Technology for a better society

Parallel and Heterogeneous Computing

Trond Hagen, SINTEF ICT

Collab Workshop, April 12, 2010

Outline

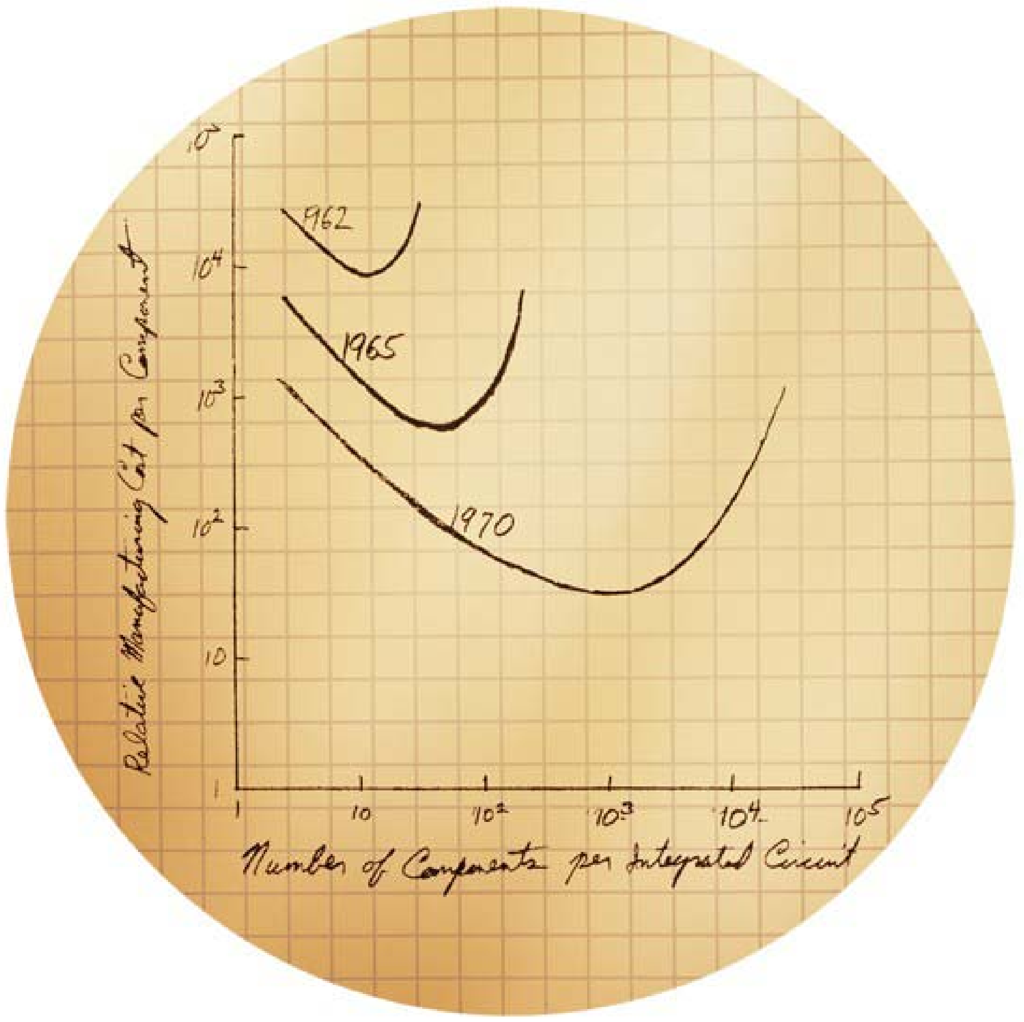
- Hardware Evolution
- Parallel Computing
- Graphics Processing Units
- Algorithm Design
- GPU-based Cloud Computing

Hardware Evolution

Moore's Law

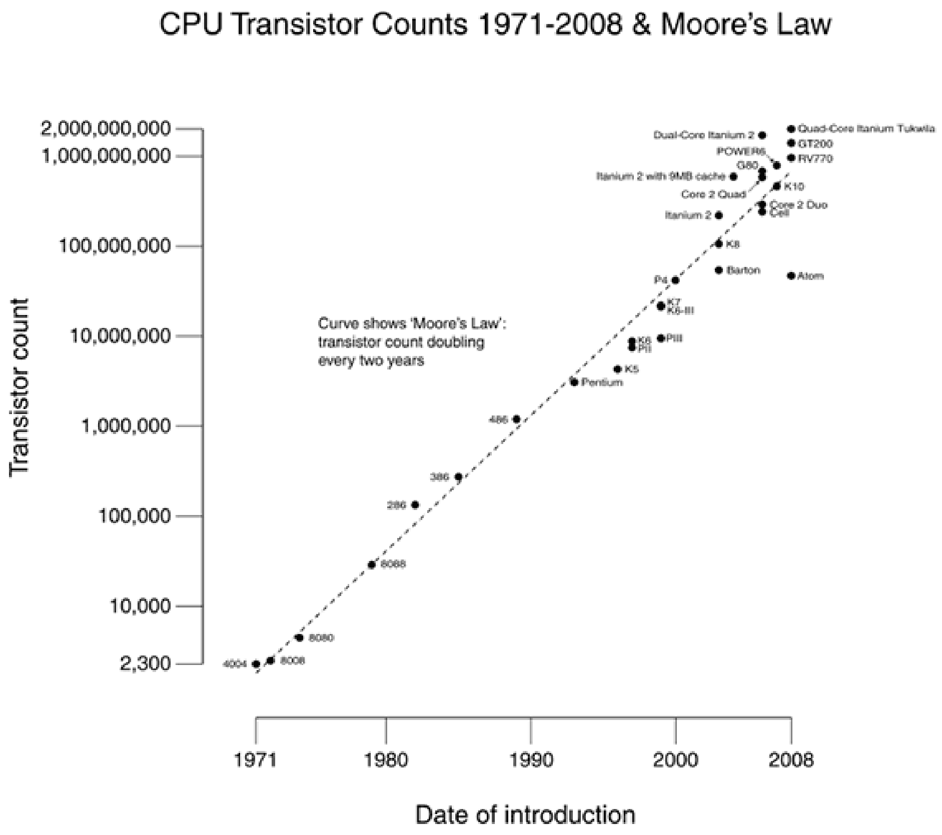
- *"The number of transistors on an integrated circuit for minimum component cost doubles every 24 months"*

  – Gordon Moore.



Brief history of computing: 1970-2004

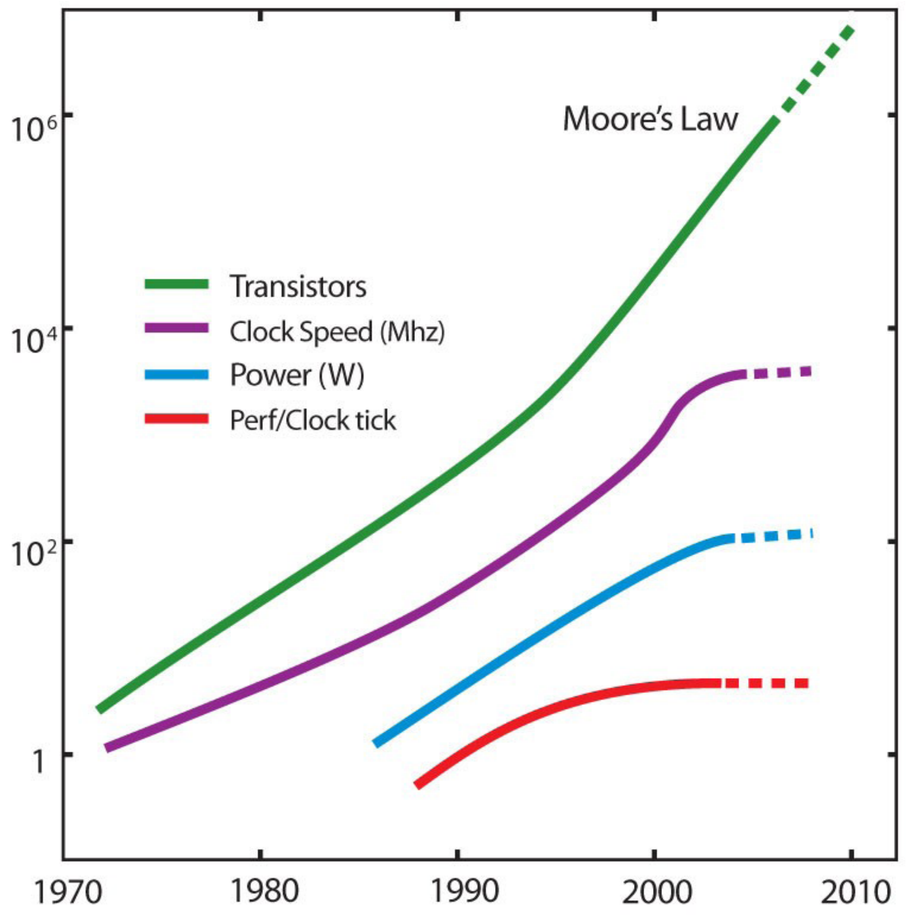
- 1971 (4004)
  2300 transistors
  1 x 0.000740 GHz
- 2004 (Pentium 4 Prescott)
  125 000 000 transistors
  1 x 4 GHz
- 2008 (Core i7 Quad)
  731 000 000 transisitors
  4 x 3.33 GHz
- 2010+ (Itanium Tukwila/Poulson)
  2 000 000 000+ transistors
  8 x 2 GHz?



