Technologies des réseaux et NTICs

Journée Calcul Hybride

« le projet OpenGPU : un an plus tard »

Mercredi 8 juin 2011

Coordination scientifique :

- Association Aristote et le projet `OpenGPU`

Amphithéâtre Gay-Lussac, École Polytechnique, Palaiseau

`http://www.association-aristote.fr` **`info@association-aristote.fr`**

Edition du 4 messidor an CCXIX (*vulg.* 23 juin 2011)

ARISTOTE Association Loi de 1901. Siège social : CEA-DSI CEN Saclay Bât. 474, 91191 Gif-sur-Yvette Cedex.
Secrétariat : Aristote, École Polytechnique, 91128 Palaiseau Cedex.
Tél. : +33(0)1 69 33 99 66 Fax : +33(0)1 69 33 99 67 Courriel : `Marie.Tetard@polytechnique.edu`
Site internet `http://www.association-aristote.fr`

Table des matières

Chapitre 1

Programme de la journée

1.1 Introduction

Un an après son lancement au mois de mars 2010, les partenaires du projet `OpenGPU` organisent un colloque national pour partager avec la communauté scientifique les premiers résultats quantitatifs et opérationnels obtenus par les différentes équipes de recherche.

Cette journée, organisée en partenariat avec l'association Aristote, sera suivie d'une journée consacrée aux nouvelles tendances du *Big Data* pour lesquelles les architectures hybrides `GPU/CPU` joueront un rôle essentiel.

Les partenaires de l'événement

1.2 Programme

8h30-9h15	*Accueil des participants, café*	
9h15-9h40	**Ouverture du séminaire**	
	Jean-Noël de Galzain (Wallix) **Eric Mahé** (Minds Planet)	`OpenGPU` : un projet au cœur des évolutions technologiques
9h40-10h45	**Les outils d'aide à la parallélisation**	
	Mathias Bourgoin (LIP6)	Le langage `CAML` et la programmation des `GPU`
	Michel Barreteau (Thales Research & Technology)	De l'`OpenCL` performant par couplage d'outils de parallélisation
	Tarik Saidani (AS+)	Retours d'utilisation du langage `OpenCL`
10h45-11h15	*Pause café*	
11h15-12h10	**Philippe Coucaud** (Ateji)	`Java` sur `GPU` avec `Ateji PX`
	Guillaume BARAT (CAPS)	Migration *manycore* : méthodologie et outils
	Basile Starynkevitch (CEA)	`OpenGPU` et `GCC-MELT`
	Questions-réponses avec les orateurs de la matinée	
12h10-13h45	*Repas (salle Detoeuf)*	
13h45-16h00	**Applications opérationnelles des architectures hybrides**	
	Vivien Clauzon (Numtech)	Retours d'expérience d'une `PME` sur le `GPGPU` pour des applications de dispersion atmosphérique
	Jean-Pierre Panziera (Bull)	Vision de `Bull` sur les architectures hybrides.
	Rémi Barrère (Thales Research & Technology)	Évaluation d'OpenCL et intégration des enseignements pour une programmation performante à haut niveau
	Pierre Leca (CEA - DAM)	Le prototype de *cluster* `OpenGPU` au `CEA-DAM` Contexte et retour d'expérience
	Antoine Petitet et **Stefanos Vlachoutsis** (ESI-Group)	Premiers retours d'expérience sur l'utilisation de `GPU` pour des applications de mecanique des structures
	Fouad Boumezbeur (INRA)	Applications du `GPU` à l'investigation métagénomique : illustration avec le projet européen `MetaHIT`
16h00-16h20	*Pause*	
16h20-17h25	**Parallélisation : la place de l'*open source***	
	Ronan Keryell (HPC Project)	L'outil de parallélisation automatique `Par4All`
	Sylvestre Ledru (Scilab)	Retour sur l'intégration de fonctionnalités `GPGPU` dans `Scilab`
	Albert Cohen (INRIA)	Parallélisation automatique de noyaux de calcul pour `GPGPU`
17h25-18h15	**Conférences invitées**	
	Denis Caromel (INRIA/ActiveEON)	Orchestration de *Workflows* `CPU` et `GPU` avec `OW2` ProActive Parallel Suite
	Yann Le Du (ENSCP)	`HPU4Science`, calcul haute performance sur `CPU/GPU` avec du matériel grand public : conception et choix logiciels
	Questions-réponses avec les orateurs de l'après-midi	
18h15-19h30	***Cocktail (Salon de Marbre)***	

Chapitre 2

Présentations

Photos ©2011 Philippe Lavialle

Jean-Michel Batto (INRA), Marie Tétard (Aristote), Éric Mahé (Minds Planet)

2.1 Jean-Noël de Galzain (Wallix), Eric Mahé (Minds Planet)

OpenGPU : un projet au cœur des évolutions technologiques

Agenda - Introduction

- Le projet OpenGPU
- Ecosystème
- La pertinence des choix technologiques
- Les évolutions possibles du projet OpenGPU

Le projet OpenGPU

- Enjeux
  - » Plate-forme logicielle d'aide à la parallélisation (SP2)
    - » Compilateurs, translateurs, intégration Eclipse
  - » Plate-forme matérielle (SP3)
    - » Tests d'architectures hybrides CPU-GPU
  - » Codes industriels et académiques optimisés (SP4)
    - » Evaluation des performances de parallélisation
- Acteurs et partenaires
  - » Grands groupes, PME, laboratoires universitaires, associations
- Chiffres
  - » Durée : 2 ans, 19 partenaires, + 5M€

Ecosystème

- Positionnement dans l'écosystème des Pôles
  - » Labellisation et coordination avec System@tic, Cap Digital, Ter@tec
- Synergies Industriels / PME / Académiques
  - » Echanges technologiques et « bonnes pratiques »
- Engagement dans les futures directions
  - » Investissements d'avenir (Cloud Computing), IRT, FUI
  - » Dynamique de l'écosystème
    - » Open Source, Nvidia, OpenCL, HPC, multimédia, ...

La pertinence des choix technologiques

- Evolution de la puissance des GPU
  - » 2009 : 1 Tflops → 2011 : 2,7 Tflops → 2012 : 4 Tflops ?
- APU : intégration poussée CPU – GPU
  - » Sandy Bridge/Intel, Fusion/AMD
- ARM vs X86 : une (r)évolution importante
  - » Puissance élevée et consommation réduite
- OpenCL : le consensus
  - » Version 1.1
  - » Nombreux outils : Intel, AMD, Nvidia, IBM, ...

Evolutions possibles du Projet OpenGPU

- Une orientation « Big Data »
  - » Utiliser la puissance des GPUs pour les très gros volumes de données
  - » Journée Aristote du 9 juin
- Open GPU2
  - » Prédiction de performances pour architectures hybrides
  - » Dérivé d'un projet ITEA
- OpenKGPU
  - » Optimisation des mécanismes GPU au niveau du noyau Linux
  - » Réutilisation des travaux du projet OpenGPU


Remerciements
Partenaires écosystèmes
SYSTEMATIC
cap-digital
Caristote
CARRI Systems
Ter@tec
Partenaires financiers
Essonne
dgcis
hauts-de-seine
CONSEIL GÉNÉRAL
Partenaires du projet
BULL
THALES
TOTAL
WALLIX
CAPS
NUMTECH
HPC
ARMINES
Scilab
CENTRALE
GENCI
ibisc
LIP6
INRA
INRIA
openGPU

2.2 Mathias Bourgoin (LIP6)

Le langage CAML et la programmation des GPU

Le langage OCaml et la programmation des GPU

GPU programming with OCaml

Mathias Bourgoin - Emmanuel Chailloux - Jean-Luc Lamotte

Le projet OpenGPU : un an plus tard
Ecole Polytechnique - 8 juin 2011

Outline

GPGPU Programming

Two main frameworks
- **Cuda**
- **OpenCL**

Different Languages
- To write kernels
  - **Assembly** (ptx, il,...)
  - subsets of **C/C++**
- To manage kernels
  - ***C/C++/Objective-C***
  - Fortran
  - Python
  - Scala
  - ***Java***
  - ***Scilab***
  - ...

Who needs GPGPU

Two kinds of programmers

- HPC / Scientific
  - Known Hardware
  - Heavy Optimisation
  - Problem Driven
- General Purpose
  - Unknown Hardware/Multiplatform
  - Light Optimisation
  - User Driven

Main Difficulties

- From the managing program
  - Memory transfers
  - Multiple Devices management
  - Many different kind of Devices
- From the kernel
  - Highly parallel
  - Different levels of parallism
  - Different kinds of memory (global, local,...)

What for?

- Data parallelism
- Distributed Computation
- Hopefully High Speed-Ups

OCaml

- High-Level language
  - **Statically Typed**
  - **Type inference**
  - **Multiparadigm** (imperative, object, functionnal, modular)
  - Compile to **Bytecode/native Code**
  - Memory Manager (very efficient **Garbage Collector**)
  - Interactive **Toplevel** (to learn, test and debug)
  - **Interoperability with C**
- Portable
  - System : Windows - Unix (OS-X, Linux...)
  - Architecture : x86, x86-64, PowerPC, ARM...

Caml
caml.inria.fr/

OCaml - Usage

Domains
- Education
- Research
- Industry

Software Examples
- The Coq Proof Assistant
- LexiFi's Modeling Language for Finance
- FFTW
- Scade Suite

The Caml Consortium:

LexiFi, mylife, Dassault Aviation, Dassault Systèmes, Esterel Technologies, Citrix, OCamlCore, Microsoft, SimCorp, mlstate, Jane Street Capital

Motivations

OCaml and GPGPU complement each other

GPGPU frameworks are
- Highly Parallel
- Architecture Sensitive
- Very Low-Level

Ocaml is
- Mainly Sequential
- Multi-platform/architecture
- Very High-Level

Idea

- Allow OCaml developers to use GPGPU with their favorite language.
- Use OCaml to develop high level abstractions for GPGPU.
- Make GPGPU programming safer and easier

Main Objectives

Goals

- Allow complete use of Cuda/OpenCL frameworks with OCaml
- Abstract these two frameworks
- Abstract memory
- Abstract memory transfers
- Use OCaml type-checking to ensure kernels type safety
- Propose Abstractions for GPGPU programming

Solution

SPOC (***S**tream **P**rogramming **OC**aml*)

SPOC: Abstracting frameworks

Our choice

- **Dynamic linking**.
- The Cuda implementation uses the Cuda Driver API instead of the Runtime Library (lower level API, does not need the cudart library which is only provided with the Cuda SDK).

Compilation doesn't need any specific hardware
(no need of a Cuda/OpenCL compatible Device) or SDK.

Allows

- development **for multiple architectures from a single system**;
- executables to use **any OpenCL/Cuda Devices conjointly**;
- distribution of a **single executable for multiple architectures**.

SPOC: Abstracting Transfers

Automatic Transfers

Vectors automatically move from CPU to Devices

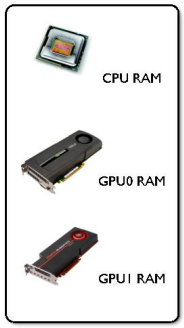
- When a cpu function uses a vector, SPOC moves it to the CPU RAM
- When a kernel uses a vector, SPOC moves it to the Device Global Memory
- Unused vectors do not move
- SPOC allows users to explicitely force transfers

OCaml memory manager

Vectors are managed by the OCaml memory manager

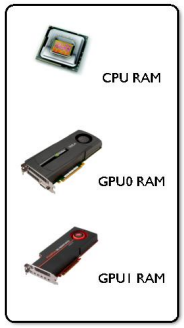
- **Automatic allocation** when created
- The garbage collector **automatically frees** vectors (on the CPU or on Devices)
- Allocation failure during a transfer triggers a collection

A Little Example



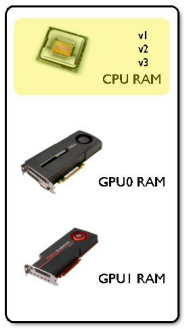

```
1  open Spoc
2  let dev = Devices.init ()
3  let n = 1_000_000
4  let v1 = Vector.create Vector.float32 n
5  let v2 = Vector.create Vector.float32 n
6  let v3 = Vector.create Vector.float32 n
7  let k = vector_add v3 v1 v2 n
8  let block = {blockX = 1024; blockY = 1; blockZ = 1}
9  let grid = {gridX = (n+1024-1)/1024; gridY = 1; gridZ = 1}
10
11 let main () =
12   random_fill(v1);
13   random_fill(v2);
14   Kernel.run dev.(0) (block,grid) k;
15   for i = 0 to Vector.length v3 do
16    Printf.printf "res[%d] = %f; "
17      i (Mem.get v3 i)
18   done;
```


A Little Example



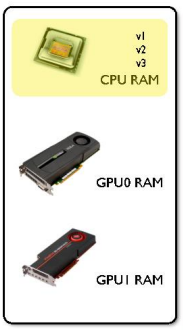

```
1  open Spoc
2   let dev = Devices.init ()
3  let n = 1_000_000
4  let v1 = Vector.create Vector.float32 n
5  let v2 = Vector.create Vector.float32 n
6  let v3 = Vector.create Vector.float32 n  let k = vec-
tor_add v3 v1 v2 n
7  let block = {blockX = 1024; blockY = 1; blockZ = 1}
8  let grid = {gridX = (n+1024-1)/1024; gridY = 1; gridZ = 1}
9  let main () =
10   random_fill(v1);
11   random_fill(v2);
12   Kernel.run dev.(0) (block,grid) k;
13   for i = 0 to Vector.length v3 do
14    Printf.printf "res[%d] = %f; "
15      i (Mem.get v3 i)
16   done;
```

A Little Example



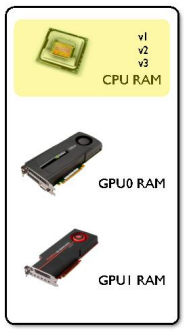

```
1 open Spoc
2 let dev = Devices.init ()
3 let n = 1_000_000
4 let v1 = Vector.create Vector.float32 n
5 let v2 = Vector.create Vector.float32 n
6 let v3 = Vector.create Vector.float32 n
7 let k = vector_add v3 v1 v2 n
8 let block = {blockX = 1024; blockY = 1; blockZ = 1}
9 let grid = {gridX = (n+1024-1)/1024; gridY = 1; gridZ = 1}
10 let main () =
11   random_fill(v1);
12   random_fill(v2);
13   Kernel.run dev.(0) (block,grid) k;
14   for i = 0 to Vector.length v3 do
15     Printf.printf "res[%d] = %f; "
16       i (Mem.get v3 i)
17   done;
```


A Little Example



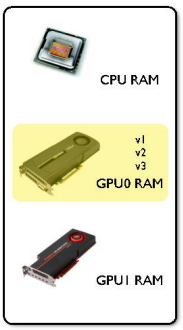

```
1 open Spoc
2 let dev = Devices.init ()
3 let n = 1_000_000
4 let v1 = Vector.create Vector.float32 n
5 let v2 = Vector.create Vector.float32 n
6 let v3 = Vector.create Vector.float32 n
7 let k = vector_add v3 v1 v2 n
8 let block = {blockX = 1024; blockY = 1; blockZ = 1}
9 let grid = {gridX = (n+1024-1)/1024; gridY = 1; gridZ = 1}
10 let main () =
11   random_fill(v1);
12   random_fill(v2);
13   Kernel.run dev.(0) (block,grid) k;
14   for i = 0 to Vector.length v3 do
15     Printf.printf "res[%d] = %f; "
16       i (Mem.get v3 i)
17   done;
```


A Little Example



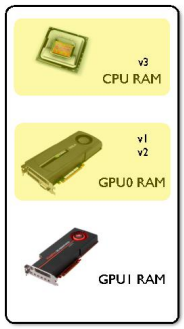

```
1 open Spoc
2 let dev = Devices.init ()
3 let n = 1_000_000
4 let v1 = Vector.create Vector.float32 n
5 let v2 = Vector.create Vector.float32 n
6 let v3 = Vector.create Vector.float32 n
7 let k = vector_add v3 v1 v2 n
8 let block = {blockX = 1024; blockY = 1; blockZ = 1}
9 let grid = {gridX = (n+1024-1)/1024; gridY = 1; gridZ = 1}
10 let main () =
11   random_fill(v1);
12   random_fill(v2);
13   Kernel.run dev.(0) (block,grid) k;
14   for i = 0 to Vector.length v3 do
15     Printf.printf "res[%d] = %f; "
16       i (Mem.get v3 i)
17   done;
```


A Little Example




```
1 open Spoc
2 let dev = Devices.init ()
3 let n = 1_000_000
4 let v1 = Vector.create Vector.float32 n
5 let v2 = Vector.create Vector.float32 n
6 let v3 = Vector.create Vector.float32 n
7 let k = vector_add v3 v1 v2 n
8 let block = {blockX = 1024; blockY = 1; blockZ = 1}
9 let grid = {gridX = (n+1024-1)/1024; gridY = 1; gridZ = 1}
10 let main () =
11   random_fill(v1);
12   random_fill(v2);
13     Kernel.run dev.(0) (block,grid) k;
14   for i = 0 to Vector.length v3 do
15     Printf.printf "res[%d] = %f; "
16       i (Mem.get v3 i)
17   done;
```


A Little Example



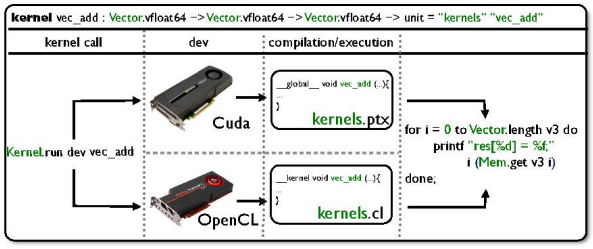

```
1 open Spoc
2 let dev = Devices.init ()
3 let n = 1_000_000
4 let v1 = Vector.create Vector.float32 n
5 let v2 = Vector.create Vector.float32 n
6 let v3 = Vector.create Vector.float32 n
7 let k = vector_add v3 v1 v2 n
8 let block = {blockX = 1024; blockY = 1; blockZ = 1}
9 let grid = {gridX = (n+1024-1)/1024; gridY = 1; gridZ = 1}
10 let main () =
11   random_fill(v1);
12   random_fill(v2);
13   Kernel.run dev.(0) (block,grid) k;
14     for i = 0 to Vector.length v3 do
15       Printf.printf "res[%d] = %f; "
16         i (Mem.get v3 i)
17     done;
```


Kernels

Type-Safe Kernel Declaration

kernel vector_add : Vector.vfloat64 → Vector.vfloat64 → Vector.vfloat64 → unit = "source_file" "kernel_name"

- Static arguments types checking (compilation time)
- Kernel.run compiles kernel from source (.ptx / .cl)



Errors

Type-Checking: Error at compile-time

```
1 kernel vector_sum: Vector.float64 → unit = "my_file" "kernel_sum"
2 let v = Vector.create Vector.float32 1024 in
3 Kernel.run device (block, grid) vector_sum v;
```

Type-Checking : Correct

```
1 kernel vector_sum: Vector.float64 → unit = "my_file" "kernel_sum"
2 let v = Vector.create Vector.float64 1024 in
3 Kernel.run device (block, grid) vector_sum v;
```

Exceptions

SPOC raises OCaml exceptions when

- Kernel compilation/execution fails
- Not enough memory on devices

Tools

Development tools : OCaml Interactive Toplevel

```
$ ./spoclevel_cublas
        Objective Caml version 3.12.0

        Camlp4 Parsing version 3.12.0

# open Spoc
  Random.self_init();
  let dev = (Devices.init()).(0) in
  let a = Vector.create Vector.float32 10240 in
  let res = ref 0. in
  for i = 0 to 10239 do
  let tmp = Random.float 32. in
  Mem.set a i tmp;
  res := !res +. tmp;
  done;
  let gpu_res = Cublas.run dev (Cublas.cublasSasum 10240 a 1) in
  (!res, gpu_res)
  ;;
- : float * float = (163561.160761084117, 163561.15625)
#
```


Tools

Development tools

- IDE (OCaide plugin for Eclipse)
  `http://www.algo-prog.info/ocaide/`
- Cublas (v1)
- Optimized vector iterators

MultiGPU?

```
open Spoc
let dev = Devices.init ()
let n = 1_000_000
let v1 = Vector.create Vector.float32 n
let v2 = Vector.create Vector.float32 n
let v3 = Vector.create Vector.float32 n

let k = vector_add v3 v1 v2 n

let block = {blockX = 1024; blockY = 1; blockZ = 1}
let grid = {gridX = (n+1024-1)/1024; gridY = 1; gridZ = 1}

let main () =
  random_fill(v1);
  random_fill(v2);
  Kernel.run dev.(0) (block,grid) k;
  for i = 0 to Vector.length v3 do
    Printf.printf "res[%d] = %f; " i (Mem.get v3 i)
  done;
```

```
open Spoc
let dev = Devices.init ()
let n = 1_000_000
let v1 = Vector.create Vector.float32 n
let v2 = Vector.create Vector.float32 n
let v3 = Vector.create Vector.float32 n
let v1_1 = Mem.sub_vector v1 0 (n/2)
let v1_2 = Mem.sub_vector v1 (n/2) (n/2)
let v2_1 = Mem.sub_vector v2 0 (n/2)
let v2_2 = Mem.sub_vector v2 (n/2) (n/2)
let v3_1 = Mem.sub_vector v3 0 (n/2)
let v3_2 = Mem.sub_vector v3 (n/2) (n/2)

let k1 = vector_add v3_1 v1_1 v2_2 (n/2)
let k2 = vector_add v3_2 v1_2 v2_2 (n/2)

let block = {blockX = 1024; blockY = 1; blockZ = 1}
let grid = {gridX = (n+1024-1)/(1024/2); gridY = 1; gridZ = 1}

let main () =
  random_fill(v1);
  random_fill(v2);
  Kernel.run dev.(0) (block,grid) k1;
  Kernel.run dev.(1) (block,grid) k2;

  Mem.to_cpu v3_1 ();
  Mem.to_cpu v3_2 ();
  Devices.flush dev.(0);
  Devices.flush dev.(1);

  for i = 0 to Vector.length v3 do
    Printf.printf "res[%d] = %f; " i (Mem.get v3 i)
  done;
```


Multi-GPU

Devices/Frameworks

- SPOC allows to use **any Device** from both frameworks **indifferently**
- SPOC allows to use **any Device** from both frameworks **conjointly**
- Tested with **Cuda** used **conjointly** with **OpenCL**
- Tested with **Tesla C2070** used **conjointly** with **AMD 6970**

Transfers

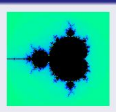
- Automatic Transfers **from CPU to Device**
- Automatic Transfers **from Device to CPU**
- Automatic Transfers **from Device to Device**

Benchmarks - 1

Spoc easily speeds OCaml programs up

Mandelbrot

- Naive implementation
- Non optimized kernels
- Graphic display handled by CPU



Mandelbrot

	OCaml Corei7 960	C i7 960	Spoc - Cuda NVIDIA C2070	Spoc - OpenCL AMD 6970	Cuda + OpenCL C2070 + 6970
Time	913s	741s	6.49s	9.70s	4.74s
speedups	x1	x1.2	x140.7	x94.2	x192.6

opencl kernel not vectorized

Benchmarks - 2

OCaml+Spoc actually useable as a kernel composition language

Matrix Multiply SP

- 2 optimized kernels
  - Nvidia → Cublas sgemm
  - AMD → kernel from AMD OpenCL SDK

Matrix Multiply SP

	Average Sequential	Cuda NVIDIA C2070	OpenCL AMD 6970	Cuda + OpenCL NVIDIA C2070 + AMD 6970
C	-	485 GFlops	330 GFlops	657 GFlops
OCaml	-	457 GFlops	322 GFlops	678 GFlops
Overhead	25.5%	5.7%	2.4%	-3.2%

Conclusion - SPOC

OCaml meets GPGPU

- OCaml developers can now use GPGPU programming
- SPOC allows to easily develop efficient GPGPU applications
  - Abstracted frameworks (Cuda/Opencl)
  - Automatic transfers
  - Kernel type safety
  - Efficient memory manager
- Can also be used as a tool for non OCaml developers
  - OCaml Toplevel allows to test kernels
  - OCaml can be used to quickly express new algorithms
  - Still possible to use C externals...

Conclusion - Future Work

Real world use case: PROP

- 2DRMP : Dimensional R-matrix propagation (Computer Physics Communications)
- Simulates electron scattering from H-like atoms and ions at intermediate energies
- Multi-Architecture: MultiCore, GPGPU, Clusters, GPU Clusters
- Translate from **Fortran + Cuda** to **OCaml+SPOC + Cuda/OpenCL**
- Test on Bull GPU Cluster

Objectives

- Use OCaml+Spoc to simplify parallelism and transfers
- Verify that OCaml and SPOC enhance **development speed**, **safety** and **maintainability** while keeping **high performances**

Thanks

SPOC sources : `http://www.algo-prog.info/spoc/`
Spoc is compatible with x86_64: Unix (Linux, Mac OS X), Windows

For more information
mathias.bourgoin@lip6.fr

2.3 Michel Barreteau (Thales Research & Technology)

De l'OpenCL performant par couplage d'outils de parallélisation

De l'OpenCL performant par couplage d'outils de parallélisation

Michel BARRETEAU (Thales Research & Technology)

SYSTEMATIC PARIS REGION SYSTEMS & ICT CLUSTER

Research & Technology

THALES

---

2 / Agenda

- *Contexte*
- *Motivations*
- *« Philosophies » des outils de parallélisation OpenGPU*
- *Complémentarités*
- *Couplages front-end et back-end*
- *Etat d'avancement*
- *Conclusions*

THALES

---

3 / Contexte

Architecture GPU

- Complexité (ex: hiérarchie mémoire) et hétérogénéité (ATI, NVIDIA) croissantes : une connaissance approfondie du matériel reste requise

OpenCL

- Standard : un même code peut-être supporté par : CPU, GPU (ATI, NVIDIA), APU (Fusion, Sandy Bridge), Cell … mais il faut souvent le retravailler pour obtenir de réelles performances
- Très verbeux comparé à CUDA

Programmation

- Le code « hôte » est pénible à implémenter et reste néanmoins clé pour obtenir des performances dignes de ce nom
- Un code legacy est quasiment systématiquement à ré-écrire pour être performant sur GPU

THALES

---

4 / Motivations pour l'outillage logiciel

Porter un code en OpenCL reste « lourd » mais incontournable si l'on vise la performance !

