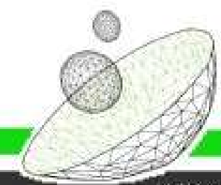
Using GPU for model calibration

Florent Duguet

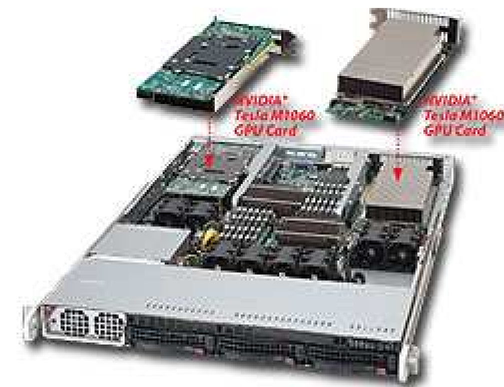
Parallel Computing Architect for BNP PARIBAS



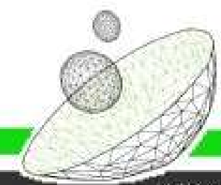
© Altimesh 2009 – all rights reserved

Project Results

500 cores



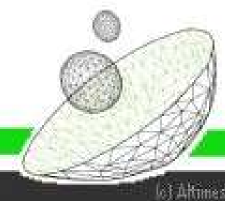
8 cores 2 GPUs



© Altimesh 2009 – all rights reserved

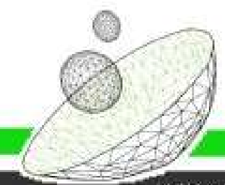
Summary

- Manycore architectures
- The Problem – The Project
- Issues and Pitfalls
- Results
- Applicability to other problems



