VSCSE summer school - short course

Introduction to CUDA

Lecture 1

Introduction to Many-Core Processors

Joshua A. Anderson

Modern video games demand copious processing power.

2003

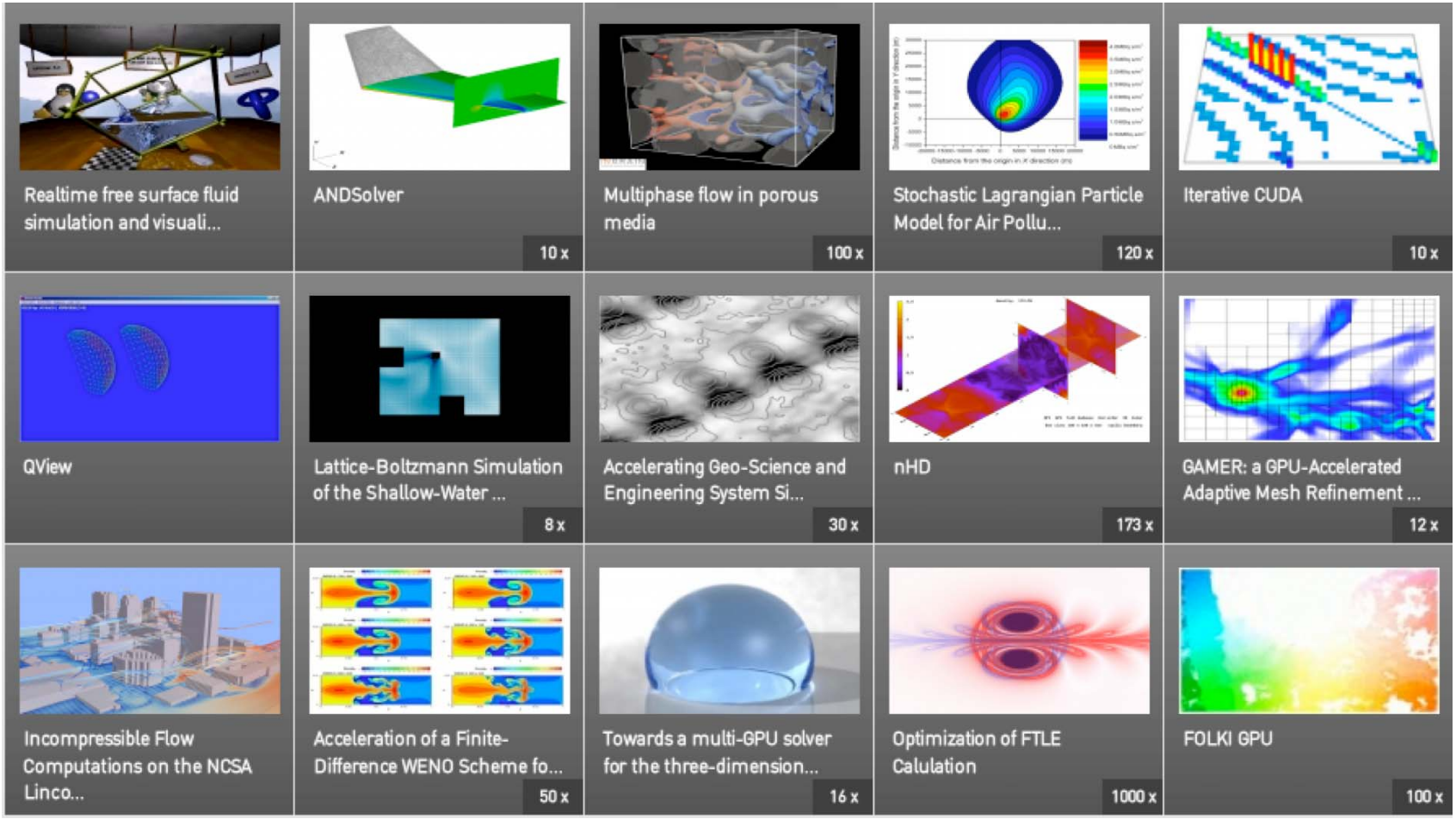
2006

2009

- ***Millions*** of pixels
- ***Thousands*** of calculations per pixel
- One ***hundred*** frames per second
- 100's of GFLOPS are needed

