Introduction to Stream Processing

Trond Hagen
SINTEF ICT, Applied Mathematics

Geilo, January 2008

Schedule, Thursday

09:00 - 09:40	
Introduction to stream processing	Trond Hagen
09:50 - 10:30	
Introduction to stream processing cont'd	Trond Hagen
15:00-15-40	
CUDA programming	Johan Seland
15:50-16-30	
CUDA programming cont'd	Johan Seland
17:00-18-30	
Examples of applications	André Brodtkorb, Johan Seland,

Schedule, Friday

09:00 - 09:45

Introduction to Cell BE	Trond Hagen

10:00 - 10:45

Programming Cell BE	André Brodtkorb

11:00-12:00

“Birds of a feather” – parallel processing	Johan Seland
Summary and discussion	

Thursday evening

- Don’t miss the quiz in the bar after the dinner!
- Chance to win a ~10 000,- NOK HPC graphics card sponsored by NVIDIA

NVIDIA

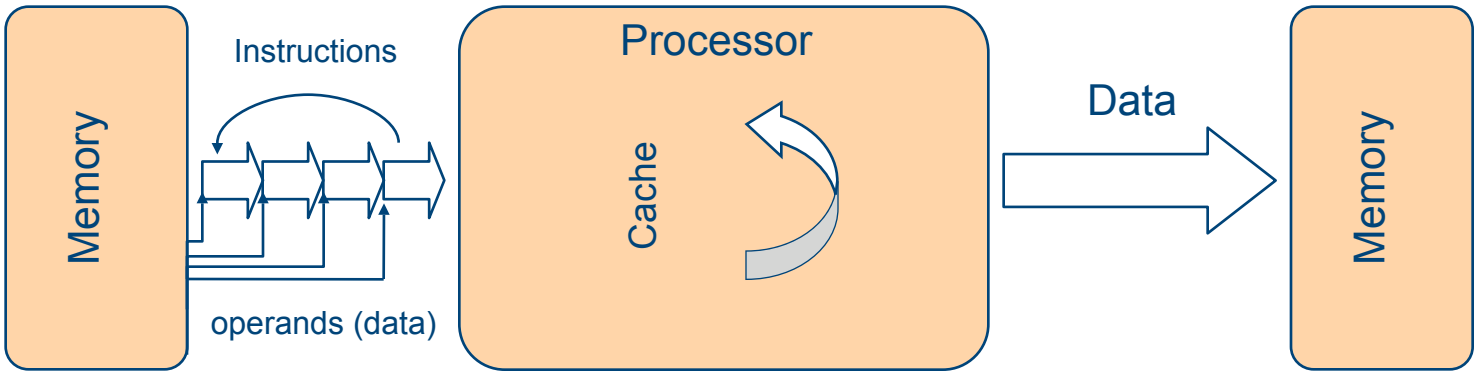
Outline

- Introduction to Stream Processing
- Introduction to Graphics Processing Units (GPUs)
- GPU Architecture
- GPU Programming Models
- Examples
- Looking Forward

What is Stream Processing?

- A stream is a set of input and output data
- Stream processing is a series of operations (kernel functions) applied for each element in a stream
- Uniform streaming is most typical. One kernel at a time is applied to all elements of the stream
- Single Instruction Multiple Data (SIMD)

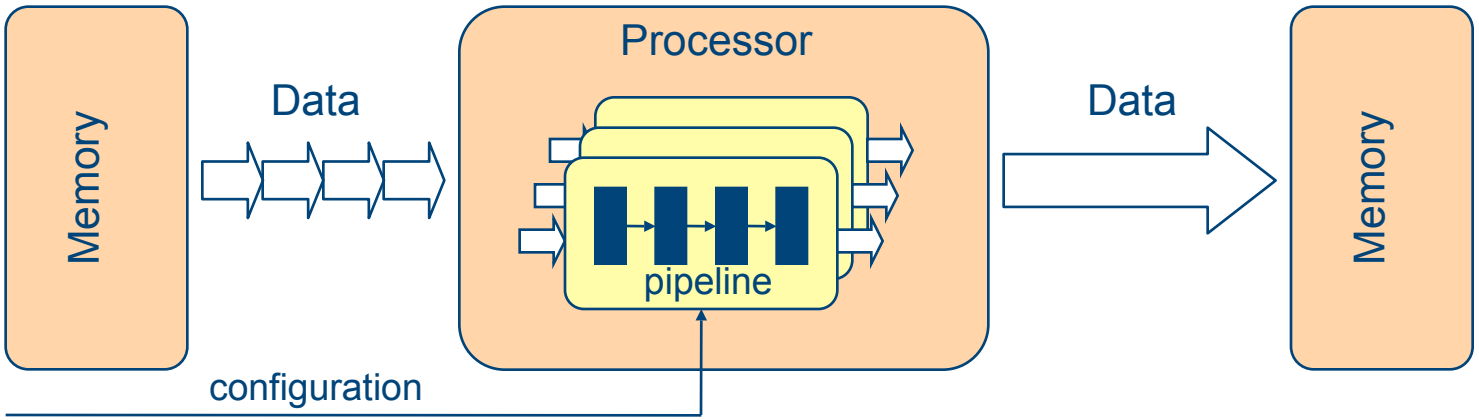
Instruction-Based Processing



During processing, the data required for an instruction's execution is loaded into the cache, if not already present.

Very flexible model, but has the disadvantage that the data-sequence is completely driven by the instruction sequence, yielding inefficient performance for uniform operations on large data blocks.

Data Stream Processing



The processor is first configured by the instructions that need to be performed and in the next step a data-stream is processed.

The execution is distributed among several pipelines.

Different Computing Paradigms

- Data stream processing is advantageous when large data blocks undergo the same operation, because this allows the memory efficient streaming and parallel processing of the data.

- Example: Matrix-matrix addition C = A+B

```
// instruction based
for(i=0; i<numRows; i++)
  for(j=0; j<numCols; j++)
    C[i][j]=A[i][j]+B[i][j];
```

```
// data stream
 setInputArrays(A, B);
 setOutputArrays(C);
 loadKernel("return a+b");
 execute();
```

Hardware Evolution

- *“The number of transistors on an integrated circuit for minimum component cost doubles every 24 months” - Moore’s Law*
- Moore’s law still seems to hold true
- Increases in transistor density have driven roughly proportional increases in processor performance
- Performance gains have also relied heavily on the increase of clock frequency, which is closely related to transistor size
  - Smaller transistors can switch faster

Hardware Evolution (cont'd)

- Higher frequency gives easy and instant performance benefits
  - Software runs faster without any modification
  - This has allowed the computing industry to realize increasing value from their existing code base, with relatively little effort
- In the resent years we have not seen the traditional increase in core frequency
- Increasing the frequency has several implications:

Memory Problem

- Memory speeds have not increased as fast as core frequencies
  - A processor can wait through hundreds of clock cycles if it has to get data or instructions from main memory
- Larger caches combined with instruction level parallelism can reduce the memory-wait time

Longer Instruction Pipelines

- Longer instruction pipelines allow for higher core frequencies
- Longer pipelines results in more cache-miss penalty and lower the number of completed instructions per cycle
- The Intel Pentium 4 Prescott had a pipeline length of 31 stages and a frequency > 3 GHz
- Industry seems to converge between 9 and 11 stages

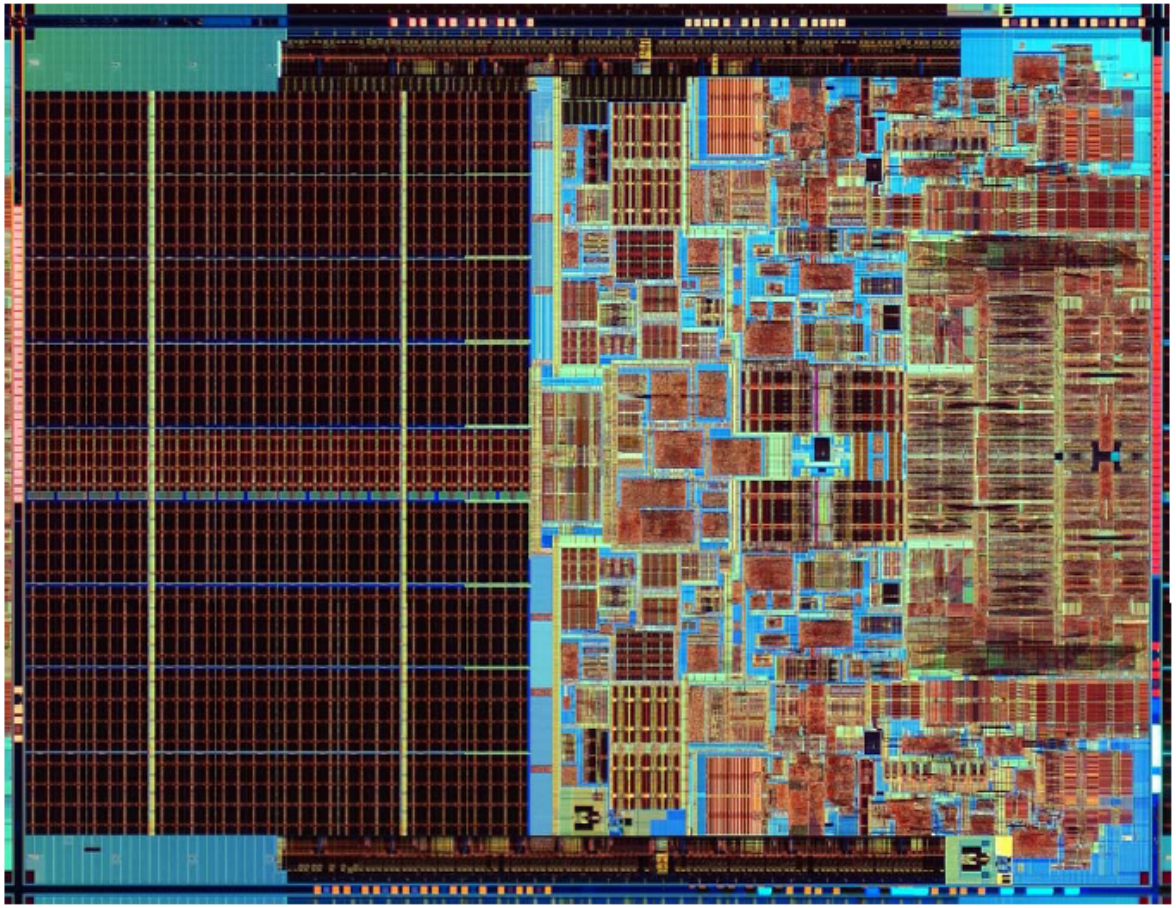
Power Consumption Problem

- When the frequency increases, the power consumption increases disproportionately
- The dependency between core frequency and power consumption is often said to be quadratic

