Calcul – Cours 1: Introduction, Définition et Matériel

Jonathan Rouzaud-Cornabas

LIRIS / Insa de Lyon – Inria Beagle

Cours inspiré de ceux de Frédéric Desprez (DR Inria – Grenoble)

Définitions

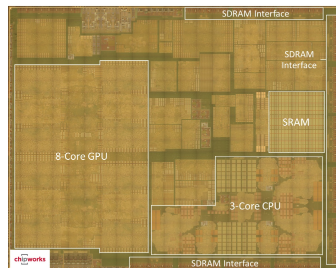
- Parallèle VS Distribué
- Compute intensive VS data intensive VS Big Data
- High Performance Computing (HPC) VS High Throughput Computing (HTC)
- Durée de vie d'un code VS durée de vie d'une machine

- Parallel Programming – For Multicore and Cluster System (T. Rauber, G. Rünger)
- Sourcebook of Parallel Computing (J.J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy,L. Torczon, A. White)
- Parallel Algorithms (H. Casanova, A. Legrand, Y. Robert)
- Parallel Computer Architecture (D.E. Culler, J. Pal Singh)
- Advanced Parallel Architecture - Parallelism, Scalability, Programmability (K. Hwang)
- Super-ordinateurs – Aux extrêmes du calcul (Numéro spécial de la recherche, Nov. 2011)
- Cours en ligne :
  - Why parallel, why now, Dr Clay Breshears, Intel
  - Architecture et Système des Calculateurs Parallèles, F. Pellegrini, LaBRI
  - Applications of Parallel Computers, J. Demmel, U.C. Berkeley CS267
  - K. Yelick
  - U.E. Parallélisme (Lyon 1), Frédéric Desprez

Pourquoi paralléliser et Loi de Moore

Pourquoi un cours sur le parallélisme ?

Le parallélisme est partout !



Mais pourquoi en a-t-on besoin ?

- Résoudre des problèmes plus rapidement
  - Traiter plus de requêtes à la seconde (exemple Google)
  - Améliorer les temps de réponse des applications intéractives
- Obtenir des meilleurs résultats dans le même temps
  - Rafiner les modèles (exemple Météo France)
  - Compliquer les modèles (multi-échelles)
- Traiter des problèmes de plus grande taille
  - Maillages plus important
  - Données des réseaux de capteurs, des réseaux sociaux, ...
  - Recherche dans les pages web (Google)
  - LHC (CERN)

Pourquoi ne pas accélérer les processeurs ?

- Faisons un calcul rapide, pour avoir un processeur séquentiel à 1Tflop
  - Les données doivent aller de la mémoire au CPU (distance r)
  - Récupérer une donnée par cycle (10^{12} fois par seconde) à la vitesse de la lumière ($c = 299,792,458m/s \approx 3 \times 10^8 m/s$)
  - Donc $r < \frac{c}{10^{12}} = 0.3mm^2$
- Conséquence: Mettre 1 To de données dans $0.3mm^2$
  - Un mot occupe $\approx 3Angstroms^2$ (la taille d'un petit atome)
- Impossible à réaliser avec la technologie actuelle (gravure et interférence)
- Attention aussi à la chaleur que dégagerait un processeur comme ça !

Densité et puissance

- Les processeurs séquentiels puissants gachent de la puissance électrique
  - Spéculation, vérification de dépendances dynamique
  - Découverte de parallélisme
- Les systèmes concurrents sont plus efficaces d'un point de vue énergétique
  - L'augmentation de la fréquence (f) augmente le voltage (V)
  - L'augmentation du nombre de cores augmente la capacité (C) mais linérairement
  - La puissance dynamique est proportionnelle à $V^2 * f * C$
  - Tendance: Economiser de la puissance en baissant la fréquence

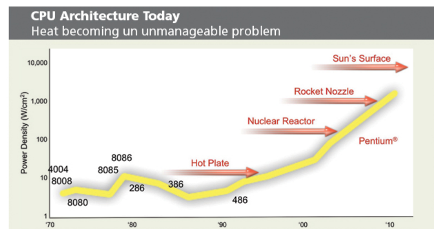


Figure 1. In CPU architecture today, heat is becoming an unmanageable problem. (Courtesy of Pat Gelsinger, Intel Developer Forum, Spring 2004)

Loi de Moore

Gordon Moore (co-fondateur d'Intel) a prédit en 1965 que la densité des processeurs doublerait environ tous les 18 mois.