Approches

1. **Parallélisation automatique (HMPP, Par4All)**
   1. Retranscription manuelle du code legacy selon des règles de codage appropriées à l'outil et à la machine GPU.
   2. Ajout (conseillé) des directives de compilation
   3. Interprétation de ces directives pour générer automatiquement un code performant
   4. Exécution du code parallélisé
   5. Analyse des performances
   6. Amélioration du code parallélisé en fonction du feedback
2. **Parallélisation semi-automatique (SPEAR)**
   1. Capture graphique de l'application et de la machine GPU au niveau modèle
   2. Ajout manuel du code OpenCL optimisé pour chaque kernel
   3. Parallélisation guidée graphiquement par l'utilisateur
   4. Génération de code OpenCL automatique pour le reste de l'application
   5. Éventuellement modifier la parallélisation (design space exploration)

THALES

---

5 / « Philosophies » des outils

Par4All / HMPP

- Paralléliser itérativement (en étant guidé par des pragmas utilisateur de préférence) un code legacy sur architecture hybride pour obtenir rapidement les meilleures performances possibles

SPEAR

- À partir d'une capture à haut niveau (modèles), sélectionner la meilleure architecture (en terme de performance) adaptée à une classe d'applications en un temps raisonnable (comparé à l'existant) en produisant un code parallèle performant

THALES

---

6 / Complémentarités des outils

HMPP / Par4All sait :

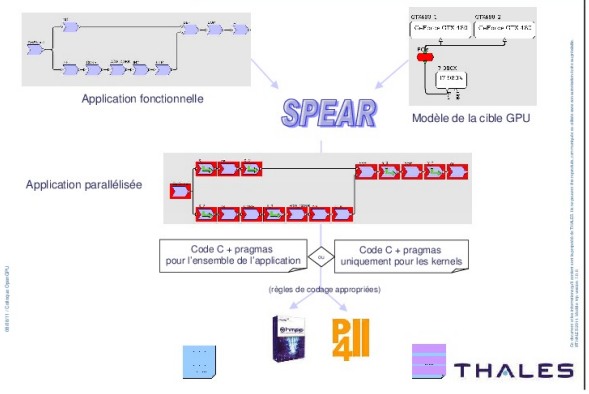
- analyser un code legacy selon certaines règles de codage (appropriées à l'analyse de dépendances)
  - Ex: pas de pointeur ni d'indirection
- gérer les transferts CPU / GPU avec l'appui de directives (de préférence)
  - Ex: éviter des communications multiples entre CPU et GPU
- optimiser le code des kernels sur les cibles choisies
  - … même si cela semble être réservé aux experts métier

SPEAR sait :

- introduire les directives de compilation correspondant à la parallélisation dictée par l'utilisateur
  - Ex: grouper les échanges de données pour réutiliser des tableaux
- générer les règles de codage souhaitées par les outils back-ends
  - Ex: pas de linéarisation de tableaux, accès affines en les indices de boucles

THALES

7 / Couplages « back-end » envisagés

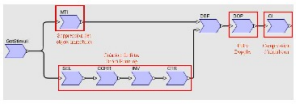


THALES

8 / Couplage « front-end »

PIPS : un outil d'analyse de code utilisable par SPEAR pour

- Extraire automatiquement les paramètres de parallélisme (relatifs à la tâche élémentaire sur laquelle itère la tâche de calcul)
  - Cf analyse de la tâche élémentaire COR (planche suivante)
- Importer un graphe d'application à partir d'une application C « bien écrite » (très rare)
  - Cf vidéo

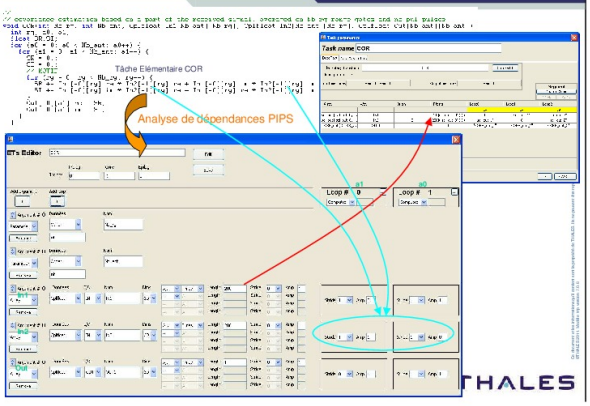


Objectif

- Présenter à l'utilisateur uniquement les informations spécifiques métier

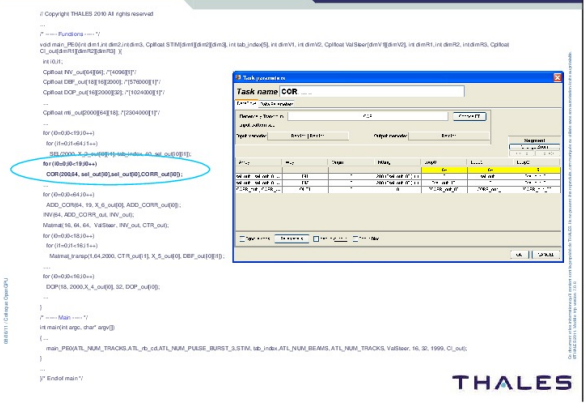
THALES

9 / Exemple d'analyse de code par PIPS



THALES

10 / Exemple d'application analysée par PIPS



THALES

11 / Etat d'avancement

PIPS – SPEAR

- En cours de finalisation

SPEAR – HMPP

- Parallélisation manuelle avec les pragmas HMPP
- Générateur de code SPEAR associé prévu pour S2/11

SPEAR – Par4All

- Discussion sur le choix des pragmas en cours (si besoin est !)
- Générateur de code SPEAR associé prévu pour S2/11

THALES

12 / Conclusions

Le choix d'un outil de parallélisation sur GPU dépend de plusieurs critères :

- « Préservation » d'un code legacy
- Optimisation manuelle des kernels imposée
- Nécessité des meilleures performances possibles
- « Habitudes » de l'utilisateur
- …

Coupler les outils de parallélisation permet de jouer sur plusieurs tableaux pour générer un code OpenCL performant

THALES

2.4 Tarik Saidani (AS+)

Retours d'utilisation du langage `OpenCL`

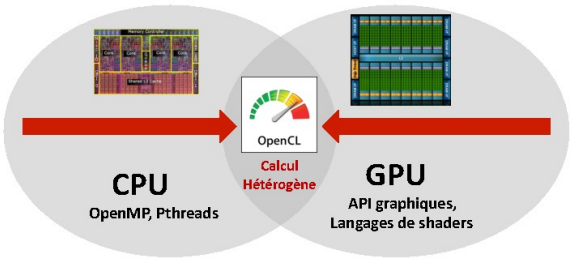
Retour d'expérience sur le calcul sur GPU en OpenCL

Tarik Saidani
Alliance Services Plus

Plan

- Aperçu d'OpenCL
- Implémentations disponibles
- CUDA Vs OpenCL
- Outils et librairies
- Futur d'OpenCL

Architectures parallèles



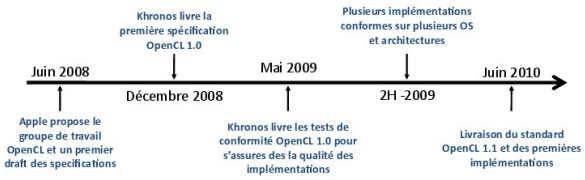
OpenCL est un environnement de programmation parallèle pour architectures hétérogènes

OpenCL Working Group

- Plusieurs acteurs de l'industrie
  - Fabricants de semi-conducteurs, OEM, éditeurs de middleware, développeurs d'applications
- Plusieurs leaders de l'industrie impliqué dans la conception
  - Richesse et généricité du modèle
- Apple a été à l'origine de la proposition et est très actif
  - Éditeur des spécifications

Chronologie d'OpenCL

- Six mois entre la proposition et les spécifications OpenCL 1.0
  - Forte interaction initiale et retombées commerciales importantes
- Plusieurs implémentations ont été livrées depuis
  - Mac OS X Snow Leopard est livré avec le support OpenCL
- Délais de 18 mois entre OpenCL 1.0 et OpenCL 1.1
  - Rétrocompatibilité



OpenCL 1.1

- Amélioration des fonctionnalités pour de meilleures performances et programabilité
  - Suite à feedback de la communauté des développeurs
- Nouvelles fonctionnalité rétro-comptabilité avec OpenCL 1.0
  - Nouveaux types de données
  - Gestion de commandes de plusieurs hosts
    - Utiliser des buffers partagés par plusieurs devices
  - Opérations sur une région de buffer
    - Lecture, écriture et copie de zones de 1D, 2D et 3D rectangulaires
  - Meilleures utilisation des events
  - De nouvelles fonctions natives OpenCL
  - Meilleure interaction avec OpenGL

ATI Stream DSK

- Version 2.0 08/2010
  - support partiel d'OpenCL
- Version 2.01 09/2010
  - Support complet OpenCL 1.0
- Version 2.3 12/2010
  - Support OpenCL 1.1
- Version 2.4 04/2011

Nvidia OpenCL

- OpenCL Early Access Program (ouvert au développeurs) 04/2009
- Premiers drivers compatibles OpenCL 1.0 09/2009
- Pre-release des drivers CUDA conforme OpenCL 1.1 06/2011

Intel® OpenCL SDK

- Premiers drivers compatibles OpenCL 1.0 11/2009
  - Support Windows 7/ Windows Vista
- Beta-release 1.1 conforme au standard OpenCL 1.1 05/2011
  - Ajout de support Linux 64-bit uniquement
    - Novell SUSE, RedHat Entreprise (paquet rpm)

Apple

- Initiateur d'OpenCL
- Mac OS X Snow Leopard 08/2009
  - Framework conforme OpenCL 1.0
- Support CPU Intel, GPU Nvidia et ATI

IBM

- Première version compatible OpenCL 1.0 06/2010
  - Pour le processeur Cell et les multi-core PowerPC
- Mise à jour compatible OpenCL 1.1 03/2011

CUDA vs OpenCL

- CUDA
  - Propriétaire Nvidia
  - GPU Nvidia, multi-os (Windows, Linux, MacOS)
  - Niveau de maturité avancé
  - Runtime API simple à utiliser
- OpenCL
  - Standard ouvert
  - GPU Nvidia, ATI , ARM, CPU : Intel, PowerPC, Cell
  - OpenCL 1.0 : niveau mature OpenCL 1.1 : encore en bêta
  - Platform Layer API+ Runtime API, verbeuses, bas-niveau mise-en-œuvre complexe

CUDA vs OpenCL (constats)

- Chez Nvidia
  - le driver CUDA est plus performant que le driver OpenCL
  - le profiler fonctionne mieux sur CUDA que sur OpenCL
  - Certains types de données dans les objets de type Image sont mal supportés
- Chez ATI
  - Problèmes d'installation de driver chez ATI récurrents (essentiellement sous linux)
  - Il est très difficile de faire fonctionner plusieurs types de devices en même temps (GPU ATI + Nvidia)

Carré d'un vecteur en OpenCL (1/3)

```
#include <iostream>
using namespace std;

#include <CL/cl.h>
#define DATA_SIZE (1024*1240)

const char *KernelSource =  "\n" \
"__kernel void square( __global float* input, __global float*
      output,           \n" \
"  const unsigned int count)          \n" \
"{                                    \n" \
"   int i = get_global_id(0);         \n" \
"   if(i < count)                     \n" \
"     output[i] = input[i] * input[i]; \n" \
"}                                    \n" \
"\n";
int main(int argc, char* argv[])
{
 int devType=CL_DEVICE_TYPE_GPU;
 cl_int err;
 size_t global;
 size_t local;
 cl_platform_id cpPlatform;
 cl_device_id device_id;
```

```
// ...
cl_context context;
cl_command_queue commands;
cl_program program;
cl_kernel kernel;

//get platform IDs
err = clGetPlatformIDs(1,  &cpPlatform, NULL);

// get devices IDs
err = clGetDeviceIDs( cpPlatform, devType, 1, &device_id, NULL);

// create platform context
context = clCreateContext(0, 1, &device_id, NULL, NULL, &err);

// create command queue
commands = clCreateCommandQueue(context, device_id, 0, &err);

// create program object
program = clCreateProgramWithSource(context,1, (const char **) &KernelSource,
                                    NULL, &err);

// Build the program executable
err = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
commands = clCreateCommandQueue(context, device_id, 0, &err);
```


Carré d'un vecteur en OpenCL (2/3)

```
// ...
 // Create the compute kernel in the program
 kernel = clCreateKernel(program, "square", &err);
 if (!kernel || err != CL_SUCCESS) {
  cerr << "Error: Failed to create compute kernel!" << endl;
  exit(1);
 }

 // create data for the run
 float* data = new float[DATA_SIZE];
 float* results = new float[DATA_SIZE];
 unsigned int correct;
 cl_mem input;
 cl_mem output;

 // Fill the vector with random float values
  unsigned int count = DATA_SIZE;
 for(int i = 0; i < count; i++)
  data[i] = rand() / (float)RAND_MAX;

 // Create the device memory vectors
 input = clCreateBuffer(context,  CL_MEM_READ_ONLY,
                                  sizeof(float) * count, NULL, NULL);
 output = clCreateBuffer(context, CL_MEM_WRITE_ONLY,
                                  sizeof(float) * count, NULL, NULL);
```

```
// Transfer the input vector into device memory
 err = clEnqueueWriteBuffer(commands, input, CL_TRUE, 0,
 sizeof(float) * count, data, 0, NULL, NULL);

 // Set the arguments to the compute kernel
 err = 0;
 err = clSetKernelArg(kernel, 0, sizeof(cl_mem), &input);
 err |= clSetKernelArg(kernel, 1, sizeof(cl_mem), &output);
 err |= clSetKernelArg(kernel, 2, sizeof(unsigned int), &count);

 // Get the maximum work group size for executing the kernel on the
 device
 err= clGetKernelWorkGroupInfo(kernel, device_id,
 CL_KERNEL_WORK_GROUP_SIZE, sizeof(local), &local, NULL);

 // Execute the kernel over the vector using the
 // maximum number of work group items for this device
  global= count;
 err = clEnqueueNDRangeKernel(commands, kernel, 1, NULL,
 &global, &local, 0, NULL, NULL);
 // Wait for all commands to complete
 clFinish(commands);

 // Read back the results from the device to verify the output
 err= clEnqueueReadBuffer( commands, output, CL_TRUE, 0,  sizeof
 (float) * count, results, 0, NULL, NULL );
```


Carré d'un vecteur en OpenCL (3/3)

```
// Wait for all commands to complete
 clFinish(commands);

// Read back the results from the device to verify the output
err = clEnqueueReadBuffer( commands, output, CL_TRUE, 0,
sizeof(float) * count, results, 0, NULL, NULL );


// Shutdown and cleanup
delete [] data;
delete [] results;


// Release ressources
clReleaseMemObject(input);
clReleaseMemObject(output);
clReleaseProgram(program);
clReleaseKernel(kernel);
clReleaseCommandQueue(commands);
clReleaseContext(context);

return 0;
}
```


Carré d'un vecteur en CUDA (4/4)

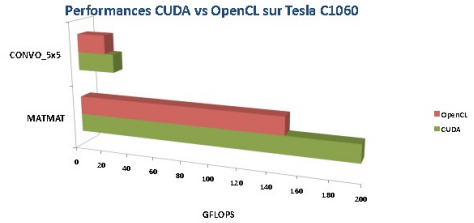
```
#include <stdio.h>
#include <assert.h>

#define N_BLOCKS 16
#define N_THREADS_PER_BLOCK 256

// square computation kernel
__global__ void myFirstKernel(float *d_input, float *d_output, int
      count)
{
  int idx= blockIdx.x*blockDim.x + threadIdx.x;
  d_output[idx] = d_input[idx]*d_input[idx];
}

int main( int argc, char** argv)
{
 // pointer for host memory
 int *h_a;

 // pointer for device memory
 int *d_a;

 // define grid and block size
 int numBlocks = N_BLOCKS;
 int numThreadsPerBlock = N_THREADS_PER_BLOCK;

// ...
```

```
// ...

//allocate host and device memory
size_t memSize = numBlocks * numThreadsPerBlock * sizeof(float);
h_a = (float *) malloc(memSize);
cudaMalloc( (void **) &d_a, memSize );

//launch kernel
dim3 dimGrid(numBlocks);
dim3 dimBlock(numThreadsPerBlock);
myFirstKernel<<<dimGrid, dimBlock>>>(d_a);

// block until the device has completed
cudaThreadSynchronize();

// device to host copy
cudaMemcpy( h_a, d_a, memSize, cudaMemcpyDeviceToHost );

// free device memory
cudaFree(d_a);

// free host memory
free(h_a);

 return 0;
}
```


CUDA vs OpenCL

- Le kernel est quasi-similaire entre CUDA et OpenCL.
- La grande différence réside dans la partie API de runtime
  - On paye le prix de la flexibilité et de la portabilité par un code plus verbeux
- Attention! : une grande partie de la verbosité d'OpenCL peut être atténuée en passant par des API plus haut-niveau
  - Bindings C++
  - stdcl (Standard Compute Layer Interface)

Performances



- Pour un même kernel dans les deux cas
- On observe une différence de performances de 30% environs, à la faveur de CUDA sur les architectures Nvidia

Outils et librairies

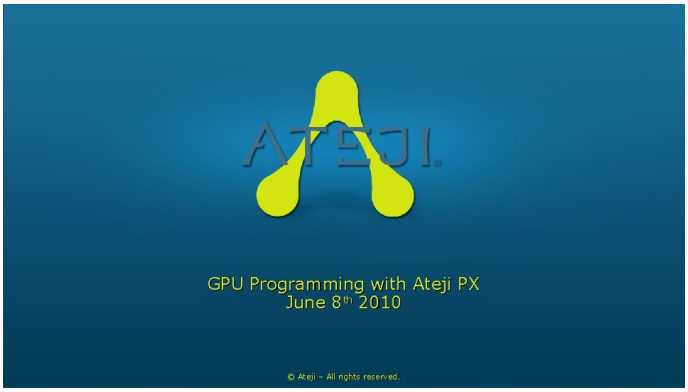
- Par4All
  - Pas de génération de code OpenCL (prévue dans la version 1.3)
  - Pas d'exploitation de la shared memory en CUDA
- HMPP
  - Supporte la génération de code OpenCL uniquement sur GPU (ATI & Nvidia)
  - L'utilisation d'optimisation avancées (shared memory) nécessite une modification du code original (rupture du code legacy)

Futur d'OpenCL

- Création d'un groupe de travail Khronos : WebCL™
  - À la suite de la publication des spécifications WebGL 1.0
  - Ouvrir une porte pour le calcul parallèle sur CPU/GPU à partir d'un navigateur web en HTML 5
- Nvidia remplacera-t-elle CUDA par OpenCL?
  - Une large communauté de développeurs CUDA
  - Une API de runtime haut-niveau beaucoup plus simple à utiliser
  - A l'évidence la réponse est NON!

2.5 Philippe Coucaud (Ateji)

Java sur GPU avec Ateji PX



Goals

- "Write once, run everywhere", even on a GPU
- Target heterogeneous architectures from Java
  - GPU accelerators
  - OpenCL standard
- Get a reasonable out-of-the-box performance improvement on mainstream devices
- Provide enhanced constructs allowing to get an extra speed-up for HPC workloads

The Road to GPU

- The Ateji PX language already provides constructs allowing to express many parallel patterns …
- … including the kind of data-parallel programs that fit well on a GPU
  - Parallel branch ↔ OpenCL Kernel[1]
  - Branch quantification ↔ N Dimensional Range[2]
  - Branch scopes ↔ Private, local, global memories[3]

```
final int K = 10;
[ ||(#OpenCL())
   final int[] result = new int[N];
   [ ||( int i: result.length , #Kernel("C2070") )
      int x = i * i + K; result[i] = x;
   ]
   display(result);
]
```

One-dimensional kernel executed N times on C2070 GPU

1. Thread 2. Grid 3. Local, shared, global

Architecture Overview

- Compiler produces Java and OpenCL source files
- Annotations allow user to explicitly select a particular OpenCL device and pass parameters to the compiler
  - `#Kernel("AMD-9350","-cl-mad-enable")`
  - `#Kernel()`: first GPU, default compiler settings
  - Multi-GPU configurations supported
- Runtime uses OpenCL driver to:
  - Compile programs and cache binaries
  - Transfer memory
  - Set kernel parameters
  - Launch kernels

Supported Java Subset

- Functions
  - Local & static, no recursion, no overloading
  - Cloning to support polymorphism on memory space qualifiers
  - +, -, *, /, %, …
  - `java.lang.Math.*`
  - `System.out.format()` (OpenCL extension)
- Types
  - Primitive types
  - 1-D arrays of primitive types
- Standard control-flow statements
  - No labels (not supported by some OpenCL compilers)

OpenCL Platforms

- Tested platforms
  - NVIDIA CUDA (GPU)
  - AMD Accelerated Parallel Processing (CPU & GPU)
  - Intel OpenCL (CPU)
- Using OpenCL to target multi-core is a realistic path
  - Generated OpenCL code exhibits more vectorization opportunities than plain Java code
  - Intel® compiler based on the Threading Building Block library could open the door to many-core architectures
- Diversity of OpenCL devices forces to write (performance) portable code

N Dimensional Range (1/2)

- Defines the index space
  - 1 work-item[1] per point of the index space
  - Optionally organized into explicit work-groups[2] (coarse-grained decomposition)
- Main parameter of `clEnqueueNDRangeKernel()`
- Expressed using existing branch quantifiers of the Ateji PX language
  - `|| … ;` Unique instance
  - `|| (int I : N) … ;` N instances

1. thread 2. blocks

N Dimensional Range (2/2)

- 2-D index space
  - `|| (int x: X, int y: Y, #Kernel())`
  - NDRange = [**X,Y**] / [-,-]
    - **X*Y** work-items
    - Implicit work-groups (not always optimal)
- 1-D index space with explicit work-groups
  - `|| (int n: N, #Kernel()) // N work-groups`
    `|| (int m: M) // M work-items per work-group`
  - NDRange = [**N*M**] / [**M**]
- Generated code automatically references `get_global_id(x)` and `get_local_id(x)` primitives[1]

1. blockIdx.x * blockDim.x + threadIdx.x and threadIdx.x

Memory Hierarchy

- Mapped on Java scopes, no extra qualifier required
  - Local and private variables are defined in nested parallel blocks

```
[ || (int n: N, #Kernel())
    // local variable shared by all work-items
    // pertaining to the same work-group
    int[] t = new int[N];
    [ || (int m: M)
        // private variables
        int x = A[N*n + m];
    ]
]
```

  - Other variables allocated in global address space
  - Constant data generated as globally scoped OpenCL constants
  - Images not yet supported

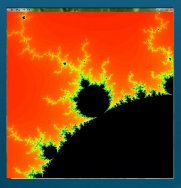
Buffer Management

- Automatic
  - Compiler automatically
    - Creates memory buffers with appropriate flags (`READ_ONLY`, `WRITE_ONLY`, `READ_WRITE`)
    - Releases buffers
    - Enqueues read/write commands
  - Transparent to the user, but transfers may not start at optimal locations
- Manual
  - User has to manually create channels to/from the OpenCL device to explicitly transfer memory
  - Memory transfers can be optimized

Performance

- Mandelbrot set (zoom in/out, 128 frames)
  - Massively parallel
  - No manual optimization
  - Small memory buffers (1024x1024 pixels + color palette)

Execution	FPS	Speed-up	
		Serial	Multi-core
Serial	~2		
Multi-core (Intel i5, 2 cores HT)	~7	3,5	
GPU (NVIDIA GT330M)	~39	19,5	5,5



Issues & Limitations

- Language
  - No direct access (yet) to OpenCL vector types that some architectures could benefit from
- NVIDIA
  - Public drivers still only supporting OpenCL 1.0
  - New CUDA 4.0 features not (yet) available in OpenCL
- Non-standard platform extensions
  - `cl_khr_fp64` vs `cl_amd_fp64`
  - `cl_amd_printf` vs `cl_intel_printf`
  - All OpenCL implementations are slowly converging
- Profilers not Java-aware
  - Cumbersome configuration
  - No integration into Java IDEs

Ongoing work

- OpenGPU
  - RNA-folding
  - Intrinsically parallel: window sliding over an RNA sequence ...
  - ... but current implementation heavily relies on a shared state to minimize number of candidates
- European Space Agency
  - Gaia project: characterizing the photometric and spectral variability of stars
  - Intrinsically parallel: light sources are independent
  - Current bottleneck is memory (RAM & throughput)

Questions

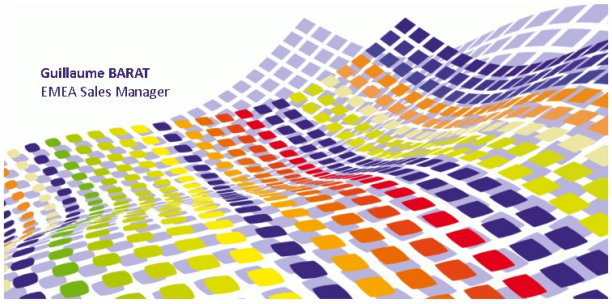
2.6 Guillaume Barat (CAPS)

Migration *manycore* : méthodologie et outils

CAPS

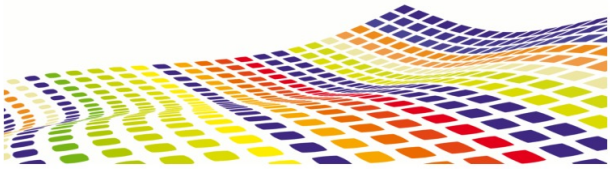
Cost effective migration to manycore

Methodology and tools



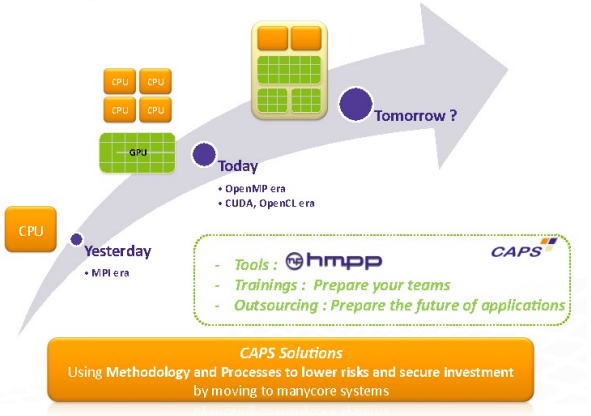
CAPS

CAPS overview



Be ready for the manycore

CAPS



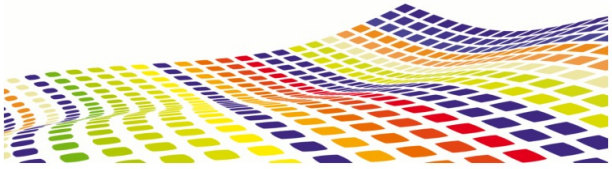
CAPS

CAPS

- Worldwide company
  Americas, APAC, EMEA
- History
  - 10 years
- Turnover:
  - 2009 **1.6** M$
  - 2010 **3.0** M$
  - 2011 4.5 M$ expected
  - Profitable

CAPS

Cost effective migration process

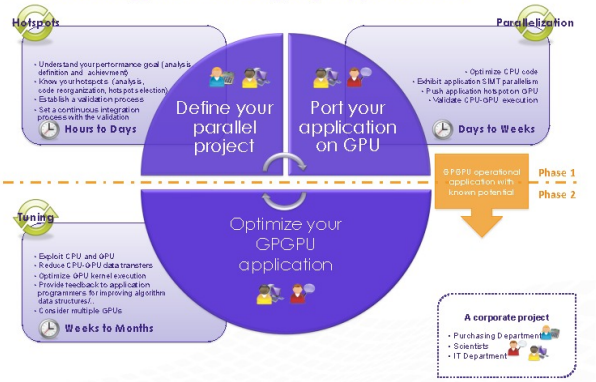


Objectives & constraints

CAPS

- Improving performance *at reasonable cost*
  - CPU (+ GPU)
- Keeping the code alive for the application developers
  - Migrated code has to remain understandable
  - And easy to maintain
- (Portability)
  - Ready for future manycore architecture

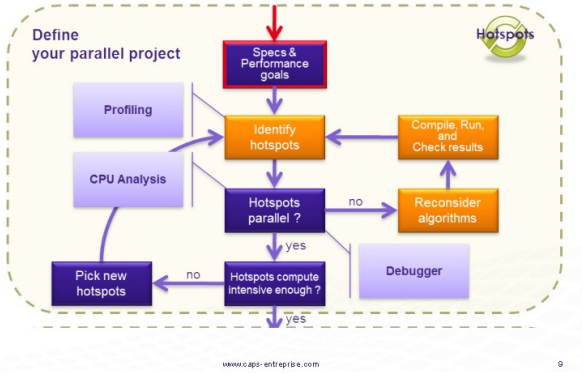
Methodology to Port Legacy Applications



Define your parallel project *(Phase 1a)*

- Understand your performance goal
- Ensure you know how to validate the code results
  - Rounding may differ on GPUs
- Determine if your goal can be achieved
  - Application profiling, analysis and assesment
- Define an incremental approach

Phase 1a (details)



What if not enough parallelism can be found ?