Applications using CUDA

http://www.nvidia.com/object/cuda_apps_flash_new.html#

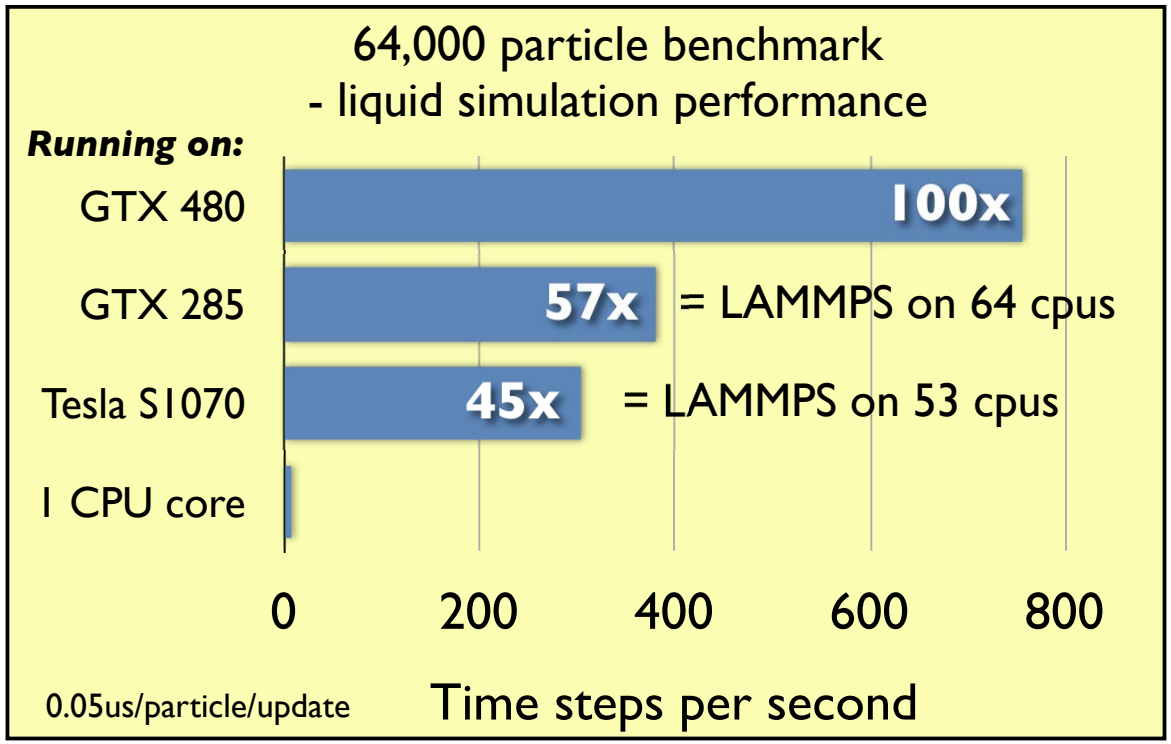


Recorded for the Virtual School of Computational Science and Engineering

HOOMD blue

Open source:
http://codeblue.umich.edu/hoomd-blue

General purpose Many-particle Dynamics
fully implemented on GPU hardware



32 dual quad core **CPU nodes** w/ fast Infiniband

$140,000

One desktop w/
4 GPUs

~

$5000

images from http://www.dell.com

Recorded for the Virtual School of Computational Science and Engineering

Desktop GPU: Cost per performance

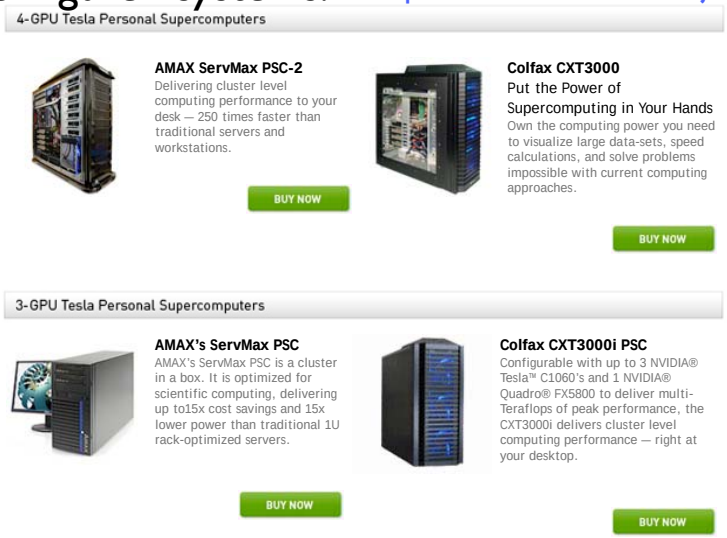
	GTX 285	Tesla C1060	GTX 480	Tesla C20X0
Processor elements	240	240	480	448
Peak compute (single)	1.062 TFLOPS	933 GFLOPS	1.3 TFLOPS	1.03 TFLOPS
