Technologies des réseaux et NTICs

«CPU-GPU : La convergence ?»

Comment la révolution des «multi coeur» va-t-elle changer la programmation des applications ?

Jeudi 16 octobre 2008

Organisation : GENCI, CAPS-entreprise, Aristote

Coordination scientifique :

- ***Stéphane Requena (IFP GENCI) ;***
- ***Jean-Michel Batto (INRA) ;***
- ***François Bodin (CAPS-entreprise, IRISA) ;***
- ***Philippe d'Anfray (CEA-délégation CI).***

Amphithéâtre Becquerel, École Polytechnique, Palaiseau

http://www.aristote.asso.fr Contact : **info@aristote.asso.fr**

Edition du 2 nivôse an CCXVII (*vulg.* 22 décembre 2008)

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.aristote.asso.fr

Table des matières

Chapitre 1

Programme de la journée

1.1 Introduction

La diffusion des processeurs multi-coeur généralistes et spécialisés, tels les processeurs graphiques, permet de proposer des puissances de calcul potentielles extraordinaires. Cependant l'exploitation efficace de ces nouvelles architectures repose sur une programmation parallèle à multiples niveaux (parallélisme de tâche, de données, vectoriel) qui complexifie la mise en oeuvre des applications. A moyens terme une profonde évolution des pratiques de développement des logiciels sera nécessaire pour tirer parti de cette nouvelle offre. Cette journée se propose d'offrir un aperçu des tendances et défis de l'utilisation des architectures multi-coeurs hétérogènes, en faisant une large part aux présentations et démonstrations des constructeurs ainsi qu'aux retours d'expérience des utilisateurs

1.2 Programme

9h00-9h30	*Accueil-café*	
9h30-10h00	**François Bodin** CAPS-entreprise, IRISA	Introduction : défis et enjeux
10h00-10h30	**William Jalby** PRISM, UVSQ	Future trends
10h30-10h55	**Jean-Christophe Baratault** NVIDIA	Utiliser le GPU comme coprocesseur de calcul, l'écosystème NVIDIA
10h55-11h15	**Bruno Stefanizzi** AMD/ATI	Hybrid Computing
11h15-11h50	*Pause café*	
11h50-12h10	**Jean-François Lemerre** Bull	La vision de Bull pour les futurs systèmes HPC
12h10-12h30	**Olivier Multon** IBM	Le processeur Cell, un multi-cœurs innovant et efficace pour le HPC et le multimédia
12h30-14h10	*Repas* (salle «aquarium»)	
14h10-14h20	**Stephane Requena** GENCI	Actualités GENCI
14h20-14h45	**Raymond Namyst** LABRI	Architectures multicoeurs hétérogènes : à quoi vont ressembler les supports exécutifs «next-gen»
14h45-15h10	**Ronan Keryell** HPC Project	Outils pour programmation multicœurs, GPGPU et autres MP-SoC
15h10-15h30	**Henri Calandra** TOTAL	Accélération de la reverse time migration (RTM) à l'aide de GPGPU, où en sommes nous ?
15h30-15h50	**Thomas Guignon** IFP	Implémentation d'un solveur creux sur GPU
15h50-16h10	*Pause*	
16h10-16h30	**Laurent Domingos** BNP Parisbas	Application du GPU à la calibration de modèles sur processus stochastiques pour la finance
16h30-16h50	**Thierry Deutsch** CEA	Utilisation des GPU et des ondelettes pour le calcul des structures électronique
16h50-17h10	**Mathieu Giraud** LIFL	Bioinformatique et calcul haute-performance
17h10-17h30	**David Delfour** Univ. Perpignan	Transferts d'applications sur le GPU
18h00——	*Cocktail Aristote*	

Chapitre 2

Présentations

2.1 Introduction : défis et enjeux

François Bodin (CAPS-entreprise, IRISA)

Les architectures multi-cœurs homogènes d'Intel ou d'AMD sont dorénavant la panacée des équipements informatiques avec déjà la sortie toute prochaine des processeurs octo-coeurs. Mais lorsque performance et consommation électrique sont de mises, l'utilisation d'accélérateurs matériels spécialisés peut être incontournable.

Certains processeurs spécialisés, tels les GPU, sont peu coûteux et largement diffusés sur leur marché d'origine. Leur exploitation dans des applications autres que leur domaine d'utilisation représente l'exemple parfait du dilemme auquel les développeurs sont confrontés. Les GPU peuvent offrir des gains en performance de plusieurs ordres de grandeur pour des calculs exprimant un parallélisme de données élevé. Pourtant, contrairement aux systèmes multi-cœurs homogènes dont le parallélisme de tâches peut être exploité avec OpenMP ou MPI, aucun modèle data parallel (ou streaming) standard n'est aujourd'hui disponible pour la programmation des architectures spécialisées. Ceci complexifie la tâche des développeurs qui doivent également tenir compte du caractère distribué de la mémoire pour intégrer ces accélérateurs dans leurs applications. Même si l'avenir des GPU dans le domaine du calcul généraliste n'est pas certain, ceux-ci représentent certainement un bon exemple sur comment utiliser efficacement des centaines de cœurs.

Dans cette présentation nous abordons les défis et enjeux de la programmation de ces systèmes hybrides combinant processeurs multi-cœurs généralistes et accélérateurs matériels.

CAPS

Innovative tools for a new paradigm

Défis et enjeux des architectures multicoeurs

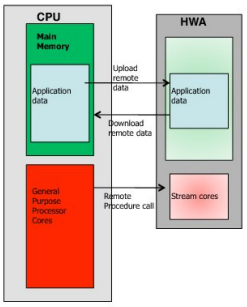
Aristote, Paris, 2008 Octobre 16th

Introduction

- Manycore architectures
  - It is not about parallelism but performance
  - Not only for the HPC market
- Hardware
  - General purpose multicores
  - Application specific multicores
- Moore's law still applies
  - Doubling number of cores every 18 months
  - Power/flop is the new efficiency scale

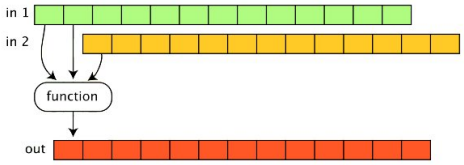
Manycore Architectures

- General purpose cores
  - Share a main memory
  - Core ISA provides fast SIMD instructions
- Streaming engines (GPU, Cell, ...)
  - Application specific architectures ("*narrow band*")
  - Vector/SIMD
  - Can be extremely fast
  - PCIx based communication with main memory
- Hundreds of cumulated Gigaflops (DP)
  - But not easy to take advantage of
  - One platform type cannot satisfy everyone



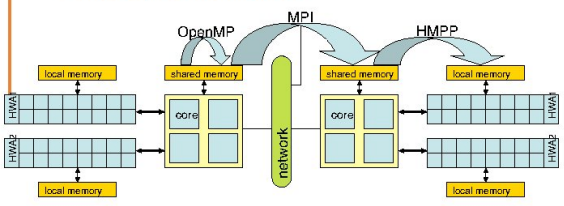
Streaming

- A similar computation is performed on a collection of data (*stream*)
  - There is no data dependence between the computation on different stream elements



Multiple Parallelism Levels

- Amdahl's law is forever, all levels of parallelism need to be exploited
  - Hybrid parallelism needed

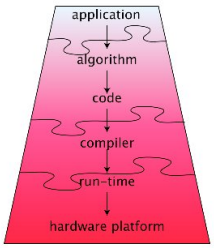


The Past of Parallel Computing, the Future of Manycore?

- The Past
  - Hundreds of parallel languages were proposed
  - HPC focused
  - Microprocessor or vector based, homogeneous architectures
  - Trained programmers
- The Future
  - HPC@home: New applications (medical, ray tracing, ...)
  - Thousands of heterogeneous systems configurations
  - Asymmetry Issues
  - A new competition between "*high value computing cycles*" providers

The Challenges

- Programming
  - Medium
- Resources management
  - Medium
- Application deployment
  - Hard
- Portable performance
  - Extremely hard



Programming

- MPI and OpenMP are the defaults for GP core
  - Hybrid computing difficult but more suited to current architectures
- GPU
  - OpenCL, CUDA, CAL/IL, Brook, …
- Automatic parallelization has not shown to be effective
- New languages/approaches should provide
  - Portability, programming efficiency, performance on a large set of targets

Hardware Resources Management

- Applications on a node are in competition for hardware resources
  - especially non time shared ones such as accelerators
- Runtime should provide answer to
  - How to deal with resources allocation/workload?
  - How to set/deal with applications priority?
  - How to deal with resources unavailability?
  - How to deal with asymmetry?
- Programs should have a view of available resources
  - What runtime API

Application Deployment

- Binary codes that can run on many platforms
  - Simple software distribution
  - Performance preservation
- Questions to answer
  - How to avoid application recompilation on the target node?
  - How to adapt to hardware configuration?
  - How to deal with new hardware?

Portable Performance

- *Portable performance* is achieved when moving to the new generation of multicore efficiency/ scalability is automatically obtained
- The most difficult challenge
  - No existing accurate performance predictive model
- How to deal with
  - Heterogeneity?
  - Changing workloads?
  - Memory hierarchy, data layout, …?

Research Directions

- New Languages
  - X10, Fortress, Co-Array Fortran, UPC, …
- Libraries
  - Atlas, MKL, Global Array, Spiral, Telescoping languages, TBB, …
- Compilers – Key for the short/mid term
  - Classical compiler flow needs to be revisited
  - Acknowledge lack of static performance model
  - Adaptative code generation
- Architectures
  - Integration on the chip of the accelerators
    - Fusion, Larrabee, …
  - Alleviate data transfers costs

About Compilers

- Current compilers
  - Have insufficient understanding about program input, architecture
  - Have to deal with a very large optimization space
  - General purpose tools
- Compilers are not good at
  - Understanding whole programs
  - Understanding performance
  - Making decisions
    - Finding global optimization strategies
    - What code (suite of) transformations and when
- Compilers are good at
  - Dealing with local compute intensive tasks
  - Transforming, duplicating, specializing, generating codes

Future of Compilers for Manycores

- What's new!
  - More processing time can be spent on the code generation and optimization processes
- Mix offline and online techniques
  - Iterative compilation
  - Machine learning
  - Speculative techniques
  - Adaptation
  - Runtime compilation and optimization
  - Better understanding of libraries

Conclusion

- Very exciting time!
- The huge amount of computing power proposed is excellent news
  - Many new applications coming
- Will impact on all software industry
  - Division of labor between performance oriented programmers and application oriented programmers
  - Domain specific approaches

CAPS

Innovative tools for a new paradigm

2.2 Principes des architectures multicoeurs

William Jalby (PRISM, UVSQ)

In 3 years, the computer architecture landscape has deeply changed : after hitting the menmory wall and the ILP wall a third wall has popped up : power. Taking into account all of these constraints is becoming a key challenge. The multicore avenue solves (but only partially) these problems. We will review a few trends in processor architecture and discuss some of their impact on applications.

FUTURE TRENDS

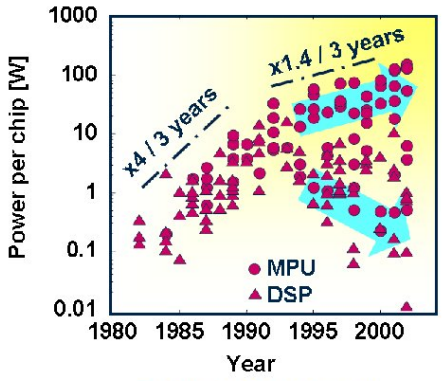
William Jalby

LRC ITA@CA (CEA DAM/ University of Versailles St-Quentin-en-Yvelines)
FRANCE

Outline

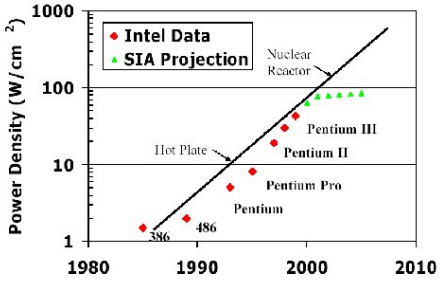
- Power issues
- A few ideas on architecture
- Software

The Power Challenge: Hottest chips published in ISSCC



T. Kuroda, Keio University

Power Density [Hu et al, MICRO '03 tutorial]



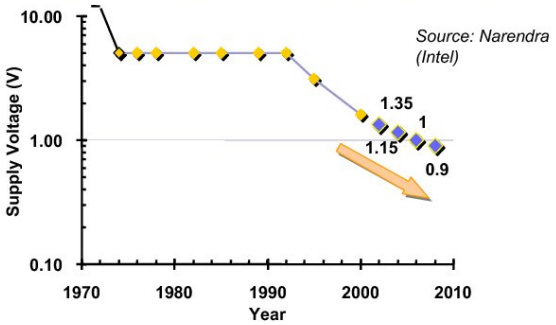
- Power density increasing exponentially
  - Power delivery, packaging, thermal implications
  - Thermal effects on leakage, delay, reliability, etc.

Dynamic Power

$$P_{dyn} \approx kCV^2Af$$

- Aka AC power, switching power
- Static CMOS: current flows when transistors turn on/off
  - Combinational logic evaluates
  - Sequential logic (flip-flop, latch) captures new value (clock edge)
- Terms
  - C: capacitance of circuit (wire length, no. & size of transistors)
  - V: supply voltage
  - A: activity factor
  - f: frequency
- Up to now, Voltage scaling has been saving our bacon!

Vcc will continue to reduce



Only 15% Vcc reduction to meet frequency demand

Leakage Current

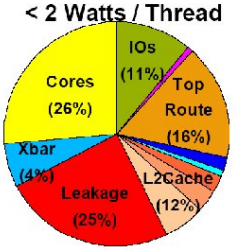
THE PREVIOUS FORMULA ON POWER IGNORES A KEY TERM: POWER WASTED DUE TO LEAKAGE

- Transistors aren't perfect on/off switches
- Even in static CMOS, transistors leak
  - Channel (source/drain) leakage
  - Gate leakage through insulator
    - High-K dielectric replacing SiO_2 will help
- Leakage compounded by
  - Low threshold voltage
    - Low V_{th} => fast switching, more leakage
    - High V_{th} => slow switching, less leakage
  - Higher temperature
    - Temperature increases with power
    - Power increases with C, V^2, A, f
- Rough approximation: leakage proportional to area
  - Transistors aren't free
- Huge problem in current technologies (25% in Niagara and Montecito) and in future technologies: estimates are 40%-50% of total power

Power Consumption Niagara

[Source: J. Laudon]

63W @ 1.2Ghz / 1.2V
< 2 Watts / Thread



- Fully static design
- Fine granularity clock gating for datapaths (30% flops disabled)
- Lower 1.5 P/N width ratio for library cells
- Interconnect wire classes optimized for power x delay
- SRAM activation control

SPARC Cores, L2Data, Crossbar, IO's, Leakage, L2Tag Unit, Global Clock, Floating Point, Interconnect, L2 Buff Unit, Misc Units

Reducing Dynamic Power

- Reduce capacitance
  - Simpler, smaller design ☺
  - Reduce IPC at design (Instructions per Cycle)
- Reduce activity
  - Smarter design
  - Reduce IPC at design (Instructions per Cycle)
- Reduce frequency
  - Often in conjunction with reduced voltage
- Reduce voltage
  - Biggest hammer due to quadratic effect, widely employed
  - Can be static (binning/sorting of parts), and/or
  - Dynamic (power modes)
    - E.g. Transmeta Long Run, AMD PowerNow, Intel Speedstep

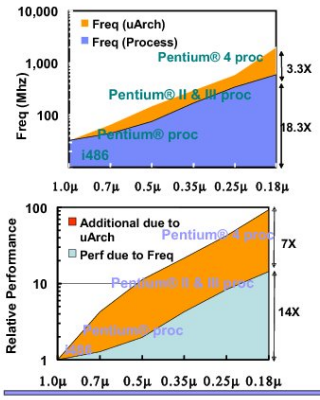
Frequency/Voltage relationship

- Lower voltage implies lower frequency
  - Lower V_{th} increases delay to sense/latch 0/1
- Conversely, higher voltage enables higher frequency
  - Overclocking
- Sorting/binning and setting various Voltages
  - Characterize device, circuit, chip under varying stress conditions
  - Black art – very empirical & closely guarded trade secret
  - Implications on reliability
    - Safety margins, product lifetime
    - This is why *overclocking* is possible

Dynamic Techniques

- BASIC IDEA: turn on/off, part of the chip depending upon activity
- Dynamic Voltage Scaling
- Dynamic Frequency Scaling
- Could be much more aggressive:
  - Dynamic reduction of superscalar width: for a code with low ILP, downsize from a 4 way superscalar down to a 2 way superscalar
  - For applications with low cache usage, turn off part of the caches
  - Turn off speculation

Frequency & Performance



Source: Narendra (Intel)

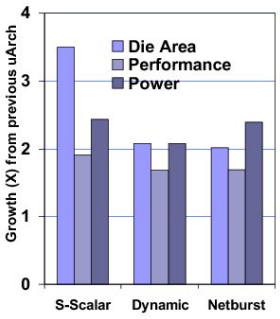
Frequency increased 61X

- **18.3X → process technology**
- **Additional 3.3X → uArch**

Performance increased 100X

- **14X → process technology**
- **Additional 7X → uArch, design**

Design Efficiency—μ Arch



In the same process technology, compare:
Scalar ⇨ Super-scalar
⇨ Dynamic
⇨ Netburst

2-3X Growth in area

~1.4X Growth in Integer Performance

~1.7X Growth in Total Performance

2-2.5X Growth in Power

Source: Narendra (Intel)

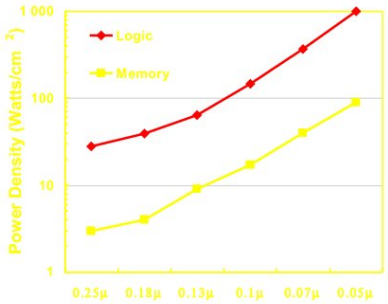
Pollack's Rule in action—Power inefficiency

Can DRAM help?

- Transistor perf not critical for DRAM
- Don't need large retention time
- 10X more storage in same area & power
- TB/sec Bandwidth, at <10ns latency

	SRAM	DRAM
Cell size (f^2)	~ 150	~ 10
Array efficiency	85%	60%
Memory density	1	11

Memory has lower power density



Source: Narendra (Intel)

Exploit memory !

Special-Purpose HW

- Special-purpose performance ? more MIPS/mm²

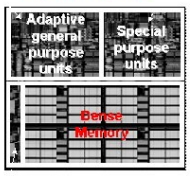
Improve power efficiency with Valued Performance

- SIMD integer and FP instructions in several ISAs
- Multimedia Units :
  - Area: less than 10% of the CPU area (without caches)
  - Power: less than 10% of the CPU power
  - Performance Improvement: 1,5x to 4x
- Integration of other platform components, e.g. memory controller, graphics
- Special-purpose logic, programmable logic, and separately programmable engines

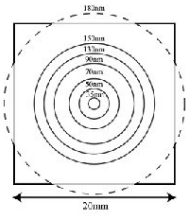
Narendra's View

Present

Next decade



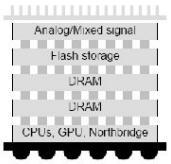
Distance on a chip



Area reachable in one cycle

On chip delays become a problem

3 D Integration



Cooling won't be so easy ☺

CELL Rationale

- **Cell** is a microprocessor architecture jointly developed by Sony Computer Entertainment, Toshiba, and IBM (STI alliance)
- 2001- 2005: 400 million US$ budget
- The first major commercial application of Cell was in Sony's PlayStation 3 game console.
- Mercury offers special boards with Cell
- Toshiba has announced plans to incorporate Cell in high definition television sets.
- The architecture:
  - emphasizes efficiency/watt,
  - prioritizes bandwidth over latency,
  - favors peak computational throughput over simplicity of program code (cf wikipedia)
- CELL has shown excellent performance potential at the expense a large programming effort "Cell is widely regarded as a challenging environment for software development" Wikipedia

CELL evolutions

- By using process scaling, IBM could choose:
  - Smaller
  - Faster
  - Lower Power
- IBM took lower power

CELL Evolutions

Source: © realworldtech

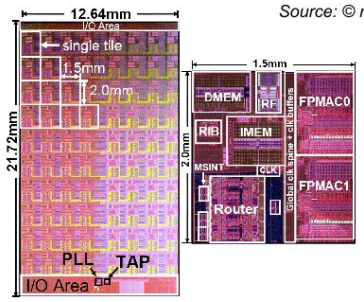
Cell/B.E.					
Generation	W (mm)	H (mm)	Area (mm^2)	Scaling from 90nm	Scaling from 65nm
90nm	19.17	12.29	235.48	100.0%	
65nm	15.59	11.20	174.61	74.2%	100.0%
45nm	12.75	9.06	115.46	49.0%	66.1%
Synergistic Processor Element (SPE)					
Generation	W (mm)	H (mm)	Area (mm^2)	Scaling from 90nm	Scaling from 65nm
90nm	2.54	5.81	14.76	100.0%	
65nm	2.09	5.30	11.08	75.0%	100.0%
45nm	1.59	4.09	6.47	43.9%	58.5%
Power Processor Element (PPE)					
Generation	W (mm)	H (mm)	Area (mm^2)	Scaling from 90nm	Scaling from 65nm
90nm	4.44	6.05	26.86	100.0%	
65nm	3.50	5.60	19.60	73.0%	100.0%
45nm	2.66	4.26	11.32	42.1%	57.7%

Intel's Teraflop Design

- Research project developed by INTEL
- GOAL; investigate how to take advantage of future process scaling and exploit greater parallelism
- Presented at ISSCC 07
- 65 nm process, 100 M transistors
- Grid of simple cores (called tiles)
- High performance network between the tiles: 1.62 Tbit/s: perhaps the most interesting part of the project
- 1 Tflop on a specific application, 0.95 V, 3.16 GHz

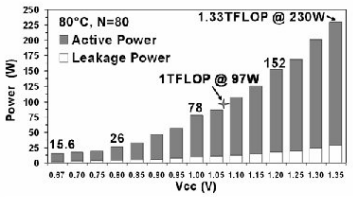
Intel TERAFLOP chip

Source: © realworldtech

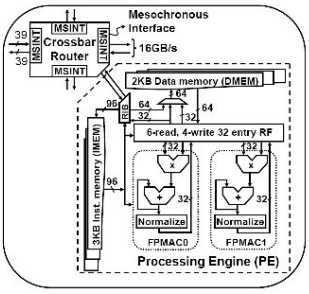


Intel TERAFLOP (power)

Source: © realworldtech



Intel TERAFLOP: tile



Tile Architecture

- Minimal VLIW: 96 bit instruction word encoding up to 8 operations
- No cache (then no cache coherency ☺)
- No Virtual Memory
- SP units

Source: © realworldtech

Intel TERAFLOP: Performance

	TFLOPs	Efficiency
Stencil	1	73.3%
SGEMM	0.51	37.5%
Spreadsheet	0.45	33.0%
2D FFT	0.02	2.7%

Stencil: Heat Equation (80 C, 1.07 V, 4.27 GHz, 97 W)

SGEMM: Single precision matrix multiply

Spreadsheet: financial modeling

2D FFT: 2 dimensional FFT require a lot of communications

COMMUNICATION IS A MAJOR PROBLEM

Trends in microprocessor architecture

- What to do with all of these transistors offered by Moore's Law??
- TAKE INTO ACCOUNT POWER
- NO MAJOR PUSH ON UNICORE PERFORMANCE EXCEPT VIA MULTIMEDIA/VECTOR UNITS:
  - out of order execution has shown its limit (between 6 to 8 instruction simultaneously issued). Lack of ILP limits ROI.
  - Improving branch prediction
  - Increase very moderately pipeline depths (a few extra stages)
  - Reinforce multimedia instruction sets (SSE, Altivec, VIS) and increase "vector" length

Trends in microprocessor architecture (2)

- LARGER AND LARGER CACHES: more and more complex memory hierarchies (L3 sizes between 16 and 32 MB)
- Embedded DRAM
- More cores, three avenues:
  - A few complex cores
  - A larger number of simpler cores
  - Heterogeneous multicore: specialized cryptographic cores, general purpose cores, graphic unit
  - TAKE INTO ACCOUNT COMMUNICATION AND COHERENCY (drop hardware coherency ??)

Software

- NIKLAUS WIRTH's LAW: " Software gets faster slower than hardware gets faster"
- Productivity of software developers does not increase exponentially ☺

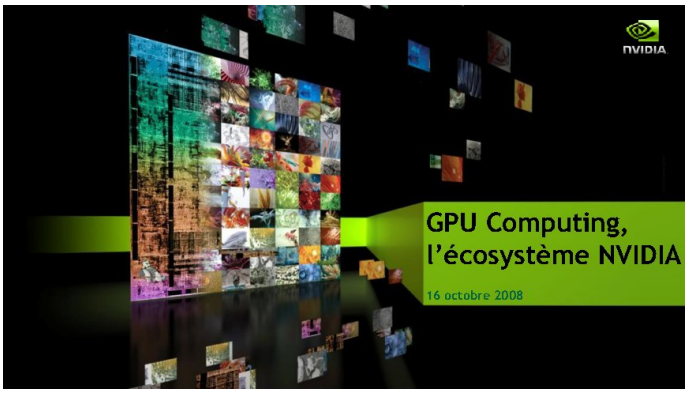
Acknowledgements

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)
  - Mikko Lipasti (UWisc)
  - INTEL
  - IC Knowledge LLC

2.3 Utiliser le GPU comme coprocesseur de calcul, l'écosystème NVIDIA

Jean-Christophe Baratault NVIDIA

L'architecture des GPU est massivement parallèle depuis plus de 10 ans, ce pour traiter des données graphiques 3D photo réalistes en temps réel . Pressentant l'émergence de solutions de calcul hybrides multi-cœur en environnement hétérogène, NVIDIA a massivement investi depuis 4 ans pour offrir un environnement complet de développement et de déploiement axé sur le traitement des données non-graphiques par le GPU. La présentation détaille la stratégie de NVIDIA en ce domaine ainsi que l'écosystème mis en œuvre pour répondre aux attentes des clients industriels.



Why didn't GPU Computing took off sooner?

- GPU Architecture
  - Gaming oriented, process pixel for display
  - Single threaded operations
  - No memory cache
- Development Tools
  - Graphics oriented (OpenGL, GLSL)
  - University research (Brook)
  - Assembly language
  - No profiler/debugger
- Deployment
  - Gaming solutions with limited lifetime
  - Expensive OpenGL professional graphics boards
  - No HPC compatible products

2

Disruptive Technology?

NO
- Transparent to the user
- 'Natural' game/consumer app evolution

YES
- Engineer & Decision Maker MUST think parallel
- A 'no limit' new path
- Not anymore a CPU vendors-only world

MAYBE
- Can I rely on a "pixel processing" company?
- Am I going to be locked to NVIDIA?

3

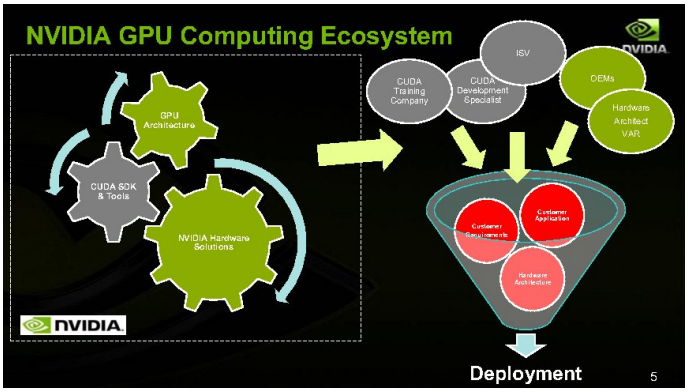
NVIDIA invested in GPU Computing back in 2004

- Strategic move for the company
  - Expand GPU architecture beyond pixel processing
  - Future platforms will be hybrid, multi/many cores based
- Hired key industry experts
  - x86 architecture
  - x86 compiler
  - HPC hardware specialist

→ Provide a GPU based Compute Ecosystem by 2008

4

NVIDIA GPU Computing Ecosystem

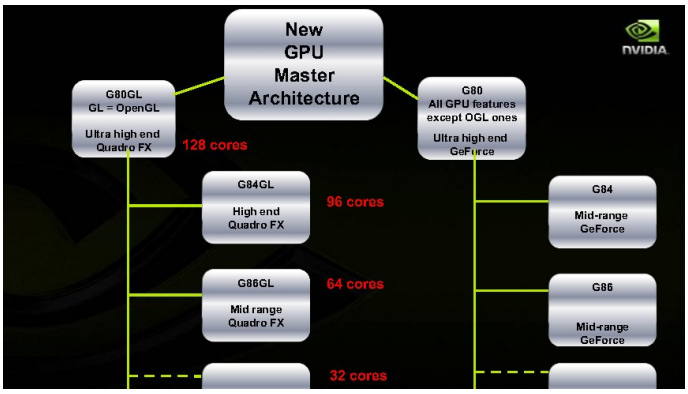


5

The Past 2 years

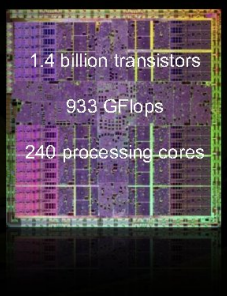
- 2006
  - G80, first GPU with built-in Compute features
    - 128 core, scalable multi-threaded architecture
  - CUDA SDK Beta
- 2007
  - Tesla product line
  - CUDA SDK 1.0, 1.1
  - University trainings programs

6



June 2008: NVIDIA GT200 GPU

2nd Generation Parallel Computing Architecture



NVIDIA GPU marketing names

- GT200 — Consumer GeForce
- GT200GL — Professional Quadro
- T10P — HPC Tesla

8

Parallelism is Scaling Rapidly

- CPUs and GPUs are parallel processors
  - CPUs now have 2, 4, 8, … processors
  - GPUs now have 32, 64, 128, 240, … processors
- Parallelism is increasing rapidly with Moore's Law
  - Processor count is doubling every 18 – 24 months
  - Individual processor cores no longer getting faster
- Challenge: Develop parallel application software
  - Scale software parallelism to use more and more processors
  - Same source for parallel GPUs and CPUs

9

CUDA is C for Parallel Processors

- CUDA is industry-standard C
  - Write a program for one thread
  - Instantiate it on many parallel threads
  - Familiar programming model and language
- CUDA is a scalable parallel programming model
  - Program runs on any number of processors without recompiling
- CUDA parallelism applies to both CPUs and GPUs
  - Compile the same program source to run on different platforms with widely different parallelism
  - Map to CUDA threads to GPU threads or to CPU vectors

10

CUDA Zone: www.nvidia.com/cuda

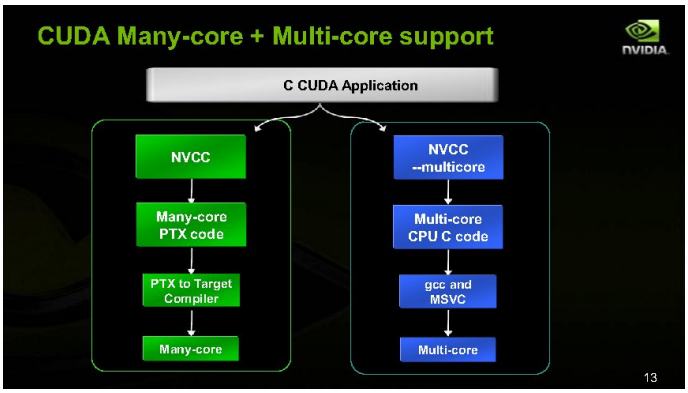
- Resources, examples, and pointers for CUDA developers

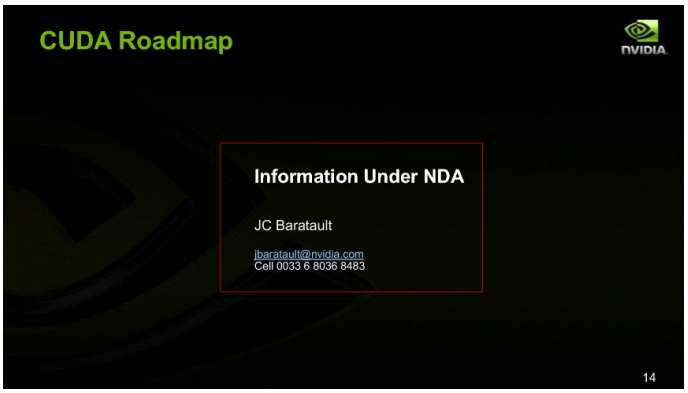
11

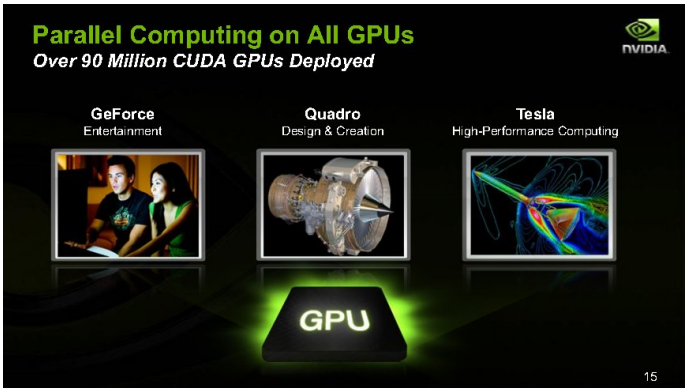
What's Next for CUDA

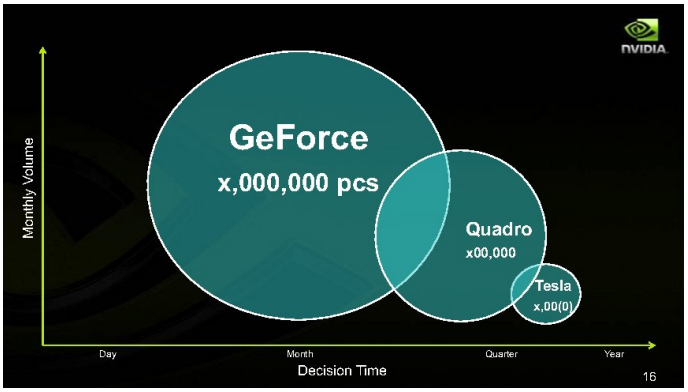
Fortran	C++	Multiple GPUs
Debugger	Profiler	Linker

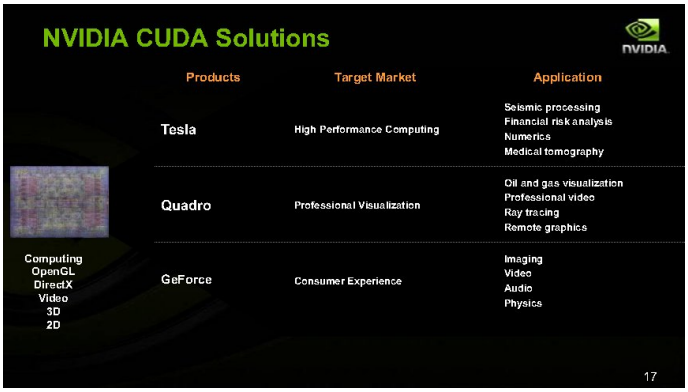
12

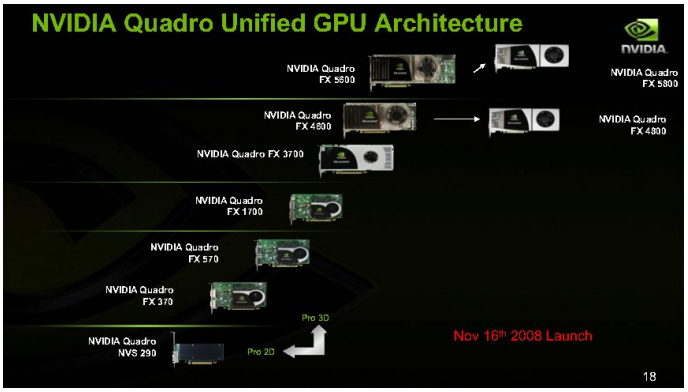

CUDA Many-core + Multi-core support
C CUDA Application
NVCC
Many-core PTX code
PTX to Target Compiler
Many-core
NVCC --multicore
Multi-core CPU C code
gcc and MSVC
Multi-core
13

