



Technology for a better society

J. Seland, C. Dyken, T. R. Hagen,  
A. R. Brodtkorb, J. Hjelmervik, E. Bjønnes

# GPU Computing

USIT Course Week

16th November 2011



9:30 – 10:15 Introduction to GPU Computing

10:15 – 10:30 Break

10:30 – 11:00 CUDA Intermediate Example

11:00 – 11:30 Design, Test and Lifecycle

GPUs are everywhere



Highest performing "chip" in all classes of computers

# What is GPU Computing?

- GPU = Graphics Processing Unit = Video Card
- Delivers **extreme floating point performance**
  - FLOPS / \$
  - FLOPS / Watt
  - FLOPS /Volume
- Massively parallel
- Fine grained parallelism
- TOP500 Supercomputer list:
  - Num 2, 4 and 5 use GPUs



# Hardware characteristics – Q2 2011

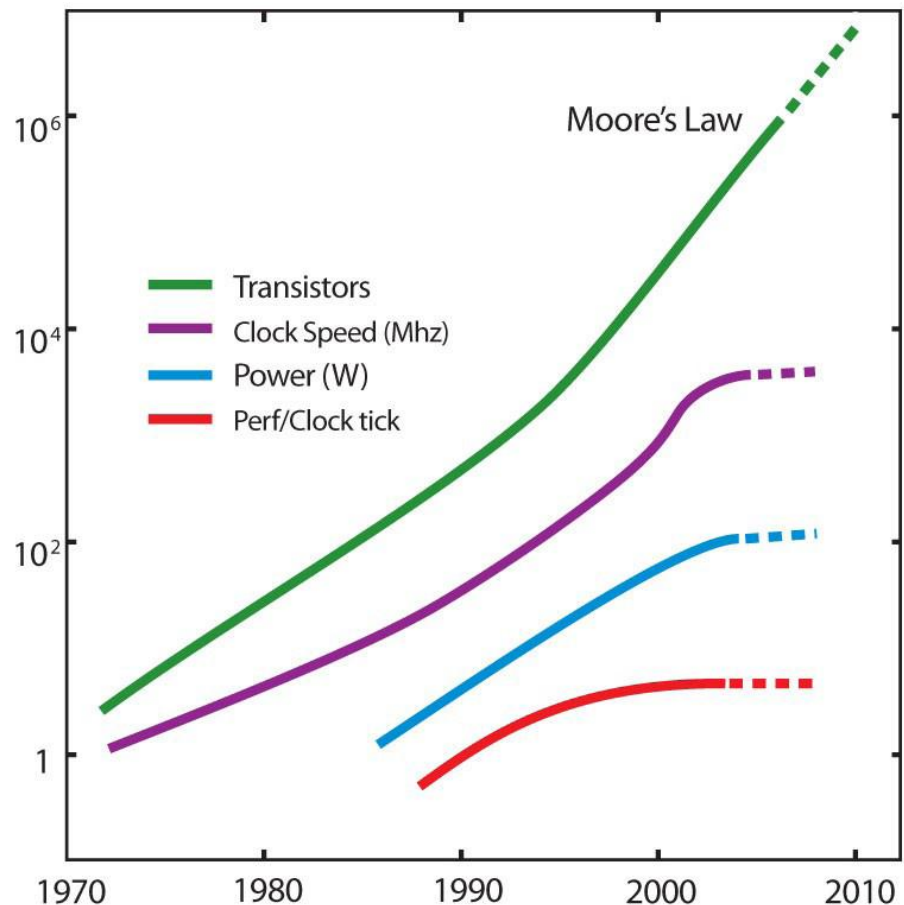
## Workstation Hardware

|                             | I7-2600K<br>Sandy Br, | Fermi GPU<br>Geforce 580 | Xeon<br>X7560 | Fermi GPU<br>Quadro 6000 |
|-----------------------------|-----------------------|--------------------------|---------------|--------------------------|
| Number of cores             | 4                     | 16                       | 8             | 14                       |
| # of float arithmetic units | 32                    | 512                      | 32            | 448                      |
| Clock frequency (GHz)       | 3.4                   | 1.2                      | 2.66          | 1.1                      |
| Single precision gigaflops  | <b>217</b>            | <b>1581</b>              | <b>144</b>    | <b>985</b>               |
| Double:single performance   | 1:2                   | <b>1:8</b>               | 1:2           | 1:2                      |
| Gigaflops per watt          | 2.3                   | 6.5                      | 1.1           | 6.5                      |
| Gigaflops per \$            | 0.68                  | 3.2                      | 0.048         | 0.20                     |
| Memory bandwidth (GiB/s)    | 21.3                  | 192.4                    | 26            | 160                      |