Microprocessor Transistor Counts 1971-2011 & Moore's Law

curve shows transistor count doubling every two years

Transistor count

Date of introduction

Loi de Moore

- En 1965, raisonnement empirique basé sur une relation entre la complexité des circuits et le temps
- Loi qui a perduré à travers les années
- Croissance due à plusieurs facteurs
  - Augmentation de la complexité des processeurs (densité en transistors, augmentation de la taille)
  - Ajout de fonctionnalités (caches internes, buffers d'instructions plus grands, lancement de plusieurs instructions par cycles, multithreading, profondeur des pipelines, ré-arrangement des instructions)

“The free lunch is over“ – Herb Sutter

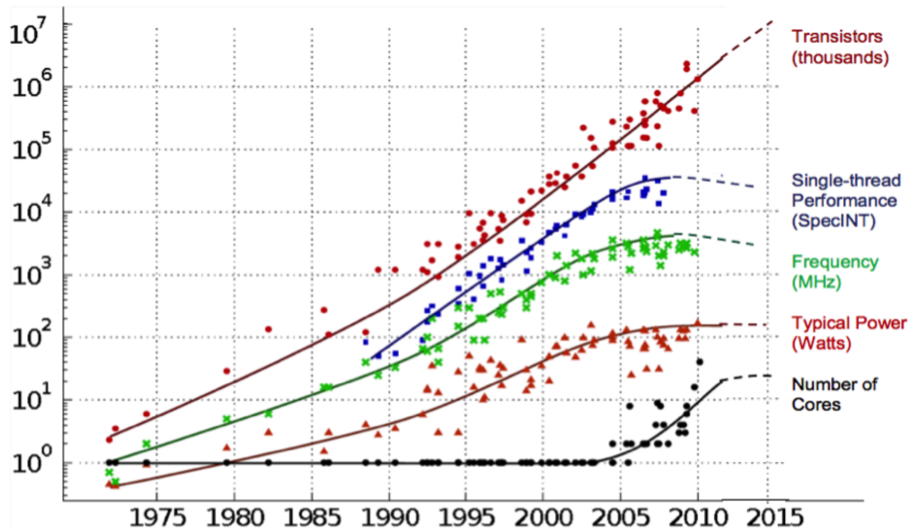
- La vitesse d’horloge ne va plus continuer à doubler ...
- ... mais on veut que les applications continuent à accélérer

- Quelques problèmes dues à l’augmentation de la vitesse d’horloge
  - Consommation électrique
  - Dissipation de la chaleur
  - Fuites

- Mais aussi
  - Limite physique due à la vitesse de la lumière (propagation des signaux)

The Free Lunch Is Over, A Fundamental Turn Toward Concurrency in Software, Herb Sutter, Dr. Dobb’s Journal, 30(3), March 2005.
`http://www.gotw.ca/publications/concurrency-ddj.htm`

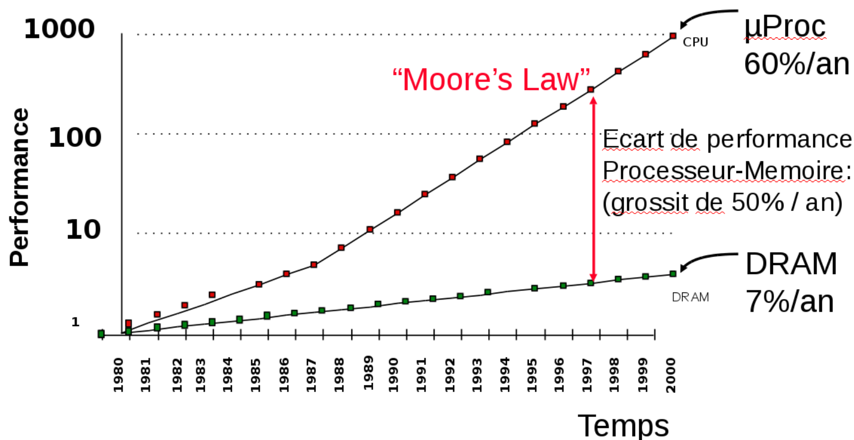
Loi de Moore : La réalité



Loi de Moore: Conséquence

- Le seul moyen d'augmenter les performances est l'augmentation du nombre d'unité de traitement travaillant en parallèle
- L'argument du coût des transferts des données tient toujours !
  - Il faut se débrouiller avec le rapport entre le grain de calcul et le grain de communication
  - Faire en sorte que le volume total des calculs dépasse de loin le volume des communications
- On va voir que c'est loin d'être simple !

Ecart entre le processeur et la mémoire



But: trouver des algorithmes qui minimisent les transferts de données, pas forcement les calculs

Problème de chargement des données

- La vitesse d'exécution des calculs ne dépend pas uniquement de la vitesse du processeur
- Il faut prendre en compte la vitesse de chargement des données de la mémoire vers le processeur
- La vitesse d'accès mémoire n'a augmenté que de 10% par an
  - Goulot d'étranglement important qui tend à augmenter

- Performance du système de mémoire dépendent de la fraction de la mémoire totale qui peut être stockée dans le cache

- Meilleures performances pour les unités parallèles car
  - Caches agrégés plus grands
  - Bande-passante agrégée plus importante
  - Attention à la localité des données (on verra ça plus tard)

Loi de Moore et Parallélisme: Conclusion

- La Loi de Moore tient toujours mais pas comme on s'y attendait
- Le parallélisme est obligatoire et présent partout
- Le chargement des données est aussi important que la rapidité de calcul
- Même problème sur les grands systèmes parallèles
    - Clusters
    - Super-calculateurs
    - Clouds
    - ...

