquence !

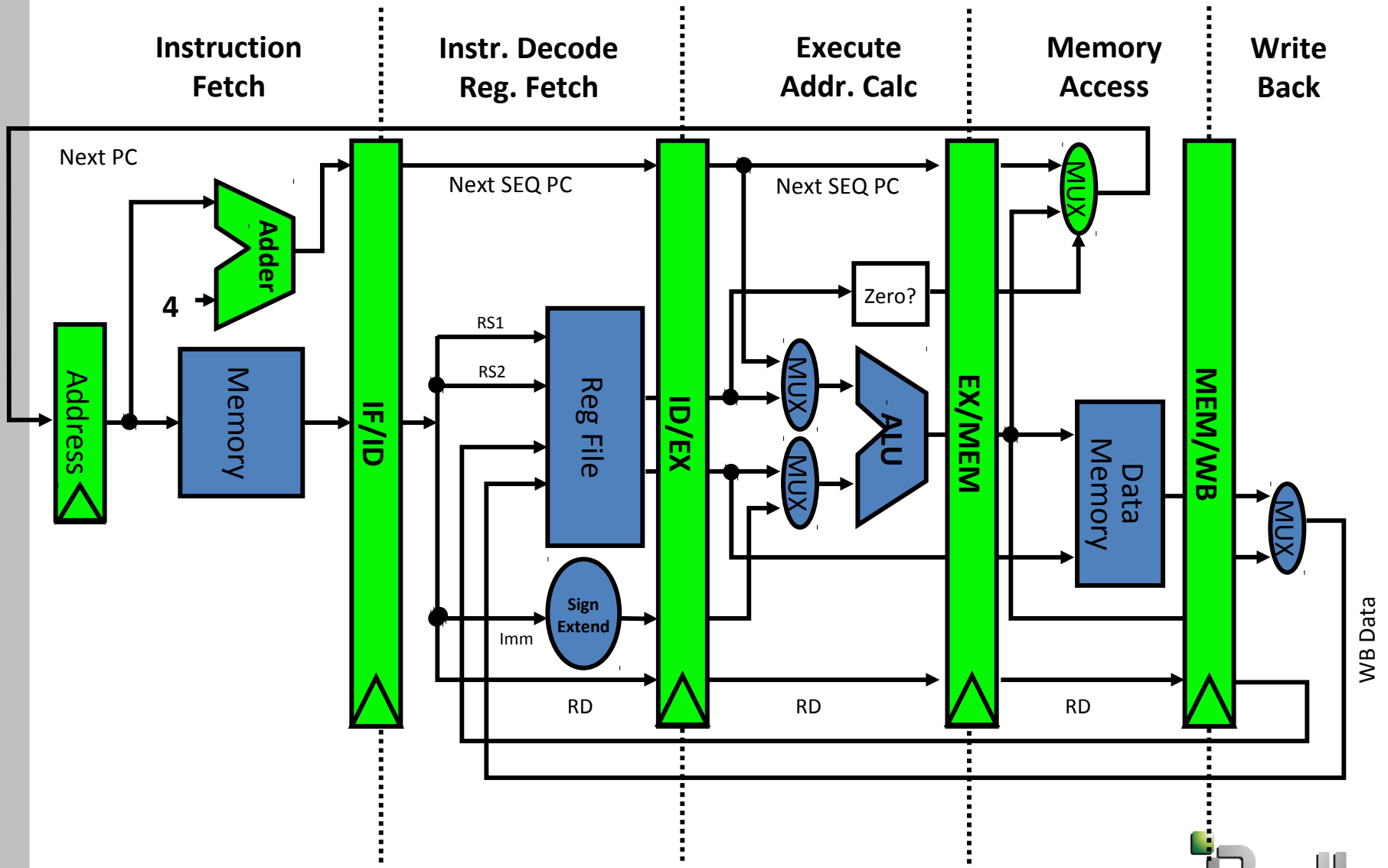
Need for speed ?

- Diminuer le “CPI” en exploitant l’“ILP”
  - Pipelining
    - Simple ou super- ou hyper- pipelinning
    - Instruction : latence (latency) et debit (troughput)
  - Superscalar
    - Multiplication des unités fonctionnelles (FU ou Port)
  - Execution out-of-order (OOO)
    - “Reordering” et “Register Renaming”
  - Prediction de branches et Execution speculative
  - “Hyperthreading”
- Parallèlisme de données (DLP) : SIMD (vectorisation)
  - NEON, 3DNow!, Altivitec, MMX, SSE(x.y), AVX …

Pipeline | Exemple, le MIPS (1)



Pipeline | Exemple, le MIPS (2)