Multi-Core Processors

- Multi-core can be used to take advantage of the continuously increase in transistor density
- Doubling the number of cores and halving frequency gives roughly the same performance, while the power consumption is reduced drastically
- Adding cores makes it possible to get higher performance without increasing the power density

Intel Core 2 Duo



Two “fat” cores

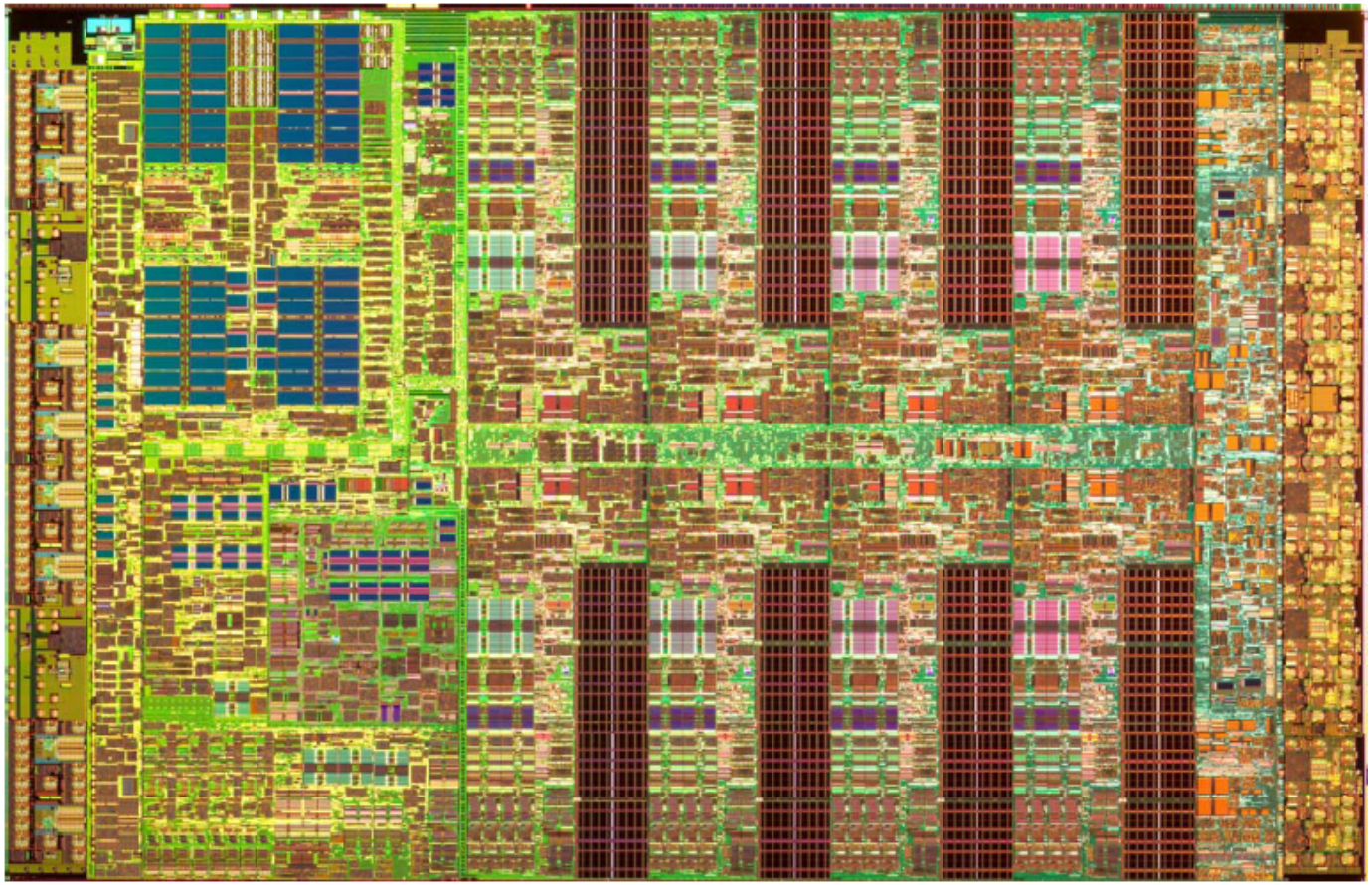
General-purpose CPUs

- ~50 % of the die area is cache
- An addition to that a lot of the area is used for instruction level parallelism
  - Instruction pipelining (execution of multiple instructions can be partially overlapped)
  - Superscalar (parallel instruction pipelines)
  - Branch prediction (predict outcome of decisions)
  - Out-of-order execution
- The amount of transistors doing direct computation is shrinking relative to the total number of transistors

Solution: Heterogeneous Architectures?

- Heterogeneous computing is the strategy of using multiple types of processing elements within a single workflow, and allowing each to perform the tasks to which it is best suited.
- Do not need only “fat” general purpose cores.
- Instead one can design smaller and simpler computational units, which can very effectively perform special tasks, e.g. floating point operations.
- Two most important heterogeneous systems:
  - Graphics Processing Units
  - Cell BE

Cell Broadband Engine



One “fat” core and eight “thin” cores

Graphics Processing Units (GPUs)

What is a GPU?

- Graphics rendering device
- Efficient at displaying computer graphics
- Their highly parallel architecture makes them very effective for a range of complex algorithms
- Driven by demand for increased realism in games
- All PCs have a GPU

NVIDIA GeForce 8800 GTX

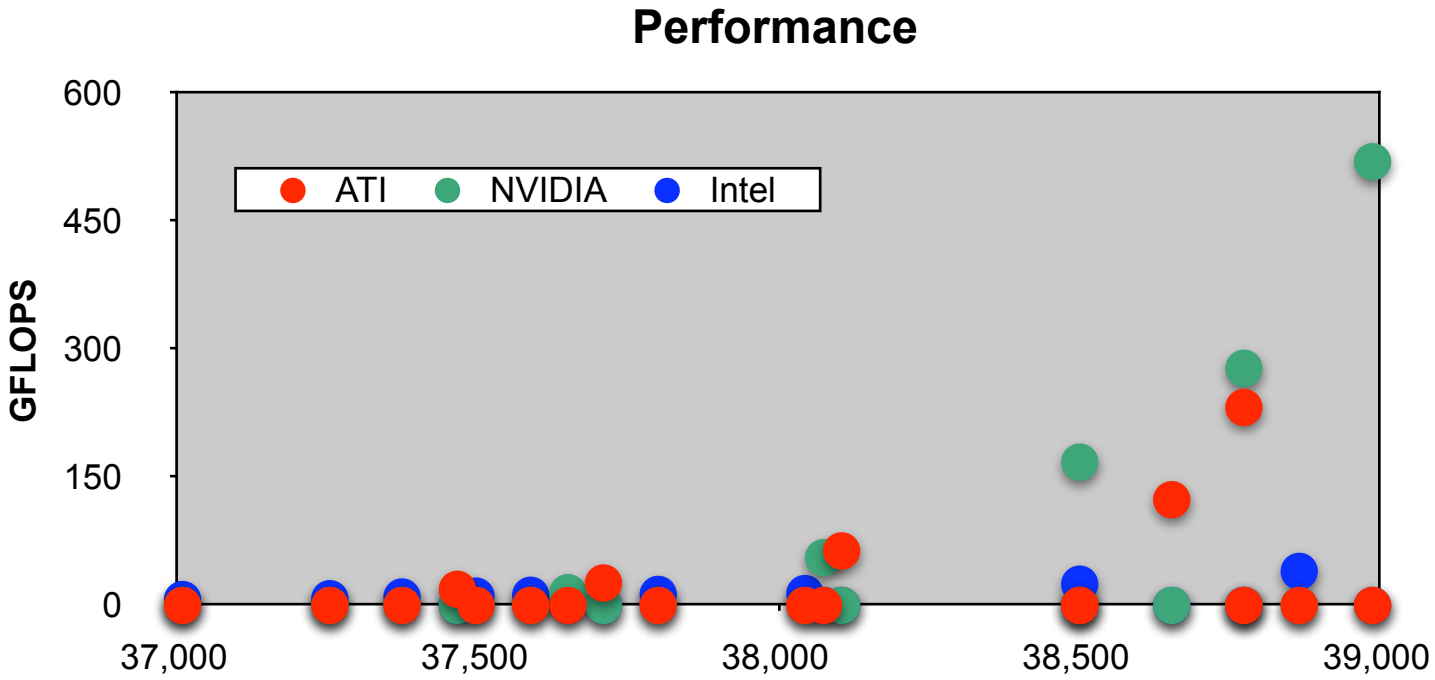
GPU-Design Premises

- Games typically renders 10 000s triangles @ 60 fps
- Screen resolution is typically 1600 x 1200
  - Each pixel is recalculated every frame
- A pixel loosely corresponds to a lightweight thread
- This corresponds to creating, running and destroying 115 200 000 threads per second
- GPUs are designed to make these operations fast