Algorithm is parallel but implemtation serializes it	• Some rewriting is required to recover the parallelism
Computing method is intresically serial	• Fine grain (SIMD : instruction level parallelism • Large grain (SPMD : message passing parallelism)
Method is parallel but not enough individual computing tasks exhibited	• Running multiple hotspots in parallel • Usually requires heavy rewriting of the code

Go / No Go Analysis

Go

- **Dense hotspot** (many computation per data)
- **Fast GPU kernels**
- **Low CPU-GPU data transfers**
- **Midterm perspective** (Ready for manycore)

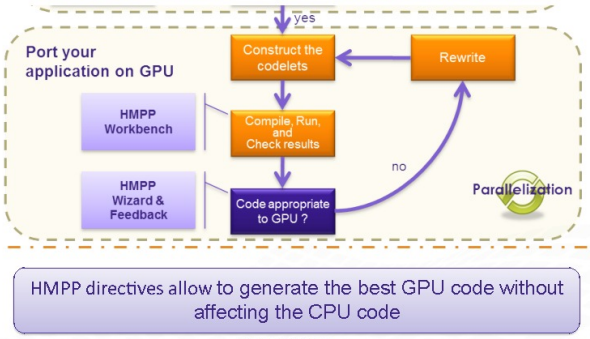
No Go

- **Binary exact CPU-GPU results constraints**
- **Slow GPU kernels** (transfers won't be compensated
- **Flat execution profile** (expensive & risky)
- **Complex data structures**
- **Big memory space requirement**

Port your application on GPU *(Phase 1b)*

- Get an hybrid GPGPU program
  - Convert hotspots in GPU kernels
  - Make your code more GPU friendly
    - *Get accurate GPU profil and traces of your application*
- Identify GPGPU issues
- Validate the parallel implementation
- Last Go / NoGo decision

Phase 1b (details)



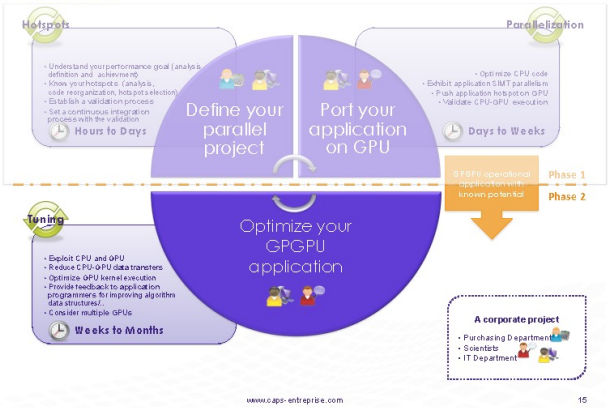
HMPP directives allow to generate the best GPU code without affecting the CPU code

Make your kernels more GPU friendly

HMPP Wizard

- Interactive diagnosis tool to step-by-step improve the performance of GPU generated code.
- Main features
  - Generate performance advices based on static analysis
  - Detects standard kernels and provide specific advices
  - Analyze memory access patterns
  - Use HMPP Directives and make your kernel GPU friendly

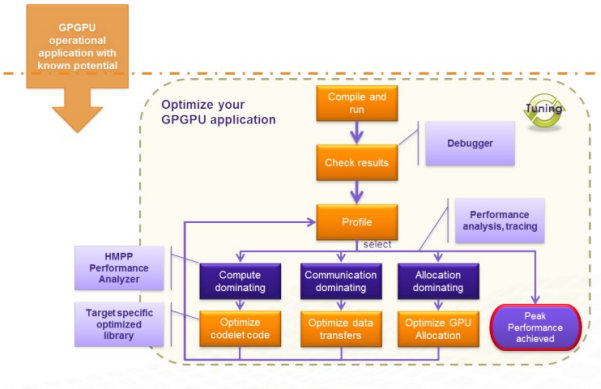
Phase 2



Optimize your GPGPU application *(Phase 2)*

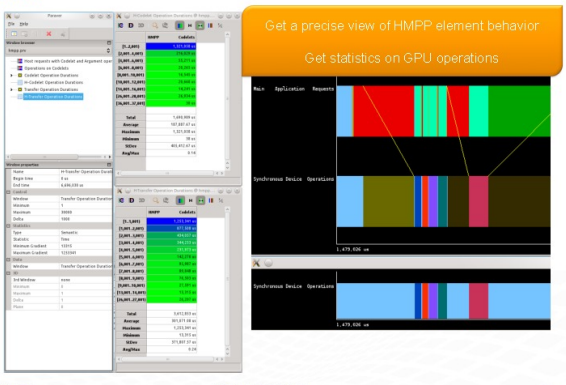
- Analyzing GPU performance
  - Mesure the CPU-GPU transfers time and kernels execution time
  - Understand how well kernels are performing
- Optimization
  - Allocation optimization
  - Data transfer optimization
  - GPU kernels optimization

Phase 2 (details)



Mesure the transfers and execution time

Paraver



Understand how well kernels are performing

HMPP Performance Analyzer

- Provide fine metrics at source code level
- Main features
  - Analyze and profile kernel execution on the GPU
    - Get precise and specific information about the kernel behavior
    - Explore and exploit at best the GPU power from the source level
    - Memory throughput, grid size, computation density…
  - Fine tune of kernels
    - Profil linked to the hardware

Optimizations

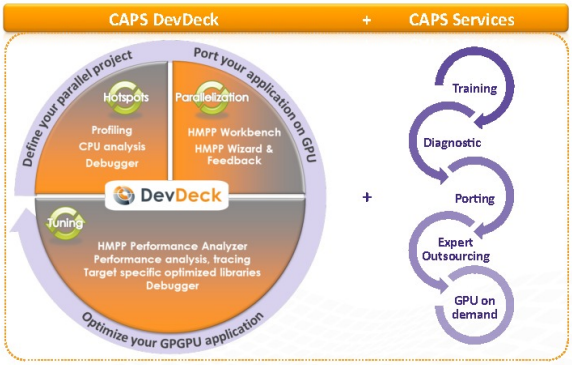
- Reducing data transfers
  - Leaving data on the GPU to exploit reuses
  - Sharing data on the GPU between codelets
  - Partial data transfer
  - Moving CPU serial code to GPU
- Optimizing GPU kernels
  - Tuned memory accesses to exploit fast GPU pipeline memory
  - Tuned resources usages to maximise the use of all HW resources in parallel

HMPP provides a set of directives to address these optimizations

Cost-effective migration requirements

Go / NoGo Analysis	• Application Diagnosis by CAPS
Incremental approach	• Provides by HMPP
Don't redevelop what already exists	• Import GPU libraries/function into HMPP
Understand finely the data transfers	• Use tools like paraver, Vampire, TAU
Good understanding of the HW	• HMPP Performance Analyser
Lots of experimentation at a lower cost	• HMPPCG directives

CAPS Solution and Services to Port Your Applications



Examples of porting applications

Solar Energy

Monte Carlo simulation for thermal radiation

- Resource spent
  - 1,5 man-month
- GPU C2050 improvement
  - x6 over 8 Nehalem cores
  - **x23 with 8 nodes compare to the 8 core machine**

Oil & Gas

Seismic depth imaging

- 1 GPU solution = 4.4 CPU solution
  - GPU: 16 x [8 cores + 2 GPU]
  - CPU: 64 x [8 cores]

Weather Forecasting

Global cloud resolving model

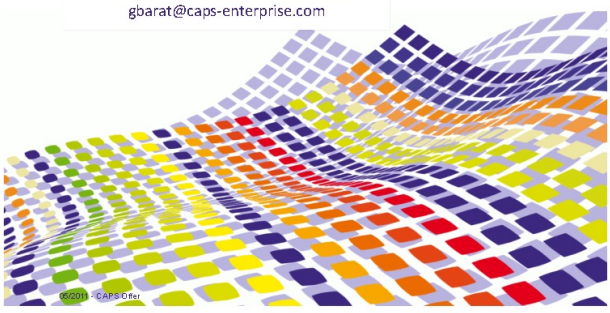
- Resource spent
  - 1 man-month (customer work)
- GPU C1060 improvement
  - 11x over serial code on Nehalem

Computer Vision & Medical imaging

MultiView Stereo

- Resource spent
  - 1 man-month
- CPU Improvement
  - **x 4,86**
- GPU C2050 improvement
  - **x 120** over serial code on Nehalem

http://www.caps-entreprise.com
http://twitter.com/CAPSentreprise
gbarat@caps-enterprise.com



2.7 Basile Starynkevitch (CEA)

OpenGPU et GCC–MELT

OpenGPU et GCC-MELT

Basile STARYNKEVITCH
basile@starynkevitch.net (ou basile.starynkevitch@cea.fr)

energie atomique • energies alternatives

8 juin 2011 - colloque OpenGPU

Merci à la DGCIS qui finance ce travail

GCC est partout ☺

www.gcc.fr (plateau de Saclay)

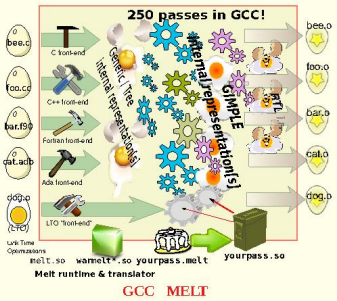
Mais **GCC** (Gnu Compiler Collection) est un **compilateur libre** (GPLv3) pour C, C++, Go, Ada, Fortran → **gcc.gnu.org**

Tout le monde utilise du code compilé par Gcc [1]

Question : **Qui utilise gcc–4.6** [paru fin mars 2011] [2] **?**

1. Dans un supercalculateur, une voiture, un serveur web Linux, ou un téléphone !
2. Il vous faudrait compiler Gcc depuis son code source.

GCC est complexe [3], extensible grâce à MELT



MELT
http://**gcc-melt.org**/ sert à **étendre** "facilement" Gcc pour **vos propres besoins spécifiques**

par exemple : *Google Summer Of Code 2011* (Pierre Vittet)
Customizable warnings with a GCC plugin

Rêvez-vous d'un "Coverity™-like" **libre**, bâti sur Gcc & Melt ?

3. 5MLOC, +7%/an, 400 développeurs [généralement professionnels à temps plein]

GCC-MELT et OpenCL

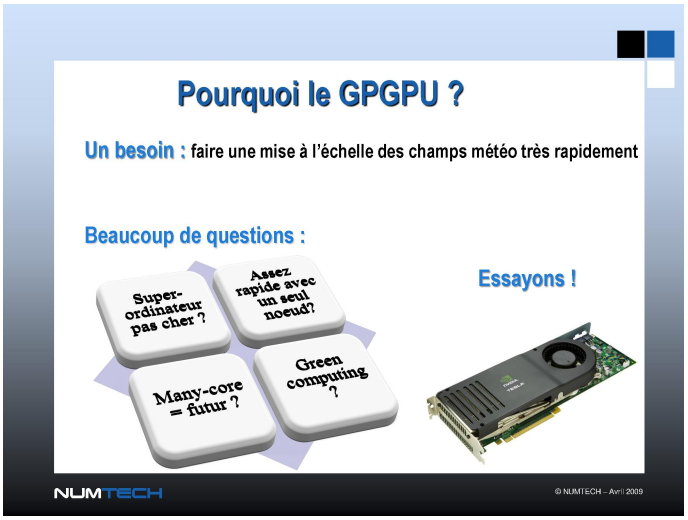
OpenCL™ est un langage source, standard industriel, pour GPU.
Il y a des variantes **experimentales / académiques** de Gcc générant un peu d'OpenCL.
Un générateur d'OpenCL est en cours de développement dans Melt :

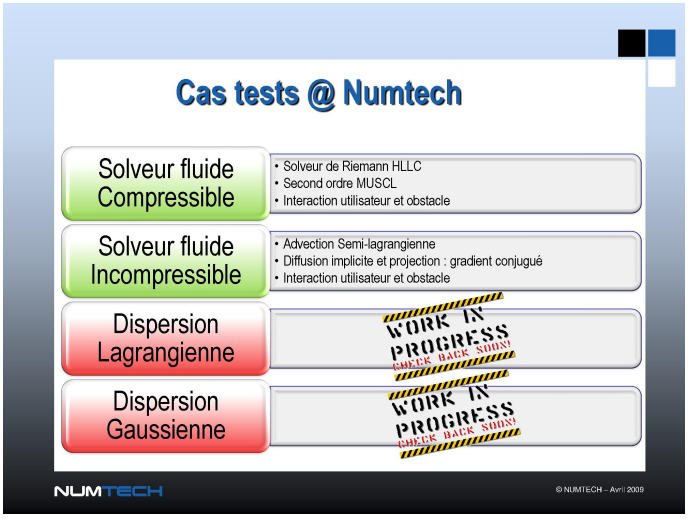
- ce n'est pas un compilateur OpenCL
  (on utilise par exemple celui d'AMD Stream)
- c'est **complexe**, car on doit se restreindre à l'interface disponible dans gcc-4.6 ; il faut aussi **étendre Melt** lui-même
- bibliothèque **glue** vers OpenCL **disponible**
- il s'agit de convertir une représentation interne Gimple → OpenCL
- il faut éviter de tout convertir [4] (donc il faut détecter les boucles, les tailles de tableau, les nombres d'itérations . . .)
- on s'appuie sur Graphite (INRIA & AMD) [optimiseur dans Gcc]
- la génération n'est pas complète en juin 2011

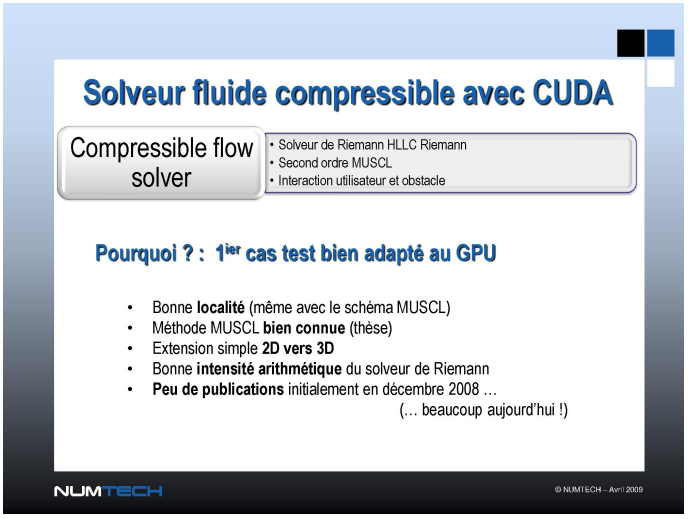
4. Le transfert de données CPU ↔ GPU est coûteux et ne vaut la peine que pour les gros tableaux

2.8 Vivien Clauzon (Numtech)

Retours d'expérience d'une PME sur le GPGPU pour des applications de dispersion atmosphérique







Solveur fluide compressible avec CUDA

Quoi ? : Equations d'Euler

L'inconnue est (sous forme conservative) $U = {}^t(\rho, \rho u, \rho v, \rho w, E)$

$$\frac{\partial \boldsymbol{U}}{\partial t} + \frac{\partial \boldsymbol{F}(\boldsymbol{U})}{\partial x} + \frac{\partial \boldsymbol{G}(\boldsymbol{U})}{\partial y} + \frac{\partial \boldsymbol{H}(\boldsymbol{U})}{\partial z} = 0.$$

Avec les flux

$$\boldsymbol{F}(\boldsymbol{U}) = \begin{pmatrix} \rho u \\ \rho u^2 + P \\ \rho uw \\ \rho uw \\ u(E+P) \end{pmatrix}, \quad \boldsymbol{G}(\boldsymbol{U}) = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + P \\ \rho vw \\ v(E+P) \end{pmatrix},$$

$$\boldsymbol{H}(\boldsymbol{U}) = \begin{pmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + P \\ w(E+P) \end{pmatrix}.$$

NUMTECH © NUMTECH – Avril 2009

Solveur fluide compressible avec CUDA

Comment ? :

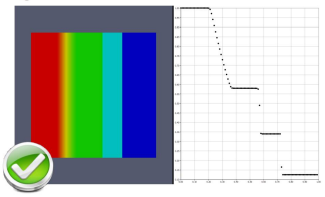
- Schéma en temps d'Euler explicite ⇒ *simple, rapide, faible coût memoire*
- Méthode MUSCL ⇒ *précision et localité (Shared Memory)*
- Limiteur *Minmod* ⇒ *stable et rapide*
- Solveur de Riemann
  - *Bonne intensité : 89 flops pour 12 reads et 6 writes*
  - *Mais … 22 registres utilisés*
  - *33% occupancy*
  - *Bon load-balancing*
  - *Robuste*

NUMTECH © NUMTECH – Avril 2009

Solveur fluide compressible avec CUDA

Validation

Cas test pseudo 1D : **shock tube**
Toro, E. F. (1999), Riemann Solvers and Numerical Methods for Fluid Dynamics.



Cas test 2D : **forward facing step**
Woodward & Colella (1984), The Numerical Simulation of Two-Dimensional Fluid Flow with Strong Shocks.

NUMTECH © NUMTECH – Avril 2009

Solveur fluide compressible avec CUDA

Performances

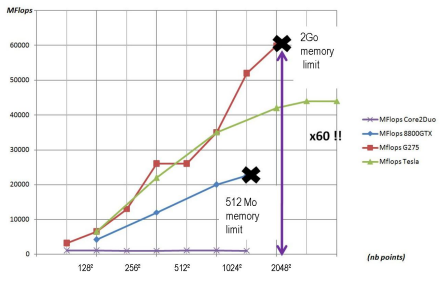
Maillage	temps sur GPU	temps sur CPU	speedup
256^2	1 minute	15 minutes	15
512^2	4 minutes	2 heures	30
1024^2	25 minutes	16 heures	28.5
2048×1024	1 heure	44 heures	44

Comparaison entre les temps de retour pour un test standard entre la version CPU séquentielle (Corei7 920) et la version GPU (Tesla S1060) en fonction du nombre de point de maillage.

NUMTECH © NUMTECH – Avril 2009

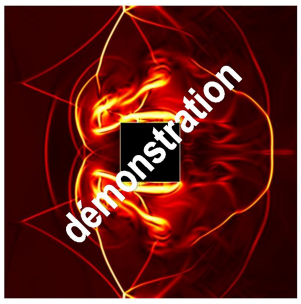
Solveur fluide compressible avec CUDA

Performances



NUMTECH © NUMTECH – Avril 2009

Solveur fluide compressible avec CUDA



NUMTECH © NUMTECH – Avril 2009

Solveur fluide incompressible avec CUDA

Incompressible flow solver

- Advection semi-lagrangienne
- Diffusion implicite et projection : gradient conjugué
- Interaction utilisateur et obstacle

Pourquoi ? : besoin de vitesse

- Champs météo petite échelle requis pour calcul de dispersion locale
- Besoin d'être rapide pour répondre en cas de crise
- Géométries complexes : site urbain et topographie
- Hardware "mobile" : meilleure réactivité

Solveur fluide incompressible avec CUDA

Quoi ? : Equation de Navier Stokes incompressible

$$\begin{cases} \rho\dfrac{\partial\vec{u}}{\partial t} + \underbrace{\rho(\vec{u}.\nabla)\vec{u}}_{\text{advection}} + \underbrace{\nabla P}_{\text{pression}} - \underbrace{\mu\Delta\vec{u}}_{\text{diffusion}} + \underbrace{\vec{F}}_{\text{forces}} = 0 \\ \nabla.\vec{u} = 0 \\ \dfrac{\partial\theta}{\partial t} = -(\vec{u}.\nabla)\theta + \mu_\theta\Delta\theta \end{cases}$$

$$\vec{F} = \underbrace{C_\theta\theta\vec{y}}_{\text{buoy}} + \underbrace{\frac{\vec{u}-\vec{u}_{ref}}{C_{obs}}\Omega_{obs}}_{\text{obstacle}} - \underbrace{g\vec{y}}_{\text{gravité}}$$

Solveur fluide incompressible avec CUDA

Comment ? :

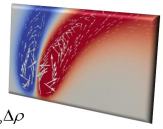
- Advection semi-lagrangienne $q(\vec{x},t+\delta t) = q(\vec{x}-\vec{u}(\vec{x},t)\delta t,t)$
- Interpolation bi/tri-cubic
- Diffusion implicite $(I-\nu\delta t\Delta)\vec{u}(\vec{x},t+\delta t) = \vec{u}(\vec{x},t)$
- Méthode de projection (Gradient *Conjugate* pour l'équation de Poisson)

$$\vec{w}^{n+1} = (A+D+F)(\vec{u}^n)$$
$$\Delta P^{n+1} = \nabla.\vec{w}^{n+1}$$
$$\vec{u}^{n+1} = \vec{w}^{n+1} - \nabla P^{n+1}$$

Solveur fluide incompressible avec CUDA

Add-ons

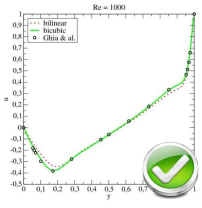
- Schéma de MacCormack
  $1°)\ \tilde{\Psi}^{n+1} = A(\Psi^n)$ ← Standard SL
  $2°)\ \tilde{\Psi}^n = A^R(\tilde{\Psi}^{n+1})$ ← Backward SL
  $3°)\ \Psi^{n+1} = \tilde{\Psi}^{n+1} + \frac{(\Psi^n - \tilde{\Psi}^n)}{2}$ ← Error correction
- Confinement de vorticité
  $N = \frac{\eta}{|\eta|},\ (\eta = \nabla|\omega|)$
  $F_{conf} = \varepsilon\, N \times \omega$



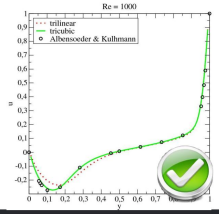
- Densité variable (Guermond) $\frac{\partial\rho}{\partial t} = -(\vec{u}.\nabla)\rho + \mu_\rho\Delta\rho$
- Taux de réaction (effet de flamme) $\frac{\partial Y}{\partial t} = -(\vec{u}.\nabla)Y + \mu_Y\Delta Y - \underbrace{kY}_{\text{reaction}}$

Solveur fluide incompressible avec CUDA

Validation



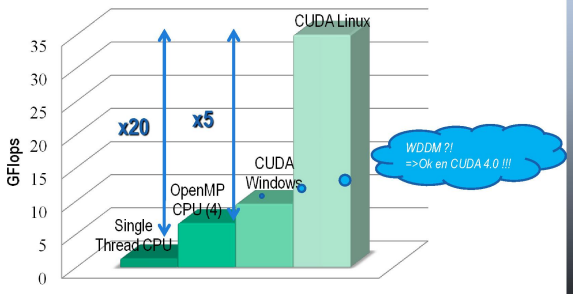
Cas test 2D : **lid-driven cavity (100<Re<10 000)**
Ghia, Ghia, and Shin (1982), High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method.



Cas test 3D : **lid-driven cavity (Re=1000)**
Albensoeder & Kuhlmann (2005) Accurate three-dimensional lid-driven cavity flow

Solveur fluide incompressible avec CUDA

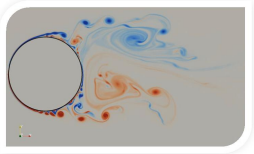
Performances (Tesla C1060)

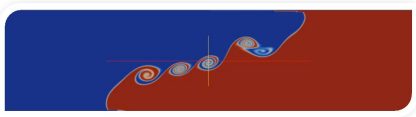


Solveur fluide incompressible avec CUDA

Et maintenant ?

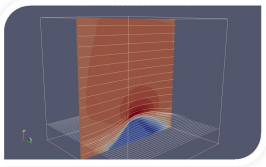
- Validation densité variable et obstacle
- Conditions aux limites + complexes
- 1st Release Q4-2011



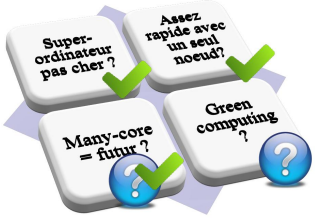


Et maintenant ?

- Validation densité variable et obstacle
- Conditions aux limites + complexes
- 1st Release Q4-2011

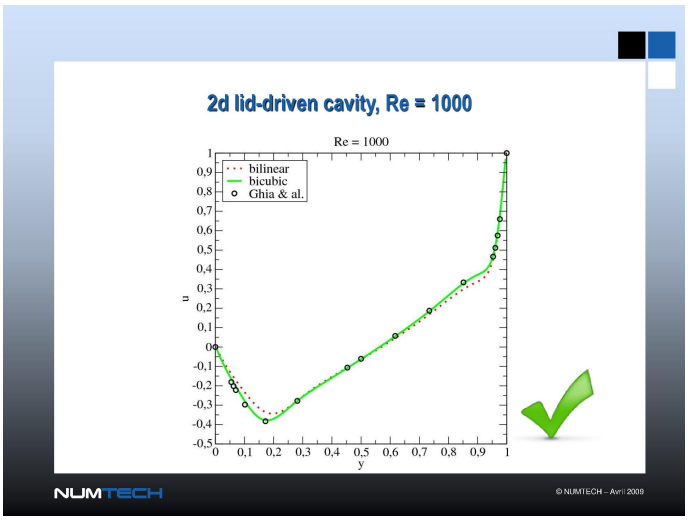


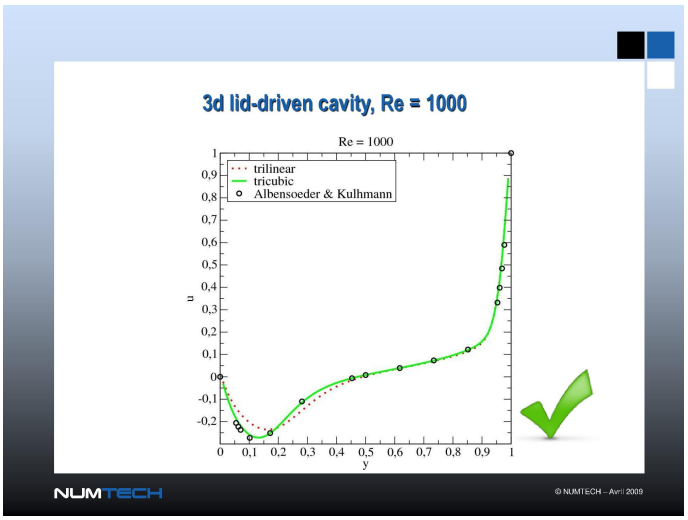
Et maintenant ?

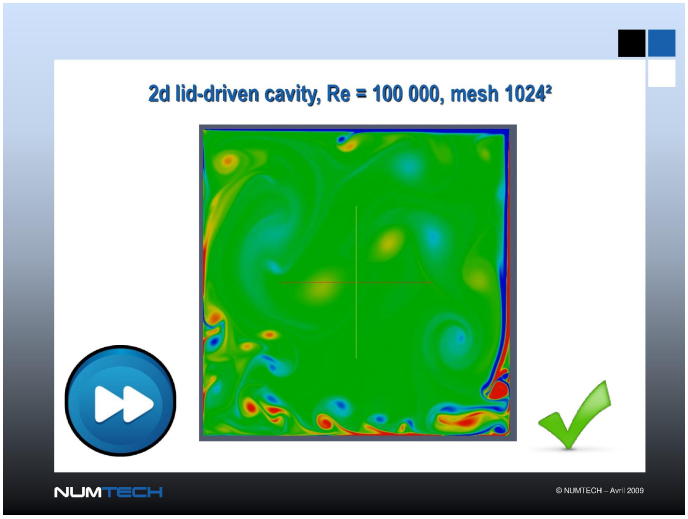


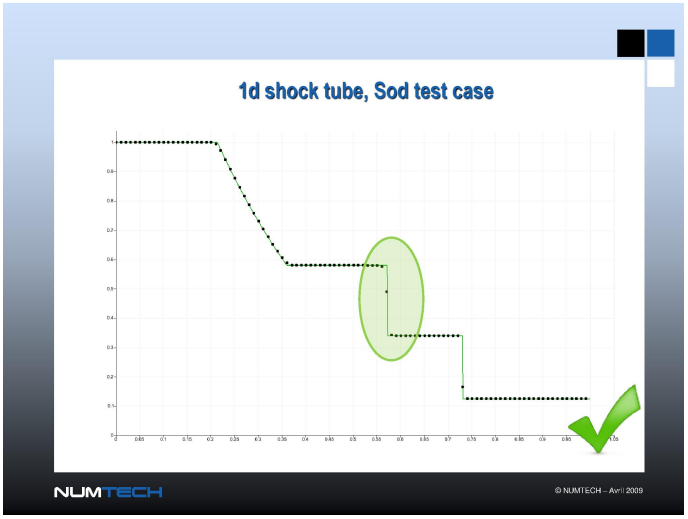
Merci!

Contact :
clauzon@numtech.fr


2d lid-driven cavity, Re = 1000
Re = 1000
bilinear
bicubic
Ghia & al.
u
y
NUMTECH
© NUMTECH – Avril 2009


3d lid-driven cavity, Re = 1000
Re = 1000
trilinear
tricubic
Albensoeder & Kulhmann
u
y
NUMTECH
© NUMTECH – Avril 2009


2d lid-driven cavity, Re = 100 000, mesh 1024²
NUMTECH
© NUMTECH – Avril 2009


1d shock tube, Sod test case
NUMTECH
© NUMTECH – Avril 2009

2.9 Jean-Pierre Panziera (Bull)

Vision de Bull sur les architectures hybrides

La [r]évolution des technologies et des architectures

Jean-Pierre Panziera

Bull, 2011

Applications HPC

Electro-Magnetics | Computational Chemistry Quantum Mechanics | Computational Chemistry Molecular Dynamics | Computational Biology | Structural Mechanics Implicit

Structural Mechanics Explicit | Seismic Processing

Une grande variété de logiciels et de domaines d'applications

Computational Fluid Dynamics | Reservoir Simulation | Rendering / Ray Tracing | Climate / Weather Ocean Simulation | Data Analytics

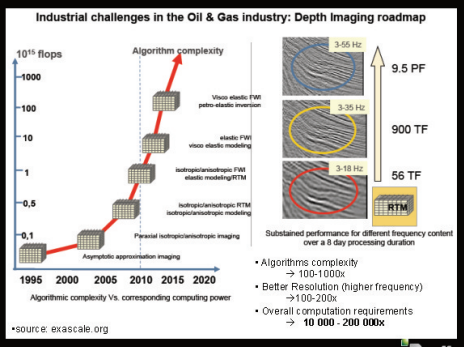
2 ©Bull, 2011 OpenHPU- jpp

[r]évolution des technologies et des architectures

- Des besoins de calcul toujours plus importants
- Du Pétaflop à l'Exaflop en moins de 10 ans
  1 ExaFlops = 10^18 opérations flottantes / seconde
- Evolution des Processeurs
- Accélérateurs de Calcul
- Evolution/Convergence Processeurs & Accélérateurs

3 ©Bull, 2011 OpenHPU- jpp

Increasing demand for HPC performance



4 ©Bull, 2011 OpenHPU- jpp

Du Petascale à l'Exascale x1000 en <10 ans

Extrapolation d'un système Petascale d'aujourd'hui à l'Exascale …

	2010:	≤2020:	
Flops	1 PFlops	1 EFlops	1,000 x
Noeuds	4,000	>128,000	32 x
Coeurs	>100,000	>100,000,000	1,000 x
Capacité Mémoire	300 TB	150 PB	500 x
Bande Passante Mem	>500 TB/s	> 250 PB/s	500 x
Capacité Stockage	20 PB	10 EB	500 x
BW Interconnect	40 Gb/s	8 Tb/s	200 x
BW Stockage	500 GB/s	100 TB/s	200 x
Conso. Elec.	5 MW	100 MW	20 x

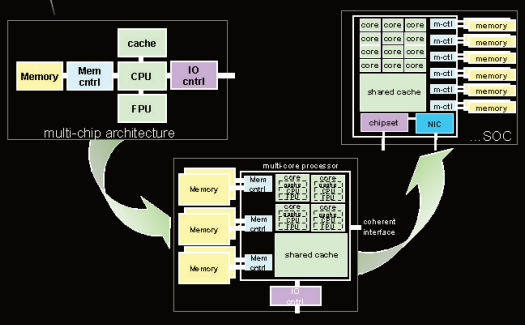
5 ©Bull, 2011 OpenHPU- jpp

Défis Technologiques de l'Exascale

- Conception Processeur : architecture et fréquence
  Multi/Many-cores, Accélérateurs, …
- Capacité Memoire & BW → MCM, 3D Packaging ?
  Alimenter les moteurs de calculs (Flops) en données (Bytes)
- BW Réseau, Latence, Topologie et Routage
  connections/câbles optique, moins de hops, packaging compact
- Scalabilité E/S et Flexibilité
  Données XXXLarge & calculs + rapides → explosion données
- Résilience et fiabilité du Système tout entier
  Calculs durant semaine(s) malgré les défauts HW
- Consommation et Refroidissement
  - Composants moins nombreux, faible consommation, amélioration de l'efficacité énergétiques (PUE)
- Prix ?