Pipeline | stalls

- Le pipeline peut se mettre à “buller”
  - Bulles (“bubbles”), “breaks ou pipeline “Stalls” sous certaines conditions (“hazards”):
    - Structural hazards: conflit de ressources
      - → Solution: out-of-ordering
    - Data hazards: possible quand une instruction depend du résultat de la précédente.
      - → 3 types
        - RAW: “Read after Write” ou “Flow dependency”

```
ADD     r0,r1
MUL     r2,r0
```

        - WAR: “Write after Read” ou “Anti-dependency”

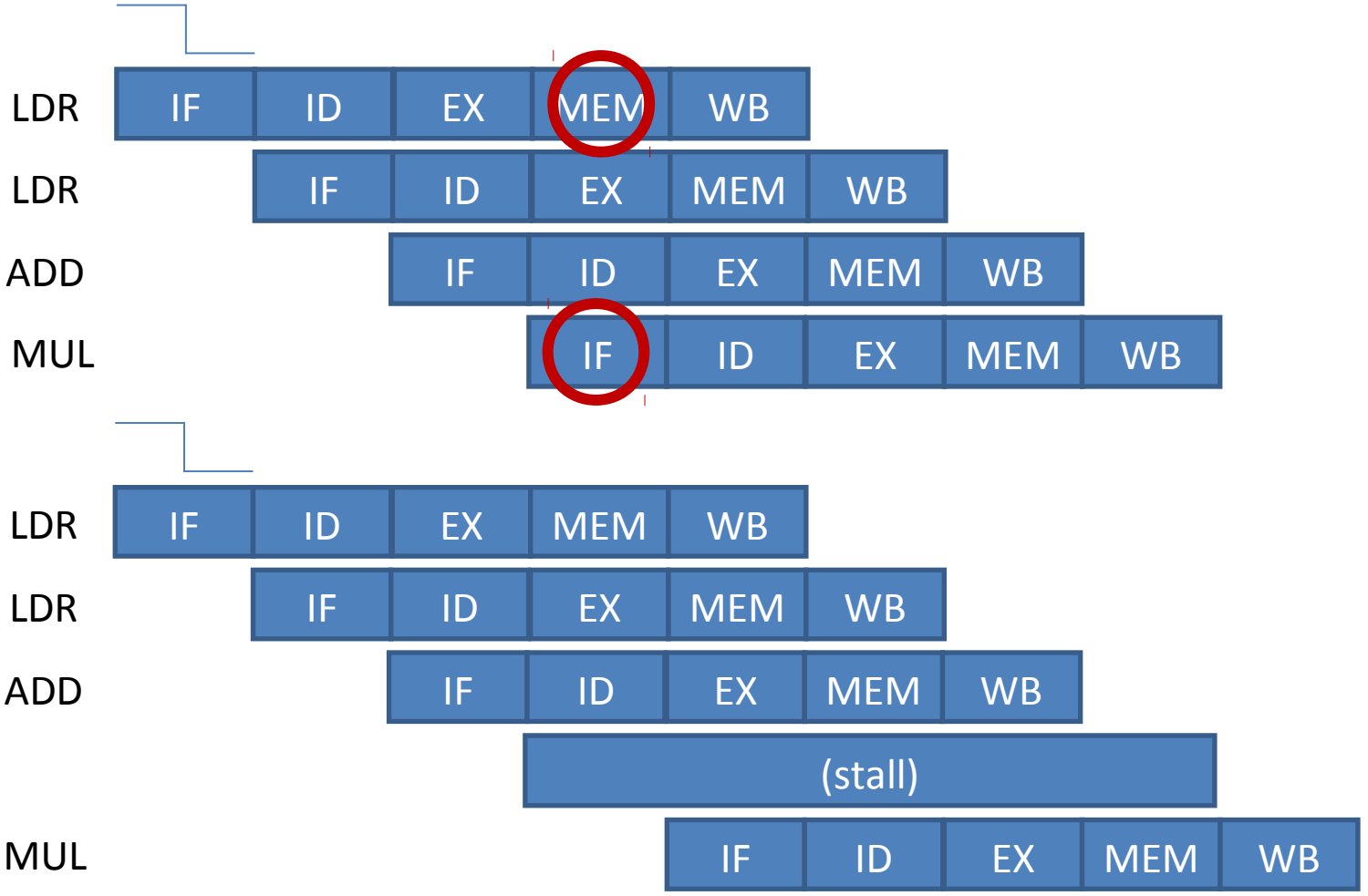
```
ADD     r4,r1,r3
MUL     r1,r2,r3
```