State of the art real-time rendering

State of the art real-time rendering

GPU Versus CPU Performance



GPU vs. CPU Performance (cont'd)

- Nr. 1 top 500 November 2007: BlueGene /L
  - 212 992 processors
  - ~596 TFLOPS
- Theoretically the same processing power as ~1150 high-end GPUs

http://www.top500.org

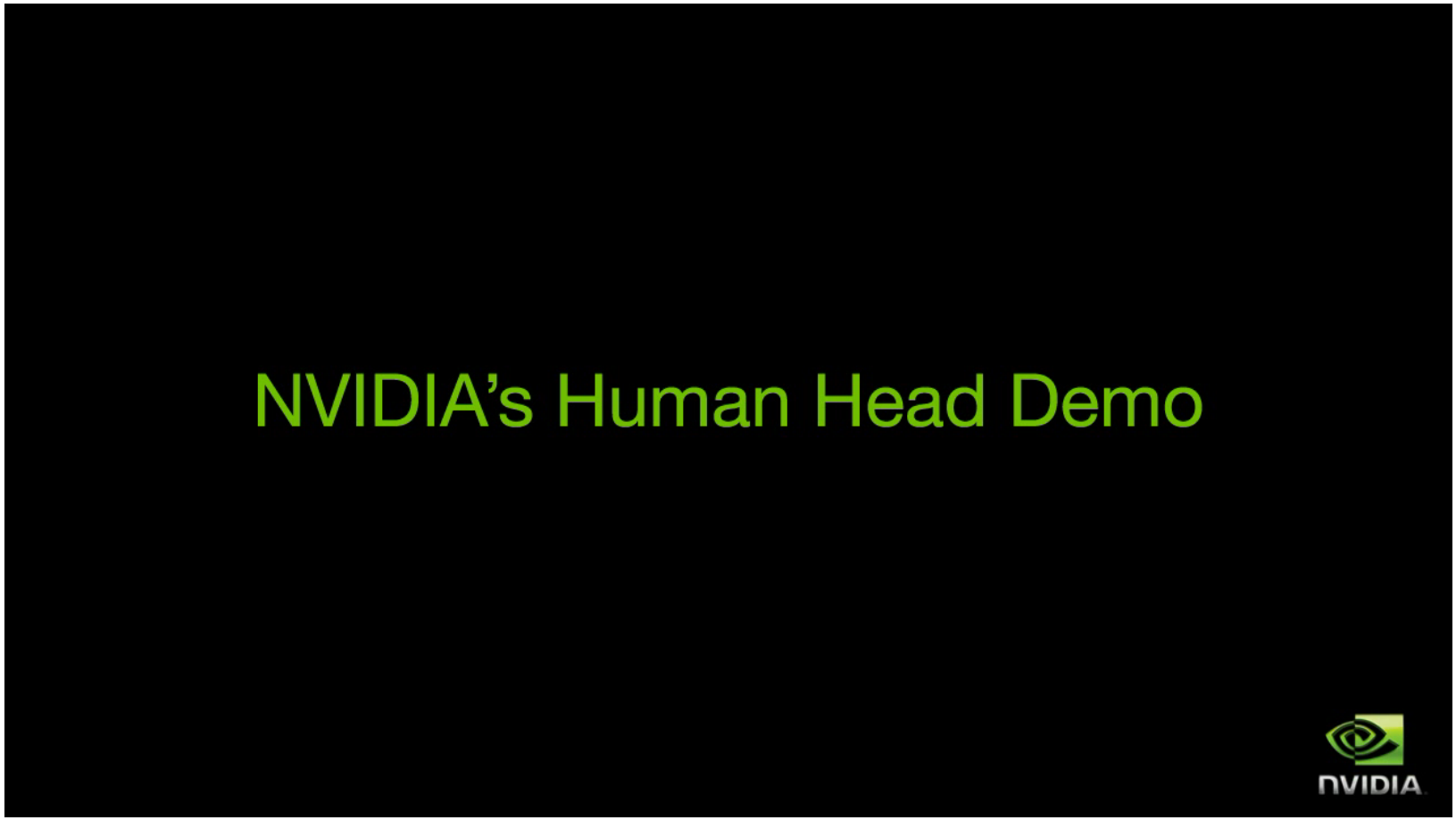
Power / Performance Ratio

- “If the performance per watt of today’s computers doesn’t improve, the electrical costs of running them could end up far greater than the initial hardware price tag” - google
- CPU – Intel Core 2 Duo “Conroe”
  - 65 W / 20 GFLOPs = **3.25** Watt / GFLOPS
- GPU
  - 150 W / 500 GFLOPs = **0.3** Watt / GFLOPS
- Cell BE
  - 70 W / 250 GFLOPs = **0.28** Watt / GFLOPS

  (5 W per SPE)

State of the art real-time rendering (cont'd)

State of the art real-time rendering (cont'd)



3D Navier-Stokes Fluid Simulator

3D Navier-Stokes Fluid Simulator

Deformable Skin Demo

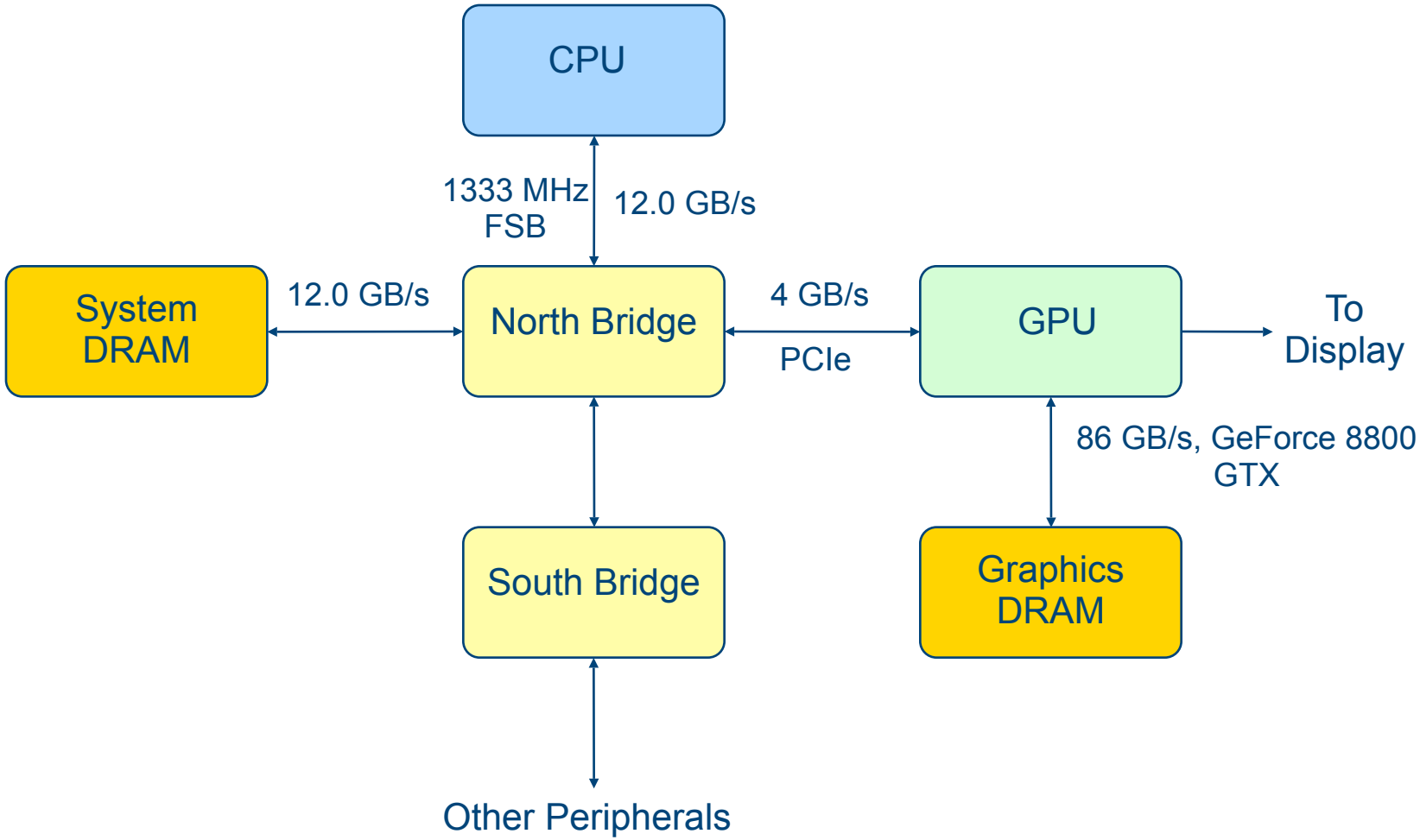
Deformable Skin Demo

More Simulation and Physics in Games

- Human head: Sum-of-Gaussian algorithm for realistic real-time images of human skin
  - Gaussian convolution filter for blurring, image processing.
- Smoke: Fluid simulator for smoke
  - More realistic simulations in future games
- Frog: Real-time mesh deformation in combination with rendering is not possible on a CPU.
- Real-time simulations require heavy computational power

GPU Architecture

Overall System Architecture



NVIDIA GeForce 8800 GTX

- Price: 3000 NOK.
- ~520 GFLOPS performance
- Core clock: 675 MHz
- 681 million transistors
- Internal bandwith: 86.4 GB/s
- 128 stream processors
  - 1.35 GHz
- Memory
  - GDDR3
  - 768 MB
  - 1.8 GHz
- Programmable in high-level language