What happened in 2004?

- Pentium 4 original target frequency: **10 GHz**

- Increasing the frequency has several implications (three walls):
  - Memory
  - Instruction Level Parallelism
  - Power



Memory Wall

- 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

Instruction Level Parallelism Wall

- Difficulty to find enough parallelism in the instructions stream of a single process to keep higher performance processor cores busy.

- ILP:
  - Instruction pipelining (execution of multiple instructions can be partially overlapped)
  - Superscalar (parallel instruction pipelines)
  - Branch prediction (predict outcome of decisions)
  - Out-of-order execution

Power Consumption Wall

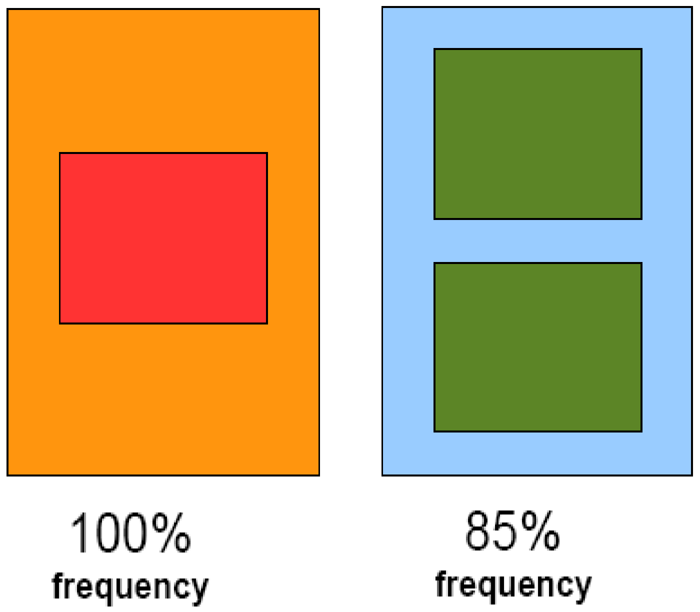
- When the frequency increases, the power consumption increases disproportionately

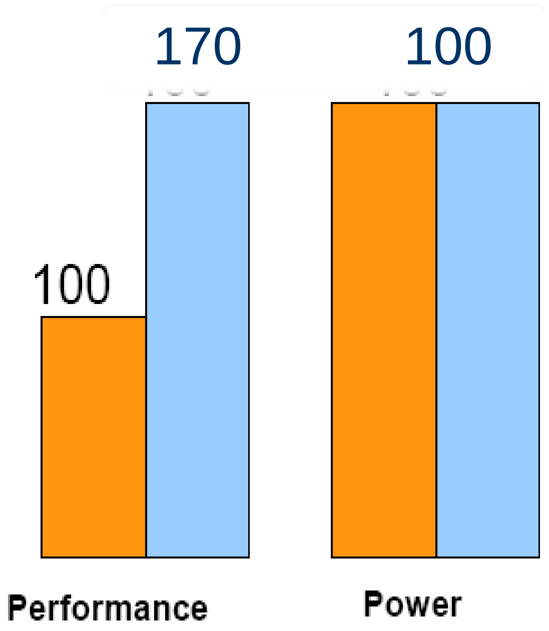
- Power density relative to **cube of frequency.**
  - Frequency limited to around 4 GHz

$$P = C \cdot \rho \cdot f \cdot V_{dd}^{2} \approx V_{dd}^{3}$$

P is the power density in watts per unit area, *C* is the total capacitance, *ρ* is the transistor density, *f* is the processor frequency, and V_{dd} is the supply voltage.

How can multi-core help?





- Two cores running at 85% frequency vs. one core at 100% frequency:
  - 100% power consumption
  - 170% performance

Why not 100-core Pentium IV?

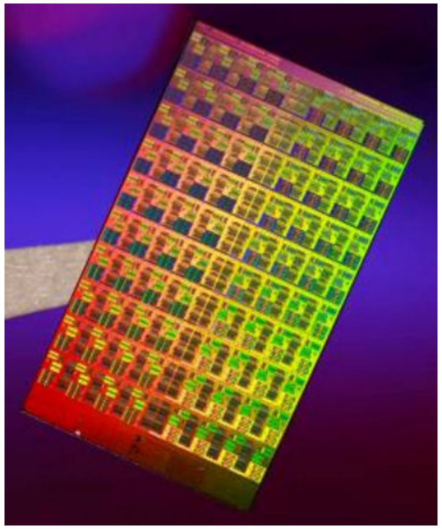
- Deep pipelines,
- heavy ILP use, and
- huge caches drain a **lot of power.**

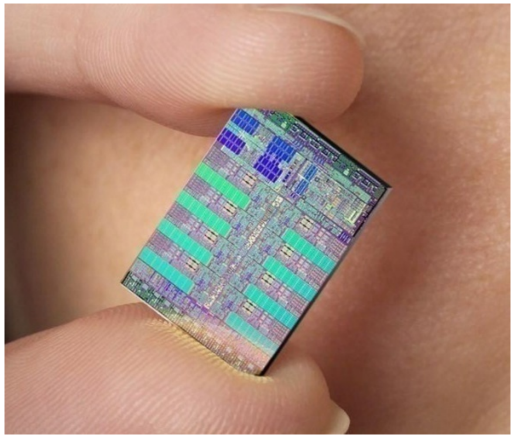
About 50% of the cost of a supercomputer is the electricity bill!

- CPUs have low CPI (cycles per instruction), but high “WPI” (Watts per Instruction)
- Accelerator cores have shallow pipelines, low or no ILP, and small or no caches!