© Altimesh 2009 – all rights reserved

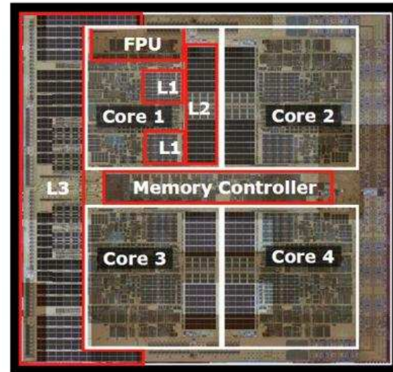
Manycore architectures

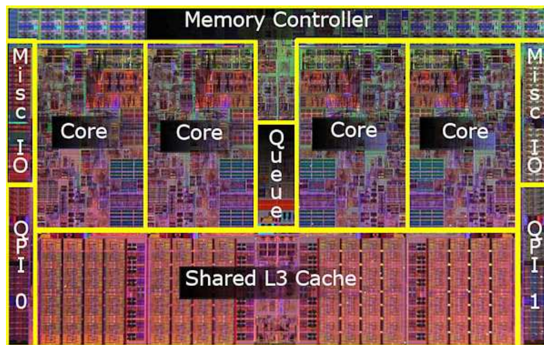


© Altimesh 2009 – all rights reserved

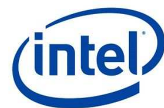
Manycore architectures

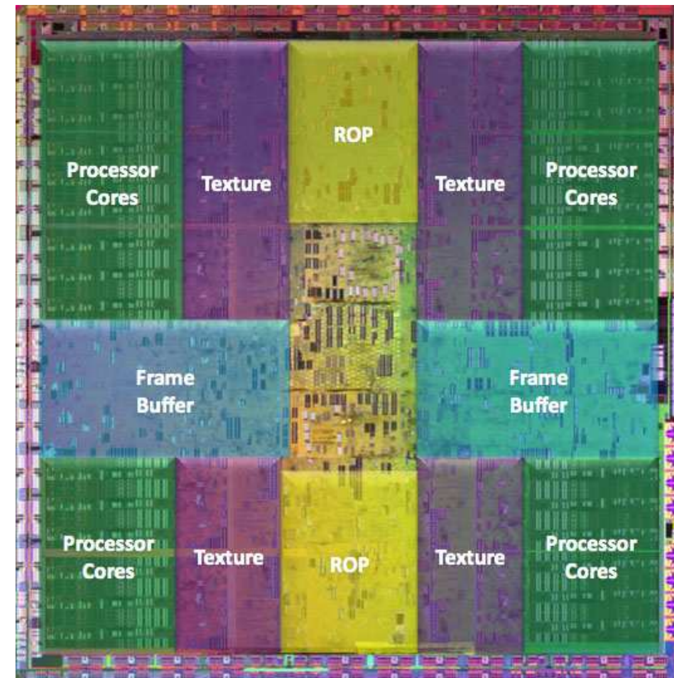
Phenom X4



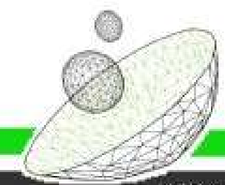


Nehalem





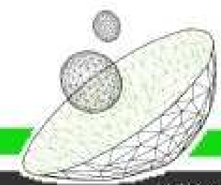
GT200



© Altimesh 2009 – all rights reserved

Manycore architectures

- Conventional (few cores)
  - Cache and out of order execution
  - Backwards compatibility (support for 8086)
  - Frequency limit reached – 3.6 GHz
- Manycore (hundreds of cores)
  - More transistors dedicated to compute
  - No need of backwards compatibility
    - Graphics APIs only (OpenGL, DirectX)
  - Wider memory bandwidth

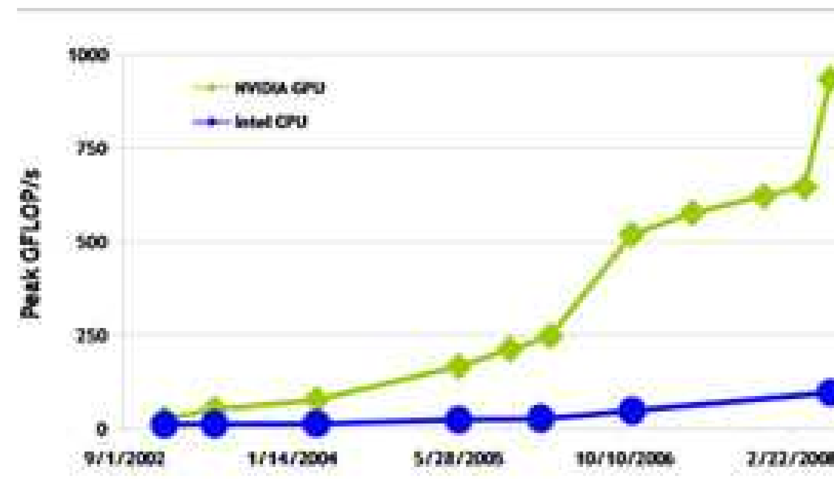


© Altimesh 2009 – all rights reserved

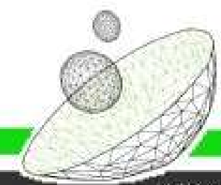
Manycore architectures

- Core count increase at higher pace than CPUs
  - 512 for nVIDIA - Fermi
  - 1600 for ATI - HD 5870

- Memory bandwidth
  - 150 GB/s for GPU
  - 20 GB/s for CPU



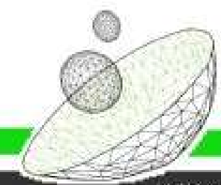
FLOP = floating point operation
(source=nVIDIA)



© Altimesh 2009 – all rights reserved

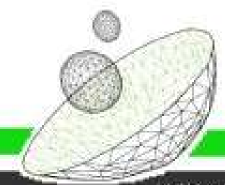
Manycore architectures

- Until mid 2000, very limited programmability
  - Need for in depth knowledge of Graphics
  - Few standard programming features (no functions, no stack)
- More tools arise – towards a standard
  - BrookGPU (2004), **CUDA** (2007)
  - **OpenCL** – standard (2009)
- Big players
  - nVIDIA
  - AMD (ATI)
  - Intel (announced)



© Altimesh 2009 – all rights reserved

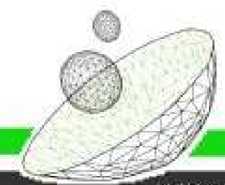
The Problem – The Project



© Altimesh 2009 – all rights reserved

The Problem

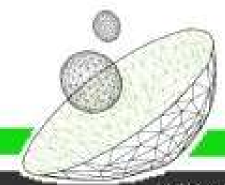
- Calibration with Monte Carlo Simulations
  - Very complex model (no closed form or Fourier formulas)
  - Very compute intensive calibration
  - Updates needed as often as possible
- First shot of the algorithm
  - Conventional CPU clusters
  - No global optimization (algorithm part of a bigger library)
- Computation time : **45 minutes** on a **50 cores** cluster