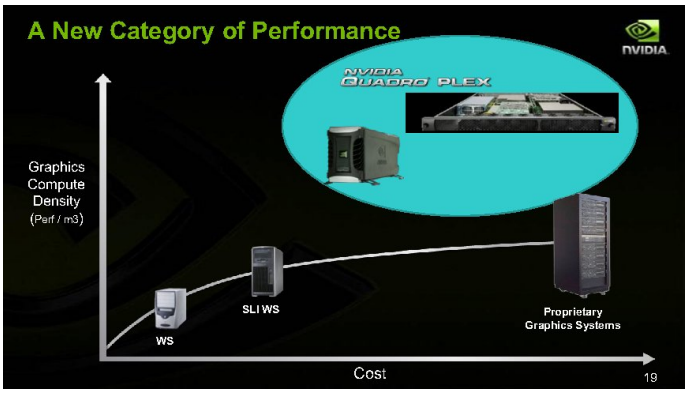

CUDA Roadmap
Information Under NDA
JC Baratault
jbaratault@nvidia.com
Cell 0033 6 8036 8483
14

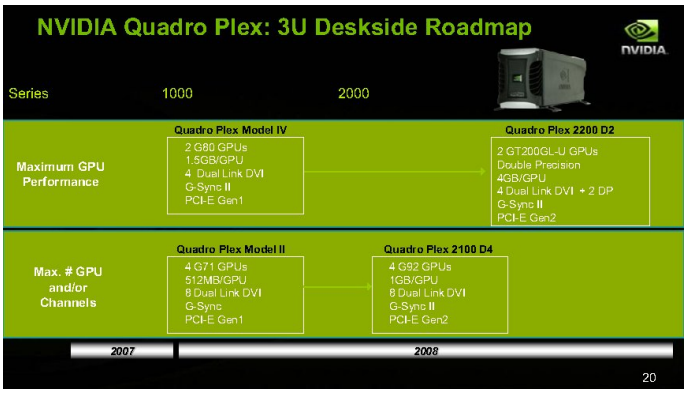

Parallel Computing on All GPUs
Over 90 Million CUDA GPUs Deployed
GeForce
Entertainment
Quadro
Design & Creation
Tesla
High-Performance Computing
GPU
15

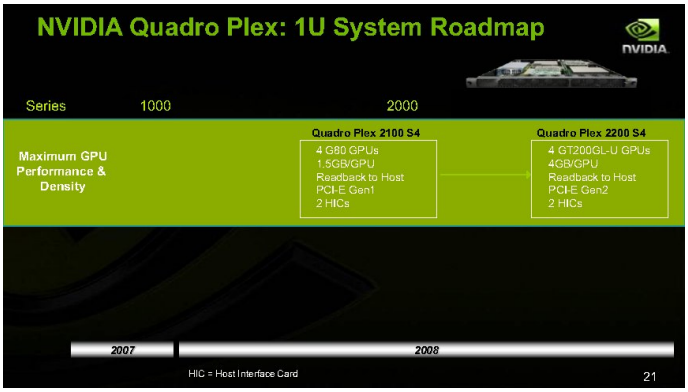

Monthly Volume
GeForce
x,000,000 pcs
Quadro
x00,000
Tesla
x,00(0)
Day
Month
Quarter
Year
Decision Time
16

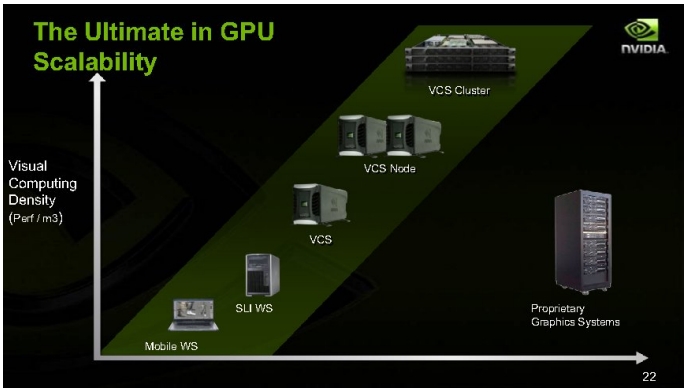

NVIDIA CUDA Solutions
Products
Target Market
Application
Tesla
High Performance Computing
Seismic processing
Financial risk analysis
Numerics
Medical tomography
Quadro
Professional Visualization
Oil and gas visualization
Professional video
Ray tracing
Remote graphics
GeForce
Consumer Experience
Imaging
Video
Audio
Physics
Computing
OpenGL
DirectX
Video
3D
2D
17


NVIDIA Quadro Unified GPU Architecture
NVIDIA Quadro FX 5800
NVIDIA Quadro FX 4800
NVIDIA Quadro FX 3700
NVIDIA Quadro FX 1700
NVIDIA Quadro FX 570
NVIDIA Quadro FX 370
NVIDIA Quadro NVS 290
Pro 3D
Pro 2D
Nov 16th 2008 Launch
18

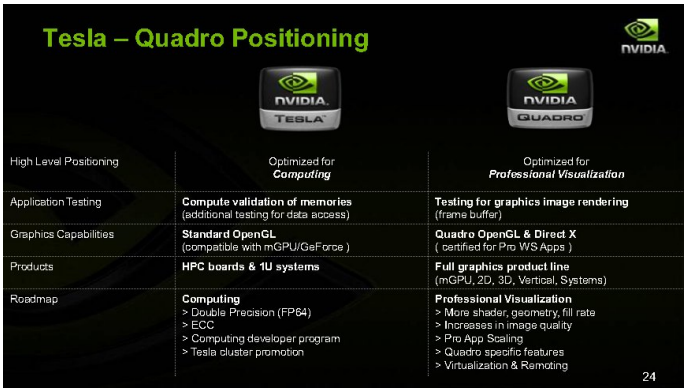

A New Category of Performance
NVIDIA
NVIDIA QUADRO PLEX
Graphics Compute Density (Perf / m3)
WS
SLI WS
Proprietary Graphics Systems
Cost
19


NVIDIA Quadro Plex: 3U Deskside Roadmap
NVIDIA
Series
1000
2000
Maximum GPU Performance
Quadro Plex Model IV
2 G80 GPUs
1.5GB/GPU
4 Dual Link DVI
G-Sync II
PCI-E Gen1
Quadro Plex 2200 D2
2 GT200GL-U GPUs
Double Precision
4GB/GPU
4 Dual Link DVI + 2 DP
G-Sync II
PCI-E Gen2
Max. # GPU and/or Channels
Quadro Plex Model II
4 G71 GPUs
512MB/GPU
8 Dual Link DVI
G-Sync
PCI-E Gen1
Quadro Plex 2100 D4
4 G92 GPUs
1GB/GPU
8 Dual Link DVI
G-Sync II
PCI-E Gen2
2007
2008
20


NVIDIA Quadro Plex: 1U System Roadmap
NVIDIA
Series
1000
2000
Maximum GPU Performance & Density
Quadro Plex 2100 S4
4 G80 GPUs
1.5GB/GPU
Readback to Host
PCI-E Gen1
2 HICs
Quadro Plex 2200 S4
4 GT200GL-U GPUs
4GB/GPU
Readback to Host
PCI-E Gen2
2 HICs
2007
2008
HIC = Host Interface Card
21


The Ultimate in GPU Scalability
NVIDIA
VCS Cluster
VCS Node
VCS
SLI WS
Mobile WS
Proprietary Graphics Systems
Visual Computing Density (Perf / m3)
22


TESLA
C1060
Card
S1070
1U Compute System


Tesla – Quadro Positioning
NVIDIA
NVIDIA TESLA
NVIDIA QUADRO
High Level Positioning
Optimized for Computing
Optimized for Professional Visualization
Application Testing
Compute validation of memories (additional testing for data access)
Testing for graphics image rendering (frame buffer)
Graphics Capabilities
Standard OpenGL (compatible with mGPU/GeForce)
Quadro OpenGL & Direct X (certified for Pro WS Apps)
Products
HPC boards & 1U systems
Full graphics product line (mGPU, 2D, 3D, Vertical, Systems)
Roadmap
Computing
> Double Precision (FP64)
> ECC
> Computing developer program
> Tesla cluster promotion
Professional Visualization
> More shader, geometry, fill rate
> Increases in image quality
> Pro App Scaling
> Quadro specific features
> Virtualization & Remoting
24

NVIDIA GPU Brand Feature Comparison

	NVIDIA Tesla	NVIDIA Quadro	NVIDIA GeForce
GPU Designed and Mfg by	NVIDIA	NVIDIA	NVIDIA
Product Engineered By	NVIDIA	NVIDIA	Add In Card maker (AIC)
Components Selected and Sourced by	NVIDIA	NVIDIA	AIC
ECO Control	NVIDIA	NVIDIA	AIC
Quality Testing	Compute and Memory	Professional Graphics	Consumer Graphics
Form Factors	Card and 1U	Card, Deskside and 1U	Card
Roadmap	High Performance Computing	Professional Graphics (Open GL & DirectX Applications)	Consumer (Gaming) (DirectX Games)
Operating Specifications	Corporate Compute Environment	Professional Workstation, Thin Client (passive)	Consumer (Gaming)
Supported Provided By	NVIDIA	NVIDIA	AIC
Max Data Readback	3 GB/s (CUDA)	3 GB/s (OGL, DX)	1 GB/s
Max Frame Buffer/GPU (On Board Memory)	4 GB	4 GB	1 GB
Lifecycle	24 months Managed by NVIDIA	24-36 months Managed by NVIDIA	9-12 month Varies by AIC manufacturer

25

Tesla C1060 Computing Card

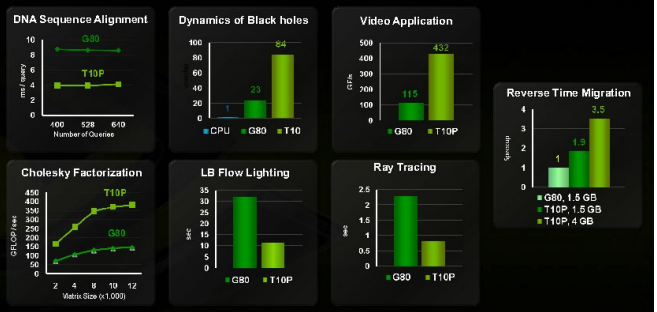
Processor	1 x Tesla T10P
Number of cores	240
Core Clock	1.29 GHz
On-board memory	4.0 GB
Memory bandwidth	102 GB/sec peak
Memory I/O	512-bit, 800MHz GDDR3
Form factor	Full ATX: 4.736" x 10.5" Dual slot wide
System I/O	PCIe x16 Gen2
Typical power	160 W

26

	Tesla 8-series C870 card	Tesla 10-series C1060 card
Number of Cores	128	240
32-bit FP Performance	0.5 Teraflop	1 Teraflop
On-board Memory	1.5 GB	4.0 GB
Memory interface	384-bit GDDR3	512-bit GDDR3
Memory I/O bandwidth	77 GBytes/sec	102 GBytes/sec
System interface	PCIe x16 Gen1	PCIe x16 Gen2

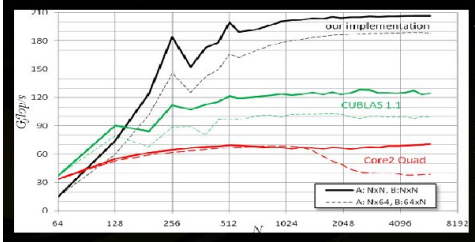
27

Doubling Performance With The New GPU



28

New CUBLAS library SGEMM achieves over 200 GFLOPS



Tesla C870 card (G80 GPU)

150 GFLOPS starting with 256x256

Over 200 GFLOPS for square 1024x1024 and larger matrices

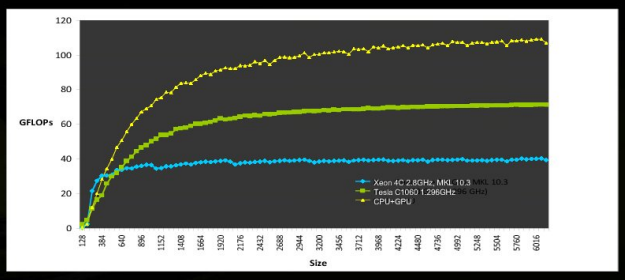
Performance computation did not include data transfers between CPU and GPU across PCIe (inputs and output were in GPU memory)

Core2 numbers were measured using MKL 10.0

http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-49.pdf

29

DGEMM Performance



30

Introducing the *Supercomputing PC*

960-core Parallel Supercomputer
- Up to 4 TFlops
- Performance of a large cluster
- 100x the compute performance of a PC

CUDA Architecture
- Emerging standard for GPU parallel computing
- Thousands of application developers

Accessible to Anyone
- Supercomputing at your desk
- Windows or Linux
- Plugs into standard power strip

31

Scalable Professional Development Platforms

32

Key Message

- Powerful GPU computing deskside supercomputer
- Ease of deployment
- Cluster in a box at your deskside
- Configurable for ideal mix of Quadro + Tesla

33

Marketing activities

- **Launch Nov 17 @ Supercomputing'08 – Austin, TX**
  - Press Event
  - OEM & partner announcements
  - Message "Supercomputing PC"
  - "What will you Discover?" contest & lead generation
- **"MY Supercomputer" Campaign**
  - Reaching scientists and engineers at top universities worldwide
  - Life, Physical, Earth Sciences
- **Supercomputing PCs for Sale Worldwide on Launch Day**

34

Tesla C1060 Computing Card

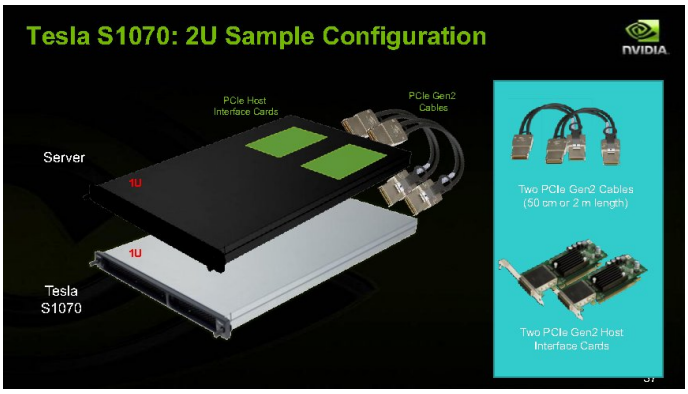
Processor	1 x Tesla T10P
Number of cores	240
Core Clock	1.29 GHz
On-board memory	4.0 GB
Memory bandwidth	102 GB/sec peak
Memory I/O	512-bit, 800MHz GDDR3
Form factor	Full ATX: 4.736" x 10.5" Dual slot wide
System I/O	PCIe x16 Gen2
Power	200 W maximum 160 W typical (5.83 GFlops/Watt) 25 W idle

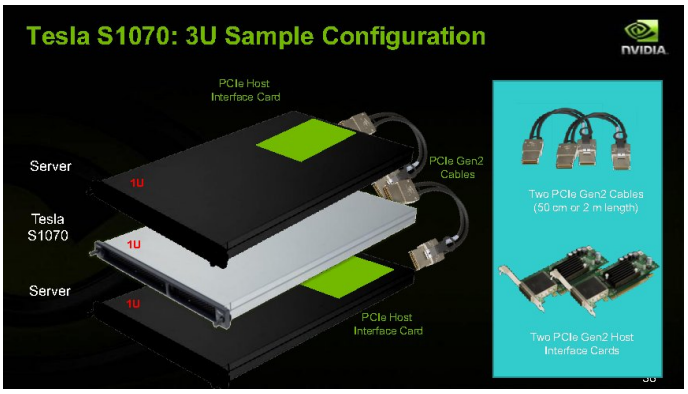
Tesla S1070 1U System

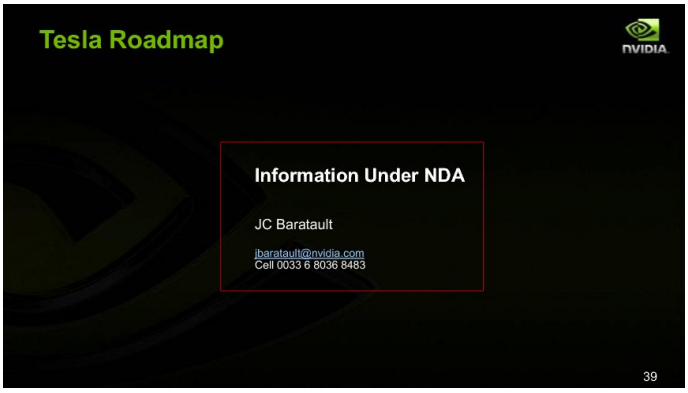
Processors	4 x Tesla T10P
Number of cores	960
Core Clock	1.5 GHz
Performance	4 Teraflops
Total system memory	16.0 GB (4.0 GB per T10P)
Memory bandwidth	408 GB/sec peak (102 GB/sec per T10P)
Memory I/O	2048-bit 800MHz GDDR3 (512-bit per T10P)
Form factor	1U (EIA 19" rack)
System I/O	2 PCIe x16 Gen2
Typical power	700 W

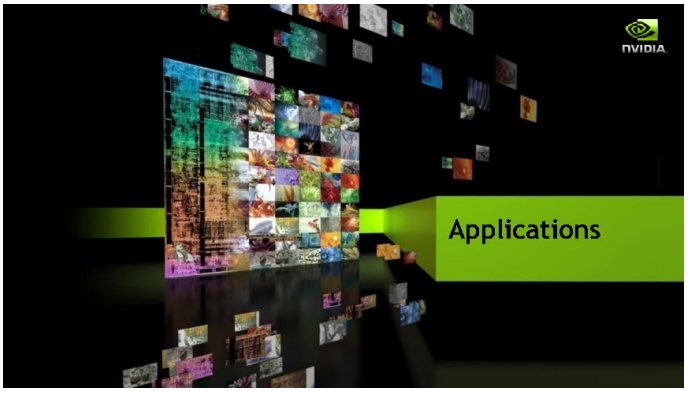
Connection to host system(s) using two PCIe interface cards

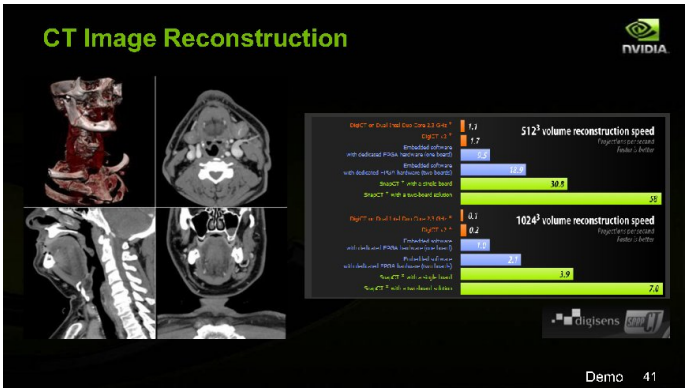
36

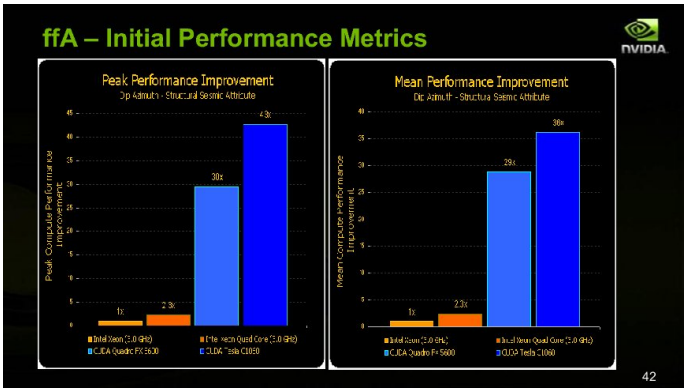

Tesla S1070: 2U Sample Configuration
PCIe Host Interface Cards
PCIe Gen2 Cables
Server
1U
1U
Tesla S1070
Two PCIe Gen2 Cables (50 cm or 2 m length)
Two PCIe Gen2 Host Interface Cards

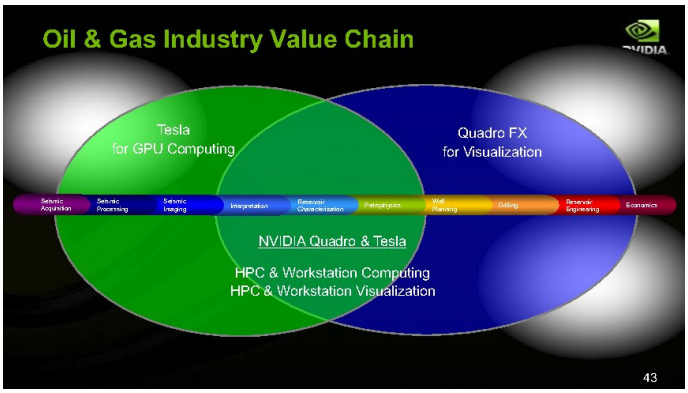

Tesla S1070: 3U Sample Configuration
PCIe Host Interface Card
Server
PCIe Gen2 Cables
Tesla S1070
1U
1U
Server
1U
PCIe Host Interface Card
Two PCIe Gen2 Cables (50 cm or 2 m length)
Two PCIe Gen2 Host Interface Cards


Tesla Roadmap
Information Under NDA
JC Baratault
jbaratault@nvidia.com
Cell 0033 6 8036 8483
39

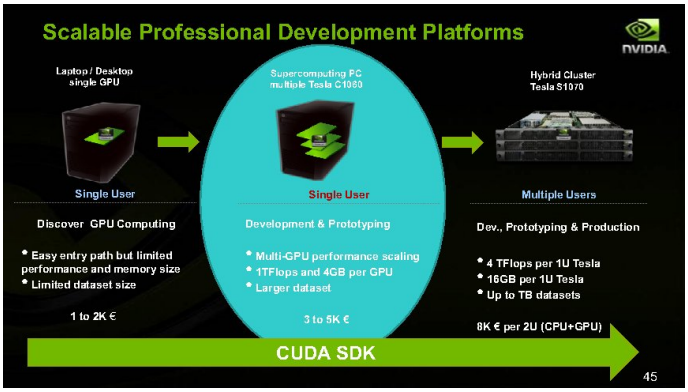

Applications

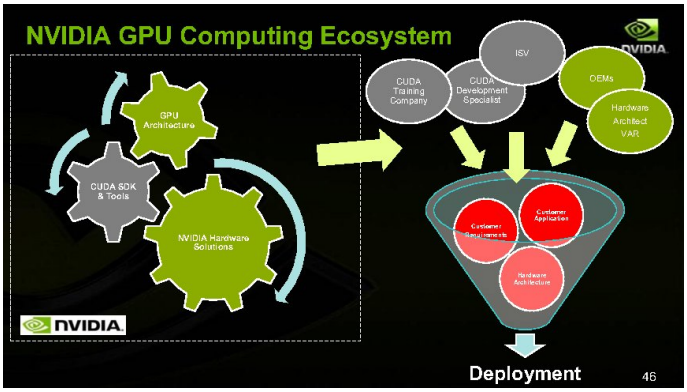

CT Image Reconstruction
512³ volume reconstruction speed
1024³ volume reconstruction speed
digisens
Demo 41


ffA – Initial Performance Metrics
Peak Performance Improvement
Mean Performance Improvement
42


Oil & Gas Industry Value Chain
Tesla
for GPU Computing
Quadro FX
for Visualization
Seismic Acquisition
Seismic Processing
Seismic Imaging
Interpretation
Reservoir Characterization
Petrophysics
Well Planning
Drilling
Reservoir Engineering
Economics
NVIDIA Quadro & Tesla
HPC & Workstation Computing
HPC & Workstation Visualization
43


Super Computing 2007
Editors' Choice Award: Top 5 Companies to Watch in 2008: NVIDIA
Editors' Choice Award for Most Significant New HPC Hardware Product: NVIDIA Tesla
Readers' Choice Award for Most Significant New HPC Software Product: NVIDIA CUDA
"Best of What's New" Award: NVIDIA CUDA
44


Scalable Professional Development Platforms
Laptop / Desktop single GPU
Supercomputing PC multiple Tesla C1060
Hybrid Cluster Tesla S1070
Single User
Single User
Multiple Users
Discover GPU Computing
Development & Prototyping
Dev., Prototyping & Production
• Easy entry path but limited performance and memory size
• Limited dataset size
1 to 2K €
• Multi-GPU performance scaling
• 1TFlops and 4GB per GPU
• Larger dataset
3 to 5K €
• 4 TFlops per 1U Tesla
• 16GB per 1U Tesla
• Up to TB datasets
8K € per 2U (CPU+GPU)
CUDA SDK
45


NVIDIA GPU Computing Ecosystem
GPU Architecture
CUDA SDK & Tools
NVIDIA Hardware Solutions
ISV
OEMs
CUDA Training Company
CUDA Development Specialist
Hardware Architect VAR
Customer Applications
Deployment
46


NVIDIA

2.4 Hybrid Computing

Bruno Stefanizzi AMD/ATI

Parallel HPC applications benefit from multi core CPU technology and have been able to multiply the computation density by a factor of 2 to 4 and later by 8. This improvement is not enough compared to the computation requirements of today's applications. This is why people have been looking for new hardware and specialized processors which could give applications gains from 20 up to 100.

Specialized processors like GPUs have improved performance at a greater pace than Moore law predicts. They started 10 years ago with a technology using 350nm, 5 million transistors at 75Mhz and now are using 55nm, 700Millions transistors at 800Mhz being able to deliver 512GFlops or more than 3.5GFlops/Watt.

This leads to improvements factors of 1.7x/year in transistors count, 1.3x/year in clock speed, 2.0x/year in processing units and 1.3x/year in memory bandwidth. Using such powerful dedicated processors as well as CPU in a highly parallel environment of Multi-core for both is showing the requirement to be able to use in the most efficient way this heterogeneous environment of Hybrid computing.

This hardware environment exists and can be used today. The first challenge is on the software development side. Development tools need to integrate heterogeneous programming as well as multi core from the core of their language being able to support code generation on different processors types as well as handling asynchronous behaviors. This comes with compilers and libraries supporting this and being design or extended for it. Obviously those tools need to support multiple hardware platforms to lead to some standards.

The second key challenge change is the evolution of buses and bandwidth linking together the different cores of CPU and GPUs. And the way they talk to each other. Fusion projects will address those evolutions in the future by defining new architectures around those processors to improve the data flow between them which will be the key to use all the power available. Cross bar memory controllers will allow GPUs to talk each other very quickly without breaking parallelism. Hyper Transport bus will improve communication between GPUs and CPUs. Finally Multi core GPUs and CPUS on the same die will increase even more the compute density.

Different benchmarks and application codes have been used to demonstrate already the benefits so such architecture. We will present SGEMM results as well as different algorithms.

The results will highlight the fact that performance is affected by in/out copy of the data on the GPU at the moment and that finer tunings allows huge jump in performance. We will also show that changing the way algorithms have been implemented for CPU to fit GPU architecture adds even more performance gains.

the future is

fusion

Hybrid Computing

Bruno Stefanizzi
AMD

Hybrid computing - Why

Today's HPC applications requirements

- More computing power (50x, 100x)
- Reduced cost, power and space requirements
- Growth more than Moore's law

CPU performance improvement only seems not enough

Hybrid Computing - Why

Specialized processors are one interesting option

- FPGA
  - Too specific, don't solve common issues
- CELL
  - Interesting to experiment but too difficult to program
  - No strong roadmap, not really a mass production chip
- GPU
  - GPU are faster and faster
  - GPU are more and more programmable
  - GPU are available everywhere, used by everyone, and cheap with a strong technical and business roadmap

Hybrid Computing - Why

Let's compare CPU and GPU

-Trends

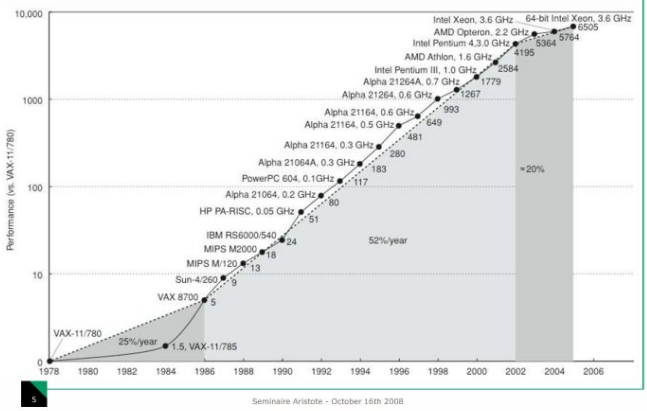
-Features

See if it makes sense to only use one type of processor (GPU only?)

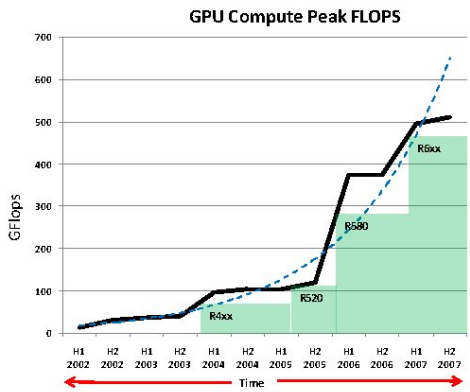
Let's find out how do we use them in real apps and what are the limitations

What solutions do we bring?

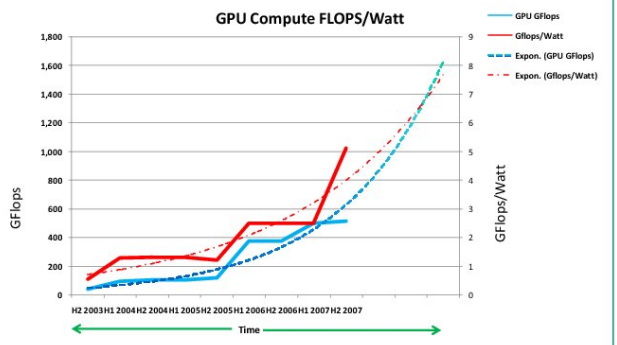
Hybrid computing – Why CPU Performance Trends



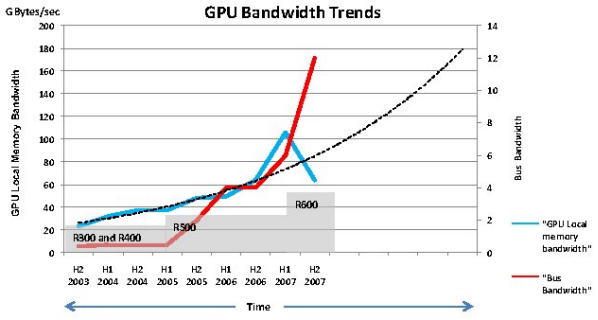
Hybrid computing – Why GPU Performance Trends



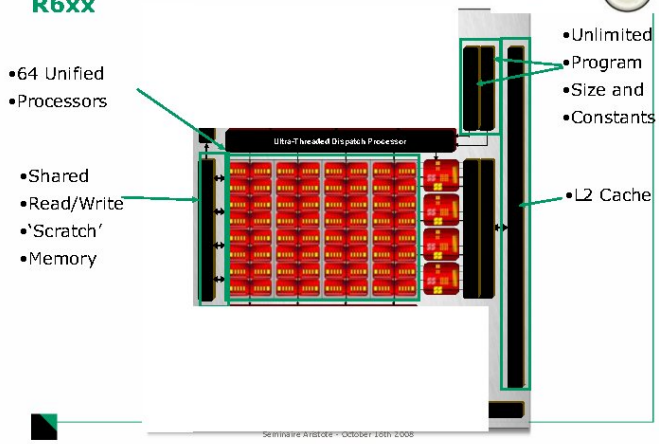
Hybrid computing – Why GPU Performance Trends



Hybrid computing – Why GPU Performance Trends



Hybrid Computing – Why R6xx



Hybrid computing – Why CPU vs GPU

	GPU	CPU
Program Style	Few instructions, lots of data	Lots of instructions, little data
Control Flow	SIMD Hardware threading	Out of order execution Branch prediction
Access Patterns	Little reuse	Reuse + locality
Program Model	Data parallel	Task parallel
Synchronization	Very simple sync	Complex sync
ISA	Proprietary	Standardized
Legacy Support	Not necessarily	Backwards compatible
Functional Deltas	Large and frequent	Small and infrequent

Hybrid computing – Why CPU vs GPU

	GPU	CPU
# Processors	64+	4
ALU area	~40% of die	~5% of die
Memory System	Max bandwidth (10x)	Min latency (0.1x)
Memory Access	Complex (tiling + arithmetic in memory)	Simple LD/ST
Cache	Small cache	Large cache (10x)
FP Compliance	Partial IEEE DP/SP	Full IEEE 754 DP/SP

Hybrid computing - Why

GPU

- High Arithmetic Intensity is desirable because
  - Modern computing systems are limited by communication bandwidth rather than arithmetic
  - GPUs are great at arithmetic operations due to high number of ALU units
  - Lots of on-chip logic goes into hiding the latency incurred due to memory operations

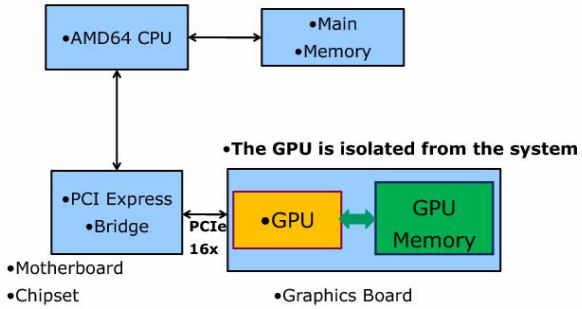
CPU

- Provides flexibility
- Very efficient with low compute data parts of algorithms

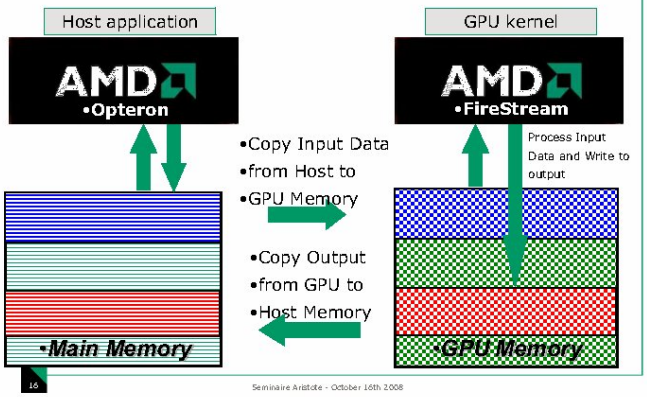
We need both obviously!

How to program them easily in a real industrial development environment?

Hybrid Computing – Why Current Typical PC/Server Architect



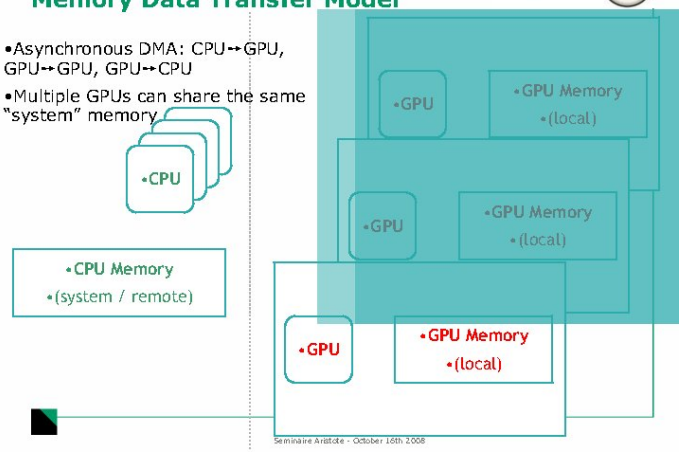
Hybrid computing – Why Typical naïve execution



Hybrid Computing – Why Memory Data Transfer Model

•Asynchronous DMA: CPU↔GPU, GPU↔GPU, GPU↔CPU

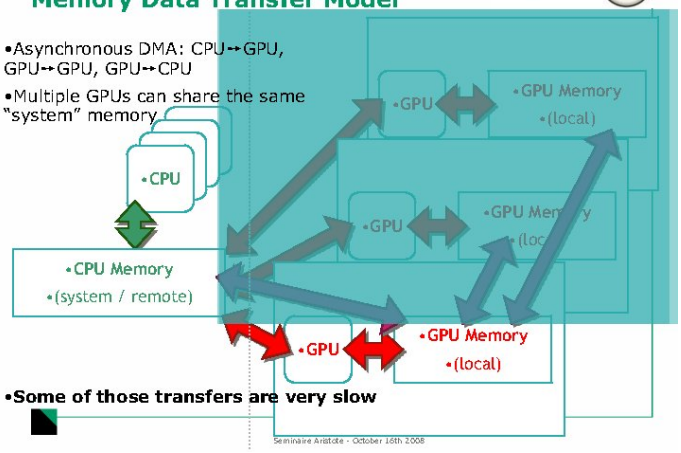
•Multiple GPUs can share the same "system" memory



Hybrid Computing – Why Memory Data Transfer Model

•Asynchronous DMA: CPU↔GPU, GPU↔GPU, GPU↔CPU

•Multiple GPUs can share the same "system" memory



•**Some of those transfers are very slow**

Hybrid Computing – How to get the best of it

CPUs are good in some areas and required for many tasks

GPUs are good in others areas but **alone** are not enough

In order to use them efficiently we need:

- New Software Ecosystem –
  - To program easily the heterogeneous system
- New Specific Hardware architecture
  - To get best memory bandwidth to all the different processors

Hybrid Computing – New Software Ecosystem

Software Ecosystem needs to provide

- **Compilers** for a new Programming Model
  - Provide some standards languages with compilers
  - Manage Asynchronous computing model
  - Data locality and asynchronous transfers
  - Manage heterogeneous Instruction Set Architectures
  - **Multiplatform (multi OS, multi HW vendors)**
  - Not expose an architecture specific programming model but **a general one**
- **Libraries** (Math, domain specialized, open source, etc..)
- **Tools** (Debuggers, Profilers, etc..)