GeForce 8800 Architecture Overview

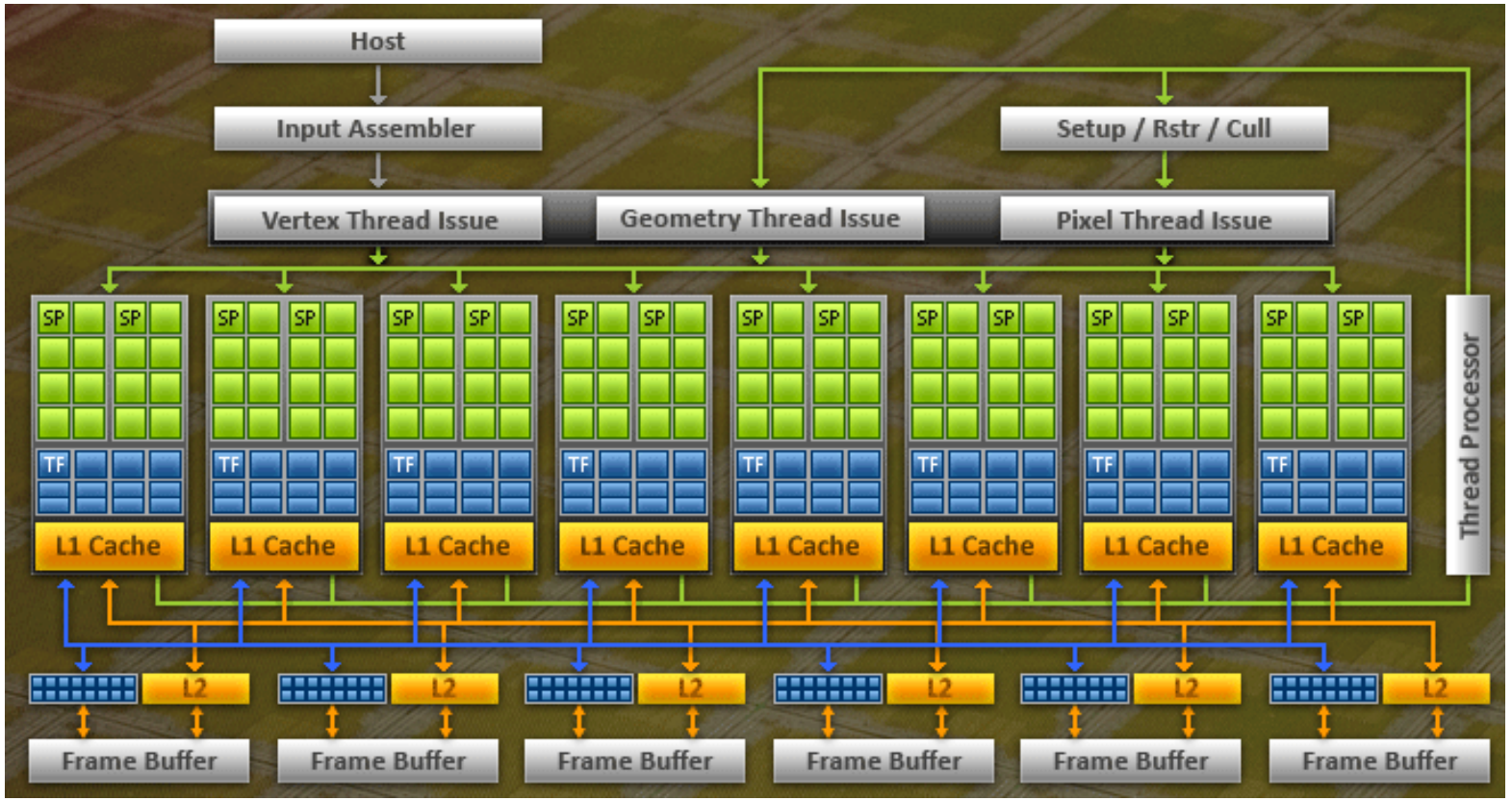


Image courtesy of rage3d.com

GeForce 8800 Architecture Overview (cont'd)

- 16 stream processors in each multiprocessor
- 128 stream processors in total.
- L1 cache shared between all stream processors in a multiprocessor

Image courtesy of rage3d.com

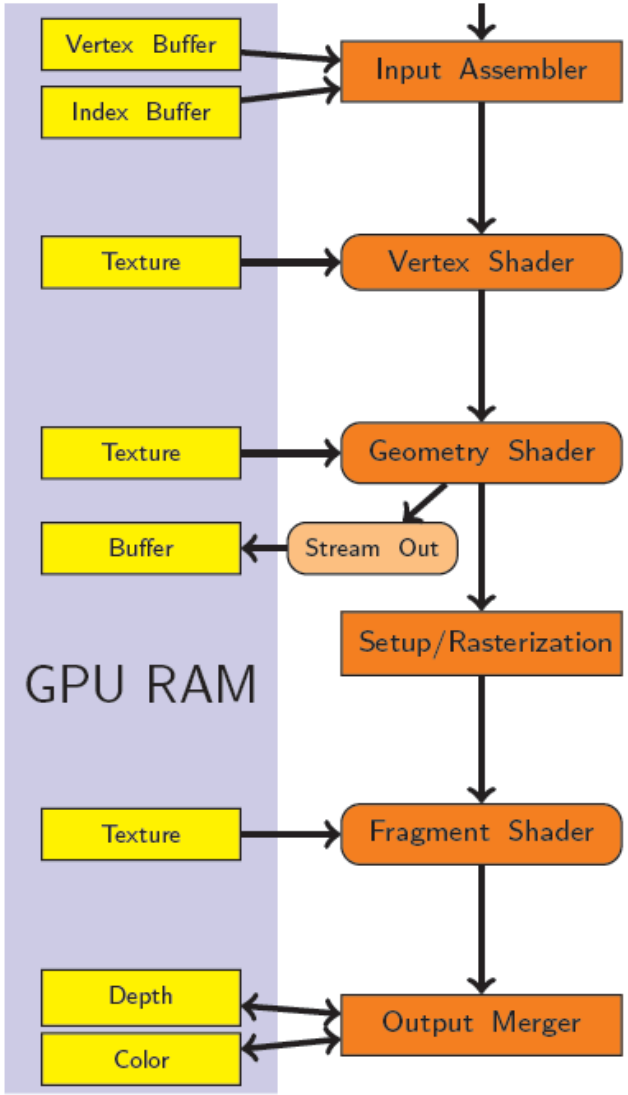
GPU Programming Models

Traditional Problems with GPU Programming

- The user must be an expert in computer graphics and its computational idioms to make effective use of the hardware
- The programming model is unusual
- Debugging is difficult
- Few libraries

Graphics Processing Pipeline

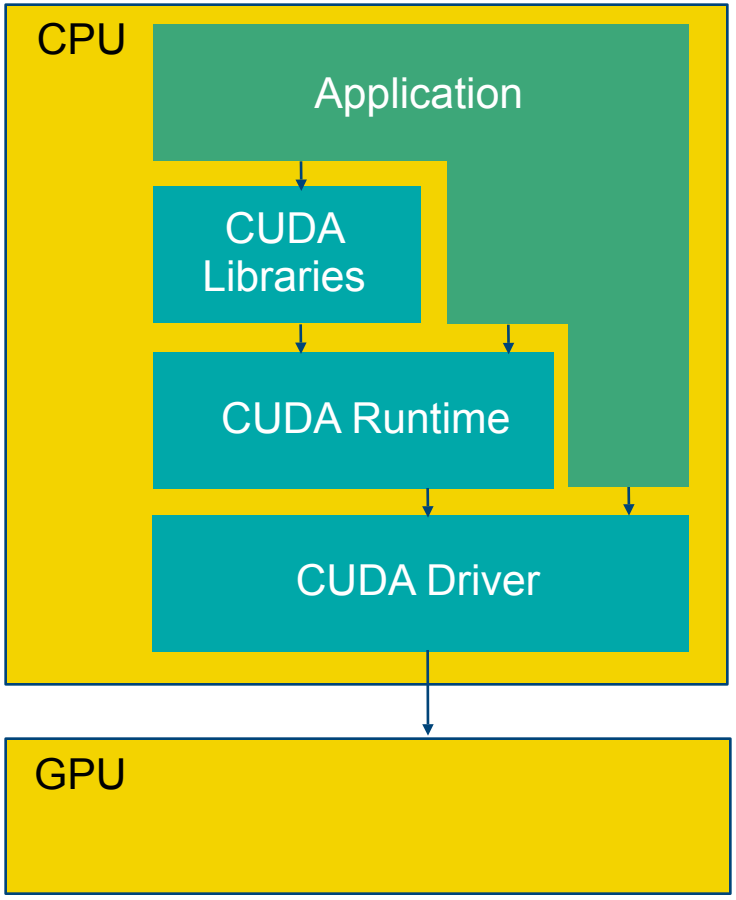
- This is the abstraction the user is exposed to by the DirectX and OpenGL graphics APIs
- Hard to understand for non-graphics programmers



Compute Unified Driver Architecture (CUDA)

- Software interface for utilizing the GPU as a data-parallel computing device developed by NVIDIA.
- No need for graphics API
- Available for NVIDIA GeForce 8000-series GPUs and the HPC Tesla-series.