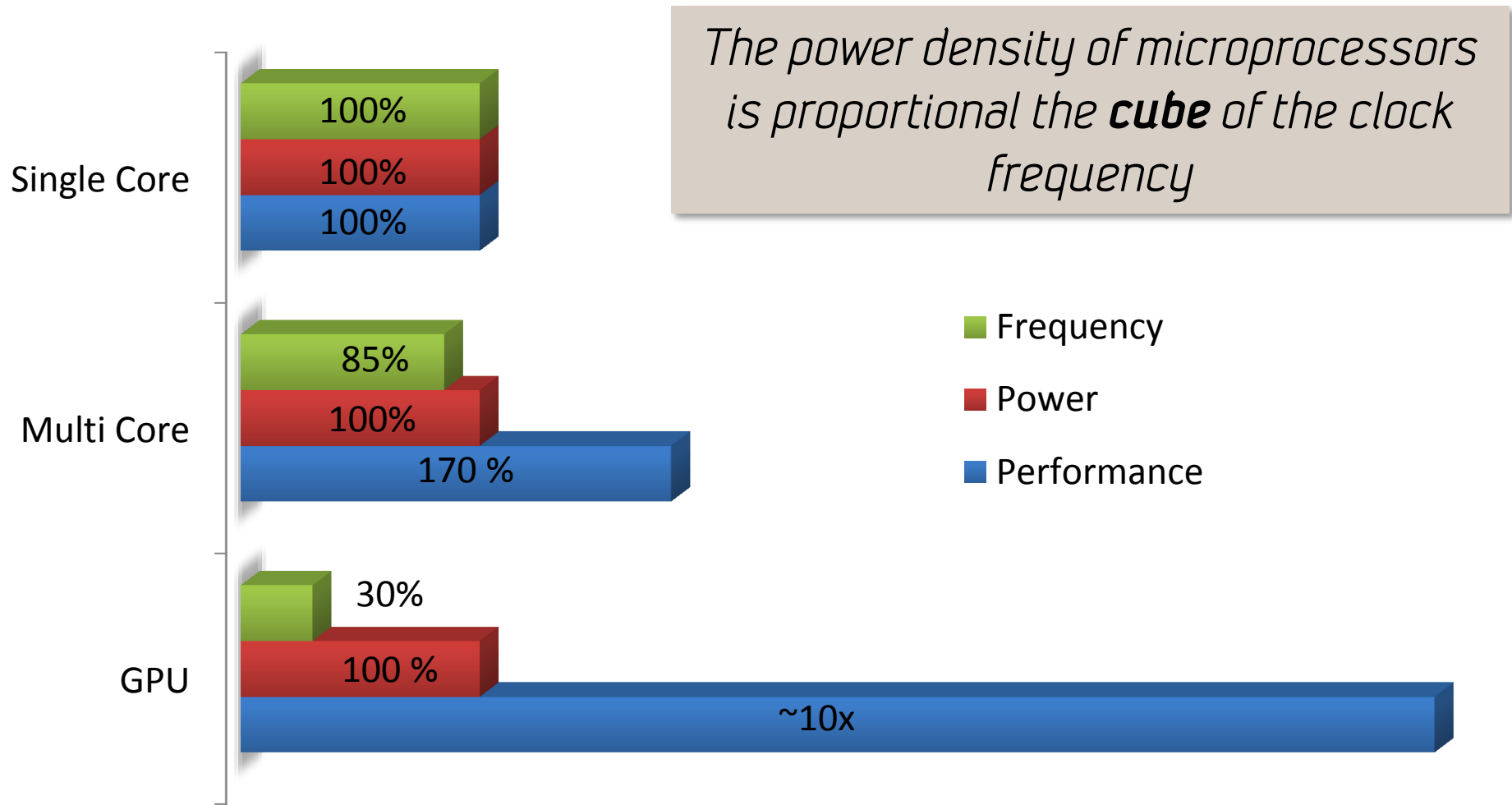
# What happened 2000 – 2010?

Increasing frequency hits several walls:

- Memory
  - Expensive to build fast memory
  - Remedy: Caches
- Instruction Level Parallelism
  - Complex to identify
- Power Density
  - Relative to frequency **cubed**