        - WAW: “Write after Write”

```
ADD     r1,r4,r3
MUL     r1,r2,r3
```

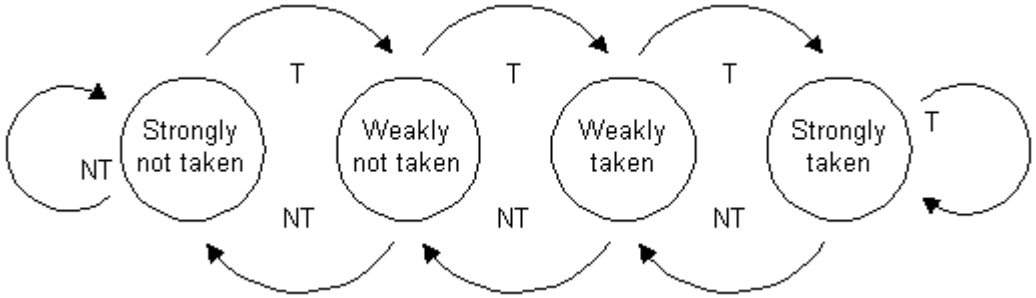
      - → Solutions: Register forwarding, Register renaming, re-ordering, insertion de bulles …
      - → **Instruction reordering**
    - Control hazards: branches, ou autres instructions modifiant la valeur du PC
      - → Solution: prediction de branch

Pipeline | exemple de “péril structurel”



Prédiction de branches et exécution spéculative

- Eviter la “famine” au niveau du pipeline
  - Prédire l’avenir avec la connaissance du passé
  - Spéculer et exécuter ce qu’on pense le plus à même de se passer

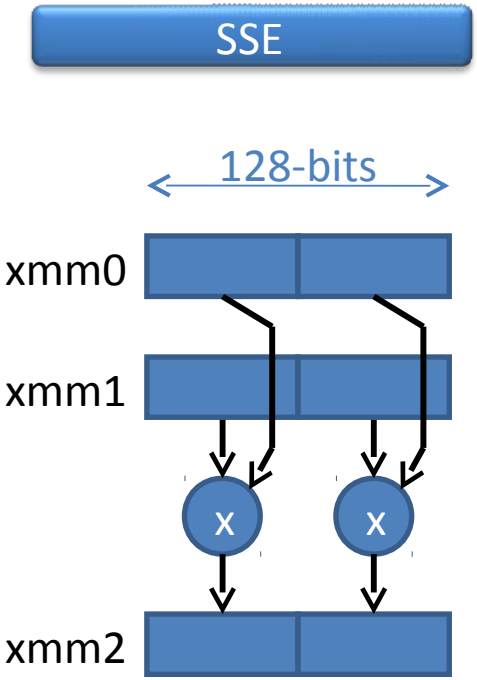


- Le coût d’une mauvaise prédiction est important (vidange et remplissage/réamorçage du pipeline)
- Le coût d’une branche est de manière générale important
- Attention aux “if”
  - Ne pas abuser du “if”
    - Réduire leur utilisation au strict minimum
      - → Réserver au débogage
      - → Quand c’est possible, remplacer par des directives de pré-proccesing `#if…#else…#endif`
    - Cf. programmation du kernel: fonctions courtes, “outlinés”
    - Certaines architectures (ARM) définissent des instructions “conditionnelles”

