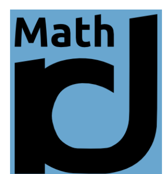
Introduction à l'architecture et aux langages

Violaine Louvet

Institut Camille Jordan - CNRS
louvet@math.univ-lyon1.fr

ANF R, Octobre 2015



Pourquoi ce GRR!@#!! de code ne va pas plus vite!

Pourquoi ce GRR !@# ! ! de code ne va pas plus vite !

- Ma machine est trop vieille

Pourquoi ce GRR !@# ! ! de code ne va pas plus vite !

- Ma machine est trop vieille
- Je viens pourtant d'acheter une nouvelle machine !

Pourquoi ce GRR!@#!! de code ne va pas plus vite!

- Ma machine est trop vieille
- Je viens pourtant d'acheter une nouvelle machine!
- Mon programme est écrit en java

Pourquoi ce GRR!@#!! de code ne va pas plus vite!

- Ma machine est trop vieille
- Je viens pourtant d'acheter une nouvelle machine!
- Mon programme est écrit en java
- Ce code a été programmé par une multitude de stagiaires / thésards

Pourquoi ce GRR!@#!! de code ne va pas plus vite!

- Ma machine est trop vieille
- Je viens pourtant d'acheter une nouvelle machine!

- Mon programme est écrit en java

- Ce code a été programmé par une multitude de stagiaires / thésards
- Je code au fur et à mesure des idées qui viennent

Pourquoi ce GRR !@# ! ! de code ne va pas plus vite !

- Ma machine est trop vieille
- Je viens pourtant d'acheter une nouvelle machine !
  - En quoi cela impacte-t-il les performances du code ?
- Mon programme est écrit en java

- Ce code a été programmé par une multitude de stagiaires / thésards
- Je code au fur et à mesure des idées qui viennent

Pourquoi ce GRR!@#!! de code ne va pas plus vite!

- Ma machine est trop vieille
- Je viens pourtant d'acheter une nouvelle machine!
  - En quoi cela impacte-t-il les performances du code?
- Mon programme est écrit en java
  - Ce langage est-il le plus approprié?
- Ce code a été programmé par une multitude de stagiaires / thésards
- Je code au fur et à mesure des idées qui viennent

Pourquoi ce GRR!@# !! de code ne va pas plus vite!

- Ma machine est trop vieille
- Je viens pourtant d'acheter une nouvelle machine!
  - En quoi cela impacte-t-il les performances du code?
- Mon programme est écrit en java
  - Ce langage est-il le plus approprié?
- Ce code a été programmé par une multitude de stagiaires / thésards
- Je code au fur et à mesure des idées qui viennent
  - Le programme est-i adapté? La structure des données est-elle appropriée, cohérente? Quel est l'endroit où l'exécution passe le plus de temps? Quelles sont les potentialités d'optimisation?

Sommaire

1. Le rôle de l'architecture
   - Le processeur
   - La mémoire
   - Multi-cœurs, multi-nœuds
   - Evolution des architectures
2. Représentation de l'information
   - Du binaire au flottant
   - Joies et déboires du calcul flottant
   - Erreurs : quelle confiance dans les calculs ?
3. Système et Langages
   - Rôle du système d'exploitation
   - Topologie des langages de programmation
4. Conclusions

Sommaire

1 Le rôle de l'architecture
- Le processeur
- La mémoire
- Multi-cœurs, multi-nœuds
- Evolution des architectures

2 Représentation de l'information

3 Système et Langages

4 Conclusions

« The free lunch is over »

Avant

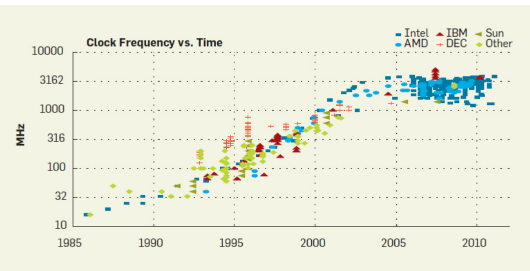
- Jusqu'en 2005, un seul CPU
- Augmentation de la performance par l'augmentation de la fréquence

« The free lunch is over »

Avant

- Jusqu'en 2005, un seul CPU
- Augmentation de la performance par l'augmentation de la fréquence

Mais ...



Evolution depuis 2005

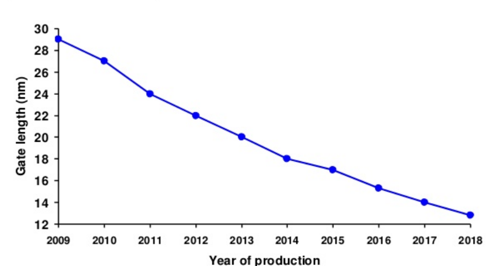
Pourquoi ne peut-on plus augmenter la fréquence ?

