AlGorille
Project Team

Calcul sur GPUs

Journée de prospective industrielle d'ASR
UPMC-Paris 6, 3/11/2011

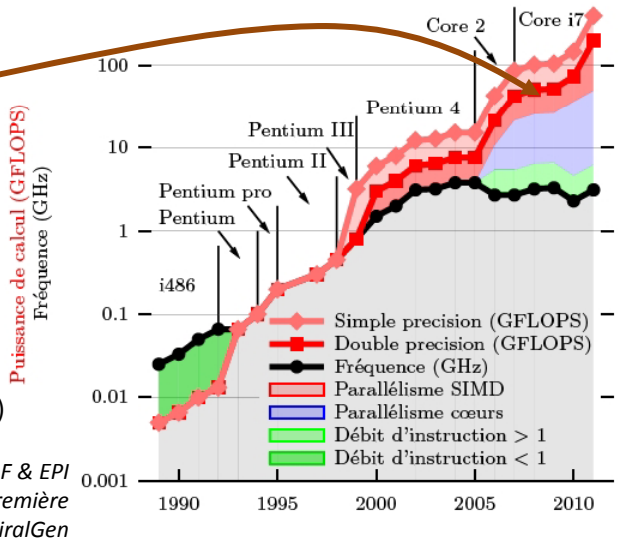
Stephane Vialle
SUPELEC – UMI GT-CNRS 2958 &
AlGorille INRIA Project Team

I – Les unités vectorielles s'invitent dans les nœuds de calculs

Evolution des CPUs

Chaque cœur CPU inclut une petite unité vectorielle (ex: SSE ou AVX)

- La puissance des CPUs n'augmentent plus qu'avec le parallélisme
- Les CPUs cumulent un grain moyen et un grain fin de parallélisme (multi-cœurs + unités vect.)



W. Kirshenmann, EDF & EPI AlGorille, d'après une première étude menée par SpiralGen

Calcul « vectoriel »

Type de code source	Exécution
`float a, b, r;` `r = a+b;` — scalaire	séquentielle

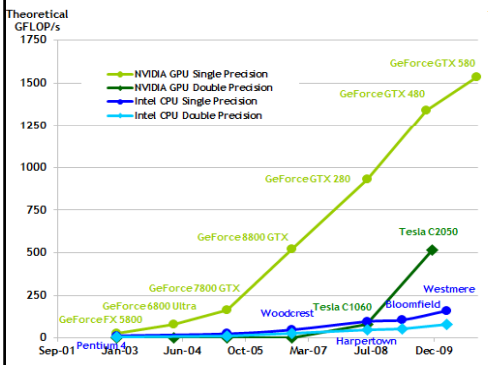
`float a[N], b[N], r[N ];`
`for (int i = 0; i < N; i++)`
`  r[i] = a[i] + b[i];` — scalaire, mais vectorisable
	séquentielle, mais parallélisable

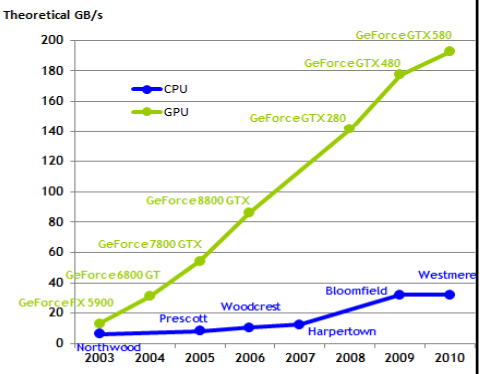
`float a[N], b[N], r[N];` `  r = a + b;` — vectoriel	séquentielle sur un cœur scalaire, ou parallèle sur un multi-cœurs scalaires, ou **massivement parallèle** sur des unités vectorielles

Possibilité d'accélérations importantes

Les unités vectorielles s'invitent dans les nœuds de calcul

Evolution des GPUs





Le GPU est un co-processeur vectoriel puissant et bon marché, porté par le jeu vidéo

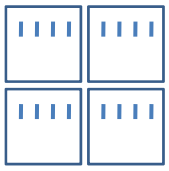
Le gain de puissance peut être significatif (x100), ou modeste (x2), ou inexistant !