6 ©Bull, 2011 OpenHPU- jpp

Evolution de l'architecture des processeurs



Stagnation des performance des Processeurs

- Toujours plus de transistors
- Fréquence des Processeurs stoppée à 2-5GHz
- Consommation des processeurs plafonnée à ~100W
- Efficacité des Processeurs augmente peu (Instruction Level Parallelism, ILP)

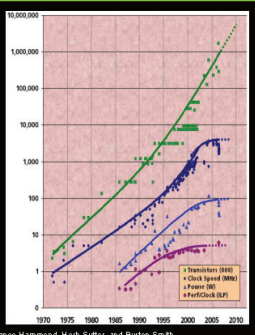
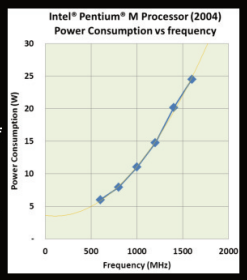


Figure credit:
Kunle Olukotun, Lance Hammond, Herb Sutter, and Burton Smith

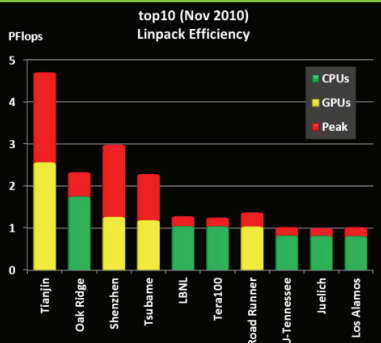
Consommation et Performance

- $P_{ener} = k \cdot V^2 \cdot f$
  V (Voltage), f (Frequence)
- $P_{ener} = k \cdot V(f)^2 \cdot f$
- $P_{ener} \approx k \cdot \#coeurs \cdot f^{\geq 2}$
- Performance Crête= #coeurs . f
- Performance / $P_{ener} \approx 1 / f$

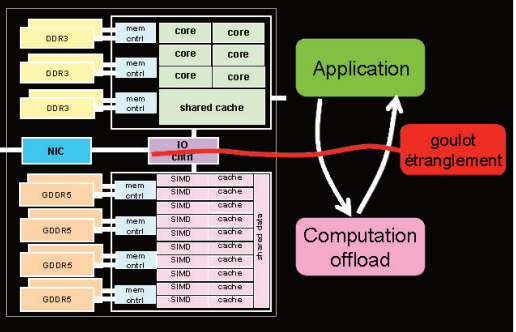
Pour une même Consommation,
Meilleure performance globale
Avec plus de cœurs (lents)
Qu'avec des cœurs rapides



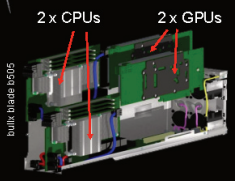
Accélérateurs GPUs dans le top10 HPC



Architecture hybride CPU-GPU



Accélérateurs GPU et applications



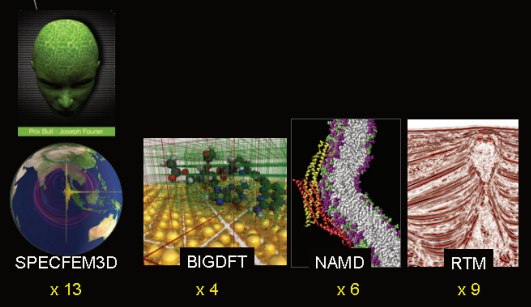
	GPU / CPU ratio
GFlops (DP)	7
Memory BW	4.5
consumption	2
Memory Size	1 / 8

Applications utilisant efficacement les GPUs :

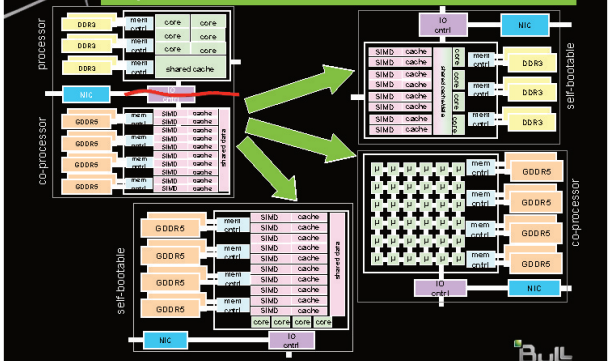
- Noyaux réduits
- Données de taille limitée ou très bonne localité
- (calculs SP)
- Peu de communications
- Cuda ou OpenCL ou HMPP

- *Rendu Graphique*
- *Imagerie Sismique*
- *Dynamique Moléculaire, Astrophysique*
- *Simulations Financières*
- *Analyses Structures, Electromagnétisme*
- *Génomique*
- *Météo/Climatologie/Océanographie*
- *... et plus ...*

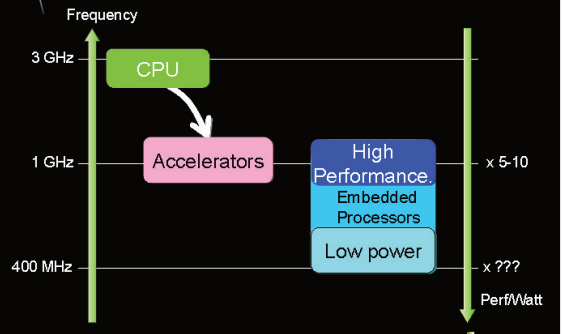
Exemples d'applications accélérées par GPU



Hybrid CPU-GPU architecture evolutionS



Processing Elements



Consommation & Performance

- Les besoins des applications HPC toujours croissants … au delà du Petascale → Exascale → …
- Les accélérateurs vont booster les applications HPC
- Les Accélérateurs Exaflopiques = [r]évolution GPUs de 2011
- De nombreux autres challenges pour les systèmes Exascale (Mémoire & Interconnect BW/Latences, Résilience, Consommation)
- Outils de Développement Exascale encore en gestation
- Modifier, repenser, réécrire les applications HPC pour l'Exascale
- Masses de données à analyser

Une aventure passionnante

bullx
instruments for innovation

2.10 Rémi Barrère (Thales Research & Technology)

Évaluation d'OpenCL et intégration des enseignements pour une programmation performante à haut niveau

www.thalesgroup.com

Évaluation d'OpenCL et intégration des enseignements acquis pour une programmation performante à haut niveau

Rémi BARRERE (Thales Research & Technology)

SYSTEMATIC PARIS REGION SYSTEMS & ICT CLUSTER

Research & Technology

THALES

Agenda

- Contexte
- Portage manuel
- Speed-ups obtenus
- Enseignements acquis
- Utilité des générateurs
- Implémentations réalisées
- Premiers résultats de génération
- Fonctionnalités à venir
- Vidéo
- Questions

THALES

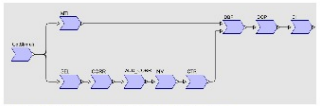
Contexte

Motivations

- **Pourquoi les GPUs ?**
  - Puissance de calcul, large mémoire embarquée
  - Utilisables industriellement ?
- **Pourquoi OpenCL ?**
  - Standard multiplateforme supporté par le Khronos Group
  - Plus verbeux mais très similaire à CUDA
  - Montée en puissance: plus d'architectures et plus de bibliothèques au fil du temps

Application de référence : Adaptive BeamForming

- **Parfait candidat pour les GPUs**
  - Représentative d'une application industrielle, avec nids de boucles imbriqués, régulière



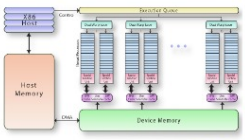
- **Partagée avec d'autres partenaires du projet**
  - Ecole des Mines, HPC Project, CAPS Entreprise, INRIA

THALES

Portage manuel

Portage manuel de Adaptive BeamForming (ABF)

- **Conditions de benchmarking**
  - i7 980X (hexacore)
  - 12 Go RAM
  - Linux 2.6.3x – GCC 4.4.3/4.6.0 – ICC 12.0.1
  - GPUs (+ pilotes/SDK) :
    - carte ATI HD5870
    - carte ATI HD6970
    - carte Nvidia GTX 480
- **Quatre niveaux d'optimisation**
  - Portage initial
  - Optimisation au niveau "host"
  - Optimisation des "kernels"
  - Optimisation complète



THALES

Speed-ups

Résultats du portage manuel

- **Temps de portage (avec validation)**
  - Initial : **6 jours**
  - Optimisation du code host : **4 jours** * spécifique *
  - Optimisation des kernels : **3 jours** * spécifique *
  - Optimisation complète : **1 jour**
- **Comparaison des performances avec le code C séquentiel**

(référence : gcc 4.6.0)

Speed-up	Initial	Host	Kernel	Both	Comments
i7 980X	2.6	3,0	3,2	6,4	Efficacité homogène entre calculs et accès aux données (les gains host/kernels sont cumulatifs)
HD5870	3.5	10.3	9.2	18.6	Plus grande efficacité pour les calculs (les gains host/kernels sont cumulatifs)
HD6970	3.7	12.1	9.7	22.0	Plus grande efficacité pour les calculs (les gains host/kernels sont cumulatifs)
GTX 480	4.4	16.6	10.2	20.2	Plus grande efficacité pour les accès aux données (host primordial)
GTX 580	-	-	-	22.3	(donné à titre de comparaison)

THALES

Enseignements acquis

Premiers enseignements

- Code OpenCL "host" laborieux à écrire, mais essentiel pour la performance (gridification, asynchronisme,...)
- Identification des optimisations spécifiques pour chaque architecture testée :
  - au niveau host (gridification, communications, ...)
  - au niveau des kernels (loop unrolling, vectorisation, mémoires utilisées,...)
- Mise en oeuvre du langage OpenCL sur différentes configurations :
  - mono-gpu
  - multi-gpu homogène
  - multi-gpu hétérogène
- Problématique des kernels (coalescing, shared memory, ...) nettement diminuée avec l'arrivée des nouvelles architectures (utilisation du cache,...)
- Bottleneck situé au niveau du bus PCI express et de l'accès mémoire

Le code host a une importance déterminante pour la performance d'une application portée sur GPU

THALES

Utilité des générateurs

Pourquoi un générateur de code "host" ?

- OpenCL verbeux et souvent répétitif, exemples :
  - fonction main() en C : 50 lignes, en OpenCL : 280 lignes
  - appel à une fonction en C : 1 ligne, en OpenCL : entre 7 et 15 lignes
- Déclaration, allocation et libération des ressources effectuées proprement
- Spécificités des plateformes prises en compte à la génération de code :
  - Gridification : tailles globales et locales
  - primitives utilisées
- Changement de plateforme ou de mapping quasiment immédiat
- Aide au debug et au profiling sur GPU

Pourquoi un générateur de « kernels » ?

- Chaine complète de portage à partir du code C : host + kernel
- Nettement plus compliqués à optimiser que le code « host » : cache, processeurs SIMD, utilisation des différentes mémoires, images, etc...

Le but des générateurs de code est d'avoir une bien meilleure productivité, et non d'obtenir la performance maximale

THALES

Implémentations réalisées

Fonctionnalités déjà implémentées dans SPEAR

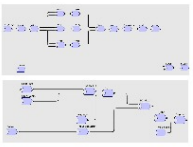
- Génération de code host *spécifique* pour la plateforme utilisée
- Gridification automatique des kernels :
  - Génération basique
  - Génération optimisée (spécifique à l'accélérateur)
- Gestion des architectures multi-plateformes hétérogènes, avec parallélisme de tâche ou de données
- Accélérateur imposé par l'utilisateur ou sélection de la meilleure architecture sur la plateforme spécifiée
- « Pinned memory » utilisée pour les transferts CPU ⇔ GPU
- Gestion de différentes « queues » (software pipelining possible)
- Gestion des « events » pour la synchronisation des calculs et/ou communications
- Code asynchrone généré : le CPU peut effectuer des calculs en même temps que le GPU (problème avec le multi-gpu à régler)
- Support au développeur :
  - Aide au debug avec récupération des tableaux intermédiaires
  - Aide à la performance avec instrumentation du code pour profiling

THALES

Premiers résultats obtenus

Premiers résultats de parallélisation semi-automatique (SPEAR) avec utilisation de kernels « basiques »

- Application ABF :
  - Saisie du graphe en 0.5 jour
  - Mapping sur accélérateur (ATI, NVIDIA, Intel, autres...) en quelques minutes
  - Speed-up similaire à la version « host » (portage manuel en 10 jours)
- Autres applications écrites en C et portées sur GPU
  - Application « Harris » :
    - Portage + optimisation « host » : < 1 jour
    - Speed-up : 10
  - Application « Compression d'Impulsion »
    - Portage + optimisation « host » : < 1 jour
    - Speed-up : 14



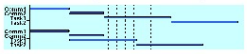
Par rapport à un portage manuel, le gain de productivité est en nette amélioration, et ce pour des performances similaires

THALES

Fonctionnalités à venir

Fonctionnalités en cours d'implémentation

- Software pipelining (recouvrement des communications par des calculs) :
  - Interne => traitement par bloc (strip-mining)
  - Externe => utilisation du double-buffering



- Utilisation du bus PCIe en full-duplex
  (possible avec les cartes Fermi professionnelles)

- Association du générateur de code OpenCL avec d'autres générateurs :
  - C99 (actuel, en cours d'amélioration)
  - MPI (pour des clusters de PCs avec GPUs)
  - OpenMP (pour le code exécuté par un CPU multicore en parallèle du GPU asynchrone)

- Génération de kernels à partir du code C
  - Génération par SPEAR
  - Couplage de SPEAR avec un outil spécialisé (HMPP, PAR4ALL)

THALES

Video

THALES

Questions

Questions ?

THALES

2.11 Pierre Leca (CEA - DAM)

Le prototype de *cluster* OpenGPU au CEA–DAM Contexte et retour d'expérience

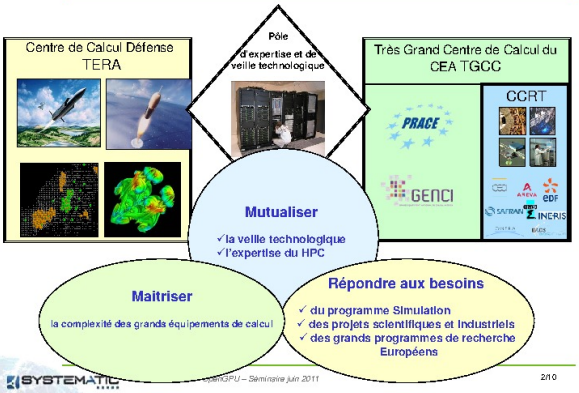
Le prototype de cluster OpenGPU au CEA/DAM

Contexte et Retour d'expérience

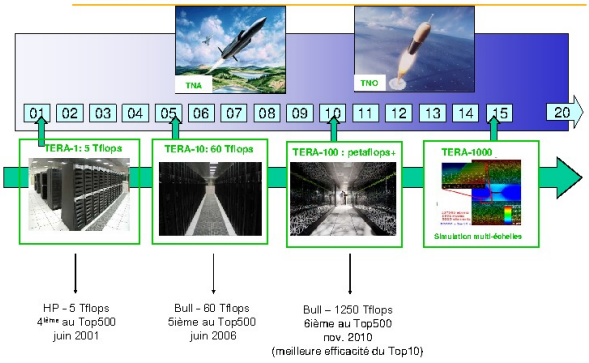
G. Colin de Verdière
P. Leca

CEA/DAM

Le Complexe de Calcul Scientifique du CEA



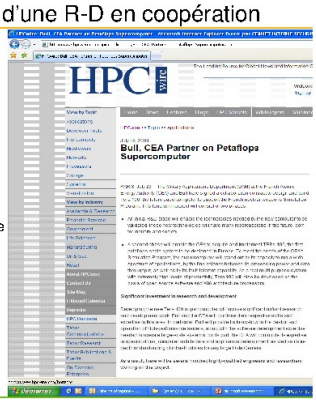
La feuille de route Tera



Tera100: le résultat d'une R-D en coopération

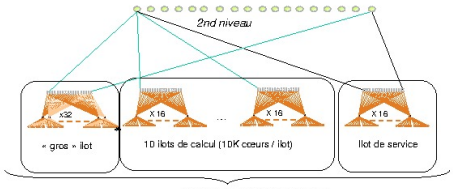
Thématiques de R-D (juin 2008):

1. Architecture
2. Déploiement et évolutivité
3. Maintien en Condition Opérationnelle
4. Coût Total de Possession
5. Extensibilité
6. Liaison avec les infrastructures



Tera100: un cluster hiérarchique et modulaire

Un réseau d'interconnexion à 2 niveaux

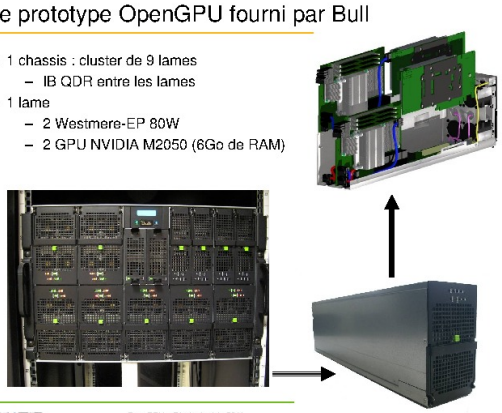


caractéristiques principales

- *1250 Tflops performance max.*
- *4 322 nœuds (198 nœuds de service) - SMP - 4 procs (32 cores)*
- *17 266 Intel Xeon X7560 procs (Nehalem-EX 8 cœurs @ 2.266 Ghz)*
- *138 304 cœurs – 2GO/cœur (5% des nœuds à 4 GO/cœur)*
- *291 TO mémoire*
- *500 GO/s débit vers les PFS*
- *20 PO disques*

Le prototype OpenGPU fourni par Bull

- 1 chassis : cluster de 9 lames
  - IB QDR entre les lames
- 1 lame
  - 2 Westmere-EP 80W
  - 2 GPU NVIDIA M2050 (6Go de RAM)



Utilisation du cluster

- Tests intensifs de CUDA, d'OpenCL et de HMPP
  - ECC actif en permanence
- Tests de tenue aux diverses sollicitations
  - Thermique (charge de calcul)
  - Système (grande stabilité de la configuration)
  - Des ajustements ont eu lieu
- Tests de l'utilisation en mode serveur graphique
  - Cohabitation OpenCL / OpenGL
  - Usage en mode VisuPortal (Oxalya) envisagé

OpenGPU: des acquis et des gains importants

- Contexte : portage de code de physique (orig: 1071 loc)
- Portage
  - CUDA 6136 loc, dével. pénible
  - HMPP 2214 loc, dével. facile
- Bilan
  - Performances au rendez-vous
  - HMPP est une bonne solution
  - Etablissement d'une méthodologie de portage de codes scientifiques

Scalaire	19467.62s
OMPx2	5040.1s
HMPP	820.34s
CUDA	1367.85s
	speedup
OMP / SEQ	3.86
HMPP / SEQ	22.50
CUDA / SEQ	14.36

```
Gain à l'usage de Fermi avec ECC: GPU Titane 2144.s GPU OpenGPU 872.s
Activité des GPU pendant le calcul : (GPU: 87%,Memory: 22%)
```


OpenGPU: retombée au CEA/DAM

- Ajout de 22 chassis du type OpenGPU (Bull B505) dans TERA 100
- M2090
  - Fermi nouvelle génération (T20A)
  - Mémoire plus importante (6Go)
    - Indispensable pour les performances
- HMPP sera proposé en production sur TERA100 pour des codes du CEA/DAM

2.12 Stefanos Vlachoutsis, Antoine Petitet (ESI-Group)

Premiers retours d'expérience sur l'utilisation de GPU pour des applications de mecanique des structures

get it right™

OpenGPU

Premiers retours d'expérience sur l'utilisation de GPU pour des applications de mécanique des structures

Antoine Petitet et Stefanos Vlachoutsis

Juin 2011

get it right™

Sommaire

- Travaux réalisés dans le cadre du projet OpenGPU grâce au soutien de la DGCIS.
- Méthode implicite: résolution de systèmes linéaires creux
- Méthode explicite: Smoothed Particle Hydrodynamics (SPH)

get it right™

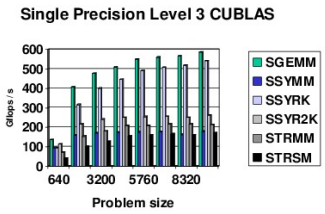
Multi-frontal Solver and CUBLAS

- One of the major workhorses of VPS implicit is the (multi-frontal) linear system direct solver (MUMPS).
- The multi-frontal method operates by design on dense sub-matrices for performance: GEMM and TRSM BLAS Level 3 kernels with sometime a large number of RHS.
- In VPS, main focus is on double precision real and complex operands.
- What about using the CUBLAS provided by NVIDIA …
- … and see what happens on some industrial test cases ?

get it right™

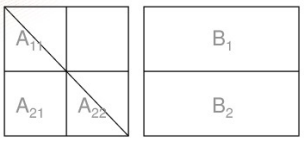
CUBLAS (3.2) Level 3 Performance

- Performance on C2070 (ECC on) including data transfers.
- GEMM and [SY,HE]RK optimized.
- Little has been done for the performance of the other Level 3 BLAS routines.
- True for all other precisions D, C and Z.
- TRSM is important for multiple RHS solve.



get it right™

Recursive GEMM based Level 3 BLAS



$B_1 := A_{11}^{-1} B_1$ (TRSM)

$B_2 := B_2 - A_{21} B_1$ (GEMM)

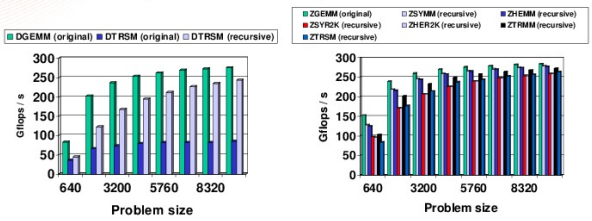
$B_2 := A_{22}^{-1} B_2$ (TRSM)

- Recursive formulation of the TRSM operation.
- Use of native (slower) TRSM on leaves of the tree and (fast) GEMM elsewhere.
- Method can be applied to all Level 3 (and 2) operations.

get it right™

(Recursive) CUBLAS Level 3 Performance

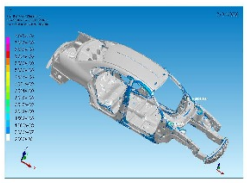
- Asymptotically achieves GEMM performance.

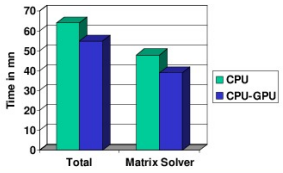


- [SY,HE] rank-2k updates should be implemented by a GEMM call followed by a triangular inplace copy-add.
- The recursive algorithm should be used until there is enough memory to use the above algorithm.

VPS Implicit: Non-Linear Static Test Case

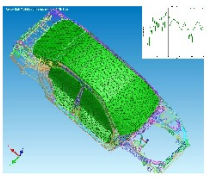
- Double precision real, 1 rhs.
- 12 numerical factorizations and 12 solves.
- Problem size = 4207059, non-zero terms = 130732938.
- Speed-up: 20% over 1 Nehalem core.

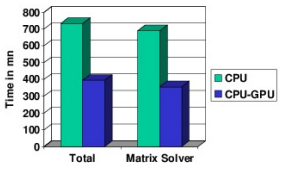




VPS-Implicit: NVH Frequency Response

- Double precision complex, 1258 rhs.
- 25 numerical factorizations and 175 solves.
- Problem size = 409813, non-zero terms = 37229935.
- Speed-up: 2x over 1 Nehalem core.





Internal Acoustics

Conclusions

- Naïve (no data transfer / computation overlap) recursive GEMM based implementation was necessary to handle efficiently large number of rhs.
- The library approach makes the GPU particularly easy to use within complex applications …
- … the performance gain however remains limited. More work is necessary to get better speedups for sparse direct solvers on GPUs.

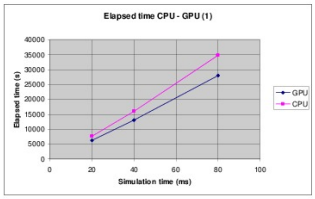
SPH

- La granularité des calculs effectués en SPH en fait une méthode de choix pour le calcul sur GPU.
- Calculs réel simple précision.
- La majeure partie des calculs est uniformément répartie dans (seulement) 3 hot-spots de 5 routines au total.
- Les temps d'exécution reportés inclus les transferts de données vers la carte (pas de recouvrement).
- Comparaison des temps de calcul entre 1 cœur Nehalem W5590 et une carte Nvidia Fermi (C2070 6Gb de RAM).
- Cas industriel: Véhicule roulant sur de l'eau (2730202 points, 1927277 particules, 782575 plaques).

Cuda kernels for one hot-spot

Simulation (ms)	GPU (s)	CPU (s)	Gain(%)
20	6189	7572	18
40	13100	16010	18
80	27980	34770	19

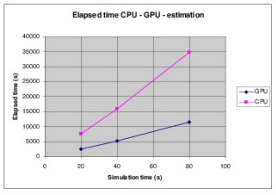
Speedup seems to slightly increase with the simulation time.



Estimation for 3 hot-spots

Simulation (ms)	GPU (s)	CPU (s)	Gain(%)
20	2500	7572	67
40	5300	16010	67
80	11500	34770	67

Data re-use (less data transfers) as the numbers of kernels increase should lead to an even better speedup.



Cuda kernels for 3 hot-spots

Simulation (ms)	GPU (s)	CPU (s)	Gain(%)
20	4846	7572	36

- Number of registers is constant: need to reduce the size of thread blocks to run successfully: performance loss.
- Size of argument list is limited in bytes:
  - 256 Bytes 1.x (C1060)
  - 4 Kbytes 2.0 (C2070)

Conclusions – future work

- SPH: very promising for GPU computing … still need to work on kernels to achieve the potential.
- Hybrid GPU(s) – CPU computing: to investigate.
- Other explicit method topics to investigate: Finite Pointset Method (FPM), Internal forces computing, Contact mechanics, …
- Experiments on clusters of GPUs (MPI+OpenMP+GPUs)
- Tools evaluation for kernel generation: HMPP, PGI

get it right™

www.esi-group.com

2.13 Fouad Boumezbeur (INRA)

Applications du GPU à l'investigation métagénomique : illustration avec le projet européen MetaHIT

Séminaire Aristote – OpenGPU : 1 an après…
Mercredi 8 juin 2011

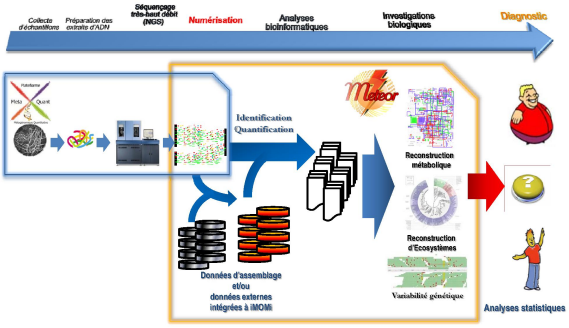
Applications du GPU à l'investigation métagénomique : illustration avec le projet européen MetaHIT

Fouad Boumezbeur
doctorant en bioinformatique
fouad.boumezbeur@jouy.inra.fr

Sous la direction de S. Dusko Ehrlich, Pierre Renault et Jean-Michel Batto
Université Paris-Sud XI - Ecole Doctorale GGC (Gènes, Génomes, Cellules)

Institut MICALIS (MIcrobiologie de la Chaîne ALImentaire au service de la Santé)
Equipe BAC (Bactéries Alimentaires et Commensales)
INRA Centre de Jouy-en-Josas
78350 Jouy-en-Josas (France)

MetaQuant/MetEor
une chaîne de traitement
pour la métagénomique quantitative



Données de séquençage à traiter

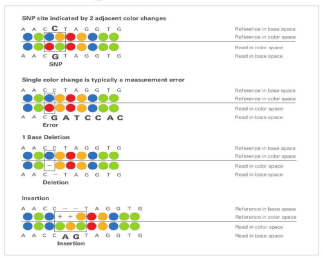
Technologie SOLiD System (Applied Biosystems, Life technologies)

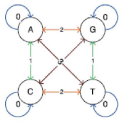
Avantages

- Très haut débit (SOLiD 4 : 700 millions de séquences, soit ~50 Go/semaine)

Contraintes

- Courtes séquences (SOLiD 4 : 50 bases nucléotidiques)
- Lecture des bases par paire (Code couleur) → étape de décodage
- Nouvelle technologie → distribution d'erreurs ?