- l'augmentation de la fréquence d'horloge entraîne une augmentation de la quantité de chaleur à dissiper.
- Il n'est plus possible de refroidir les processeurs de façon économiquement rentable
- Les fréquences actuelles sont stabilisées autour de 3 à 4 GHz

Evolution depuis 2005

Miniaturisation

- La dimension des transistors continue à diminuer (augmentation de la finesse de gravure), et donc leur nombre par unité de surface continue d'augmenter



Quelles conséquences ?

- Multiplication des cœurs de calcul sur une même puce
- Sophistication extrême de l'architecture interne
- Stagnation, et même baisse de la fréquence des cœurs

Quelles conséquences ?

- Multiplication des cœurs de calcul sur une même puce
- Sophistication extrême de l'architecture interne
- Stagnation, et même baisse de la fréquence des cœurs

Et pour les logiciels ?

- Un programme séquentiel classique ne pourra s'exécuter que sur un seul cœur à la fois et ne bénéficiera pas de l'évolution des architectures
- Voire même perdra en performance

Sommaire

1 Le rôle de l'architecture
- Le processeur
- La mémoire
- Multi-cœurs, multi-nœuds
- Evolution des architectures

2 Représentation de l'information
- Du binaire au flottant
- Joies et déboires du calcul flottant
- Erreurs : quelle confiance dans les calculs ?

3 Système et Langages
- Rôle du système d'exploitation
- Topologie des langages de programmation

4 Conclusions

Comprendre l'architecture de sa machine

Démo : lscpu

Comprendre l'architecture de sa machine

Démo : lscpu

- CPU : Central Processing Unit. Selon le contexte, désigne le processeur physique, ou un simple cœur de calcul.

Comprendre l'architecture de sa machine

Démo : lscpu

- CPU : Central Processing Unit. Selon le contexte, désigne le processeur physique, ou un simple cœur de calcul.
- Thread : processus léger, ou fil d'exécution.

Comprendre l'architecture de sa machine

Démo : lscpu

- CPU : Central Processing Unit. Selon le contexte, désigne le processeur physique, ou un simple cœur de calcul.
- Thread : processus léger, ou fil d'exécution.
- Cœur : unité de calcul pouvant exécuter des programmes ou des threads. Le processeur physique contient plusieurs cœurs.

Comprendre l'architecture de sa machine

Démo : lscpu

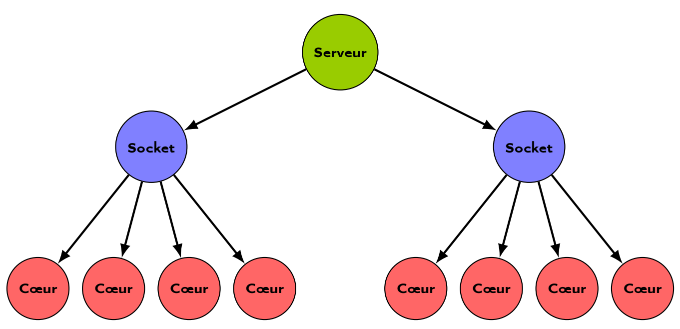
- CPU : Central Processing Unit. Selon le contexte, désigne le processeur physique, ou un simple cœur de calcul.
- Thread : processus léger, ou fil d'exécution.
- Cœur : unité de calcul pouvant exécuter des programmes ou des threads. Le processeur physique contient plusieurs cœurs.
- Socket : support physique du processeur sur la carte mère.

Comprendre l'architecture de sa machine

Démo : lscpu

- CPU : Central Processing Unit. Selon le contexte, désigne le processeur physique, ou un simple cœur de calcul.
- Thread : processus léger, ou fil d'exécution.
- Cœur : unité de calcul pouvant exécuter des programmes ou des threads. Le processeur physique contient plusieurs cœurs.
- Socket : support physique du processeur sur la carte mère.
- Noeud NUMA : Non Uniform Memory Access.

Pour résumer



Serveur bi-sockets quadri-cœurs

Que faut-il retenir de la composition et du fonctionnement d'un cœur de calcul ?

Un cœur de calcul comprend :

- des unités de calcul : calculs arithmétiques élémentaires sur des nombres entiers, opérations logiques, calculs flottants
- de la mémoire : registres et cache L1
- du contrôle pour séquencer le déroulement des calculs

Que faut-il retenir de la composition et du fonctionnement d'un cœur de calcul ?

Fonctionnement : le cœur inclut beaucoup de mécanismes d'optimisation complexes :

- Pipelining, processeur superscalaire : pour pouvoir traiter plusieurs instructions en même temps
- Exécution spéculative, prédiction de branchement : pour prendre de l'avance sur les instructions du programme
- Hyperthreading : deux processeurs logiques sur une seule puce, permettant d'utiliser au mieux les ressources du processeur.
- Instructions vectorielles : le processeur, en fonction de son jeu d'instructions, est capable d'appliquer la même instruction simultanément à plusieurs données (SSE, AVX).