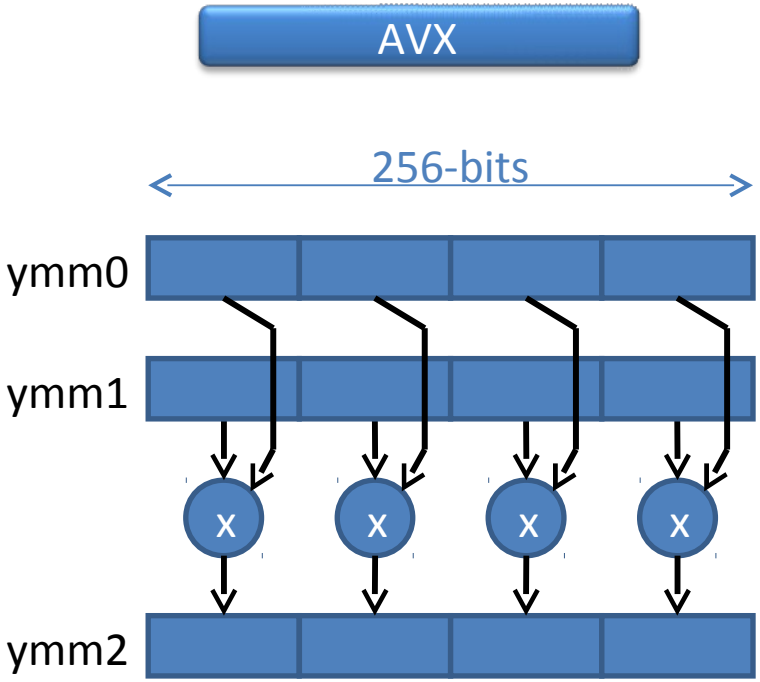
Vectorisation

- C’est un des moyens d’augmenter le nombre d’opérations (flottantes et entières) par cycles en travaillant sur le parallèlisme des données
  - ET pas le nombre d’instructions ;-)
- Jeux d’instructions spécifiques, extension de l’ISA (registres, instructions)
  - PowerPC : Altivitec
  - X86 : 3DNow!, MMX, SSE, AVX, …
  - ARM : Neon
- Principal paramètre : la profondeur du vecteur
  - 128-bits pour SSE (soit 4 float ou 2 double)
  - 256-bits pour AVX (soit 8 float ou 4 double)
  - 512-bits pour AVX2 (futur)
- Le compilateur peut (dans les phases d’optimisation) tenter lui-même d’extraire le parallélisme des données (boucle) et générer ces instructions
  - Necessite de “bonnes pratiques”
- Si l’on est bornée sur la bande passante, c’est sans effet (voire …)
- Faire un run sans vectorisation peut-être intéressant
- L’alignement est important