Un déluge de données à gérer et à analyser !

~ 500 échantillons analysés durant les 24 derniers mois
→ 200.000 fichiers (20 To !)

Acquisition de 2 machines SOLiD 5500 (mise en service : rentrée 2011)

→ Le nombre d'échantillons à analyser va plus que doubler !

Solution actuelle (en place depuis 2008) :
Heuristique de projection des séquences Corona Lite (Applied Biosystems)

Un déluge de données à gérer et à analyser !

Peut-on apporter une solution optimale au problème de projection des courtes séquences ?

Objectif 1
implémenter un algorithme « exact »
pouvant concurrencer les solutions sub-optimales existantes

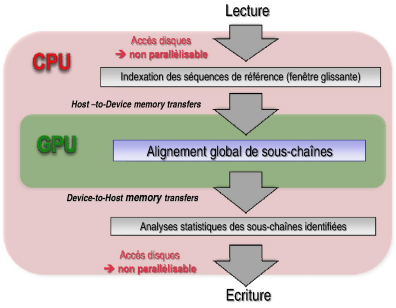
→ plus précis qu'une heuristique ☺
→ plus coûteuse en temps de calcul ! ☹

Comment accélérer cette étape de la chaîne de traitement ?

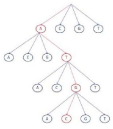
Objectif 2
implémenter un algorithme massivement parallèle

GRIMA (GPU Read Index Mapper and Analyser)

'Mapper' optimal
ET
Outil d'analyse statistique des mappings
développé en C/C++/CUDA



GRIMA (GPU Read Index Mapper and Analyser)



quad trie < arbre à préfixes < DFA

Avantages :

- → recherche de clés plus rapides que dans une table de hachage
- → pas de collisions
- → efficace en termes d'espace mémoire pour manipuler un grand nombre de chaînes courtes
- → Adaptation au paradigme des cartes GPU : **quad trie → double tableau**

Propriétés :

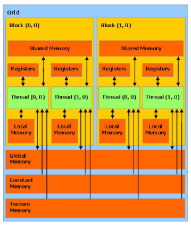
correspondance exacte : temps de complexité en O(l) dans le pire des cas,
avec l = longueur de la chaîne

correspondance approximative : temps de complexité en ~$O(3^{|m|m})$ dans le pire des cas,
avec m = nombre de différences tolérées (en code couleur)

Bonnes pratiques : usage de la mémoire

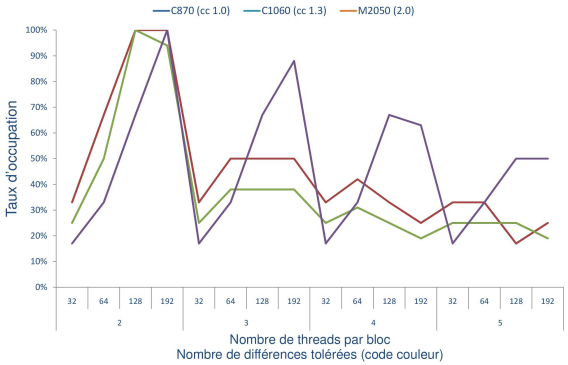
- Registres : 4 cycles ☺
- 'Shared memory' : 20 cycles ☺
- Global/'Device memory' : 200-400 cycles ! ☹

→ *minimiser les accès à la mémoire globale (instructions load/store)*
→ *limiter les transferts sur le bus PCI Express*
→ *privilégier l'utilisation de la 'shared'*



8

Bonnes pratiques : taux d'occupation des SM

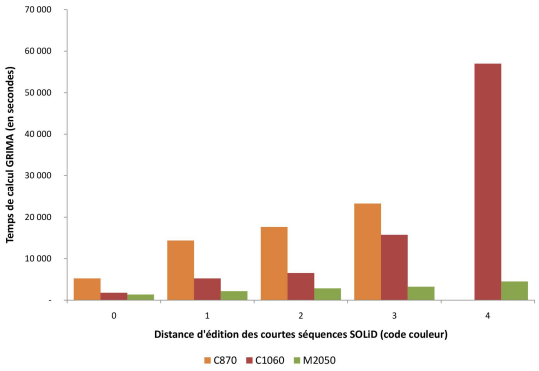


Une évolution rapide de la technologie GPU

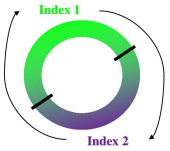
NVIDIA

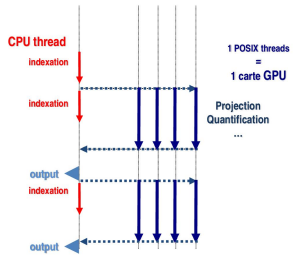
Une évolution rapide de la technologie GPU

Illustration avec l'application GRIMA



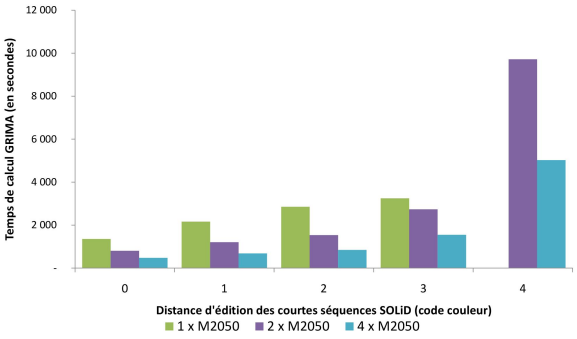
Bonnes pratiques : multi-GPU





Tests de performances de GRIMA

Mise à l'échelle sur nœud de calcul hybride Dell C6100
2 x Intel Xeon 5660 @ 2,6 GHz (2 x 96 Go RAM) + châssis d'extension Dell C410x



Le projet MetaHIT

www.metahit.eu

Vol 464|4 March 2010|doi:10.1038/nature08821 nature

ARTICLES

A human gut microbial gene catalogue established by metagenomic sequencing

Junjie Qin[1*], Ruiqiang Li[1*], Jeroen Raes[2,3], Manimozhiyan Arumugam[2], Kristoffer Solvsten Burgdorf[4], Chaysavanh Manichanh[5], Trine Nielsen[6], Nicolas Pons[7], Florence Levenez[7], Takuji Yamada[1], Daniel R. Mende[2], Junhua Li[1,8], Junming Xu[1], Shaochuan Li[1], Dongfang Li[1,9], Jianjun Cao[1], Bo Wang[1], Huiqing Liang[1], Huisong Zheng[1], Yinlong Xie[1,10], Julien Tap[2], Patricia Lepage[2], Marcelo Bertalan[2], Jean-Michel Batto[2], Torben Hansen[2], Denis Le Paslier[2,11], Allan Linneberg[12], H. Bjørn Nielsen[6], Eric Pelletier[13], Pierre Renault[14], Thomas Sicheritz-Ponten[6], Keith Turner[15], Hongmei Zhu[1], Chang Yu[1], Shengting Li[1], Min Jian[1], Yan Zhou[1], Yingrui Li[1], Xiuqing Zhang[1], Songgang Li[1], Nan Qin[1], Huanming Yang[1], Jian Wang[1], Søren Brunak[4], Joel Doré[5], Francisco Guarner[16], Karsten Kristiansen[17], Oluf Pedersen[18,19], Julian Parkhill[15], Jean Weissenbach[20], MetaHIT Consortium†, Peer Bork[2], S. Dusko Ehrlich[7] & Jun Wang[1,21]

Consortium **MetaHIT** (**Meta**genomics of the **H**uman **I**ntestinal **T**ract), **UE-FP6**

18 Partenaires, dont : BGI-Shenzhen (China), EMBL (Germany), Wellcome Trust Sanger Institute (GB), Université de Wageningen (Pays-Bas), Technical University of Denmark, CHU de Val d'Hebron, Barcelone (Espagne), Danone Research, CEA Genoscope, **CRJ INRA** ...

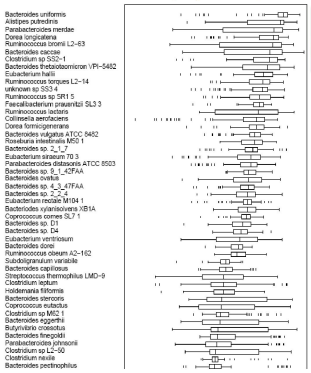
Pays représentés : Grande-Bretagne, Espagne, Pays-Bas, Belgique, Danemark, France, Italie, Chine

- près de 150x la taille du génome humain
- plus de 1.000 espèces à séquencer
- 3.3 millions de gènes prédits

14

La flore intestinale (chez l'homme)

Les 50 bactéries « best hit » (Junjie Qin et al, Nature 2010)



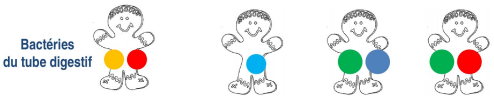
Comment identifier les autres génomes présents dans le catalogue de 3,3 millions de gènes ?

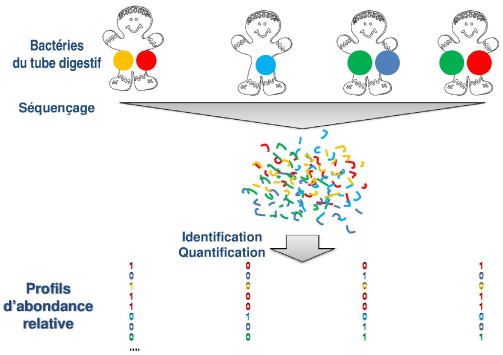
Objectif :

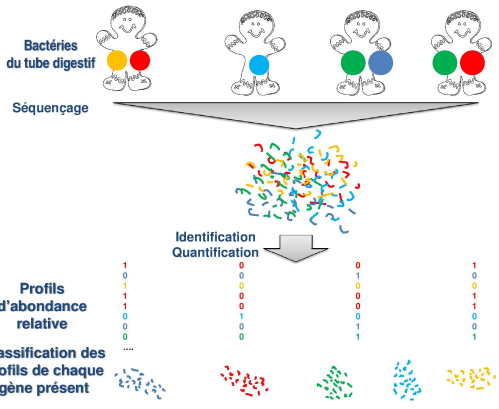
Assigner le maximum de gènes du catalogue à des espèces bactériennes connues ou à tout autre espèce connue ou non

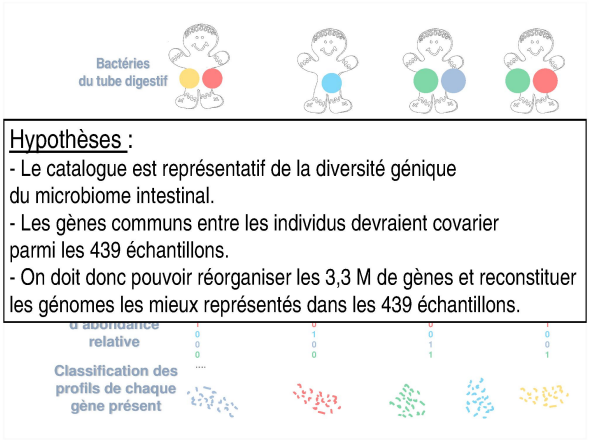
➔ Implémentation d'un "pipeline' de regroupement des gènes pour l'exploration biologique

M. Almeida, Ph.D student
N. Pons, IR





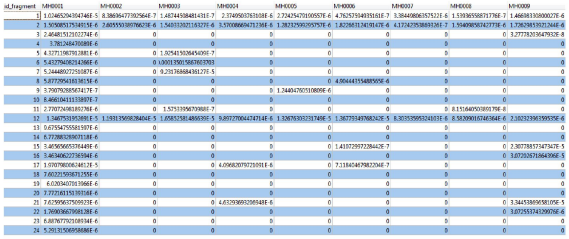




Hypothèses :
- Le catalogue est représentatif de la diversité génique du microbiome intestinal.
- Les gènes communs entre les individus devraient covarier parmi les 439 échantillons.
- On doit donc pouvoir réorganiser les 3,3 M de gènes et reconstituer les génomes les mieux représentés dans les 439 échantillons.

La flore intestinale (chez l'homme)

Analyse métagénomique de 439 échantillons contre le catalogue de 3,3 millions de genes



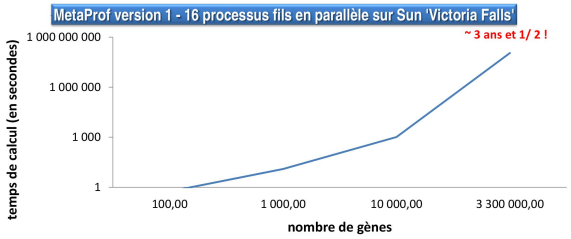
Comment tirer partie de cette analyse pour exploiter le catalogue de gènes ?

MetaProf (Metagenomic Profiles)

Calcul de coefficients de corrélation de Spearman pour chaque paire de gènes du catalogue MetaHIT

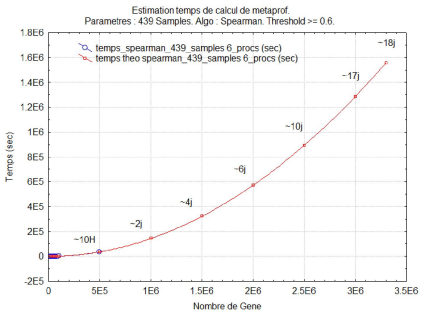
3,3 millions ($3{,}3 \times 10^6$) de gènes
→5500 milliards ($5{,}5 \times 10^{12}$) correlations à calculer
(à 10^{-6} près → simple précision)

MetaProf version 1 - 16 processus fils en parallèle sur Sun 'Victoria Falls'



MetaProf (Metagenomic Profiles)

MetaProf version 2
6 processus openMP en parallèle sur Intel Xeon 5650 x12 @ 2,67 GHz



MetaProf (Metagenomic Profiles)

MetaProf version 3 – cluster hybride Titane (Bull)

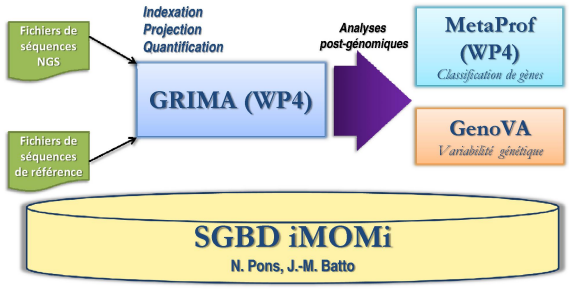
Version CUDA C/C++
Mise à l'échelle sur machine Titane du CEA-CCRT
(modèle : Novascale R422 et serveurs BullX)

10 serveurs S1070 (40 cartes nVidia T10P, 9600 cœurs GPU, 4 Go RAM /carte)
20 nœuds (40 processeurs Intel Nehalem quadri-cœurs,160 cœurs CPU @ 2,93 GHz, 24 Go RAM /noeud)

5×10^{12} correlations en 1h 10min !

Fouad Boumezbeur (INRA), Victor Arslan (AS+)

Perspectives



Intégration des 2 démonstrateurs
à la chaîne de traitement Metagenomic Explorator

Rappel : la loi d'Amdahl

Evaluation de l'accélération maximale théorique après parallélisation

$$S = \frac{1}{1 - P + \frac{P}{N}}$$

S : accélération maximale théorique
P : proportion de temps de calcul CPU du code parallélisable.
N : Nombre de processeurs

INRA UNIVERSITÉ PARIS-SUD 11

MetaHIT HGM catalogue de genes

~300 samples — **Human feces**

DNA extraction and sequencing — *Illumina GAIIx*

~700 Gb — **75b paired-end**

Assembly — *SOAPdenovo [Li et al., Genome Research 20:265-272, 2010]*

12-15 Gb, ~80% aligned, ≥90% identity — **Contigs > 500b**

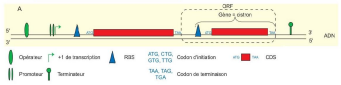
ORF prediction* — *MetaGene [Noguchi et al., Nucleic Acid Res. 34:5623-30, 2006]*

>14M ORFs — **ORFs > 100b**

Redondancy elimination — *Pair-wise ORF comparison : eliminated if ≥ 95% identity over ≥ 90% length*

~3,3M ORFs — **Unique ORFs**

*Open Reading Frame



2.14 Ronan Keryell (HPC Project)

L'outil de parallélisation automatique `Par4All`

Par4All
—
multi-target parallelization

Mehdi Amini[1,2] Béatrice Creusillet[1,4] Stéphanie Even[3]
Onil Goubier[1] Serge Guelton[3,2] Ronan Keryell[1,3]
Janice Onanian McMahon[1] François-Xavier Pasquier[1]
Grégoire Péan[1] Pierre Villalon[1]

[1]HPC Project
[2]Mines ParisTech/CRI
[3]Institut TÉLÉCOM/TÉLÉCOM Bretagne/HPCAS
[4]Université Pierre & Marie Curie/LIP6

2011/06/08
Journée Aristote OpenGPU, École Polytechnique

HPC Project hardware: WildNode from Wild Systems

Through its Wild Systems subsidiary company
- WildNode hardware desktop accelerator
  - Low noise for in-office operation
  - x86 manycore
  - nVidia Tesla GPU Computing
  - Linux & Windows

`http://www.wild-systems.com`

HPC Project software and services

- Parallelize and optimize customer applications, co-branded as a bundle product in a WildNode (e.g. Presagis Stage battle-field simulator, Wild Cruncher for Scilab//...)
- Acceleration software for the WildNode
  - GPU-accelerated libraries for C/Fortran/Scilab/Matlab/Octave/R
  - Transparent execution on the WildNode
- Remote display software for Windows on the WildNode

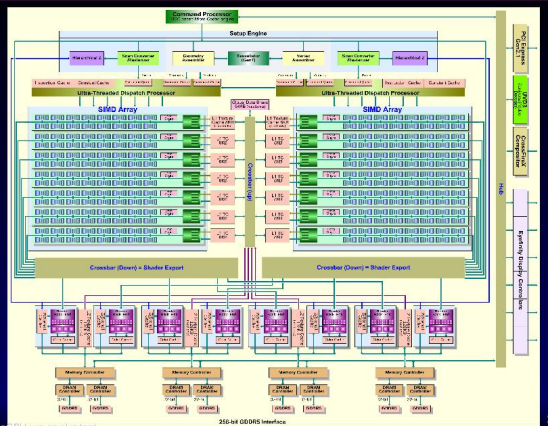
HPC consulting
- Optimization and parallelization of applications
- *High Performance?...* not only TOP500-class systems: power-efficiency, embedded systems, green computing...
- ⇝ Embedded system and application design
- Training in parallel programming (OpenMP, MPI, TBB, CUDA, OpenCL...)

Off-the-shelf AMD/ATI Radeon HD 6970 GPU

- 2.64 billion 40nm transistors
- 1536 stream processors @ 880 MHz, 2.7 TFLOPS SP, 675 GFLOPS DP
- + External 1 GB GDDR5 memory 5.5 Gt/s, 176 GB/s, 384b GDDR5
- 250 W on board (20 idle), PCI Express 2.1 x16 bus interface
- OpenGL, OpenCL
- Radeon HD 6990 double chip card

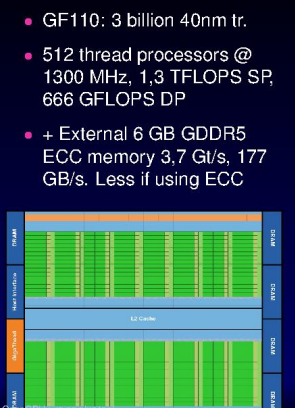
More integration:
- Llano APU (FUSION Accelerated Processing Unit) : x86 multicore + GPU 32nm, OpenCL

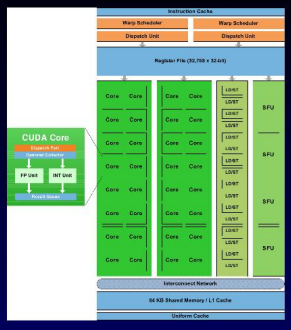
Radeon HD 6870 — big picture



Off-the-shelf nVidia Tesla Fermi M2090 & GTX580

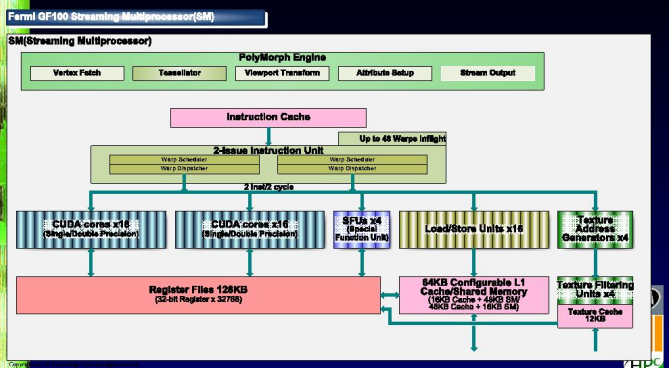
- GF110: 3 billion 40nm tr.
- 512 thread processors @ 1300 MHz, 1,3 TFLOPS SP, 666 GFLOPS DP
- + External 6 GB GDDR5 ECC memory 3,7 Gt/s, 177 GB/s. Less if using ECC





- 247 W on board PCI Express 2.1 x16 bus interface
- OpenGL, OpenCL, CUDA

GF100 Stream Multiprocessor



ARM yourself (I)

- Do some computations where the captors are...
- Smartphone and other sensor networks
- Trade-off between communication energy and inside/remote computations
- Texas Instrument OMAP4470 announced on 2011/06/02
  - 2 ARM Cortex-A9 MPCores @ 1.8GHz with Neon vector instructions
  - 2 ARM Cortex-M3 cores (low-power and real-time responsiveness, multimedia, avoiding to wake up the Cortex-A9...)
  - **SGX544 graphics core with OpenCL 1.1 support**, with 4 USSE2 core @ 384 MHz producing each 4 FMAD/cycle: 12.3 GFLOPS
  - 2D graphics accelerator
  - 3 HD displays and up to QXGA (2048x1536) resolution + stereoscopic 3D
  - Dual-channel, 466 MHz LPDDR2 memory
- ⇝ Current course to have non-x86 servers based on ARM...

ARM yourself (II)

- ∃ Experiments on low power clusters
- Think to evaluate power consumption on your application

Use the Source, Luke...

Hardware is moving quite (too) fast but...

What has survived for 50+ years?
Fortran programs...

What has survived for 40+ years?
IDL, Matlab, Scilab...

What has survived for 30+ years?
C programs, Unix...

- A lot of legacy code could be pushed onto parallel hardware (accelerators) with automatic tools...
- Need automatic tools for source-to-source transformation to leverage existing software tools for a given hardware
- Not as efficient as hand-tuned programs, but quick production phase

We need software tools

- Application development: long-term business ⇝ long-term commitment in a tool that needs to survive to (too fast) technology change
- HPC Project needs tools for its hardware accelerators (*Wild Nodes* from *Wild Systems*) and to parallelize, port & optimize customer applications
- ⇝ OpenGPU project

Outline

Not reinventing the wheel... No NIH syndrome please!

Want to create your own tool?

- House-keeping and infrastructure in a compiler is a huge task
- Unreasonable to begin yet another new compiler project...
- Many academic Open Source projects are available...
- ...But customers need products ☺
- ⇝ Integrate your ideas and developments in existing project
- ...or buy one if you can afford (ST with PGI...) ☺
- Some projects to consider
  - Old projects: gcc, PIPS... and many dead ones (SUIF...)
  - But new ones appear too: LLVM, RoseCompiler, Cetus...