Comment évalue-t-on la performance d'une machine ?

Opérations (additions ou multiplications) à virgule flottante par seconde = FLoating point Operations Per Second (FLOps)

Puissance crête (point de vue théorique)

Exemple sur un processeur Intel Sandybridge

- fréquence d'un coeur : 3.2GHz
- nombre de coeurs : 6
- registres vectoriels : 16 en simple précision (c'est à dire quand les flottants sont stockés sur 32 bits), 8 en double précision (c'est à dire quand les flottants sont stockés sur 64 bits)

La performance crête d'un processeur de ce type sera donc :

$$6 \times 3.2 \times 16 = 307.2 \text{ GFLOPS}$$

Revenons sur terre

Revenons sur terre

- Un code classique est séquentiel

Revenons sur terre

- Un code classique est séquentiel
- En général, la vectorisation est peu ou pas faite

Revenons sur terre

- Un code classique est séquentiel
- En général, la vectorisation est peu ou pas faite
- Du coup, on utilise théoriquement en fait que 3.2 GFLOPS ... en gros la puissance crête d'un processeur pentium 4 de 2000 !

Revenons sur terre

- Un code classique est séquentiel
- En général, la vectorisation est peu ou pas faite
- Du coup, on utilise théoriquement en fait que 3.2 GFLOPS ... en gros la puissance crête d'un processeur pentium 4 de 2000 !

Et ça c'est juste la théorie !

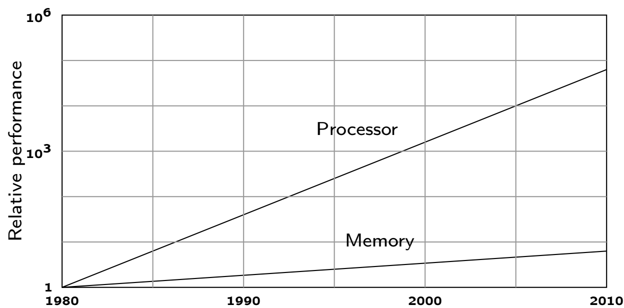
En pratique, si on regarde le nombre d'opérations flottantes par rapport au temps de calcul, on est loin du compte ...

Pourquoi ? ?

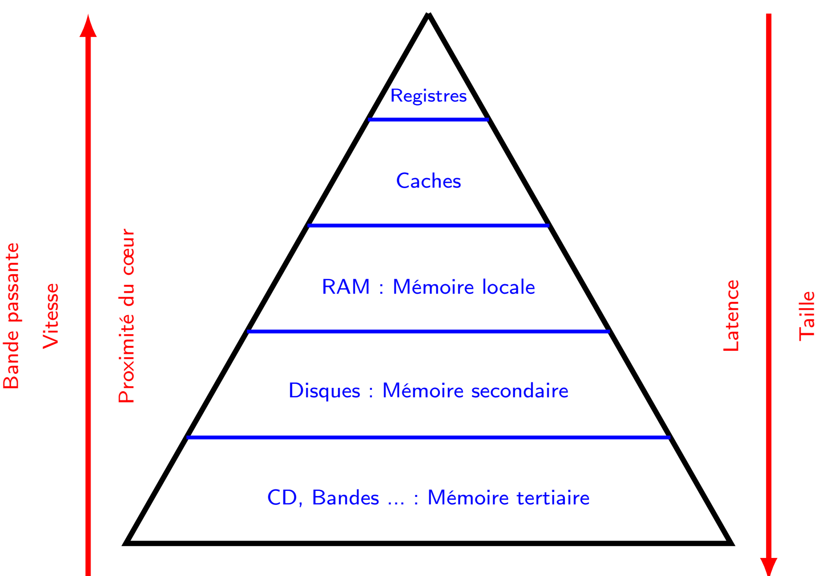
Sommaire

1. Le rôle de l'architecture
   - Le processeur
   - La mémoire
   - Multi-cœurs, multi-nœuds
   - Evolution des architectures
2. Représentation de l'information
   - Du binaire au flottant
   - Joies et déboires du calcul flottant
   - Erreurs : quelle confiance dans les calculs ?
3. Système et Langages
   - Rôle du système d'exploitation
   - Topologie des langages de programmation
4. Conclusions