Vectorisation | Exemple de SSE et AVX

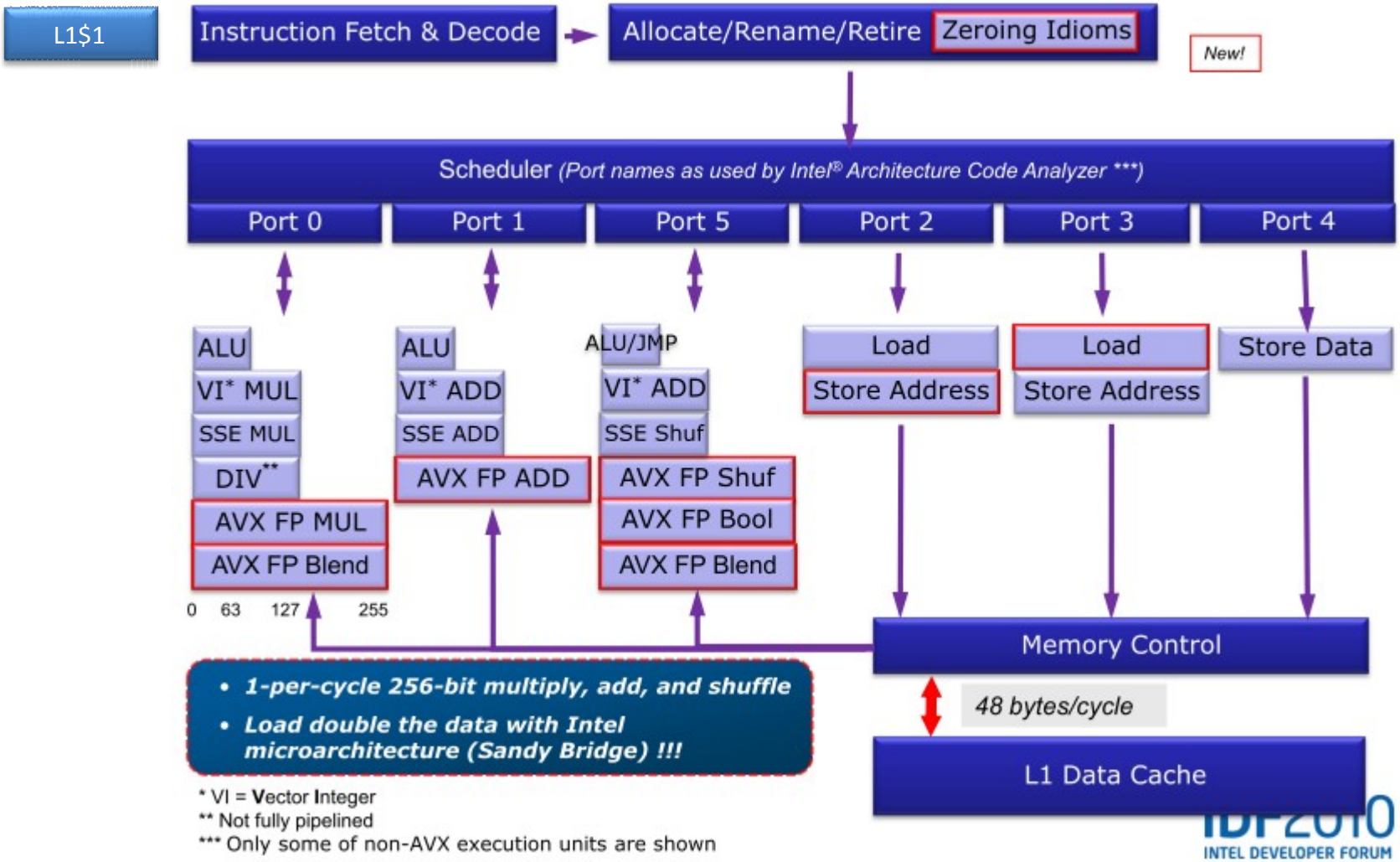


= 2 opérations simultanées



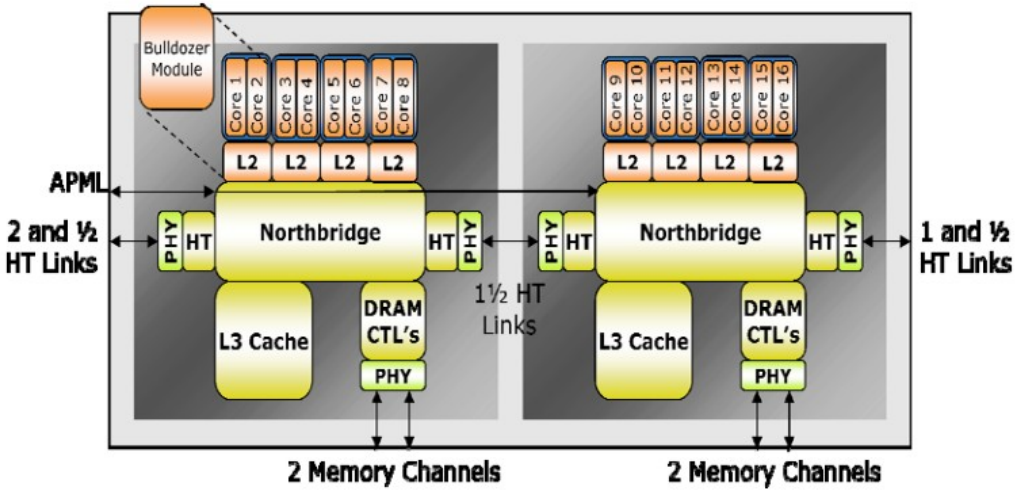
= 4 opérations simultanées

Intel Sandy Bridge EP micro-architecture

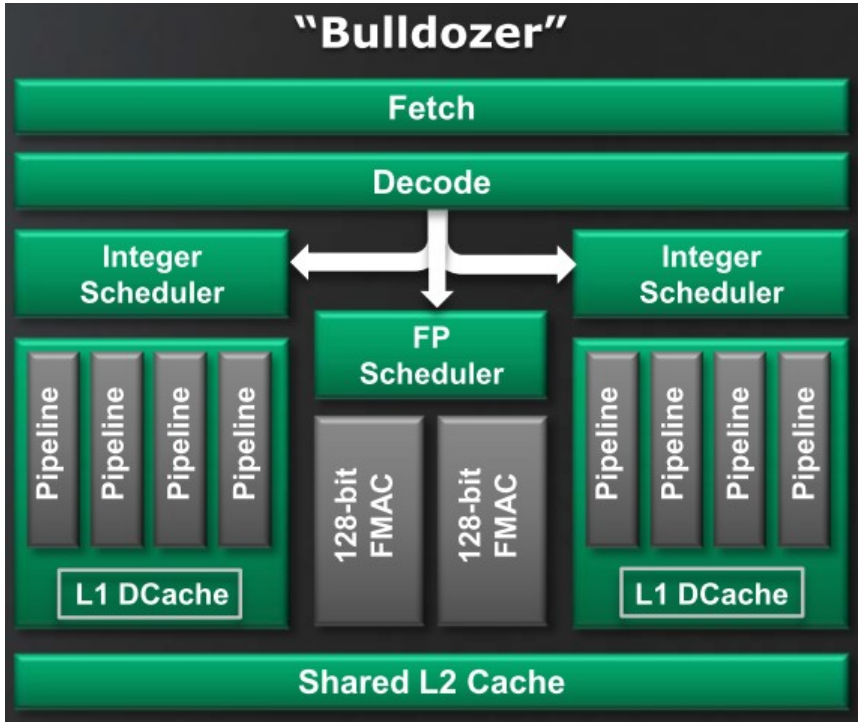


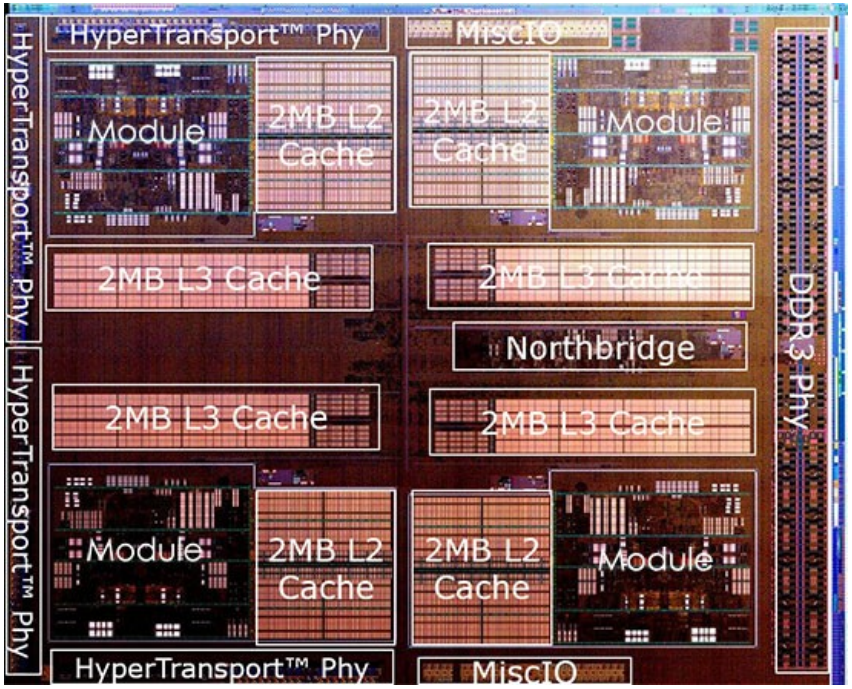
G34 package

- Un système “hiérarchique”
  - Une platforme est composée de 2 ou 4 processeur “G34” (processor packages)
  - Chaque “G34” est composé de 2 chip “Interlagos” (ou die)
  - Chaque die “Interlagos” est composé de 4, 6 ou 8 modules “bullodozer”