Peak compute (double)	88.5 GFLOPs	77 GFLOPS	158 GFLOPS	515 GFLOPS
Memory Bandwidth	159 GB/s	102 GB/s	177 GB/s	144 GB/s
Memory size	1 GB	4 GB	1.5 GB	3 - 6 GB
Cost	**$330**	**$1300**	**$500**	**$2500**

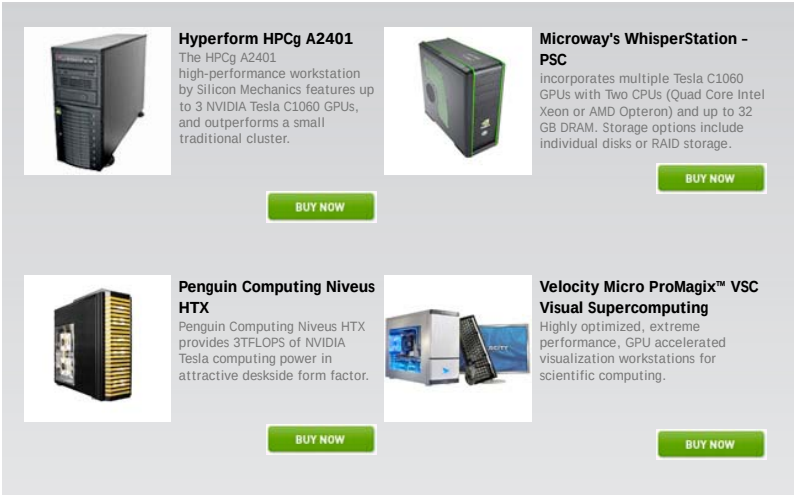
2008-2009 2010

Workstation and datacenter GPUs

Build your own: http://www.nvidia.com/object/tesla_build_your_own.html

Pre-configured systems: http://www.nvidia.com/object/tesla_supercomputer_wtb.html





AC http://iacat.uiuc.edu/resources/cluster/

Tesla S1070

= 4x Tesla C1060 + power supply + fans

Large GPU clusters

- Lincoln - at NCSA
  - 1536 CPU cores
  - 384 Tesla GPUs
  - A TeraGrid resource

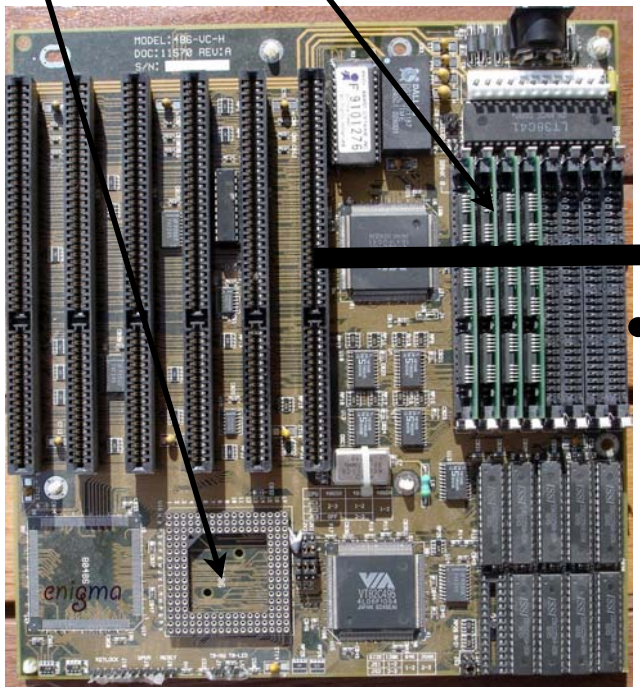
http://www.ncsa.illinois.edu/UserInfo/Resources/Hardware/Intel64TeslaCluster/