« Memory Wall » : $Vitesse_{CPU} \gg Vitesse_{Memory}$

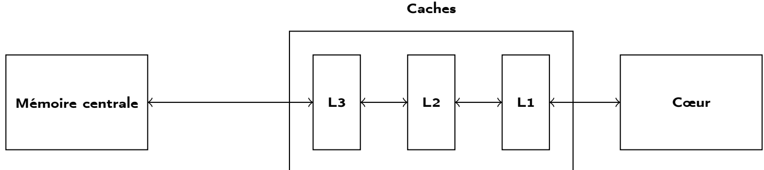


Hiérarchie Mémoire



Quelques ordres de grandeur (Pour un core i7 / Xeon 5500)

- Registre : 1 cycle
- Cache L1 : 4 cycles
- Cache L2 : 10 cycles
- Cache L3 : de 40 à 75 cycles (partagé par les cœurs d'une même socket)
- RAM locale : 60 ns (locale à la socket)
- RAM distante : 100 ns (sur une autre socket)



Quelques ordres de grandeur (Pour un core i7 / Xeon 5500)

- Registre : 1 cycle
- Cache L1 : 4 cycles
- Cache L2 : 10 cycles
- Cache L3 : de 40 à 75 cycles (partagé par les cœurs d'une même socket)
- RAM locale : 60 ns (locale à la socket)
- RAM distante : 100 ns (sur une autre socket)

En résumé

- le CPU effectue une opération par cycle (à la louche, en fait plus)
- Si il a besoin d'une donnée qui se trouve dans le cache L3, il va attendre jusqu'à 75 cycles alors qu'il aurait pu exécuter 75 instructions

Principe de fonctionnement des caches

Lorsque le CPU a besoin d'une donnée :

- Hit : Si l'information se trouve dans le cache L1, le processeur y accède sans état d'attente
- Miss : sinon il faut aller la chercher plus loin (niveau de cache supérieur ou mémoire).

Les transferts mémoire ne se font pas donnée par donnée, mais par ensemble de données.

- Le CPU travaille sur des mots (par ex 32 ou 64 bits)
- les transferts mémoire se font par lignes (ou blocs) de mots (par ex 256 octets)

Quelles conséquences pour nous ?

Exemple du parcours d'une matrice

- On considère un tableau 2D
- Allocation mémoire : les données sont stockées dans un bloc mémoire contigu sous la forme d'un vecteur
- En C/C++ : stockage des lignes les unes derrière les autres
- En Fortran : stockage des colonnes les unes derrière les autres

for $i \in 1, n$ **do**
  for $j \in 1, n$ **do**
    $a_{i,j} = 0.$
  end for
end for

for $j \in 1, n$ **do**
  for $i \in 1, n$ **do**
    $a_{i,j} = 0.$
  end for
end for

Une version est « bonne », l'autre pas ! Démonstration par l'exemple

Sommaire

1. Le rôle de l'architecture
   - Le processeur
   - La mémoire
   - Multi-cœurs, multi-nœuds
   - Evolution des architectures
2. Représentation de l'information
   - Du binaire au flottant
   - Joies et déboires du calcul flottant
   - Erreurs : quelle confiance dans les calculs ?
3. Système et Langages
   - Rôle du système d'exploitation
   - Topologie des langages de programmation
4. Conclusions

Revenons à notre portable

Démo : hwloc-ls

- Un code séquentiel n'exploitera qu'un seul des cœurs de calcul
- Il faut arriver à utiliser le maximum des ressources de la machine

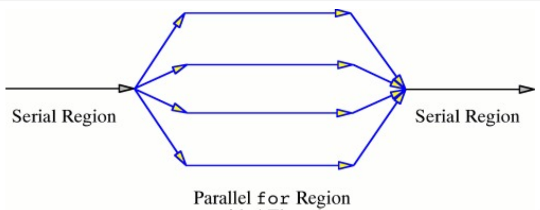
Revenons à notre portable

Démo : hwloc-ls

- Un code séquentiel n'exploitera qu'un seul des cœurs de calcul
- Il faut arriver à utiliser le maximum des ressources de la machine

Parallélisme en mémoire partagée

- Décomposition du calcul en tâches paralléles
- Ces tâches vont être exécutées en même temps via des threads ou processus légers
- Les threads ont tous accès à la mémoire commune
- Il faut faire attention aux accès concurrents sur les données



Exemple de parallélisme par threads

OpenMP

- Standard défini par un consortium (`http://www.openmp.org`)
- Essentiellement des directives de compilation

Démo : comparaison de génération pseudo-aléatoire / htop

Utiliser plusieurs machines

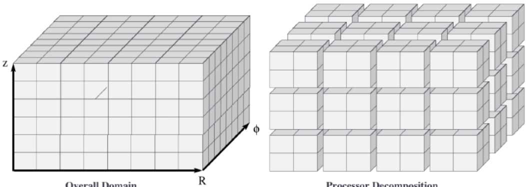
Pourquoi ?