- identifier les cas favorables au GPGPU
- savoir à quoi se comparer pour mesurer le gain de perf.
- savoir écrire des codes GPUs optimisés

Les unités vectorielles s'invitent dans les nœuds de calcul

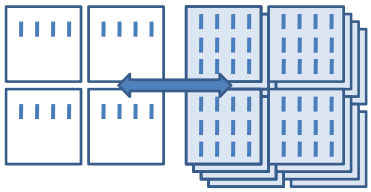
CPUs et unités vectorielles

1 CPU multi-cœurs avec des unités SSE/AVX



1 CPU multi-cœurs dont un cœur GPU

1 CPU multi-cœurs associé à un (ou *n*) GPU many-coeurs



On trouve des unités vectorielles à divers niveaux vis-à-vis du CPU :

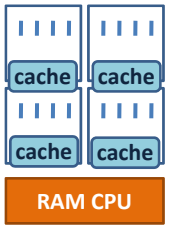
- petites et dans les cœurs CPUs
- moyennes dans un cœur spécifique du processeur
- puissantes dans la carte fille à coté de la carte mère

→ Quelle(s) unités vestorielles utiliser ?

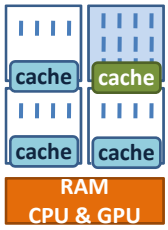
Les unités vectorielles s'invitent dans les nœuds de calcul

Multiplication des mémoires

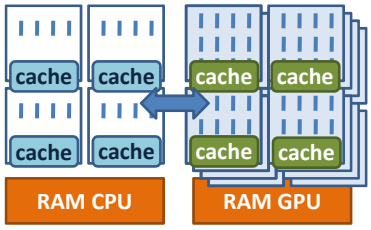
1 CPU multi-cœurs avec des unités SSE/AVX



1 CPU multi-cœurs dont un cœur GPU



1 CPU multi-cœurs associé à un (ou *n*) GPU many-coeurs



Plus on utilise un co-processeur vectoriel puissant :

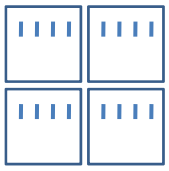
- plus il a sa propre mémoire (RAM et cache)
- plus il faut de temps pour transférer les données entre les cœurs CPUs et les cœurs vectoriels

→ Ne pas négliger les temps de transferts des données

Les unités vectorielles s'invitent dans les nœuds de calcul

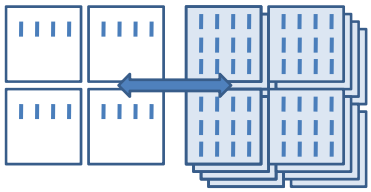
Complexification de la gestion des données

1 CPU multi-cœurs avec des unités SSE/AVX



1 CPU multi-cœurs dont un cœur GPU

1 CPU multi-cœurs associé à un (ou *n*) GPU many-coeurs



Pour être efficaces les unités vectorielles demandent :

- **un stockage des données dans un ordre « favorable »**
- **des accès aux données « dans le bon ordre » (accès « coalescents » sur GPUs)**

Pour chaque technologie vectorielle la stratégie de stockage et d'accès est différente !

Etude menée avec EDF

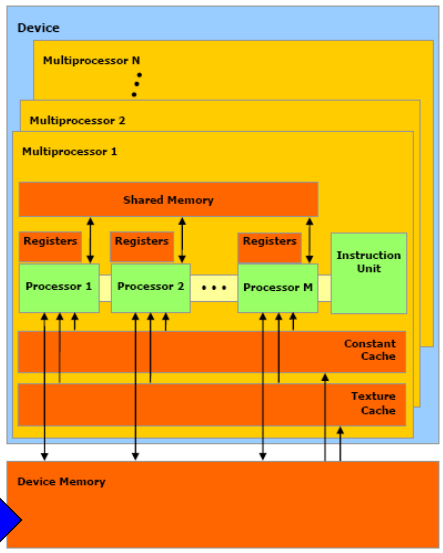
→ Portage des calculs + portage du stockage et de l'ordre d'accès aux données
→ Quelle pérennité des codes de calculs ?

II – Programmation GPGPU native
(en CUDA sur GPU Nvidia)

Aspects matériels

Un **« GPU » Nvidia** est :

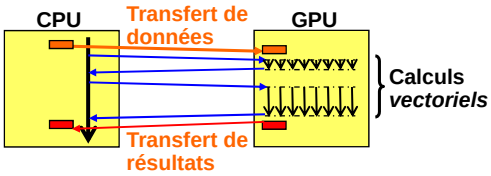
- un ensemble de machines vectorielles indépendantes partageant une mémoire globale
- une hiérarchie de mémoires de vitesses et de volumes différents
- un co-processeur vectoriel sous le contrôle du CPU

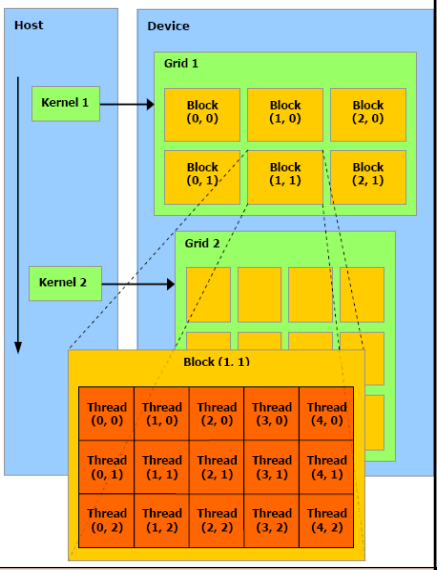


Aspects logiciels

Un **programme CPU+GPU** est :