We need an Open Industry Standard Ecosystem

AMD Stream 2.0

Hybrid Computing – New Open Indu Standard Ecosystem

AMD Stream 2.0

Compilers

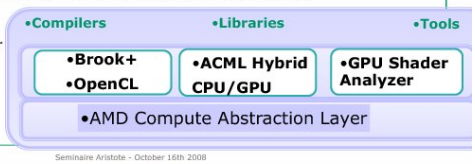
- Brook+
  - High-level language, C extensions for the GPU
  - Based on Brook from Stanford; AMD enhancements will be open-sourced
  - Future options OpenMP Ext, gcc GNU compilers, Microsoft Visual Studio, CAPS
- OpenCL
  - Common Language for Stream and hybrid architectures

Libraries

- AMD's math library ACML provides GPU-accelerated math functions

Tools

- GPU Shader Analyzer
- AMD Code Analyst



21 Seminaire Aristote - October 16th 2008

Hybrid Computing Brook+ Architecture

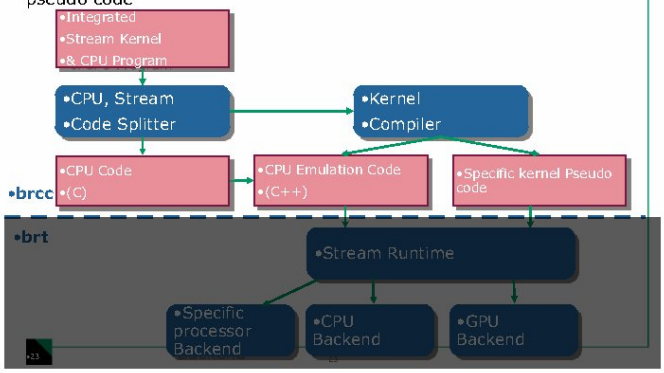
Consists of two components

- Brook+ Compiler, `brcc`
  - Source-to-source compiler
  - Relies on third party tools for target code generation and optimizations
  - Based on the cTool **Open-source** C parser
  - Generates C++ code to be used by the runtime
  - CAL-specific code generation added by AMD
- Brook+ Runtime, BRT
  - **Supports multiple runtime *backends***
  - Provides **common interface** for each backend supported by brcc
  - Originally supported DirectX9, OpenGL and CPU backends
  - CAL backend added by AMD
  - Cell and other backends exists or will soon

22 Seminaire Aristote - October 16th 2008

Hybrid Computing Brook+ Compiler

Converts Brook+ files into C++ code. Kernels, written in C, are compiled to AMD's IL code for the GPU or C code for the CPU or any pseudo code



23 Seminaire Aristote - October 16th 2008

Hybrid Computing OpenCL – The New Standard

Customer demand for vendor-neutral open programming interface vs. vendor-proprietary languages (such as CUDA)

- Prevents users from being held captive by a single hardware company
- Provides longevity and portability of code porting effort

Increases the tool and library options for the customer

- More companies and groups willing to port to a vendor-neutral interface
- Porting results will no longer be locked to a single vendor

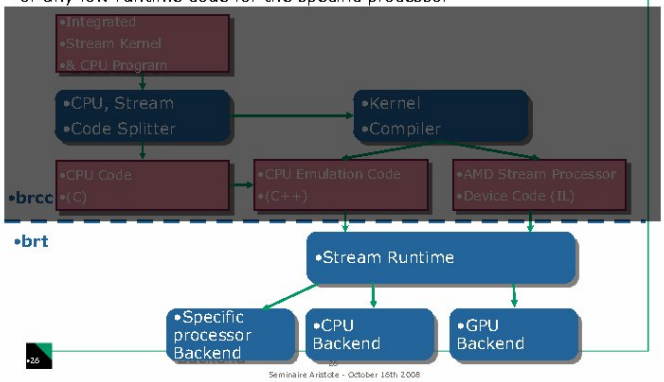
AMD pushes OpenCL to help new Hybrid architectures

Specification and first implementations soon!

24 Seminaire Aristote - October 16th 2008

Hybrid Computing Brook+ Runtime

Pseudo code is executed on the GPU. The backend is written in CAL or any low runtime code for the specific processor



26 Seminaire Aristote - October 16th 2008

Hybrid Computing ACML Hybrid Version

AMD core Math library - Suite of highly tuned math functions for high performance computing

BLAS – Basic Linear Algebra Subprograms
- Full Level 1, 2, and 3 support
- Highly optimized DGEMM, other Level 3 BLAS.
- OpenMP support for key routines

Lapack – Linear Algebra package
- Uses calls to BLAS to solve linear algebra systems
- Matrix factorization/solve, eigenvalue solutions
- OpenMP support for key routines

FFTs – Fast Fourier Transforms
- Time-to-frequency domain
- Hand-tuned assembly
- OpenMP support for 2D, 3D transforms

Fast/vector transcendental math library
- 1, 2, 4, or N values per call
- Single, Double precision

RNGs -Random Number Generators
- Comprehensive reference implementation

Double, Single, Single Complex, Double Complex

27 Seminaire Aristote - October 16th 2008

Hybrid Computing ACML Hybrid Version

Selected routines ported to GPU will run on it if present

BLAS library
- DGEMM, SGEMM
- Will split work between GPU, CPUs
- Supports AMD Barcelona

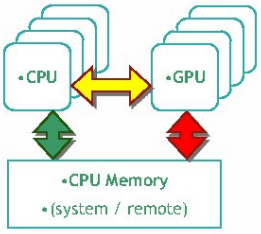
Windows 64, PGI 7.1 compiler
- compatible with Visual Studio 2005
- Linux
- GCC/GFORTRAN 4.2, PGI
- ZGEMM, CGEMM
- L1, L2 routines

FFTs
- 1D CFFT and ZFFT
- 2D,3D
- real-to-complex
- non power-of-2 radices

RNG and select LAPACK functions if they improve performance

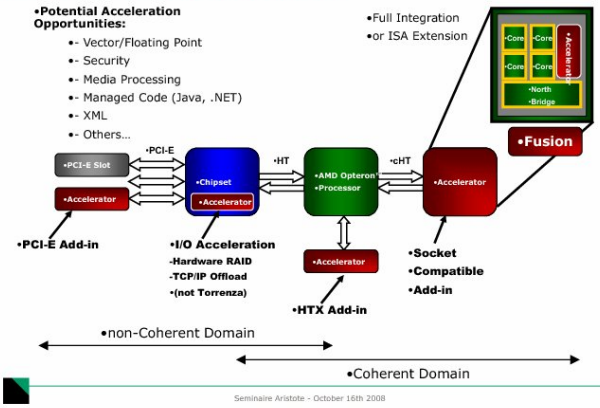
Hybrid Computing – New Hardware Architecture

Hardware Architecture needs to provide
- Reduce power consumption with Tied integration
- High Bandwidth between the different processors types
- High Bandwidth with system memory for any processor



Hybrid Computing - AMD's Accelerat Computing

Accelerator Integration Options



Hybrid Computing -AMD FireStream 9170 Stream Processor

- First GPU with **double-precision floating point**
- 320 Stream cores
- Up to 500 GFLOPS single precision peak performance
- Up to 102 GFLOPS double-precision performance
- First GPU in **55nm process technology**
- <100W power consumption with over 5 GFLOPS/Watt!
- **2GB** on-board GDDR3 memory
- Supported in variety of server and desktop systems

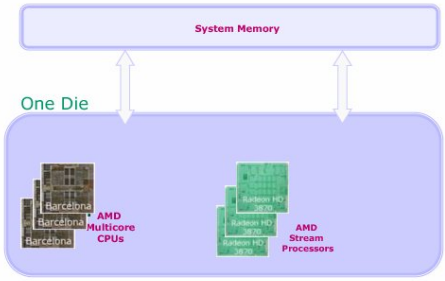
- **Future is not just growing the GPU computation power**

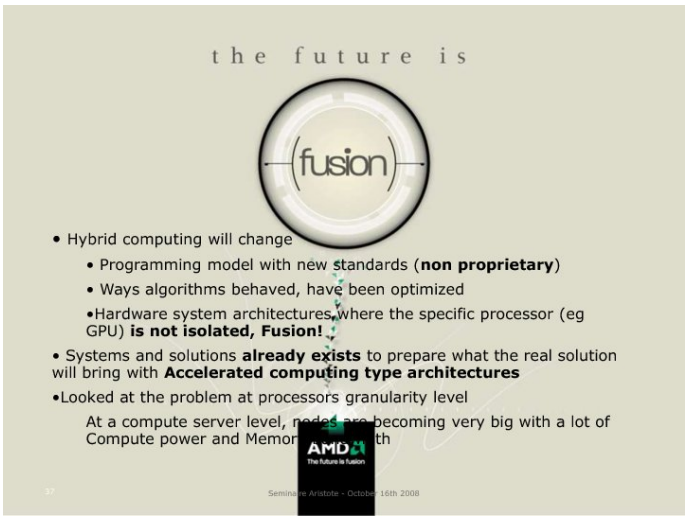
Hybrid Computing -AMD FireStream 9250 Stream Processor

- **Breaks the 1TFLOPS barrier!**
- Power-efficient architecture delivers up to 8 GFLOPS per watt
- 2nd generation double-precision floating point hardware achieving >200GFLOPS
- Fast FFTs: Mercury Computer Systems reports it is achieving 192 GFLOPS on large 1D complex single-precision FFTs
- 1 GB GDDR3 memory
- Single-slot, low-power solution ideal for 1U servers and most workstation and desktop configurations
- PCIe 2.0 x16 interface
- MSRP = $999

Hybrid Computing - Fusion

Fusion will bring hybrid/accelerated computing everywhere
In any system by default




the future is
fusion
• Hybrid computing will change
• Programming model with new standards (**non proprietary**)
• Ways algorithms behaved, have been optimized
•Hardware system architectures where the specific processor (eg GPU) **is not isolated, Fusion!**
• Systems and solutions **already exists** to prepare what the real solution will bring with **Accelerated computing type architectures**
•Looked at the problem at processors granularity level
At a compute server level, nodes are becoming very big with a lot of Compute power and Memor th
AMD
The future is fusion
37
Seminaire Aristote - October 16th 2008


the future is
fusion
• Andre Gomez – Montalvo (andre.gomez-montalvo@amd.com
• Herve Chevanne (herve.chevanne@amd.com)
• Bruno Stefanizzi (bruno.stefanizzi@amd.com)
AMD
The future is fusion
38
Seminaire Aristote - October 16th 2008

2.5 La vision de Bull pour les futurs systèmes HPC

Jean-François Lemerre (Bull)

Exposé des principaux développements dans le domaine de l'architecture en cours chez Bull pour fournir les futurs systèmes HPC capables de couvrir les besoins allant de quelques teraflops jusqu'au petaflops

Architect of an Open World™

Bull's vision for upcoming HPC systems

Jean-François Lemerre

LIBERATE IT

Objectives of Bull developments

- Making the best of existing technologies
  - Powerful standard processors, GPU
  - Interconnect
  - OpenSource software components
- Focus on key areas for innovation
  - Best performance
  - Flexible architecture
  - Scalability to petaflops
- As a consequence investment in :
  - Servers for HPC systems with and without GPUs
  - Integration and key infrastructure elements
  - Software stacks

2 ©Bull, 2008 Aristote 16 Octobre 2008

Future servers for HPC systems

LIBERATE IT

©Bull, 2008 Aristote 16 Octobre 2008

Twin 1U server

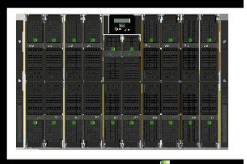
192 GFLOPS

2 nodes in 1U with
2*2 sockets Intel Xeon Nehalem → 8 cores
Memory, disk, GbEthernet, IB

4 ©Bull, 2008 Aristote 16 Octobre 2008

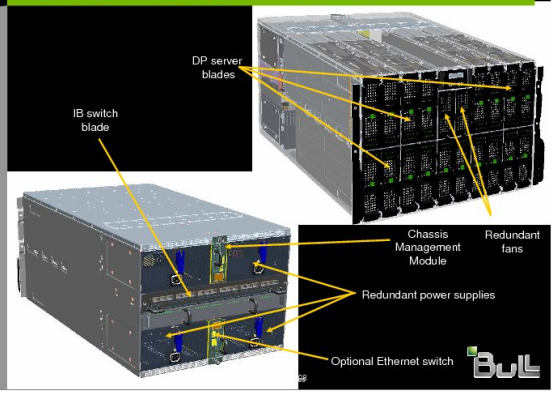
Inca : blade-based slim node cluster

- Optimized for HPC
- Support of the best technologies
  - Nehalem EP up to 95W
  - Memory performance
  - IB up to QDR
- Reduction of components
  - Cost reduction
  - Better reliability
- Versatile building block
  - Interconnect technology (IB, GEth)
  - Interconnect topology



5 ©Bull, 2008 Aristote 16 Octobre 2008

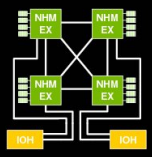
Inca : blade-based slim node cluster



6

Mesca : SMP node cluster

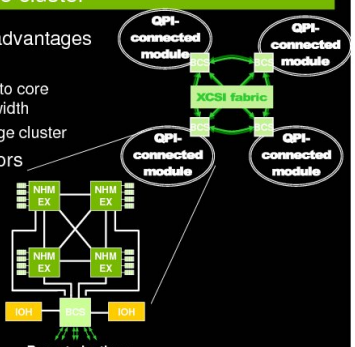
- Large SMP HPC advantages
  - Large memory
  - Best in class core to core latency and bandwidth
  - Less nodes for large cluster
- Mesca differentiators
  - Integration
  - BCS performance



For 4 socket-only systems

Mesca : SMP node cluster

- Large SMP HPC advantages
  - Large memory
  - Best in class core to core latency and bandwidth
  - Less nodes for large cluster
- Mesca differentiators
  - Integration
  - BCS performance

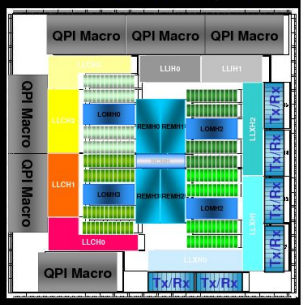


Repeated n times for >4 socket systems

Bull's Coherent Switch , the BCS

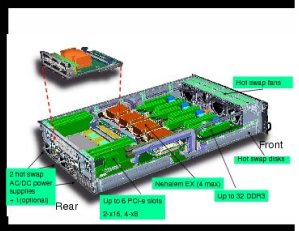
- At the heart of the SMP
  - For insuring global memory and cache coherence
    - both IPF and XPF variants
  - For optimizing traffic and latencies
  - BCS extends QPI protocol into Bull's 2-level hierarchical coherent protocol over QPI and XCSI
- Some key characteristics
  - 18x18 mm in 90 nm technology
  - 6 QPI and 6 XCSI
  - High speed serial interfaces up to 8GT/s
  - Power-conscious design with selective power-down capabilities
  - Aggregate data rate : 230GB/s

Sketch of BCS floor plan



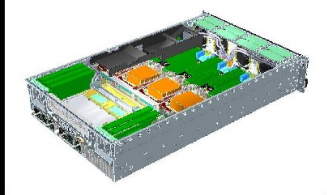
Modular system

- From 4S to 16S
- Drawer for compute nodes
- and IO nodes



Example of advantages : MPI barriers

- Hardware mechanism to reduce message exchanges
- Implemented through Bull BCS for server from 4 to 16 sockets
- Speed-up in excess of 2.3 achieved for just 32 core barrier



Integration of GPU

Standard servers

INCA blade

Mesca Server

Comparison of some solutions for GPU integration

	System definition				System figures of merit			
	Module size (in U)	Nb of servers	Cores per server	Nb of GPUs per server	Cores per GPU	PCIe over-subs-cribing	Density in GPUs per U	Total power per GPU
Twin 1U + S1070	2 U	2	8	2	4	x2	2,00	475 W
Specific+4xS1070	5 U	2	8	8	1	x2	3,20	305 W
			32					
18xC1060 in Inca	7 U	9	4	2	2	x2	2,57	335 W
Mesca + 3xS1070	6 U	1	32	12	2,67	x3	2,00	378 W
Mesca + 12xS1070	24 U	1	128	48	2,67	x3	2,00	380 W

13 ©Bull, 2008 Aristote 16 Octobre 2008

Bull's position

- From a hardware viewpoint, Bull believes that current connection architecture for GPGPUs (i.e. the PCI-express) is both
  - A valid trade-off for the 2 years, or so, to come
  - An interim approach before solutions with a better architected memory connection appear
- PCI express connection (even with Gen3) creates a memory access bottleneck resulting either in
  - Limitation in the ability to take advantage of available processing performance (performance impact and market reach impact)
  - Duplication of central memory capabilities (cost impact)

We recommend to view the initial investment in the current technology as foundation work to reap full benefits with better integrated schemes

14 ©Bull, 2008 Aristote 16 Octobre 2008

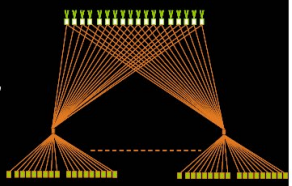
Interconnection, integration, cooling

LIBERATE IT

©Bull, 2008 Aristote 16 Octobre 2008

"Smart" pruning

- A strongly recommended option for 50TFlops and beyond
  - to optimize interconnect expenses
  - without reducing performances
- System seen as a set of "islands"
  - densely intra-connected (high bandwidth and low latency)
  - more lightly inter-connected (still maintaining low latency)
  - Typical island size : ~ 4000 cores



- It works :
  - Applications that do scale beyond 4000 cores tend to have modest global bandwidth requirements
  - The approach is in line with current practice in operating large HPC clusters

16 ©Bull, 2008 Aristote 16 Octobre 2008

Integration

- Physical integration is mandatory
  - Reduction of connection length at all levels
- Improved density means a much higher thermal density
  - Four-fold increase over 5 years
  - From 7kW/m² in 2005 to 28 kW/m² in 2010
- Bull achievements with standard racks
  - 33 U/m² (using six 32A «zero U» PDUs)
  - Approximately 600 cores/m² for both MESCA & INCA projects
  - Power up to 1kW/U

17 ©Bull, 2008 Aristote 16 Octobre 2008

Cooling solution

- Water Cooled Rack
  - An option that can be added to the standard rack, as needed
  - Calories are extracted thru air/water heat exchangers located close to the servers
  - WCR is sized for 40kW (approximately 1KW/U)
  - Main advantages are
    - no water into the servers
    - high heat transfer capability of chilled water (4000x air capability)

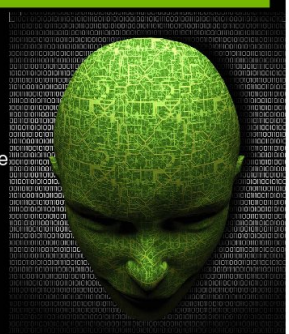
18 ©Bull, 2008 Aristote 16 Octobre 2008

Summary

- Bull committed to HPC
- Strong investment and especially in R&D
- Comprehensive offerings to serve from mid-size to petascale systems
- GPGPU will be an important part of our offer

Prix Bull – Joseph Fourier en association avec Genci

- Récompense des travaux dans le domaine de la parallélisation des applications
- Prix : 15 000 € et heures de calcul
- www.prix-bull-fourier.fr



Architect of an Open World™

LIBERATE IT

2.6 Le processeur Cell, un multi-cœurs innovant et efficace pour le HPC et le multimédia

Olivier Multon (IBM)

Pendant de nombreuses années, la vitesse d'horloge des microprocesseurs a augmenté en vue d'accroître leurs performances. Cette technique est à présent dépassée compte tenu des limites physiques des semi-conducteurs et des architectures des processeurs traditionnels. Les problèmes de consommation énergétique, de dissipation et de temps de latence des mémoires réduisent en effet la réalité des gains réalisés sur les performances réelles des applications.

Ces nouvelles contraintes ont amené IBM, Sony et Toshiba à s'associer pour la conception, l'industrialisation et la commercialisation en volume du processeur multi-cœurs Cell qui décline de nombreuses innovations technologiques tout en équipant déjà aujourd'hui plusieurs millions de consoles de jeux Sony PS3.

Processeur multi-cœurs hérérogène (de type 8 SPE + 1 PPE) privilégiant la manipulation de gros volumes de données, le processeur IBM Cell est maintenant disponible dans des serveurs industriels à base de lames Cell BE QS22 de 200Gflops (DP) intégrables dans des configurations multi-nœuds de clusters ainsi dotés de ces accélérateurs spécifiques.

Le processeur Cell est doté d'un environnement de programmation utilisable sur toutes les plateformes qui l'intègrent, allant de la Sony PS3 au cluster de lames QS22 en passant par les cartes accélératrices comme celles de Mercury Computer. Les techniques de programmation y évoluent elles aussi régulièrement : elles incluent désormais des modules optimisés de bibliothèques numériques « standardisées » et l'utilisation de compilateurs de hauts niveaux capables de gérer efficacement les 8 cœurs de traitement.

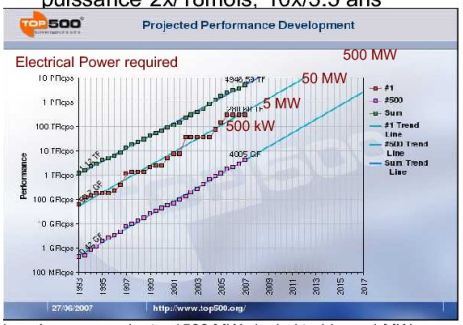
Les applications de calcul haute performance en cluster, la création de contenus numériques, la surveillance vidéo numérique, le traitement de l'image et du signal, les algorithmes financiers, la recherche scientifique et les analyses sismiques peuvent ainsi utiliser ce tout nouveau type d'architecture pour obtenir des niveaux de performances très intéressants.

Architecture et programmation du processeur Cell

Olivier Multon
HPC Business Development

IBM Systems

Top 500 trends assuming constant W/flops
puissance 2x/18mois, 10x/3.5 ans

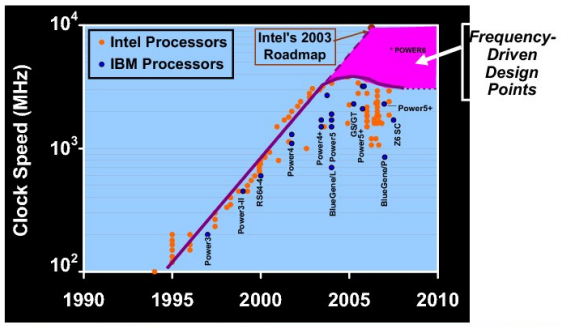


1 nuclear power plant = 1500 MW, 1 wind turbine = 1 MW

IBM Systems

Microprocessor Clock Speed Trends

Managing power dissipation is limiting clock speed increases



IBM Systems

IBM BladeCenter QS22 and Blue Gene/P Sweep Top Mflops/Watt

Top 10 Green500 are all from IBM

IBM Delivers most energy efficient supercomputer (June 2008)

Green500 Rank 1-10	Vendor	Site	System	TOP500 Rank	Total Power (kW)	Mflops /Watt
1	IBM	IBM Germany	BladeCenter QS22 Cluster, PowerXCell 8i 3.2 GHz, Infiniband	324	22.76	488.14
2	IBM	Fraunhofer ITWM	BladeCenter QS22 Cluster, PowerXCell 8i 3.2 GHz, Infiniband	464	18.97	488.09
3	IBM	DOE/NNSA/LANL	BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 GHz / Opteron DC 1.8 GHz , Voltaire Infiniband	1	2345.5	437.43
4	IBM	Argonne National Laboratory	Blue Gene/P Solution	304	31.5	371.75
5	IBM	Dublin Institute for Advanced Studies/ICHEC	Blue Gene/P Solution	305	31.5	371.75
6	IBM	Science and Technology Facilities Council - Daresbury Laboratory	Blue Gene/P Solution	306	31.5	371.75
7	IBM	ASTRON/University Groningen	Blue Gene/P Solution	51	94.5	371.67
8	IBM	RZG/Max-Planck-Gesellschaft MPI/IPP	Blue Gene/P Solution	52	94.5	371.67
9	IBM	IBM - Rochester	Blue Gene/P Solution	37	126	371.67
10	IBM	Oak Ridge National Laboratory	Blue Gene/P Solution	74	63	371.67

Green500 systems ranked 11-15 are also Blue Gene/P systems

4 06/18/08 IBM Systems

Cell BE Systems Details

IBM Systems

Cell hardware

- Sony PlayStation 3
  - 1 Cell BE, 6 SPE, 256 MB ...
  - dual boot Linux - GameOS
- Blade QS20/QS21/QS22 IBM
  - 2 Cell BE, 2x8 SPE, jusqu'à 32GB de SDRAM
  - 8.5 Tflops DP / rack !
- Accelerator PCI-e board from Mercury Computer Systems
  - 1 Cell BE, 8 SPE, 2GB
- The Sony PS3 is a valid Cell software development system fully supported by the current SDK 3.0
- Cell full system simulator on x86

IBM Systems

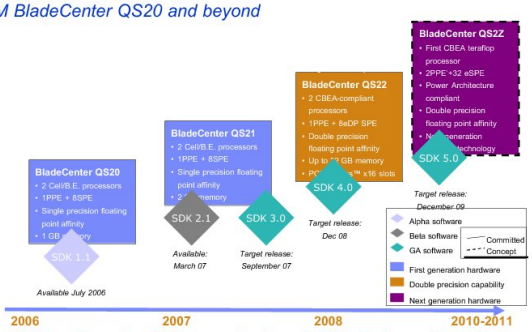
IBM BladeCenter QS22

Premier blade for HPC workloads: 408GFlops/SP,204GFlops/DP

- QS22 is the RIGHT choice for intensive streaming and/or single and double precision floating point workloads
- QS22 is OPEN – based on Power Architecture and running Linux® OS
- QS22 is EASY to deploy and to integrate into the existing IT infrastructure and/or workloads:
  - Co-exist and complement all other Blade servers offerings (Intel®, AMD®, POWER®)
  - Ready to scale out and deploy in production environments
- QS22 is GREEN – more than 1.7 SP (or 0.8 DP) GFLOPS per watt.

Cell Broadband Engine Architecture Blades

IBM BladeCenter QS20 and beyond



All future dates and specifications are estimations only; Subject to change without notice.

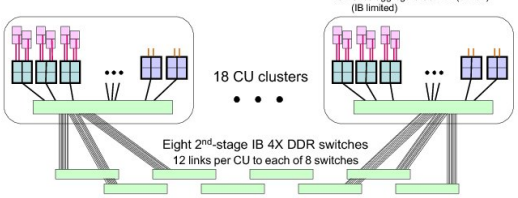
Los Alamos Roadrunner is 1rst Petascale system

Full Roadrunner Specifications:

6,912 dual-core Opterons
49.8 TF DP peak Opteron
27.6 TB Opteron memory

12,960 Cell eDP chips
1.33 PF DP peak Cell eDP
2.65 PF SP peak Cell eDP
51.8 TB Cell memory
277 TB/s Cell memory BW

3,456 nodes on 2-stage IB 4X DDR
13.8 TB/s aggregate BW (bi-dir) (1st stage)
6.9 TB/s aggregate BW (bi-dir) (2nd stage)
3.5 TB/s bi-section BW (bi-dir) (2nd stage)
432 10 GigE I/O links on 216 I/O nodes
432 GB/s aggregate I/O BW (uni-dir)
(IB limited)



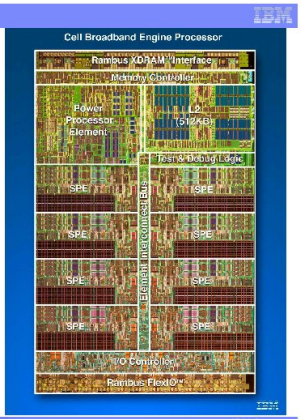
IBM Roadrunner at a glance

- Cluster of Connected Units
  - 12,240 IBM PowerXCell 8i accelerators
  - 6,120 AMD dual-core Opterons (comp)
  - 408 AMD dual-core Opterons (I/O)
  - 34 AMD dual-core Opterons (man)
  - 1.332 Petaflop/s peak (PowerXCell)
  - 44 Teraflop/s peak (Opteron-comp)
  - 1.026 Petaflop/s sustained Linpack
- InfiniBand 4x DDR fabric
  - 2-stage fat-tree; all-optical cables
  - Full CU bi-section bi-directional BW
    - 384 GB/s (CU)
    - 3.3 TB/s (system)
  - Non-disruptive expansion to 24 CUs
- 98 TB aggregate memory
  - 49 TB Opteron
  - 49 TB Cell
- 408 GB/s peak File System I/O:
  - 204x2 10G Ethernets to Panasas
- RHEL & Fedora Linux
- SDK for Multicore Acceleration
- xCAT Cluster Management
  - System-wide GigEnet network
- 2.35 MW Power (Linpack):
  - 437 Megaflop/s per Watt
- Other:
  - 278 racks
  - 5200 ft²
  - 500,000 lbs.
  - 55 miles of IB cables

Le processeur Cell en détails

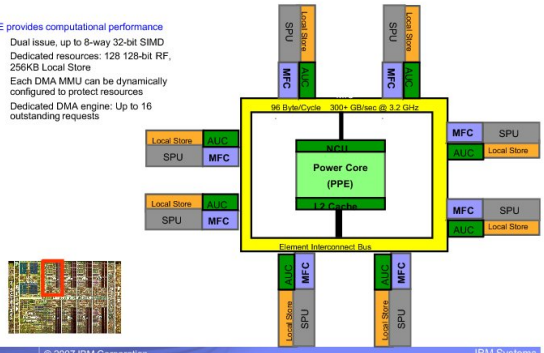
CellBE Processor

- ~250M transistors
- ~235mm2
- Top frequency >3GHz
- 9 cores, 10 threads
- > 200+ GFlops (SP) @3.2 GHz
- > 100+ Gflops (DP) @ 3.2 Ghz
- 300 GB/s interconnect B/W
- Up to 25.6GB/s memory B/W
- Up to 25+ GB/s I/O B/W
- ~400M$(US) design investment



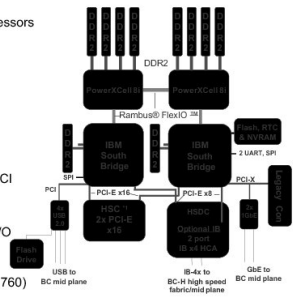
Cell Processor Components

- SPE provides computational performance
  - Dual issue, up to 8-way 32-bit SIMD
  - Dedicated resources: 128 128-bit RF, 256KB Local Store
  - Each DMA MMU can be dynamically configured to protect resources
  - Dedicated DMA engine: Up to 16 outstanding requests



BladeCenter® QS22 – 2 processeurs PowerXCell 8i

- **Core Electronics**
  - Two 3.2GHz PowerXCell 8i Processors
  - SP: 460 GFlops peak per blade
  - DP: 217 GFlops peak per blade
  - Up to 32GB DDR2 800MHz
  - Standard blade form factor
  - Support BladeCenter H chassis
- **Integrated features**
  - Dual 1Gb Ethernet (BCM5704)
  - Serial/Console port, 4x USB on PCI
- **Optional**
  - Pair 1GB DDR2 VLP DIMMs as I/O buffer (2GB total) (46C0501)
  - 4x SDR InfiniBand adapter (32R1760)
  - SAS expansion card (39Y9190)
  - 8GB Flash Drive (43W3934)



*The HSC interface is not enabled on the standard products. This interface can be enabled on "custom" system implementations for clients by working with the Cell services organization in IBM Industry Systems.

Cell at work, a few examples

Motion detection – Orsay University

- Cell BE used to perform sophisticated image processing in real time.
- Requires huge streaming capabilities to take decisions in less than 50ms.
- Other topics include image indexing, automatic reading, etc.

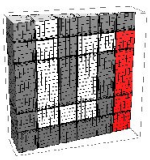
A most difficult test was proving that real applications would perform well on Roadrunner: October 17-19, 2007.

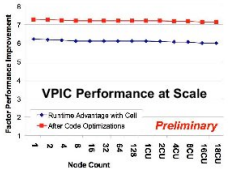
Advanced Algorithms and Applications team

Application	Class	eDP Cell
SPaSM	full app	4.5x
VPIC	full app	9x
Milagro	full app	6.5x
Sweep3D	kernel	9x

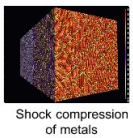
Performance and Architecture Laboratory

Double bend test problem for Milagro
- RZWedge mesh
- optically thick cells (grey)
- optically thin cells (white)
- Hot source (red)



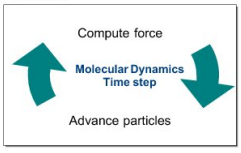


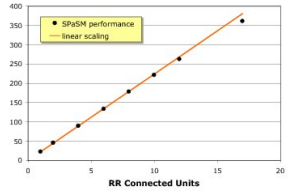
SPaSM is a framework for materials science research as well as epidemiology, fluid instabilities and turbulence.



Shock compression of metals

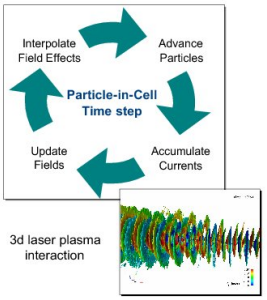
Pandemic



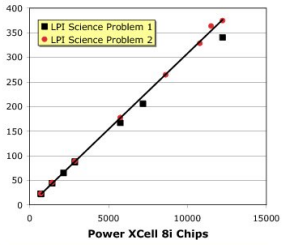


SPaSM achieved a sustained 361 TF/s on the full RR system (DP).

VPIC will address grand challenges in plasma physics.



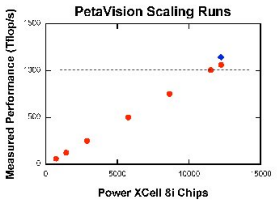
3d laser plasma interaction



VPIC achieved a sustained 374 TF/s on the full RR system (SP)

The goal of Petavision is synthetic visual cognition

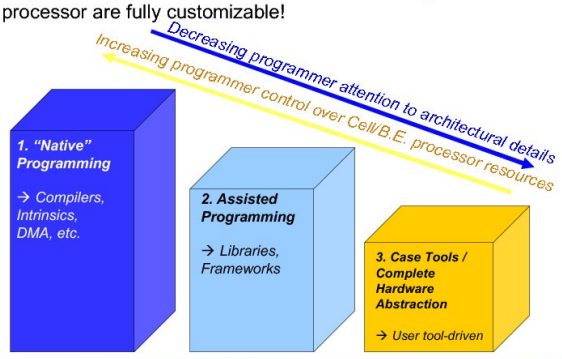
- A billion (10^9) neuron simulation of visual cortex on full Roadrunner system
  - Used simple neurons with Zucker connection weights
  - First large-scale calculation with Zucker weights and spiking neurons
  - Next step is to add a complex neuron layer with stored weights to add learning
- Petavision is a collaboration of scientists from Los Alamos, Yale, MIT and IBM



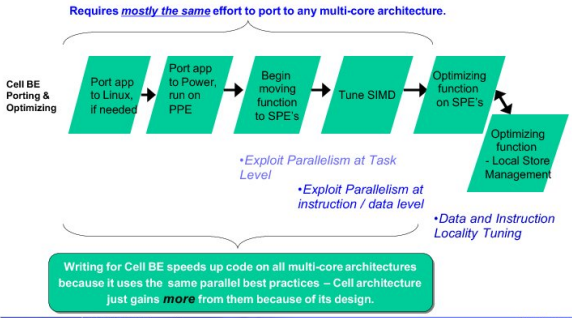
Petavision achieved a sustained 1.144 PF/s on the full RR system (SP).