- Le nombre de cœurs sur une seule machine est limité
- La mémoire disponible est, elle aussi, limitée
- On ne peut donc pas traiter de très gros volume de données, ou simuler des domaines très grands

Utiliser plusieurs machines

Parallélisme en mémoire distribuée

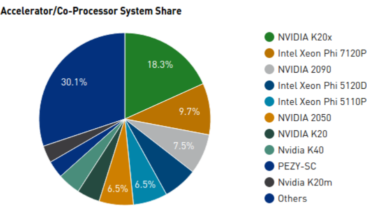
- Cluster : Regroupement d'un ensemble de machines via un réseau très rapide
- Chaque machine doit communiquer avec les autres : le programmeur doit définir explicitement les données qui doivent aller d'une machine à l'autre : messages
- Utilisation de la bibliothèque MPI



Sommaire

1. Le rôle de l'architecture
   - Le processeur
   - La mémoire
   - Multi-cœurs, multi-nœuds
   - Evolution des architectures
2. Représentation de l'information
   - Du binaire au flottant
   - Joies et déboires du calcul flottant
   - Erreurs : quelle confiance dans les calculs ?
3. Système et Langages
   - Rôle du système d'exploitation
   - Topologie des langages de programmation
4. Conclusions

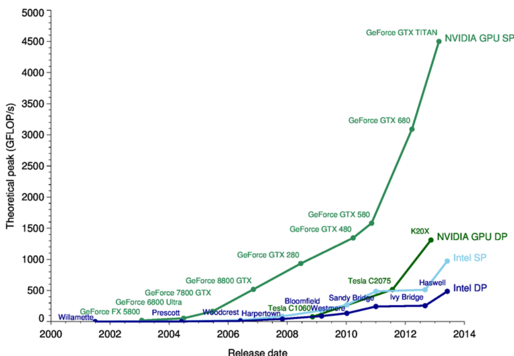
Projection dans le futur



Top500, juin 2015

- Les supercalculateurs sont et seront accélérés
- Tendance qui va arriver sur les serveurs de calcul de plus petite taille

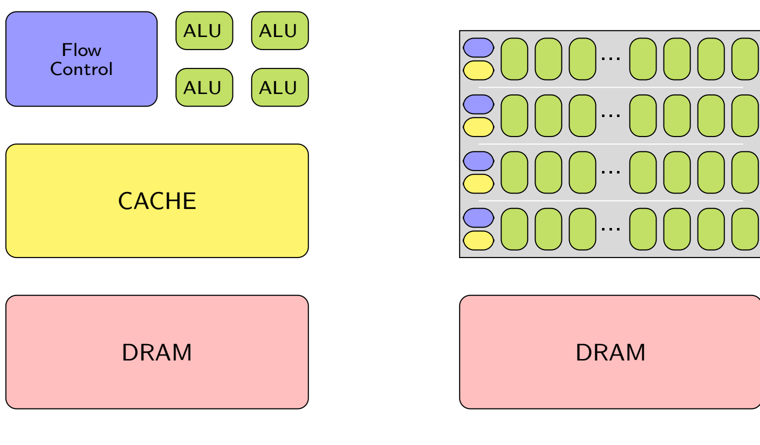
Pourquoi les GPU ?



A performance théorique égale, les plateformes à base de GPU :

- occupent moins de place
- sont moins chères
- sont moins consommatrices d'électricité

Architecture comparée d'un CPU et d'un GPU



Quelles applications pour le GPU ?

Exploitation d'un GPU

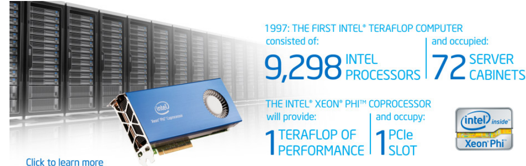
- Programmation de type SIMD, gestion de milliers de thread
- Transfert CPU-GPU encore très couteux
- De grosses évolutions en ce qui concerne la programmation

→ Reste relativement complexe à exploiter pour en tirer des performances raisonnables, toutes les applications ne s'y prêtent pas :

- Problématique avec beaucoup d'algèbre linéaire (utilisation des bibliothèques CUDA)
- Applications avec de nombreuses tâches indépendantes
- Il faut pouvoir exploiter des milliers/millions de threads

Many cœurs

- Apparition sur le marché, en 2012, de cartes Many Integrated Core : les Xeon Phi©
- Intel Xeon Phi 5110P : 60 coeurs à 1053 MHz, 8 Go de mémoire



Spécificités

Une architecture x86 native - en fait à la fois un complément idéal à Xeon et un concurrent crédible aux accélérateurs GPU.