Par4All

- ⇝ Funding an initiative to industrialize Open Source tools
- PIPS is the first project to enter the Par4All initiative

`http://www.par4all.org`

PIPS (I)

- PIPS (Interprocedural Parallelizer of Scientific Programs): Open Source project from Mines ParisTech... 23-year old! ☺
- Funded by many people (French DoD, Industry & Research Departments, University, CEA, IFP, Onera, ANR (French NSF), European projects, regional research clusters...)
- One of the projects that introduced polytope model-based compilation
- ≈ 456 KLOC according to David A. Wheeler's `SLOCCount`
- ... but modular and sensible approach to pass through the years
  - ≈300 phases (parsers, analyzers, transformations, optimizers, parallelizers, code generators, pretty-printers...) that can be combined for the right purpose
  - Polytope lattice (sparse linear algebra) used for semantics analysis, transformations, code generation... to deal with big programs, not only loop-nests

PIPS (II)

- NewGen object description language for language-agnostic automatic generation of methods, persistence, object introspection, visitors, accessors, constructors, XML marshaling for interfacing with external tools...
- Interprocedural *à la* `make` engine to chain the phases as needed. Lazy construction of resources
- On-going efforts to extend the semantics analysis for C
- Around 15 programmers currently developing in PIPS (Mines ParisTech, HPC Project, IT SudParis, TÉLÉCOM Bretagne, RPI) with public `svn`, Trac, `git`, mailing lists, IRC, Plone, Skype... and use it for many projects
- But still...
  - Huge need of documentation (even if PIPS uses literate programming...)
  - Need of industrialization
  - Need further communication to increase community size

Current PIPS usage

- Automatic parallelization (Par4All C & Fortran to OpenMP)
- Distributed memory computing with OpenMP-to-MPI translation [STEP project]
- Generic vectorization for SIMD instructions (SSE, VMX, Neon, CUDA, OpenCL...) (SAC project) [SCALOPES]
- Parallelization for embedded systems [SCALOPES]
- Compilation for hardware accelerators (Ter@PIX, SPoC, SIMD, FPGA...) [FREIA, SCALOPES]
- High-level hardware accelerators synthesis generation for FPGA [PHRASE, CoMap]
- Reverse engineering & decompiler (reconstruction from binary to C)
- Genetic algorithm-based optimization [Luxembourg university+TB]
- Code instrumentation for performance measures
- GPU with CUDA & OpenCL [TransMedi@, FREIA, OpenGPU]

Par4All usage

- OpenGPU 1 year later: Par4All published in the meantime ☺
- Generate from sequential C, Fortran & Scilab code
  - OpenMP for SMP
  - CUDA for nVidia GPU
  - OpenCL for GPU & ST Platform 2012 (on-going)
  - Code for various accelerators [SMECY], Kalray [SIMILAN]... (on-going)
  - SCMP task programs for SCMP machine from CEA and for... cloud computing ☺

Outline

Par4All global infrastructure

Par4All ≡ PyPS scripting in the backstage (I)

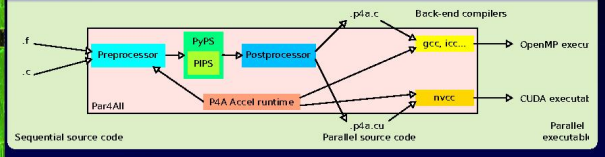
- PIPS is a great tool-box to do source-to-source compilation
- ...but not really usable by λ end-user ☹
- ⇝ Development of Par4All
- Add a user compiler-like infrastructure
  - ⇝ `p4a` script as simple as
    - `p4a --openmp toto.c -o toto`
    - `p4a --cuda toto.c -o toto -lm`
- Be multi-target
- Apply some adaptative transformations
- Up to now PIPS was scripted with a special shell-like language: `tpips`
- Not enough powerful (not a programming language)
- Develop a SWIG Python interface to PIPS phases and interface

Par4All global infrastructure

Par4All ≡ PyPS scripting in the backstage (II)

- All the power of a widely spread real language
- Automate with introspection through the compilation flow
- Easy to add any glue, pre-/post-processing to generate target code

Overview



- Invoke PIPS transformations
  - With different recipes according to generated stuff
  - Special treatments on kernels...
- Compilation and linking infrastructure: can use `gcc`, `icc`, `nvcc`, `nvcc+gcc`, `nvcc+icc`

Par4All global infrastructure

Par4All ≡ PyPS scripting in the backstage (III)

- House keeping code
- Fundamental: colorizing and filtering some PIPS output, running cursor... ☺

Par4All global infrastructure

Coding rules

- Automatic parallelization is not magic
- Use abstract interpretation to « understand » programs
- Undecidable in the generic case ($\approx$ halting problem)
- Quite easier for well written programs
- Develop a coding rule manual to help parallelization and... sequential quality!
  - Avoid useless pointers
  - Take advantage of C99 (arrays of non static size...)
  - Use higher-level C, do not linearize arrays...
  - Organize execution in cleaner loops expressing better parallelism
  - ...
- Prototype of coding rules report on-line on `par4all.org`

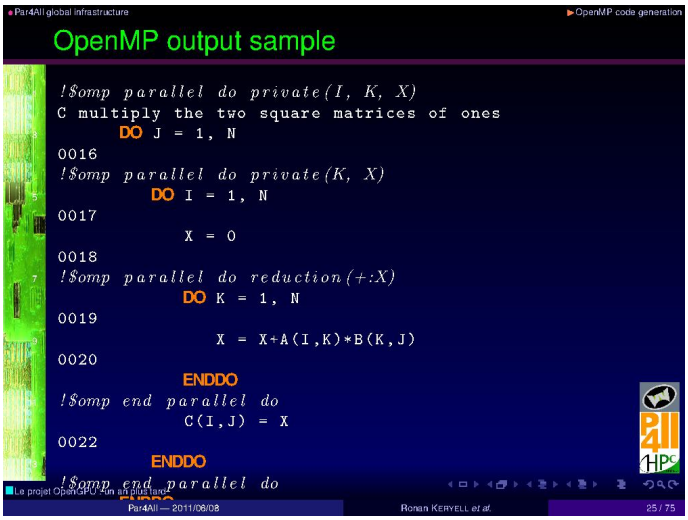
Par4All global infrastructure ▸ OpenMP code generation

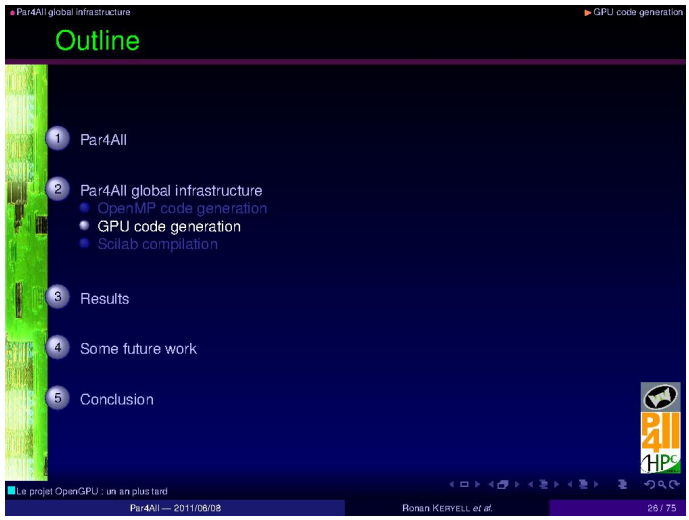
Outline

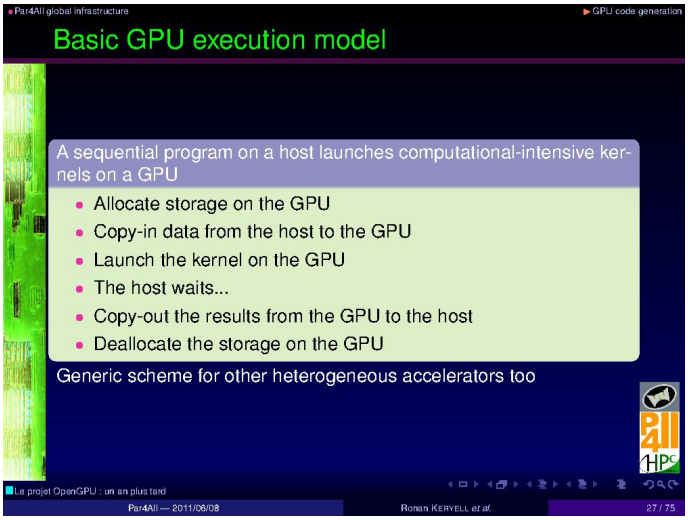
Par4All global infrastructure ▸ OpenMP code generation

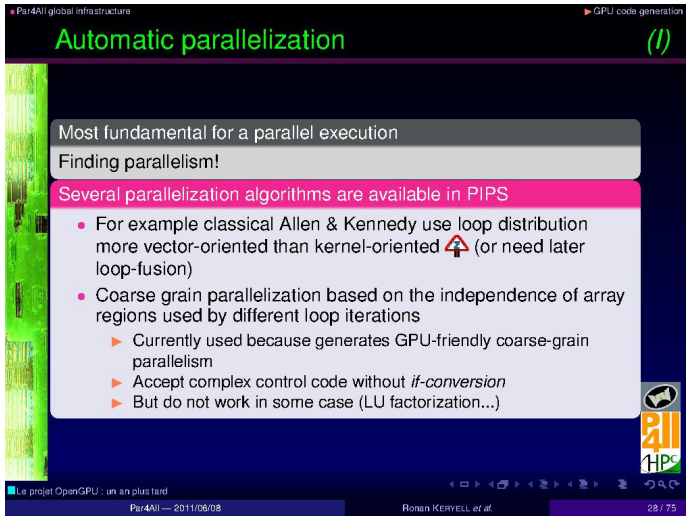
Parallelization to OpenMP (I)

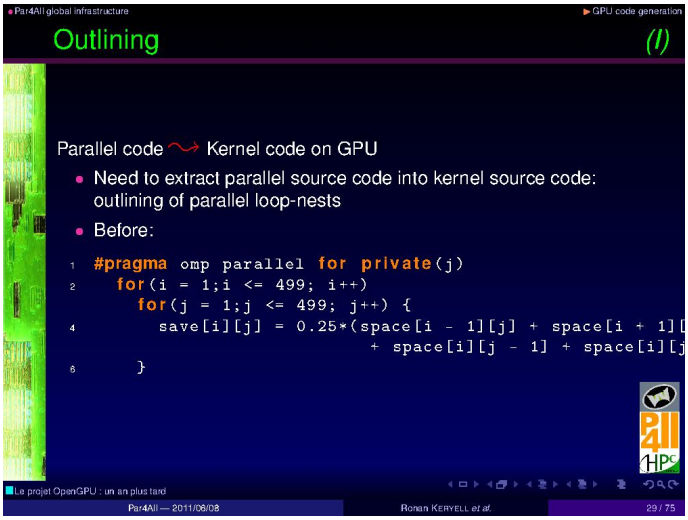
- The easy way... Already in PIPS
- Used to bootstrap the start-up with stage-0 investors ☺
- Indeed, we used only `bash`-generated `tpips` at this time (2008, no PyPS yet), but needed a lot of bug squashing on C support in PIPS...

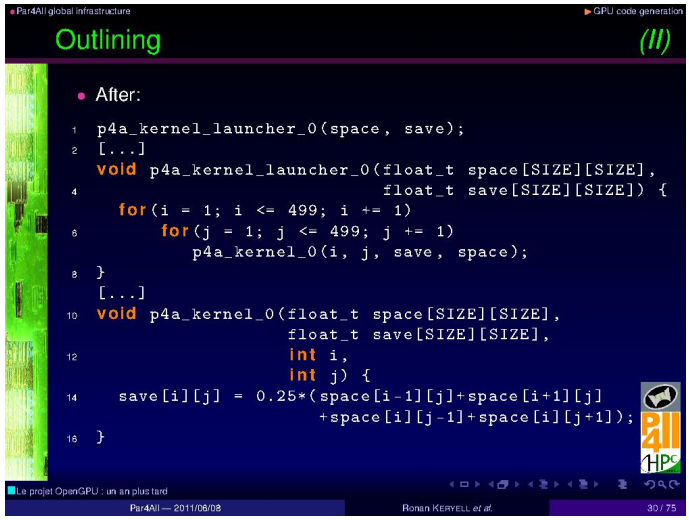

Par4All global infrastructure
OpenMP code generation
OpenMP output sample
!$omp parallel do private(I, K, X)
C multiply the two square matrices of ones
DO J = 1, N
0016
!$omp parallel do private(K, X)
DO I = 1, N
0017
X = 0
0018
!$omp parallel do reduction(+:X)
DO K = 1, N
0019
X = X+A(I,K)*B(K,J)
0020
ENDDO
!$omp end parallel do
C(I,J) = X
0022
ENDDO
!$omp end parallel do
Le projet OpenGPU : un an plus tard
Par4All — 2011/06/08
Ronan KERYELL et al.
25 / 75


Par4All global infrastructure
GPU code generation
Outline
1 Par4All
2 Par4All global infrastructure
OpenMP code generation
GPU code generation
Scilab compilation
3 Results
4 Some future work
5 Conclusion
Le projet OpenGPU : un an plus tard
Par4All — 2011/06/08
Ronan KERYELL et al.
26 / 75


Par4All global infrastructure
GPU code generation
Basic GPU execution model
A sequential program on a host launches computational-intensive kernels on a GPU
Allocate storage on the GPU
Copy-in data from the host to the GPU
Launch the kernel on the GPU
The host waits...
Copy-out the results from the GPU to the host
Deallocate the storage on the GPU
Generic scheme for other heterogeneous accelerators too
Le projet OpenGPU : un an plus tard
Par4All — 2011/06/08
Ronan KERYELL et al.
27 / 75


Par4All global infrastructure
GPU code generation
Automatic parallelization (I)
Most fundamental for a parallel execution
Finding parallelism!
Several parallelization algorithms are available in PIPS
For example classical Allen & Kennedy use loop distribution more vector-oriented than kernel-oriented (or need later loop-fusion)
Coarse grain parallelization based on the independence of array regions used by different loop iterations
Currently used because generates GPU-friendly coarse-grain parallelism
Accept complex control code without if-conversion
But do not work in some case (LU factorization...)
Le projet OpenGPU : un an plus tard
Par4All — 2011/06/08
Ronan KERYELL et al.
28 / 75


Par4All global infrastructure
GPU code generation
Outlining (I)
Parallel code ⇝ Kernel code on GPU
Need to extract parallel source code into kernel source code: outlining of parallel loop-nests
Before:
#pragma omp parallel for private(j)
for(i = 1;i <= 499; i++)
for(j = 1;j <= 499; j++) {
save[i][j] = 0.25*(space[i - 1][j] + space[i + 1][
+ space[i][j - 1] + space[i][j
}
Le projet OpenGPU : un an plus tard
Par4All — 2011/06/08
Ronan KERYELL et al.
29 / 75


Par4All global infrastructure
GPU code generation
Outlining (II)
After:
p4a_kernel_launcher_0(space, save);
[...]
void p4a_kernel_launcher_0(float_t space[SIZE][SIZE],
float_t save[SIZE][SIZE]) {
for(i = 1; i <= 499; i += 1)
for(j = 1; j <= 499; j += 1)
p4a_kernel_0(i, j, save, space);
}
[...]
void p4a_kernel_0(float_t space[SIZE][SIZE],
float_t save[SIZE][SIZE],
int i,
int j) {
save[i][j] = 0.25*(space[i-1][j]+space[i+1][j]
+space[i][j-1]+space[i][j+1]);
}
Le projet OpenGPU : un an plus tard
Par4All — 2011/06/08
Ronan KERYELL et al.
30 / 75

From array regions to GPU memory allocation (I)

- Memory accesses are summed up for each statement as *regions* for array accesses: integer polytope lattice
- There are regions for write access and regions for read access
- The regions can be exact if PIPS can prove that only these points are accessed, or they can be inexact, if PIPS can only find an over-approximation of what is really accessed

From array regions to GPU memory allocation (II)

Example

```
for(i = 0; i <= n-1; i += 1)
  for(j = i; j <= n-1; j += 1)
    h_A[i][j] = 1;
```

can be decorated by PIPS with write array regions as:

```
1 // <h_A[PHI1][PHI2]-W-EXACT-{0<=PHI1, PHI2+1<=n, PHI1<=PHI2}>
  for(i = 0; i <= n-1; i += 1)
3 // <h_A[PHI1][PHI2]-W-EXACT-{PHI1==i, i<=PHI2, PHI2+1<=n, 0<=i}>
    for(j = i; j <= n-1; j += 1)
5 // <h_A[PHI1][PHI2]-W-EXACT-{PHI1==i, PHI2==j, 0<=i, i<=j, 1+j<=n}>
      h_A[i][j] = 1;
```

- These read/write regions for a kernel are used to allocate with a `cudaMalloc()` in the host code the memory used inside a kernel and to deallocate it later with a `cudaFree()`

Communication generation (I)

Use IN/OUT regions

PIPS gives 2 very interesting region types for this purpose

- In-region abstracts what really needed by a statement
- Out-region abstracts what really produced by a statement to be used later elsewhere

- In-Out regions can directly be translated with CUDA into
  - copy-in

```
1 cudaMemcpy(accel_address, host_address,
2            size, cudaMemcpyHostToDevice)
```

  - copy-out

```
1 cudaMemcpy(host_address, accel_address,
2            size, cudaMemcpyDeviceToHost)
```


Loop normalization (I)

- Hardware accelerators use fixed iteration space (thread index starting from 0...)
- Parallel loops: more general iteration space
- Loop normalization

Before

```
for(i = 1;i < SIZE - 1; i++)
  for(j = 1;j < SIZE - 1; j++) {
    save[i][j] = 0.25*(space[i - 1][j] + space[i + 1][j]
                       + space[i][j - 1] + space[i][j
  }
```


Loop normalization (II)

After

```
for(i = 0;i < SIZE - 2; i++)
  for(j = 0;j < SIZE - 2; j++) {
    save[i+1][j+1] = 0.25*(space[i][j + 1] + space[i
                           + space[i + 1][j] + space[i +
  }
```


From preconditions to iteration clamping (I)

- Parallel loop nests are compiled into a CUDA kernel wrapper launch
- The kernel wrapper itself gets its virtual processor index with some `blockIdx.x*blockDim.x + threadIdx.x`
- Since only full blocks of threads are executed, if the number of iterations in a given dimension is not a multiple of the `blockDim`, there are incomplete blocks ☺
- An incomplete block means that some index overrun occurs if all the threads of the block are executed ⚠

From preconditions to iteration clamping (II)

- So we need to generate code such as

```
1 void p4a_kernel_wrapper_0(int k, int l,...)
2 {
    k = blockIdx.x*blockDim.x + threadIdx.x;
4   l = blockIdx.y*blockDim.y + threadIdx.y;
    if (k >= 0 && k <= M - 1 && l >= 0 && l <= M - 1)
6     kernel(k, l, ...);
  }
```

But how to insert these guards?

- The good news is that PIPS owns *preconditions* that are predicates on integer variables. Preconditions at entry of the kernel are:

```
1 //  P(i,j,k,l) {0<=k, k<=63, 0<=l, l<=63}
```

- Guard ≡ directly translation in C of preconditions on loop indices that are GPU thread indices

Optimized reduction generation

- Reduction are common patterns that need special care to be correctly parallelized

$$s = \sum_{i=0}^{N} x_i$$

- Reduction detection already implemented in PIPS
- Efficient computation on GPU needs to create local reduction trees in the thread-blocks
  - Use existing libraries (CuDPP) but may need several kernels?
  - Inline reduction code?
- Right now in Par4All: generate atomic updates

Communication optimization

- Naive approach : load/compute/store
- Useless communications if a data on GPU is not used on host between 2 kernels... ☹
- ⇝ Use static interprocedural data-flow communications
  - Fuse various GPU arrays : remove GPU (de)allocation
  - Remove redundant communications

⇝ New `p4a --com-optimization` option in `p4a 1.1`
Shown at RenPar'2011

Fortran to C-based GPU languages

- Fortran 77 parser available in PIPS
- CUDA & OpenCL are C++/C99 with some restrictions on the GPU-executed parts
- Need a Fortran to C translator (`f2c...`)?
- Only one internal representation is used in PIPS
  - Use the Fortran parser
  - Use the... C pretty-printer!
- But the IO Fortran library is complex to use... and to translate
  - If you have IO instructions in a Fortran loop-nest, it is not parallelized anyway because of sequential side effects ☺
  - So keep the Fortran output everywhere but in the parallel CUDA kernels
  - Apply a memory access transposition phase `a(i,j)` ⇝ `a[j-1][i-1]` inside the kernels to be pretty-printed as C
- Compile and link C GPU kernel parts + Fortran main parts
- Quite harder than expected... Use Fortran 2003 for C interfaces...

Par4All Accel runtime (I)

- CUDA or OpenCL can not be directly represented in the internal representation (IR, abstract syntax tree) such as `__device__` or `<<< >>>`
- PIPS motto: keep the IR as simple as possible by design
- Use some calls to intrinsics functions that can be represented directly
- Intrinsics functions are implemented with (macro-)functions
  - `p4a_accel.h` has indeed currently 3 implementations
    - `p4a_accel-CUDA.h` than can be compiled with CUDA for nVidia GPU execution or emulation on CPU
    - `p4a_accel-OpenMP.h` that can be compiled with an OpenMP compiler for simulation on a (multicore) CPU
- Add CUDA support for complex numbers

Par4All Accel runtime (II)

- On-going support of OpenCL (`p4a-opencl` git branch) written in C/CPP/C++
- Can be used to simplify manual programming too (OpenCL...) ☺
  - Manual radar electromagnetic simulation code @TB
  - One code target CUDA/OpenCL/OpenMP (without Par4All)
- OpenMP emulation for almost free
  - Use Valgrind to debug GPU-like and communication code! (Nice side effect of source-to-source...)
  - May even improve performance compared to native OpenMP generation because of memory layout change

Par4All global infrastructure ▸ GPU code generation

Working around CUDA limitations

- CUDA is not based on C99 but rather on C89 + few C++ extensions
- Some PIPS generated code from C99 user code may not compile ☺
- ⇝ Use some array linearization at some places

Par4All global infrastructure ▸ Scilab compilation

Outline

Par4All global infrastructure ▸ Scilab compilation

Scilab & Matlab *(I)*

- Side effect of MediaGPU ANR project...
- Scilab/Matlab input : *sequential* or array syntax
- Compilation to C code
  - Our COLD compiler is *not* Open Source
  - There is such Open Source compiler from hArtes European project written in... Scilab ☺
- Parallelization of the generated C code
- Use parallel runtime too
- Type inference to guess (crazy ☹) semantics
  - Heuristic: first encountered type is forever
- May get speedup > 1000 ☺
- Wild Cruncher product from HPC Project: *x*86+GPU appliance with nice interface
  - Scilab — mathematical model & simulation
  - Par4All — automatic parallelization
  - //Geometry — polynomial-based 3D rendering & modelling

Results ▸

Outline

Results ▸

Hyantes *(I)*

- Geographical application: library to compute neighbourhood population potential with scale control
- WildNode with 2 Intel Xeon X5670 @ 2.93GHz (12 cores) and a nVidia Tesla C2050 (Fermi), Linux/Ubuntu 10.04, gcc 4.4.3, CUDA 3.1
  - Sequential execution time on CPU: 30.355s
  - OpenMP parallel execution time on CPUs: 3.859s, speed-up: 7.87
  - CUDA parallel execution time on GPU: 0.441s, speed-up: 68.8
- With single precision on a HP EliteBook 8730w laptop (with an Intel Core2 Extreme Q9300 @ 2.53GHz (4 cores) and a nVidia GPU Quadro FX 3700M (16 multiprocessors, 128 cores, architecture 1.1)) with Linux/Debian/sid, gcc 4.4.5, CUDA 3.1:
  - Sequential execution time on CPU: 34.7s
  - OpenMP parallel execution time on CPUs: 13.7s, speed-up: 2.53
  - OpenMP emulation of GPU on CPUs: 9.7s, speed-up: 3.6
  - CUDA parallel execution time on GPU: 1.57s, speed-up: 24.2

Results ▸

Hyantes *(II)*

Original main C kernel:

```
void run(data_t xmin, data_t ymin, data_t xmax, data_t ymax
  town pt[rangex][rangey], town t[nb])
{
    size_t i,j,k;

    fprintf(stderr,"begin computation ...\n");

    for(i=0;i<rangex;i++)
        for(j=0;j<rangey;j++) {
            pt[i][j].latitude =(xmin+step*i)*180/M_PI;
            pt[i][j].longitude =(ymin+step*j)*180/M_PI;
            pt[i][j].stock =0.;
            for(k=0;k<nb;k++) {
                data_t tmp = 6368.* acos(cos(xmin+step*
                    * cos((ymin+step*j)-t[k].longitude
                    + sin(xmin+step*i)*sin(t[k].latitu
```

Results

Hyantes (III)

```
                if( tmp < range )
                    pt[i][j].stock += t[k].stock  / (1 + tm
            }
        }
    fprintf(stderr,"end computation ...\n");
}
```

Example given in par4all.org distribution

Results

Hyantes (IV)

OpenMP code:

```
void run(data_t xmin, data_t ymin, data_t xmax, data_t ymax
{
    size_t i, j, k;

    fprintf(stderr, "begin computation ...\n");

#pragma omp parallel for private(k, j)
    for(i = 0; i <= 289; i += 1)
        for(j = 0; j <= 298; j += 1) {
            pt[i][j].latitude = (xmin+step*i)*180/3.1415926535
            pt[i][j].longitude = (ymin+step*j)*180/3.141592653
            pt[i][j].stock = 0.;
            for(k = 0; k <= 2877; k += 1) {
                data_t tmp = 6368.*acos(cos(xmin+step*i)*co
                if (tmp<range)
                    pt[i][j].stock += t[k].stock/(1+tmp);
```


Results

Hyantes (V)

```
        }
    }
    fprintf(stderr, "end computation ...\n");
}
void display(town pt[290][299])
{
    size_t i, j;
    for(i = 0; i <= 289; i += 1) {
        for(j = 0; j <= 298; j += 1)
            printf("%lf %lf %lf\n", pt[i][j].latitude, pt[i][j
        printf("\n");
    }
}
```


Results

Hyantes (VI)

Generated GPU code:

```
void run(data_t xmin, data_t ymin, data_t xmax, data_t ymax
  town pt[290][299], town t[2878])
{
    size_t i, j, k;
    //PIPS generated variable
    town (*P_0)[2878] = (town (*)[2878]) 0, (*P_1)[290][299]

    fprintf(stderr, "begin computation ...\n");
    P4A_accel_malloc(&P_1, sizeof(town[290][299])-1+1);
    P4A_accel_malloc(&P_0, sizeof(town[2878])-1+1);
    P4A_copy_to_accel(pt, *P_1, sizeof(town[290][299])-1+1);
    P4A_copy_to_accel(t, *P_0, sizeof(town[2878])-1+1);

    p4a_kernel_launcher_0(*P_1, range, step, *P_0, xmin,
    P4A_copy_from_accel(pt, *P_1, sizeof(town[290][299])
    P4A_accel_free(*P_1);
```


Results

Hyantes (VII)

```
    P4A_accel_free(*P_0);
    fprintf(stderr, "end computation ...\n");
}

void p4a_kernel_launcher_0(town pt[290][299], data_t range,
  data_t xmin, data_t ymin)
{
    //PIPS generated variable
    size_t i, j, k;
    P4A_call_accel_kernel_2d(p4a_kernel_wrapper_0, 290,299,
                             step, t, xmin, ymin);
}

P4A_accel_kernel_wrapper void p4a_kernel_wrapper_0(size_t i
  data_t range, data_t step, town t[2878], data_t xmin,
{
    // Index has been replaced by P4A_vp_0:
    i = P4A_vp_0;
```


Results

Hyantes (VIII)

```
    // Index has been replaced by P4A_vp_1:
    j = P4A_vp_1;
    // Loop nest P4A end
    p4a_kernel_0(i, j, &pt[0][0], range, step, &t[0], xmin,
}

P4A_accel_kernel void p4a_kernel_0(size_t i, size_t j, town
  data_t step, town *t, data_t xmin, data_t ymin)
{
    //PIPS generated variable
    size_t k;
    // Loop nest P4A end
    if (i<=289&&j<=298) {
        pt[299*i+j].latitude = (xmin+step*i)*180/3.1415926535
        pt[299*i+j].longitude = (ymin+step*j)*180/3.14159265
        pt[299*i+j].stock = 0.;
        for(k = 0; k <= 2877; k += 1) {
            data_t tmp = 6368.*acos(cos(xmin+step*i)*cos((
```

Results

Hyantes (IX)

```
                -(*(t+k)).longitude)+sin(xmin+step*i)*sin((*
            if (tmp<range)
               pt[299*i+j].stock += t[k].stock/(1+tmp);
         }
      }
   }
```


Results

Stars-PM

- *Particle-Mesh* N-body cosmological simulation
- C code from Observatoire Astronomique de Strasbourg
- Use FFT 3D
- Example given in `par4all.org` distribution

Results

Stars-PM time step (I)

```
void iteration(coord pos[NP][NP][NP],
               coord vel[NP][NP][NP],
               float dens[NP][NP][NP],
               int data[NP][NP][NP],
               int histo[NP][NP][NP]) {
  /* Split space into regular 3D grid: */
  discretisation(pos, data);
  /* Compute density on the grid: */
  histogram(data, histo);
  /* Compute attraction potential
     in Fourier's space: */
  potential(histo, dens);
  /* Compute in each dimension the resulting forces and
     integrate the acceleration to update the speeds: */
  forcex(dens, force);
  updatevel(vel, force, data, 0, dt);
  forcey(dens, force);
```


Results

Stars-PM time step (II)

```
  updatevel(vel, force, data, 1, dt);
  forcez(dens, force);
  updatevel(vel, force, data, 2, dt);
  /* Move the particles: */
  updatepos(pos, vel);
}
```


Results

Stars-PM & Jacobi results with `p4a 1.1.2`

- 2 Xeon Nehalem X5670 (12 cores @ 2,93 GHz)
- 1 GPU nVidia Tesla C2050 CUDA 3.2
- Automatic call to CuFFT instead of FFTW (stubs...)
- 150 iterations of Stars-PM

Execution time	p4a	Simulation Cosmo. 32^3	64^3	128^3	Jacobi
Sequential	(gcc -O3)	0,68	6,30	98,4	24,5
OpenMP *6 threads*	--openmp	0,16	1,28	16,6	13,8
CUDA base	--cuda	0,88	5,21	31,4	67,7
Optim. comm. 1.1	--cuda --com-opt.	0,20	1,17	8,9	6,5
Optim. comm. 1.1.2	--cuda --com-opt.	0,10	0,32	2,1	3,8
Manual optim.	(gcc -O3)	0,05	0,26	1,8	

`p4a 1.1.2` introduce generation of CUDA atomic updates for PIPS detected reductions. Other solution to investigate: CuDPP call generation

Some future work

Outline

Some future work

Eclipse interfacing (I)

Motivation

- Not easy to understand why some code is not parallel or inefficient ☺
- ⇝ Need to represent internal representation to he user
- Need interaction between tools and user
- Useful also to debug the tools themselves ☺
- Reuse existing IDE

Some future work

Eclipse interfacing (II)

⇝ Integration into Eclipse

- Invoke various compilation steps
- Need to attach information to source code
- Source code with hyperlinks and tooltips...
- Graph representation (data dependence, control flow...)
- Hiding/merging information

Some ideas

- Rely on CDT and PTP mode
- Improve existing OpenCL mode
- Define simple tool interaction framework by Eclipse expert
  - Simple textual communication (YAML...)
  - No need to develop inside Eclipse for tool providers
- Still to be defined... WIP

Some future work

Eclipse interfacing (III)

In PIPS: rely on attachments to have IR information flowing through the prettyprinter (back to Emacs PIPS...)

Some future work

Towards multi-GPU with OpenCL & *PU

- Side effect of MediaGPU project...
- LaBRI is working on *PU to distribute kernel code on any node, on any accelerator (GPU, SPU, CPU) ⇝ *PU
- Recent developments: virtual OpenCL device that rely on *PU for execution
  - ⇝ Load balancing on CPU & GPU of kernels ☺
- ⚠ Only work on asynchronous and independent kernels
- Need new phases to detect independent kernels
- Need new phases to split loop nests into independent multi-GPU kernels that cannot shared memory...
- Some hard work to be efficient ☺

Some future work

Stub abstractions (I)

- Real code ≡ source code + library code, intrinsic functions, system library...
- Parallelizer « understands » (at some level...) source code
- ⇝ How to parallelize source code with library code, intrinsic functions, system library... without source code?
- Example : a programmer uses a special IO function that is missed by the compiler ⇝ no results really used ⇝ dead-code elimination
- Use stubs and stub recipes
  - Provide stub functions to satisfy the analysis without having to describe them in the compiler
    For PIPS:
    - Equivalent memory effect for interprocedural abstract interpretation
    - Equivalent I/O effect
    - Set variable values in a polyhedral-friendly way when available

Some future work

Stub abstractions (II)

- Provide some sequential implementation that can be well parallelized
- Provide some parallelized implementation for GPU
- Give information for Par4All Accel runtime or static estimator to know if it is worth to move data on GPU or execute on OpenMP on CPU if not already on GPU (reminiscent HPF (re)distribution...)
- Provide Makefiles/glue/... to compile everything
- ...