Accelerator cores

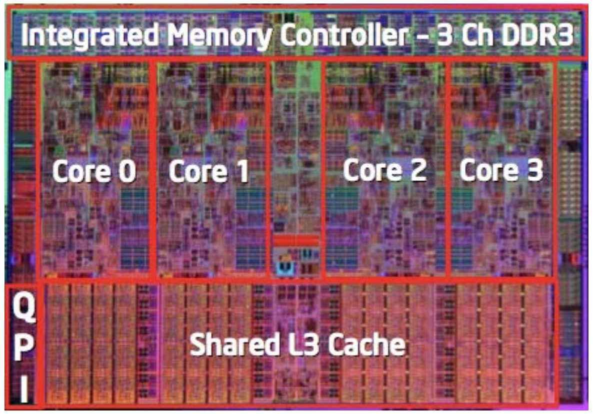
- Modern accelerators are power efficient!
  - Intel Teraflops research chip (AKA Polaris, 80-core): 1 teraflops using 97 watt (Mattson 2008)
  - Core i7 Quad offers 100 gigaflops for 130 watt...
  - GPUs, FPGAs, and the Cell BE are examples of commodity accelerators



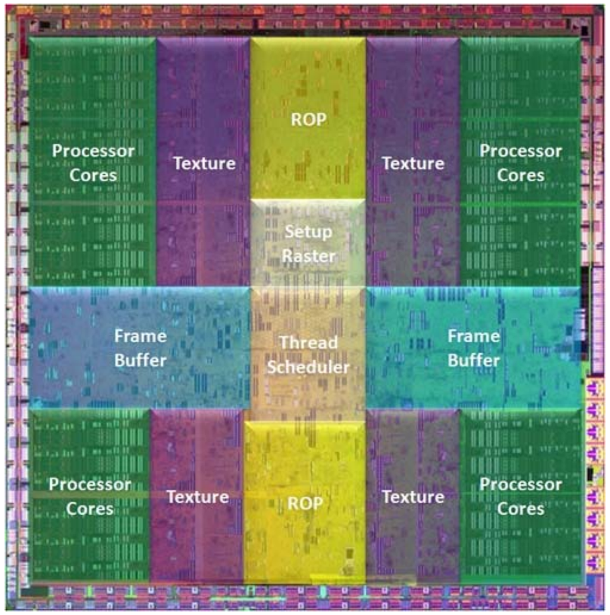


Heterogeneous Computer

- A *heterogeneous computer* is a tightly coupled system of processing units with distinct characteristics.
- A modern desktop or laptop computer is an example of such a system, as most systems include both a task-parallel, multi-core CPU and one or more data-parallel processors in the form of programmable graphics processing units(GPUs).



CPU (Intel Nehalem)



GPU (NVIDIA GeForce 280)

Parallel Computing

Task vs. Data Parallelization

Task parallelization:

- Each processor executes a different thread (or process) on the same or different data.
- CPUs

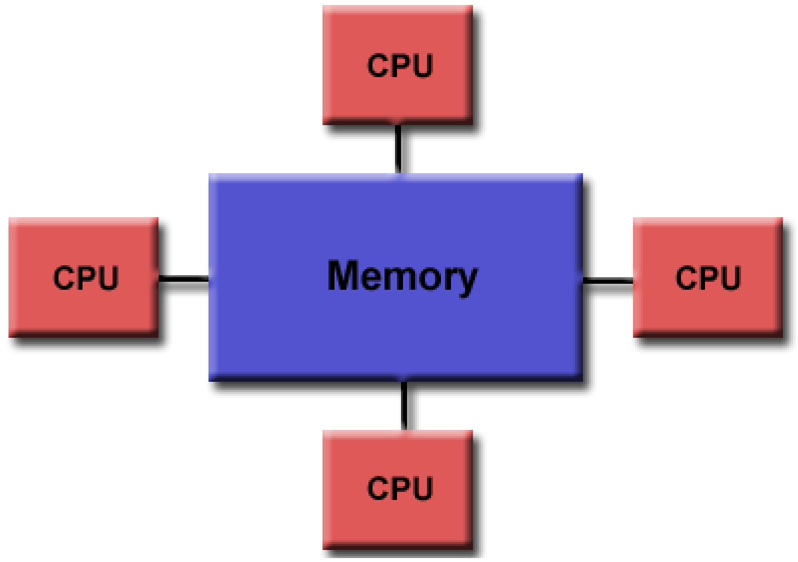
Data parallelization:

- Each processor performs the same task on different pieces of distributed data
- GPUs

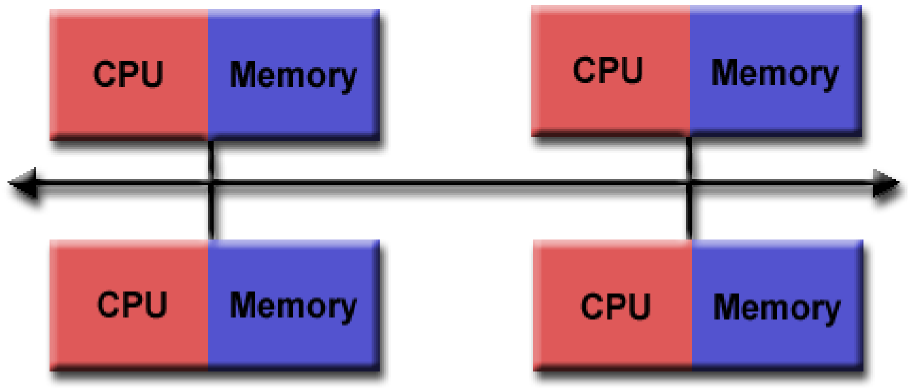
Shared Memory vs. Distributed Memory

- Multiple processors can operate independently but share the same memory resources
- OpenMP

- Distributed memory systems require a communication network to connect inter-processor memory
- MPI



Shared memory system

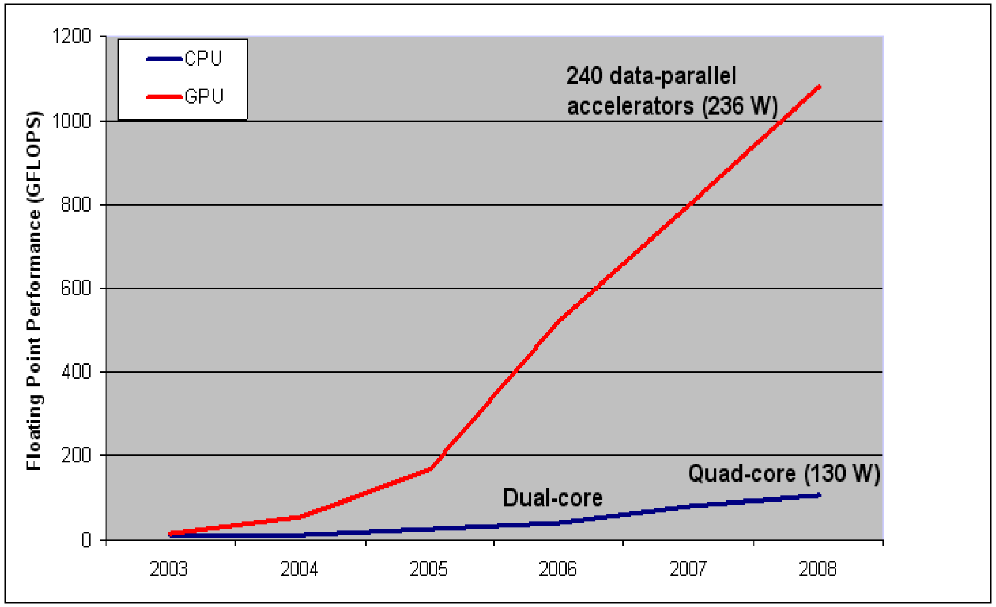


Distributed memory system

Graphics Processing Units

Graphics Processing Unit (GPU)

- Background: Game industry
- Massive parallel architecture (>500 cores per chip)
- Typically 10-50 times speedup compared to CPU applications