Programmer sur le Phi

- La carte exécute son propre OS, dispose de sa propre adresse IP et passe par le bus PCIe pour communiquer avec les autres éléments du système
- On peut donc se connecter dessus par ssh et travailler presque comme sur un serveur classique. Deux modes d'exploitation :
  - déporter les parties parallèles sur le Phi (offload)
  - exécuter nativement tout le code sur le Phi
- 60 cœurs, 240 threads
- Vecteurs de 512 bits

Pour exploiter efficacement un Phi

- Application massivement parallèle
- Boucles vectorisées (alignement des données)
- Localité des données optimisée

Sommaire

1 Le rôle de l'architecture

2 Représentation de l'information
- Du binaire au flottant
- Joies et déboires du calcul flottant
- Erreurs : quelle confiance dans les calculs ?

3 Système et Langages

4 Conclusions

Sommaire

1 Le rôle de l'architecture
- Le processeur
- La mémoire
- Multi-cœurs, multi-nœuds
- Evolution des architectures

2 Représentation de l'information
- Du binaire au flottant
- Joies et déboires du calcul flottant
- Erreurs : quelle confiance dans les calculs ?

3 Système et Langages
- Rôle du système d'exploitation
- Topologie des langages de programmation

4 Conclusions

Binaire

Tout type d'information (nombres, texte, image, son ...) doit être codé en binaire. En simulation numérique, on manipule des nombres à virgule (réels). Il est impossible de représenter exactement tous les nombres réels avec une quantité d'information finie !

Comment représenter les nombres réels de manière approchée avec une quantité d'information fixe ?

Représentation flottante

En virgule flottante, en base β, un nombre réel x est représenté par :

- un signe $s \in \{0, 1\}$
  - 0 : positif
  - 1 : négatif
- une mantisse m, écrite en virgule fixe en base β sur p chiffres appelés digit
- un exposant $e \in \{e_{min}, ..., e_{max}\}$

$$x = (-1)^s \times m \times \beta^e$$

avec pour $k \in \{0, \ldots - 1\}$:

$$m = m_0 \cdots m_i . m_{i+1} \cdots m_{p-1} \text{ et } m_k \in \{0, \ldots, \beta - 1\}$$

- On dit que le nombre flottant x est de précision p (avec $p \geq 1$)

Limites de la représentation flottante

- Plus petit nombre représentable : $\epsilon = \pm 0.1\beta^{e_{min}}$
- Plus grand nombre représentable : $M = \pm 0.(\beta - 1)(\beta - 1)\dots(\beta - 1)\beta^{e_{max}}$
- Dépassement de capacité : Si l'on cherche à représenter un nombre plus petit que ϵ, on produit un débordement par valeur inférieure (underflow). Si l'on cherche à représenter un nombre plus grand que M, on produit un débordement par valeur supérieure (overflow).
- Certains réels sont par définition impossibles à représenter en numération classique : $1/3$, π ...
- La représentation en un nombre fixe d'octets oblige le processeur à faire appel à des approximations afin de représenter les réels.
- Le degré de précision de la représentation par virgule flottante des réels est directement proportionnel au nombre de bits alloué à la mantisse, alors que le nombre de bits alloué à l'exposant conditionnera l'amplitude de l'intervalle des nombres représentables.

Sommaire

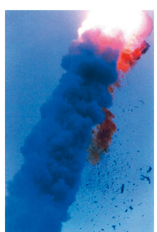
1. Le rôle de l'architecture
   - Le processeur
   - La mémoire
   - Multi-cœurs, multi-nœuds
   - Evolution des architectures
2. Représentation de l'information
   - Du binaire au flottant
   - Joies et déboires du calcul flottant
   - Erreurs : quelle confiance dans les calculs ?
3. Système et Langages
   - Rôle du système d'exploitation
   - Topologie des langages de programmation
4. Conclusions

Opérations flottantes

Attention

Plusieurs propriétés de l'arithmétique (associativité, distributivité, ...) ne sont plus valides en arithmétique flottante !

Conséquence : le même programme, compilé par deux compilateurs (optimiseurs) différents ne donne pas toujours *exactement* le même résultat.



Quelques exemples inattendus

Catastrophic cancellation

Perte de précision qui résulte de la soustraction de deux nombres voisins.

Demo

Quelques exemples inattendus

Catastrophic cancellation

Perte de précision qui résulte de la soustraction de deux nombres voisins.

Demo

Calcul de récurrences

$$u_{n+1} = 4u_n - 1$$

avec $u_0 = 1/3$.

Demo

Quelques exemples inattendus

Catastrophic cancellation

Perte de précision qui résulte de la soustraction de deux nombres voisins.

Demo

Calcul de récurrences

$$u_{n+1} = 4u_n - 1$$

avec $u_0 = 1/3$.

Demo

Réaction du calculateur malin : *comportement normal : puisqu'on ne calcule pas exactement les* u_i*, les erreurs sont multipliées par 4 à chaque itération.*

Quelques exemples inattendus

Catastrophic cancellation

Perte de précision qui résulte de la soustraction de deux nombres voisins.