Cell programming

Programming Approaches for Cell Broadband Engine processor are fully customizable!

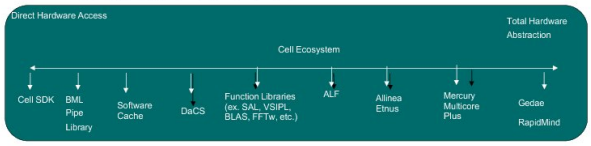


Cell Multi-core Programming Effort Roadmap



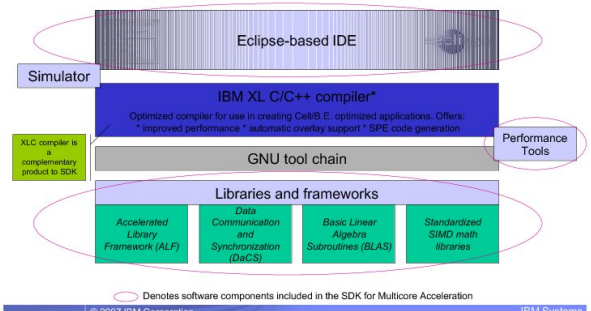
What does Cell BE look like to the Programmer?

- A Programmer's view of Cell depends upon the programming model and tools being used.
- IBM is developing programming models and tools in conjunction with its partners.
  - A properly selected Cell programming model provides a programmer with a systematic and cost-effective framework to apply Cell resources to a particular class of applications.
  - A Cell programming model may be supported by language constructs, runtime, libraries, or object-oriented frameworks.



IBM SDK for Multicore Acceleration and related tools

The IBM SDK is a complete tools package that simplifies programming for the Cell Broadband Engine Architecture



General structure of a Cell application's source code

- A PPE part
  - real main() is here with OS related activities
  - controls and manages the SPE workloads
- SPE parts
  - responsible for performing the actual work
- Control and synchronization through libspe2 calls
  - synchronization primitives (signals, mailbox)
  - DMA calls to access main memory
- **Single source compiler** coming before end of 2008
  - C beta compiler actually, GA december 08
  - Fortran beta compiler before end 08, GA in 1Q09

Single Source Compiler Supporting OpenMP Directives

- **Source files containing user directives as input**
  - OpenMP directives natural choice – wide acceptance, simplicity
- **Compiler, guided by these directives, takes care of the following**
  - code partitioning and parallelization between PPE and SPEs
  - auto-simdization
  - data transfers and synchronization
  - code size (overlay management)
- **User interactions**
  - Performance tuning compiler options
  - Hand-optimized functions provided by user

beta right now, GA planned 12/08

Cell Broadband Engine resource center: http://www.ibm.com/developerworks/power/cell/

Cell/B.E. Software & Tools

Note: 3rd party offering support may not be available from IBM

Public information current as of May 2008

Legend: IBM Offering | 3rd Party Offering | Open Source

Linux distros for Cell

- Terrasoft Solutions
  - YDL 5.0 (YellowDog Linux), tested on PlayStation 3
- Fedora Core 6, 7, 8
  - tested on QS20, QS21, QS22
- RHEL5.1 on QS21, RHEL5.2 on QS22
- Cell is a known supported sub arch of ppc in the Linux kernel since 2.6.15, thanks to the Barcelona SuperComputing Center
- SDK to be installed separately

Web Cell links

- Online virtual cell programming classes at IBM Education Assistant
  - http://publib.boulder.ibm.com/infocenter/ieduasst/stgv1r0/index.jsp
- Cell BE workshop at IBM Innovation Centers for Business Partners
  - https://www-304.ibm.com/jct09002c/isv/spc/events/cbea.html
- Cell BE training podcast series at power.org
  - http://www.power.org/resources/devcorner/cellcorner/CellTraining_Track1
- Cell BE resource center at developerWorks
  - http://www-128.ibm.com/developerworks/power/cell/
- Cell developer's corner at power.org
  - http://www.power.org/resources/devcorner/cellcorner/
- The cell project at IBM Research
  - http://www.research.ibm.com/cell/
- The Cell BE at IBM alphaWorks
  - http://www.alphaworks.ibm.com/topics/cell
- Cell BE at IBM Engineering & Technical Services
  - http://www-03.ibm.com/technology/
- IBM Power Architecture
  - http://www-03.ibm.com/chips/power/
- Cell BE documentation at IBM Microelectronics
  - http://www-306.ibm.com/chips/techlib/techlib.nsf/products/Cell_Broadband_EngineCell
- Linux info at the Barcelona Supercomputing Center website
  - http://www.bsc.es/projects/deepcomputing/linuxoncell/

2.7 Actualités GENCI

Stephane Requena (GENCI)

GENCI

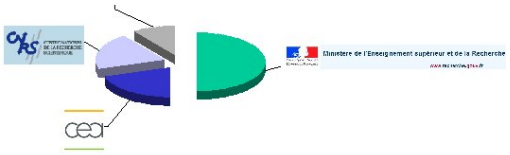
GRAND EQUIPEMENT NATIONAL DE CALCUL INTENSIF

Grand Équipement National de Calcul Intensif

GENCI

Grand Équipement National de Calcul Intensif

- Société civile créée en 2007
- 4 associés



- 25 M€ de budget annuel pendant 4 ans.

GENCI

Missions de GENCI

- Mettre en place et assurer la **coordination des principaux équipements** des grands centres nationaux civils dont elle assure le financement et est propriétaire

  → **Maîtrise d'ouvrage nationale**
- Promouvoir l'**organisation d'un espace européen** du calcul intensif et participer à ses réalisations

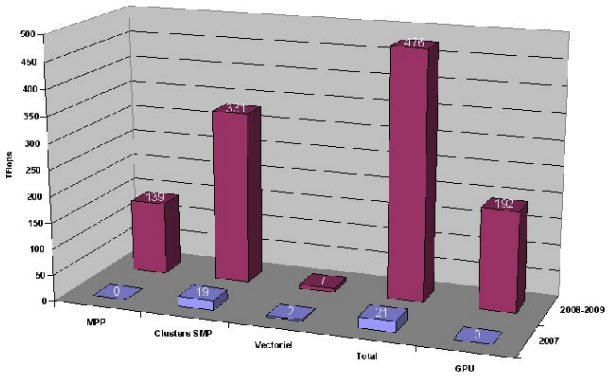
  → **Participation à l'initiative PRACE**
- **Promouvoir la simulation et le calcul intensif** dans la recherche fondamentale et industrielle

et aussi

- Faire exécuter les **travaux de recherche** nécessaires au développement et à l'optimisation des moyens de calcul
- **Ouvrir ses équipements** à toutes les communautés scientifiques intéressées, académiques ou industrielles, nationales, européennes ou internationales.

GENCI

Capacité de calcul disponible à l'ensemble de la communauté scientifique à début 2009



GENCI

Extension des moyens du CCRT

ccrt

- Calculateur hybride BULL
  - 534 serveurs de calcul BULL Novascale R422-E2
    1068 nœuds, 2136 processeurs quad cœur Intel Nehalem
  - Interconnexion IB DDR, espace Lustre 0.5 Po, 20 Go/s
  - Système Linux avec outils d'administration et environnement de programmation
  - >100 TFlops, refroidissement par eau

  - 48 serveurs graphiques nVIDIA Tesla S1070
    - 4 processeurs graphiques T10P (gt200), 16 Go de mémoire par nœud, 2 liens PCIe Gen2 16x, attachement des 48 serveurs vers 48 serveurs R422, 1 T10P pour 1 socket Intel Nehalem
    - 192 GPUs, 192 TFlops simple précision, CUDA2
    - 800W par serveur S1070 (550W typical)
  - Un des plus gros calculateurs hybrides en Europe
  - Disponibilité: Q2 2009 (scalaire), Q3 2009 (gpu)

GENCI

Prototype hybride au CINES

CINES BULL

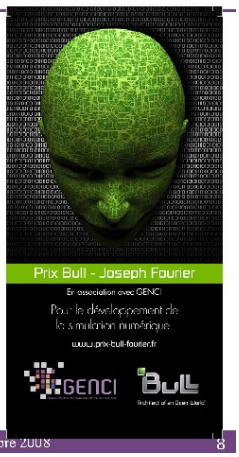
- Convention GENCI / BULL
  - Installation (13/10) d'un cluster hybride au CINES
    - 6 serveurs BULL Novascale R422-E1 avec par serveur 2 nœuds de calcul bi procs quad cœur Intel Hapertown 3.0 Ghz, 16 Go de mémoire
    - 6 serveurs graphiques nVIDIA Tesla S1070
    - Intégration dans environnement cluster BULL Iblis existant (IB, Lustre)
  - Objectifs
    - Installer un prototype de technologie proche du cluster CCRT
    - Donner accès au prototype à la communauté hybride française
    - Former les utilisateurs au GPGPU
    - Préparer un écosystème apte à produire sur la machine de production CCRT installée en 2009
- Contacts :
  - francis.daumas@cines.fr
  - stephane.[illegible]

Club utilisateurs du calcul hybride

- Intérêt du calcul hybride en France
  - GPGPU, processeurs CELL, Clearspeed, FPGA
  - De nombreuses expérimentations en cours en France
  - Tant au niveau académique qu'au niveau industriel
- Proposition GENCI de fédérer ces initiatives en un club d'utilisateurs
  - La promotion du calcul hybride,
  - La veille technologique autour des architectures logicielles et matérielles,
  - La mise en place d'un benchmark « accélérateurs »,
  - La promotion de solutions innovantes (ex: suite HMPP CAPS),
  - Une action de lobbying auprès des fournisseurs de solutions visant à définir au plus vite des modèles de programmation qui deviennent des standards (au sens OpenMP du terme),
  - Une mise à disposition de ressources auprès de la communauté (site WEB, matériels et logiciels, ...)
  - Une action auprès des écoles afin de voir avec eux comment proposer aux étudiants des cursus autour du calcul hybride
- Membres actuels
  - CEA, TOTAL, IFP, Aristote et GENCI
- Contacts
  - stephane.requena@genci.fr

Prix BULL - Joseph Fourier

- Annonce faite par BULL lors du dernier séminaire ORAP
- Récompense travaux sur parallélisation d'applications de simulation numérique sur architectures traditionnelles ou hybrides
- Prix : 15 000€ et heures de calcul sur supercalculateurs
- Comités de sélection et d'attribution indépendant (académiques, industriels)
- Premiers lauréats mi 2009
- www.prix-bull-fourier.fr
- A vous de jouer !!!!



Merci !!!

2.8 Architectures multicoeurs hétérogènes : à quoi vont ressembler les supports exécutifs «next-gen»

Raymond Namyst (LABRI)

Alors que les langages et environnements de programmation parallèle actuels accusent déjà un certain retard vis-à-vis des architectures multicoeurs disponibles depuis quelques années, c'est peu dire que le fossé entre le matériel et le logiciel va se creuser davantage avec la démocratisation des architectures hétérogènes (Cell, GPCPU) et/ou massivement multicoeurs. Ainsi faut-il non seulement composer avec les "pénalités NUMA" exhibées par le matériel, mais aussi avec l'absence de cohérence mémoire entre certains accélérateurs ou encore avec le mode de fonctionnement SIMD de certaines unités de calcul.

Dans cet exposé, nous nous focaliserons sur la conception de supports exécutifs pour de telles machines. En nous appuyant sur des travaux récents dans ce domaine, nous montrerons pourquoi il est nécessaire de repenser leur articulation avec les langages et environnements de programmation, autour d'une vision abstraite – mais surtout pas totalement masquée – de l'architecture sous-jacente. Nous insisterons également sur la necessité de générer du parallélisme "structuré" et "polymorphe", que le support exécutif sera alors en mesure d'appréhender efficacement tout au long de l'exécution...

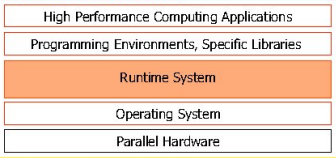
Architectures multicœurs hétérogènes: à quoi vont ressembler les supports exécutifs “next-gen” ?

Raymond Namyst

Centre de Recherche INRIA Bordeaux Sud-Ouest
LaBRI, Université Bordeaux 1

Du rôle des supports d’exécution

- Support d’exécution:
  - Effectue dynamiquement ce que l’on ne sait pas faire statiquement
- Objectif ultime: concevoir des supports d’exécution performants
  - i.e. s’approchant des caractéristiques du matériel
- …préservant la portabilité des applications
  - = portabilité des performances

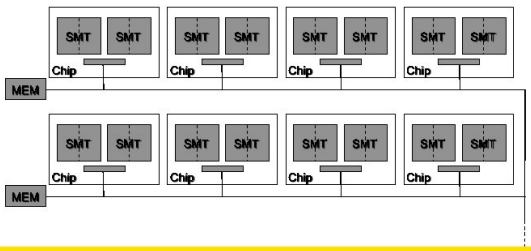


L’ère pré-multicœurs

- Machines à mémoire partagées souvent « uniformes »
  - Symmetric multiprocessing (SMP)
- Préoccupations des supports d’exécution
  - Efficacité des primitives de base (création de tâches, ordonnancement)
    - Cf optimisations au sein des principaux runtime OpenMP
      - Allocations paresseuses de piles, etc.
  - Ordonnancement de threads
    - Cilk
    - Linux 2.6
  - Bref: minimiser le surcoût de gestion des tâches

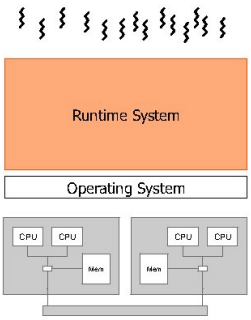
L’émergence des processeurs multicœurs

- Conséquence pour les machines parallèles:
  - Retour à l’ère des architectures CC-NUMA
- Nouveau défi : composer avec la hiérarchie mémoire



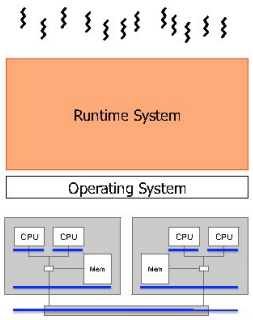
Comment ordonnancer des processus sur de telles machines?

- Les environnements existants n’aident pas…
  - MPI, Posix Threads, OpenMP
- Il y a certes des outils
  - Contrôle précis
    - Ex: Libnuma, liblgroup, etc.
  - Portabilité médiocre
- Qu’est-ce qui cloche?
  - Pas d’abstraction de l’architecture
  - Pas possible d’exprimer des relations entre tâches



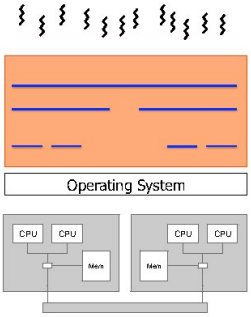
Abstraction de l’architecture

- Modélisation « classique »
  - Hiérarchie de listes d’ordonnancement
    - Peu de contention
    - Plusieurs compromis possibles entre
      - Equilibrage de charge
      - Respect des affinités



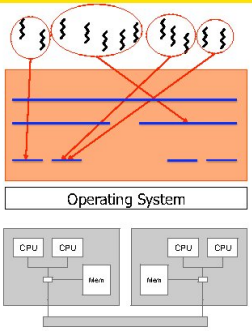
Abstraction de l'architecture

- Comment projeter, à l'exécution, les tâches de l'application sur cette hiérarchie de listes ?
  - Comment abstraire le comportement de l'application ?
    - Partage de données
    - Synchronisations
  - Comment le faire de manière indépendante de l'architecture ?
    - Ordonnancement portable



Le rôle structurant des environnements/langages de programmation

- Les supports d'exécutions ont besoin d'informations
  - Directives
  - Structure du parallélisme
- Les compilateurs et/ou environnements doivent les leur transmettre !
- Ex: OpenMP au-dessus de l'environnement BubbleSched
  - Sections parallèles = outil structurant

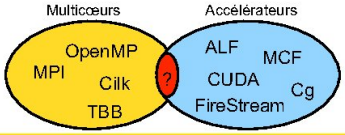


Architectures multicœurs hétérogènes

- Nouvelle tendance :architectures dotées d'accélérateurs/ coprocesseurs
  - CPU + GPGPU + SPU ??
- Par rapport aux multicœurs homogènes, cela signifie
  - Plus de cohérence mémoire implicite
    - Transferts explicites des données à la charge du support d'exécution
    - Contraintes mémoire sur les tâches (mémoire embarquées)
  - Fonctionnement différent des unités de calcul
    - SIMD, précision numérique
    - Implémentation des tâches différentes en fonction de la cible
  - Problème de granularité des calculs
    - Certains accélérateurs nécessitent un degré de parallélisme extrêmement fin
      - *Embarassingly parallel machines?*
  - Comment on débogue ? Comment analyse-t-on les performances ?

Quels supports d'exécution pour ces machines ?

- Beaucoup de travaux portent sur l'utilisation des accélérateurs en tant que… coprocesseurs
  - Ex: délégation de certains calculs au GPGPU
  - Ex: OpenMP on Cell
- Le vrai défi est d'exploiter 100% des ressources matérielles
  - Considérer l'ensemble des unités de calcul simultanément
  - Dans le cas général, il est difficile de pré-affecter statiquement les tâches
    - Accepter l'idée que certaines tâches puissent s'exécuter sur plusieurs types d'unité

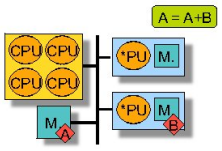


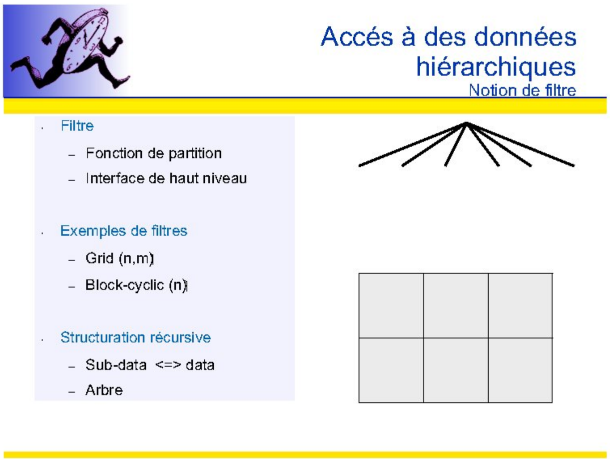
Vers des supports d'exécution unifiés

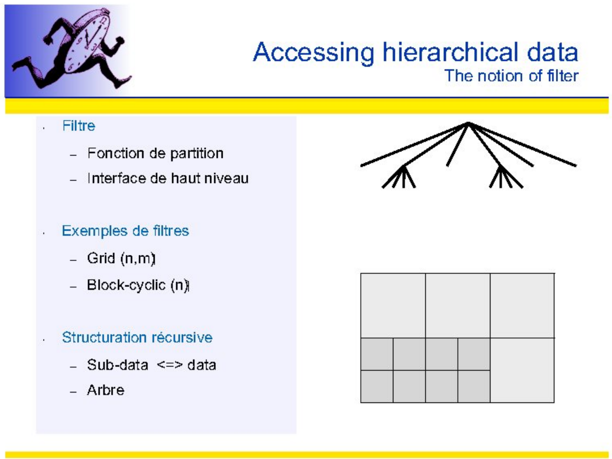
- Prérequis
  - Le support doit disposer
    - Des dépendances de données
    - De différentes versions du code exécutable par tâche
  - Plusieurs environnements de programmation le fournissent déjà
    - HMPP, Brooks, RapidMind, etc.
- Fonctionnalités nouvelles
  - Transferts mémoire explicites entre les unités de calcul
    - Éventuellement, aide au placement de tâches
  - Ordonnancement des tâches dirigé par un modèle de coûts
  - Outils de traçage d'exécution

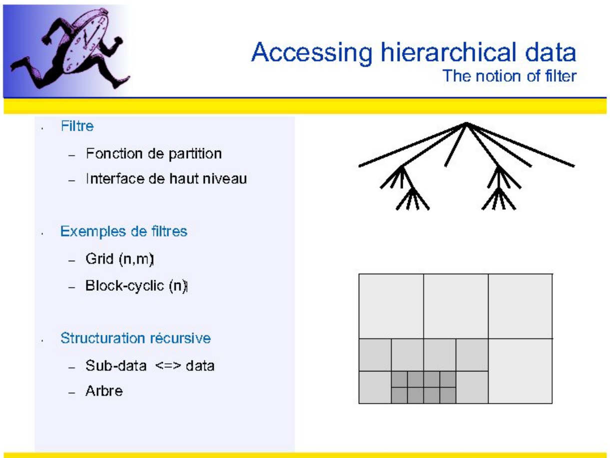
Exemple: support d'exécution développé au sein de l'équipe Runtime

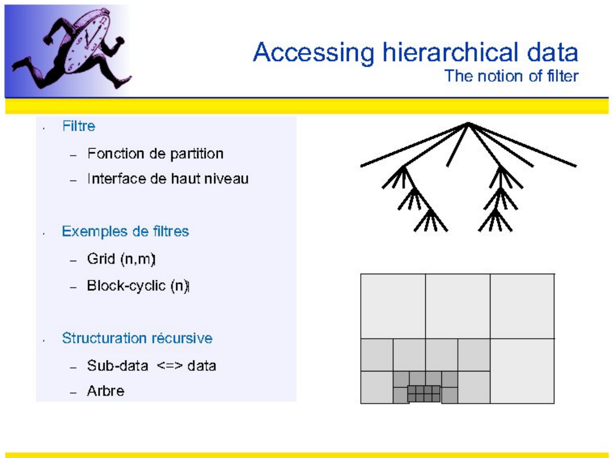
- Bibliothèque de gestion de données
  - Transférer les données
  - Gérer la concurrence
  - Conserver les données cohérentes
  - Offrir une interface de haut niveau
    - ~ *software DSM* !

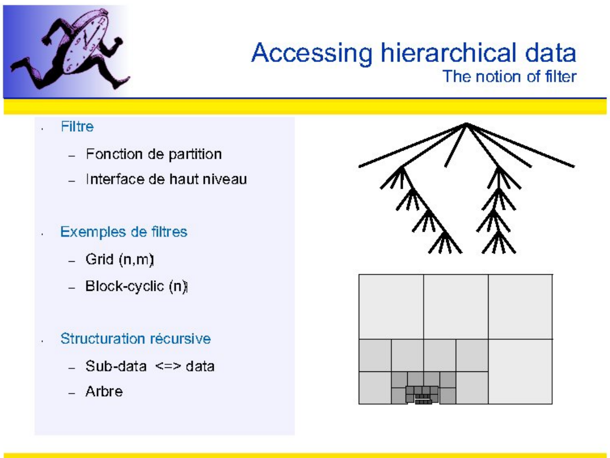


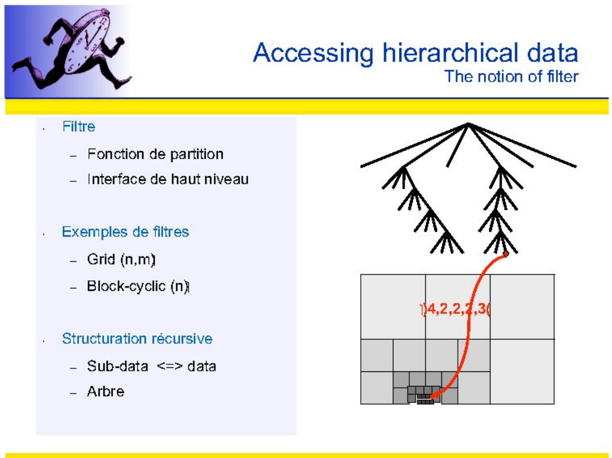

Accés à des données hiérarchiques
Notion de filtre
Filtre
– Fonction de partition
– Interface de haut niveau
Exemples de filtres
– Grid (n,m)
– Block-cyclic (n)
Structuration récursive
– Sub-data <=> data
– Arbre

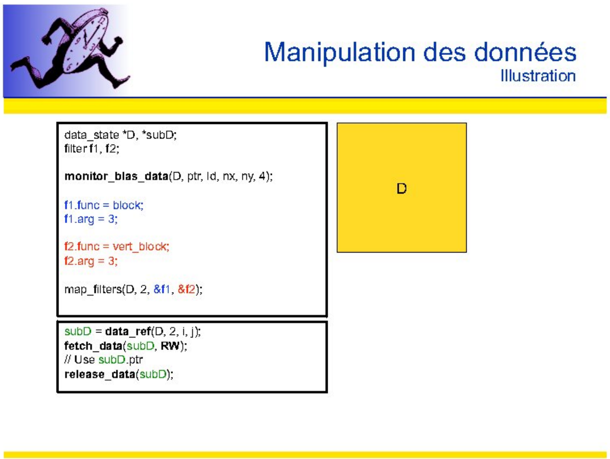

Accessing hierarchical data
The notion of filter
Filtre
– Fonction de partition
– Interface de haut niveau
Exemples de filtres
– Grid (n,m)
– Block-cyclic (n)
Structuration récursive
– Sub-data <=> data
– Arbre

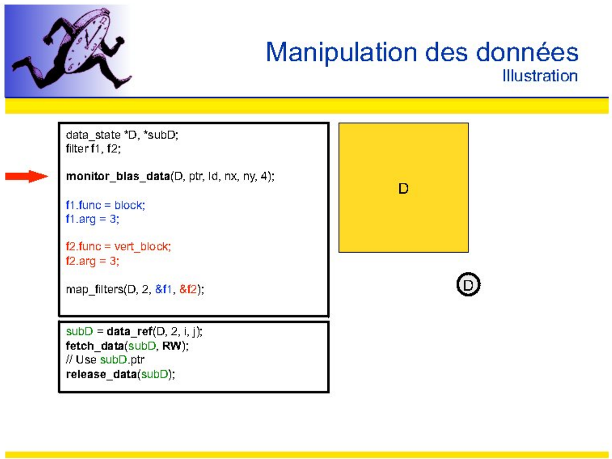

Accessing hierarchical data
The notion of filter
Filtre
– Fonction de partition
– Interface de haut niveau
Exemples de filtres
– Grid (n,m)
– Block-cyclic (n)
Structuration récursive
– Sub-data <=> data
– Arbre

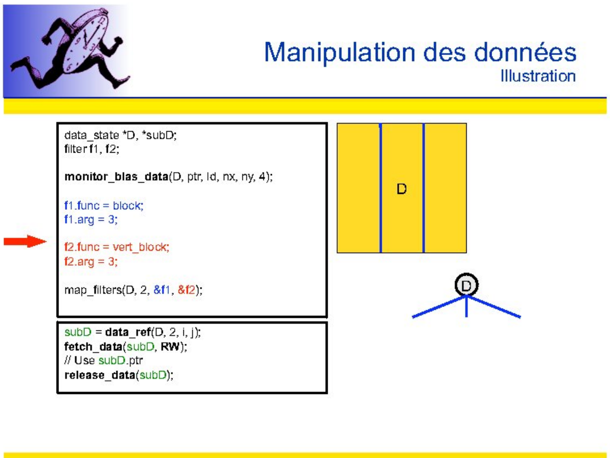

Accessing hierarchical data
The notion of filter
Filtre
– Fonction de partition
– Interface de haut niveau
Exemples de filtres
– Grid (n,m)
– Block-cyclic (n)
Structuration récursive
– Sub-data <=> data
– Arbre

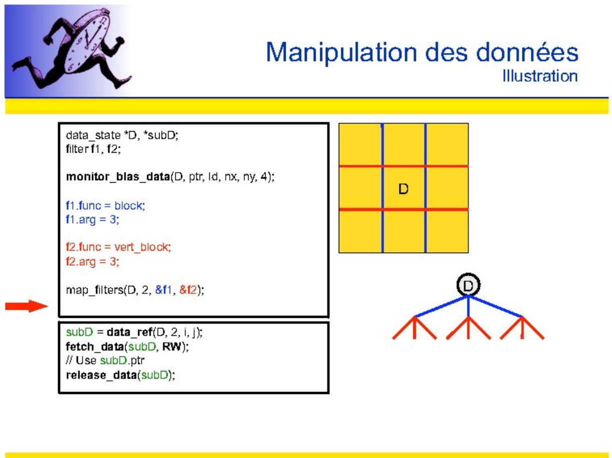

Accessing hierarchical data
The notion of filter
Filtre
– Fonction de partition
– Interface de haut niveau
Exemples de filtres
– Grid (n,m)
– Block-cyclic (n)
Structuration récursive
– Sub-data <=> data
– Arbre

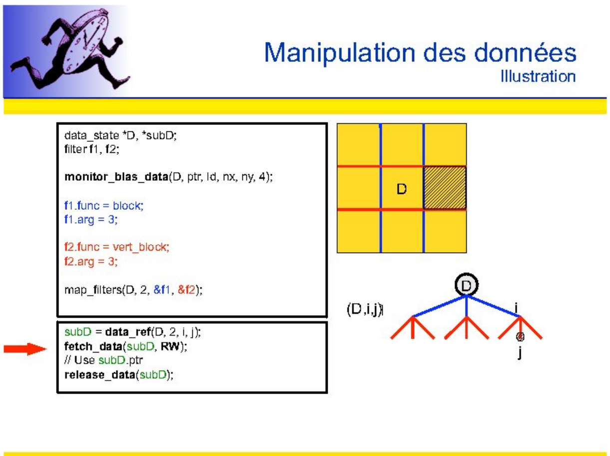

Accessing hierarchical data
The notion of filter
Filtre
– Fonction de partition
– Interface de haut niveau
Exemples de filtres
– Grid (n,m)
– Block-cyclic (n)
Structuration récursive
– Sub-data <=> data
– Arbre

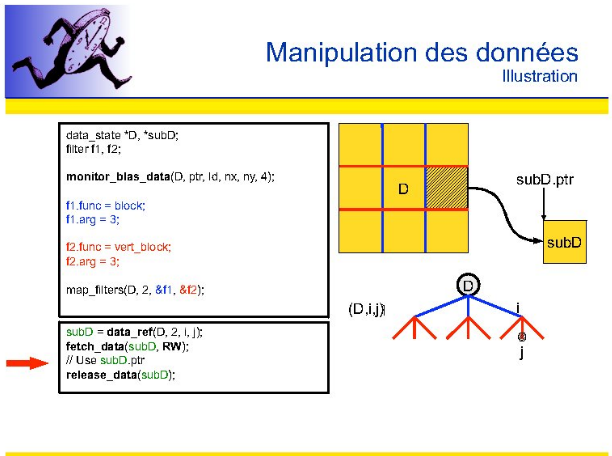

Manipulation des données
Illustration
data_state *D, *subD;
filter f1, f2;
monitor_blas_data(D, ptr, ld, nx, ny, 4);
f1.func = block;
f1.arg = 3;
f2.func = vert_block;
f2.arg = 3;
map_filters(D, 2, &f1, &f2);
subD = data_ref(D, 2, i, j);
fetch_data(subD, RW);
// Use subD.ptr
release_data(subD);
D


Manipulation des données
Illustration
data_state *D, *subD;
filter f1, f2;
monitor_blas_data(D, ptr, ld, nx, ny, 4);
f1.func = block;
f1.arg = 3;
f2.func = vert_block;
f2.arg = 3;
map_filters(D, 2, &f1, &f2);
subD = data_ref(D, 2, i, j);
fetch_data(subD, RW);
// Use subD.ptr
release_data(subD);
D
D


Manipulation des données
Illustration
data_state *D, *subD;
filter f1, f2;
monitor_blas_data(D, ptr, ld, nx, ny, 4);
f1.func = block;
f1.arg = 3;
f2.func = vert_block;
f2.arg = 3;
map_filters(D, 2, &f1, &f2);
subD = data_ref(D, 2, i, j);
fetch_data(subD, RW);
// Use subD.ptr
release_data(subD);
D
D

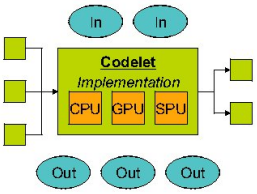

Manipulation des données
Illustration
data_state *D, *subD;
filter f1, f2;
monitor_blas_data(D, ptr, ld, nx, ny, 4);
f1.func = block;
f1.arg = 3;
f2.func = vert_block;
f2.arg = 3;
map_filters(D, 2, &f1, &f2);
subD = data_ref(D, 2, i, j);
fetch_data(subD, RW);
// Use subD.ptr
release_data(subD);
D
D


Manipulation des données
Illustration
data_state *D, *subD;
filter f1, f2;
monitor_blas_data(D, ptr, ld, nx, ny, 4);
f1.func = block;
f1.arg = 3;
f2.func = vert_block;
f2.arg = 3;
map_filters(D, 2, &f1, &f2);
subD = data_ref(D, 2, i, j);
fetch_data(subD, RW);
// Use subD.ptr
release_data(subD);
D
D
(D,i,j)
i
j


Manipulation des données
Illustration
data_state *D, *subD;
filter f1, f2;
monitor_blas_data(D, ptr, ld, nx, ny, 4);
f1.func = block;
f1.arg = 3;
f2.func = vert_block;
f2.arg = 3;
map_filters(D, 2, &f1, &f2);
subD = data_ref(D, 2, i, j);
fetch_data(subD, RW);
// Use subD.ptr
release_data(subD);
D
subD.ptr
subD
D
(D,i,j)
i
j

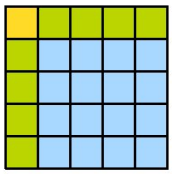
Modélisation des tâches
Notion de *codelet*

- Modèle de tâche unifié
  - Un code par architecture
  - Application = DAG de codelets
- Dépendances
  - Données
  - Tâches
- Exécution asynchrone
  - Réordonnancements possibles
  - Modèle fondé sur les « continuations »

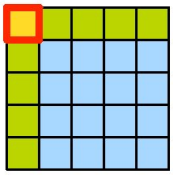


Evaluation
Décomposition LU dense

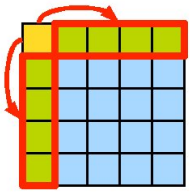
- Sur une machine quadcore Intel Xeon + nVidia Quadro FX4600



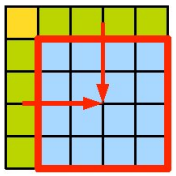
Evaluation
Décomposition LU dense



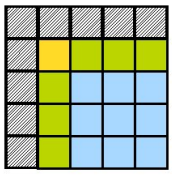
Evaluation
Décomposition LU dense



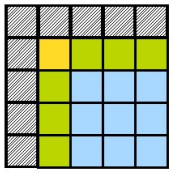
Evaluation
Décomposition LU dense

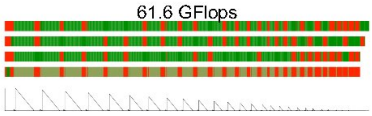


Evaluation
Décomposition LU dense



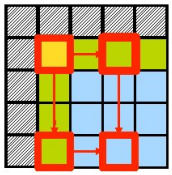
Evaluation
Extraction de davantage de parallélisme

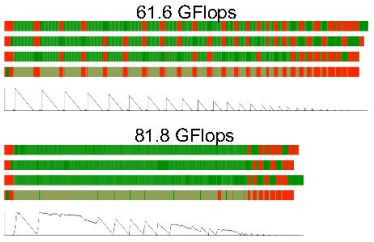




Manque de parallélisme
Pas possible d'alimenter tous les *PU

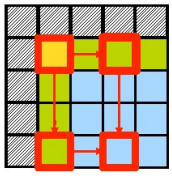
Evaluation
Extraction de davantage de parallélisme

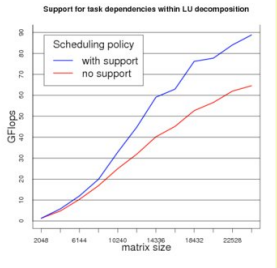




Avec des dépendances plus fines : gain ~ 30%

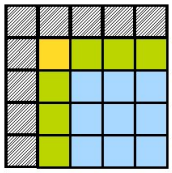
Evaluation
Extraction de davantage de parallélisme





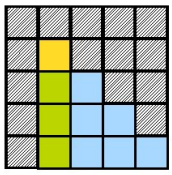
Avec des dépendances plus fines : gain ~ 30%

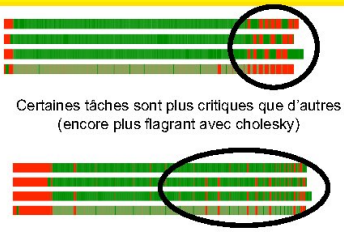
Evaluation
Extraction de davantage de parallélisme



Certaines tâches sont plus critiques que d'autres

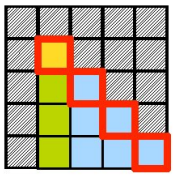
Evaluation
Extraction de davantage de parallélisme

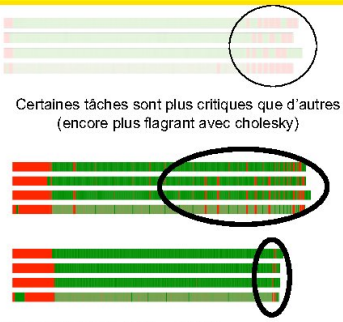




Certaines tâches sont plus critiques que d'autres
(encore plus flagrant avec cholesky)

Evaluation
Extraction de davantage de parallélisme

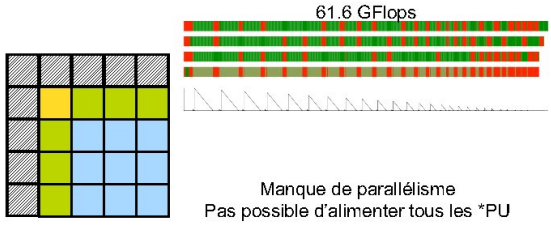




Certaines tâches sont plus critiques que d'autres
(encore plus flagrant avec cholesky)

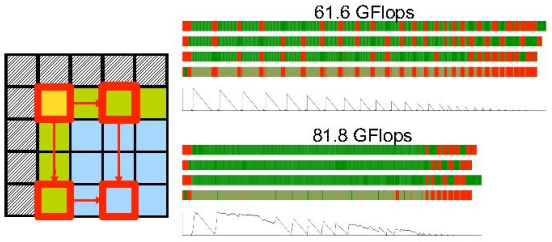
Utilisation de priorités : gain ~ 10 %

Evaluation
Extraction de davantage de parallélisme



Manque de parallélisme
Pas possible d'alimenter tous les *PU

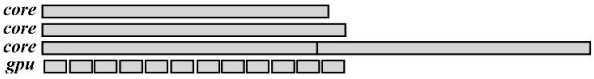
Evaluation
Extraction de davantage de parallélisme



Avec des dépendances plus fines : gain ~ 30%

Evaluation
Limites d'un algorithme glouton

Cas pathologique typique



- Modeling tasks using performance models
  - "mathematical" model
    - number of Flop BLAS3 = O(n^3)
    - use *PU's theoretical Flop/s
  - execution time : benchmark actual kernels
    - Single benchmark + "1 GPU = 10x faster than 1 CPU"
    - Per-architecture benchmark

Evaluation
An efficient parallelization

Stratégie triviale : "1 GPU = 10x faster than 1 CPU"

3 CPUs + 1 GPU	3CPUs	1 GPU
95.41 GFlops	21.24 GFlops	75.04 GFlops

95,41 = 99% de (21,24 + 75,04)

Modèle de performance par accélérateur :

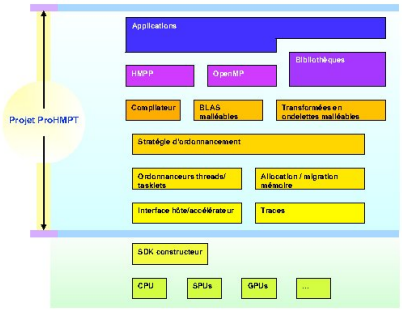
3 CPUs + 1 GPU	3CPUs	1 GPU
98.21 GFlops	21.68 GFlops	75.07 GFlops

98,21 = 101.5% of (21,68 + 75,07)

Conclusion

- Les supports d'exécution ont encore du pain sur la planche…
  - Un modèle unifié pour ttoutes les unités de calcul est trop réducteur
    - Il faut pouvoir intégrer threads / tasklets / codelets
  - Il faut enrichir les modèles pour pouvoir adapter la granularité dynamiquement
    - Supporter efficacement les tâches divisibles
  - Standard pour la génération des tâches
    - OpenCL ?
- Vers des modèles de programmations plus adaptés
  - Inciter au parallélisme massif, structuré
  - Transmettre des directives
  - Remonter un « feedback » exploitable sur les performances
- Concevoir des noyaux numériques portables est crucial
  - Exploitation transparente de CPU + GPGPU + SPU

Projet ANR ProHMPT



2.9 Les outils de programmations

Ronan Keryell (HPC Project)

L'architecture des moyens de calculs est en pleine révolution avec la généralisation des processeurs multi-cœurs, l'apparition d'accélérateurs hétérogènes (Cell, Larrabee, ...) et bien sûr la démocratisation des GPGPU (nVidia, AMD/ATI). La programmation de telles "bêtes de course" pose de nombreux problèmes scientifiquement intéressants mais en même temps effrayants, si on considère que c'est une voie obligée pour survivre à la fin de la loi de Moore en terme de vitesse d'horloge des processeurs.