Applications Parallèles

Quelques exemples d'applications

- Sans ordinateurs, soit
  - on étudie les phénomènes sur papier (théorie),
  - on construit un instrument et effectuer des expérimentations
- Limitations de la théorie et expérimentale
  - Trop compliqué (*ex.* : modéliser un Tsunami)
  - Trop cher (*ex.* : test de crash d'avion)
  - Trop lent (*ex.* : évolution climatique, des planètes, des organismes)
  - Trop dangereux (*ex.* : nucléaire, médicaments)
- Utiliser l'informatique pour simuler un phénomène
  - Basé sur les lois de la physique et l'utilisation de méthodes numériques

  - Expérience d'étude des lois de l'évolution
  - 30 ans d'expérience sur E. Coli → 60,000 générations
  - Tous les jours, avoir une personne pour manipuler
  - Simuler le même phénomène est faisable en quelques heures !

Problèmes complexes

- Science
  - Comprendre la matière depuis les particules jusqu'à le cosmos
  - Prévoir la météo et le climat
  - Comprendre la bio-chimie des organismes vivants
- Ingénierie
  - Combustion et conception de moteurs
  - Mécanique des fluides numériques et la conception d'avion
  - Modélisation de structures et des tremblements de terre
  - Modélisation de la polution et des moyens de la contrer
  - Nanotechnologies
- Entreprise
  - Finance numérique (HFT: marche très fort)
  - Recherche d'information
  - Exploration de données
- Défense
  - Essai nucléaire (uniquement numérique en France)
  - Cryptographie

Des besoins grandissants

- Croissance exponentielle de la puissance de calcul demandée (et acquise)
  - La simulation devient le troisième pilier de la science (à côté de la théorie et de l'expérimentation)
- Croissance exponentielle des données expérimentales
  - Amélioration des techniques et technologies en acquisition, analyse, visualisation de données,
  - Aspects transports et stockage
  - Outils collaboratifs
  - Big data

Animation

- Le rendu utilisé pour appliquer des lumières, des textures et des ombres sur des modèles 3D afin de générer des images 2D pour un film
- Utilisation massive du calcul parallèle pour générer le bon nombre d'images pour un film complet (24 images/seconde)
- Quelques exemples
  - Pixar, 1995, Toy Story: premier film entièrement généré par ordinateur ("renderfarm“ constituée de 100 machines dualprocs)
  - Pixar, 1999, Toy Story 2: utilisation d'un système à 1400 processeurs pour une meilleure qualité d'image
  - 2001, Monstres et compagnie: 250 serveurs à 14 procs pour un total de 3500 processeurs
  - Transformers 2, Industrial Light and Magic: render farm avec 5700 cores

Vidéo Massive Software

Biologie

- Augmentation importante des calculs avec l'arrivée massive de séquences d'ADN pour un grand nombre d'organismes, y compris des humains
- Celera corp. : Comparaison de génomes complets
  - Division du génome en petits segments, trouver les séquences ADN des segments expérimentalement, utiliser un calculateur pour construire la séquence complète en trouvant les zones de recouvrements
  - Nombres de comparaisons énorme
- Human Brain Project: Simuler le cerveau humain

Astrophysique

Explorer l'évolution des galaxies, processus thermonucléaires, traiter les données issues des téléscopes

- Analyse d'ensembles de données très larges
  - Ensemble de données "Sky surveys"
  - Sloan Digital Sky Surveys, http://www.sdss.org/
  - Analyse de ces ensembles de données pour trouver des nouvelles planètes, comprendre l'évolution des galaxies

Vidéo Supernova

Modéliser le climat

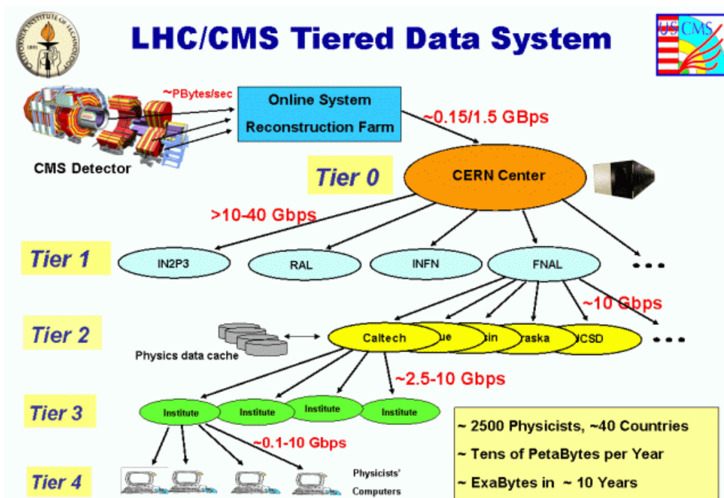
- Calculer (temperature, pression, humidité, vitesse du vent) = f(latitude, longitude, hauteur, temps)
- Approche
  - Discrétiser le domaine (un point de mesure tous les 10 kms)
  - Exécuter un algorithme qui prédit le temps à $t+1$ en fonction du temps à t
- Utilisation
  - La météo
  - Prédire les catastrophes naturelles
  - Evaluer les changements climatiques
  - Compétitions sportives