- Only discover function use during compilation
- Need to cope with dynamic linking...

On-going generalization through stub concept abstraction + stub broker

Some future work

Future challenges for the real world (I)

- Make a compiler with features that compose: able to generate heterogeneous code for heterogeneous machines with all together:
  - MPI code generation between nodes
  - Generate OpenMP parallel code for SMP Processors inside node
  - Multi-GPU with each SMP thread controlling a GPU
  - Work distribution (*à la *PU?*) between GPU and OpenMP
  - Generate CUDA/OpenCL GPU or other accelerator code
  - Generate SIMD vector code in OpenMP
  - Generate SIMD vector code in GPU code
  - Generating KPN-style task code for special accelerators (SCMP...) [SCALOPES, SMECY] or cloud-computing
- Source-to-source transformation is a key technology
- Rely a lot on Par4All Accel run-time
  - Define good minimal abstractions
  - Simplify compiler infrastructure

Some future work

Future challenges for the real world (II)

- Improve target portability
- Finding a good ratio between specific architecture features and global efficiency
- Future is to static compilation + run-time optimizations...
- PyPS as a generic source-to-source compiler
  - Started as Python abstraction of PIPS pass manager
  - Evolved in a generic source-to-source compiler infrastructure
  - Parallel evolution of Par4All & PyPS ⇝ refactoring of Par4All back to PyPS future features
  - Python in PyPS through multiple inheritance, mix-ins, dynamicity... helps a lot
  - Common refactoring of PyPS+Par4All soon
  - Light-weight (YAML...) interaction with Eclipse [OpenGPU]
  - Combine different source-to-source compilers at phase level
  - Use source core + `#pragma`/decorations + intrinsic functions [SMECY, SIMILAN]
  - Useful to have non-compiler expert writing specialized compilers

Some future work

Future challenges for the real world (III)

Software engineering issue

- How to have compiler parts combine seamlessly? ☹

Conclusion

Outline

Conclusion

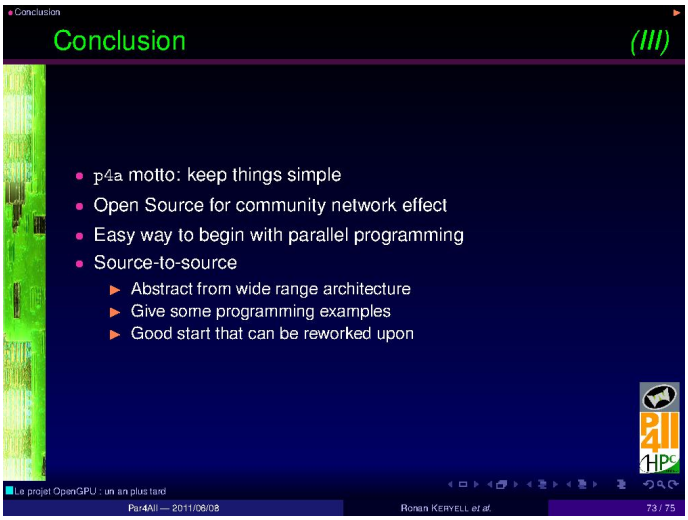
Conclusion (I)

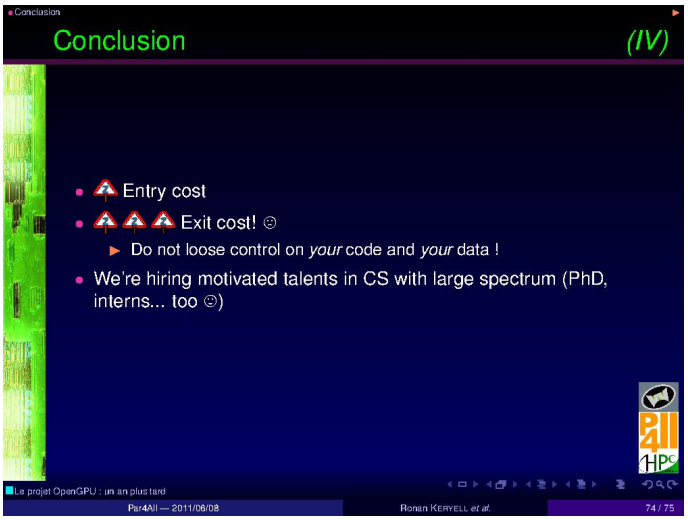
- GPU (and other heterogeneous accelerators): impressive peak performances and memory bandwidth, power efficient
- OpenCL: good opportunity for collaboration between different projects
- Domain is maturing: any languages, libraries, applications, tools... Just choose the good one ☺
- Real codes are often not well written to be parallelized... even by human being ☹
- At least writing clean C99/Fortran/Scilab... code should be a prerequisite
  - Coding rules provided by OpenGPU project
- Take a positive attitude... Parallelization is a good opportunity for deep cleaning (refactoring, modernization...) ⇝ improve also the original code

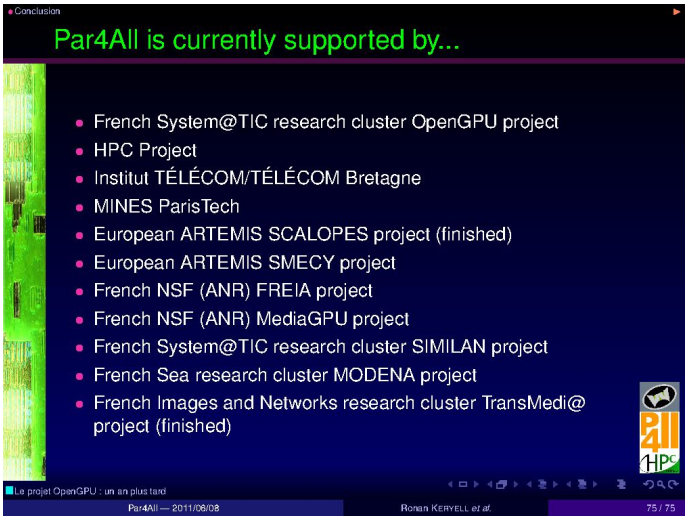
Conclusion

Conclusion (II)

- Time to help application providers from OpenGPU to parallelize their code (SP4) ☺
- Open standards to avoid sticking to some architectures
- Need software tools and environments that will last through business plans or companies
- Open implementations are a warranty for long time support for a technology (cf. current tendency in military and national security projects)


Conclusion
Conclusion (III)
p4a motto: keep things simple
Open Source for community network effect
Easy way to begin with parallel programming
Source-to-source
Abstract from wide range architecture
Give some programming examples
Good start that can be reworked upon
Le projet OpenGPU : un an plus tard
Par4All — 2011/06/08
Ronan KERYELL et al.
73 / 75


Conclusion
Conclusion (IV)
Entry cost
Exit cost! ☹
Do not loose control on your code and your data !
We're hiring motivated talents in CS with large spectrum (PhD, interns... too ☺)
Le projet OpenGPU : un an plus tard
Par4All — 2011/06/08
Ronan KERYELL et al.
74 / 75


Conclusion
Par4All is currently supported by...
French System@TIC research cluster OpenGPU project
HPC Project
Institut TÉLÉCOM/TÉLÉCOM Bretagne
MINES ParisTech
European ARTEMIS SCALOPES project (finished)
European ARTEMIS SMECY project
French NSF (ANR) FREIA project
French NSF (ANR) MediaGPU project
French System@TIC research cluster SIMILAN project
French Sea research cluster MODENA project
French Images and Networks research cluster TransMedi@ project (finished)
Le projet OpenGPU : un an plus tard
Par4All — 2011/06/08
Ronan KERYELL et al.
75 / 75

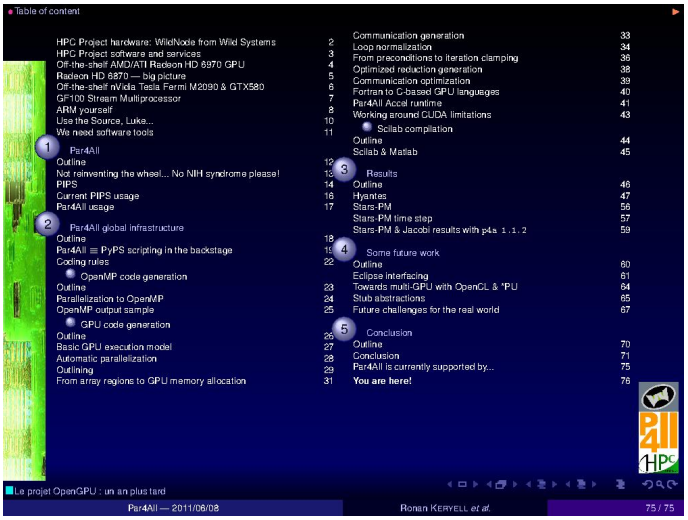


Table of content
HPC Project hardware: WildNode from Wild Systems 2
HPC Project software and services 3
Off-the-shelf AMD/ATI Radeon HD 6970 GPU 4
Radeon HD 6970 — big picture 5
Off-the-shelf nVidia Tesla Fermi M2090 & GTX580 6
GF100 Stream Multiprocessor 7
ARM yourself 8
Use the Source, Luke... 10
We need software tools 11
1 Par4All
Outline 12
Not reinventing the wheel... No NIH syndrome please! 13
PIPS 14
Current PIPS usage 16
Par4All usage 17
2 Par4All global infrastructure
Outline 18
Par4All ≡ PyPS scripting in the backstage 19
Coding rules 22
OpenMP code generation
Outline 23
Parallelization to OpenMP 24
OpenMP output sample 25
GPU code generation
Outline 26
Basic GPU execution model 27
Automatic parallelization 28
Outlining 29
From array regions to GPU memory allocation 31
Communication generation 33
Loop normalization 34
From preconditions to iteration clamping 36
Optimized reduction generation 38
Communication optimization 39
Fortran to C-based GPU languages 40
Par4All Accel runtime 41
Working around CUDA limitations 43
Scilab compilation
Outline 44
Scilab & Matlab 45
3 Results
Outline 46
Hyantes 47
Stars-PM 56
Stars-PM time step 57
Stars-PM & Jacobi results with p4a 1.1.2 59
4 Some future work
Outline 60
Eclipse interfacing 61
Towards multi-GPU with OpenCL & *PU 64
Stub abstractions 65
Future challenges for the real world 67
5 Conclusion
Outline 70
Conclusion 71
Par4All is currently supported by... 75
You are here! 76
Le projet OpenGPU : un an plus tard
Par4All — 2011/06/08
Ronan KERYELL et al.
75 / 75

2.15 Sylvestre Ledru (Scilab)

Retour sur l'intégration de fonctionnalités `GPGPU` dans `Scilab`

Feedbacks on the GPGPU integration in Scilab

June, 8th 2011

Sylvestre Ledru
Cédric Delamarre

Scilab

Scilab

- Numerical computing software
- Interpreted language
- Weakly dynamically typed
- Opensource (Scilab licence) since 1994 and free since Scilab 5.0 (under the CeCILL license – GPL compatible)

Scilab

Objectives in the OpenGPU context

Scilab

Identified issues with GPGPU programming

- Low level programming
- Need (advanced) programming skills
- Not trivial to package
- Interaction with other codes can be hard

Scilab

Objectives

- Provide GPGPU features from Scilab
- Fits into Scilab model and language (high level, math orientied)
- Easy to use
- Simple switch between CUDA and OpenCL
- And ...

Scilab

... Main objective

Fast computations !

Scilab

Technological aspects of sciGPGPU

- Free software (CeCILL licence)
- Scilab is all about double => only double
- Two main components :
  - CUDA / OpenCL kernel approach
  - Wrapping of CUBLAS & CUFFT

Kernel approach

- Provide kernel building and loading

1) Write a simple kernel (OpenCL or CUDA)

2) Build it thanks to the function gpuBuild(file)

3) Load the function

4) Execute the kernel with the Scilab data

5) Retrieve data from the GPU

Kernel approach - Example

1) Write a simple kernel (OpenCL or CUDA)

--- MatrixAdd.cl ---

```
 #pragma OPENCL EXTENSION cl_khr_fp64: enable
__kernel void matrixAdd_sci(__global double* c, __global const double* a, __global const
double* b, int rows, int cols) {
   int idx = get_global_id(0);
   if (idx < rows*cols)  {
      c[idx] = a[idx] + b[idx];
   }
}
```


Kernel approach - Example

2) Build it thanks to the function gpuBuild(file)

```
bin=gpuBuild('matrixAdd')
```

This function takes the *matrixAdd.{cl, cu}* file and compiles thanks to the appropriate compiler the kernel.

Kernel approach - Example

3) Load the function

```
fonc=gpuLoadFunction(bin,"matrixAdd_sci");
```

From the previous built kernel called *bin*, load the kernel function *matrixAdd_sci* into Scilab.

Kernel approach - Example

4) Execute the kernel with the Scilab data

```
rows=8 ; cols=8 ;
A=rand(rows,cols);              // Declare a matrix of size 8, 8
gA=gpuSetData(A);               // Send data to the GPU
gB=gpuAlloc(rows,cols);         // Create a variable of size 8,8 in the GPU
lst=list(gB,gA,gA,int32(rows),int32(cols)); // Declare the datastructure
gpuApplyFunction(fonc,lst,rows/2,cols/2,2,2) ; // Execute the kernel
// block_height, block_width, grid_height, grid_width
```

Kernel approach - Example

5) Retrieve data from the GPU

```
B=gpuGetData(gB);
gpuFree(gA); // To free the memory on the GPU
gpuFree(gB);
```

13

Example : the transpose of a matrix

Example – the kernel is provided in sciGPGPU

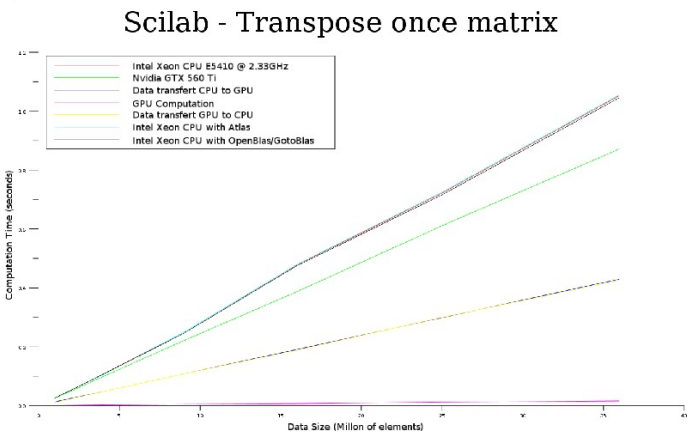
```
MatSize=2000 ;
a = rand(matSize,matSize);
```

CPU
```
cpu = a';
```

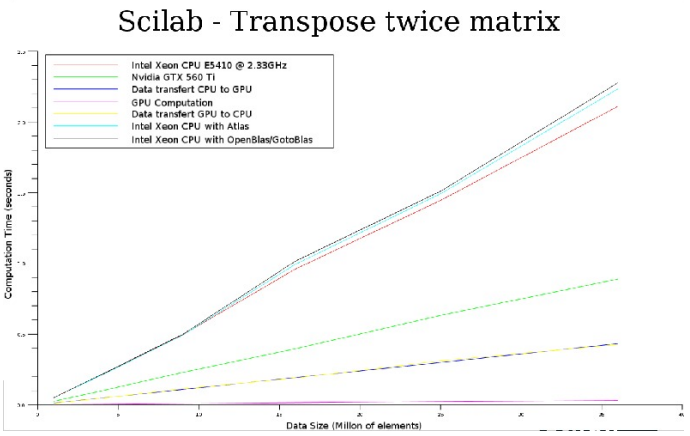
GPU
```
da    = gpuSetData(a);
dress = gpuTranspose(da) ;
gpu = gpuGetData(dres) ;
```

14

Matrix transpose - benchmark 1



Matrix transpose twice - benchmark 2



CUBLAS based features

CUBLAS ?

- Provides linear algebra operations
- Close to the BLAS API (base of most of numerical computing software)

18

CUBLAS wrapping

- Almost as easy as the standard matrix operations : *a=rand(10,10) ; b=rand(10,10) ; c=a*b*
- Provide high level functions managing transparently the memory
- Allow customization on the locality of the data (GPU, CPU, etc) for input and output arguments.
- Various functions mapped (mult, norm, add, sum, etc)

CUBLAS in Scilab - Matrix multiplications

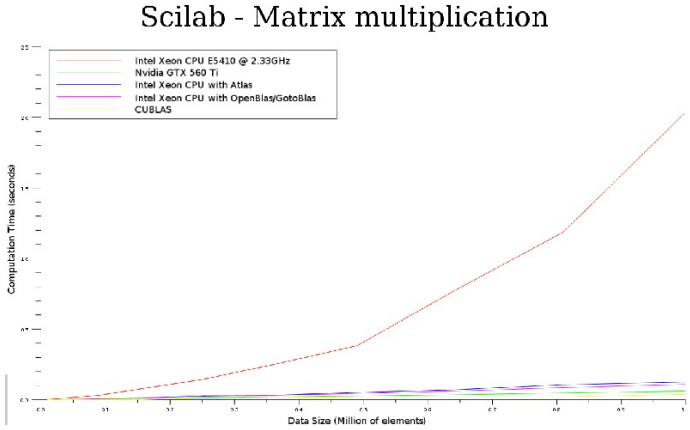
Example

```
MatSize=2000 ;
b = rand(matSize,matSize);
a = rand(matSize,matSize);
```

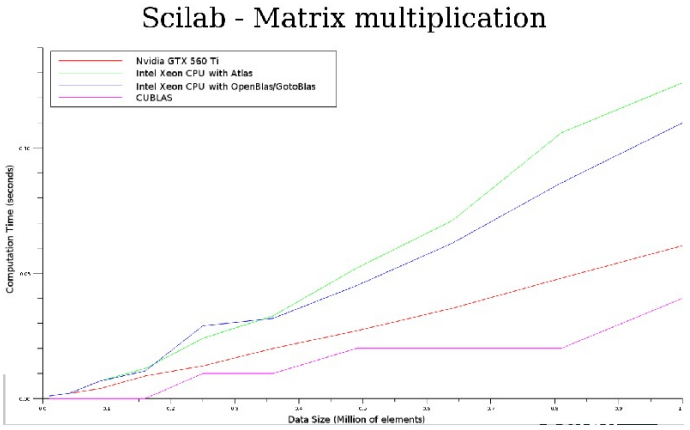
CPU
```
cpu = a * b;
```

GPU
```
gpu = gpuMult(a,b);
```


Matrix multiplications - benchmark 1



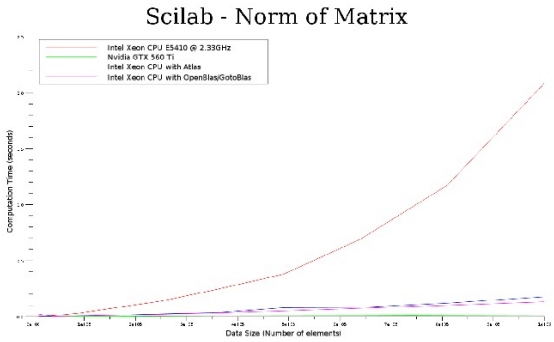
Matrix multiplications - benchmark 2 (without refblas)



Norm of matrix - benchmark

CPU
```
cpu = norm(a, 'fro') ;
```

GPU
```
gpu = gpuNorm(a);
```



CUFFT based features

CUFFT ?

- FFT = Fast Fourier Transform
- CUDA capable Signal processing library

25

CUFFT wrapping

- Provide an alternative to the Scilab functions fft or fftw
- Example :

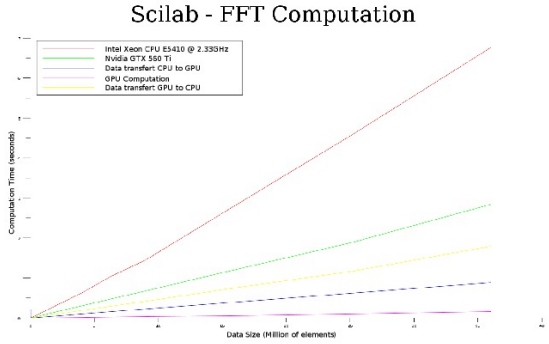
```
a   = rand(100,100) +%i*rand(100,100);
```

CPU
```
cpu = fft(a)
```

GPU
```
da    = gpuSetData(a);
dress = gpuFFT(da) ;
gpu = gpuGetData(dres) ;
```

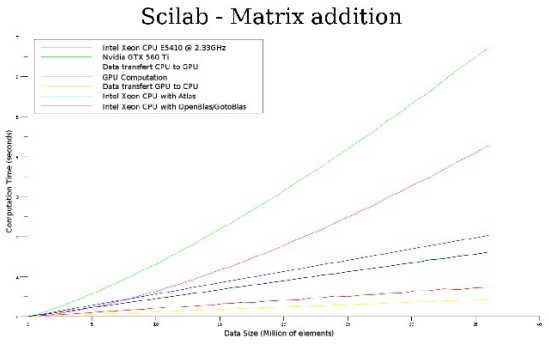
26

Matrix FFT - benchmark



Failed experimentations

(Bad) benchmark on matrix addition



Future

Future

- New release !
- Communication on this module
- Continue to map CUBLAS / CUFFT functions
- Map also CURAND (?) / CUSPARSE (?)
- Introduce some real world demos
- Improve the design pattern library

www.scilab.org

2.16 Albert Cohen (INRIA)

Parallélisation automatique de noyaux de calcul pour GPGPU

Parallélisation automatique de noyaux de calcul pour GPGPU

Soufiane Baghdadi[1], Cedric Bastoul[1], **Albert Cohen**[2], Armin Größlinger[1,3], Konrad Trifunovic[1,2], Sven Verdoolaege[2]

[1]INRIA Saclay – Île-de-France et LRI, Université Paris-Sud 11
[2]INRIA Paris Rocquencourt et École Normale Supérieure
[3]FMI, Universität Passau, Allemagne

Travaux soutenus par la DGCIS

8 juin 2011 – Colloque OpenGPU

1. Motivation

Multi-level parallelism: the need for automatic parallelization

- Instruction-Level-Parallelism (instruction scheduling)
- Fine-grain data parallelism (vectorization)
- Coarse-grain thread-level parallelism

Memory hierarchy

- Registers
- Caches and local memories
- Interconnect

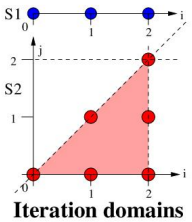
Need for complex loop nest and storage optimizations

Polyhedral compilation

```
for (i=0; i<N; i++)
  {
S1: A[i] = 0;
    for (j=0; j<=i; j++)
S2:   A[i] += B[i][j];
  }
```


Polyhedral compilation

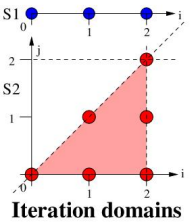
```
for (i=0; i<N; i++)
  {
S1: A[i] = 0;
    for (j=0; j<=i; j++)
S2:   A[i] += B[i][j];
  }
```



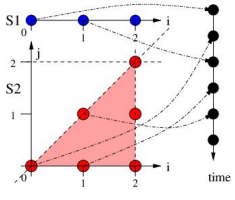
Iteration domains

Polyhedral compilation

```
for (i=0; i<N; i++)
  {
S1: A[i] = 0;
    for (j=0; j<=i; j++)
S2:   A[i] += B[i][j];
  }
```



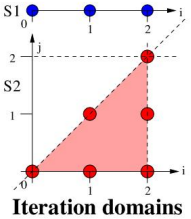
Iteration domains



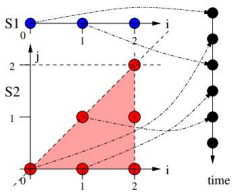
Schedules

Polyhedral compilation

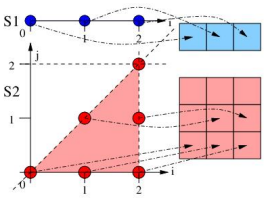
```
for (i=0; i<N; i++)
  {
S1: A[i] = 0;
    for (j=0; j<=i; j++)
S2:   A[i] += B[i][j];
  }
```



Iteration domains



Schedules

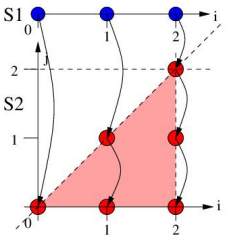


Data accesses

Data dependences

```
for (i=0; i<N; i++)
  {
S1: A[i] = 0;
    for (j=0; j<i; j++)
S2:   A[i] += B[i][j];
  }
```

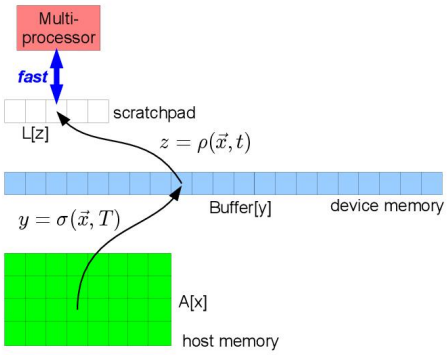
- Affine schedule → complex compositions of loop transformations
- Affine placement → distribution over a multidimensional grid of threads
- Affine storage mapping and layout



Dependences

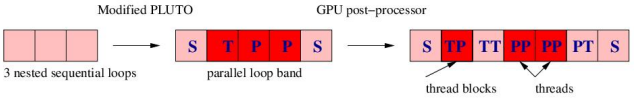
Polyhedral compilation for GPUs

- Device memory and shared memory allocation
- Localization to enable memory access coalescing



Polyhedral compilation for GPUs

- Code generation strategy: modified PLUTO + post-processing
  - **S** outer sequential loops → host
  - **T** parallel tile loop → additional stripmining
    - **TP** parallel bounded iteration tile-point loop → implicit thread block loop
    - **TT** tile-tile loop is sunk inwards
  - **P** parallel point loop → additional stripmining
    - **PP** parallel bounded iteration point-point loop → implicit thread loop
    - **PT** point-tile is sunk inwards, enclosing inner sequential loops
  - Additional steps
    - Systematic separation and hoisting of control flow or not, depending on the type of loop band
    - Generation of memory transfer code (host ↔ device ↔ shared)
    - Custom unrolling strategy



2. Graphite

Why polyhedral model in GCC?

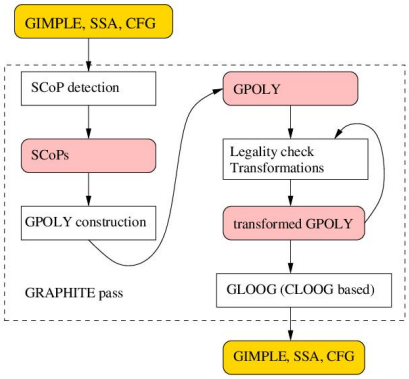
Source to source compilers

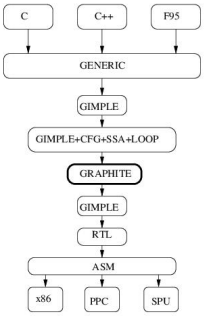
- PLUTO, PoCC, PIPS...
- Usually requires some source code normalization
- Machine details are not exposed
- Need to re-implement compiler analyses and optimizations

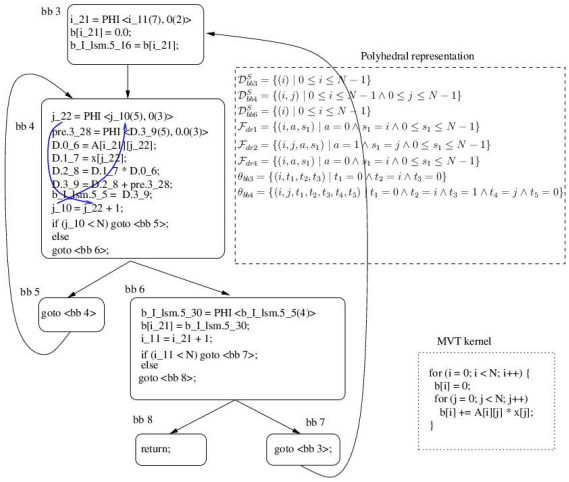
Low level internal polyhedral representation

- Normalized, SSA form
- Scalar evolution analysis (induction variables, reductions)
- Leveraging > **200** passes of GCC
- Tight interaction with vectorizer, parallelizer and memory layout optimizations

Compilation workflow







3. Tools

Compilers and Libraries

- CLooG – The Chunky Loop Generator `http://www.cloog.org`
- PLUTO – Polyhedral parallelizer and locality optimizer
  Currently, working on a variant of PLUTO for CUDA code generation
- PoCC – Polyhedral Compiler Collection
  `http://pocc.sourceforge.net`
- Work in progress to connect PoCC with PIPS/Par4All
  Collaboration with Mines ParisTech and HPC Project
- GCC/Graphite: `http://gcc.gnu.org/wiki/Graphite`
- Work in progress: OpenCL code generation with Graphite
  Basile Starynkevitch's presentation
- PPL – The Parma Polyhedra Library `http://www.cs.unipr.it/ppl`
- ISL – Integer Set Library `http://freshmeat.net/projects/isl`
  (and barvinok, iscc)

2.17 Denis Caromel (INRIA/ActiveEON)

Orchestration de *Workflows* CPU et GPU avec OW2 ProActive Parallel Suite

ProActive Parallel Suite

Orchestration de Workflows CPU et GPU avec OW2 ProActive Parallel Suite

Denis Caromel (INRIA & ActiveEon)
Brian Amedro (INRIA)

Accelerate and Orchestrate Enterprise Applications

Hybrid Cloud Solutions (Private, Public Burst, Multi-Tenants)

OpenGPU Project, Ecole Polytechnique, Palaiseau, June 8th 2011

ProActive Parallel Suite

1

INRIA OASIS Team Composition (35)

- Researchers (5):
  - D. Caromel (UNSA, Det. INRIA
  - E. Madelaine (INRIA)
  - F. Baude (UNSA)
  - F. Huet (UNSA)
  - L. Henrio (CNRS)
- PhDs (11):
  - Antonio Cansado (INRIA, Coni
  - Brian Amedro (SCS-Agos)
  - Cristian Ruz (INRIA, Conicyt)
  - Elton Mathias (INRIA-Cordi)
  - Imen Filali (SCS-Agos / FP7 S
  - Marcela Rivera (INRIA, Conicy
  - Muhammad Khan (STIC-Asia)
  - Paul Naoumenko (INRIA/Régio
  - Viet Dung Doan (FP6 Bionets)
  - Virginie Contes (SOA4ALL)
  - Guilherme Pezzi (AGOS, CIFR
- + Visitors + Interns

Located in Sophia Antipolis, between Nice and Cannes,
Visitors Welcome!