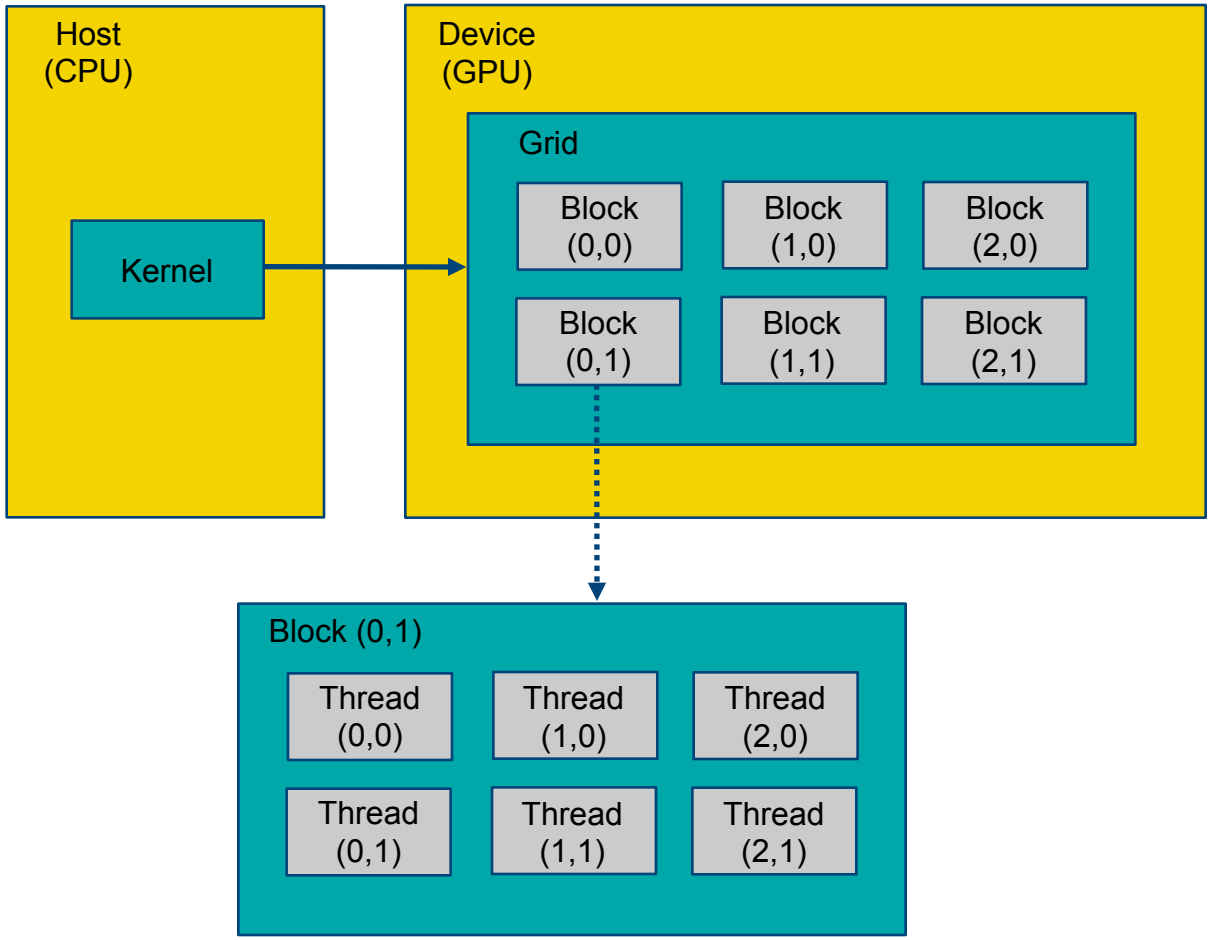
CUDA – Software Layers



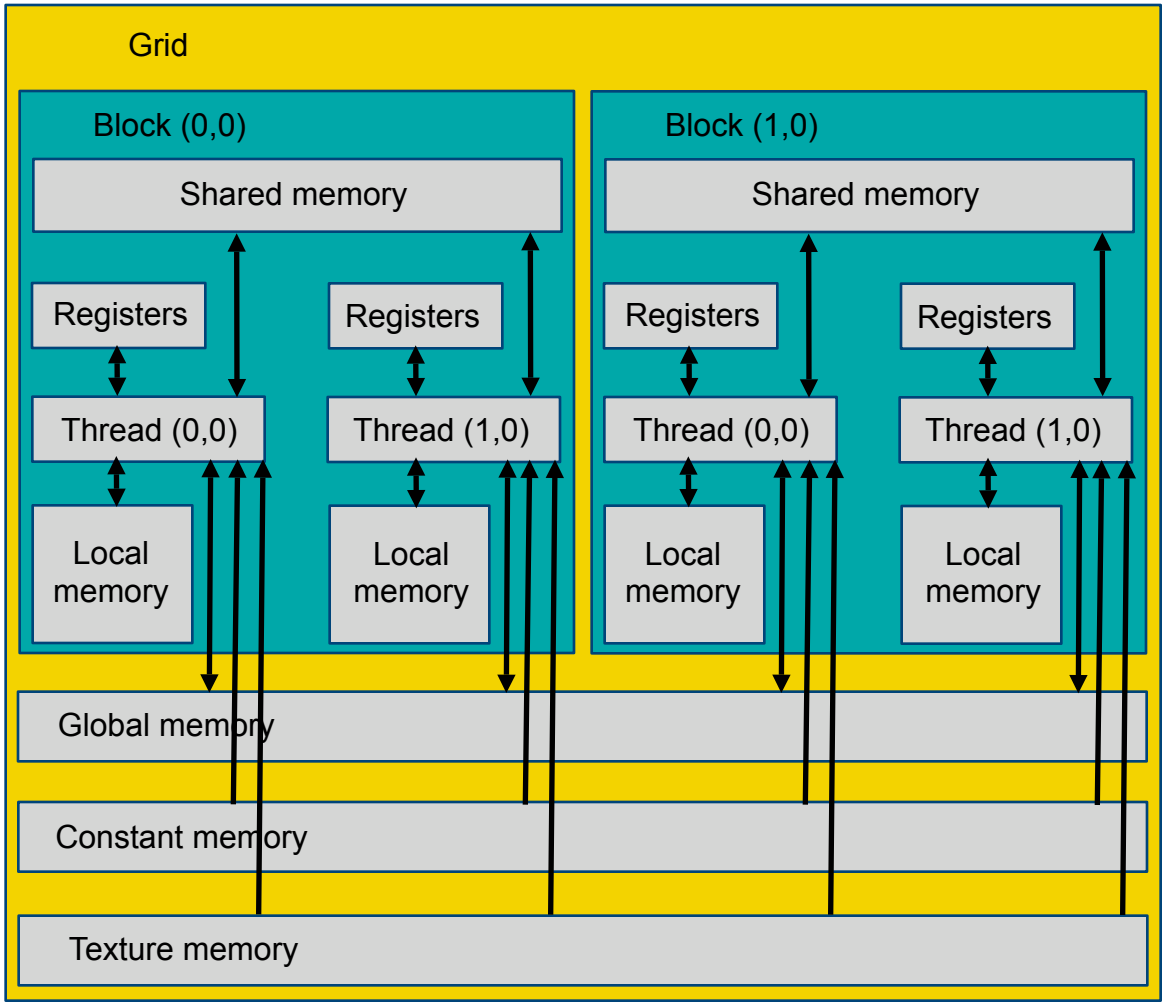
CUDA – Programming Model

- The GPU is viewed as a *device* that operates as a coprocessor to the CPU (*host*).
- A *kernel* is a function that is executed on the device as many different threads, doing the same type of operations.
- The batch of threads that executes a kernel is organized as a *grid* of *thread blocks*.
  - A thread block is a batch of threads that can communicate with each other.
  - All threads in a block runs on the same multiprocessor.
  - A block can contain a maximum number of threads, but blocks that execute the same kernel can be batched together into a grid of blocks.

CUDA – Programming Model



CUDA – Memory Model



Cuda Libraries

- BLAS library
  - Basic linear algebra subprograms
- FFT library
  - Fast Fourier Transform
- CUDPP
  - Parallel sum, sort and reduction algorithms

CTM / Brook+ from AMD

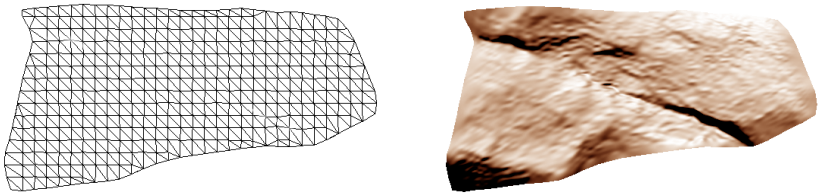
- Close to the Metal API
  - AMD's GPU programming model
  - HAL: Hardware Abstraction Level
    - device specific, driver like interface
  - CAL: Compute Abstraction Level
    - Core API device independent
    - Optimized multi-core implementation as well as optimized GPU implementations
    - Heterogeneous computing
- Brook+
  - Brool+ is an implementation by AMD of the Brook GPU spec on AMD's CAL. Higher level of abstraction.

Rapidmind

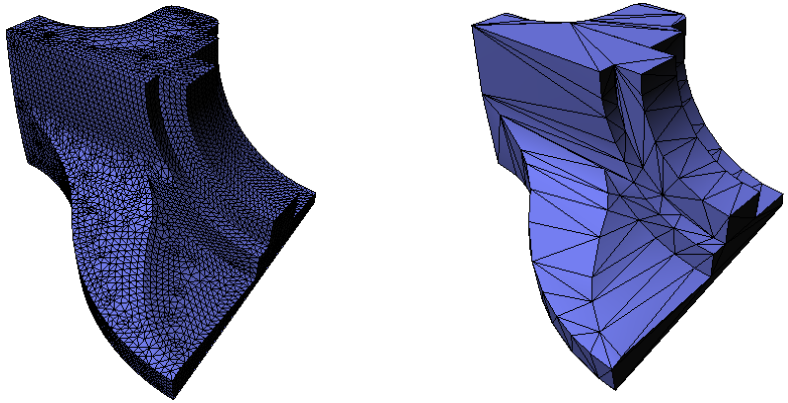
- Program multi-core systems using standard C++ / C. No specialized or vendor specific software development is required.
- Maps a single programming model to multiple hardware platforms.
  - GPUs, CPUs and the Cell processor.
- Higher level of abstraction, but also a performance penalty
- http://www.rapidmind.net