Demo

Calcul de récurrences

$$u_{n+1} = 4u_n - 1$$

$$\text{avec } u_0 = 1/3.$$

Demo

Réaction du calculateur malin : *comportement normal : puisqu'on ne calcule pas exactement les u_i, les erreurs sont multipliées par 4 à chaque itération.*

On recommence avec :

$$u_{n+1} = 3u_n - 1$$

$$\text{avec } u_0 = 1/2.$$

Demo

Sommaire

1. Le rôle de l'architecture
   - Le processeur
   - La mémoire
   - Multi-cœurs, multi-nœuds
   - Evolution des architectures
2. Représentation de l'information
   - Du binaire au flottant
   - Joies et déboires du calcul flottant
   - Erreurs : quelle confiance dans les calculs ?
3. Système et Langages
   - Rôle du système d'exploitation
   - Topologie des langages de programmation
4. Conclusions

Erreurs de calcul

Quand on cherche à résoudre un problème physique, biologique, ... avec un ordinateur, plusieurs sources d'erreur :

- Erreur sur les données : imprécision de mesures physiques
- Erreur sur les modèles : simplification de la complexité d'un phénomène
- Erreur d'approximation ou de discrétisation : l'analyse numérique nous permet de la quantifier
- Erreur due à l'algorithme si il est itératif : accumulation d'erreurs d'arrondi
- Erreur due à l'arithmétique flottante

Finalement, que peut on calculer avec les nombres à virgule flottante ?

On ne peut effectuer que des calculs pour lesquels la solution dépend gentiment des données (problèmes bien posés).

Sensibilité aux erreurs

- Conditionnement : sensibilité du résultat à une petite variation des données.
- Problème mal conditionné = grande sensibilité vis à vis des données.
- Stabilité : sensibilité de l'algorithme vis à vis des erreurs numériques

Calcul de la précision machine

$$\frac{\|a - fl(a)\|}{\|a\|} \leq \epsilon$$

Demo

Sommaire

1 Le rôle de l'architecture

2 Représentation de l'information

3 Système et Langages
- Rôle du système d'exploitation
- Topologie des langages de programmation

4 Conclusions

Sommaire

1. Le rôle de l'architecture
   - Le processeur
   - La mémoire
   - Multi-cœurs, multi-nœuds
   - Evolution des architectures
2. Représentation de l'information
   - Du binaire au flottant
   - Joies et déboires du calcul flottant
   - Erreurs : quelle confiance dans les calculs ?
3. Système et Langages
   - Rôle du système d'exploitation
   - Topologie des langages de programmation
4. Conclusions

Impact sur la performance

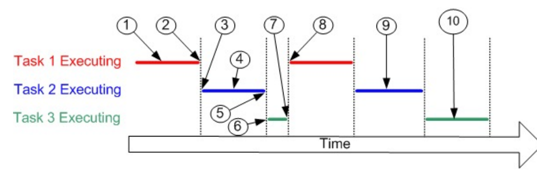
Le rôle du système d'exploitation n'est pas anodin :

- L'OS est chargé du lancement des processus.
- L'OS attribue à chaque processus un identifiant nommé pid (process id).
- L'OS attribue un environnement mémoire au processus.
- L'OS contrôle l'exécution d'un processus, il peut l'interrompre, l'arrêter, le faire continuer, etc.
- Un OS multitâche est capable d'entrelacer l'exécution de ses processus. On peut donc exécuter plusieurs processus sur un même processeur.

Conséquences

L'exécution d'un code est très dépendant de ce qui tourne déjà !

Gestion des processus



- ps : liste des processus et de leurs caractéristiques
- top ou htop : états des processus
- kill : suppression plus ou moins radicale des processus
- nice : gestion de la priorité du processus (de -20 à 19 pour un processus prioritaire)

Démo

Sommaire

1 Le rôle de l'architecture
- Le processeur
- La mémoire
- Multi-cœurs, multi-nœuds
- Evolution des architectures

2 Représentation de l'information
- Du binaire au flottant
- Joies et déboires du calcul flottant
- Erreurs : quelle confiance dans les calculs ?

3 Système et Langages
- Rôle du système d'exploitation
- Topologie des langages de programmation

4 Conclusions

Quel langage de programmation faut-il choisir ?

Pragmatisme !

Un langage qui assure :

- la performance
- la lisibilité/maintenabilité
- l'adaptabilité au calcul parallèle
- de ne pas réinventer la roue : utiliser ce qui existe déjà

Quel langage de programmation faut-il choisir ?

Pragmatisme !