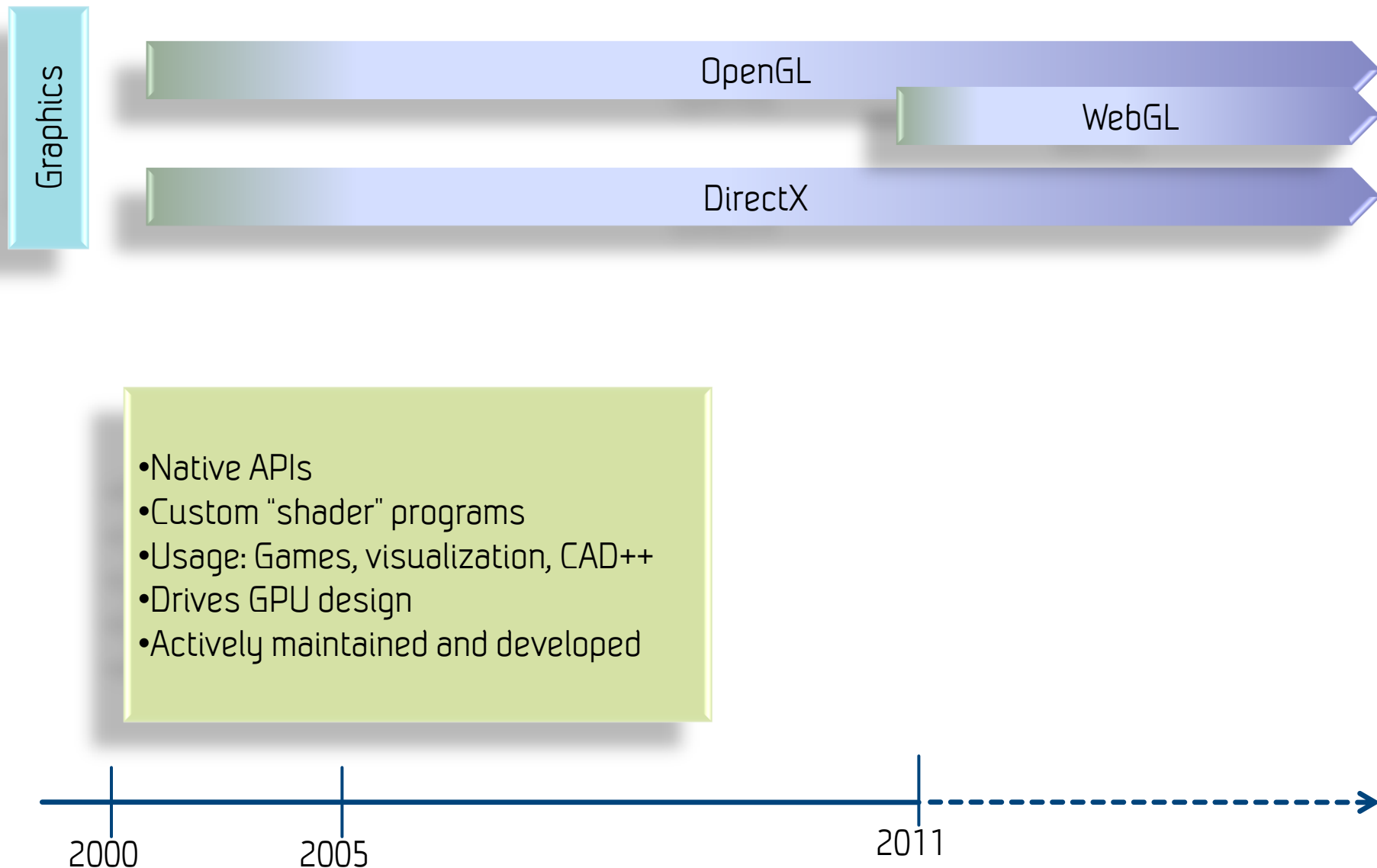
## How Parallelism Can Help



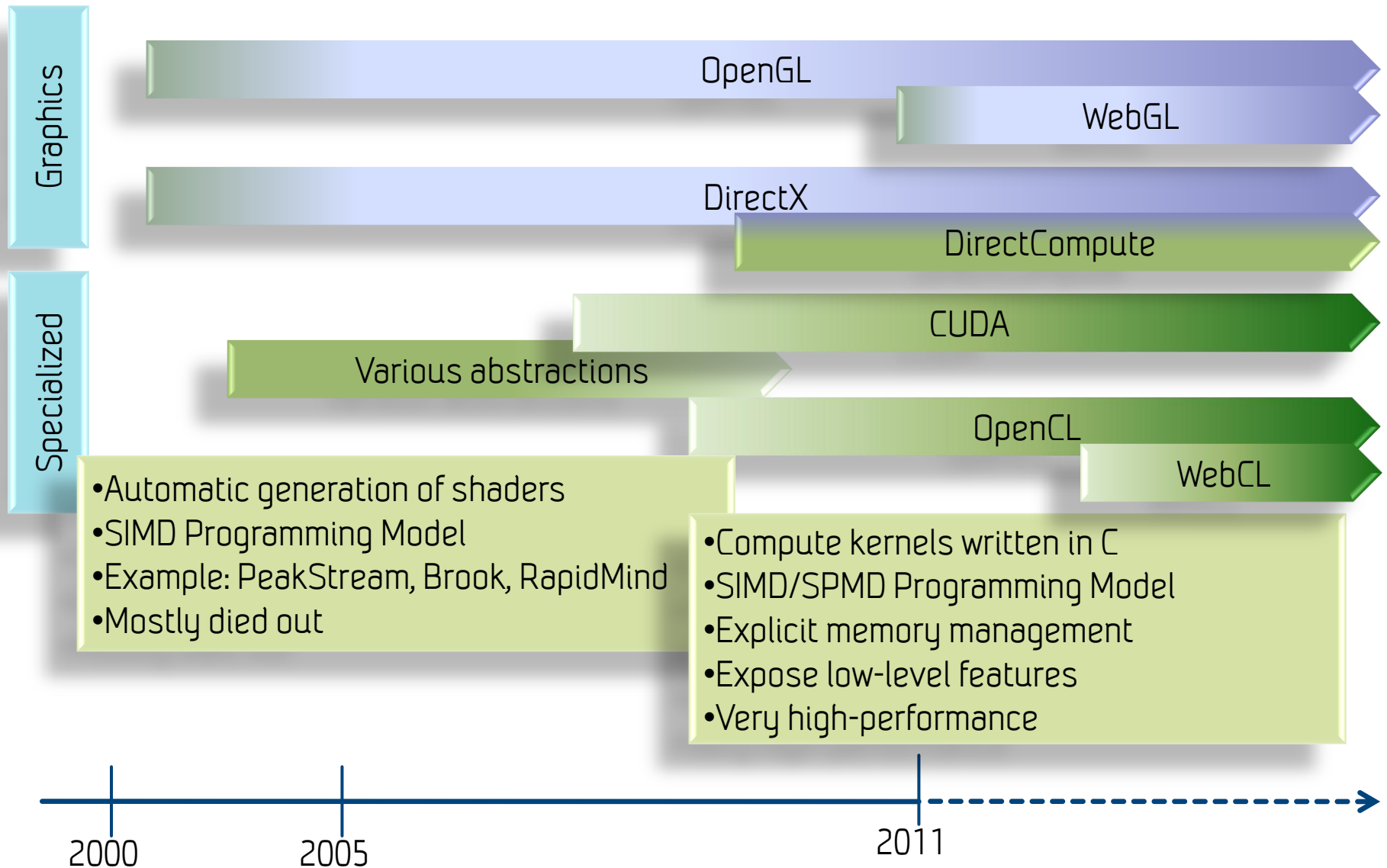
# GPU Programming Models

- GPU Hardware
  - Parallel
  - Complex
  - Changing
  - Proprietary
- Device driver act as operating system
  - Memory management
  - Task scheduling
  - Just-in-time compilation
- Various abstractions hide details
  - Remarkably successful!