Vidéo Climat et Tsunami

LHC: Large Hadron Collider – Grand collisionneur de hadrons

- Les accélérateurs de particules sont la clef pour découvrir des particules massives ($E = mc^2$)
- En particulier, le LHC avait pour but de découvrir/prouver le boson de Higgs
- Le LHC est l'accélérateur de particule le plus puissant jamais construit
- Quelques données:
  - 40 millions de collisions par seconde
  - Après filtrage, 100 collisions intéressantes par seconde
  - Un megabyte de données numérisé à chaque collision (soit $0.1 Gb/s$)
  - 10^{10} collisions enregistrées par an (soit $10 Pb/year$)

LHC: Large Hadron Collider – Grand collisionneur de hadrons



Data Science: Vers un quatrième paradigme

- Les données (scientifiques et autres) augmentent exponentiellement
- La capacité de générer des données a excédé notre capacité à les stocker et les analyser
- Les systèmes de calcul et les capteurs suivent la loi de Moore
- Les données de plusieurs Petabytes sont maintenant communs
  - Climat: quelques dizaines de Pb
  - Genome: Séquenceurs haute débit (5Gb/jour VS 1genome/10ans)
  - Physique nucléaire: Prévision LHC à $16Pb/an$
  - Astrophysique: LSST et SKA pourront produire des données $Pb/nuit$
  - Entreprise: Un ordre plus faible mais croissance rapide

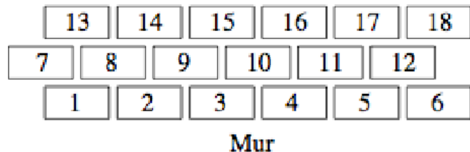
Qu'est ce que le parallèlisme ?

Qu'est-ce que le calcul parallèle ?

- Pouvoir accélérer une application en
  1. Divisant cette applications en sous-tâches
  2. Exécuter ces sous-tâches en parallèles sur des unités différentes
- Pour réussir, il faut être capable de
  1. Trouver le parallélisme dans l'application
  2. Trouver le bon grain de calcul/échange de données
  3. Avoir des connaissances pour concevoir une solution efficace sur la machine cible

Aperçu du parallélisme

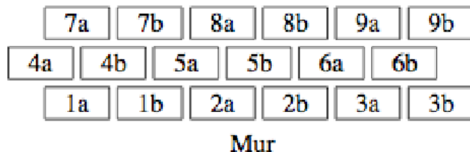
- Travail d'un maçon montant un mur de briques



- Seul il procède par rangées → C'est lent !

Aperçu du parallélisme

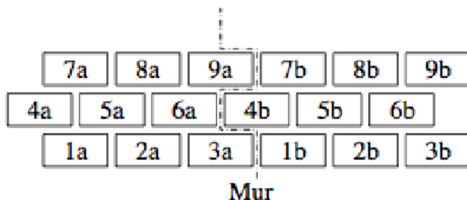
- Travail de 2 maçons (a et b) montant un mur de briques



- Une brique après l'autre → ils se gènent pour prendre des briques et les mettre en place !

Aperçu du parallélisme

- Travail d'un maçon montant un mur de briques



- Chacun s'attribue une portion de mur pour travailler
- Plus efficace mais
  - b a plus de chemin à faire pour récupérer les briques
  - ils se gènent pour prendre les briques
- Variantes
  - a travaille de droite à gauche et b commence plus vite mais problèmes de synchronisation

Aperçu du parallélisme

- Quelques réflexions sur l'exemple
  - Plus efficace que le travail d'un seul maçon mais
  - Plus de travail en général car organisation entre les maçons
- En général
  - Pour avoir une application parallèle, il faut que l'application soit décomposable en sous-problèmes suffisamment indépendants
  - Il faut pouvoir organiser le travail à répartir.
  - Surcoût dû à la répartition du travail (transmission des briques)
  - Trouver le meilleur algorithme parallèle → Pas forcément celui qui est le plus efficace en séquentiel !

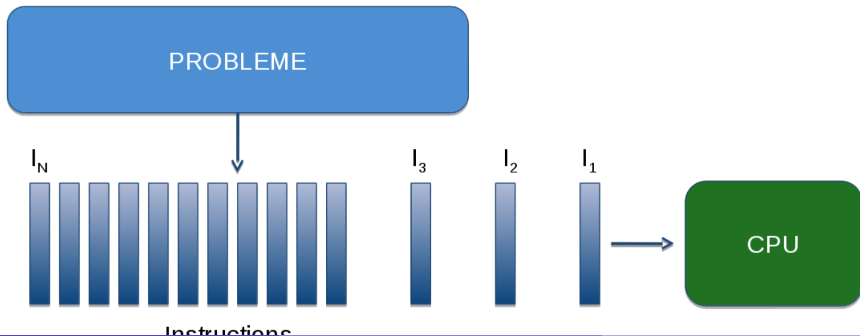
Qu'espère-t-on ?