  - Chaque “Bulldozer” comprends 2 coeurs “entier” et partage la même FPU



“Bulldozer” modules





Processeurs

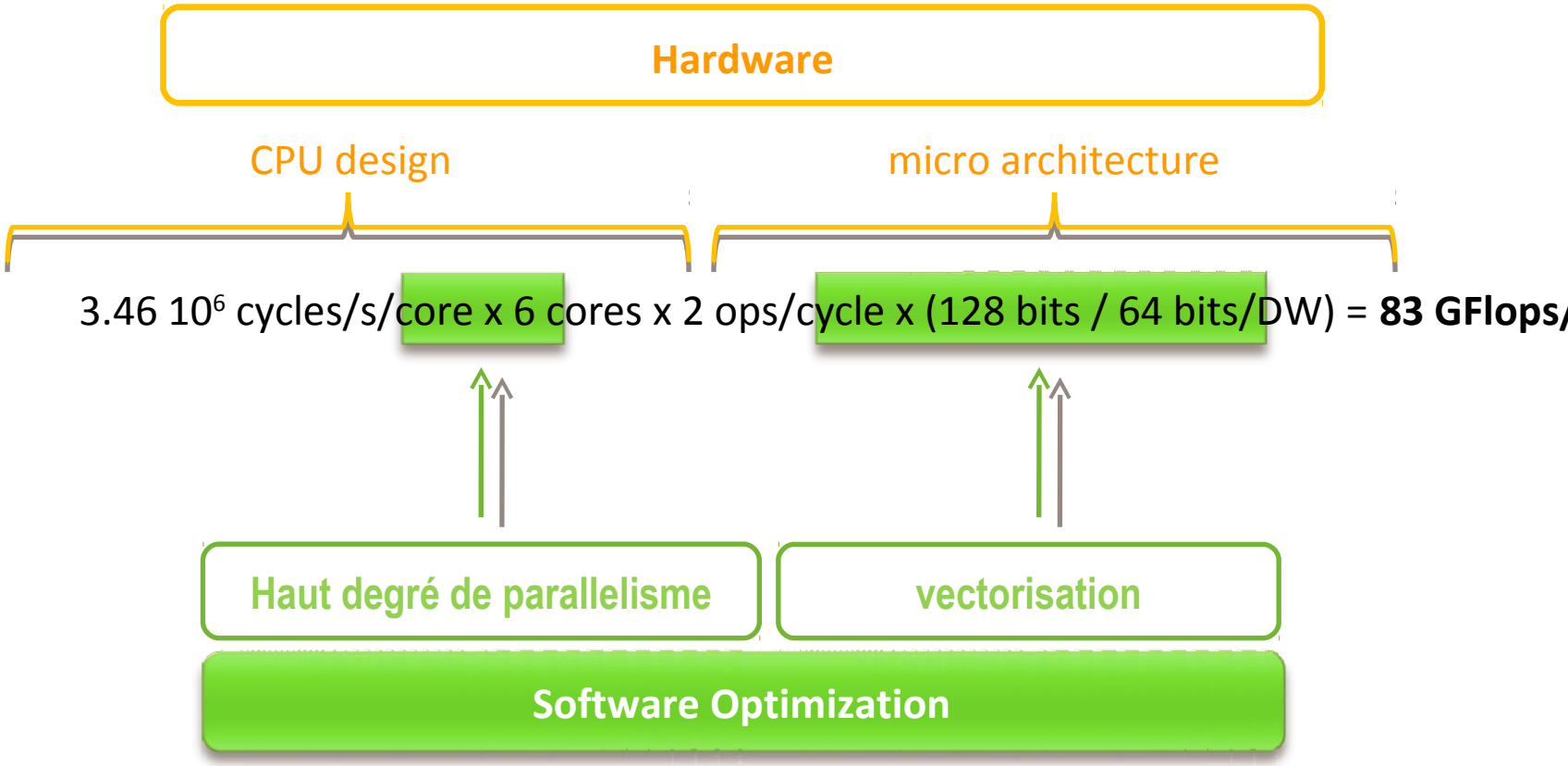
- Capacité à effectuer des calculs
  - Nombre d'instructions traités par cycles
    - Généralement accessible par un compteur du processeur
    - Certains noyaux permettent via des drivers particuliers de remonter cette information dans l'espace utilisateur
  - **Dans le cadre d'un calcul scientifique, c'est (en général) 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 »