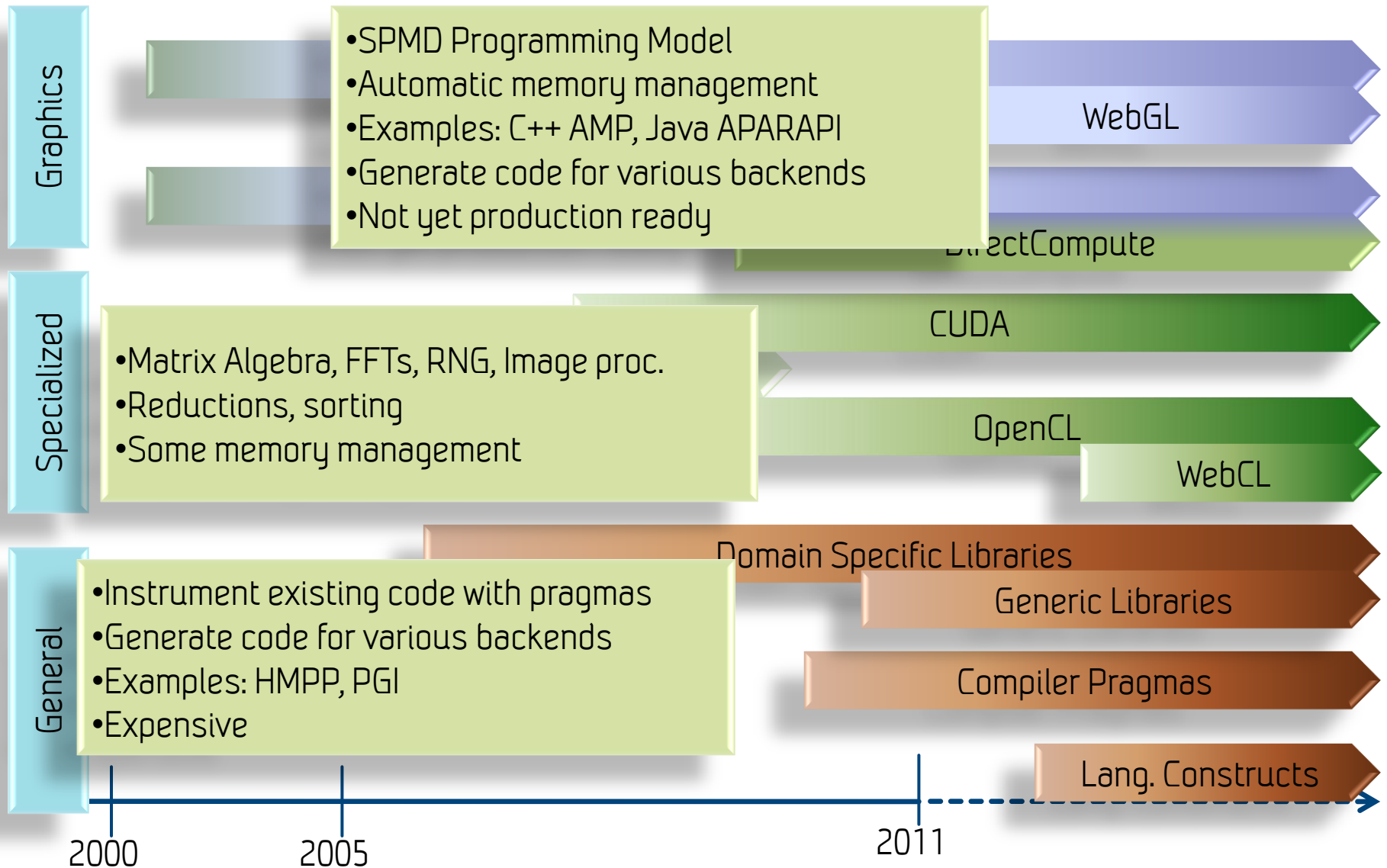
# GPU Programming Models



# GPU Programming Models



# GPU Programming Models



# OpenCL vs CUDA

- Two APIs for directly programming GPUs
- Expose the same programming model (SPMD)
- OpenCL has a public standard



|                     | OpenCL                       | Nvidia CUDA                    |
|---------------------|------------------------------|--------------------------------|
| Owner               | Khronos Group                | Nvidia                         |
| Target Platform     | GPUs, CPUs, cell phones      | Nvidia GPUs                    |
| Programming Model   | SPMD                         | SPMD                           |
| Language            | C                            | C/C++ (templates, virt. funcs) |
| Low-Level HW Access | Proprietary extensions       | Full                           |
| HW Fragmentation    | Much                         | Some                           |
| Tools               | Some                         | Mature                         |
| Vendor support      | Apple, AMD, Nvidia , Intel++ | Nvidia                         |

# Approaches to GPU Programming

| Approach                                                  | Language                                                         | Description                                                     | Domain                                          |
|-----------------------------------------------------------|------------------------------------------------------------------|-----------------------------------------------------------------|-------------------------------------------------|
| Graphics API<br>•OpenGL (WebGL)<br>•DirectX               | C(++), Fortran, .NET,<br>Java, Python, Perl,<br>Ruby, Javascript | What GPUs were<br>designed for                                  | Graphics (Games,<br>Visualization, CAD<br>etc.) |
| Matlab/Mathematica                                        | Matlab/Mathematica                                               | "Semi"-automatic                                                | Scientific                                      |
| OpenMP like pragmas<br>•PGI Accelerator<br>•HMPP<br>•Cray | C/C++ / Fortran                                                  | Easy porting of legacy<br>applications                          | Scientific applications                         |
| GPU libraries                                             | C/C++<br>(call from anything)                                    | Easy to integrate into<br>existing apps, if<br>algorithm exists | Scientific,<br>encoding/decoding                |
| Dedicated languages<br>•CUDA<br>•OpenCL                   | C(++ ) dialects<br>(call from anything)                          | Expose GPU features<br>Hand tuned algs.<br>Manual mem. alloc.   | Scientific                                      |