Dans cet exposé on présentera la problématique de la programmation de ces systèmes, les outils principaux disponibles pour développer et mettre au points les programmes sur ces architectures, avec des langages comme OpenMP, Cuda, Brook+, HMPP, ... ou des bibliothèques spécialisées, ACML, CuBLAS, EcoLib...

Outils pour programmation multicœurs, GPGPU et autres MP-SoC

Ronan KERYELL
(rk@hpc-project.com)

HPC Project
9 Route du Colonel Marcel Moraine
92360 Meudon La Forêt

Rond Point Benjamin Franklin
34000 Montpellier

16 octobre 2008

La crise logicielle

The "Software Crisis"

"To put it quite bluntly : as long as there were no machines, programming was no problem at all ; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem."

Edsger Dijkstra, 1972 Turing Award Lecture, « The Humble Programmer » `http://en.wikipedia.org/wiki/Software_crisis`

⚠ ... et c'était avant la démocratisation du parallélisme...☹

Évolution logicielle

Progression jusqu'à présent

- Langages d'assemblage
- Langages de haut niveau pour machines à la von NEUMANN (Fortran, C...)
- Programmation orientée objet pour composabilité, malléabilité et maintenabilité de gros programmes
- Bibliothèques de composants, outils, patrons de conception, spécifications, modélisation, méthodologies, tests...

Hautes performances ?

Bah... Loi de MOORE ☺

Programmeurs inconscients des processeurs...

- On ne mélange pas les ~~moucheirs en dentelles~~softeux avec les ~~serpillières~~hardeux ☺
- Langages de haut niveau loin du matériel (encore pire avec JVM, CLI...)
- Abstraction qui a permis beaucoup de liberté créatrice aux programmeurs ☺
- L'avenir est au Web 3.0 et au Cloud Computing ! ☺
- La vitesse ? Travailleurs de l'ombre (loi de MOORE...) font qu'un programme antique va beaucoup plus vite aujourd'hui sur n'importe quel processeur ! ☺

⇝ Un programmeur peut tout ignorer des processeurs
∃? encore cours d'architecture des ordinateurs en école d'ingénieurs ? ☹

Fin de l'augmentation des performances séquentielles

- Passe d'un facteur 2 tous les 1,5 ans à 2 tous les ≈ 5 ans... ☹
- Pourtant besoin de toujours plus de performances
  - Devise de Delphes « rien de trop » μηδὲν ἄγαν
  - Données traitées plus grandes
  - Plus de fonctionnalités

Seule solution : faire du parallélisme...

...et garder le moral : « composabilité, malléabilité et maintenabilité, portabilité... »

Comment faire face ?

Extraire du parallélisme dans les applications

- De manière implicite
  - Matériel : processeurs superscalaires
  - Compilateurs auto-parallélisants
- De manière explicite
  - Langages : OpenMP, UPC, HPF, CUDA, Cal, Brook+, OpenCL, HMPP...
  - Bibliothèques : applicatives (mathématiques), parallélisme (MPI, concurrence *pthreads*...), objets (STL parallèles, TBB, Ct)

Programmation ex(a)tensible

- ⚠ Ne pas essayer de programmer avec *quelques* threads/processus/données parallèles
- Il faut des *centaines* ou des *milliers* de threads !
  - Paralléliser code : souvent compliqué
  - Prend du temps
  - ⚠ Sera en retard par rapport aux nouvelles machines sinon... ☹
- Penser « pannes »
  - Déploiement correction d'erreur à tout niveau
    - Correction d'erreur 1 bit/mot classique dans ordinateurs professionnels, devrait arriver dans GPGPU
    - *Check-pointing* : comment sauver 200 To à 200 Go/s (disque SAS coûte 10 × SATA2 ☺) ? 20 mn sur RoadRunner, 30 mn sur BG/P IDRIS...
    - Avec 10^5 cœurs en 2012-2013 : ⇒ 1 panne/heure
    - ⇝ Domaine de recherche en explosion
  - Surtout pannes logicielles en fait (heureusement ?)
  - Algorithmique tolérante aux pannes ? Dans certains cas spécifiques (solveurs itératifs)

⇝ Programmation ex(a)tensible pour viser machines exaflopiques à venir

Hétérogénéité à plusieurs dimensions

- Hétérogénéité des modèles d'exécution
  - SMP multicœurs ± couplés par caches
  - Instructions SIMD dans processeurs (SSE 4.2, 3DNow !, VMX...)
  - Accélérateurs matériels (MIMD, MISD ou SIMD)
- Hétérogénéité avec nouvelles hiérarchies mémoires
  - Classique caches/mémoire physique/disques
  - NUMA (*Non Uniform Memory Access*) : bancs mémoires attachés à des sockets, voire nœuds distants
  - Si mémoire non partagée : mémoires distantes, disques distants
  - Périphériques associés à des sockets : NUPA (*Non Uniform Peripheral Access*)
- Communications hétérogènes
  - Réseaux anisotropes
  - Protocoles variés

⚠ Plusieurs dimensions simultanément dans même machine à gérer ☹

Cycle de développement

1. Analyse de complexité du problème
2. Analyse du programme disponible
   - Analyse performances, profiling (`gprof`, CodeAnalyst, VTune...)
3. Conception de la parallélisation
   - Bibliothèques déjà optimisées (mathématique)
   - Objets parallèles (STL parallèles, TBB...)
   - Langages parallèles (OpenMP, UPC, Fortran 2008...)
   - Bibliothèques parallèles (MPI, threads systèmes...)
4. Mise au point
   - Tests de non régression (⚠ non associativité flottant)
   - Débogage (`gdb`, TotalView...)
   - Correction concurrence (Intel Thread Checker, Helgrind)
5. Optimisation
   - Profiling : Intel Thread Profiler, `gprof`, VTune...

Le plan

Instructions multimédia SIMD (I)

- Explosion du marché du multimédia et des télécommunications
  - Image GIF : 8 bits
  - Image RVBα en vrai couleur 4 × 8 bits
  - son téléphone encodage A ou μ : 8 bits
  - son qualité CD : 2 × 16 bits
- Dans processeur généraliste, transistors sous-utilisés pour ces applications (multiplieur double précision...) ☹
- Idée : traiter données 128 bits interprétées comme des vecteurs de 16 éléments indépendants de 8 bits ou 8 de 16 ou... mais aussi 2 flottants DP ou 4 flottants SP
- ⇝ Extension SIMD à processeur classique (i860, , SSE 4.2 du i7, AMD 3Dnow !VMX du Cell et Power, ARM, SPARC...) pour calcul numérique, gestion de chaînes de caractères, (dé)compactage données, cryptographie...
- SSE4.2 à 3,2 GHz : 2 opérations 128 bits/cycle 819 GOP1b/s, 102 GOP8b/s *par cœur* ☺

Instructions multimédia SIMD (II)

- Programmation
  - Très compliquée ☹ car plein d'instructions *ad hoc*, arithmétique à saturation...
  - Assembleur ☹
  - Fonctions intrinsèques en C/C++ (GCC, Intel...)
  - Extensions C/C++ avec rajout de nouveaux types de données vectoriels (GCC, Intel...)
  - Auto-vectorisation (Intel, GCC, outils génériques style PIPS)
  - Utilisation de bibliothèques métiers pré-optimisées
- Disponible dans toute machine ☺

OpenMP

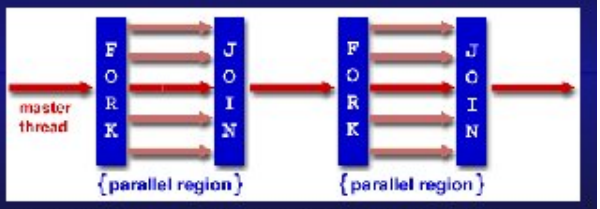
- « Le multithread pour les nuls » ☺
- Vise machines à mémoire partagée
- Sauf si programmation système compliquée, pas besoin de faire de la programmation de thread explicites
- Idée : saupoudrer un programme de directives pour aider compilateur à paralléliser
- Philosophie : `#pragma`tisme avec une certaine élégance esthétique
- ⚠ Si pas de directives, pas de parallélisme exploité (*a priori*)
- ⚠ ⚠ ⚠ Directive ≡ déclaration sur l'honneur
- Langages supportés : Fortran et C/C++

`http://openmp.org`

Modèle d'exécution d'OpenMP (I)

`http://openmp.org/wp/openmp-specifications`

- Exécution parallèle SPMD basée sur le *fork/join*



- Création de thread implicite ou explicite avec des *directives*

Modèle d'exécution d'OpenMP (II)

- Astuce : un programme OpenMP peut être exécuté comme
  - Programme séquentiel
  - Programme parallèle
  - ↝ portabilité, coût de sortie nul ! ☺
- Threads créées dans des sections `parallel` et stoppées à la fin avec une barrière
- Constructions de synchronisation et dans bibliothèque
- Contrôle de l'environnement d'exécution par variables d'environnement et fonctions de bibliothèque

Exemple

```
#pragma omp parallel default(none) \
            shared(n,x,y) private(i)
{
  /* Ceci s'exécute sur plusieurs threads en // */
  #pragma omp for
    for (i=0; i<n; i++)
      // Les itérations sont réparties sur les threads
      x[i] += y[i];
  // Synchronisation implicite ici
} /* End of parallel region */
// Synchronisation implicite ici
```

Task en OpenMP 3.0

- Rajout de la notion de tâches explicites ≡ bout de programme exécuté sur une thread
- Tâche créée dans thread par construction `task` (TBB, Cilk...)

```
#pragma omp parallel
{
  #pragma omp single private(p)
  {
    p = listhead ;
    while (p) {
      #pragma omp task
      {
        process (p);
      }
      p=next (p) ;
    }
  }
}
```

- Extensions en vue (`target`...)

CUDA (I)

- Extension de C++ pour déclarer fonctions s'exécutant sur GPGPU nVidia et à terme multicœurs
- Parallélisme à 2 niveaux : threads dans bloc de threads, pavage de blocs
- Dans bloc de thread : communication par mémoire partagée et synchronisation via `__syncthreads()`

```
__global__ void
add_matrix_gpu(float *a, float *b, float *c, int N) {
  int i=blockIdx.x*blockDim.x+threadIdx.x;
  int j=blockIdx.y*blockDim.y+threadIdx.y;
  int index =i+j*N;

  if( i < N && j < N)
    c[index]=a[index]+b[index];
}

void main() {
```

CUDA (II)

```
12    float ha[N][N], hb[N][N], hc[N][N];
      /* Allocate array on the GPU with cudaMalloc */
14    float *a, *b, *c;
      cudaMalloc((void **) &a, sizeof(float)*N*N);
16    cudaMalloc((void **) &b, sizeof(float)*N*N);
      cudaMalloc((void **) &c, sizeof(float)*N*N);
18
      cudaMemcpy(a, ha, sizeof(float)*N*N, cudaMemcpyHostToDevice);
20    cudaMemcpy(b, hb, sizeof(float)*N*N, cudaMemcpyHostToDevice);

22    // Décrit le pavage des itérations (strip-mining bidimensionnel)
      dim3 dimBlock (blocksize,blocksize);
24    dim3 dimGrid (N/dimBlock.x,N/dimBlock.y);
      add_matrix_gpu<<<dimGrid,dimBlock>>>(a,b,c,N);
26    cudaMemcpy(c, hc, sizeof(float)*N*N, cudaMemcpyDeviceToHost);
    }
```

- Nécessite pas mal de restructuration du code

Brook+ (I)

- Modèle de calcul orienté flux, extension du C pour GPGPU AMD-ATI
- Précompilateur + C++ étendant Brook de Stanford

```
1   #include <stdio.h>
    kernel void sum (float a<>, float b<>, out float c<>) {
3     c = a + b;
    }
5
    int main() {
7     float data[4] = {1.0, 2.0, 3.0, 4.0};
      int i;
9     float a<4>;
      float c<4>;
11    streamRead(a, data);
      sum(a, a, c);
13    streamWrite(c, data);
      for (i = 0; i < 4; i++)
15      printf("%f ", data[i]);
```

Brook+ (II)

```
      printf("\n");
17    return 0;
    }
19
    reduce void sumR (float a<>, reduce float c<>) {
21    c = c + a;    // c += a;
    }
```

- ⇝ Nécessite de modifier structure du code
- ∃ interface de plus bas niveau pour GPGPU AMD-ATI : CaI
- GPUShaderAnalyser pour débogage

RapidMind (I)

- Basé sur C++ & préprocesseur standard
- Modèle de calcul SPMD avec rajouts de types `Program`, `Array` de type `Value`
- Cible *x86*, GPU, Cell

```
1   #include <rapidmind/platform.hpp>
2   #include <rapidmind/shortcuts.hpp>
    using namespace rapidmind;
4
    Program mul_add = BEGIN {
6     In<Value3f> i1, i2;
      Out<Value3f> o;
8     o = i1 + i2*f;
    } END;
10
    int main() {
12    Value1f f;
      rapidmind::init();
```

RapidMind (II)

```
14    const unsigned int w=512,h=512;
      Array<2,Value3f> a(w,h), b(w,h);
16
      a = mul_add(a,b);
18  }
```

⇝ Nécessite de modifier structure du code

HMPP de CAPS Entreprise (I)

- Approche #pragmatique avec compilateur de détourage
- Introduit notion de `codelet` qui sont exécutés en parallèle ± synchrone
- Cible `pthreads`, GPGPU (nVidia CUDA & AMD-ATI Brook+), orthogonal à OpenMP, MPI
- Si pas d'accélérateur disponible, exécution du programme normalement
- #pragma pour optimiser transferts, gérer synchronisme...

HMPP de CAPS Entreprise (II)

```
#pragma hmpp trivial codelet, output=outv
void trivial(int n, float a,
  float *inv, unsigned int N1[1],
  float *outv, unsigned int N3[1]) {
  int i, j;
  for (i = 0 ; i < n ; i++) {
    outv[i] = a * inv[i];
  }
}

enum { TAILLE = 100; };
// Pour préciser la taille des données transférées :
unsigned int dimension_tableau[1] = { TAILLE };

int main() {
  float inc[TAILLE], outv[TAILLE];

#pragma hmpp trivial callsite
  trivial(n, 2.0f, inc, dimension_tableau, outv,
          dimension_tableau);
```

HMPP de CAPS Entreprise (III)

```
}
```

- ~ Pas obligé de trop modifier structure du code
- Possible d'écrire `codelet` à la main pour une architecture donnée (FPGA, MPI...)

Profiter du C99

- Revenir aux C$_{99}$hoses C$_{99}$imples de la vie
- Si programmaC$_{99}$ion en C, C99 ajoute C$_{99}$hoses C$_{99}$ympathiques
  - Tableaux multidimenC$_{99}$ionnels avec taille non statique
    - Évite `malloc()` et C$_{99}$onstrucC$_{99}$ions inutiles
    - Passage de tableaux à taille dynamique en paramètre (comme en Fortran)
    - Évite calculs de linéarisation inutiles `a[i+n*j]`
    - Évite `alloca()`
  - ExtenC$_{99}$ion pour TLS (*Thread-Local Storage*) C$_{99}$ontenant données propres aux threads
  - Nombres C$_{99}$omplexes et booléens
- C$_{99}$implifie le C$_{99}$ode qui devient mieux parallélisable
- Tous C$_{99}$ompilateurs sont C99 sauf... MS Visual C 2008
- Vivement le C++0x qui intègrera entre autre C99

Retour au #pragmatisme (I)

- Permet d'étendre facilement langage existant : \coût d'entrée
- ⚠ Ne pas oublier prépocesseur très pratique pour adapter/générer programmes + portables et + adaptables !
- Problème en « vieux » C : `#pragma` en conflit avec `#` du préprocesseur
- ~ Possible de générer des pragmas en C99 ☺

```
_Pragma ("GCC dependency \"parse.y\"")
```

- Couplé avec génération de chaîne du C dans macro avec `#arg`

```
#define DO_PRAGMA(x) _Pragma (#x)
    [...]
DO_PRAGMA (GCC dependency "parse.y")
```

- Très pra(gma)tique pour générer OpenMP ou HMPP ou... ☺

Retour au #pragmatisme (II)

```
enum {CHUNK_IMAGE = 4};
#define SCHEDULE_IMAGE schedule(guided, CHUNK_IMAGE)
#define OMP_BOUCLE_IMAGE DO_PRAGMA(omp parallel \
                                   for SCHEDULE_IMAGE)
    [...]
    OMP_BOUCLE_IMAGE
    for(i = 0; i < TAILLE_X; i++)
        for(j = 0; j < TAILLE_Y; j++)
            [...]

    [...]
    OMP_BOUCLE_IMAGE
    for(i = 0; i < TAILLE_X; i++)
        for(j = 0; j < TAILLE_Y; j++)
            [...]
```

- ~ Permet de centraliser/factoriser configuration OpenMP
- Permet de viser plusieurs cibles en plus d'OpenMP (HMPP...)

Retour au #pragmatisme (III)

Message subliminal

C$_{99}$ bon pour vous ! ☺

Le retour de la simple précision

- Sur première génération de Cell et GPGPU : seulement flottant simple précision efficace
- A entraîné développement recherche nouveaux algorithmes hybrides
  - Dégrossissage de la solution en flottant SP : rapide
  - Solution finie en flottant DP : lent
- En fait intéressant pour machines en général car SP souvent plus rapide et réduit d'un facteur 2 impact mémoire & cache, double débit SIMD

Le plan

1. Langages
2. Bibliothèques
3. Classes
4. Outils d'instrumentation

Message Passing Interface (MPI)

- Le passage de message pour les nuls ☺
- Modèle : programmation par tâches communicantes
- Bibliothèque de fonctions de communication disponible pour de nombreux langages et systèmes d'exploitation
- Portabilité et nivellement par le bas : programmation de SMP aussi en MPI...
- ∃ implémentations sous ensemble MPI pour Cell
- Ressources
  - `http://en.wikipedia.org/wiki/Message_Passing_Interface`
  - MPI Forum `http://www.mpi-forum.org`
  - MPICH : A Portable Implementation of MPI `http://www-unix.mcs.anl.gov/mpi/mpich/`
  - LAM / MPI Parallel Computing `http://www.mpi.nd.edu/lam/`
  - MPE Graphics–Scalable X11 Graphics in MPI `http://www-fp.mcs.anl.gov/~lusk/papers/mpe/`

Bibliothèques systèmes

- Proches du système d'exploitation
- Contrôle fin de tous les paramètres : priorité, affinité tâche/processeur...
- Posix normalise une API système proche d'Unix. Contient aussi des threads
  - Windows a aussi une API Posix
  - ∃ version libre des *pthreads* pour Windows `http://sourceware.org/pthreads-win32`
  - Assure portabilité multi-OS
- `libnuma` sous Linux pour contrôler comportement mémoire sur machine à *Non Uniform Memory Access* : allocation sur bancs, nœuds, partie de la machine, locale à 1 thread...

Bibliothèques mathématiques

- Souvent bibliothèques constructeurs optimisées pour leur machines
  - Intel Math Kernel Library (MKL)
  - AMD Performance Library (Framewave libre)
  - CuBLAS : sous-traite calculs au GPGPU nVidia
  - ACML : sous-traite calculs au GPGPU AMD/ATI ou CPU
  - ⚠ Bien mettre données au bon endroit et éviter ping-pong entre mémoire hôte et mémoire GPGPU
- FFT : FFTW (*Fastest Fourier Transform in the West*, en C généré par du OCaml)...
- Beaucoup d'algèbre linéaire
  - BLAS (*Basic Linear Algebra Subprograms*) et PBLAS
  - LAPACK (*Linear Algebra PACKage*)
  - ScaLAPACK : version SPMD avec MPI
  - SuperLU : solution directe de gros systèmes creux
  - PETSc (*Portable, Extensible Toolkit for Scientific Computation*) : large spectre, au dessus de MPI

Processeur STI Cell (I)

- DOE/NNSA/LANL United States : Roadrunner - BladeCenter QS22/LS21 Cluster, PowerXCell 8i 3.2 Ghz / Opteron DC 1.8 GHz, Voltaire Infiniband, 122400 cœurs, 2,345 MW, 1+ TFLOPS : TOP1 06/2008
- Démocratisation des machines en pannes : Sony PlayStation 3 avec Cell & 1 cœur en panne (?)
  - PS3 : bonne machine pour tester Cell à 400€ mais seulement 256 Mo (enseignement, tests)
  - Linux dans machine virtuelle
- PowerPC SMT 2 voies + 8 SPE PowerPC SIMD reliés par NoC
- Chaque SPE a 256 Ko mémoire *locale*, pas de MMU

Le retour de la simple précision

- Sur première génération de Cell et GPGPU : seulement flottant simple précision efficace
- A entraîné développement recherche nouveaux algorithmes hybrides
  - Dégrossissage de la solution en flottant SP : rapide
  - Solution finie en flottant DP : lent
- En fait intéressant pour machines en général car SP souvent plus rapide et réduit d'un facteur 2 impact mémoire & cache, double débit SIMD

Le plan

1. Langages
2. Bibliothèques
3. Classes
4. Outils d'instrumentation

Processeur STI Cell (II)

- Communication bas niveau : explicite par transferts DMA via `libspe`, synchronisations, boîtes aux lettres, contrôle des threads sur SPE...
  - Le retour des *overlays* ! Car programmes doivent aussi loger dans mémoire locale...
  - Attention aux plantages lors de phase de mise au point
  - Demande contrôle très fin chorégraphie des données (pavage, ≈*out-of-core*...)
- Environnement Linux bien intégré
- ∃ environnements de plus haut niveau
  - OpenMP avec simulation mémoire globale partagée (VSDM)
  - Paralléliseur automatique et VSDM

  Performances en fonction de l'application
- Simulateur intégré à Eclipse pour développer

Le plan

1. Langages
2. Bibliothèques
3. Classes
4. Outils d'instrumentation

Classes

- Programmation objet qui utilise classes parallélisées ⇝ programmes parallèles !
- ∃ Versions parallélisées de la STL, souvent logiciel libre (PaSTeL de l'INRIA...)
- Boost contient aussi des choses liées au parallélisme (threads) et au calcul (uBLAS...)

  `http://en.wikipedia.org/wiki/Boost_library`
- Souvent bibliothèques métiers (bases de données, interfaces graphiques...) déjà parallélisées

EcoLib

- Bibliothèque pour GPGPU développée par GPU-Tech
  - Statistiques
  - Nombres aléatoires
  - Algèbre linéaire
  - FFT et traitement d'image
- Emballage dans classes C++ ⇝ transparent pour programmeur utilisant bons objets

Thread Building Blocks (TBB) (I)

TBB (Intel Threading Building Blocks)
http://en.wikipedia.org/wiki/TBB

- Bibliothèque pour programmation générique (templates à la STL)
- Versions libre et commerciale
- Algorithmes (`for`, `reduce`, `pipeline`, `scan`...), conteneurs, allocation mémoire , exclusion mutuelle, opérations atomiques, mesure de temps, ordonnanceur
- Répartition travail par vol de travail entre tâches
- Orthogonal OpenMP & MPI

Thread Building Blocks (TBB) (II)

```
1  class ApplyFoo {
     float* const my_a;
2  public:
     ApplyFoo(float* a) : my_a(a) {};
3    void operator() (const tbb::blocked_range<size_t>&r) const {
       for (size_t i=r.begin(); i != r.end(); ++i)
7        Foo (my_a[i]);
     }
9  }

11 void ParallelApplyFoo(float a[], size_t n) {
     tbb::parallel_for (
13     tbb::blocked_range<size_t>(0,n),
       ApplyFoo(a),
15     tbb::auto_partitioner()
     );
17 }
```

- Nécessite profonde restructuration de code si pas dans esprit STL

Ct : C/C++ for Throughput Computing (I)

- Classes C++ développés par Intel pour son projet Terascale
- Rajoute conteneur de collection de données `TVEC`
- Parallélisme de données pour manipuler simplement grosses quantités de données
- Opérations parallélisme de données comme langages Nesl, HyperC...
  - Fonctions vectorielles élément/élément
  - Communications collectives : réductions, scan
  - Permutations : scatter/gather, (dé)compaction, décalages, partitions, papillon...

Ct : C/C++ for Throughput Computing (II)

```
1  TVEC<F32> colorConvert(TVEC<F32> rchannel, TVEC<F32> gchannel,
                          TVEC<F32> bchannel, TVEC<F32> achannel,
3                         F32 a0, F32 a1, F32 a2, F32 a3) {
     return (rchannel*a0 + gchannel*a1 + bchannel*a2 + achannel*a3);
5  }

7  TVEC<F32> Convolve2D3x3(TVEC<F32> pixels, TVEC<F32> kernel) {
     TVEC<F32> respixels;
9    // directions[m][n] is a constant TVEC of size
     // with values (m-1, n-1)
11   respixels += shiftPermute(pixels, directions[0][0])*kernel[0][0]
     respixels += shiftPermute(pixels, directions[0][1])*kernel[0][1]
13   respixels += shiftPermute(pixels, directions[0][2])*kernel[0][2]
     respixels += shiftPermute(pixels, directions[1][0])*kernel[1][0]
15   respixels += pixels*kernel[1][1];
     respixels += shiftPermute(pixels, directions[1][2])*kernel[1][2]
17   respixels += shiftPermute(pixels, directions[2][0])*kernel[2][0]
     respixels += shiftPermute(pixels, directions[2][1])*kernel[2][1]
19   respixels += shiftPermute(pixels, directions[2][2])*kernel[2][2]
     return respixels
```

Ct : C/C++ for Throughput Computing (III)

```
21 }
```

- Graphe de tâche pour exécuter différents calculs parallèles avec parallélisme de contrôle
- Nécessite restructuration de code pour passer en parallélisme de données

Le plan

1. Langages
2. Bibliothèques
3. Classes
4. Outils & infrastructures

Analyse de performance

- Utilise technique d'échantillonnage, instrumentation et/ou compteurs performances
- `gprof` libre
- PAPItools libre
- AMD CodeAnalyst
- Intel VTune
- Intel Thread Profiler (intégré à VTune)

Outils de débogage vérification de threads

- Vérification à l'exécution de propriétés d'accès aux données
- Helgrind `http://valgrind.org`
  - Outil libre de la suite Valgrind (instrumentation dynamique d'exécutable) pour Linux
  - `valgrind --tool=helgrind ./pi_SPMD`
- Intel Thread checker
  - Instrumentation du binaire
  - Frontal graphique pour Windows et Linux
- ⚠ Outils dynamiques donc ne vérifient que si code est exécuté...
  ⇝ Bons tests de couvertures nécessaires

Traducteurs

- Facilitent la vie du programmeur
  - Vectoriseur: transforme un programme « séquentiel » en programme avec des instructions vectorielles
  - Paralléliseur: transforme un programme « séquentiel » en programme(s) pour machine parallèle
- Évite le parallélisme explicite mais pas toujours efficace
  - Difficile dans cas général et C
  - Hors de portée en C++ général
  - Nécessite de l'aide ⇝ directives (constructeurs, HPF, OpenMP...)
- Tendance actuelle
  - Recyclage dans outils GCC, Icc, traducteur style Cuda, Brook+, HMPP
  - Mélange automatisation et `#pragma`

Des ordinateurs et des humains...

- Parallélisme : compliqué ⇝ supporte mal la médiocrité ☺
- Ne pas oublier l'outil principal : l'être humain !
- Selon NSF, 60% informaticiens sans diplôme informatique
  - Besoin de solides bases en informatique pour maîtriser parallélisme et hétérogénéité architecturale
  - Besoin de solides bases généralistes pour maîtriser les domaines applicatifs
- Réflexion au sein d'ORAP dans « groupe formation »
- ∃ Tutoriels SuperComputing, ISC/SuperComputing Europe
- Centres de calculs français, GENCI...
- Services de formation continue grandes écoles/universités
- Entreprises spécialisées en ~~HPC~~informatique normale (Oxalya, CAPS Entreprise, GPU-Tech, Aneo, HPC Project, constructeurs...) pour formation et consultance

Réingénierie pour le parallélisme

« Reengineering for Parallelism : An Entry Point into PLPP (Pattern Language for Parallel Programming) for Legacy Applications », Berna L. Massingill, Timothy G. Mattson, Beverly A. Sanders
`http://www.cise.ufl.edu/research/ParallelPatterns/plop2005.pdf`

- Guide de recettes pratiques lorsqu'on part d'un vieux programme...
- ... ou qu'on trouve que cela aide de concevoir d'abord un programme séquentiel (mais ⚠...)
- 4 espaces de conception à traverser en partant du problème, contexte et des utilisateurs
  - « Trouver de la concurrence » (*Finding Concurrency*)
  - « Structure algorithmique » (*Algorithm Structure*)
  - « Structure de support » (*Supporting Structures*)
  - « Mécanismes d'implémentation » (*Implementation Mechanisms*)

Conclusion (I)

- Contraintes matérielles ⇝ Impossible d'échapper au parallélisme désormais
- Programmation *beaucoup* plus compliquée
- Environnements et méthodes de développement jeunes comparé à programmation « classique » (séquentielle)
- Standards ? Souvent avancés par constructeurs de matériel
- ⚠ Coût d'entrée
- ⚠ ⚠ Coût de sortie ☺
- Avant d'attaquer parallélisme, optimiser programme séquentiel (algorithmique, codage, outils)
- Parallélisation : opportunité pour bon dépoussiérage de code (réorganisation, modernisation...) ⇝ amélioration aussi code d'origine...

Conclusion (II)

- ... mais nécessite excellentes compétences applicatives et informatiques
- « CPU-GPU : la convergence ? » ou *real politik* ?
  - Non si programmation sauvage ☹
    - Nœuds SMP programmés en OpenMP en interne
    - Appels explicites à des accélérateurs (Cell, GPGPU...)
    - Interconnexion des SMP via communication style MPI
  - Oui si on s'aligne sur modèle de programmation minimal (SIMD, SPMD, stream-computing) ou utilisation de bibliothèques métiers. Cf MPI sur SMP...
- Approche `#pragma`tique partant du code séquentiel
  - OpenMP permet d'exploiter en douceur multicœurs
  - HMPP permet d'exploiter en douceur hétérogénéité
  - Penser au préprocesseur pour portabilité
- Retour écriture à distance (RDMA) comme primitive de communication

Conclusion (III)

- Convergence autour de Linux dans haut de gamme
- Retour virtualisation (machines, cartes graphiques...)
- Retour des outils source à source pour capitaliser sur outils constructeurs
- Environnements logiciels toujours derrière ☺

Bibliographie (I)

- « Principles of Concurrent and Distributed Programming, 2/E », M. Ben-Ari. Addison-Wesley (2006)
- « The Art of Multiprocessor Programming », Maurice Herlihy, Nir Shavit. Morgan Kaufmann (2008)
- « Using OpenMP : Portable Shared Memory Parallel Programming », Barbara Chapman, Gabriele Jost, Ruud van der Pas, David J. Kuck. The MIT Press (2007)
- « Intel Threading Building Blocks : Outfitting C++ for Multi-core Processor Parallelism », James Reinders. O'Reilly Media, Inc. (2007)
- « Patterns for Parallel Programming », Timothy G. Mattson, Beverly A. Sanders, Berna L. Massingill. Addison-Wesley Professional (2004)
- Ian Foster, Designing and Building Parallel Programs, Addison-Wesley (1995)

Bibliographie (II)

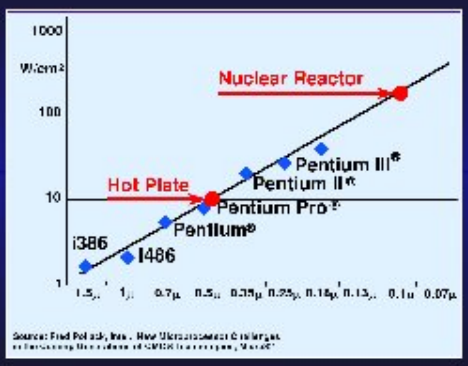
- `http://www-unix.mcs.anl.gov/dbpp/` « Designing and Building Parallel Programs », Ian Foster
- « An Introduction to Parallel Computing : Design and Analysis of Algorithms », Ananth Grama, George Karypis, Vipin Kumar, Anshul Gupta. Addison-Wesley (2003)
- « Foundations of Multithreaded, Parallel, and Distributed Programming », Gregory R. Andrews. Addison-Wesley (1999)
- « An Introduction to Parallel Algorithms », Joseph Ja'Ja. Addison-Wesley (1992)
- Selim Akl, The Design and Analysis of Parallel Algorithms, Prentice Hall (1989)
- « High Performance Compilers for Parallel Computing », Michael Wolfe
- « Supercompilers for Parallel and Vector Computers », Hans Zima & Barbara Chapman

Bibliographie (III)

- « Scheduling and Automatic Parallelization », Alain Darte, Yves Robert & Frédéric Vivien

[Barnes et al., 1968]Barnes, G. H., Brown, R. M., Kato, M., Kuck, D. J., Slotnick, D. L., and Stokes, R. A. (1968). The Illiac iv computer. *IEEE Transactions on Computers*, C-17(8):746–757.