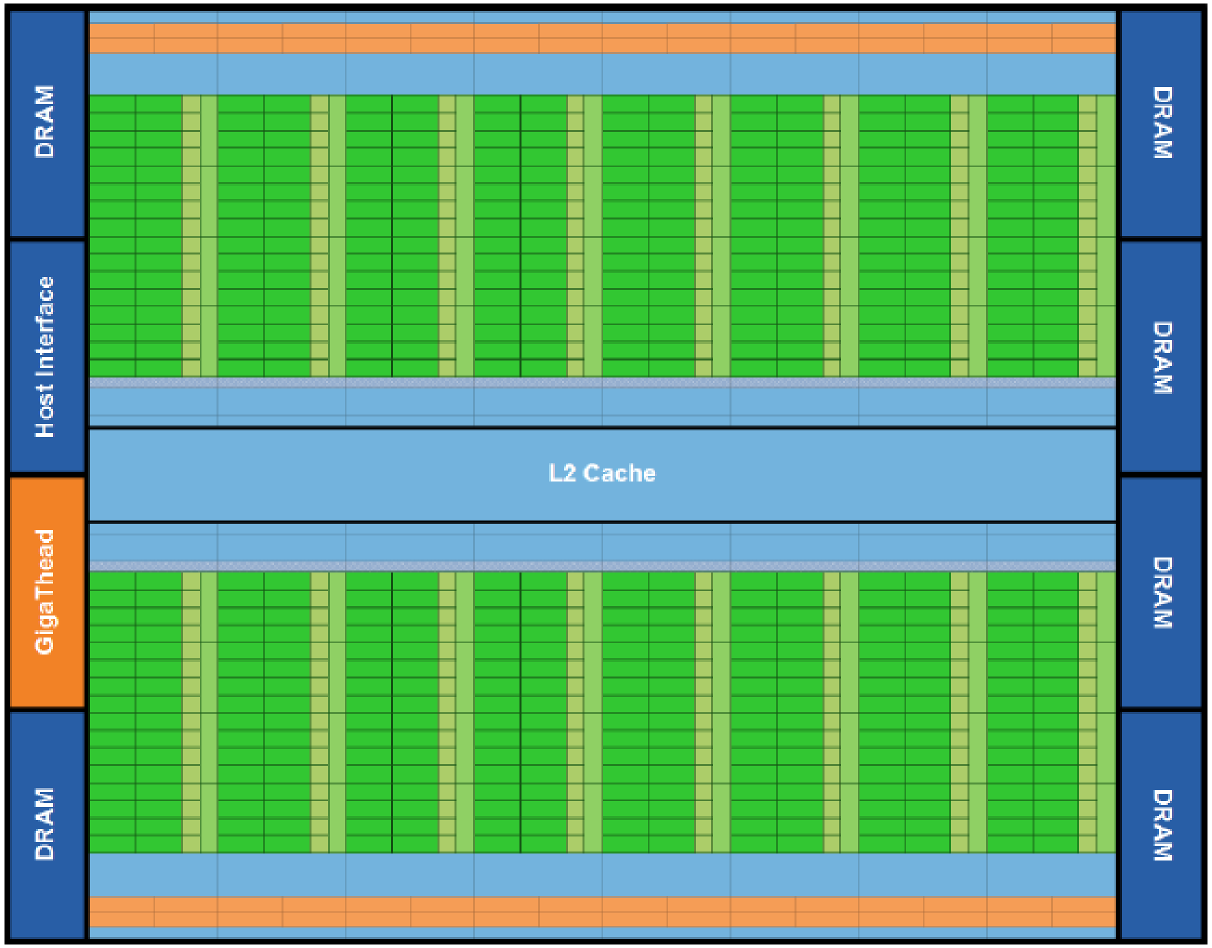


NVIDIA Fermi GPU

- 3 billion transistors
- 512 cores
- Up to 1 Terabyte of GPU memory
- 1.5 Teraflops performance
- 40 nm manufacturing process

NVIDIA GTX 480

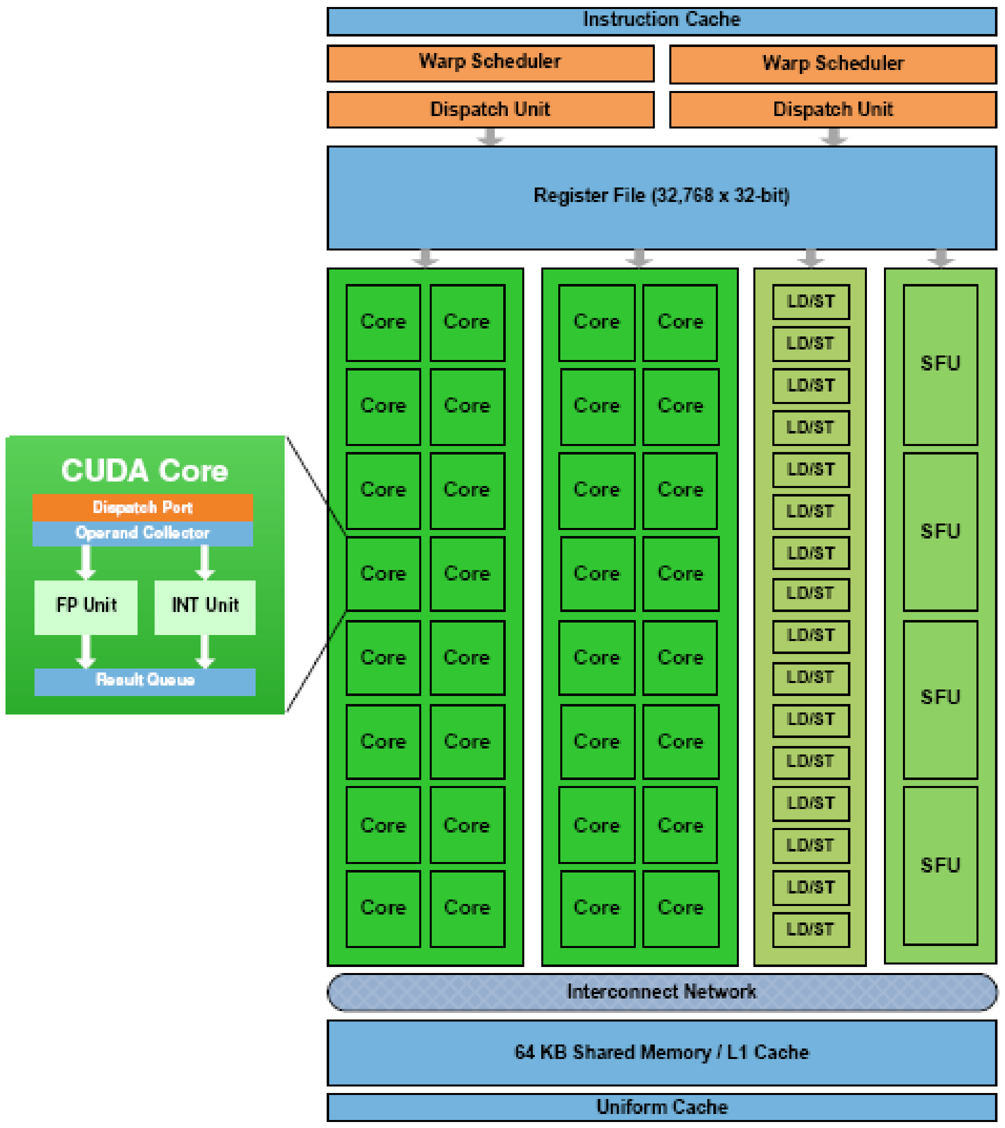
NVIDIA Fermi Architecture



16 streaming multiprocessors are positioned around a common L2 cache.
Each SM is a vertical rectangular strip that contain an orange portion (scheduler and dispatch), a green portion (execution units), and light blue portions (register file and L1 cache).

Fermi Architecture – Streaming Multiprocessor

- 32 cores per SM
- 64 KB of RAM for shared memory and L1 cache
- Double precision speed at 50% of peak single precision
- New IEEE 754-2008 floating point standard, surpassing even the most advanced CPUs.
- Newly designed integer ALU optimized for 64-bit and extended precision operations.