## Some available libraries

|                               |                                      |
|-------------------------------|--------------------------------------|
| CUFFT                         | Fast Fourier Transform               |
| CUBLAS                        | Dense Linear Algebra                 |
| CULA                          | LAPACK interface                     |
| CUSPARSE                      | Sparse Linear Algebra                |
| CUSP                          | Linear Algebra<br>Graph Computations |
| CURAND                        | Random Number Generation             |
| NPP – Nvidia Perf. Primitives | Image and Signal Processing          |
| CUDA Video Decoder/Encoder    | H.264/MPEG-2 video coding            |
| THRUST                        | STL like algorithms                  |

These libraries have various licenses

# GPU Clusters

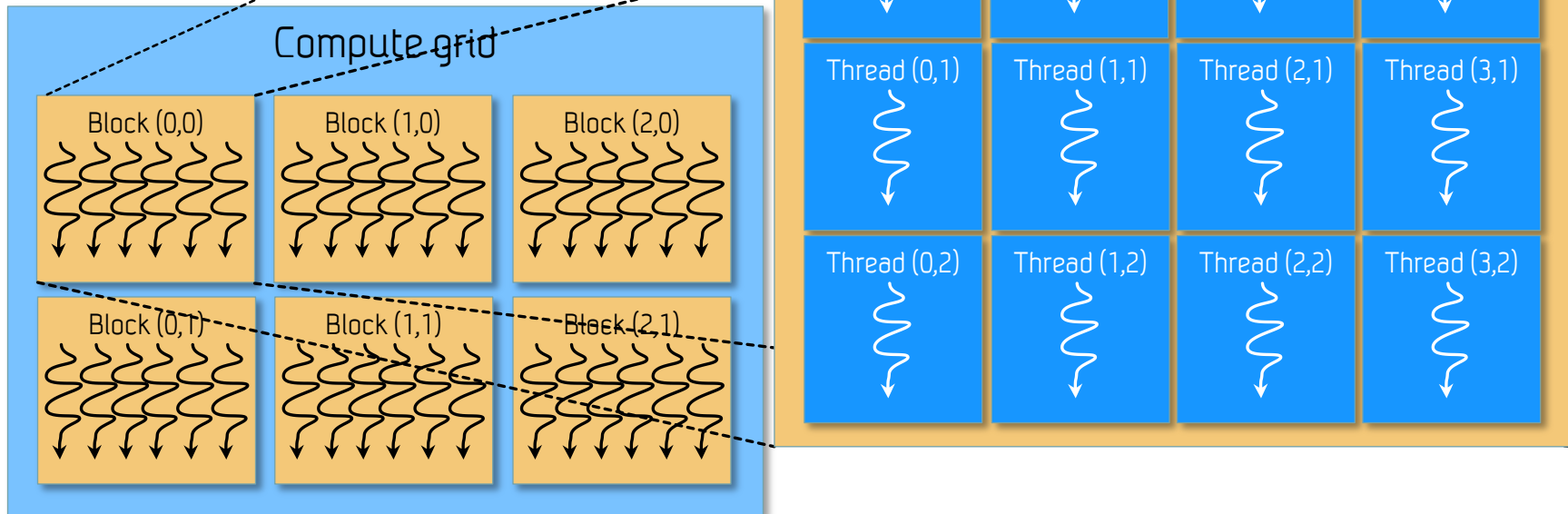
- Each node has:
  - 1-4 GPUs
  - 1-4 multi-core CPUs
- MPI-style parallelism between nodes
- MPI-style parallelism between GPUs
- MPI or thread-parallelism between CPUs



# SPMD Programming Model

- Host code, runs on CPU
  - Memory allocation
  - Memory transfer
  - Scheduling of tasks
  - Dependencies
- Device code, runs on GPU
  - **Kernel functions**
    - **Invoked over compute grids**
    - **Compute grid can be much larger than #cores**
  - Written in C/C++-like languages (CUDA/OpenCL)
  - Separate compiler