Densité de puissance



La nouvelle donne du renouveau informatique (I)

« The Landscape of Parallel Computing Research : A View from Berkeley », Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams and Katherine A. Yelick. EECS Department University of California, Berkeley Technical Report UCB/EECS-2006-183, December 18, 2006

http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf

Révision des bonnes vieilles hypothèses (*conventional wisdoms*)...

- Old CW : Power is free, but transistors are expensive.
  New CW is the "Power wall" : Power is expensive, but transistors are "free". That is, we can put more transistors on a chip than we have the power to turn on.

[...]

La nouvelle donne du renouveau informatique (II)

- Old CW : Monolithic uniprocessors in silicon are reliable internally, with errors occurring only at the pins.
  New CW : As chips drop below 65 nm feature sizes, they will have high soft and hard error rates. [Borkar 2005] [Mukherjee et al 2005]

[...]

- Old CW : Performance improvements yield both lower latency and higher bandwidth.
  New CW : Across many technologies, bandwidth improves by at least the square of the improvement in latency. [Patterson 2004]
- Old CW : Multiply is slow, but load and store is fast.
  New CW is the "Memory wall" [Wulf and McKee 1995] : Load and store is slow, but multiply is fast. Modern microprocessors can take 200 clocks to access Dynamic Random Access Memory (DRAM), but even floating-point multiplies may take only four clock cycles.

La nouvelle donne du renouveau informatique (III)

- Old CW : We can reveal more instruction-level parallelism (ILP) via compilers and architecture innovation. Examples from the past include branch prediction, out-of-order execution, speculation, and Very Long Instruction Word systems.
  New CW is the "ILP wall" : There are diminishing returns on finding more ILP. [Hennessy and Patterson 2007]
- Old CW : Uniprocessor performance doubles every 18 months.
  New CW is Power Wall + Memory Wall + ILP Wall = Brick Wall. Figure 2 plots processor performance for almost 30 years. In 2006, performance is a factor of three below the traditional doubling every 18 months that we enjoyed between 1986 and 2002. The doubling of uniprocessor performance may now take 5 years.

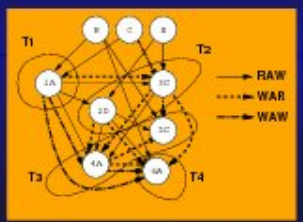
La nouvelle donne du renouveau informatique (IV)

- Old CW : Don't bother parallelizing your application, as you can just wait a little while and run it on a much faster sequential computer.
  New CW : It will be a very long wait for a faster sequential computer (see above).
- Old CW : Increasing clock frequency is the primary method of improving processor performance.
  New CW : Increasing parallelism is the primary method of improving processor performance.
- Old CW : Less than linear scaling for a multiprocessor application is failure.
  New CW : Given the switch to parallel computing, any speedup via parallelism is a success.

Base de la parallélisation (I)

Conditions d'indépendance de 2 parties i et j (BERNSTEIN, 1966) : Exécuter le plus de choses en parallèles sans enfreindre de dépendance : parcourir au plus tôt le graphe de dépendance topologiquement

```
1: A = B + C
2: D = A*E
3: C = A - E
4: A = D - A*C
5: C = D - B
6: A = B/D
```



⇝ Parallélisme maximal de 1,5 sans renommer variables

Base de la parallélisation (II)

Problèmes :

- Des millions d'instructions ☹
- Matériel fini mais de plus en plus complexe
- Boucles : graphe cyclique
- Dépendances de contrôle (branchements)
- Planification en temps non polynomial ☹
- ⇝ Bonnes heuristiques...

Compromis entre

- Parallélisme statique (compilation)
- Dynamique à la demande (à l'exécution)

Extraire du parallélisme (I)

Exemple de calcul de polynômes de vecteurs (livre « Initiation au parallélisme, concepts, architectures et algorithmes », Marc GENGLER, Stéphane UBÉDA & Frédéric DESPREZ)

pour $i = 0$ à $n-1$ faire
$\quad vv[i] = a + b.v[i] + c.v[i]^2 + d.v[i]^3 + e.v[i]^4 + f.v[i]^5 + g.v[i]^6$
fin pour

Calcul avec parallélisme de donnée (typique SIMD) ≡ faire en parallèle la même chose sur des données différentes :

pour $i = 0$ à $n-1$ faire en parallèle
$\quad vv[i] = a + b.v[i] + c.v[i]^2 + d.v[i]^3 + e.v[i]^4 + f.v[i]^5 + g.v[i]^6$
fin pour

Extraire du parallélisme (II)

Découpage en tâches (typique MIMD) ≡ faire des choses différentes sur des données différentes :

pour $i = 0$ à $n-1$ faire
$\quad$ tâches parallèles
$\quad\quad x = a + b.v[i] + c.v[i]^2 + d.v[i]^3$
$\quad$ |
$\quad\quad y = e. + f.v[i] + g.v[i]^2$
$\quad$ |
$\quad\quad z = v[i]^4$
$\quad$ fin tâches parallèles
$\quad vv[i] = x + z.y$
fin pour

Extraire du parallélisme (III)

Pipeline (typique systolique) ≡ travail à la chaîne :

7 étages de pipeline (7 processeurs) traitant 1 flux de plusieurs données.

Type de parallélisme (I)

Parallélisme de données :

- Régularité des données
- Même calcul à des données distinctes

Parallélisme de contrôle :

- Fait des choses différentes

Parallélisme de flux : pipeline

- Régularité des données
- Chaque donnée subit séquence de traitements

Grain du parallélisme (taille des tâches élémentaires) entre en ligne de compte dans choix : surcoûts de communications, accès mémoire, démarrage/arrêt des processeurs...

π avec parallel for et reduction (I)

La solution

```
1  #include <omp.h>
   #include <stdio.h>
3  const int num_steps = 100000;

5  int  main () {
     double pi;
7    double sum = 0.0;
     double step = 1.0/(double) num_steps;
9  #pragma omp parallel for reduction(+:sum)
     for (int i = 1; i <= num_steps; i++) {
11     double x = (i-0.5)*step;
       sum += 4.0/(1.0 + x*x);
13   }
     pi = step * sum;
15   return printf("pi_=_%f\n", pi);
   }
```

Construction parallel sections (I)

```
1  void XAXIS();
2  void YAXIS();
   void ZAXIS();
4  void a9()
   {
6    #pragma omp parallel sections
     {
8      #pragma omp section
         XAXIS();
10     #pragma omp section
         YAXIS();
12     #pragma omp section
         ZAXIS();
14   }
   }
```


Patron de conception pour OpenMP

Il y a des threads...

- OpenMP : modèle à mémoire partagée
  - Faire communiquer threads par variables partagées
- ∃ situations de compétition
  - Mettre des synchronisations en cas de conflit
- Synchronisations chères
  - Changer placement des données pour minimiser synchronisations

Utilisation des compilateurs (I)

En 2008 : (presque) tout le monde en OpenMP 2.5

- GNU GCC 4.3
  - `gcc -fopenmp -std=c99`
  - dans GNU/make : `CFLAGS='-fopenmp -std=c99'` ou dans environnement `export CFLAGS='-fopenmp -std=c99'`
  - Multi-cibles et OS
- Compilateur Intel 10.1
  - `icl /Qopenmp /Qstd=c99`
  - Avec *Thread Checker* : `/Qtcheck` (incompatible avec *Thread Profiler*
- Compilateurs PGI
  - `-mp omp`
- MicroSoft Visual C++ 2008
  - OpenMP 2.0, ne semble pas C99 ☹
  - `http://msdn.microsoft.com/en-us/library/h24szzh9.aspx` pour avoir la bonne version avec OpenMP...

Utilisation des compilateurs (II)

- Option `/openmp`
- Il faut mettre `#include <omp.h>` même si pas besoin

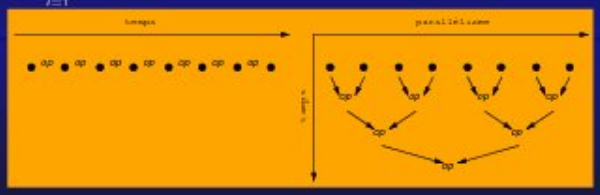
OpenMP sur mémoire distribuée

- ∃ travaux pour étendre cibles du langage à des machines à mémoire partagée
- OpenMP pour Cell par IBM
- Cluster OpenMP d'Intel pour Linux utilisant mémoire virtuelle (VSDM)
  `http://softwarecommunity.intel.com/articles/eng/3811.htm`
- STEP : OpenMP pour MPI par Institut TÉLÉCOM/IT SudParis

Opérations parallèles préfixes (I)

- Exemple d'algorithmique parallèle très en vogue années 1980

$$r = \bigoplus_{i=1}^{n} a_i$$



Opérations parallèles préfixes (II)

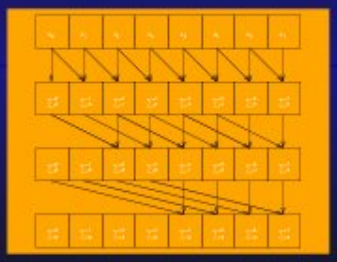
	Séquentiel	Parallèle
Temps	$n-1$	$\lceil \log_2 n \rceil$
Opérateurs	1	$\lfloor \frac{n}{2} \rfloor$ ou $n-1$
Efficacité	1	$\frac{n-1}{\lfloor \frac{n}{2} \rfloor \lceil \log_2 n \rceil}$ ou $\frac{1}{\lceil \log_2 n \rceil}$

Parallélisme : gâcher utile !

Opération préfixe parallèle (scan) (I)

Autre algorithmique parallèle classique [Barnes et al., 1968] :

$\forall i \in [0.n-1], \quad S_i = \left(\overset{i}{\underset{j=0}{\boxplus}} x_j\right)$



Opération préfixe parallèle (scan) (II)

Si réutilisation des opérateurs :

	Séquentiel	Parallèle
Temps	$n-1$	$\lceil \log_2 n \rceil$
Opérateurs	1	$n-1$
Efficacité	1	$\frac{1}{\lceil \log_2 n \rceil}$

- ⚠ non associativité des opérations flottantes... Changement de l'ordre d'évaluation ⇝ changement du résultat
- Permet d'exprimer de manière élégante la solution de nombreux problèmes (tris, selection, composantes connexes, matrices creuses, inversion matrices tridiagonales...)
- Existe en version segmentée

Transactions (I)

- Bloc de travail cohérent sur des données cohérentes
- Primitive d'exclusion mutuelle à beaucoup plus gros grain
- Protège tout une série d'opération, éventuellement distribuées
- Exemple de transaction bancaire
  - Enlever argent d'un compte dans une base de données
  - Verser même valeur sur autre compte dans autre base de données
  - Doit être conservatif : pas de création ou destruction d'argent quelque part
- Ça passe ou ça casse... totalement
- Des transactions à l'ACID
  - Atomicité : doit s'effectuer en tout ou rien
  - Cohérence : le système doit rester cohérent dans tous les cas d'échec ou de réussite
  - Isolation : transaction dans monde isolé, seule à avoir accès aux données qu'elle modifie

Transactions (II)

  - Durabilité : état stable durable à la fin de transaction (réussite ou échec)
- Utile pour paralléliser et distribuer du travail si application en rapport

Mémoire transactionnelle (I)

- Transposition du concept de transaction à un morceau de programme concurrent
  - Soit tout le code s'est exécuté sans aucun conflit en écriture avec des données manipulées par d'autres threads et toutes les écritures mémoires sont validées
  - Soit il y a eu un conflit et tous les accès sont annulés, donc la transaction. Pourra être relancée plus tard
- Évite la programmation basée sur des verrous pour protéger des bouts de code très éloignés
- Pour qu'un code soit atomique, il suffit de le mettre dans une transaction !
- Mémoire transactionnelle matérielle (HTM) (1986) : matériel pour gérer différentes versions de la mémoire, ⚠ entrées-sorties...

Mémoire transactionnelle (II)

- Mémoire transactionnelle logicielle (STM) (1995) : on note au préalable tous les accès en écriture pour faire un commit plus tard
- Rajout de primitives dans langage pour exprimer transactions

Encore dans le monde de la recherche...

Espace de conception « trouver de la concurrence » (

- Permet de structurer problème pour exposer concurrence exploitable
- Travail niveau algorithmique de haut niveau pour exposer concurrence potentielle
- Quelques patrons de conceptions possibles dans cet espace :
  - Patrons de décomposition de problèmes en morceaux concurrents
    - Décomposition en tâches : comment un problème peut-il être décomposé en tâches qui s'exécutent de manière concurrente ?
    - Décomposition des données : comment décomposer données du problème en unités qui peuvent être traitée relativement indépendamment ?

Espace de conception « trouver de la concurrence » (

- Patrons d'analyse de dépendances : regroupent tâches et analyse leur dépendances
  - Groupe des tâches : comment regrouper les tâches d'un problème pour simplifier la gestion de leur dépendances ?
  - Ordonnancement des tâches : comment ordonner des groupes de tâches (provenant d'une décomposition d'un problème et d'un regroupement des tâches) pour satisfaire les inter-dépendances ?
  - Partage des données : Comment à partir d'une décomposition des données et en tâches partager des données entre tâches ?
- Évaluation de la conception : est-ce que les résultats de la phase de décomposition et d'analyse de dépendances est suffisamment bonne pour passer à l'espace de conception suivant (structure algorithmique) ou est-ce qu'on réitère conception dans cet espace ?

Espace de conception « structure algorithmique » (I)

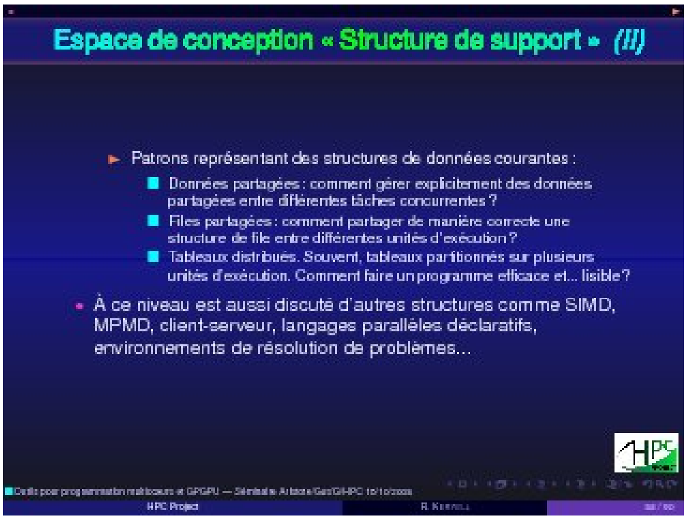
- Restructuration des algorithmes pour exploiter la concurrence potentielle obtenue dans l'espace précédent
- Patrons possibles de stratégies pour exploiter la concurrence :
  - Patrons pour applications centrées sur une organisation en tâche
    - Parallélisme de tâche : comment organiser un algorithme en un collection de tâches à exécution concurrente ?
    - Diviser pour régner : comment exploiter concurrence potentielle dans le cas d'un problème formulé avec stratégie « diviser pour régner » ?
  - Patrons pour applications centrées sur organisation par décomposition de données
    - Décomposition géométrique : comment organiser un algorithme autour de structures de données mises à jour par morceau de manière concurrente ?
    - Données récursives : dans le cas d'opérations sur une structure de donnée récursive (liste, arbre, graphe...) qui semblent séquentielles, comment réaliser ces opérations en parallèle ?
  - Patrons pour applications orientées flot de données

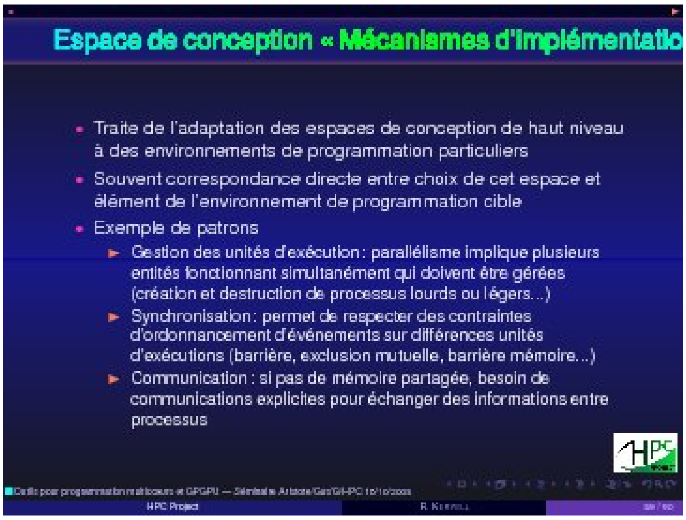
Espace de conception « structure algorithmique » (II)

- Pipeline : si application peut être vue comme un flux de données à travers une série d'étapes de calcul, comment exploiter cette concurrence ?
- Coordination par événements : si application décomposée en groupe de tâches semi-indépendantes interagissant de manière irrégulière dépendant des données (donc contraintes de dépendances entre tâches aussi...), comment réaliser cette interaction pour avoir du parallélisme ?

Espace de conception « Structure de support » (I)

- Étape intermédiaire entre description algorithmique et implémentation
- Traite de la programmation mais en restant à haut niveau
- Exemples de patrons de conception :
  - Patrons représentant les approches structurant les programmes :
    - SPMD : problèmes liés aux interaction de différentes unités d'exécution. Comment structurer programmes pour gérer au mieux interactions et faciliter intégration dans programme global ?
    - Ferme de travail : Comment organiser un programme conçu avec un besoin de distribuer de manière dynamique du travail à des travailleurs ?
    - Parallélisme de boucles : comment traduire en programme parallèle un programme séquentiel dominé par de gros nids de boucles
    - Fork/Join : Si programme avec nombre de tâches concurrentes qui varient avec relations complexes entre elles, comment construire programme parallèle avec tâches dynamiques ?


Espace de conception « Structure de support » (II)
Patrons représentant des structures de données courantes :
Données partagées : comment gérer explicitement des données partagées entre différentes tâches concurrentes ?
Files partagées : comment partager de manière correcte une structure de file entre différentes unités d'exécution ?
Tableaux distribués. Souvent, tableaux partitionnés sur plusieurs unités d'exécution. Comment faire un programme efficace et... lisible ?
À ce niveau est aussi discuté d'autres structures comme SIMD, MPMD, client-serveur, langages parallèles déclaratifs, environnements de résolution de problèmes...
HPC Project
R. Keryell


Espace de conception « Mécanismes d'implémentatio
Traite de l'adaptation des espaces de conception de haut niveau à des environnements de programmation particuliers
Souvent correspondance directe entre choix de cet espace et élément de l'environnement de programmation cible
Exemple de patrons
Gestion des unités d'exécution : parallélisme implique plusieurs entités fonctionnant simultanément qui doivent être gérées (création et destruction de processus lourds ou légers...)
Synchronisation : permet de respecter des contraintes d'ordonnancement d'événements sur différences unités d'exécutions (barrière, exclusion mutuelle, barrière mémoire...)
Communication : si pas de mémoire partagée, besoin de communications explicites pour échanger des informations entre processus
HPC Project
R. Keryell


Des ordinateurs et des humains (I)
Implique développement harmonieux
Matériel
Modèles de programmation
Langages parallèles
Compilation
Algorithmique adaptée
Compatibilité et portabilité logicielles
Utilisateur raisonne séquentiellement sur des concepts...
Communauté & langage recherche compilation happée par mode GRID computing...
Pas que des applications high-tech...
¡ Depuis 50 ans le matériel attend le logiciel !
HPC Project
R. Keryell


Table des matières
La crise logicielle 2
Évolution logicielle 3
Programmeurs inconscients des processeurs... 4
Fin de l'augmentation des performances séquentielles... 5
Extraire du parallélisme dans les applications 6
Programmation acceptable 7
Hétérogénéité à plusieurs dimensions 8
Cycle de développement 9
Le plan 10
Instructions multimédia SIMD 11
OpenMP 13
Modèle d'exécution d'OpenMP 14
Exemple 16
Task en OpenMP 3.0 17
CUDA 18
Brook+ 20
RapidMind 22
HMPP de CAPS Entreprise 24
Profiter de Cee 27
Retour au Symposium 28
Le retour de la simple précision 31
Le plan 32
Message Passing Interface (MPI) 33
Bibliothèques systèmes 34
Bibliothèques mathématiques 35
Processeur STI Cell 36
Le plan 38
Classes 39
EcoLib 40
Thread Building Blocks (TBB) 41
Ct : C/C++ for Throughput Computing 43
Le plan 46
Analyse de performance 47
Outils de débogage vérification de threads 48
Traducteurs 49
Des ordinateurs et des humains... 50
Pélerinage pour le parallélisme 51
Conclusion 52
Bibliographie 55
Densité de puissance 58
La nouvelle donne du renouveau informatique 60
Bases de la parallélisation 64
Extraire du parallélisme 66
Type de parallélisme 68
... avec parallel for et reduction 70
Construction parallel sections 71
Patron de conception pour OpenMP 72
Utilisation des compilateurs 73
OpenMP sur mémoire distribuée 75
Opérations parallèles préfixes 76
Opération préfixe parallèle (scan) 78
Transactions 80
Mémoire transactionnelle 82
Espace de conception « trouver de la concurrence » 84
Espace de conception « structure algorithmique » 85
Espace de conception « Structure de support » 88
Espace de conception « Mécanismes d'implémentation » 90
Des ordinateurs et des humains 91
Vous êtes ici ! 92
HPC Project
R. Keryell

2.10 Accélération de la reverse time migration (RTM) à l'aide de GPGPU, où en sommes nous ?

Henri Calandra (TOTAL)

Présentation conjointe Total et CAPS Auteurs :
- Total, Henri Calandra, Rached Abdelkhalek
- CAPS, Stéphane Bihan, Romain Dolbeau, Georges-Emmanuel Moulard

Très adaptée à l'imagerie de structures géologiques complexes, la méthode du reverse time migration (RTM), introduite au début des années 1980, connaît aujourd'hui dans l'industrie pétrolière un fort regain d'intérêt pour l'imagerie sismique profondeur en raison de la puissance de calcul dorénavant disponible.

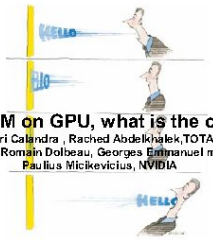
Toutefois, cette famille de méthodes d'imagerie, basée sur la résolution de l'équation des ondes complète, nécessite des ressources de calculs qui aujourd'hui encore requièrent l'utilisation de clusters de plusieurs dizaines de milliers de cœurs de processeurs généralistes.

La mise à disposition récente de nouvelles générations de cartes graphiques ouvre de nouvelles directions de recherches algorithmiques pour nos applications scientifiques. En exploitant au mieux le parallélisme naturel de nos algorithmes et en tirant partie du parallélisme massif de données offert par les cartes graphiques, nous devrions alors obtenir une accélération de calcul qui permette de résoudre des problèmes non calculables avec les technologies multi-cœurs actuelles

Toutefois l'utilisation des cartes graphiques comme technologie accélératrice pose le problème de leur intégration dans une architecture HPC ainsi que celui de la programmation des algorithmes.

Nous présentons l'état de nos travaux sur l'utilisation des cartes graphiques NVIDIA pour la résolution de la RTM. Après avoir rapidement exposé la méthode, nous décrivons notre première mise en œuvre et discutons de son impact algorithmique et des solutions apportées. Nous présentons également l'utilisation de l'environnement de programmation HMPP développé par CAPS qui, à l'aide de directives de compilation, donne au développeur un niveau élevé d'abstraction.

Time Reversal Cartoon.



Accelerating RTM on GPU, what is the current status?

Henri Calandra , Rached Abdelkhalek,TOTAL
Stephane bihan, Romain Dolbeau, Georges Emmanuel moulard, CAPS
Paulius Micikevicius, NVIDIA

From Scientific American, November 1999 (M. Fink).

CAPS TOTAL

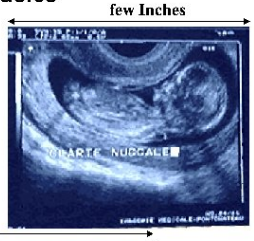
Outline

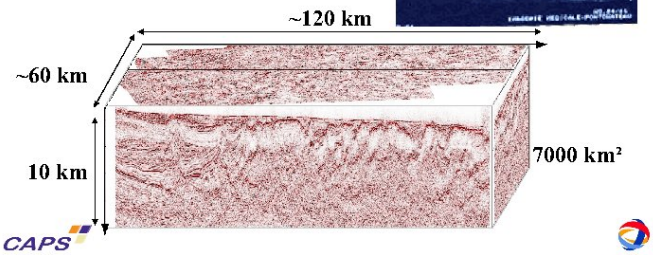
- Introduction
- RTM in a nutshell
- Implementation, current status
- Model programming
- Conclusions, what's next

CAPS TOTAL

Seismic reflection basics

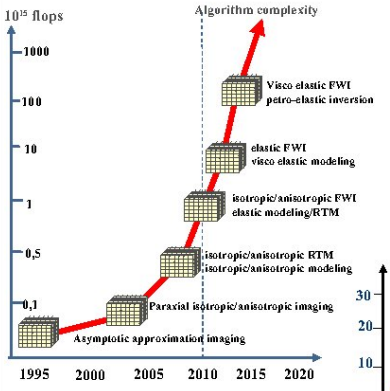
Seismic exploration: ~ ultrasound

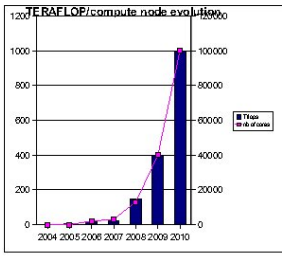


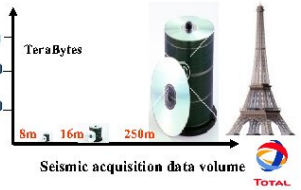


CAPS TOTAL

Codes & Computing Efforts: Some Figures



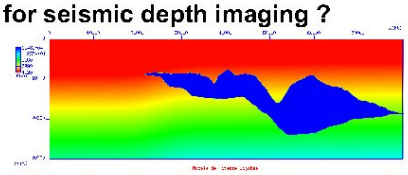




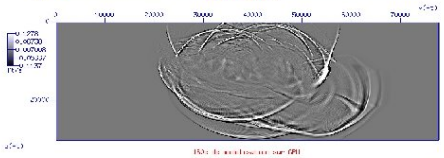
CAPS TOTAL

GPU for seismic depth imaging ?

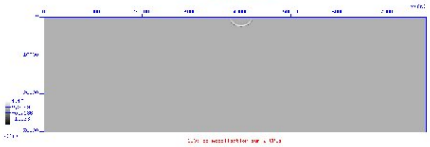
•Velocity model:
6400*1500 grid size



• GPU: 3.6s
(12808 time steps)



•2 CPUs: 0.5s
(1778 time steps).



CAPS TOTAL

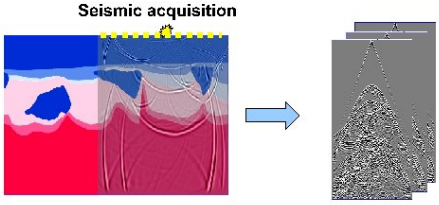
Questions

- Is it possible to take advantage of GPUs for seismic depth imaging?
- What programming model and what programming language?
- What about communications between host and GPU, best configuration?
- Can we extrapolate the performances obtained on a single GPU to a full system: Cluster ?
- What about building an industrial application (RTM) on a GPU cluster based system ?

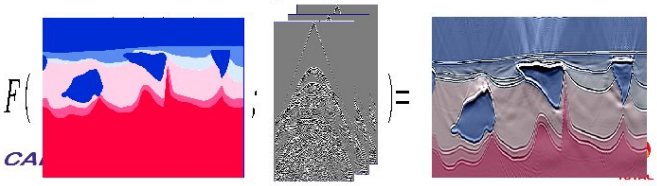
CAPS TOTAL

RTM in a nutshell

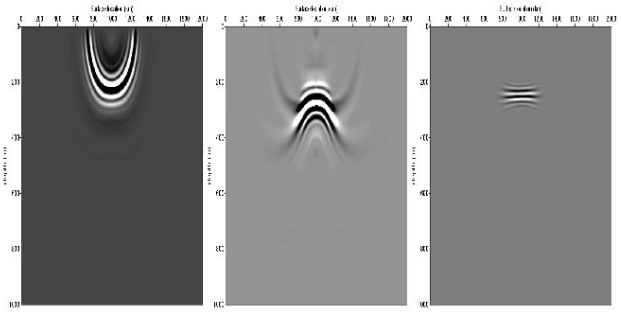
Seismic acquisition



Pre Stack Depth Migration



RTM in a nutshell



Propagate Source (Forward in time: F)

back propagate receivers (Backward in time: F')

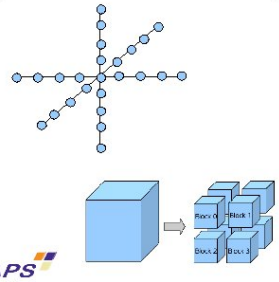
imaging condition (multiplication: FF')

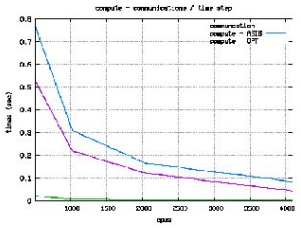
RTM in a nutshell

- F and its adjoint are the Acoustic Wave Equation

$$\left[\frac{1}{c^2}\frac{\partial^2}{\partial t^2}-\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2}\right)\right]P(x,y,z,t)=0.$$

- Explicit Finite difference scheme implementation (2-8)



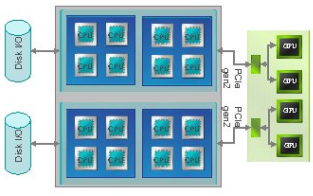


RTM in a nutshell

- RTM:
  - Propagators implementation is massively parallel
    - Domain decomposition, message passing based (MPI)
    - Data parallel within a sub-domain (OpenMP, multithreaded)
  - Two types of communications
    - Ghost nodes between sub-domains (6 faces in 3D): size =O(L*N²)
    - Snapshot store (forward phase), read (backward phase):
      - 2 strategies:
        - Full wavefield storage: size= O(N³) every seismic time step
        - Boundaries storage: size= O(L*N²)
  - Remarks
    - Full wavefield strategy is bandwith and IO intensive
    - Boundaries is less bandwidth and less IO intensive but requires to solve 1 more forward problem

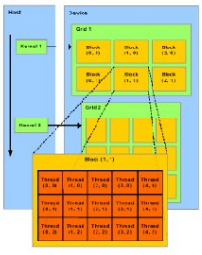
Implementation on GPU

- Objectives:
  - Take advantage of the data parallelism capabilities of GPUs to accelerate the Finite difference kernels
  - Optimize the communication between host and GPU to reduce the bottleneck of the PCI express interconnect
  - Explore different configurations host-GPU
- Machine configuration:
  - Tesla S1070
    - 4 GB memory per GPU
    - Dual PCIe gen2 (2x6.4GB/s)
  - Host server
    - Twin Xeon-based bi-socket quadcore
    - 2x26GB memory
    - 2.5GHz



Kernel optimization

- Take advantage of data parallelism
  - Mainly based on efficiently using memory hierarchy, with different approaches:
    - Use textures,
    - Use simple 3D cache blocking,
    - Replace by a 2D cache, sliding on the 3rd dimension
- The third approach allows for a larger slice and consequently better re-use
  - The extra loop doesn't penalize performance

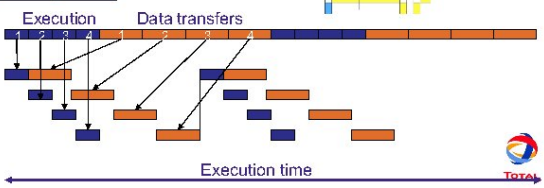


Number of read accesses per data point	
RTM 2D	4,25
RTM 3D with texture	29
RTM 3D with 3D shared memory accesses	7,5
RTM 3D with sliding 2D shared memory	4

CPU-GPU Data transfer optimization

- Take advantage of asynchronous communications to overlap data transfers
  - First implementation: 75% of elapsed time spent on PCIe communications
  - Take advantage of :
    - PCIe Gen2 bandwidth
    - DMA accessible "pinned memory"
    - Asynchronous communications

%	synch	asynch
FWD	32	15
BWD	41	25



Execution time

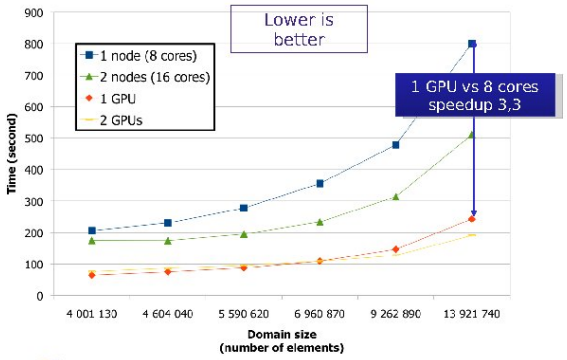
CAPS TOTAL

Model programming

- Decided to not use C CUDA directly
- We use HMPP
  - Express parallelism and communications in Fortran source code
    - Direct integration into Fortran Code through the use of directives "à la" and complementary to OpenMP: the legacy of the code is preserved
  - Standardized interface with the hardware implementation ("codelets")
    - Easier to integrate hand-tuned code into existing codelets
    - HMPP directive syntax
      - Simplify hardware resource allocation, communications
  - Programming the accelerators
    - Insulate hardware-specific computations
    - Use hardware vendor SDK
    - Domain specific code generators

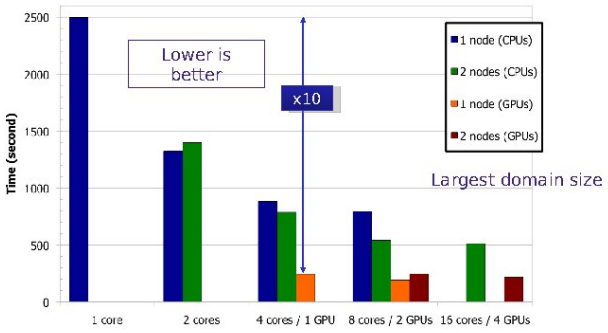
CAPS TOTAL

CPU-GPU global optimization



CAPS TOTAL

CPU-GPU global optimization



CAPS TOTAL

2.11 Implémentation d'un solveur creux sur GPU

Thomas Guignon (IFP)

Afin d'optimiser et de déterminer le plan de développement d'un gisement pétrolier, les companies pétrolières ont besoin de prédire l'historique de production d'un gisement en fonction de divers scénarios d'implémentation des puits.

Dans ce but, elles utilisent des simulateurs de réservoirs qui modélisent numériquement l'exploitation d'un champ pétrolier à partir des équations d'écoulement des fluides dans un milieu poreux.

L'IFP poursuit des travaux de recherche dans le domaine de la simulation de réservoir dont la thématique s'inscrit dans une de ses priorités stratégiques visant à repousser les limites de l'exploration pétrolière.

La résolution numérique des équations d'écoulements passe par l'utilisation intensive de solveurs itératifs creux de type biCGStab couplés à des préconditionneurs iLU ou multigrilles.