Examples of GPU Implementations

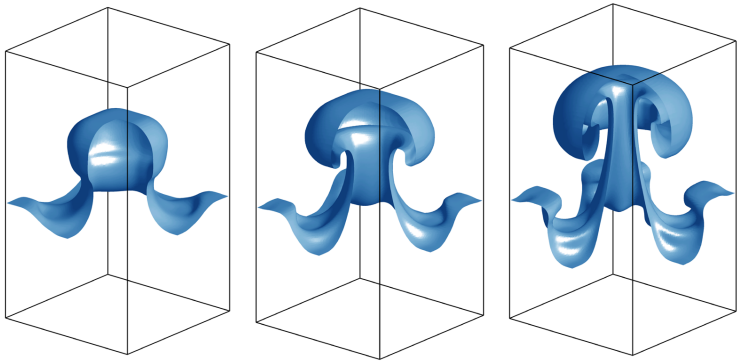
GPU Activities at SINTEF Applied Mathematics



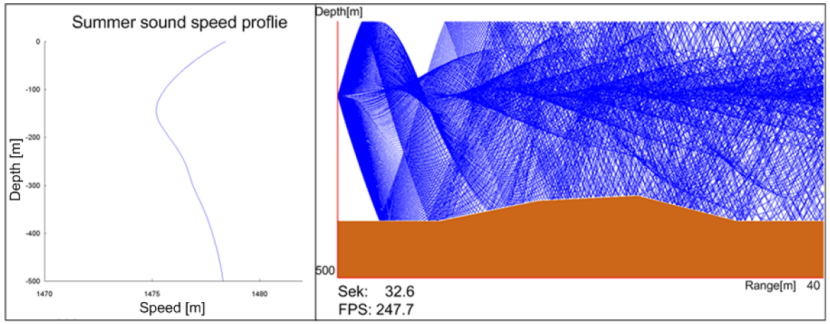
View-dependent tessellation



Preparation of finite element models (~5x)

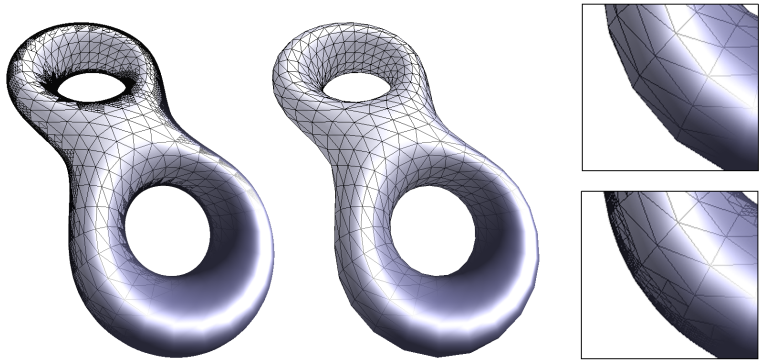


Soliving partial differential equations (~25x)

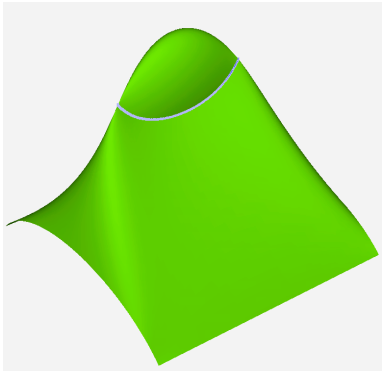


Marine aqoustics (~20x)

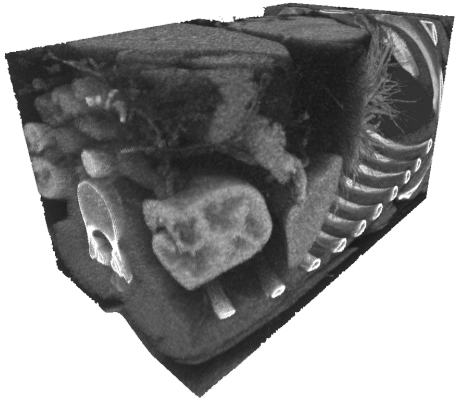
GPU Activities at SINTEF Applied Mathematics



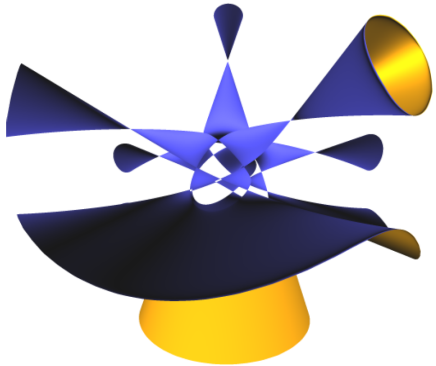
Silhouette refinement



Self-intersection detection of NURBS surfaces (~10x)

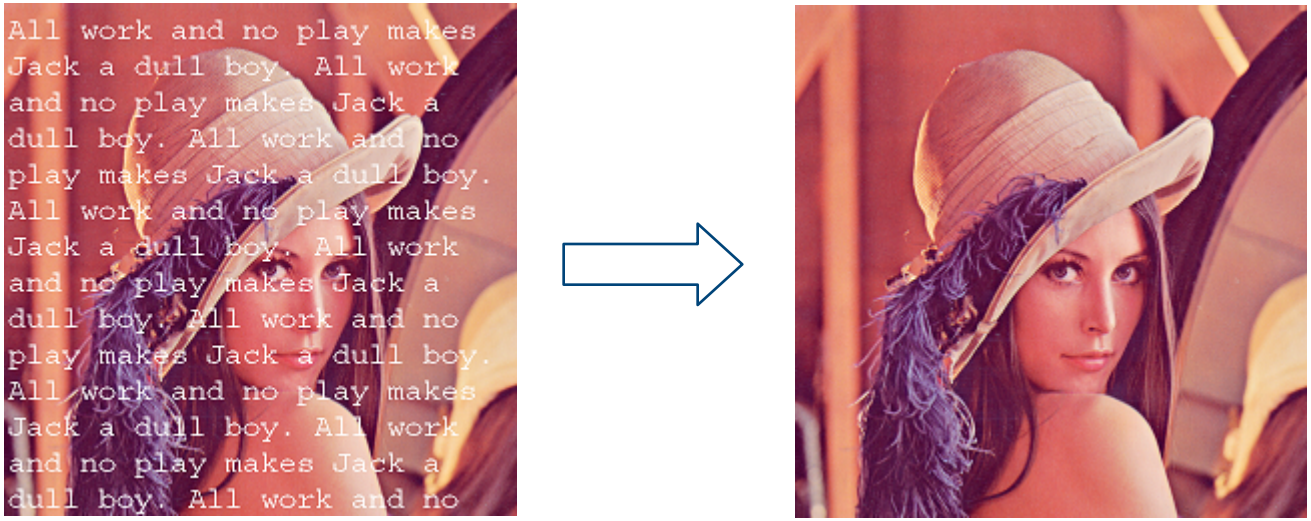


Registration of medical data (~20x)

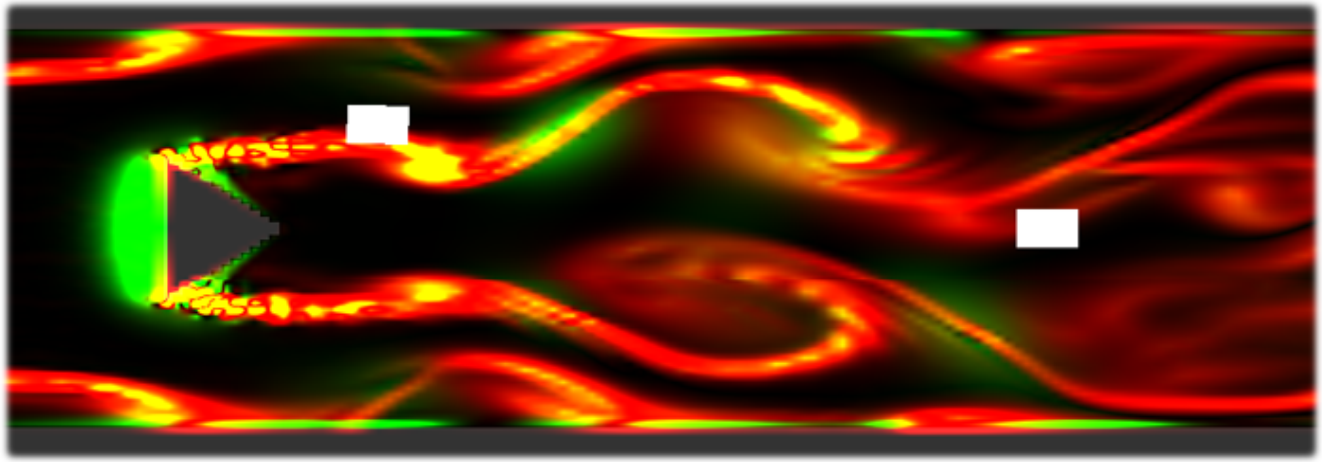


Visualization of algebraic surfaces

GPU Activities at SINTEF Applied Mathematics

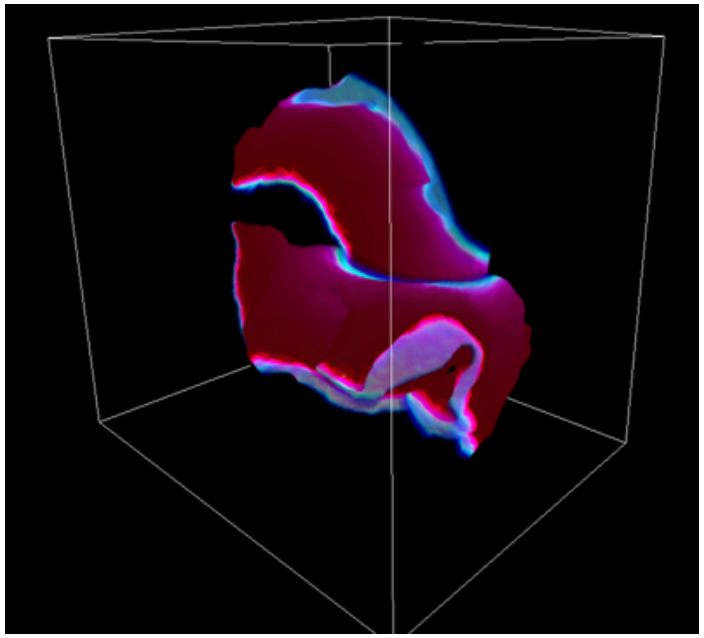


Inpainting (~400x matlab code)