- Avoir un bon speed-up !
  - Idéalement, on espère avoir une accélération de p sur p processeurs !
- Malheureusement, c'est rarement le cas
  - Parties séquentielles d'un algorithme
  - Problèmes de surcoût (overhead) dus à des calculs redonnants, les coûts de transfert de données (mémoire, disque, réseau)
- Parfois le gain est superieur à p
  - Appelé speed-up superlinéaire
  - Grâce à des différences de vitesse mémoire (mémoire vive vs caches), moins de calcul grâce au parallélisme (recherche dans des arbres)
  - Applications pour lesquelles l'exécution sur un processeur est impossible (durée infinie d'exécution)

En pratique sur une machine

Les programmes sont généralement conçus pour s'exécuter sur des processeurs séquentiels

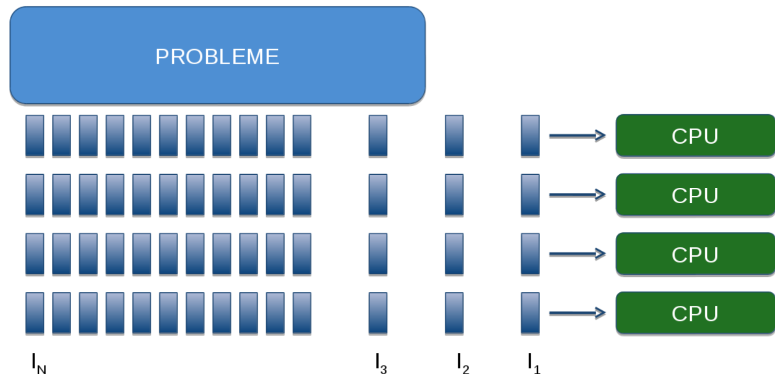
- Unité centrale (CPU) unique
- Application découpée en suite d'instructions exécutées l'une après l'autre
- Une seule instruction s'exécute à un instant donné



En pratique sur une machine

Sous la forme la plus simple, on utilise plusieurs ressources pour résoudre un problème

- Problème divisé en sous-parties indépendantes (si possible)
- Utilisant plusieurs CPU



Qu'est-ce qu'une machine parallèle?

- Une collection d'éléments de calcul capables de communiquer et de coopérer dans le but de résoudre rapidement des problèmes de grande taille

- Une collection d'éléments de calcul
  - Combien ?
  - De quelle puissance ?
  - Que sont-ils capables de réaliser?
  - Quelle est la taille de leur mémoire associée ?
  - Qu'elle en est l'organisation ?
  - Comment les entrées/sorties sont-elles réalisées ?

- ... capables de communiquer ...
  - Comment sont-ils reliés entre eux ?
  - Que sont-ils capables d'échanger ?
  - Quel est leur protocole d'échange d'information ?

Qu'est-ce qu'une machine parallèle?

- ... et de coopérer ...
  - Comment les éléments de calculs synchronisent ils leurs efforts ?
  - Quel est leur degré d'autonomie ?
  - Comment sont-ils pris en compte par le système d'exploitation ?

- dans le but de résoudre des problèmes de grande taille.
  - Quels sont les problèmes à fort potentiel de parallélisme ?
  - Quel est le modèle de calcul utilisé ?
  - Quel est le degré de spécialisation des machines à un problème donné?
  - Comment choisir les algorithmes ?
  - Quelle efficacité peut-on espérer ?
  - Comment ces machines se programment elles ?
  - Quels langages faut-il ?
  - Comment faut-il exprimer le parallélisme ?
  - Le parallélisme est il implicite ou explicite ?
  - L'extraction du parallélisme est-elle automatique ou manuelle ?

Unités de mesure des performances

- Les unités en HPC
  - Flop: floating point operation, généralement double précision
  - Flop/s: opérations flottantes par seconde
  - Octets: taille des données (8 pour un nombre en double précision)

- Des tailles typiques millions, milliards, trillions ...
  - Mega Mflop/s = 10^6 flop/sec
  - Giga Gflop/s = 10^9 flop/sec
  - Tera Tflop/s = 10^{12} flop/sec
  - Peta Pflop/s = 10^{15} flop/sec
  - Exa Eflop/s = 10^{18} flop/sec
  - Zetta Zflop/s = 10^{21} flop/sec
  - Yotta Yflop/s = 10^{24} flop/sec

- La machine la plus puissante au monde actuellement 34 Pflop/s (17 en 2013 et 8.7 en 2012) et consomme 17.8MW
  - Liste mise-à-jour deux fois par an: www.top500.org

Mais avant le parallélisme, optimiser les codes !

- Quelques optimisations classiques
  - Préfetching d'instructions
  - Ré-ordonnancement d'instructions
  - Unités fonctionnelles pipelinées
  - Prédiction de branches
  - Allocation d'unités fonctionnelles
  - Hyperthreading
- Par contre, celà nécessite une complexification du matériel (unités fonctionnelles parallèles) et du logiciel (compilateurs, système d'exploitation) pour les supporter

Trouver le parallélisme

Comment trouver le parallélisme ?