# GPU compute grids

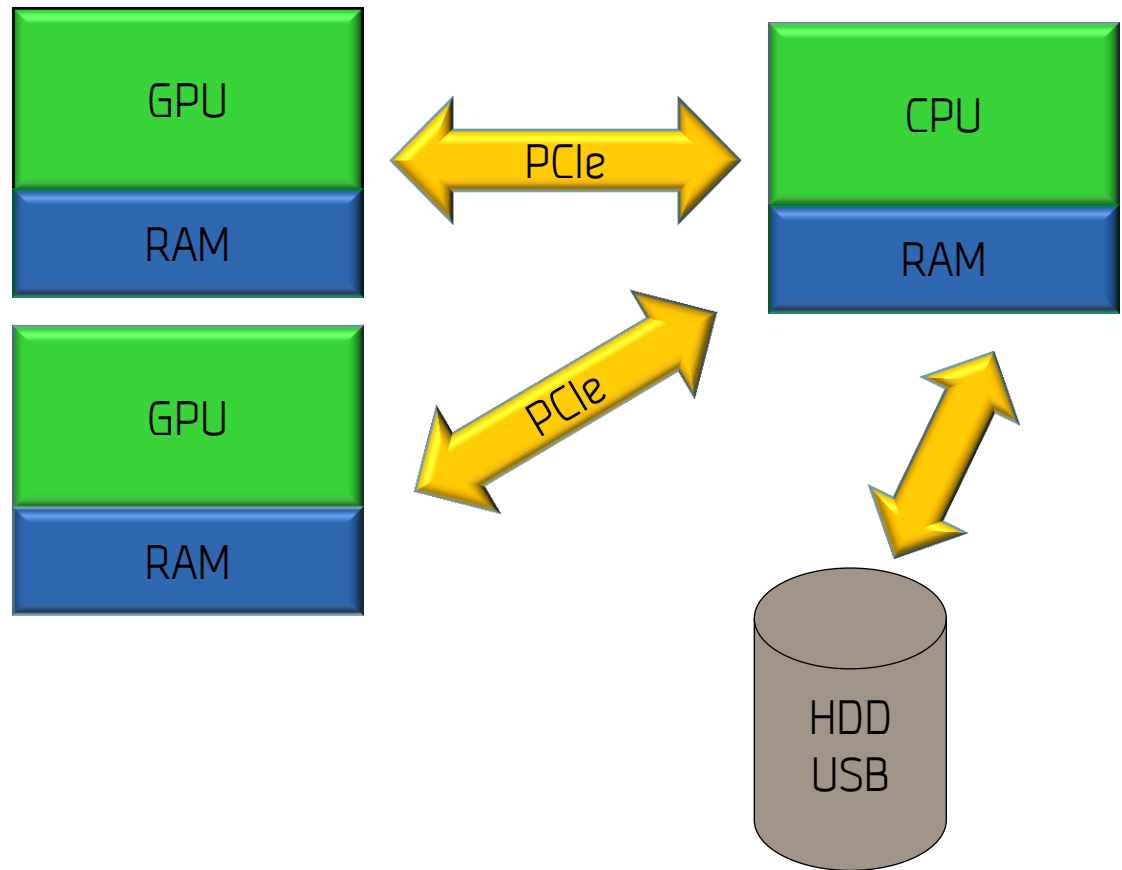


- Execution invoked by CPU over a **compute grid**
  - Compute grid subdivided into a set of **blocks**
    - Blocks contains a set of **threads**, which can access block-level **shared memory**
- All threads in the compute grid run **the same program**
  - But with **individual data** and **individual code flow**

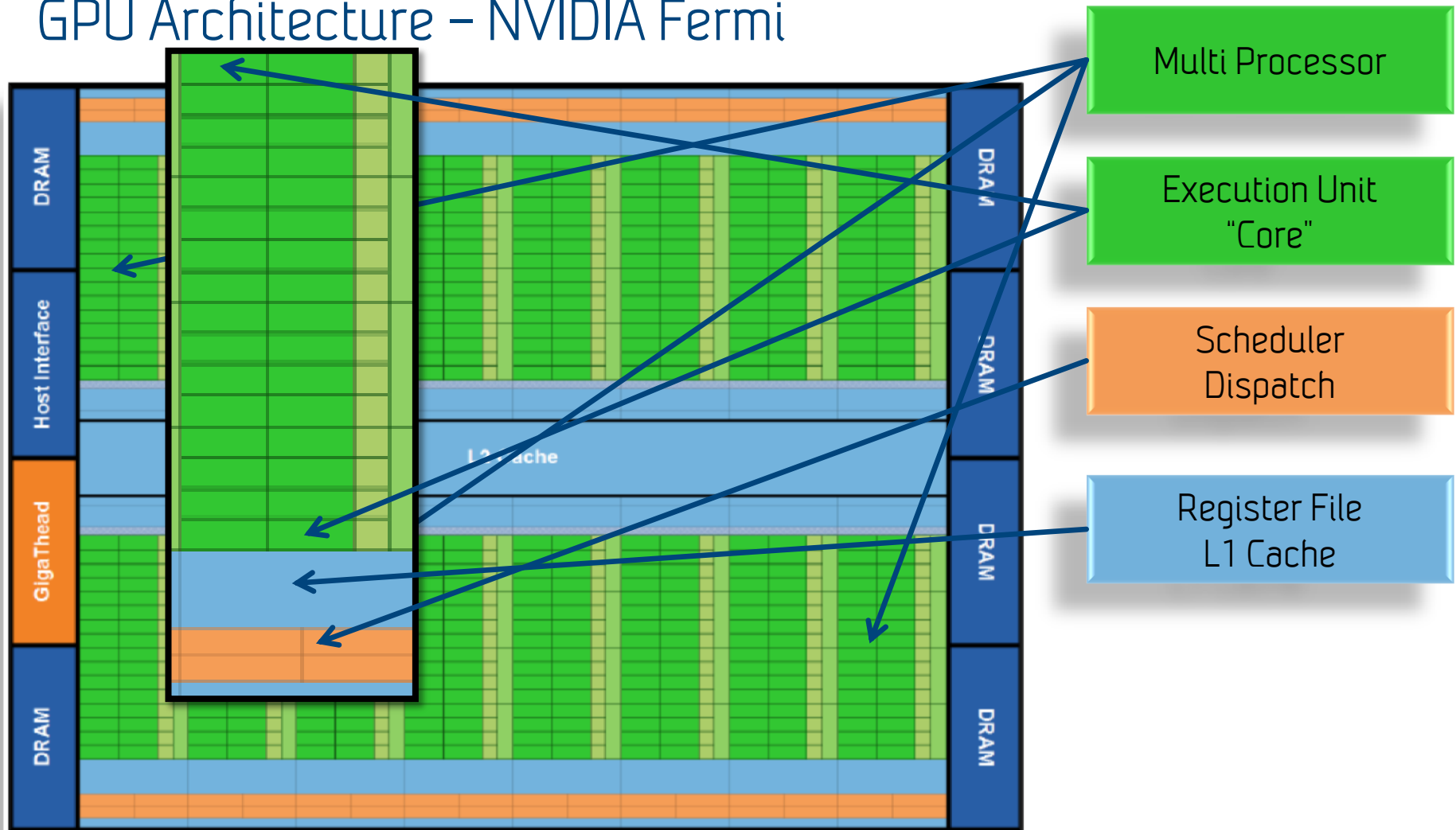
# GPU Architecture

# System overview

- GPUs are on the PCIe bus
- GPUs have their own memory
- Some recent chips have embedded GPUs
- Multi-GPU system are common