© Altimesh 2009 – all rights reserved

Approach

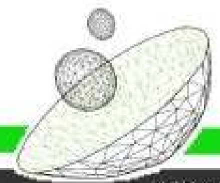
- Think global
  - Optimization is not *premature* in this case
- Use GPU for as much compute as possible
- Validate precision needs
  - GPUs are most powerful in single precision
  - What is the trust of a Monte Carlo result compared to the error of single precision
- Make the process transparent to users



© Altimesh 2009 – all rights reserved

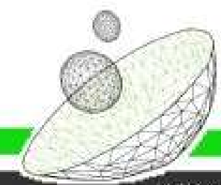
How to make it parallel

- Tens of thousands of drawings
  - Each are independent by definition (Monte Carlo)
  - All paths can fit in GPU memory (for a small set of time steps)
- Big integral for fit
  - Split into many buckets
  - Each bucket computed independently
  - For a compute chunk, all data fit in GPU cache
- Global iterative process
  - Bootstrap approach -> Sequential algorithm
  - Though each unit is compute expensive enough
  - Little memory exchange between CPU and GPU



© Altimesh 2009 – all rights reserved

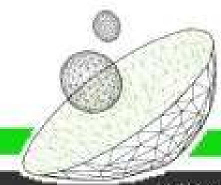
Issues and Pitfall



© Altimesh 2009 – all rights reserved

Environmental elements

- Project started in 2007
  - CUDA in its early stages (few features and samples)
  - No double precision available on hardware
  - Knowledge in computer graphics and GPU behavior was a plus
- Reference code written in Ada
  - No way to have same code for CPU and GPU
  - Need to align interleaved Ada code with brand new optimized C/CUDA code
- Comma change testing
  - Reference run, on a single core, required **hours**
  - Hard to make a one to one correspondence at startup



© Altimesh 2009 – all rights reserved

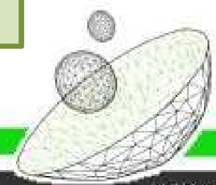
On Precision

- **Double precision**
  - Could not define whether difference was due to algorithm or single precision floating point
  - **Obtained early access to double precision hardware** during the project -> most useful

FPU registers are **80** bits – **GPU** double precision is **64** bits

- **Order of operations**
  - Applying dividends and other log/exp space operations required specific care. Precision can be lost because of bad instruction order

Sum (1/N; 1..N) can be <> 1, for N very large



© Altimesh 2009 – all rights reserved

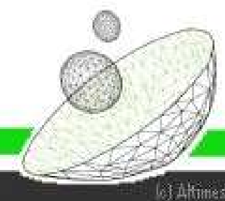
On Precision

- **Intrinsic exp operation**
  - The IEEE 754 norm is not strict on transcendents
  - bias can yield error drifts

nVIDIA provided an alternate unbiased implementation of exp

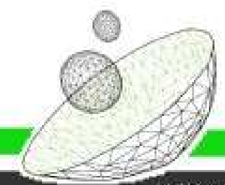
- **Epsilon and Monte Carlo trust range**
  - Epsilon = 3.6 E-7 in single precision
  - Monte Carlo trust for our amount of paths is orders of magnitude wider

For Monte Carlo, single precision is most often sufficient !



© Altimesh 2009 – all rights reserved

Results



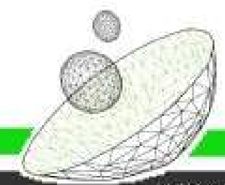
© Altimesh 2009 – all rights reserved

Validation of the project

Double precision was very useful for algorithmic validation

Single precision was sufficient in terms of accuracy

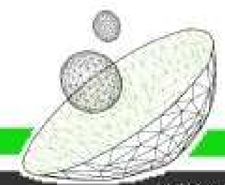
Embedded in a network service thus transparent to users



© Altimesh 2009 – all rights reserved

What to compare ?