Le temps passé dans ces solveurs pouvant représenter jusqu'à 70% du temps total de simulation, l'IFP dans le cadre du projet ANR PARA (Parallelisme et Amélioration du Rendement des Applications) explore depuis 2 ans l'utilisations d'accélérateurs graphiques afin de déporter l'exécution des solveurs et ainsi réduire les temps d'exécution.

Nous présenterons l'état de nos travaux sur l'utilisation d'accélérateurs graphiques tout en présentant dans un premier temps les méthodes numériques utilisées, la méthodologie suivie afin d'implémenter le solveur creux efficacement sur un GPU puis nous présenterons les résultats obtenus.

Ces derniers font apparaitre sur des cartes graphiques de dernière génération de type nVIDIA 8800 GTX ou C1060 des gains allant de 20x en simple précision à 8x en double précision par rapport à un coeur d'un processeur de dernière génération.

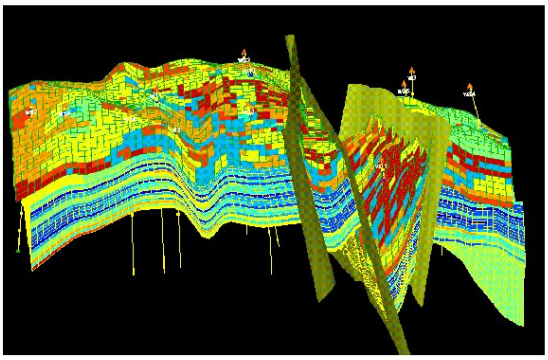
Solveur Linéaire pour la simulation de réservoir sur GPU (NVIDIA+CUDA)

Thomas GUIGNON
Stéphane REQUENA
IFP

Simulation de réservoir (1)

- **Réservoir géologique:**
  - Milieu poreux: aquifère, huile, gaz
  - Puits
- **Simuler le comportement en exploitation**
  - Prévision de production
  - Scénario de développement (forage, injection ...)
- **Écoulement multiphasique en milieu poreux:**
  - Loi de Darcy (vitesse d'écoulement, pression porosité)
  - Lois de conservation
  - Discrétisation volumes finis
  - Géométrie structurée, non structurée, CPG

Simulation de réservoir (2)



Simulation de réservoir (3)

- **Discrétisation: système non linéaire / dt**
  - Newton
  - Solveur linéaire itératif (BiCGStab préconditionné): Ax=b
  - +80 % du temps de simulation passé dans le solveur.
- **A: système creux non structuré (blocs 3x3)**
  - Format CSR
  - Lié à la structure du maillage (graphe matrice ≈ connectivité du maillage)
  - Équations puits.
- **Irrégularité dans la matrice:**
  - plan de faille,
  - mailles mortes,
  - Disparition de couches, LGR

Solveur linéaire

- **BiCGStab Préconditionné:**
  - Méthode de Krylov itérative.
  - Produit matrice creuse vecteur (+40%)
  - Préconditionneur (+40%)
  - Produits scalaires, combinaisons linéaires (-20%)
- **Strategie GPU:**
  - L'ensemble du calcul itératif est déporté:
    - Entrées: A et b, M (A précoditionnée)
    - Sortie: x (et résidu)
  - Le calcul de M reste sur le CPU

Produit matrice creuse vecteur CUDA

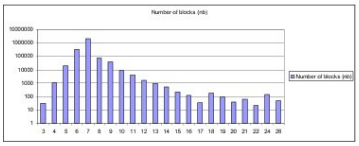
- **Contraintes architecturale: exécution SIMD**
  - Tous les threads doivent faire le même calcul.
- **Implantation simple du matvec impossible (1 thread par ligne):**
  - Nombre variable d'éléments (blocs) non nuls par ligne.
  - Threads inoccupés pour les lignes courtes.
  - Alignement des données.
- **Rechercher la régularité: permuter la matrice pour grouper les lignes de même largeur.**
- **1 kernel par largeur.**
- **Kernels autogénérés à la compilation avec des templates.**

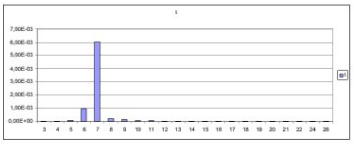
Performances: Matrice GCS2K (370K mailles)

- **matrice issue du simulateur de réservoir Pumaflow:**
  - cas réel non structuré, CPG + failles, double milieu,
  - modèle blackoil,
  - N=370982, 3x3, NB=2557714,
  - pas d'équations puits.
- **Performances CPU double précision: entre 300 et 500 MFLOPS (core 2 DUO 1.8Ghz, Opteron 2Ghz, gcc3, Intel9)**
  - sse2 mal utilisé.
- **GPU simple précision (8800 GTX): 5.750 GFLOPS, Mem BW: 17.26E9 B/S.**
- **accélération > x10**

Performances: Matrice GCS2K Analyse

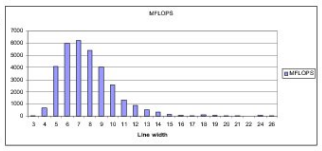
- **Majorité blocs et temps pour W= 6 et 7.**

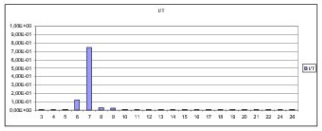




Performances: Matrice GCS2K Analyse

- **W=6,7 ~80% temps de calcul**
- **W=7 perf = 6.2 GFLOPS**
- **perfs > 3 GFLOPS pour W=5,8,9**
- **perfs équivalentes CPU pour un petits nombre de blocs.**
- **pénalité constante pour les plus petits (coût exécution kernel).**





Performances: Répartition des coûts.

- **Analyse détaillée pour W=7:**
  - **test sans lecture de X ni écriture de Y ni réduction complète => uniquement lecture de coefficients + multiplication par une constante littérale**
    => perf équiv: 15.38 GFLOPS (Mem BW: 30.8 E9 B/S)
  - **+ réduction complète:**
    => perf équiv: 15.22 GLFOPS
  - **+ écriture Y:**
    => perf: 13.53 GLOPS
  - **+ lecture X (avec indirection):**
    => perf: 6.2 GFLOPS +50% du temps

Performances: pistes d'améliorations (1)

- **plus de parallélisme pour la réduction:**
  - pas nécessaire pour des blocs 3x3 ou plus?
  - attention aux cas 1x1 ou 2x2
- **écriture de Y?**
- **lecture des coefficients de A: 30,8 GB/S**
  - max 8800 GTX=86.4 GB/S
  - **~36 % Mem BW max.**
- **Lecture de X (indirection)**
  - non continuité des accès à X.
  - +50% temps.

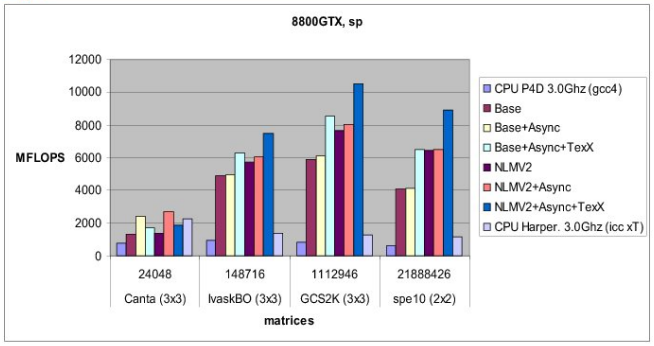
Performances: pistes d'améliorations (2)

- **Pas de conflits mémoire partagée.**
- **Appel kernel Synchrone / Asynchrone:**
  - superpositions appel kernels/calculs
  - utile pour les petits blocs de threads (pénalité appel)
  - 5.750 GFLOPS -> 5.99 GFLOPS.
- **lecture des coefficients:**
  - alignement ok ? oui
  - lecture float4 à la place de float ?
  - autres... ??
- **le gros problème: lecture de X avec indirections:**
  - permutation de la matrice: faire apparaître un maximum d'accès contigus ?
  - cache de texture (localité 2D)?

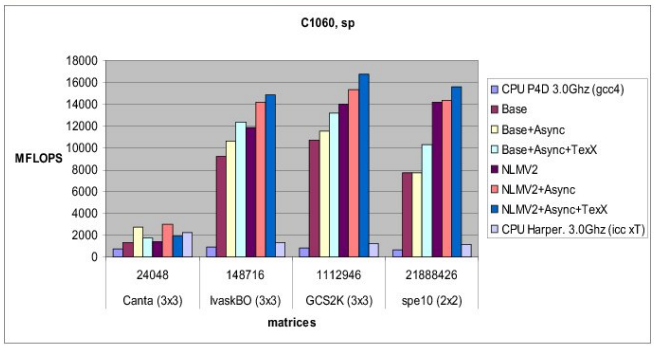
Intervention CAPS

- **Profiling (CUDA Profiler)**
  - Perte de performances due à utilisation de la mémoire locale (tableau local au kernel)
- **Portage et test double précision sur C1060**
  - diminution d'environ 50% des perfs (données x2).
- **Utilisation du cache de texture pour X**
  - améliore les perfs dans certains cas.
  - pas de double précision.

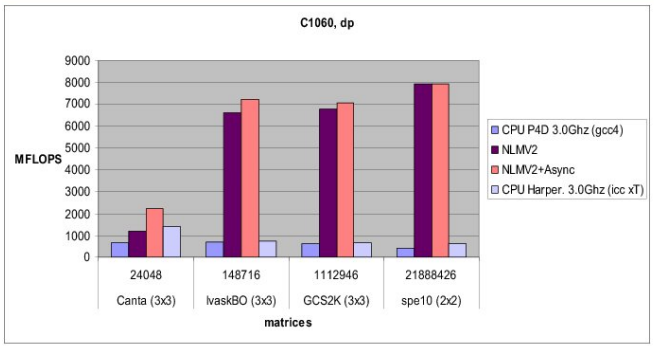
Performances: Matrices completes (8800 GTX CUDA 2.0, sp)



Performances: Matrices completes (C1060 CUDA 2.0, sp)



Performances: Matrices completes (C1060 CUDA 2.0, dp)



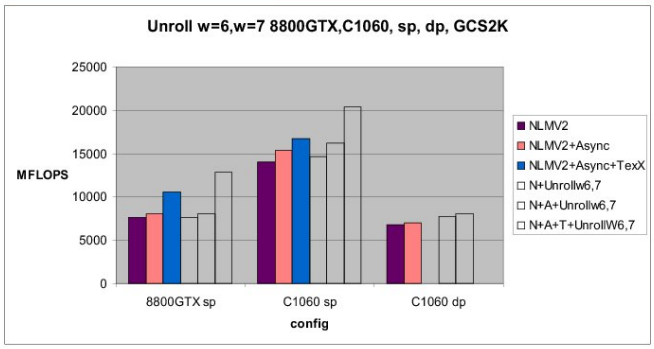
Performances: synthèse

- **Ne pas utiliser de mémoire locale**
  - **8800GTX: +7% → +58% (+9% → +37% TexX)**
  - **C1060: +4% → +85% (+10% → +50% TexX)**
- **Asynchronisme: petites matrices/nb de largeurs importants.**
  - **gains marginaux pour les autres cas.**
- **Cache de texture (localité des accès à X)**
  - **dégradation pour n petit. ??**
  - **8800 GTX: +20% → +60%**
  - **C1060: +4% → +30%; contrôleur mémoire amélioré**
  - **pas de double précision**
- **Double précision: données x2, perfs/2**

Performances: Améliorations?

- **GCS2K, bloc w=7, pas de lecture X, ni reduction, ni ecriture Y**
  - **perf. équiv. 30 GFLOPS (C1060 sp) → 60 GB/s (lecture des coéfs)**
  - **60 GB/s ≈ 60% max (102 GB/s)**
- **Déroulement manuel des boucles:**
  - **moins d'arithmétique entière.**
  - **moins de registres.**
  - **perf. équiv. 42 GFLOPS (+40%) → 84 GB/s (≈ 82% max)**
- **Test matrice complète**
  - **Unroll pour w=6 et w=7**
  - **0%→+22% (TexX) 8800GTX**
  - **+5%→+15% C1060.**

Performances: Loop Unroll

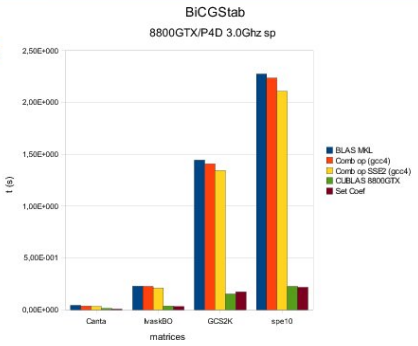


BiCGStab GPU

- **Sans préconditionneur:**
  - **75% à 98% passé dans le produit matrice vecteur**
- **CPU:**
  - **Intel MKL BLAS**
  - **Fusion de boucles**
  - **Fusion de boucles + SSE2**
- **GPU:**
  - **CUBLAS: pb C1060, test 8800GTX sp**
- **On compare le temps de 10 it. CPU avec 10 it. GPU + transfert b,x et transfert et permutation des coefficients de A.**

BiCGStab GPU

- **Transfert x,b: 3% → 9% temps it.**
- **Transfert coef A: équiv. 10 it**
- **Acc. GPU/CPU: 1.93 → 4.9**



Conclusions

- **BiCGStab GPU**
  - **Compétitif avec transfert des données.**
- **Produit matrice vecteur**
  - **Optimisation GPU importante; autres pistes?**
  - **Optimisation CPU? Comparaison équitable…**

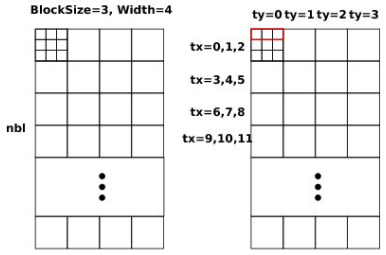
Travaux en cours, Perspectives

- **BiCGStab:**
  - **Pb CUBLAS C1060: dev. kernel avec fusion de boucles**
- **Préconditionneur ILU0 (résolution):**
  - **Stage 2009**
- **Produit matrice creuse vecteur:**
  - **Déroulement automatique des boucles pour tous les kernel**
  - **Cache de texture et double précision?**
  - **Matrice puits.**
  - **Nouveau noyau: -threads +occupation.**
- **MultiGPU**
- **MPI+GPU…**

Remerciements

- **CAPS: conseils**
- **NVIDIA: cartes TESLA**

Organisation du calcul: Structure logique d'un block de coéficients.



- Boucle la plus interne, on suppose que l'indirection du vecteur X à déjà été faite:

```
for(k=0;k<BlockSize;k++){
    Ytmp +=
    A(tx,ty*BlockSize+k)*Xtmp(..
    .);
}
```

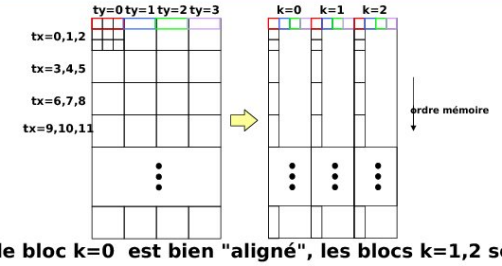
- A est lue directement depuis la mémoire principale, Xtmp depuis la mémoire partagée.

- 1 block nbl*Width == 1 block de threads nbl*BlockSize*Width
- Répartition identique pour les indices de colonnes
  - 1 indice par bloc k*k

Organisation du calcul: Structure mémoire d'un bloc de coéficients (1).

- Ordre des accès mémoire par les threads:
  - tableau (logique): T[n]
  - bloc de threads (nx,ny)
  - kernel thread (tx,ty) utilise T(ty*nx+tx):
    - thread 0 utilise t[0], thread 1 t[1] ....
- Il faut:
  - T respecte une contrainte d'alignement précise dépendante du type de T et du device.
  - Les éléments de T doivent être continus en mémoire.
- Pour notre cas:
  - A(0,0), A(1,0), A(2,0), ..., A(nbl*BlockSize-1,0), A(0,BlockSize), ..., A(nbl*BlockSize-1,BlockSize), ... continuent en mémoire.

Organisation du calcul: Structure mémoire d'un bloc de coéficients (2)



- si le bloc k=0 est bien "aligné", les blocs k=1,2 sont bien "alignés" si nbl*BlockSize est un multiple de la taille d'alignement (pour le type numérique considéré).
- nbl = 16,32, préférence pour 16.
- nbl = 1,2,4,8 pour Width trop grand (mémoire partagée) ou trop de threads (>512)

2.12 Application du GPU à la calibration de modèles sur processus stochastiques pour la finance

Laurent Domingos (BNP Parisbas)

Les modèles de processus stochastiques utilisés en finance sont toujours plus complexes et coûteux en terme de temps de simulation et de calibration. Les causes en sont multiples : appels nombreux à des fonctions non primitives (plusieurs cycles), manipulation de gros tableaux, simulation de nombreuses trajectoires “monte carlo”. Nous montrons combien le GPU nous permet de fournir un modèle rapide et réactif et comparons aux solutions existantes.

BNP PARIBAS
CORPORATE & INVESTMENT BANKING

A GPU application to quantitative finance

Domingos Laurent
Equity Quantitative research
BNPParibas

Duguet Florent
ANEO

22 October 2008

BNP PARIBAS
CORPORATE & INVESTMENT BANKING

Quantitative finance issues

22 October 2008

Quantitative finance issues

- In order to valuate and manage our portfolios, and in order to make prices for clients, we need to model the market underlyings (stocks, rates, volatilities, correlations, etc…)
- In researchers mind, computing power is not a problem anymore.
- When designing models researchers only aim at model's accuracy, not computation time.
- Then, models are more and more complex and time consuming.
- Power consumption is a major factor of the computation platform TCO.

BNP PARIBAS
CORPORATE & INVESTMENT BANKING

Our numerical equations

Firstname Secondname
Position/job title

Firstname Secondname
Position/job title

22 October 2008

Stochastic differential equation on spot

$$\Delta \ln S_t = [\mu_t - \chi(\sigma_t)]\Delta t + \sigma_t \Delta X_t$$

$$\Delta X_t = \mathrm{H}_{\Delta t}(\varepsilon_t)$$

Stochastic differential equation on volatility

$$\sigma_t = F[\phi(S_t, t), (Y_i)_{i=1..M}]$$

$$\Delta Y_{i,t} = A_t^i \Delta t + B_t^i \Delta X_t^V$$

$$\Delta X_t^V = \mathrm{H}_{\Delta t}^V(\varepsilon_t^V)$$

BNP PARIBAS
CORPORATE & INVESTMENT BANKING

Problem description

22 October 2008

Problem

- We have two models described as above
- We would like to fit spot distribution of the first model with the second

  one by changing its $\phi(S_t, t)$
- $\phi(S_t, t)$ is a grid (300x100 points)

Algorithm complexity

- For each time t_i we need to solve an implicit equation :

$$\{\varphi(S_i, t_j), i = 1..N\}$$
$$= \Gamma[\{\varphi(S_i, t_j), i = 1..N\}, distribution(Spot, Vol)]$$

BNP PARIBAS
CORPORATE & INVESTMENT BANKING

Why do we need GPU ?

Firstname Secondname
Position/job title

Firstname Secondname
Position/job title

22 October 2008

Why do we need GPU ?

- Complexity of our equations
  - Exponentials in diffusion
  - Table lookups
- Monte Carlo Simulation
  - RNG, Normal Distribution sampling (Box-Müller)
- Complexity of Gamma
  - Sorting for (Spot,Vol) distribution usage
  - Cumulative normal distribution evaluations
  - Exponentials
  - Table lookups
- Actual cost of our fit
  - 1 hour to fit 10years for one underlying
  - 500 cores during 15h -> 1MWh per day

Why do we need GPU ?

- **Algorithm is CPU bound !**
- Faster Intrinsics
  - Exponentials, Logs, trigonometric functions are more efficient on GPU than on CPU
- Better memory usage
  - Cache is better used amongst processors
  - Memory access latency is hidden by the thread count (several thousands)
  - Local cache yields fast table lookups (for our small tables)
- Actual cost of our GPU fit
  - Roughly 1 minute on single GT100 for the full fit
  - Total is 10h GPU => 3kWh (including CPU host)
  - Ratio = 330 !!!!!

BNP PARIBAS
Corporate & Investment Banking

Main issues with GPU

22 October 2008

Main issues with GPU

- We need to re-implement already existent algorithms entirely
  - CPU application is not written for high parallelisation
  - Some algorithms needed to be specifically rewritten (global sort)
- Problem with Floats
  - Actual precision of floats leads to small rewrites of function evaluations
  - Small errors actually propagate
- Is double really a solution ?
  - 52 bits is better than 23
  - Today, computing in doubles is much more expensive than computing in floats
    - For CPU bound computation, MAD is 8 times more expensive for NVIDIA
    - EXP ? LOG ?

BNP PARIBAS
Corporate & Investment Banking

GPU in production

Firstname Secondname
Position/job title
Firstname Secondname
Position/job title

22 October 2008

GPU in production

If (and only if) we solve our simple precision problems,

or if precision becomes acceptable for our numerical applications :

- Short term
  - GPU will replace all our CPU used for our model calibration
- Middle term
  - Intraday fits can be done by traders/pricers
- Long term
  - More and more components of our pricing tools to GPU
  - Pricing/Hedging with GPU

2.13 Utilisation des GPU et des ondelettes pour le calcul des structures électronique

Thierry Deutsch (CEA)

Thierry Deutsch CEA-Grenoble, INAC, Luigi Genovese, ESRF, Grenoble, Matthieu Ospici, Bull (doctorant), Jean-François Méhaut, UJF/INRIA/LIG (Laboratoire Informatique de Grenoble)

Les codes de calcul de structure électronique, fondés sur la mécanique quantique, se retrouvent dans de nombreux domaines basés sur la structure ou la propriété des systèmes atomiques : en science des matériaux, en chimie, dans le domaine pharmaceutique et aussi dans le domaine médical couplé dans ce cas à des méthodes multi-échelles. Ces codes sont très gourmands en ressources de calcul. Récemment, dans le cadre du projet européen BigDFT, nous avons développé un code performant, massivement parallèle fondée sur les ondelettes et leurs propriétés de multi-résolution.

Dans le contexte d'une collaboration pluridisciplinaire entre physique et informatique, nous avons développé une version hybride du code s'exécutant sur CPU et sur GPU. Au cours de cet exposé, nous montrerons l'approche que nous avons suivie et les performances obtenues.

Une démonstration sera, si c'est possible, réalisée soit sur un noeud GPU du futur calculateur du GENCI au CCRT, soit sur un serveur hybride CPU-GPU de l'INRIA.

Séminaire Aristote-Genci-CAPS

ÉCOLE POLYTECHNIQUE – PALAISEAU

Utilisation des GPU et des ondelettes pour le calcul des structures électronique

Thierry Deutsch, Luigi Genovese, Matthieu Ospici, Jean-François Méhaut

CEA, ESRF, LIG

16 Octobre 2008

Outline

1. Electronic structure calculations
   - Ab initio methods
   - BigDFT code
2. High Performance Computing
   - Massively parallel
   - Main operations
   - GPU
   - Scaling
3. Perspectives
   - Linear Scaling calculations
   - Conclusions

Ab initio calculations with DFT

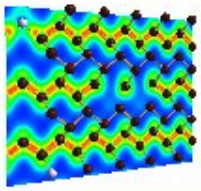
Several advantages

- ✔ Ab initio: No adjustable parameters
- ✔ DFT: Quantum mechanical (fundamental) treatment

Main limitations

- ✘ The exact XC functional is unknown
- ✘ Requires high computer power, limited to few hundreds atoms in most cases

Wide range of applications: nanoscience, biology, materials



Performing a DFT calculation (KS formalism)

Find a set of orthonormal orbitals $\Psi_i(\mathbf{r})$ that minimizes:

$$E = -\frac{1}{2}\sum_{i=1}^{N/2}\int \Psi_i^*(\mathbf{r})\nabla^2\Psi_i(\mathbf{r})\,d\mathbf{r} + \frac{1}{2}\int\frac{\rho(\mathbf{r})\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|}\,d\mathbf{r}'d\mathbf{r} + E_{xc}^{LDA}[\rho(\mathbf{r})] + \int V_{ext}(\mathbf{r})\rho(\mathbf{r})\,d\mathbf{r}$$

with $\rho(\mathbf{r}) = \sum_i \Psi_i^*(\mathbf{r})\Psi_i(\mathbf{r})$

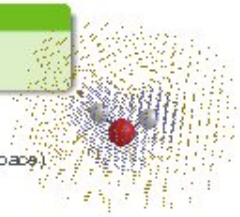
(Kohn-Sham) DFT "Ingredients"

- An XC potential, functional of the density
- An (iterative) algorithm for finding the wavefunctions $|\Psi_i\rangle$
- A basis set for expressing the $\Psi_i(\mathbf{r})$
- A (good) computer...

A DFT code based on Daubechies wavelets

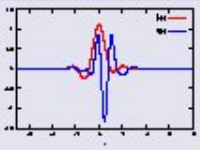
Wavelets

A basis with optimal properties for expanding localised information



- Localised in real space
- Smooth (localised in Fourier space)
- (Bi-)Orthogonal basis
- Multi-resolution basis
- Adaptive
- Systematic

Daubechies Wavelets



From early 80's

Applied in several domains

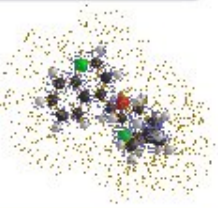
Interesting properties for DFT

A basis for nanosciences: the BigDFT project

STREP European project: BigDFT(2005-2008)

Four partners, 15 contributors:
CEA-INAC Grenoble, U. Basel, U. Louvain-la-Neuve, U. Kiel

Aim: To develop an ab-initio DFT code based on Daubechies Wavelets, to be *integrated in ABINIT*, distributed freely (GNU-GPL license)



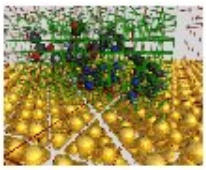
L. Genovese, A. Neelov, S. Goedecker, T. Deutsch, *et al.*,
"Daubechies wavelets as a basis set for density functional pseudopotential calculations",
J. Chem. Phys. 129, 014109 (2008)

A Versatile formalism

Flexible Boundary Conditions

Wavelet properties allows for

- Direct approach (non-iterative)
- Explicit, high precision



Clean and precise treatment of the electrostatic

Several applications:

- Impurities, dopants
- Surface properties (adsorbtion, catalysis)
- *Charged* defects, in bulk or in surfaces

These calculation may be performed at high accuracy thanks to the systematicity of the basis set

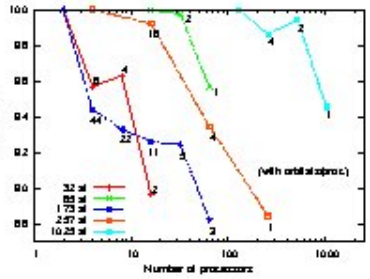
High Performance Computing

Localisation & Orthogonality → Data locality

Principal code operations can be intensively optimised

Efficiency of the order of 90%, up to thousands of processors

Ready for hybrid platforms (GPU), 30 to 60 times faster (preliminary)



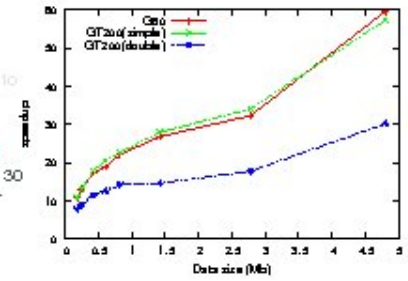
High Performance Computing

Localisation & Orthogonality → Data locality

Principal code operations can be intensively optimised

Efficiency of the order of 90%, up to thousands of processors

Ready for hybrid platforms (GPU), 30 to 60 times faster (preliminary)



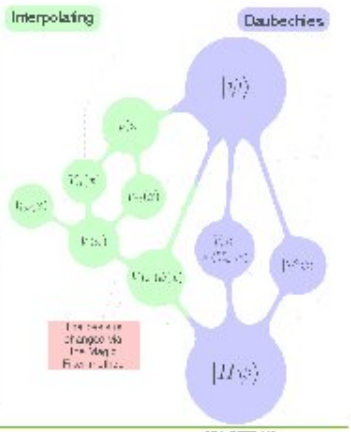
Operations performed

Different numerical operations performed:

- For each wavefunction (Hamiltonian application)
- Between wavefunctions (Linear algebra)

Numerical operations

- Convolutions with short filters
- BLAS routines
- FFT (Poisson Solver)



Separable convolutions

We must calculate

$$F(i_1,i_2,i_3) = \sum_{j_1,j_2,j_3=0}^{L} h_{j_1} h_{j_2} h_{j_3} G(i_1-j_1, i_2-j_2, i_3-j_3)$$

$$= \sum_{j_1=0}^{L} h_{j_1} \sum_{j_2=0}^{L} h_{j_2} \sum_{j_3=0}^{L} h_{j_3} G(i_1-j_1, i_2-j_2, i_3-j_3)$$

Application of three successive operations

1. $A_3(i_3,i_1,i_2) = \sum_j h_j G(i_1,i_2,i_3-j) \quad \forall i_1,i_2;$
2. $A_2(i_2,i_3,i_1) = \sum_j h_j A_3(i_3,i_1,i_2-j) \quad \forall i_3,i_1;$
3. $F(i_1,i_2,i_3) = \sum_j h_j A_2(i_2,i_3,i_1-j) \quad \forall i_2,i_3.$

Main routine: Convolution + transposition

$$F(i,a) = \sum_j h_j G(a,i-j) \quad \forall a;$$

GPU calculation

Main routine: Convolution + transposition

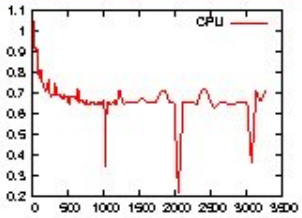
$$F(i,a) = \sum_j h_j G(a,i-j) \quad \forall a;$$

Pros and cons for GPU computation:

- ✔ Set of 1 D convolutions. Easy to parallelize
  Each multiprocessor treats a group of lines. Bank conflicts avoided.
- ✔ Short filters
  Loop unrolling, constant memory (less registers)
- ✘ Transposition: coalesced write, uncoalesced read
  May benefit from 2D texture fetching
- ✘ Boundary conditions: periodic/isolated at the borders
  Different strategy at the borders

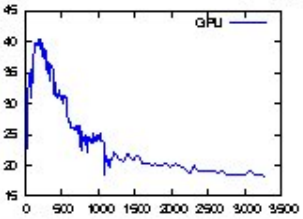
Optimal for hiding memory latency by arithmetics

Performances (Gflops *vs.* dimension))



CPU: from 1 to 0.7 Gflops

CPU suffers from cache-trashing problems

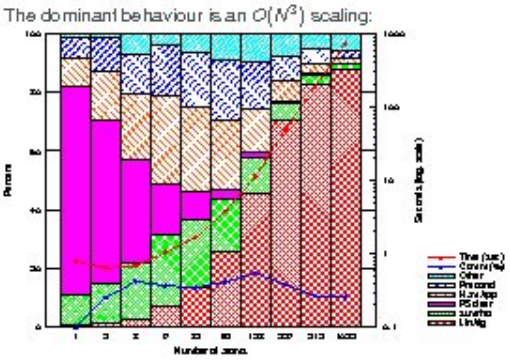


GPU: around 50 times faster

Optimal performances for common sizes in BigDFT!

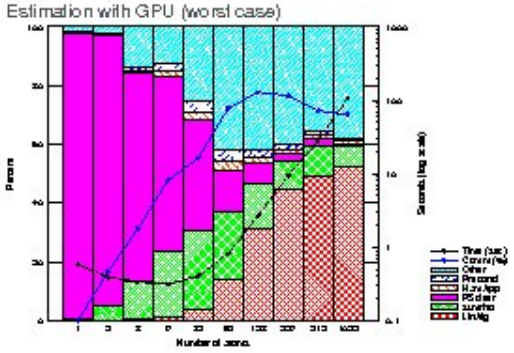
Cubic scaling code

The dominant behaviour is an $O(N^3)$ scaling:



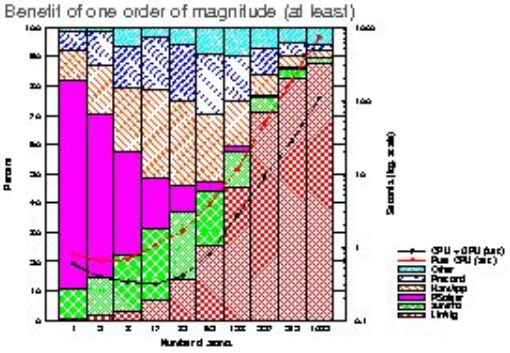
Overall code behaviour

Estimation with GPU (worst case)



Cubic scaling code

Benefit of one order of magnitude (at least)



Perspective: Wavelets for $O(N)$ methods

Benefit from the localisation property

The locality of the KS orbitals can be enforced

- Wannier or minimal basis approach (localised functions)
- Whole domain separated in localisation regions

Paper Submitted

M. Rayson, S. Goedecker, A.Neelov, A. Bergman, L.G. , *et al.*, submitted to J.Chem. Phys.

Conclusions

An unexplored domain ... up to now

(Daubechies) Wavelets represent a new formalism for ab initio methods, with many advantages:

- Adaptive → Efficiency
- Systematic → Precision
- Real-space based → Flexibility
- Localised → Parallelisation, short convolutions, $O(N)$

...to be continued

Daubechies Wavelets: Perspectives

Powerful tool for electronic structure calculations

- ✔ Ready to petascaling ($O(N)$, ...)
- ✔ Lots of applications (nanosciences, biology, ...)

2.14 Bioinformatique et calcul haute-performance

Mathieu Giraud (LIFL)

Les données bioinformatiques issues des séquençages sont toujours en croissance exponentielle. Aux génomes de référence s'ajoutent maintenant les variations individuelles tout comme les méta-génomes (séquences d'organismes prélevés dans un même milieu).

Nous présenterons dans cet exposé quelques traitements parallèles sur ces données : certains se contentent d'un parallélisme à gros grain, facile à mettre en oeuvre sur cluster ou sur GPU, d'autres demandent des analyses plus fines pour traiter au mieux les différents accès mémoire. La comparaison intensive de séquences est souvent au coeur de ces algorithmes, mais d'autres défis surgissent des dernières technologies, notamment avec les séquenceurs de dernière génération. Nous parlerons aussi d'une méthode générique pour certains problèmes de programmation dynamique.

Bioinformatique et calcul haute-performance

Mathieu Giraud
mathieu.giraud@lifl.fr

CNRS, LIFL, Université Lille 1
INRIA Lille Nord-Europe

Séminaire Aristote, École Polytechnique, 16 octobre 2008

1

Séquences génomiques : ADN et protéines

2

Séquences génomiques : ADN et protéines

- ADN : $\Sigma_4 = \{A, C, G, T\}$
- Protéines : $\Sigma_{20} = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$
- Code génétique : $\Sigma_4 \times \Sigma_4 \times \Sigma_4 \longrightarrow \Sigma_{20}$
- Bases de données : EMBL (octobre 2008)
  - $233 \cdot 10^9$ bases
  - $155 \cdot 10^6$ séquences

3

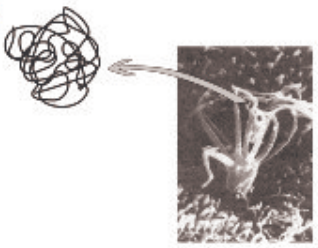
Recherches dans les séquences génomiques

- recherches par annotations

4

Recherches dans les séquences génomiques

>CfOR194 | Canis Olfactory Receptor
PMYKVILTPIMAYDRYL
KLMNIPLMNPLMSATT
TLMWNIPLMN...



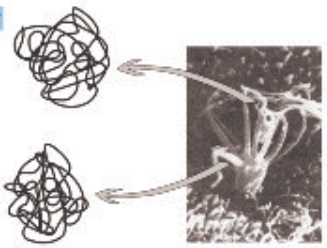
- recherches par annotations

5

Recherches dans les séquences génomiques

>CfOR194 | Canis Olfactory Receptor
PMYKVILTPIMAYDRYL
KLMNIPLMNPLMSATT
TLMWNIPLMN...

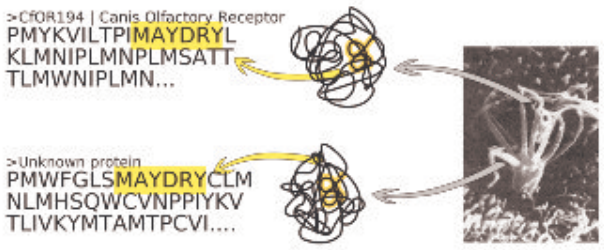
>Unknown protein
PMWFGLSMAYDRYCLM
NLMHSQWCVNPPIYKV
TLIVKYMTAMTPCVI....