- Partir d'un langage séquentiel ?
  - L'application a un parallélisme intrisèque
  - Le langage de programmation choisi ne possède pas d'extension "parallèle"
- Le compilateur, le système d'exploitation et/ou le matériel doivent se débrouiller pour trouver le parallélisme caché!
  - Fonctionnement correct pour quelques applications trivialement parallèles (parallélisation de boucles imbriquées simples par exemple)
  - Mais en général, résultats décevants et pbs liés à la dynamicité (pointeurs en C par exemple)

Automatique dans les processeurs

- Parallélisme au niveau du bit (BLP, Bit Level Parallelism)
  - Dans les opérations flottantes
- Parallélisme d'instructions (ILP, Instruction Level Parallelism)
  - Exécuter plusieurs instructions par cycle d'horloge
  - Super-scalar, VLIW, EPIC,ThLP (Thread level parallelism: multithreading)
- Parallélisme des gestionnaires de mémoire
  - Recouvrir les accès mémoire avec le calcul (prefetch)
  - Opérations vectorielles en parallèle ($A[*] = 3 \times A[*]$)

Automatique dans les processeurs

- Parallélisme au niveau du système
  - Exécuter des tâches différentes sur des processeurs (ou des cores) différents
  - fork [func1(), func2()], join [*]
- Limites à ce parallélisme “implicite“
  - Niveau d’intelligence des processeurs et des compilateurs
  - Complexité des applications
  - Nombre d’éléments en parallèle

Coopération

- Le programmeur et le compilateur travaillent ensemble
  - L'application a un parallélisme intrinsèque
  - Le langage possède des extensions permettant d'exprimer le parallélisme
  - Le compilateur va traduire le programme pour des unités multiples
- Le programmeur donne des conseils au compilateur sur les zones qu'il faut optimiser, quelles sont les boucles parallèles, ...
- Le compilateur, en partant des informations qu'il possède sur le matériel (taille des caches, nombre d'unités parallèles, informations sur les performances), va pouvoir générer un code performant.

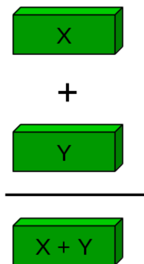
Machines vectorielles

- **But**: opération vectorielles (parallélisme de données) sans le besoin de programmation parallèle
  - Simple : $A[*] = B[*] \times C[*]$
  - Scatter/Gather : $S = sum(D[*])$
- Deux types de machines vectorielles: Pipelined et Array Vector
- Dans les années 1970 et 1980, tous les supercalculateurs étaient vectoriels (Cray, CDC, ...)
- Les GPUs peuvent être vu comme des processeurs vectorielles

SIMD: Single Instruction, Multiple Data

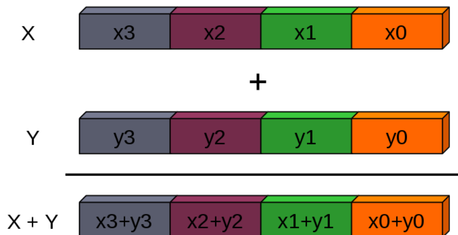
Calcul scalaire

- Mode "classique"
- 1 opération = 1 résultat



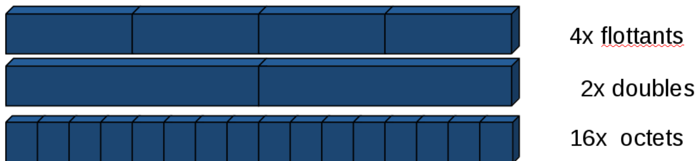
Calcul SIMD

- avec SSE (Streaming SIMD extensions) / SSE2
- 1 opération = *n* résultat



SSE/SSE2 sur processeur Intel

- Types de données SSE2: tout ce qui rentre dans 16 octets, soit



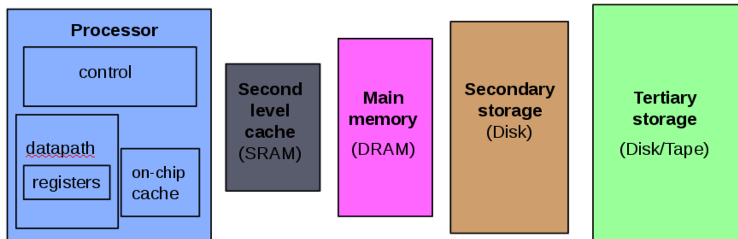
- Les instructions effectuent des additions, des multiplications, etc. sur toutes les données dans ces registres de 16 bits en parallèle
- Challenges
  - Doit être contigu en mémoire et aligné
  - Quelques instructions pour déplacer les données d'une partie d'un registre vers une autre
  - Similaire au GPU, processeurs vectoriels (mais plus d'opérations simultanées)

Instructions spéciales et compilateurs

- En plus des instructions SIMD, le processeur peut avoir aussi d'autres instructions
  - Instruction multiply-add (Fused Multiply-Add, FMA) : $x = y + x \times z$
  - Le processeur exécute ces instructions à la même fréquence qu'un $\times$ ou un $+$
- En théorie les compilateurs connaissent ces instructions
  - Lors de la compilation, le compilateur va ré-arranger les instructions pour obtenir un bon ordonnancement des instructions qui va maximiser le pipeline (FMA et SIMD)
  - Il utilise des mélanges de telles instructions dans des boucles internes
- En pratique, le compilateur a besoin d'aide
  - Prise en compte de drapeaux de compilation
  - Ré-arranger le code pour qu'il trouve plus facilement les bons "pipelines"
  - Utiliser des fonctions spéciales
  - Ecrire en assembleur !