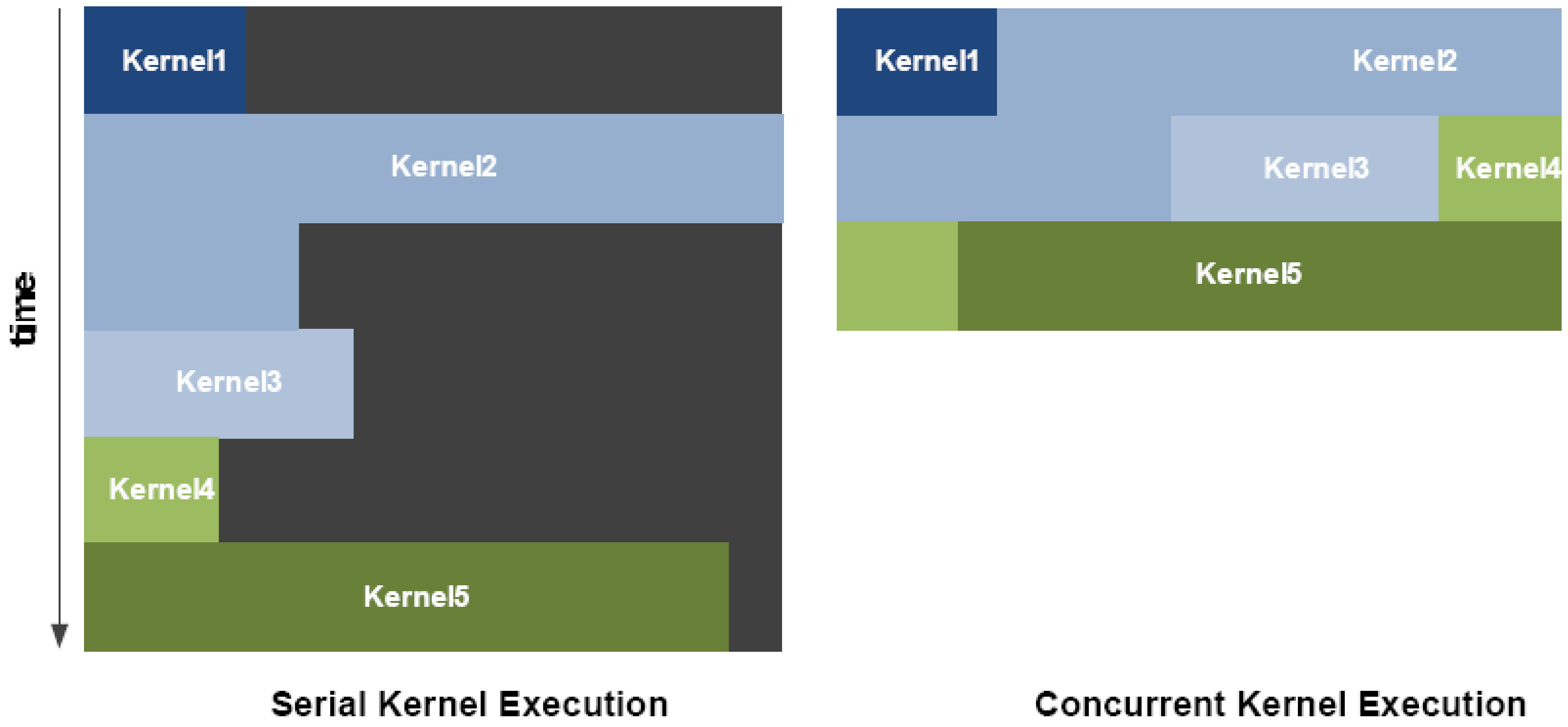


Cached Memory Hierarchy

- First GPU to support a true cache hierarchy in combination with on-chip shared memory
- L1 cache per SM (32 cores)
- Unified L2 cache (768KB)
  - Fast, coherent data sharing across all cores.

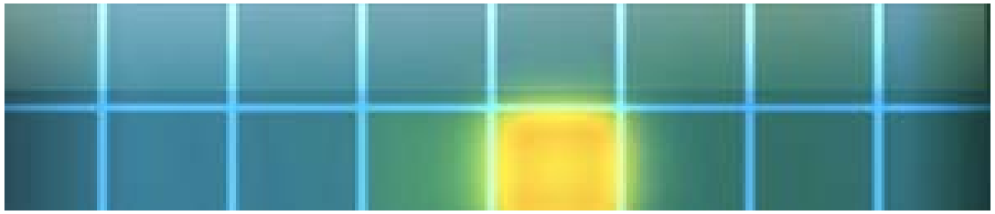
Hardware Thread Scheduler

- Concurrent kernel execution and faster contex switch



Full ECC Support

- First GPU architecture to support ECC
- Detects and corrects errors before system is affected
- Protects register files, shared memories, L1 and L2 cache, and DRAM



Enhanced Software Support

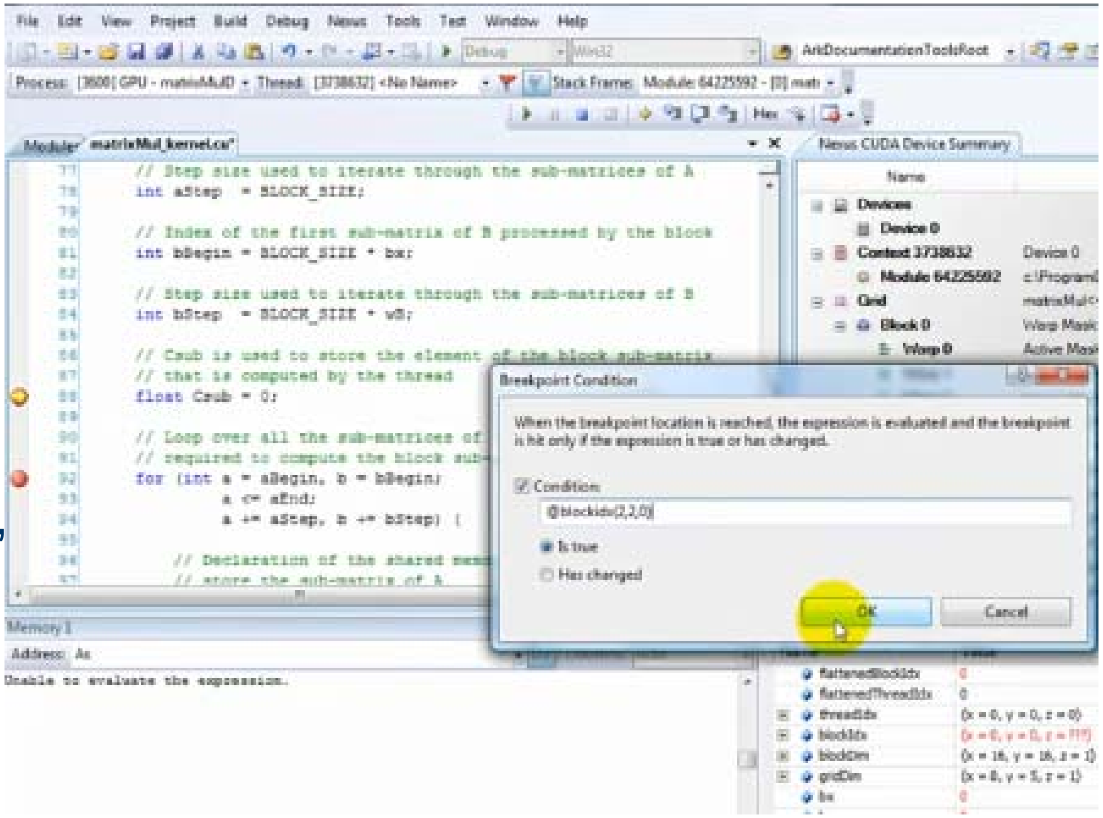
- Full C++ support
  - Virtual functions
  - Try / catch hardware support
- System call support
  - printf() etc.

Programming Frameworks

- Compute Unified Device Architecture (CUDA)
  - Developed by NVIDIA
  - Only available from NVIDIA GeForce 8 series and above
- Open Computing Language (OpenCL)
  - Specified by the Khronos group
  - Cross platform
- DirectCompute
  - Part of Microsoft DirectX

NVIDIA Parallel Nsight

- Debug, profile, and analyze GPU code
- Integrated with Microsoft Visual Studio 2008
- Will support CUDA C, OpenCL, DirectCompute, Direct3D, and OpenGL.


File Edit View Project Build Debug Nexus Tools Test Window Help
Process: [3600] GPU - matrixMulD
Thread: [3738632] <No Name>
Stack Frame: Module: 64225592 - [0] matr
ArkDocumentationToolsRoot
Modules
matrixMul_kernel.cu*
Nexus CUDA Device Summary
Name
Devices
Device 0
Context 3738632
Device 0
Module 64225592
Grid
Block 0
Warp 0
Memory 1
Address:
OK
Cancel

```
    // Step size used to iterate through the sub-matrices of A
    int aStep  = BLOCK_SIZE;

    // Index of the first sub-matrix of B processed by the block
    int bBegin = BLOCK_SIZE * bx;

    // Step size used to iterate through the sub-matrices of B
    int bStep  = BLOCK_SIZE * wB;

    // Csub is used to store the element of the block sub-matrix
    // that is computed by the thread
    float Csub = 0;

    // Loop over all the sub-matrices of
    // required to compute the block sub-
    for (int a = aBegin, b = bBegin;
             a <= aEnd;
             a += aStep, b += bStep) {

        // Declaration of the shared mem
        // store the sub-matrix of A
```

```
Unable to evaluate the expression.
```

Breakpoint Condition

When the breakpoint location is reached, the expression is evaluated and the breakpoint is hit only if the expression is true or has changed.