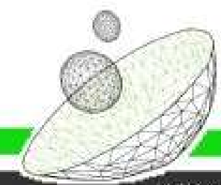
- Reference implementation is
  - Part of a larger project
  - Not prematurely optimized
  - Clusterized by default
- We have three/four configurations
  - **Reference** implementation – before optimization
  - **C** implementation optimized for CPU
  - **Single**/**Double** precision optimized for GPU



© Altimesh 2009 – all rights reserved

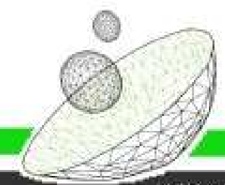
Results

	Reference	C	Double	Single
Time	7h45	45’	3”35	40”
Speedup	1	10.3	12.6 (wrt C)	68 (wrt C)
	1 CPU core	1 CPU core	+ 1 GPU	+ 1 GPU
Watts	75	75	275	275
Wh ratio ($)	1	10.3	35.3 (wrt Ref)	190 (wrt Ref)



© Altimesh 2009 – all rights reserved

Applicability to other problems



© Altimesh 2009 – all rights reserved

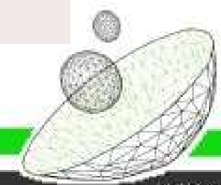
Memory Bound vs Compute Bound

- Memory bound means that accessing data takes more time than actually processing it

- Compute bound means that processing data takes more time than accessing it

- Figure Facts

	CPU (core)	GPU (GT200)	ratio
Memory bandwidth	~20 GB / s	~100 GB / s	5
Time per op (4 bytes)	200 ps	40 ps	(w/o cache)
FLOPS	~13 GFLOPS	~760 GFLOPS	60
Time per op	77 ps	1.3 ps	



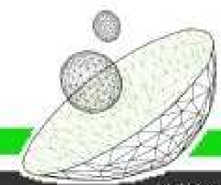
© Altimesh 2009 – all rights reserved

In practice

- Cost of operations

c = a + b : 2 mem reads 1 mem write, 1 FLOP

- On CPU
  - Memory : 600 ps, Compute : 80 ps
    - We could do 8 times more operations
- On GPU
  - Memory : 120 ps, Compute : 1.3 ps
    - We could do **100** times more operations



© Altimesh 2009 – all rights reserved

Other example

- Should we tabulate exp ?

- On CPU
  - Memory : 2 operations : 400 ps
  - Exp : ~ 40 FLOP : 1600 ps

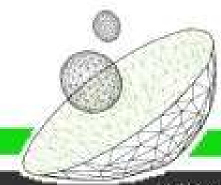
  Compute Bound ! by a factor of **4**

CPU : TABULATE

- On GPU
  - Memory : 2 operations : 80 ps
  - Exp : ~ 16 FLOP : 21 ps

  Memory Bound ! by a factor of **4**

GPU : RECOMPUTE



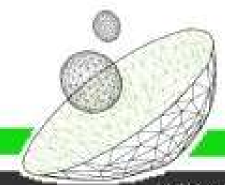
© Altimesh 2009 – all rights reserved

Some insights

- Problem is **linear algebra** or similar
  - Highly probable that problem is memory bound on both architectures
  - Speed-up is determined by **bandwidth ratio**
    - see memory capabilities for CPU (DDR3) and GPU (DDR5)

- Problem uses **mostly transcendents**
  - Example : Box-Müller for RNG of a Gaussian number
  - Speed-up will be higher than bandwidth ratio

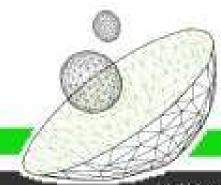
  - In **ideal cases, can be hundreds**



© Altimesh 2009 – all rights reserved

Key elements/figures

CPU		GPU
Hundreds cycles – OS kernel call	**Thread synchronization**	1 cycle – hardware assisted
Automatic Very little user control Feeling Lucky ?	**Cache handling**	Manual cache management Different cache types (RO/RW)
64 KB per core	**L1 - Cache size**	16-48 KB for 8-32 cores
Several MB	**L2 - Cache size**	-
2.6	**FLOPs per MEMOPs**	**30**
General, predicted, out of order execution	**Branching**	Limited, can be very expensive



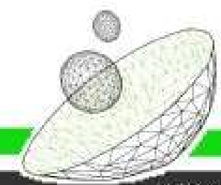
© Altimesh 2009 – all rights reserved

Take home message

Compute is cheaper than memory access

Think manycore

Verify needs of precompute



© Altimesh 2009 – all rights reserved