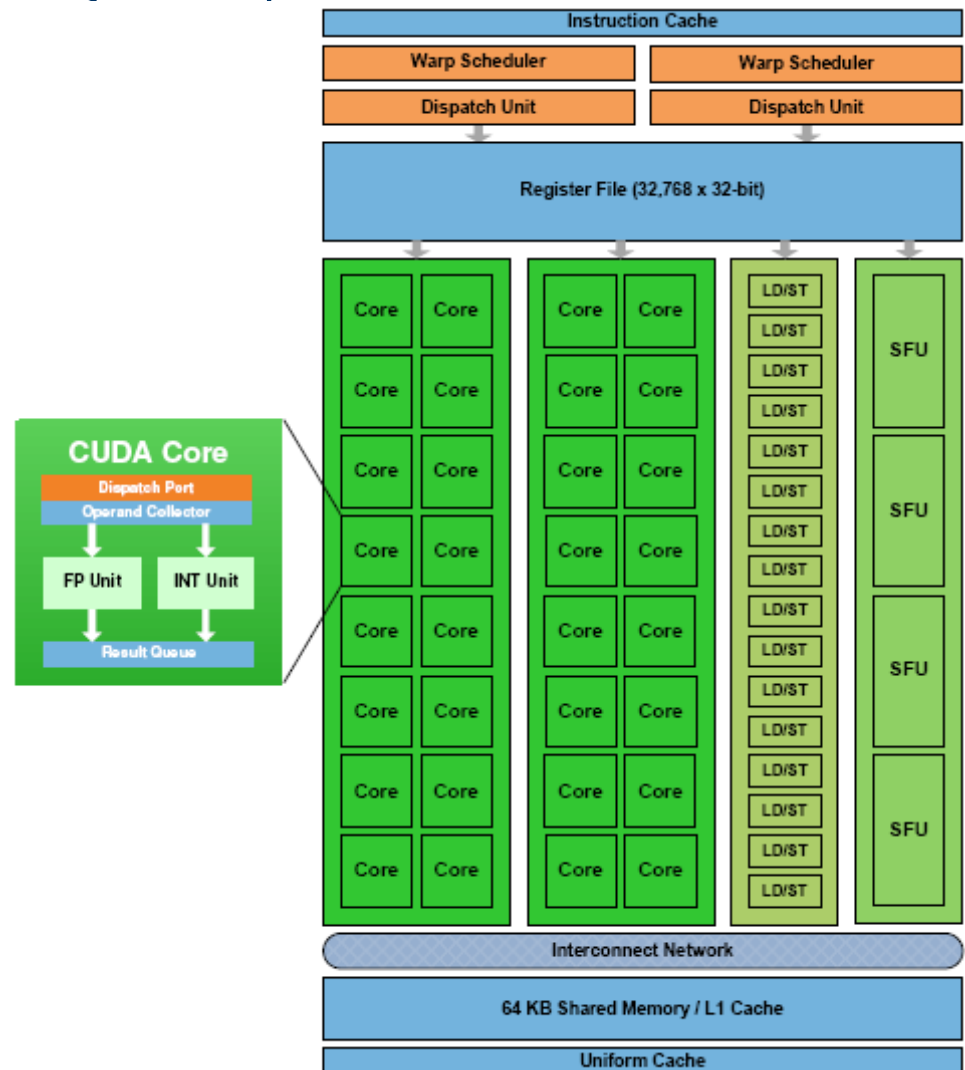


# GPU Architecture – NVIDIA Fermi

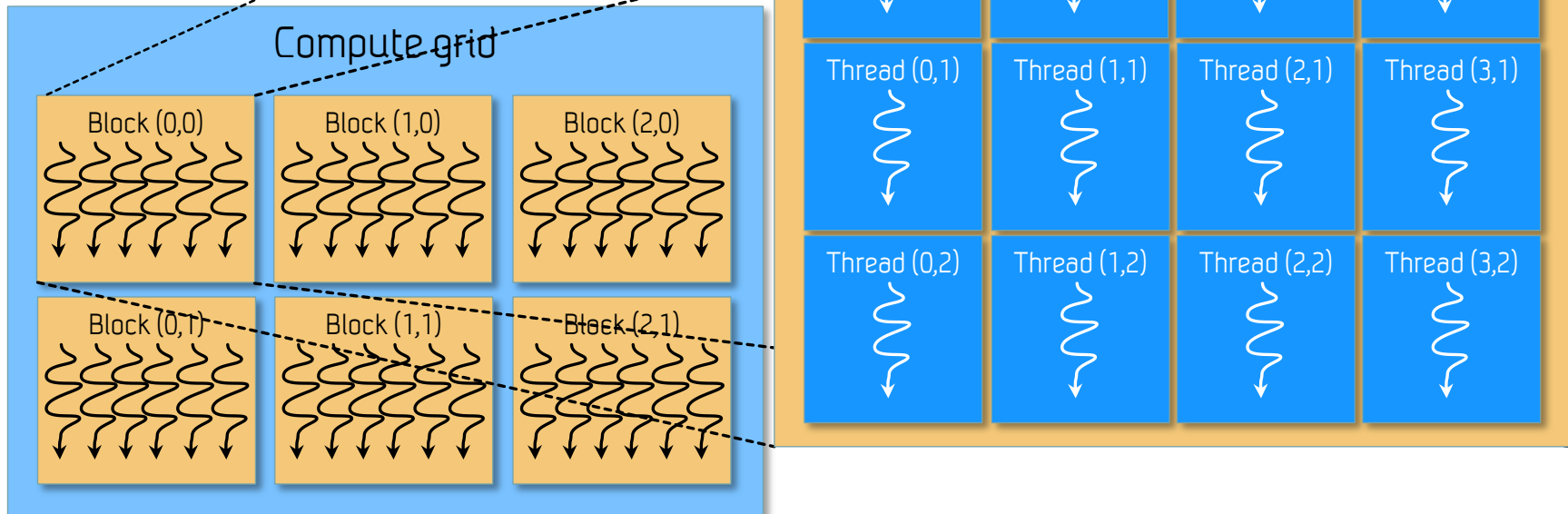


# Fermi Architecture – Streaming Multiprocessor

- 32 cores per SM
- 64 KB shared memory and L1 cache
- Special Function Unit
- Double precision at half speed
- Concurrent kernel execution
- ECC Support



# Recap – Compute Grids



- Execution invoked by CPU over a **compute grid**
  - Compute grid subdivided into a set of **blocks**
    - Blocks contains a set of **threads**, which can access block-level **shared memory**
- All threads in the compute grid run **the same program**
  - But with **individual data** and **individual code flow**

# Challenges in CUDA/OpenCL Programming

- Hard to learn
  - In our experience: 1/2 year to master if motivated
  - Do you really care that much about performance after all?
- Hardware fragmentation
- Makes the build process more complex