☑ Condition: @blockIdx(2,2,0)

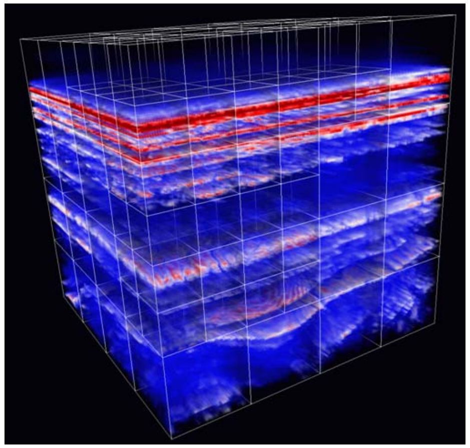
[✓ Is true] Has changed

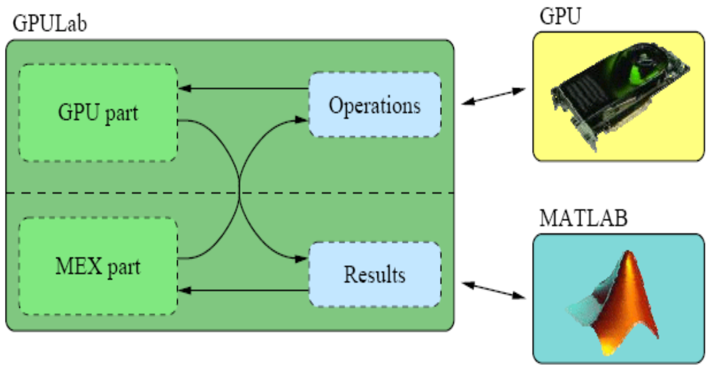
flattenedBlockIdx	0
flattenedThreadIdx	0
threadIdx	{x = 0, y = 0, z = 0}
blockIdx	{x = 0, y = 0, z = ???}
blockDim	{x = 16, y = 16, z = 1}
gridDim	{x = 8, y = 5, z = 1}
bx	0

Application Showcase (SINTEF)

Isosurface visualization

Simulations





Seismic/ reservoir processing

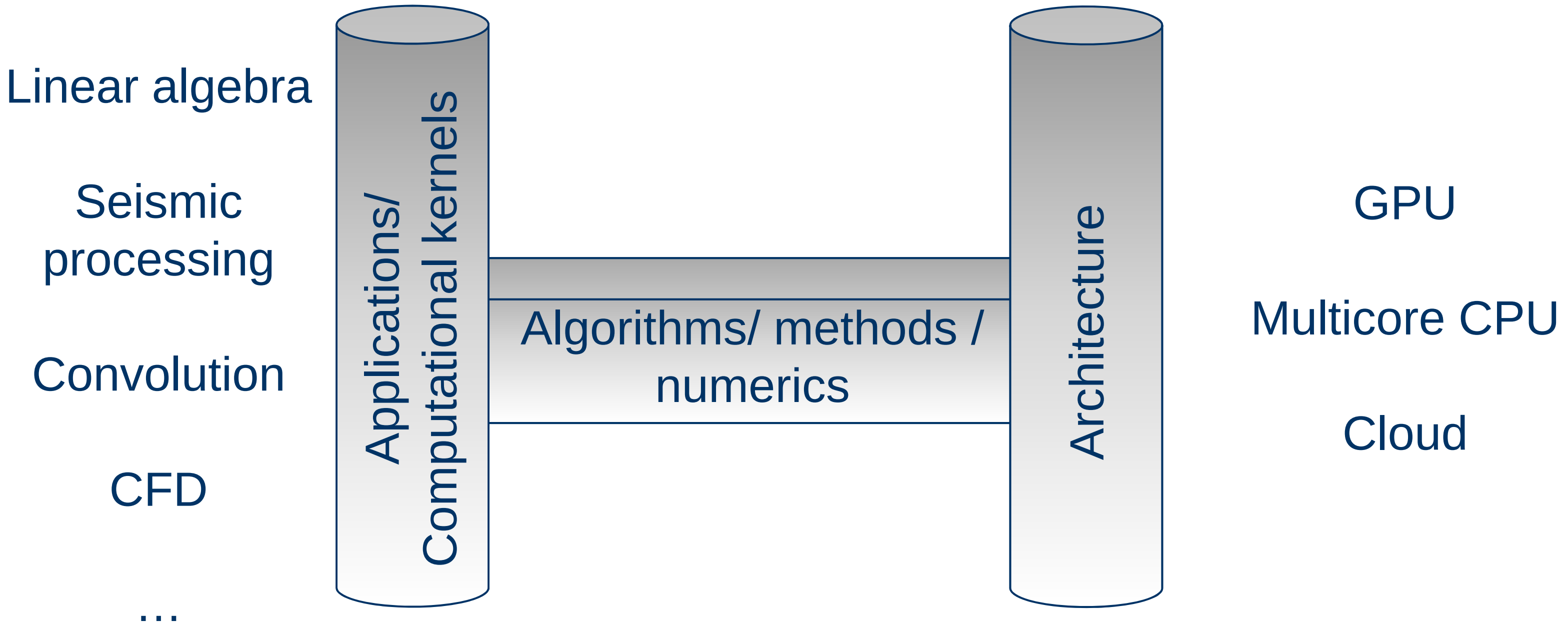
Ultrasound processing

Matlab interface for GPUs

Algorithm Design

Architecture and Algorithm Interaction

The key to performance is to understand the architecture and algorithm interaction.



”Multicore computers shift the burden of software performance from chip designers and processor architects to software developers”
(J. Larus in *Spending Moore's Dividend*, CACM May 2009)

Computations: Algorithms and hardware advances doubles the speedup

Solving linear systems of equations:

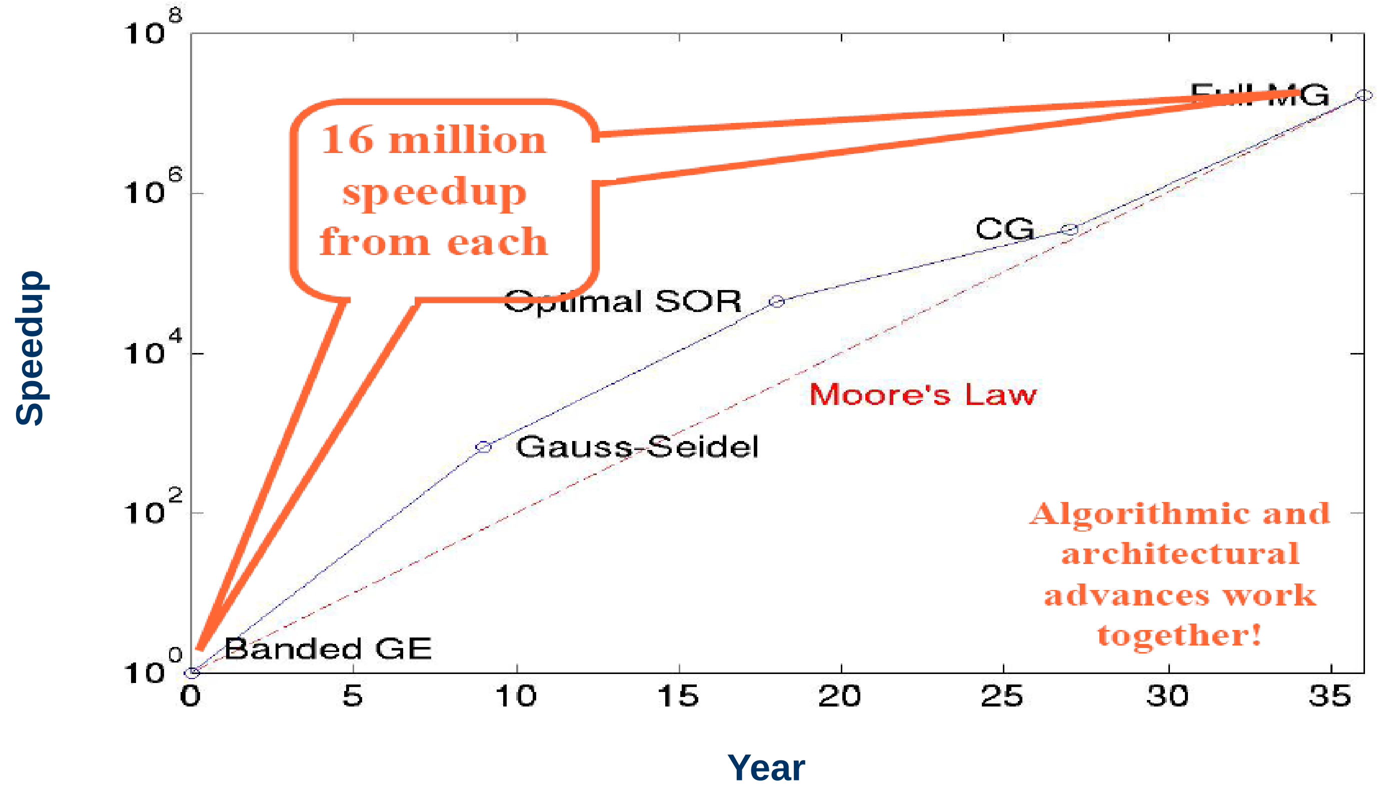
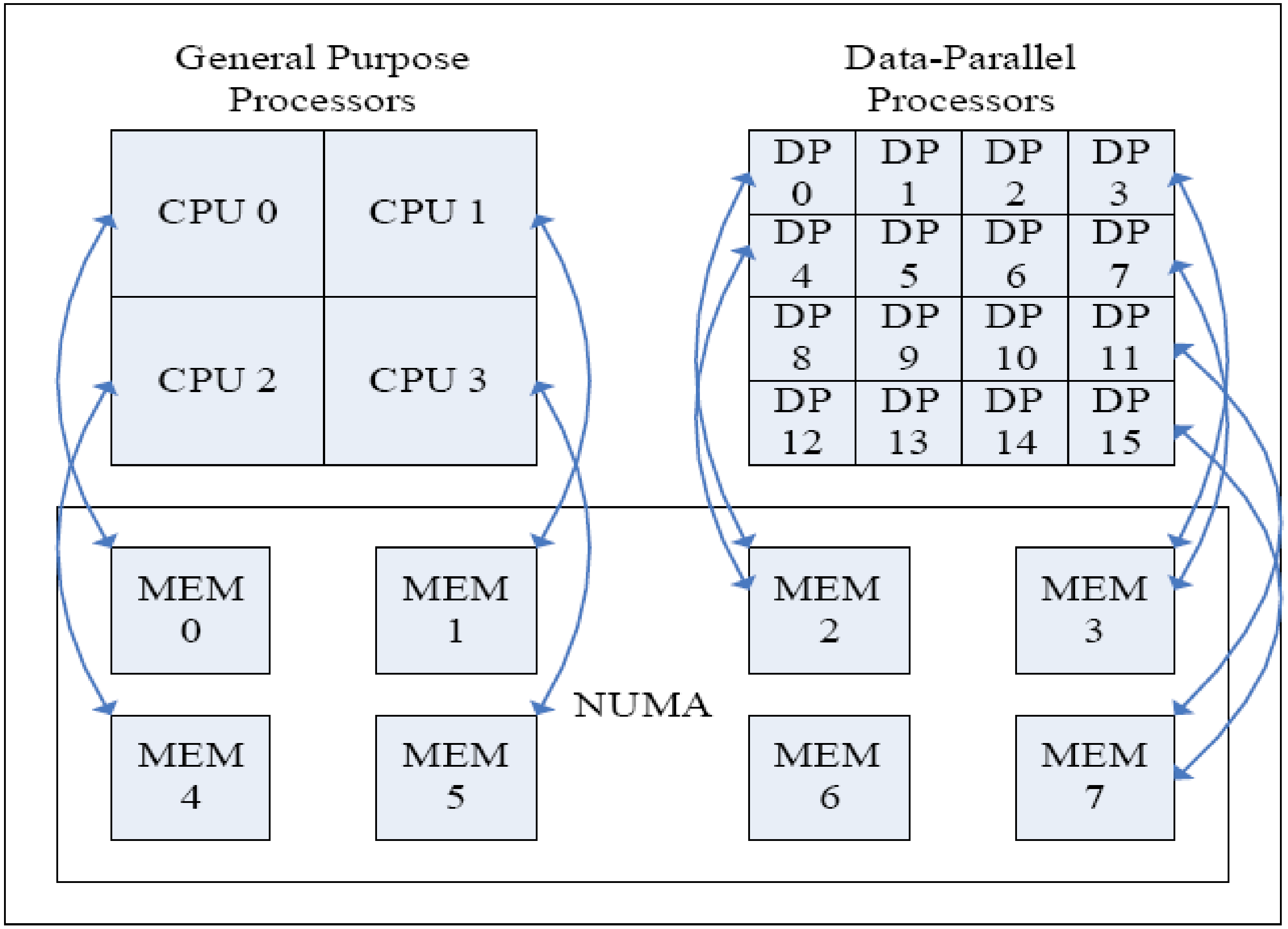


Image courtesy of David E. Keyes, Columbia University

An Idealized Heterogeneous Computer



Consequences for Computational Algorithms

- The vast computational resources available make it crucial to design algorithms that are parallel from the ground up.
- Different sequential tasks are issued to the task-parallel processors
- Inheritably parallel tasks are executed on the data-parallel processors.

NUMA memory layout of HCs

- The growth in processor performance seems to outpace the growth of memory bandwidth
- The data-flow must be organized so that processors access data that is close to it.
- Identify and minimize the dependency between data
- Use extra computations to avoid data synchronization

Precision

- Double-precision numbers requires twice the bandwidth
- Double-precision numbers compute at half the speed of single-precision
- Mixed-precision algorithms will play a key role.

What about existing code?

- Numerical software can often gain dramatic increases in performance by refactoring just a few selected parts

```
for(…){
  for(…){
    computationalKernel();
  }
}
```

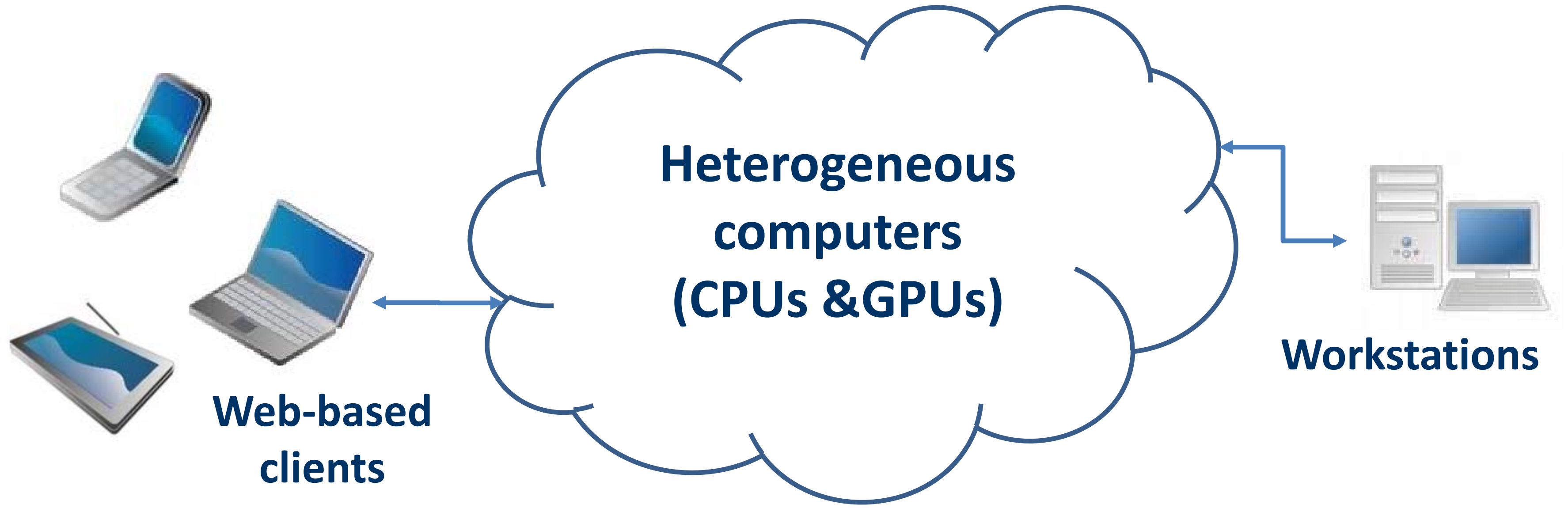
- Such approaches will most likely reach the limits of Amdahl's law: *The theoretical maximum speedup is limited by the relative time needed for the non-parallel part of the algorithm.*