2

ActiveEon Overview

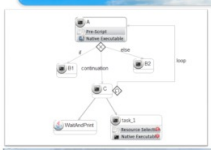
- **ActiveEon**, a software company **born of INRIA**, founded in 2007HQ in the French scientific park Sophia Antipolis
- **Co developing** with INRIA ***ProActive Parallel Suite®***, a Professional Open Source middleware for parallel, distributed, multi-core computing 30 peoples in total
- Core **mission**: Scale Beyond Limits
- Providing **a full range of services** for ProActive Parallel Suite
- **Worldwide** customers and production users:

RENAULT, hp, IFP Energies nouvelles, orange, EDF, GMO, DEXIA, THALES, sirehna, CEA, AMADEUS, EMBL-EBI

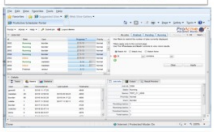
ProActive Parallel Suite

3

ProActive Parallel Suite



Workflow Execution
Studio Editor and Visualization

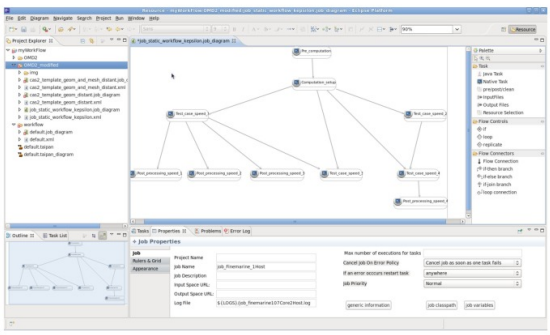


Portal, Multi-Application & Multi-Tenant Enterprise Orchestration

Physical and Virtual Machines Management

4

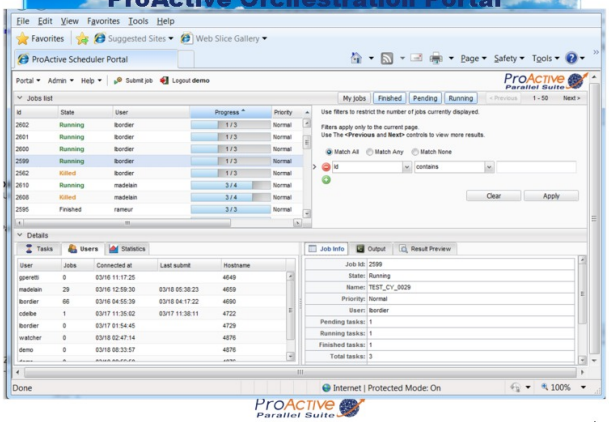
Workflow Studio



ProActive Parallel Suite

5

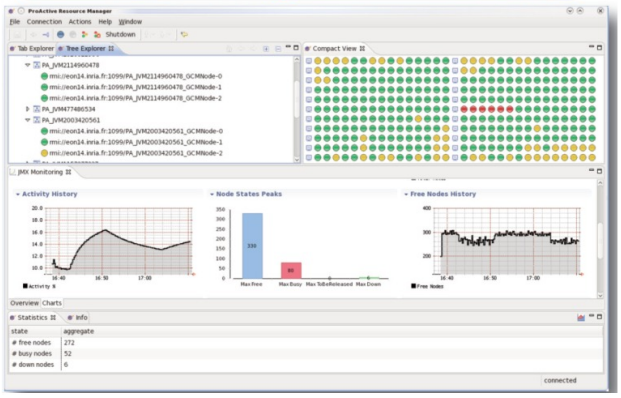
ProActive Orchestration Portal



ProActive Parallel Suite

6

RESOURCING User Interface



ProActive Parallel Suite CPU + GPU Workflow

Live Demo

ProActive Parallel Suite

8

OW2 Consortium

ProActive Parallel Suite CPU + GPU

Live Demo

 www.ow2.org 9

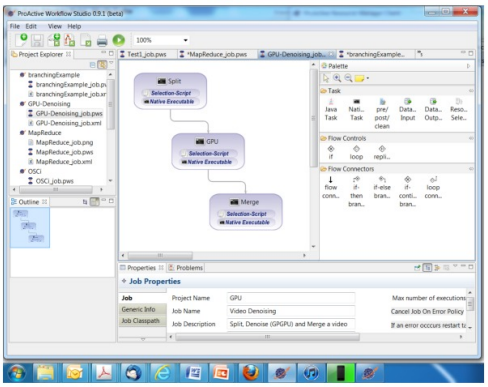
CPU + GPU ProActive Workflows

- Resource selection for each Task of a ProActive Workflow
- Selection of Host with GPU capacity
- Data Transfer to the GPU Host
- Configuration of GPU Capacity at the level of Admin (Number of GPU Nodes, size)
- Freedonm to request one or several GPU capacities for one GPU
- Global Scheduling (Multi-Tenant, Multi-Application) of GPU Tasks

ProActive Parallel Suite

10

Workflow ProActive for CPU and GPU



11

GPU CPU Demo

ProActive Parallel Suite

12

Conclusion

ProActive Parallel Suite

13

Conclusion: ProActive Parallel Suite

- Workflow Studio editor
- Workflow Execution (with Visualization)
- CPU and GPU

Connection with SPEAR for 2-level workflows ?

- Portal, and APIs (Java, REST, ...)
- Multi-Application & Multi-Tenant
- Enterprise Orchestration

- Physical and Virtual Machines Management
- *(Hyper-V, VMware, VirtualBox, KVM, Qemu, Xen)*
- Public Cloud (EC2, Windows Azure), ...

ProActive Parallel Suite

14

ProActive Parallel Suite

15

ow2 Consortium

ProActive Parallel Suite Map Reduce

Live Demo

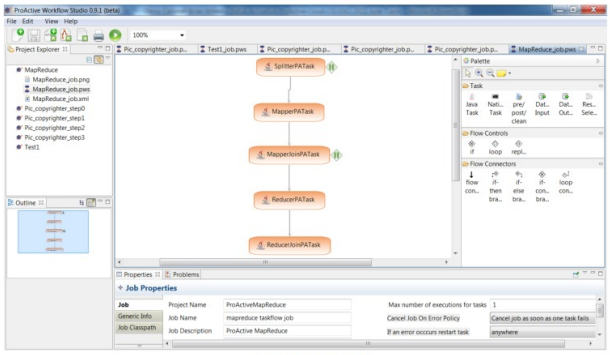
ProActive MapReduce (CO, SP2, Task 2.1)

- Same APIs as Hadoop
  (Easy switch from Hadoop to ProActive)
- Does not requires an HDFS File System
- Runs on general purpose, Multi-tenant, Multi-Applications Grids and Clouds
- Available as PaaS in Java

ProActive Parallel Suite

17

Workflow ProActive MapReduce



ProActive Parallel Suite

18

ProActive Parallel Suite

19

ProActive MapReduce vs. Hadoop+HDFS

File Size	Sequential	Hadoop	PA MapReduce	Speedup
0.7 GB	5m 04s	1m 17s	1m 05s	4.6
4.3 GB	25m 31s	2m 30s	2m 20s	10.9
7.3 GB	46m 00s	3m 31s	3m 30s	13.1
20 GB	2h 07m 00s	8m 30s	7m 09s	17.8
50 GB	5h 19m 00s	21m 05s	25m 11s	12.7
100 GB	10h 38m 00s	43m 23s	58m 42s	10.9

- Data available in a NAS (General purpose storage)
- Transfer to HDFS for Hadoop
- Used directly without copy for ProActive
- Use Case of Map/Reduce on fresh data
- Different ProActive Map/Reduce configuration for recurrent MR on in place Data (e.g. ProActive HDFS interface)

ProActive Parallel Suite

20

2.18 Yann Le Du (ENSCP)

HPU4Science, calcul haute performance sur CPU/GPU avec du matériel grand public : conception et choix logiciels

HPU4Science :

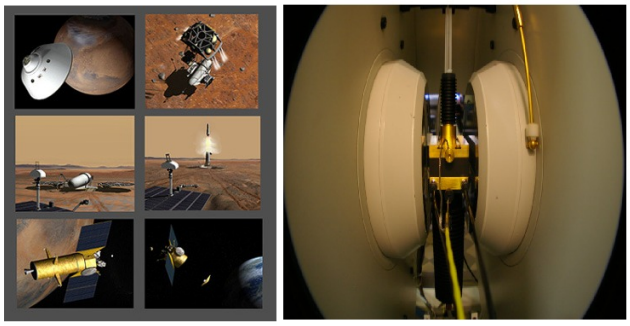
Calcul haute performance sur CPU/GPU avec du matériel grand public : conception et choix logiciels.

Retour d'expérience

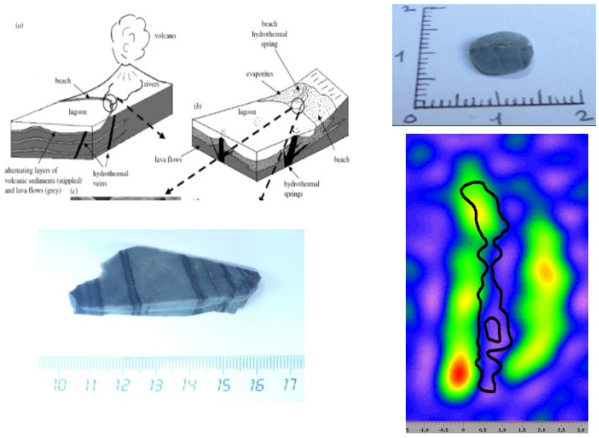
Yann Le Du, Mariem El Afrit, Laurent Binet, Didier Gourier
(LCMCP, Chimie ParisTech)

ANR ENUSIM/ORIGIN 2009-2012, CNES RTS-Exobiologie

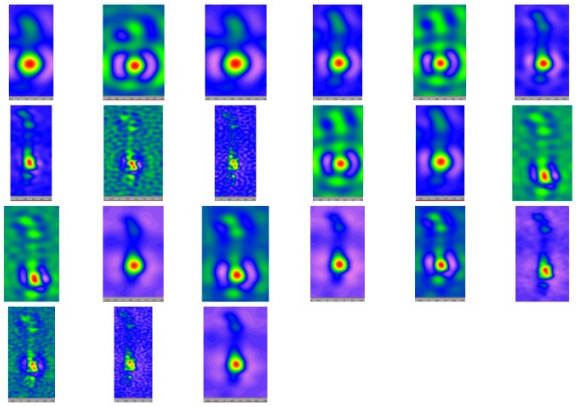
Contexte : imagerie RPE en exobiologie



Contexte : imagerie RPE en exobiologie



Le traitement manuel



Le problème à résoudre

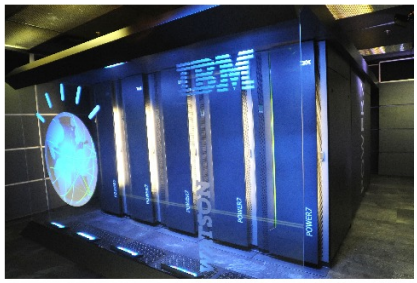
Inverser une équation de Fredholm de type 1

$$r(B) = \int_{\text{sample}} c(B + Gx)\, s(x)\, dx$$

Succès de Watson

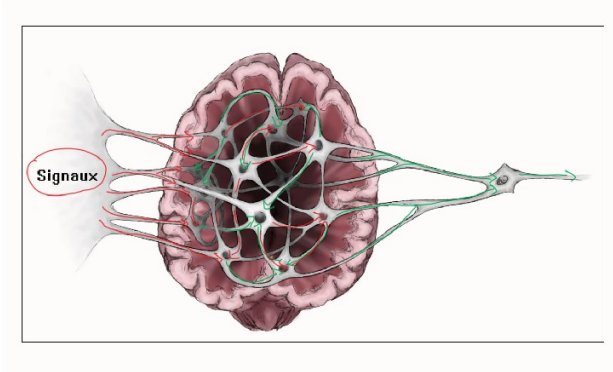
Méthode d'apprentissage automatique possible :

- générer candidats
- combiner candidats



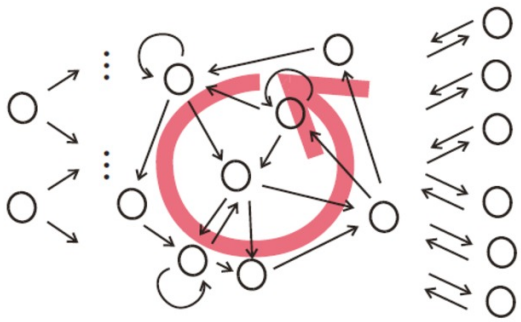
Reservoir computing

Une entrée, un réseau de neurones artificiels récurrent, une couche de neurones de sortie, une sortie.



Reservoir computing

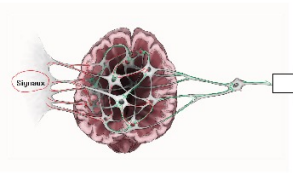
Une entrée, un réseau de neurones artificiels récurrent, une couche de neurones de sortie, une sortie.



La vision matricielle

$$r(B) = \int_{\text{sample}} c(B + Gx)\, s(x)\, dx$$

$$r = Cs \qquad s = C^{-1} r$$



$$s = W_{\text{out}} f\left(W_{\text{eq}} W_{\text{in}} r_{\text{v}}\right)$$

linéarisation

$$W_{\text{out}} = C^{-1}\left(W_{\text{eq}} W_{\text{in}}\right)^{-1}$$

Sgemv, Sgemm, Dgemv, Dgemm... de scikits.cuda.cublas

Literate Programming

D. Knuth, 1984 :

"The practitioner of literate programming can be regarded as an essayist, whose main concern is with exposition and excellence of style. Such an author, with thesaurus in hand, chooses the names of variables carefully and explains what each variable means. ***He or she strives for a program that is comprehensible because its concepts have been introduced in an order that is best for human understanding****, using a mixture of formal and informal methods that reinforce each other."*

Exemple simple, addition de vecteurs

```
__global__ void sumVectors(float
*u, float *v, float *w)

{
  int i = threadIdx.x;
  w[i] = u[i] + v[i];
}
```

3 Defining the kernel that sums the vectors on the GPU

We'll begin with that part, because it's fun. So how do we proceed ? We should first be reminded that vectors are, in practice, lists of numbers, say

$$u = \begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} \quad v = \begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix}$$

which give

$$u + v = \begin{pmatrix} u_0 + v_0 \\ u_1 + v_1 \\ u_2 + v_2 \end{pmatrix} \tag{1}$$

The reason for this comes from the fact that vectors are not simply lists of numbers, but describe an entity that lives in a vector space and that can thus be written as a linear combination of some basis vectors. We thus have

$$u = u_0\mathbf{e}_0 + u_1\mathbf{e}_1 + u_2\mathbf{e}_2$$
$$v = v_0\mathbf{e}_0 + v_1\mathbf{e}_1 + v_2\mathbf{e}_2$$

which of course leads directly to

$$u + v = (u_0 + v_0)\mathbf{e}_0 + (u_1 + v_1)\mathbf{e}_1 + (u_2 + v_2)\mathbf{e}_2$$

proving the component-wise addition of equation (1).

We should thus take three arguments, `u`, `v` and `w` and add the first two component-wise $w_i = u_i + v_i$, where i is an index that runs from 0 to the length of u minus 1. So because we are building a kernel, we use `C`, so we can map the last equation to `C` using:

2a ⟨*Sum the vector component of* `u[i]` *and* `v[i]` *to give* `w[i]` 2a⟩≡ (2d)

```
w[i] = u[i] + v[i];
```

Now, because each thread on the GPU computes the same kernel, yet has access to its own identity – that is its position in the *block* –, we can define the index i to be the thread `x` coordinate in the block,

2b ⟨*Define* `i` *as the thread* `x` *index* 2b⟩≡ (2d)

```
int i = threadIdx.x;
```

The parallelization is right there: all threads compute the same thing, i.e. implement the same function, but we can vary the argument they crunch thanks to an index that comes directly from each thread's position inside the computational *block*. Thus, if one needs to add vectors with, say, n components, it is straightforward to define a computational block of size `(n,1,1)`, so that each of the n threads works on a different vector component. We should thus make sure that the number of such coordinates corresponds to the length of `u` (which is the same as that of `v` and `u`):

2c ⟨*Define the block size of the threads running the computation as* `len(u)` 2c⟩≡

```
theBlockSize = (len(u),1,1)
```

The kernel is called with three arguments, the vectors u, v and w, which on the `C` side take the form of pointers. The function doesn't return anything, because it modifies w directly in memory. We'll call the kernel `sumVectors`:

2d ⟨*Define the sum kernel on* `u`, `v` *and* `w` 2d⟩≡ (1)

```
__global__ void sumVectors(float *u, float *v, float *w)
{
    ⟨Define i as the thread x index 2b⟩
    ⟨Sum the vector component of u[i] and v[i] to give w[i] 2a⟩
}
```

Lit. Prog. : maîtriser la complexité

E. Dijkstra, 1974

"...***one hopes that tomorrow's programming languages will differ greatly from what we are used to now***: *to a much greater extent than hitherto they should invite us to reflect in the structure of what we write down all abstractions needed* ***to cope conceptually with the complexity of what we are designing***.

> *[...]In computer programming our basic building block has an associated time grain of less than a microsecond, but our program may take hours of computation time. I do not know of any other technology covering a ratio of 10^10 or more: the computer, by virtue of its fantastic speed, seems to be the first to provide us with an environment where highly hierarchical artefacts are both possible and necessary. This challenge, viz. the confrontation with the programming task, is so unique that this novel experience can teach us a lot about ourselves.* ***It should deepen our understanding of the processes of design and creation, it should give us better control over the task of organizing our thoughts.***"

Reproducible computational research

D. Donoho, 2008 :

"*Scientific Computation is emerging as absolutely central to the scientific method. Unfortunately, it is error-prone and currently immature:* ***traditional scientific publication is incapable of finding and rooting out errors in scientific computation****; this must be recognized as a crisis.* ***Reproducible computational research, in which the full computational environment that produces a result is published along with the article****, is an important recent development, and a necessary response to this crisis.*"

Le *software*

Système
- Linux (Ubuntu server & desktop)
- Ext4 /
- Btrfs /data*

Développement
- vim
- Python, PyCUDA, Scikits
- C
- git
- Noweb (literate programming)

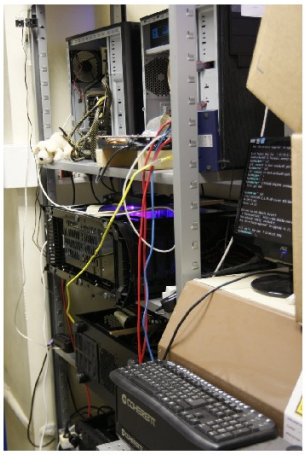
Analyse
- Scipy
- **Sage**, Mathematica
- Mayavi2

Le cluster HPU4Science

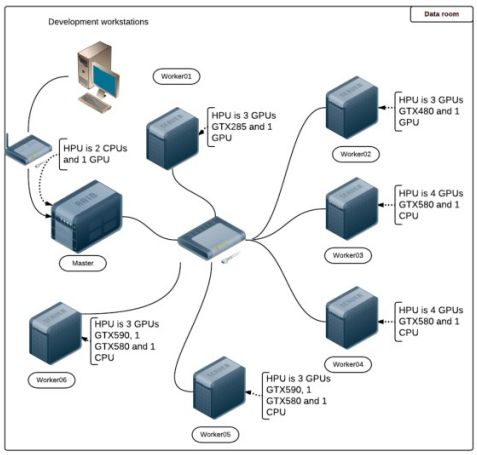
Un master :
- centralise les données, 20To
- cadre les calculs
- combine tous les candidats

6 workers :
- de 3 à 7 GPU
- chacun calcule la même chose
- enregistre les données intermédiaires

Système Linux (Ubuntu Server)

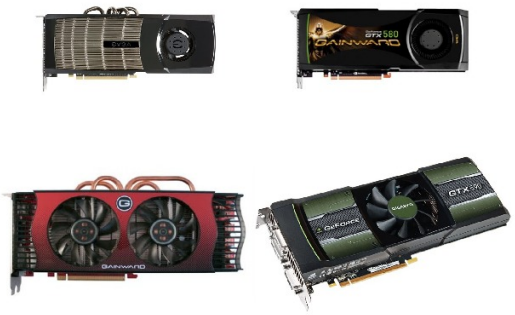


Architecture distribuée

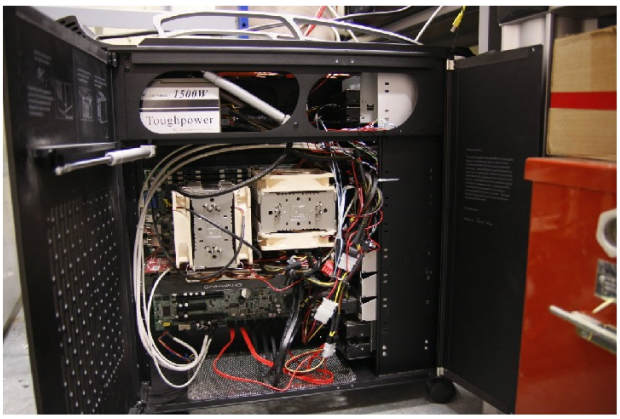


Les cartes GPU

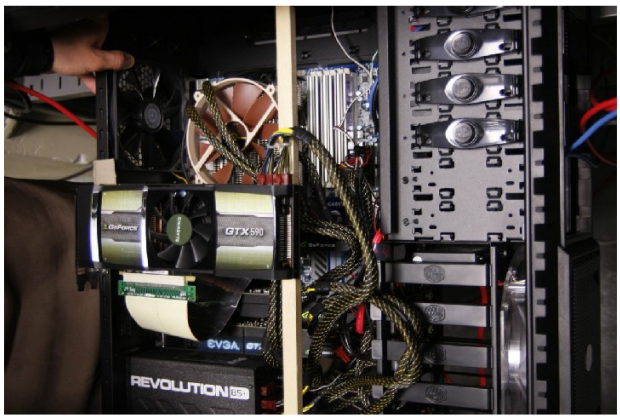
Choix NVIDIA : GTX285, GTX295, GTX480, GTX580, GTX590



Master : centraliser les données



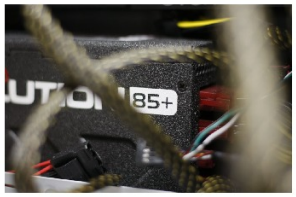
Le worker05



Problèmes rencontrés

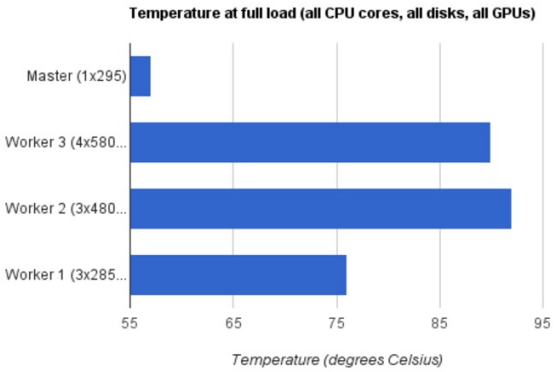
- température
  - risers
  - aération

- surconsommation électrique
  - calculer consommation précise



- composants défectueux
  - tests poussés

Problème 1 : température



Comparaison HPU4Science Tera100

HPU4Science

coût : 30.000€

puissance : 35 TFLOPS

puissance/€ : 1.170 MFLOPS/€

stockage : 20 To

Tera100

coût : 50.000.000€
puissance : 1.250 TFLOPS

puissance/€ : 21 MFLOPS/€

stockage : 20.000 To

Calcul proche des "sources chaudes"

- Nous voulons que le calcul scientifique vienne aux chercheurs, et non le contraire. Cela est rendu possible par les mésocentres.
- HPU4Science vise à encourager un mouvement d'insertion du calcul scientifique haute performance là où naissent les interrogations.
- La chimie pose un problème particulier.

Évolution à court terme (6 mois)

- tester OpenCL avec PyOpenCL
- mettre les codes en open source (License CeCILL ?)
- passer aux cartes GTX670
- développer d'autres applications
  - avec base apprentissage automatique
  - ou pour des traitements classiques
- règle d'or : rester près des "sources chaudes"

Communications

Communications scientifiques

- conférence invitée Europython 2011, juin
- séminaire Aristote, Polytechnique, juin 2011
- séminaire CNES/IDRIS, méthodes d'assimilation, juin 2011
- conférence EuroScipy 2011, août
- séminaire JDEVLOG 2011, septembre

Communications *Information Technology*

- série de trois articles dans Ars Technica (avril/mai 2011) : *High Performance Computing on Gamers PC*

L'équipe HPU4Science

The Core

Yann Le Du, CNRS engineer, launched the HPU4Science project in 2009.

Mariem El Afrit, joined the project early in 2010 as an undergrad, through an internship financed by CNES and then by ANR ORIGINS/ENUSIM. She is now headed for a PhD which should begin in fall 2011 or beginning 2012.

The Asthenosphere

Laurent Binet, Chimie ParisTech associate professor, is an EPR and material science expert at LCMCP/Chimie ParisTech and a main contributor to the use of EPR in exobiology.

Didier Gourier, Chimie ParisTech professor, is an EPR expert and material science expert at LCMCP/Chimie ParisTech and he initiated the use of EPR of carbon in exobiology.

Hervé Vezin, CNRS research director, is continuous and pulsed wave EPR expert at the LASIR in Lille, and an old time close collaborator of the EPR group at the LCMCP. Hervé is the ANR ENUSIM/ORIGIN project leader, and is equipped with bleeding edge Bruker EPR spectrometers, including an imageing continuous wave and pulsed wave EPR device.

L'équipe HPU4Science

The Crust

Yves Frapart, CNRS engineer, is the EPR imageing team leader at Paris 5, and his lab is equipped with multiple Bruker EPR spectrometers, including imageing ones. He has the first Bruker EPR imageing spectrometer that was installed in France, and at the time the most powerful one worldwide.

The Lithosphere

Jean-François Engrand, Paris 6 technician, is the man who knows all about the non computer hardware, and who always finds a solution to all the problems that lurk in those dark regions.

The Biosphere

Frédéric Mentink, a PhD student at LCMCP/Chimie ParisTech working on quantum information and Electron Paramagnetic Resonance, is an apt photographer and is working on giving us the best shots while pondering on the potentialities of compressed sensing.

Diane Robert-Magnenan, philosophy undergrad and artist, joined the project in 2009, and has contributed in many artistic ways, including the drawings for the third *Ars Technica* paper to be published at the end of May.

UPMC PARIS UNIVERSITAS

cnrs

http://hpu4science.org

Chimie Paris ParisTech

ARISTOTE Association Loi de 1901. Siège social : CEA-DSI CEN Saclay Bât. 474, 91191 Gif-sur-Yvette Cedex.
Secrétariat : Aristote, École Polytechnique, 91128 Palaiseau Cedex.
Tél. : +33(0)1 69 33 99 66 Fax : +33(0)1 69 33 99 67 Courriel : Marie.Tetard@polytechnique.edu
Site internet http://www.association-aristote.fr