- Driver/compiler version issues
- Low level of reusability of code
- Many different optimization strategies possible
  - Memory access in particular

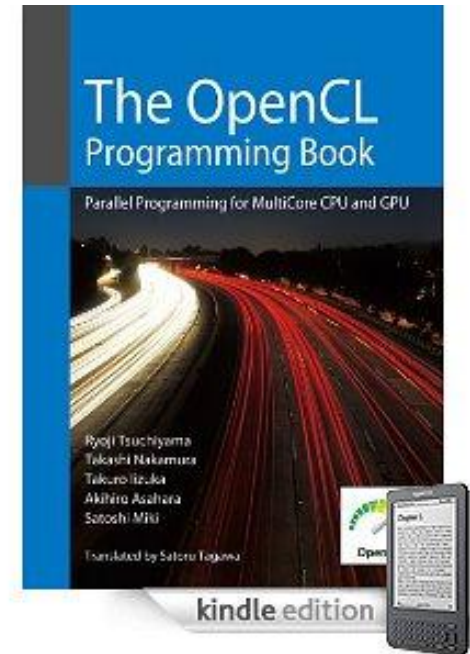
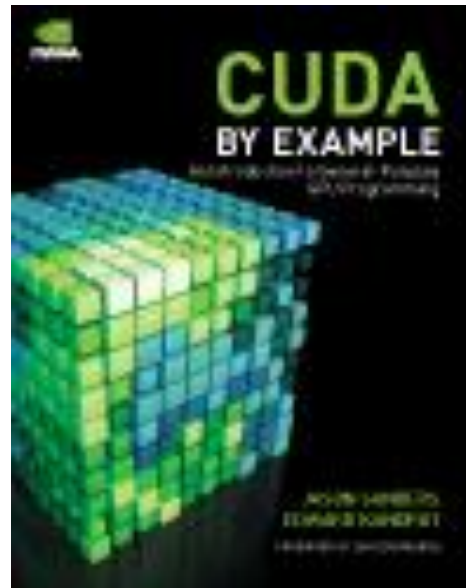
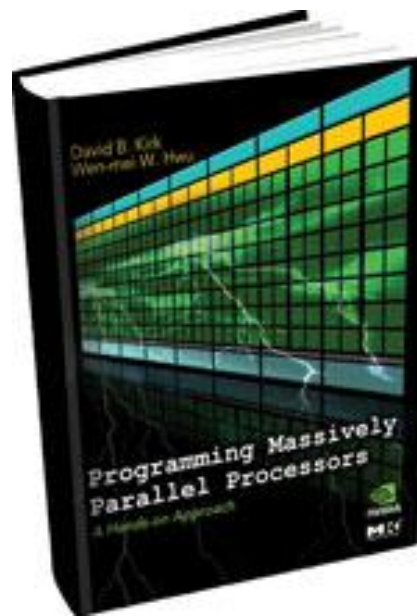
# Conclusion

# Conclusion

- GPU computing is here now
  - (So is multi-core computing)
- Widely deployed on supercomputers (2<sup>nd</sup>, 4<sup>th</sup> and 5<sup>th</sup> on TOP500)
- Easy to get started
  - Libraries can be called from existing application
- Difficult to reach peak performance
  - Requires intimate HW knowledge
- "Easy" to get some speedup
- Hard to reach optimum performance

# Overview of resources

- [nvidia.com/cuda](https://nvidia.com/cuda)
  - Programming guide
  - Tutorials
  - Forums
- [khronos.org/opencl/](https://khronos.org/opencl/)
- [gpgpu.org](https://gpgpu.org)
  - Links to papers/libraries



Questions?