- un programme CPU séquentiel ou multi-threads (mono ou multi-cœurs)
- des appels de fonctions vectorielles exécutées sur le GPU (« rpc vectoriel »)
- une décomposition explicite des calculs vectoriels en une grille de blocs de threads





Liaison matériel-logiciel

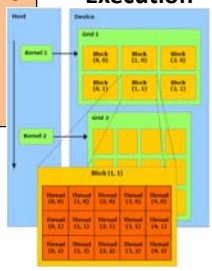
Programmation

- Du code d'un calcul élémentaire sur une unité vectorielle
- De la définition de la grille de blocs de threads GPU
- De l'exécution de cette grille de blocs

Ordonnancement par le GPU

- des blocs (de la grille) sur les machines vectorielles du GPU
- des threads (d'un bloc) sur les unités de calculs d'une machine vectorielle

Exécution



Possibilité d'influer et d'**optimiser** l'ordonnancement et l'exécution du calcul vectoriel ….
… depuis le code source.

Difficultés de la prog. GPGPU native

- Programmation à base de tableaux et d'indices d'accès
  → **nombreuses sources de « bugs »**
- Mise au point difficile : pas d'outil de debug confortable sur le « périphérique GPU »
  → **temps de développement élevés**
- Obtention de bonnes performances pas facile sur des applications réelles et complexes :
  - Optimisation des stockage et accès aux données
  - Optimisation par « programmation explicite du cache »
  - Optimisation de la granularité de la grille de blocs de threads
  - Optimisation de l'enchainement des calculs et des transferts CPU/GPU

  → **démarche demandant de l'expertise et du temps**

III – Modèles, outils et stratégie de développement

Modèles, outils et stratégie de développement

Développement mono-noeud

Un nœud de calcul des clusters de GPU de Supélec

1 CPU multi-cœurs + 1 GPU many-cœurs		
C/C++	Multithreading CPU	Multithreading GPU
gcc, icc, …	Pthreads, Windows Threads, OpenMP, TBB…	CUDA, OpenCL
gcc, icc, …	HMPP (CAPS)	
gcc, icc, …	Autres env. de haut niveau en cours de dev.	

Compétences requises

- Pouvoir combiner différents environnements de développement
- Savoir combiner différents modèles et paradigmes de programmation
- Savoir mettre au point et optimiser des architectures hybrides (CPUs + GPUs)

Modèles, outils et stratégie de développement

Développement multi-noeuds

Supercalculateur Bull « Titane » installé au CCRT (CEA)

→ On cumule **3 paradigmes** de programmation parallèle (pour grains gros, moyen et fin)

1 CPU multi-cœurs + 1 GPU many-cœurs			
C/C++	Envois de messages	Multithreading CPU	Multithreading GPU
gcc, icc, …	MPI, …	Pthreads, Windows Threads, OpenMP, TBB…	CUDA, OpenCL
gcc, icc, …	MPI (?)	HMPP (CAPS)	
gcc, icc, …	Autres env. de haut niveau en cours de dev.		CUDA, OpenCL
gcc, icc, …	Autres env. de haut niveau en cours de dev.		

Modèles, outils et stratégie de développement

Développements à base de bibliothèques vectorielles

Code C/C++ (+ multithreading CPU + envois de messages CPU)
+ appels de fonctions de bibliothèques vectorielles

Ex : bibliothèques GPU de FFT, BLAS, RNG, autres …

→ Surcout de développement : faible

Expertise complémentaire nécessaire : faible

Accroissement des performances : significatif **si**
- le problème se prête à une vectorisation
- les bibliothèques sont adaptées au problème (à tout le problème)
- les transferts de données CPU – GPU nécessaires à l'utilisation des bibliothèques sont limités

Modèles, outils et stratégie de développement

Développements complets de codes vectoriels

Code C/C++ (+ multithreading CPU + envois de messages CPU)
+ code GPU dédié au problème
(+ appels de fonctions de bibliothèques vectorielles)