Un langage qui assure :

- la performance
- la lisibilité/maintenabilité
- l'adaptabilité au calcul parallèle
- de ne pas réinventer la roue : utiliser ce qui existe déjà

Les langages les plus courants du calcul

C, C++, Fortran, Python

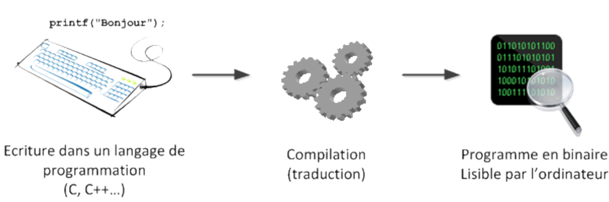
Quelles différences entre ces langages ? Avantages, inconvénients ?

Langage compilé / Langage interprété

- Langage compilé : nécessite un compilateur qui traduit le code source en instructions lisibles par la machine une fois pour toutes.
  Fortran, C, C++ ...
- Langage interprété : nécessite un interpréteur qui vérifie la syntaxe et traduit les lignes du code source au fur et à mesure de leur exécution.
  Python, Matlab, R ...

Rôle du compilateur

Traduction d'un langage source (en pratique un code source) vers un langage cible (en pratique un code objet ou un code binaire exécutable par la machine).

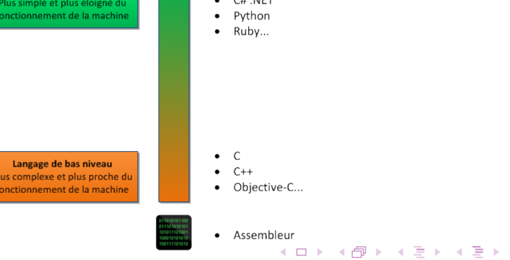


Haut niveau / Bas niveau

Niveau d'un langage

En fonction de la nécessite imposée au programmeur de connaître le fonctionnement d'un ordinateur.
Plus le niveau est bas, plus les performances sont importantes.

- Fortran, C et C++ sont considérés comme des langages de bas niveau. Ils permettent notamment une gestion fine de la mémoire.
- Python est un langage de haut niveau de même que Matlab/Scilab ou R appelés parfois « environnements de programmation »



Compilé vs interprété ?

- Programme en langage interprété : traduction en langage machine au fur et à mesure, un peu comme un interprète durant une interview.
- Un programme interprété sera plus lent qu'un programme compilé du fait de la traduction dynamique. Le compilateur peut effectuer des optimisation globales car il traite l'ensemble du programme.
- Néanmoins la correction des erreurs sera plus simple car l'interprète signale à l'exécution où se trouve l'erreur. Et le code source venant d'être écrit peut être directement testé.
- Alors qu'avec un langage compilé, chaque modification du code nécessitera une nouvelle compilation pour être prise en compte.

Typage statique et dynamique

Typage statique

Les langages de typage fort dit statique imposent la déclaration précise de toutes les variables (type, signe, taille) et les éventuelles conversions doivent être explicites.
Rigidité mais sécurité.
Fortran, C++ (NB : mot clé auto en C++11)

Typage dynamique

Les langages non typés ou de typage dynamique sont très souples avec les variables : pas de déclaration et possibilité de changement de type à la volée. Le temps de développement est réduit (moins verbose) mais des erreurs non détectables sont possibles (faute de frappe dans le nom de la variable). La grande flexibilité que permet le typage dynamique se paye en général par une surconsommation de mémoire correspondant à l'encodage du type dans la valeur.
Python, R, Matlab

Langage objet ou procédural

Langage procédural

Enchaînement de procédures/fonctions sur des données globales.
C, Fortran, Matlab ou R sont des langages procéduraux (malgré des évolutions)

Langage orienté objet

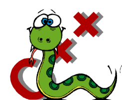
Regrouper les données et les fonctionnalités associées en entités logicielles autonomes
C++ et Python sont nativement des langages orientés objet

Interopérabilité : Prendre le meilleur de chaque monde

LA tendance : interfacer les langages pour prendre le meilleur de chacun, i.e. permettre la transmission des données entre plusieurs parties du code implémentées dans différents langages.

Schéma classique d'un code de calcul

1. lecture des données + pré-traitement : langage haut niveau interprété
2. étapes de calcul : langage bas niveau compilé
3. post-traitement, visualisation et/ou écriture des données : langage haut niveau interprété



Sommaire

1 Le rôle de l'architecture

2 Représentation de l'information

3 Système et Langages

4 Conclusions

Conclusions

Si vous êtes là, c'est que vous vous posez des questions de performances.

La performance est multi-factorielle !

- Elle dépend de l'architecture de la machine
- Elle dépend du langage, de la façon dont le programme est écrit
- Elle dépend de la charge de la machine, de la façon dont l'OS gère les processus
- Elle dépend des algorithmes et méthodes choisies ...