Hiérarchie mémoire

- La plupart des programmes possèdent une forte localité dans leurs accès aux données
  - **Localité spatiale**: accéder à des données proches des accès précédents
  - **Localité temporelle**: ré-utiliser une donnée qui a été accédée auparavant
- Les hiérarchies mémoire tentent d'exploiter la localité pour améliorer les performances (et on verra plus tard que sur les multi-cores c'est pire)



Approche pour supporter la latence mémoire

- La bande-passante a augmenté plus vite que la latence a décru
  - 23% par an contre 7% par an
- Quelques techniques
  - Eliminer les opérations en sauvegardant les données dans des mémoires petites (mais rapides, les caches) et en les ré-utilisant
    - Ajouter de la localité temporelle dans les programmes
  - Utiliser mieux la bande-passante en récupérant un morceau de mémoire et en le sauvegardant dans un cache et en utilisant le bloc entier
    - Ajouter de la localité spatiale dans les programmes
  - Utiliser mieux la bande-passante en permettant au processeur d'effectuer plusieurs lectures en même temps
    - Concurrence dans le flot d'instructions (chargement d'un tableau entier dans les processeurs vectoriels, prefetching)
  - Recouvrement calcul et opérations mémoire
    - prefetching

Quelques rappels sur les caches

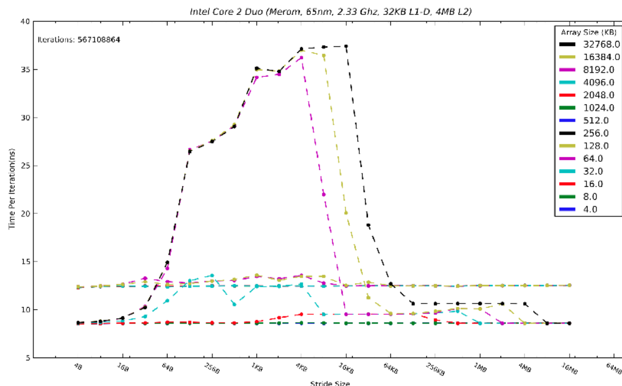
- **Cache**: mémoire rapide (et coûteuse) qui conserve des copies des données dans la mémoire principale (gestion cachée à l'utilisateur)
  - Exemple simple: une donnée à l'adresse xxxxx1101 est stockée dans le cache à l'adresse 1101
- **Cache hit**: accès à la donnée dans le cache, peu coûteux
- **Cache miss**: accès à une donnée non cachée, coûteux
  - Besoin d'accéder au niveau supérieur (plus lent)
- Longueur d'une ligne de cache: nombre d'octets chargés en même temps
- **Associativité**
  - Direct-mapping: seule une adresse (ligne)
    - La donnée stockée à l'adresse xxxxx1101 stockée à l'adresse 1101 du cache, dans un cache à 16 mots
  - N-way: $n \geq 2$ lignes avec des adresses différentes peuvent être stockées
    - Jusqu'à $n \leq 16$ mots avec des adresses xxxxx1101 peuvent être stockés à l'adresse 1101 du cache (le cache peut stocker 16n mots)
  - Caches hiérarchiques avec des vitesses décroissantes et des tailles croissantes
    - Dans les processeurs et en dehors

Pourquoi avoir des niveaux de caches multiples ?

- Le cache rapide ça coûte très cher !
- On-chip vs off-chip
  - Les caches internes sont plus rapides mais limités en taille
- Les caches de grandes tailles sont plus lents
  - Le matériel met plus de temps à vérifier les adresses plus longues
  - L'associativité, qui permet d'avoir des ensembles de données plus important, a un coût
- Quelques exemples
  - Cray a supprimé un cache pour accélérer certains accès (sur le T3E)
  - IBM (Power 5 et 6) utilise un cache appelé "victim cache" qui est moins coûteux
- Il y a d'autres niveaux de hiérarchie mémoire
  - Registres, pages (TLB, mémoire virtuelle), ...
  - Et pas toujours hiérarchique