- recherches par annotations

6

Recherches dans les séquences génomiques



- recherches par annotations / recherches par le contenu
- mutation des séquences : recherche par une partie du contenu
  $\longrightarrow$ recherches de motif, exacts ou approchés

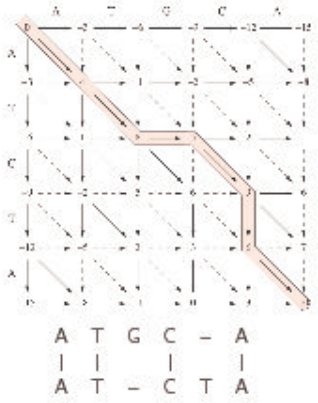
7

Alignement global : Needleman-Wunsch

Aligner ATGCA et ATCTA

match $\longrightarrow$ +4
substitution $\longrightarrow$ −2
gap $\longrightarrow$ −3

8



A T G C – A
| | | | |
A T – C T A

Alignement global : Needleman-Wunsch

- $X = (x_1, x_2 \dots x_m)$ et $Y = (y_1, y_2 \dots y_n)$
- $H(i,j)$ similarité entre $x_1 \dots x_i$ et $y_1 \dots y_j$

$\forall i: \quad H(i,0) = g_{\text{penalty}} \times i \quad \forall j: \quad H(0,j) = g_{\text{penalty}} \times j$
$\forall i,j, ij \neq 0:$

$$H(i,j) = \max \begin{cases} H(i-1,j-1) + d(x_i, y_j) & \text{(match ou substitution)} \\ H(i-1,j) - g_{\text{penalty}} & \text{(insertion)} \\ H(i,j-1) - g_{\text{penalty}} & \text{(délétion)} \end{cases}$$

$\longrightarrow$ $H(n,m)$ similarité globale entre X et Y

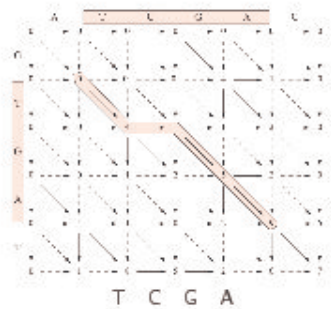
10

Alignement local : Smith-Waterman

Aligner localement ATGCAC et GTCTAT

match $\longrightarrow$ +4
substitution $\longrightarrow$ −2
gap $\longrightarrow$ −3

11



T C G A
| | | |
T – G A

Alignement local : Smith-Waterman

- $H(i,j)$ score maximum entre toutes les sous-séquences $x_a \dots x_i$ et $y_b \dots y_j$

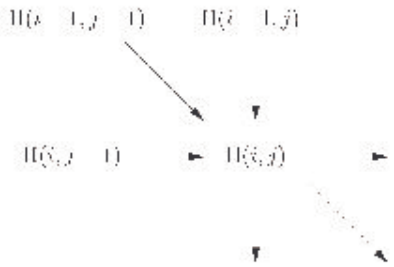
$\forall i,j : \quad H(i,0) = H(0,j) = 0 \quad \forall i,j, ij \neq 0 :$

$$H(i,j) = \max \begin{cases} 0 & \text{(début d'un nouvel alig.)} \\ H(i-1,j-1) + d(x_i, y_j) & \text{(match ou substitution)} \\ H(i-1,j) - g_{\text{penalty}} & \text{(insertion)} \\ H(i,j-1) - g_{\text{penalty}} & \text{(délétion)} \end{cases}$$

$\longrightarrow$ $\max_{i,j} H(i,j)$ meilleure similarité entre X et Y

13

Localité du calcul



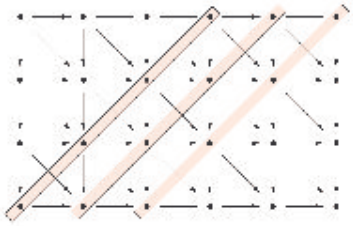
- Dépendance de trois cellules précédentes
- Seule information extérieure : $d(x_i, y_j)$
- Transmission des données vers les trois cellules suivantes

14

Complexités

Comparaison exhaustive, séquence de taille n contre séquence de taille m :

- $\mathcal{O}(mn)$ cellules
- calcul simultané de m cellules
- espace $\mathcal{O}(m)$

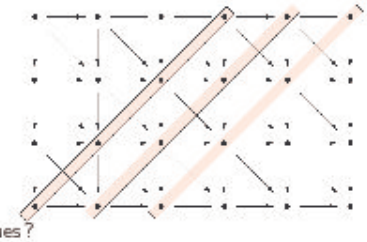


15

Complexités

Comparaison exhaustive, séquence de taille n contre séquence de taille m :

- $\mathcal{O}(mn)$ cellules
- calcul simultané de m cellules
- espace $\mathcal{O}(m)$

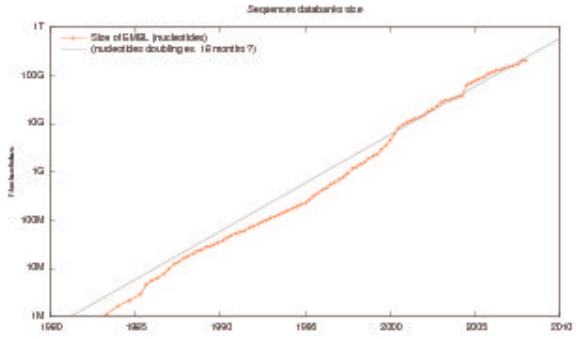


Algorithmes sous-quadratiques ?

- Masek et Paterson (1980) : $\mathcal{O}(n^2/\log n)$ pour scores rationnels
- Crochemore, Landau et Ziv-Ukelson (2002) : $\mathcal{O}(hn^2/\log n)$ (h entropie de la séquence)

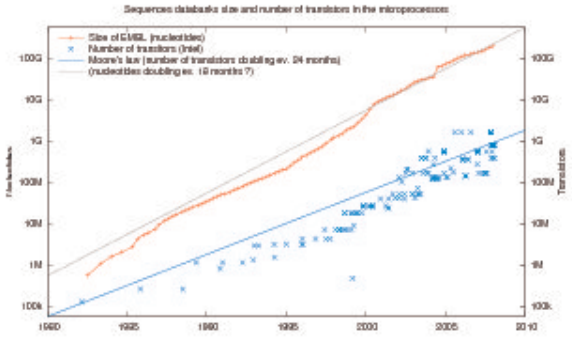
16

Genomic bank sizes and Moore's Law



17

Genomic bank sizes and Moore's Law

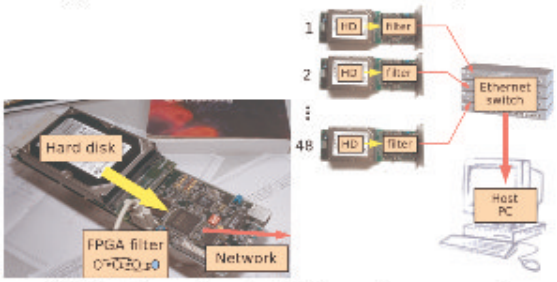


18

Architectures spécialisées FPGA

Rdisk

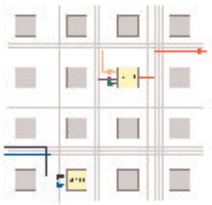
Filtrage de données directement à la sortie des disques durs



Système "économique" : < 200 euros de composants

20

Les FPGAs, une puissance de calcul reconfigurable

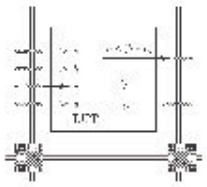


- Grille de cellules logiques
- Interconnexion (routage)
- Reconfigurable
- Prototypage
- Circuits économiques

2007 : 100×10^6 portes logiques

21

Les FPGAs, une puissance de calcul reconfigurable

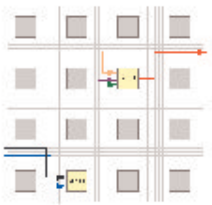


- Grille de cellules logiques
- Interconnexion (routage)
- Reconfigurable
- Prototypage
- Circuits économiques

2007 : 100×10^6 portes logiques

22

Les FPGAs, une puissance de calcul reconfigurable



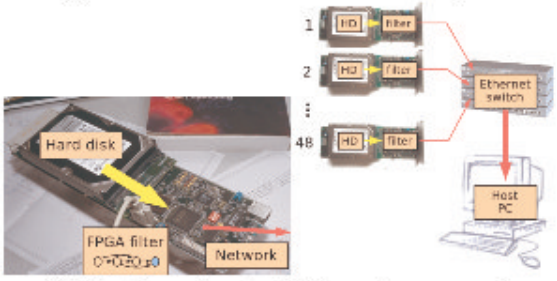
- Grille de cellules logiques
- Interconnexion (routage)
- Reconfigurable
- Prototypage
- Circuits économiques

2007 : 100×10^6 portes logiques

23

Rdisk

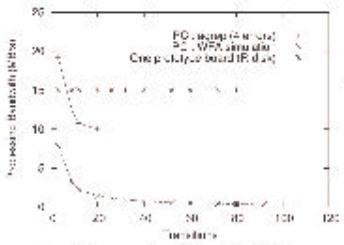
Filtrage de données directement à la sortie des disques durs



Système "économique" : < 200 euros de composants

24

wapam/Rdisk : vitesse

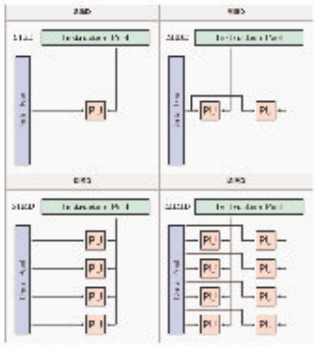


- Parallélisme
  - grain fin : 6 Gop/s (Spartan II, 40 MHz)
  - grain fort : R-disk 48
- Vitesse d'entrée : 16 Mo/s sur une carte
  - 4× – 10× speed-up vs PC (2 GHz, 728 Mo RAM)
  - Temps de calcul pour EMBL
    PC : > 2 heures, 1 carte : 35 min, 48 cartes : 40 s
- Temps de compilation : 60 – 100 secondes

25

Calculs bioinformatiques sur cartes graphiques

Instruction parallelism



Flynn's Taxonomy [wikipedia]

- No parallelism : one instruction, one data
- SIMD (single instruction, multiple data)
  - vector processors (1970's), MMX, SSE...
  - bit-parallelism
- MIMD (multiple instruction, multiple data)
  - clusters, multi-core

27

Bioinformatics computations on GPU

- 2005 : RAxML
- 2006 : ClustalW
- 2007 : mumMER
- 2008 : Smith-Waterman, spliced sequences, Cell SW
- en cours : PWM, ADP, Séquenceurs

28

2005 : RAxML (phylogeny) [BrookGPU]

Initial Experiences Porting a Bioinformatics Application to a Graphics Processor

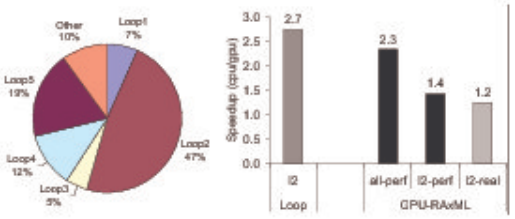
Maria Charalambous[1], Pedro Trancoso[1], and Alexandros Stamatakis[2]

29

2005 : RAxML (phylogeny) [BrookGPU]

Initial Experiences Porting a Bioinformatics Application to a Graphics Processor

Maria Charalambous[1], Pedro Trancoso[1], and Alexandros Stamatakis[2]



30

2006 : GPU-ClustalW [OpenGL Shading Lang]

GPU-ClustalW: Using Graphics Hardware to Accelerate Multiple Sequence Alignment

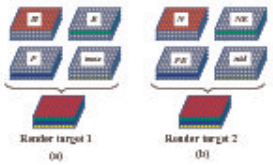
Weiguo Liu, Bertil Schmidt, Gerrit Voss, and Wolfgang Müller-Wittig

31

2006 : GPU-ClustalW [OpenGL Shading Lang]

GPU-ClustalW: Using Graphics Hardware to Accelerate Multiple Sequence Alignment

Weiguo Liu, Bertil Schmidt, Gerrit Voss, and Wolfgang Müller-Wittig



32

2006 : ClustalW GPU (Pairwise comparison)

Number of sequences (average length)		200 (412)	400 (408)	600 (482)
ClustalW (P4, 3GHz)	Overall	194.9	891.9	1818.1
	Pairalign	183.8 (94.4%)	833.1 (93.4%)	1697 (93.3%)
	Guided Tree	0.07 (0.03%)	0.8 (0.09%)	4.1 (0.2%)
	Malign	11.0 (5.6%)	58.0 (6.5%)	117.0 (6.4%)
GPU-ClustalW (GeForce 7800)	Overall	27.2	134.1	272.4
	Pairalign	16.1 (59.2%)	75.3 (56.2%)	151.3 (55.5%)
	Guided Tree	0.07 (0.3%)	0.8 (0.6%)	4.1 (1.5%)
	Malign	11.0 (40.4%)	58.0 (43.3%)	117.0 (43%)
Speedups	Overall	7.2	6.7	6.7
	Pairalign	11.4	11.1	11.2

33

2007 : MUMmerGPU (CUDA)

BMC Bioinformatics

Software

Open Access

High-throughput sequence alignment using Graphics Processing Units

Michael C Schatz*[1,2], Cole Trapnell[1,2], Arthur L Delcher[1,2] and Amitabh Varshney[2]

34

2007 : MUMmerGPU (CUDA)

BMC Bioinformatics

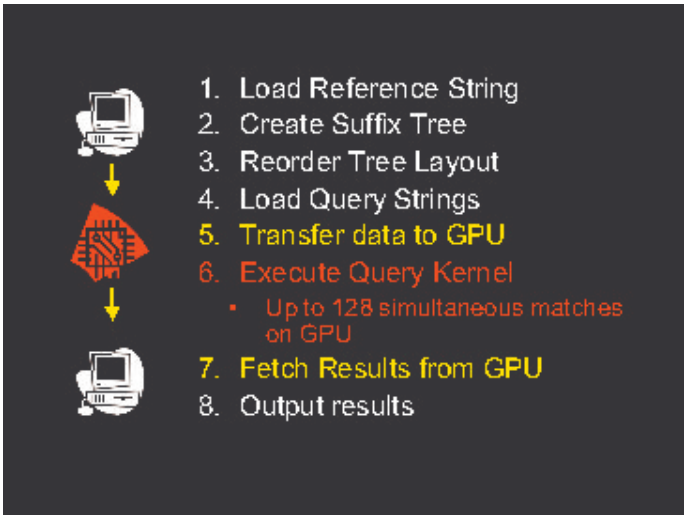
Software

Open Access

High-throughput sequence alignment using Graphics Processing Units

Michael C Schatz*[1,2], Cole Trapnell[1,2], Arthur L Delcher[1,2] and Amitabh Varshney[2]

35



2006 : GPU-ClustalW [OpenGL Shading Lang]

GPU-ClustalW: Using Graphics Hardware to Accelerate Multiple Sequence Alignment

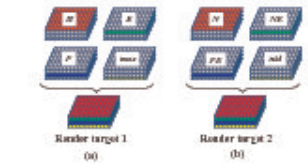
Weiguo Liu, Bertil Schmidt, Gerrit Voss, and Wolfgang Müller-Wittig

31

2006 : GPU-ClustalW [OpenGL Shading Lang]

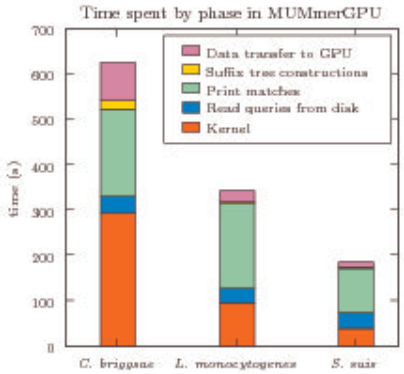
GPU-ClustalW: Using Graphics Hardware to Accelerate Multiple Sequence Alignment

Weiguo Liu, Bertil Schmidt, Gerrit Voss, and Wolfgang Müller-Wittig



32

2007 : MUMmerGPU (CUDA)



37

2008 : Smith-Waterman (CUDA)

BMC Bioinformatics

BioMed Central

Research

Open Access

CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment

Svetlin A Manavski*[1,2] and Giorgio Valle[1]

38

2008 : Smith-Waterman (CUDA)

BMC Bioinformatics

BioMed Central

Research

Open Access

CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment

Svetlin A Manavski*[1,2] and Giorgio Valle[1]

39

2008 : Alignment of spliced sequences (CUDA)

GEAgpu: Improved alignment of spliced DNA sequences to genomic data using Graphics Processing Units

Svetlin A. Manavski, Alessandro Albiero, Claudio Forcato, Nicola Vitulo, Giorgio Valle
CRIBI, University of Padova, Padova, Italy E-mail: svetlin.manavski@cribi.unipd.it

- alignement genome / EST
- scan (heuristique)
- puis extension

	Runtime [min]	# of alignments	# of queries aligned	Runtime factor
GEAGpu	8	87,596	46,382	1
Blast	2,297	56,799	39,073	277
Blat	358	75,820	43,210	47
Soap	18	69,424	43,513	2

40

2008 : Sequence Alignment (Cell)

BMC Bioinformatics

Software

Open Access

CBESW: Sequence Alignment on the Playstation 3

Adrianto Wirawan*, Chee Keong Kwoh, Nim Tri Hieu and Bertil Schmidt

41

2008 : Sequence Alignment (Cell)

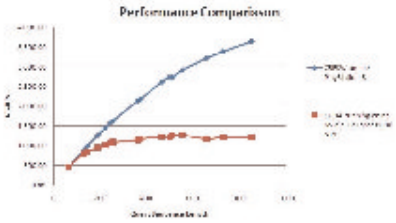
BMC Bioinformatics

Software

Open Access

CBESW: Sequence Alignment on the Playstation 3

Adrianto Wirawan*, Chee Keong Kwoh, Nim Tri Hieu and Bertil Schmidt



- 6 processeurs
- SIMD 128 bits
- 3.2 GHz

42

Projets GPU en cours, Sequoia, LIFL / INRIA Lille

- GPU PWM (Position Weight Matrices)
- Compilateur GPGPU pour méthodologie ADP
- GPU et séquenceurs à haut-débit
- Collaborations, sujets de stage : www.lifl.fr/~giraud
- Soutien de NVIDIA

43

Projets GPU en cours, Sequoia, LIFL / INRIA Lille

- GPU PWM (Position Weight Matrices)
  - avec J.-S. Varré
  - speed-up de 10× à 20× sur scan et comparaison
- Compilateur GPGPU pour méthodologie ADP
- GPU et séquenceurs à haut-débit
- Collaborations, sujets de stage : www.lifl.fr/~giraud
- Soutien de NVIDIA

44

GPU PWM Scan (with J.-S. Varré)

45

GPU PWM Scan (with J.-S. Varré)

H. Boukhatem, A. Ysmal (MSc students)

CPU	Core2 Duo 6600	2 × 2.4 GHz	2.0 Gop/s

46

GPU PWM Scan (with J.-S. Varré)

H. Boukhatem, A. Ysmal (MSc students)

CPU	Core2 Duo 6600	2 × 2.4 GHz	2.0 Gop/s
GPU 1	GeForce 8800 GTX	16 × 8 × 576 MHz	21.5 Gop/s
GPU 2	GeForce 8800 GTS	16 × 8 × 650 MHz	24.2 Gop/s
GPU 3	Quadro FX 570	4 × 8 × 208 MHz	2.7 Gop/s

47

GPU PWM Scan (with J.-S. Varré)

H. Boukhatem, A. Ysmal (MSc students)

CPU	Core2 Duo 6600 + TFM-Scan	2 × 2.4 GHz 2.4 GHz	2.0 Gop/s 2 – 8 Gop/s
GPU 1	GeForce 8800 GTX	16 × 8 × 576 MHz	21.5 Gop/s
GPU 2	GeForce 8800 GTS	16 × 8 × 650 MHz	24.2 Gop/s
GPU 3	Quadro FX 570	4 × 8 × 208 MHz	2.7 Gop/s

- 10× speed-up

48

GPU PWM Scan (with J.-S. Varré)

H. Boukhatem, A. Ysmal (MSc students)

CPU	Core2 Duo 6600 + TFM-Scan	2 × 2.4 GHz 2.4 GHz	2.0 Gop/s 2 – 8 Gop/s
GPU 1	GeForce 8800 GTX	16 × 8 × 576 MHz	21.5 Gop/s
GPU 2	GeForce 8800 GTS	16 × 8 × 650 MHz	24.2 Gop/s
GPU 3	Quadro FX 570	4 × 8 × 208 MHz	2.7 Gop/s

- 10× speed-up
- 2 – 3× speed-up compared to dedicated algorithms
- Good scaling on the GPU to 1 GB genomes (500 seconds)

49

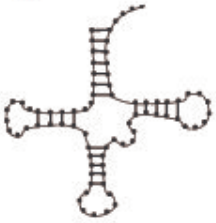
Projets GPU en cours, Sequoia, LIFL / INRIA Lille

- GPU PWM (Position Weight Matrices)
  - avec J.-S. Varré
  - speed-up de 10× à 20× sur scan et comparaison
- Compilateur GPGPU pour méthodologie ADP
  - avec P. Steffen, R. Giegerich (Univ. Bielefeld)
  - programmation dynamique générique
- GPU et séquenceurs à haut-débit
- Collaborations, sujets de stage : www.lifl.fr/~giraud
- Soutien de NVIDIA

50

ADP (Algebraic Dynamic Programming)

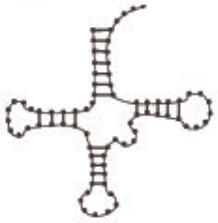
Generic framework
for dynamic programming
- Sequence alignments
- RNA foldings, co-foldings



51

ADP (Algebraic Dynamic Programming)

Generic framework
for dynamic programming
- Sequence alignments
- RNA foldings, co-foldings



nussinov78 $Z = s$

$s \rightarrow$ nil | right | pair | split with basepairing

nil: *empty*; right: s *base*; pair: *base* s *base*; split: s s

52

ADP (Algebraic Dynamic Programming)

Generic framework
for dynamic programming

- Sequence alignments
- RNA foldings, co-foldings

$$Ans_{bpmax} = IN$$
$$mil_{bpmax}(x) = 0$$
$$right_{bpmax}(x,b) = x$$
$$pair_{bpmax}(a,x,b) = x + 1$$
$$split_{bpmax}(x,x') = x + x'$$

$$Ans_{bpmax} = IN$$
$$bpmax = (nil, right, pair, split, b) \text{ where}$$
$$nil(x) = 0$$
$$right(x,b) = x$$
$$pair(a,x,b) = x + 1$$
$$split(x,x') = x + x'$$
$$b([]) = []$$
$$b([x_1,\ldots,x_n]) = [\max_{1<i<n} x_i]$$

53

Projets GPU en cours, Sequoia, LIFL / INRIA Lille

- GPU PWM (Position Weight Matrices)
  - avec J.-S. Varré
  - speed-up de 10× à 20× sur scan et comparaison
- Compilateur GPGPU pour méthodologie ADP
  - avec P. Steffen, R. Giegerich (Univ. Bielefeld)
  - programmation dynamique générique
- GPU et séquenceurs à haut-débit
  - avec J.-M. Batto, N. Pons, F. Bournezbeur (INRA Jouy)
  - projet MetaHIT : méta-génome intestinal humain
- Collaborations, sujets de stage : www.lifl.fr/~giraud
- Soutien de NVIDIA

54

Perspectives

- Calcul haute-performance : une révolution ?
  - non dans les concepts

55

Perspectives

- Calcul haute-performance : une révolution ?
  - oui dans les concepts, oui économiquement
  - 50× peak speed-up ⟶ 10× vraiment possible

56

Perspectives

- Calcul haute-performance : une révolution ?
  - oui dans les concepts, oui économiquement
  - 50× peak speed-up ⟶ 10× vraiment possible
- Côté informatique
  - Efferverscence, beaucoup de publications en 2008-09
  - Intérêt sur réflexion parallèle (et non un simple portage)
- Côté applications biologiques
  - Forte demande de solutions accélérées
  - Maturité des codes et des APIs ?

57

Perspectives

- Calcul haute-performance : une révolution ?
  - oui dans les concepts, oui économiquement
  - 50× peak speed-up ⟶ 10× vraiment possible
- Côté informatique
  - Efferverscence, beaucoup de publications en 2008-09
  - Intérêt sur réflexion parallèle (et non un simple portage)
- Côté applications biologiques
  - Forte demande de solutions accélérées
  - Maturité des codes et des APIs ?

Merci !

58

Résumé

Les données bioinformatiques issues des séquenceurs sont toujours en croissance exponentielle. Aux génomes de référence s'ajoutent maintenant les variations individuelles tout comme les méta-génomes (séquences d'organismes prélevés dans un même milieu).

Nous présenterons dans cet exposé quelques traitements parallèles sur ces données : certains se contentent d'un parallélisme à gros grain, facile à mettre en oeuvre sur cluster ou sur GPU, d'autres demandent des analyses plus fines pour traiter au mieux les différents accès mémoire. La comparaison intensive de séquences est souvent au coeur de ces algorithmes, mais d'autres défis surgissent des dernières technologies, notamment avec les séquenceurs de dernière génération. Nous parlerons aussi d'une méthode générique pour certains problèmes de programmation dynamique.

59

2.15 Transferts d'applications sur le GPU

David Delfour (Univ. Perpignan)

Au cours de ces 10 dernières années, l'augmentation de performance offerte par les processeurs généralistes, ou CPU, pour les applications généralistes mono-thread a été freinée par la faible IPC présente dans ces applications et le problème de la de la dissipation thermique. En parallèle, les processeurs graphiques ou GPU, ont de leur coté utilisé les transistors disponibles pour augmenter les performances des applications graphiques qui disposent d'un important parallélisme de données et de calcul spécifiques.

Les GPU et les API de programmation associées tendent actuellement vers plus de généricité, ce qui rend possible et à moindre coût, l'accélération de programmes généralistes. Cependant, exploiter cette puissance de calcul n'est pas sans poser un certain nombre de problème en terme d'accès mémoire, de précision des calculs et d'accès aux caractéristiques matérielles. Nous verrons dans cet exposé les problèmes que nous avons rencontrés et leurs solutions.

Retours d'expérience autour du GPGPU



Sylvain Collange
Marc Daumas
David Defour

Laboratoire Eliaus,
Université de Perpignan

UPVD

Journée Thème Émergent GPGPU du GDR ASR

http://surena.univ-perp.fr/GPGPU/
4 décembre 2008
Université Paris 6, Amphi Durand, bât. Esclangon

- Modalité de participation
  - Journée ouverte à tous, pas de frais de participation
  - Contacter David Defour (David.Defour@univ-perp.fr)
- Comité d'organisation et soutiens
  - Marc Daumas (ELIAUS, Perpignan)
  - David Defour (ELIAUS, Perpignan)
  - Ronan Keryell (HPC Project)
  - Jean-Luc Lamotte (Paris 6)
  - Dominique Lavenier (IRISA, ENS Cachan)
  - Stéphane Vialle (Supélec, Metz)

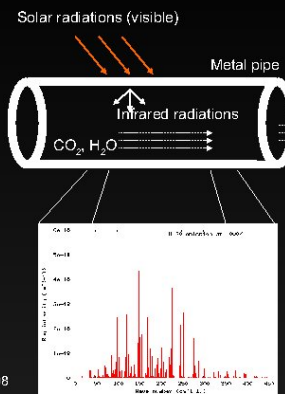
07/10/08 2

Nos travaux autour du GPGPU à Perpignan

- Applicatifs :
  - Transfert d'applications Physique
    (Simulation de récepteur solaire, échange thermique)
    OpenGL
  - Informatique légale
    (Recherche de motif sur un disque-dur, décodage PKCS-#5)
    CUDA
- Matériels :
  - Tests de cartes graphiques, veille technologique
    Cg, Assembleur deCUDA, CUDAsm
  - Évaluation de fonctions en utilisant la localité de valeur
  - Utilisation du filtrage pour l'évaluation de fonction
- Outils :
  - Bibliothèque d'opérateurs float-float
    Cg
  - Bibliothèque d'intervalle
    (appliquée au lancer de rayon certifié)
    Cg, CUDA
- En cours :
  - Interpolation / Approximation de fonctions bi-variées
  - Mesure de consommation
  - Simulation

07/10/08 3

Simulation de récepteur solaire



S. Collange, M. Daumas, D. Defour. "Line-by-line spectroscopic simulations on graphics processing units". Computer Physics Communications, 178:135-143, 2008.

07/10/08 4

Contexte

- 2005-2006
  - Programmation en OpenGL
  - Programmation des vertex/pixel shader en Cg / GLSL
  - Sauvegarde des résultats en texture
  - …

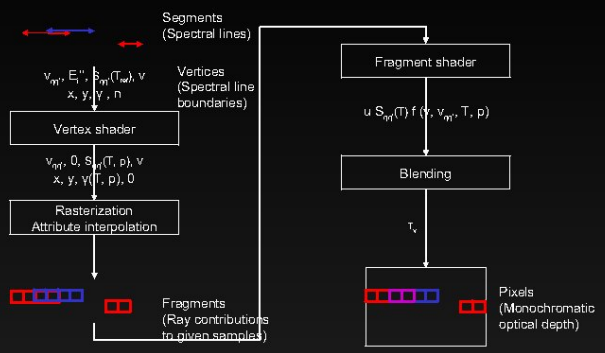
07/10/08

Implémentation

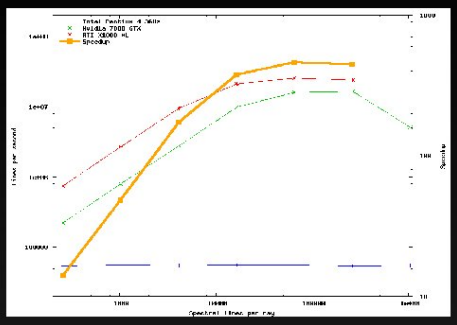
- Lancer de rayon à travers des volumes finis
  - 2x10⁶ évaluations parallèles de : $\frac{S_{gj}(T)}{S_{gj}(T_{ref})}\frac{Q(T_{ref})}{Q(T)}e^{-c_2\tilde{\nu}_{gj}\left(\frac{1}{T}-\frac{1}{T_{ref}}\right)}\frac{1-e^{-c_2\tilde{\nu}_{gj}/T}}{1-e^{-c_2\tilde{\nu}_{gj}/T_{ref}}}$
  - 16x10⁶ évaluations parallèles de : $\tau_\nu \sum_g u_g \sum_{gj} S_{gj}(T) f(\nu-\nu_{gj})$; $I_{out}=I_{in}e^{-\tau_\nu}+\frac{2h\nu^3}{c^2}\frac{1}{e^{h\nu/kT}-1}(1-e^{-\tau_\nu})$
  - intégration/sommation :

07/10/08 6

Blending : Fusion des raies



Facteur d'accélération de 400



Explication

- 2^x est cablée en matériel
- ATI X1800 XL
  - 500 Mhz
  - 16 exp commence à chaque cycle
  - 8 000 M exp/s
- Intel Pentium 4
  - 3 000 Mhz
  - 1 exp tout les 150 cycles
  - 20 M exp/s
- Ratio de 1 / 400

Informatique Légale

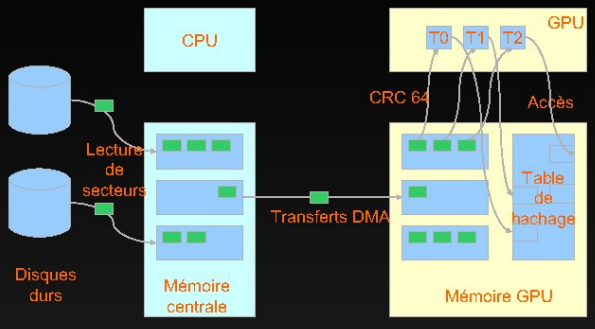
- Objectifs
  - Analyse de DD
  - Recherche de contenus illégaux connus
  - Récupération de mot de passe
- Problématiques
  - Palier les latences d'accès au DD
  - Conserver des performances raisonnables même avec de gros kernel (PKCS#5)

« Using Graphics Processors for Parallelizing Hash-based Data Carving », S. Collange, Y. Dandass, M. Daumas, D. Defour, HICSS-42, Januray 2009

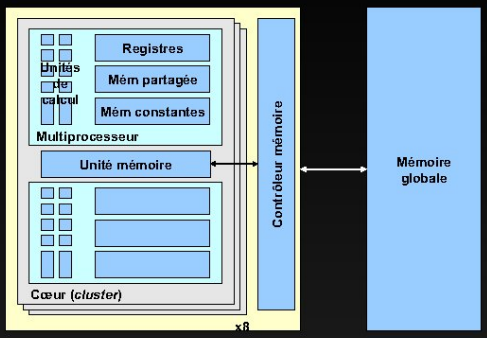
Bibliothèque de contenu recherché

- Découpé en cluster, puis en secteurs
- Calcul d'une clef de hachage pour chaque secteur
- Construction d'une base de données pour chaque secteur d'un cluster (8 pour des clusters de 4 Ko)

Pipeline implémenté

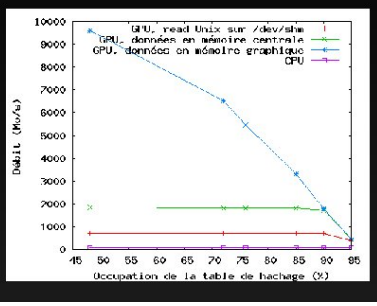


Architecture Nvidia G80 simplifiée



Résultats sur GPU

- Nvidia GeForce 8500 GT (30 €)
  - Cuda 1.1, Pentium 4 2.8 Ghz sur i945G
  - 128 threads/bloc, 16 blocs

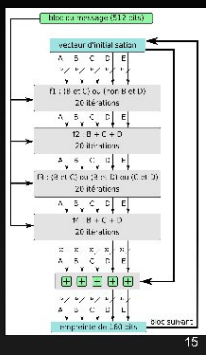


PKCS#5

- Standard de chiffrement par mot de passe RSA Laboratories
- Utilisé dans
  - WPA, WPA2
  - Microsoft Data Protection API
  - ...
- Principe :

```
PBKDF2(P, S, c)
   U:= SHA1((P ⊕ opad) || SHA1 ((P ⊕ ipad) || S || Int(1)))
   K:= U
   for (i=1; i<c; i++){
      U:= SHA1((P ⊕ opad) || SHA1 ((P ⊕ ipad) || U))
      K:= K ⊕ U;
   }
   return K;
)

P : message (512 bits)
S : salt (128 bits)
c : nombre d'itérations (~4000)
K : résultat (160 bits)
|| : concaténation
```



Résultats sur PKCS#5

- Attaque par dictionnaire
- CPU Intel Pentium 4
  - 100 mdp/s
- FPGA Xilinx vertex 4 FX-100
  - Implémentation pipelinée à 162 Mhz
  - 4 unités en parallèle
  - 510 mdp/s
- GPU Nvidia GeForce 8500 GT
  - 4200 mdp/s

Limites liées au matériel

- Pas de pile pour les arguments/variables
  - Pas de récursion
- Pas de mémoire ECC
  - Problème de fiabilité dans le cas de super-calculateur à base de GPU
- Concilier pipeline graphique et HPC
  - Complexité liée au type de thread
  - Précision arithmétique
  - Matériel accessible à l'un et pas à l'autre
- Exécution des sections séquentielles déférée au CPU
  - Différent du Larrabee, Cell

Limites liées à l'interface CUDA

- Multi-GPU
  - Gestion manuelle des communications
  - Pas de garantie de bon fonctionnement
- Compilation / Exécution
  - Erreurs liées au matériel non détectées
  - Peu d'interactions liées au matériel
  - Front-End de nvcc d'Edison Design Group conforme à la norme C++ (doc CUDA nous dit l'inverse).
  - Pas d'accès à l'assembleur
- Mémoire
  - Cuda n'est pas capable d'inférer le type de mémoire pointée
  - Gestion manuelle de l'allocation/libération de la mémoire partagée
- Pérennité
  - Fonctionnalités rajoutées par les versions de Compute Capability
  - OpenCL

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.aristote.asso.fr