→ Surcout de développement : élevé

Expertise complémentaire nécessaire : élevée

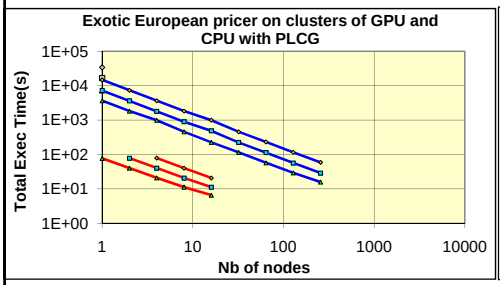
Accroissement des performances : significatif et meilleur **si**
- le problème se prête à une vectorisation
- les codes GPU développés sont correctement optimisés

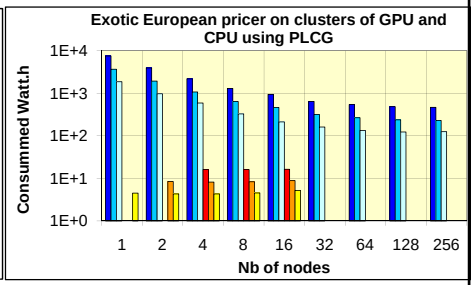
→ **Validation du code hybride : très important !**
- **comparaison des résultats à ceux d'un code purement CPU**
- **développement et validation « *step by step* » : noyau par noyau**

IV – Exemples de performances

Exemple de gains très importants

Calculs de Monte-Carlo totalement réalisés sur GPU





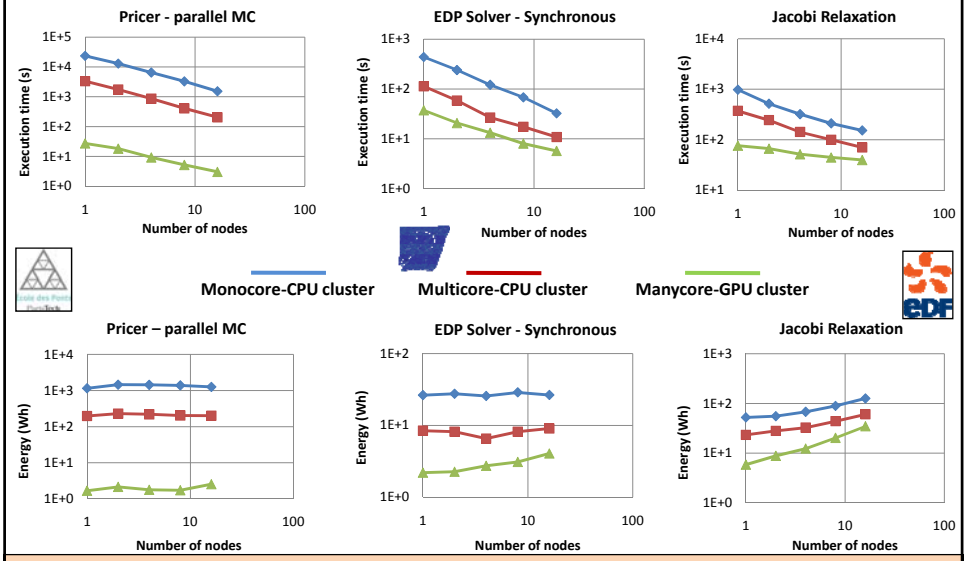
Gros travail de portage complet des calculs sur GPUs

Calculs sur GPU sans transferts CPU – GPU (sauf à l'initialisation et à la fin du pgm)

Calculs distribués « *embarrassingly parallel* » sur cluster (pas de communications)

Exemple de performances

Exemples de gains plus variées



IV – Conclusion et perspectives

Bilan actuel du GPGPU

Les clusters hybrides sont de plus en plus nombreux (grands ou petits)

Certains codes hybrides sont très performants

Certaines sociétés ont beaucoup investi dans le GPGPU
Ex : Energie: Total
Finance : Banques

Développer un code GPU complet reste une tâche longue et complexe

Les unités vectorielles des cœurs CPU gagnent en puissance
Intel « *sandy bridge* » *vs* GPU ???

→ Quels sont les investissements pérennes … ?

Vision (personnelle) de l'avenir

Les unités vectorielles sont et seront présentes dans nos calculateurs sous une forme ou sous une autre (il y a la place dans les puces, et il y a des besoins)

Programmer des unités vectorielles demande une compétence spécifique (algorithmique, programmation mise au point, optimisation)

Les modèles de programmation vectoriels sont liés à l'architecture pour être efficace (format de stockage des données et stratégie d'accès aux données), ils devraient le rester pendant « quelques années »

→ Investir dans de l'acquisition de compétences en calcul vectoriel.
Apprendre à utiliser les unités vectorielles disponibles dès aujourd'hui.

Isoler les noyaux vectoriels du reste du code.
Prévoir que le format de stockage des « vecteurs » peut changer.

Comparer les performances des codes hybrides à celles des codes scalaires optimisés pour juger l'intérêt d'un système hybride.

Calculs sur GPUs

Questions ?