Modélisation des caches

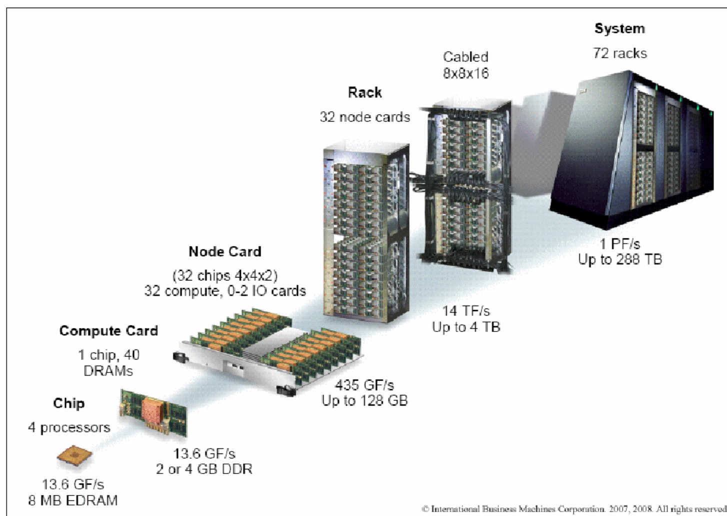
- Pourquoi ? Pour pouvoir prédire les performances mais ...
- Microbenchmarks disponibles (membench, stanza triad)
  - Fonctionnent assez bien pour des caches simples et non hiérarchiques
  - Compliqué pour les nouvelles générations de processeurs (sans parler de l'influence du système d'exploitation)



Conclusion sur les caches

- Les performances réelles d'un programme peuvent être difficiles à appréhender en fonction de l'architecture
  - La moindre modification de l'architecture et du programme a une grosse influence sur les performances
  - Pour écrire des programmes efficaces, il faut prendre en compte l'architecture
  - On voudrait des modèles simples pour concevoir des algorithmes efficaces
- Tenir compte des caches dans le programme
  - Utiliser un algorithme diviser-pour-régner pour que les données soient idéalement placées dans les caches L1 et L2
  - Utiliser une bibliothéque qui se charge de ça

Les machines parallèles d'aujourd'hui

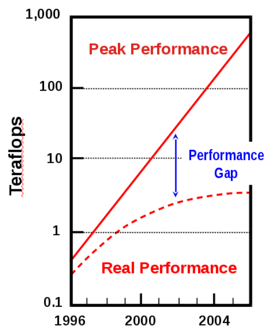


Obtenir des performances sur ces machines

- Généralement on ne parle que de puissance de calcul
- Mais il faut prendre en compte aussi la bande-passante mémoire et la latence
  - Il faut être capable de remplir la vitesse du (des) processeur(s)
  - Mémoire hiérarchique et caches
- La bande-passante en E/S vers les disques croit linéairement avec le nombre de processeurs

Améliorer les performances réelles

- Les performances de crête augmentent exponentiellement
  - En 1990's, les performances de crête ont augmenté 100x; dans les années 2000, elles augmenteront 1000x (et on espère en 2010)
- Mais l'efficicacité (les performances par rapport aux performances en crête) ont plutôt déclinées
  - 40-50% sur les supercalculateurs vectoriels des années 1990
  - Maintenant proches de 5-10% sur les supercalculateurs d'aujourd'hui



- Réduire l'écart..
  - Algorithmes qui obtiennent des performances sur un seul processeur et sont extensibles sur plusieurs milliers
  - Modèles de programmation plus efficaces et des outils pour les machines massivement parallèles

Le parallélisme de nos jours

- Tous les vendeurs de processeurs produisent des processeurs multicore
  - Toutes les machines seront bientôt parallèles
  - Pour continuer à doubler la puissance il faut doubler le parallélisme
- Quelles applications vont (bien) tirer partie du parallélisme ?
  - Est-ce qu'il faudra les redévelopper from scratch ?
  - Et les systèmes d'exploitation ?
- Est-ce que tous les programmeurs devront être des programmeurs de machines parallèles ?
  - Il faut des nouveaux modèles logiciels
  - Essayer de cacher le parallélisme au maximum
  - Mais il faut le comprendre !
- L'industrie parie sur ces changements...
- ... mais encore beaucoup de travail à effectuer

Challenges

- Les applications parallèles sont souvent très sophistiquées
  - Algorithmes adaptatifs qui nécessitent un équilibrage dynamique
- Le parallélisme multi-niveaux est difficile à gérer
- La taille des nouvelles machines donnent des problèmes d'efficacité
  - Problèmes d'extensibilité
  - Sérialisation et déséquilibrage de charge
  - Goulots d'étranglement et en communication et/ou en entrées/sorties
  - Parallélisation insuffisante ou inefficace
  - Fautes et/ou pannes
  - Gestion de l'énergie
- Difficulté d'obtenir les performances les meilleures sur les nœuds eux-mêmes
  - Contention pour la mémoire partagée
  - Utilisation de la hiérarchie mémoire sur les processeurs multi-cores
  - Influence du système d'exploitation

Conclusion

- L'ensemble des machines parallèles est constituée d'un ensemble (très) large d'éléments
  - depuis des unités parallèles dans les processeurs
  - Jusqu'à des datacenters connectés à travers le monde
- Champ d'étude du parallélisme
  - Architectures
  - Algorithmes
  - Logiciels, compilateurs,
  - Bibliothèques
  - Environnements
- Changements historiques importants
  - Jusque dans les années 90, réservées à des gros calculs de simulation
  - Aujourd'hui, parallélisme dans tous les processeurs (des téléphones aux supercalculateurs), parallélisme dans la vie courante (iPad, Google !)