Qu’est-ce qui fait la puissance d’un processeur ?

- Performance en double précision d’un Intel ® Westmere X5690



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)

- Attention :
  - On entend généralement par op. flottante les additions et les multiplications ; les divisions font partis des opérations complexes pour un processeur (et donc très coûteuses).
  - Le timing est parfois une opération compliquée, surtout sur les très petites durées (moteur ooo).

Comptage empirique | mesurer le temps

- Différentes “sources de temps” sont disponibles
  - appel système gettimeofday()
  - Time Stamp Counter (TSC)
    - compteur 64 bits, compte le nombre de cycles depuis le reset
    - Attention: la façon exacte dont il est incrémenté dépend du processeur
    - peut être lu avec l'instruction `rdtsc`
      - → Disponible pour l’utilisateur
    - Problemes :
      - → La fonction n’est pas **sérialisante** (out-of-ordering)
        - Solution : faire précéder la `rdtsc` d’un appel à l’instruction cpuid
      - → Cache instruction miss …
      - → Overhead `rtdsc+cpuid` : >200 cycles ! Il faut le prendre en compte …
  - Autres compteurs matériels
    - Précis
    - Nécessite d’utiliser les instruction privilégiées `rdmsr` et `wrmsr` (“ring 0”).
      - → Module d’interfaçage avec le noyau, au moins pour “enabler” les compteurs de performances …

Comptage empirique | mesurer le temps

- Différentes “sources de temps” sont disponibles
  - appel système gettimeofday()
  - Time Stamp Counter (TSC)
    - compteur 64 bits, compte le nombre de cycles depuis le reset
    - Attention: la façon exacte dont il est incrémenté dépend du processeur

```
cpuid
rdtsc                    ; read time stamp
mov     time, eax        ; move counter into variable
fdiv                     ; floating-point divide
cpuid
rdtsc                    ; read time stamp
sub     eax, time        ; find the differenc
```

  - Autres compteurs matériels
    - Précis
    - Nécessite d’utiliser les instruction privilégiées `rdmsr` et `wrmsr` (“ring 0”).
      - → Module d’interfaçage avec le noyau, au moins pour “enabler” les compteurs de performances …

Comptage empirique | mesurer le temps

```
#define SERIALIZE() { \
        asm volatile ("xorl %%eax,%%eax":::"eax");\
        asm volatile ("cpuid":::"eax","ebx","ecx","edx"); \
}

typedef unsigned long long ticks;

ticks getticks(void)
{
        unsigned a, d;
        SERIALIZE();
        asm volatile("rdtsc" : "=a" (a), "=d" (d));
        return ((ticks)a) | (((ticks)d) << 32);
}
```