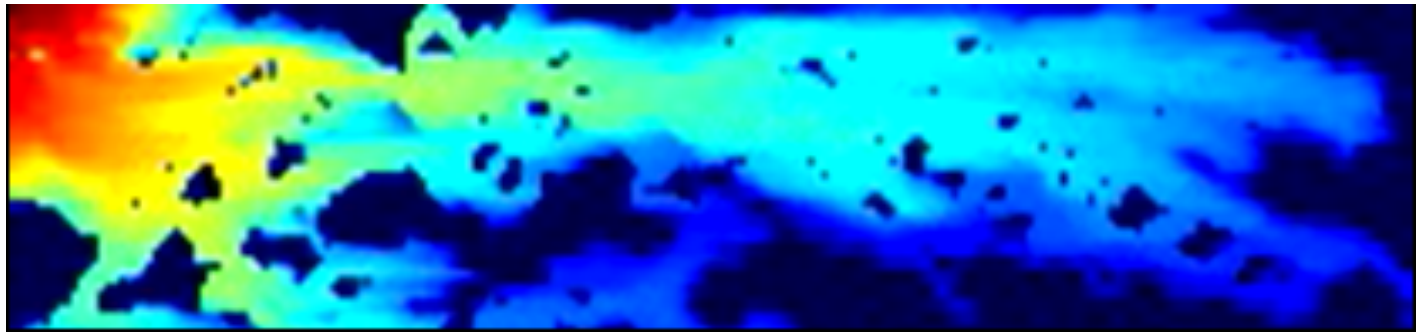
Navier-Stokes: Fluid dynamics

GPU Activities at SINTEF Applied Mathematics



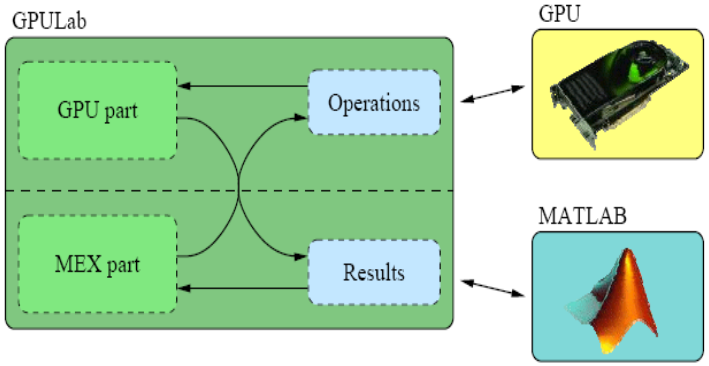
Volume visualization

Electric activity in a human heart.

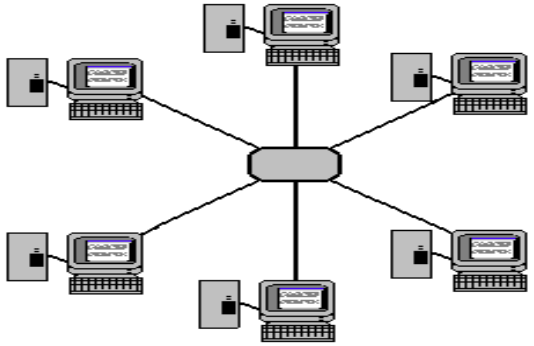


Water injection in a fluvial reservoir (20x)

GPU Activities at SINTEF Applied Mathematics



Matlab Interface to the GPU



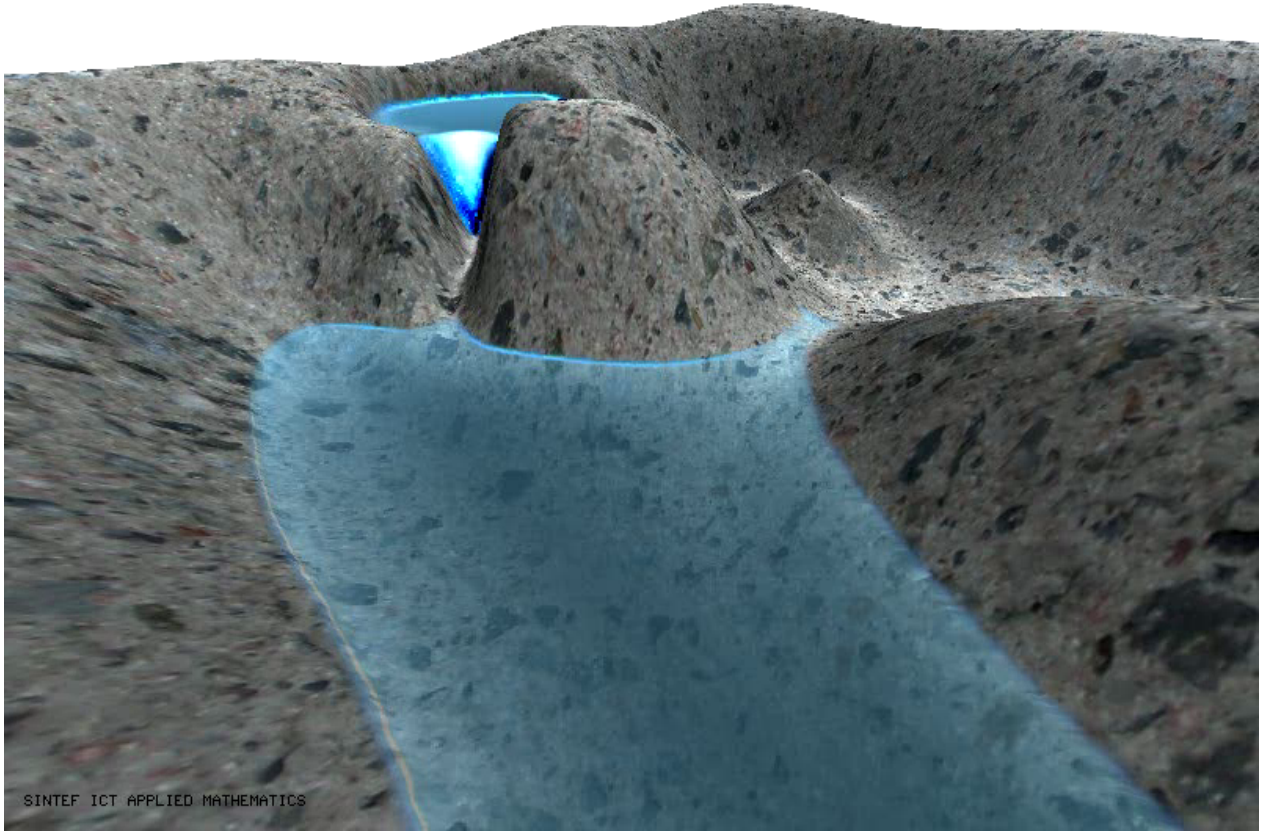
Cluster of GPU's

$$\begin{bmatrix} b_1 - a_{-\frac{1}{2}} & -a_{\frac{1}{2}} & 0 & 0 & 0 & \cdots & 0 \\ -a_{\frac{1}{2}} & b_2 & -a_{\frac{3}{2}} & 0 & 0 & \cdots & 0 \\ 0 & -a_{\frac{3}{2}} & b_3 & -a_{\frac{5}{2}} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -a_{n-\frac{5}{2}} & b_{n-2} & -a_{n-\frac{3}{2}} & 0 \\ 0 & \cdots & 0 & 0 & -a_{n-\frac{3}{2}} & b_{n-1} & -a_{n-\frac{1}{2}} \\ 0 & \cdots & 0 & 0 & 0 & a_{n-\frac{1}{2}} & b_n - a_{n+\frac{1}{2}} \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{n-2} \\ c_{n-1} \\ c_n \end{bmatrix}$$

Linear algebra / load balancing CPU - GPU

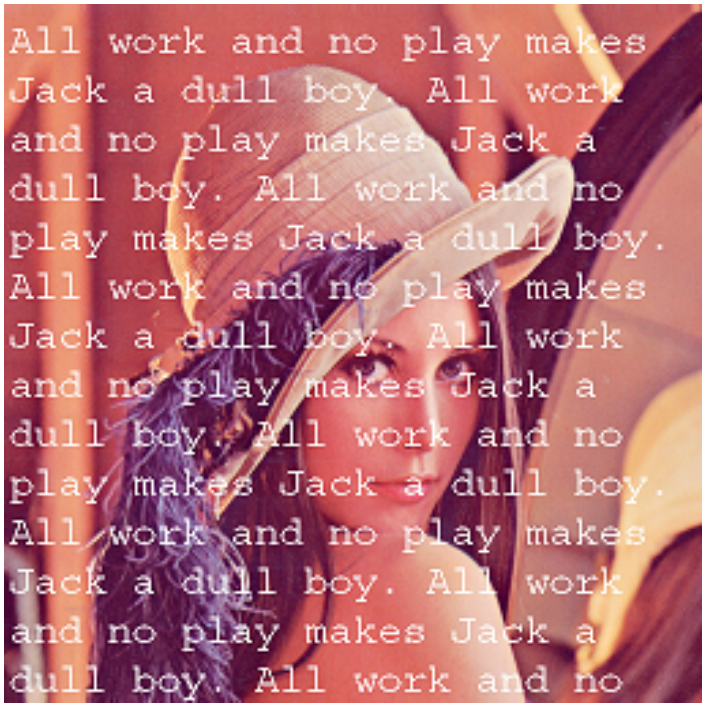
Shallow-Water Equations (~25x)

Shallow-Water Equations (~25x)