GPU-based Cloud Computing

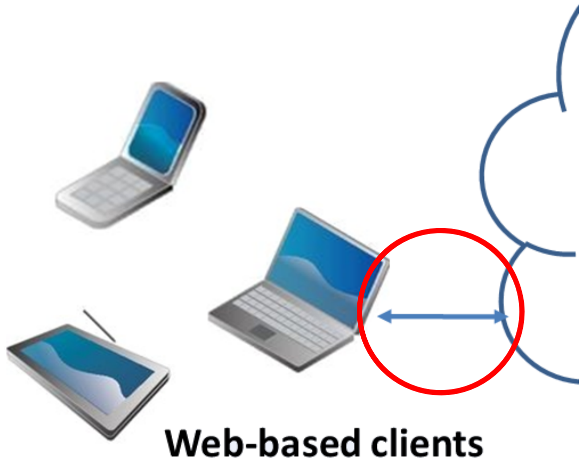
Cloud Computing Characteristics

- Customers do not own the infrastructure
- Consumers pay for only what resources they use
- Dynamic hardware scalability
- Device and location independent
  - Only need an internet connection

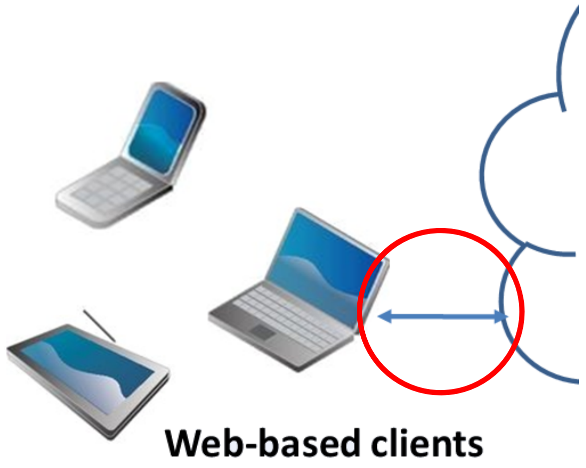
Cloud of Heterogeneous Computers



Web-Based Clients

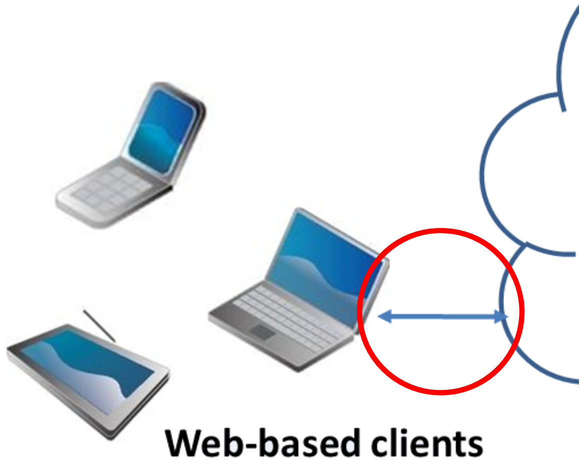


Web-Based Clients



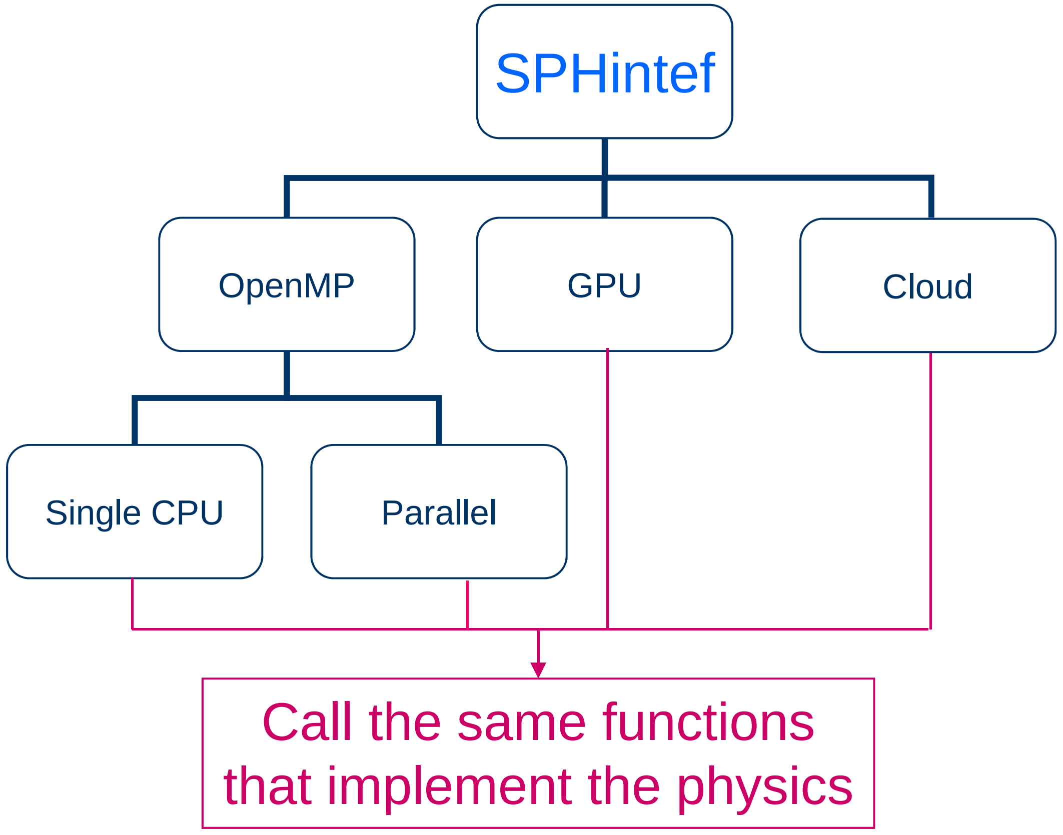
- Processing in the cloud
- Streaming of results to the clients
- User input from the client to the cloud.

Web-Based Clients

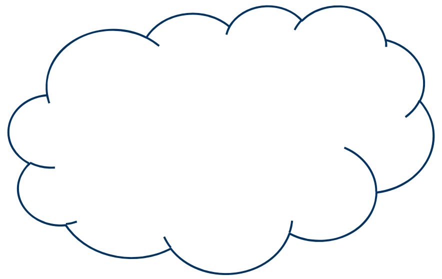


- Processing in the cloud
- Streaming of results to the clients
- User input from the client to the cloud.

Example: Code Framework – SPHintef



GPU – Graphics Processing Unit



Cloud computing

Visualization of results:

Research Challenges: Auto-Tuning in the Cloud

- Future clouds will be based on heterogeneous architectures.

- New auto-tuning algorithms must be developed
  - Traditional auto-tuning techniques do not map to new heterogeneous architectures.

- Very few generic libraries developed
  - BLAS, FFT

- Multipe levels of parallelism:
  - Distributed memory, shared memory
  - MPI, OpenMP, threads, CUDA, etc.

Thanks!

Questions?