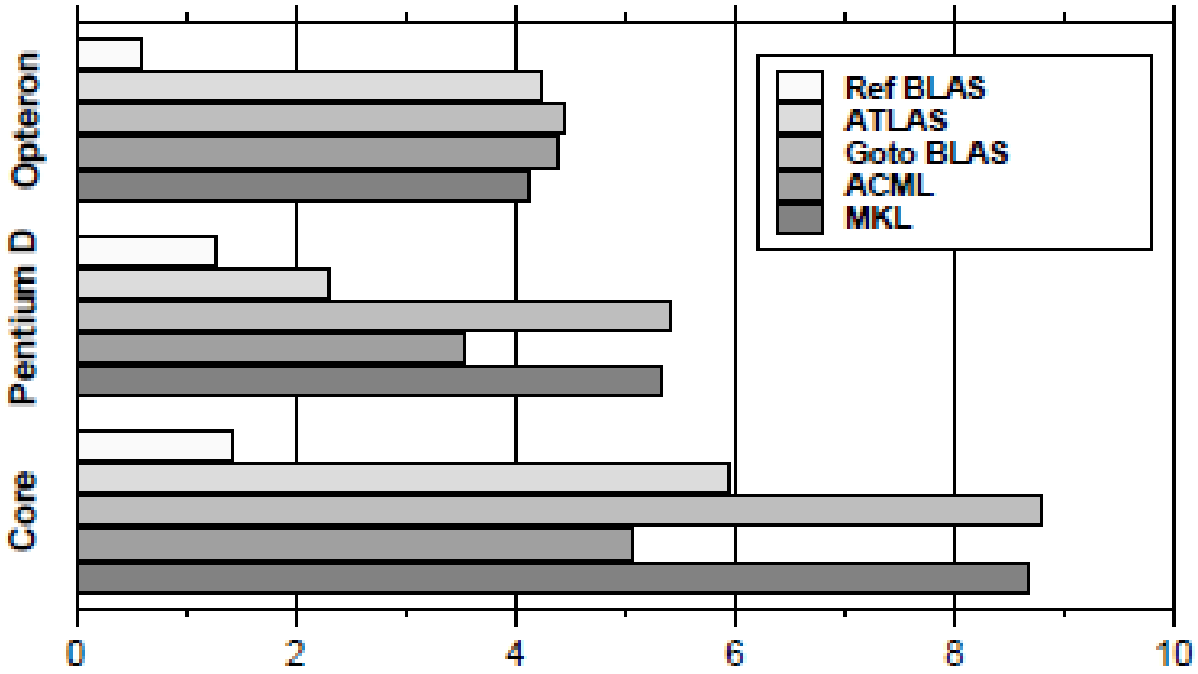
Comptage empirique | mesurer le temps

```
#define SERIALIZE() { \
        asm volatile ("xorl %%eax,%%eax":::"eax");\
        asm volatile ("cpuid":::"eax","ebx","ecx","edx"); \
}

typedef unsigned long long ticks;

ticks getticks(void)
{
        unsigned
        SERIALIZE
        asm volat
        return ((
}
```

```
int main () {
        double  t0,t1;
        ticks   T0,T1;
        T0=getticks();
        t0=dtime();
        sleep(1);
        t1=dtime();
        T1=getticks();
        printf("%.1f\n",(double)(T1-T0)/(t1-t0)*1e-6) ;

        return 0;
}
```

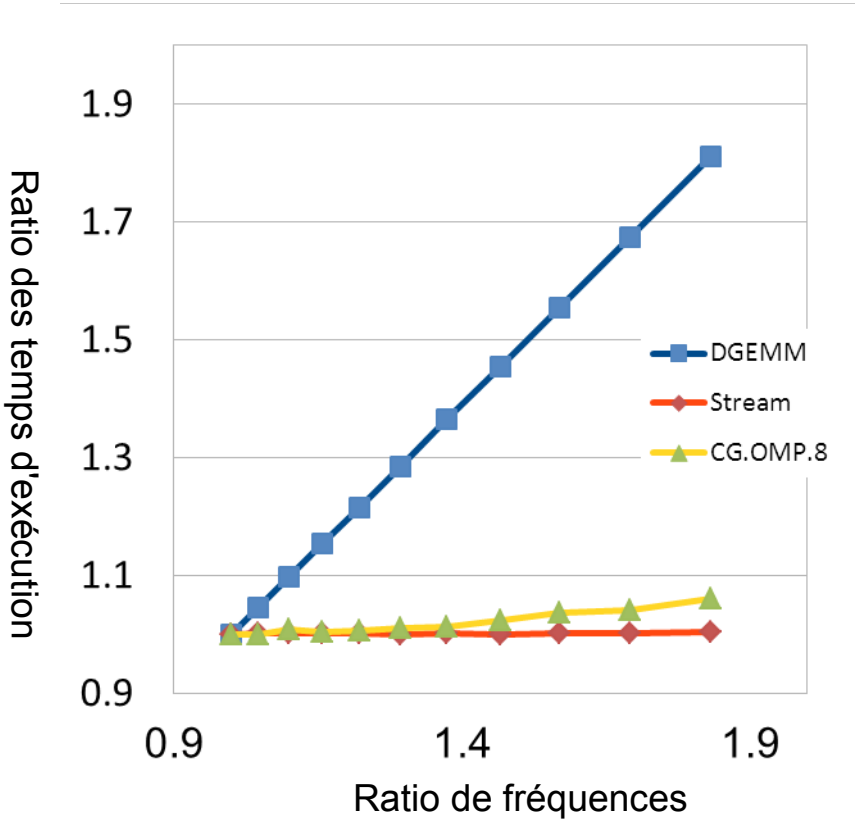
DGEMM

- 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).



mubench

- http://mubench.sourceforge.net/

mubench

« It measures latency and throughput for each individual instruction, as well as the throughput of arbitrary instruction mixes. (…) mubench fully supports all SIMD instruction sets for the x86, including SSSE3, SSE3, SSE2, SSE, MMX, MMX Ext, 3DNow! and 3DNow! Ext. »

- Portage :
  - Immédiat
  - C
- Execution :
  - Immédiat

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 (L3) 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.