TV-Stokes Inpainting

- Demo: GPU-application running on the laptop



input image

result

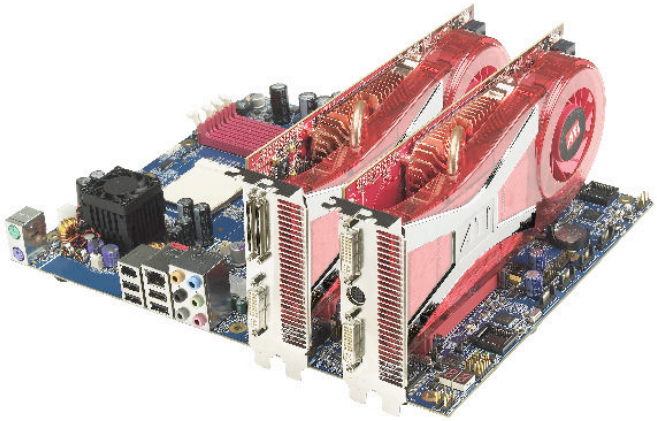
Looking Forward

Next generation architectures

NVIDIA SLI / AMD Crossfire

- Multiple graphics cards on a single motherboard

3-way SLI



Crossfire

Tesla – High Performance Computing

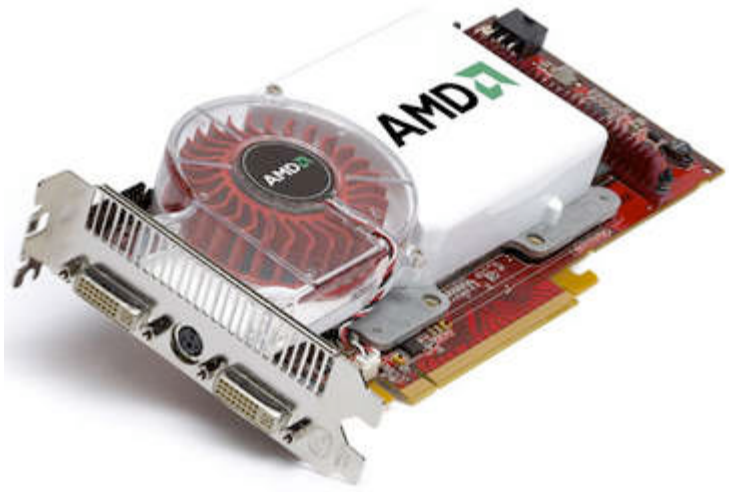
C870 – 1 GPU

D870 – 2 GPUs

S870 – 4 GPUs

AMD FireStream 9170

- First GPU with double precision floating point
- 320 stream processors
- Up to 500 GFLOPs single precision performance
- 2 GB on-board memory
- Less than 150 W power consumption
- 55 nm process technology



What about NVIDIA and double precision?

- At Supercomputing November 2006 NVIDIA announced a GPU with double precision support by the end of 2007
- NVIDIA now say they are going to release GPUs with double precision in 2008
- Only the Tesla HPC cards and the high-end Quadro cards will support double precision

AMD HD3870X2

- Released within a week
- Two GPUs on one graphics card
- Early benchmarks:

  ~40% faster than GeForce 8800 Ultra (3DMark06)

GeForce 9800 GX2

- Launch in february / march
- GeForce 9800 GX2, two GPUs
  - Two 65nm 8800 cores
  - 256 stream processors
  - 1 GB on board memory
- 30% faster than GeForce 8800 Ultra
- No new functionality
- Next generation GPUs probably in Q3 2008.

Intel's Larrabee GPU Project

- Uses a derivative of the x86 instruction set for its GPU cores instead of a traditional graphics-oriented instruction set
  - Expected to be more flexible than traditional GPUs
  - Designed for GPGPU or stream processing
- Will probably be released in late 2008 or early 2009
- Intel have never been able to compete with NVIDIA or AMD / ATI when it comes to processing power. Will they succeed this time?

Intel's Larrabee GPU Project (cont'd)

- Ten or more cores per die
- Each core have its own L1 cache (instruction / data)
- Each core supports four simultaneous threads
- Subset of the x86 instruction set architecture + GPU specific extensions
- All cores will share a large L2 cache
  - The L2 cache will be used for communication between cores
- Fixed-function unit
  - Will vary between the different Larrabees
  - A rasterizer in the GPU-based chip

80-Core Terascale Project

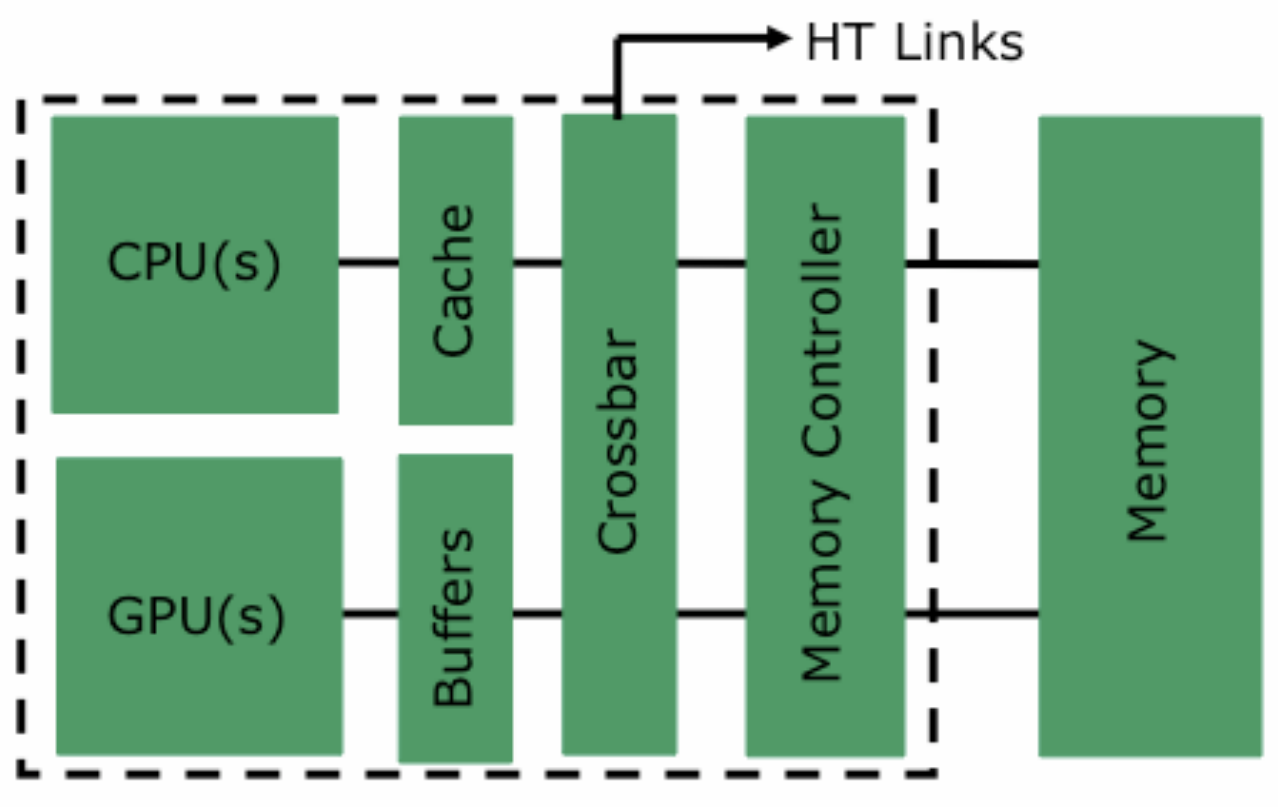
- Terascale chip is not a product, but a research initiative
- The goal of this initiative is not to produce a shipping product, but to explore and develop various technologies that might eventually find their way into future, as-yet-undesigned Intel products

AMD's Fusion Project

- Heterogeneous multi-core processor combining GPU and CPU in one chip
- The FireStream GPU is one step towards AMD's Fusion project.
- Target release 2009

"The introduction of graphics processing engines into AMD's chips is the *biggest microprocessor evolution* since the introduction of x86-64 concept back in 1999" – Phil Hester, chief technology officer at AMD

AMD's Fusion Project (cont'd)



Information transfer between CPU and GPU will be significantly faster because the information will not need to travel on the PCIe bus.

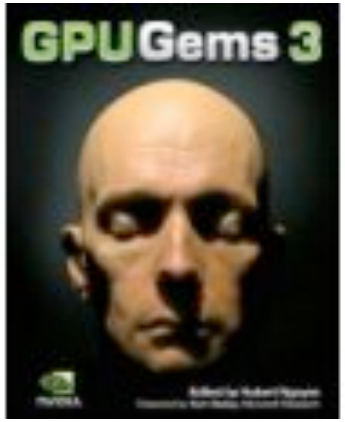
AMD's Fusion Project (cont'd)

Conclusion

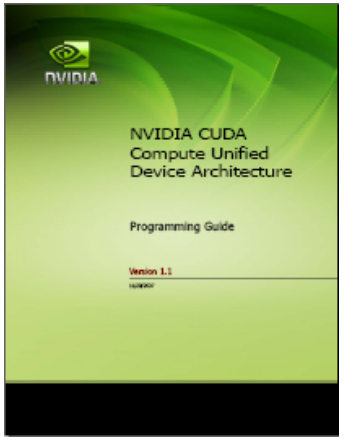
- Use all available resources!
  - If you have a GPU in your machine, you cannot let 500 GFLOPS stay idle.
- Heterogeneous architectures offers the most scalable solution.
  - Either as different type of cores inside the same chip
  - Or as dedicated accelerator cards like GPUs now
- Will GPUs be the data-parallel accelerators of the future?
  - Probably, because the game industry demands more computational power.
  - High demand for GPUs gives lower prices.
- We should start to redesign our algorithms now ...

Further Reading

- GPU Gems 2 & 3



- http://www.nvidia.com/cuda



- http://www.gpgpu.org

Thank You!