- TSUBAME - at Tokyo Tech
  - First Tesla equipped cluster on the Top500

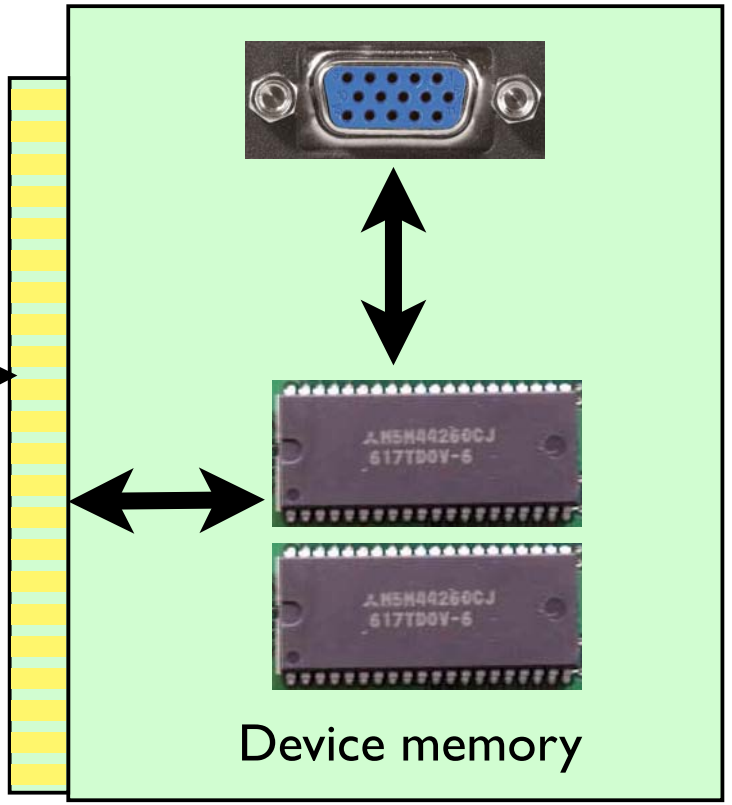
- Many more
  - http://www.nvidia.com/object/cuda_clusters.html

GPU prehistory (1981 - 1997)

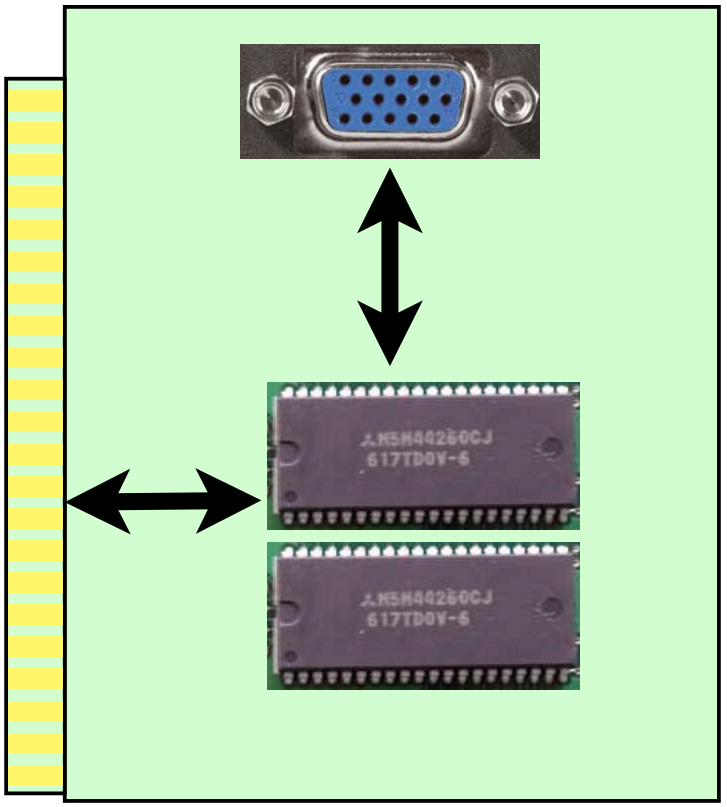
- **CPU** - central processing unit
- **Memory** - Semi-permanent storage for data

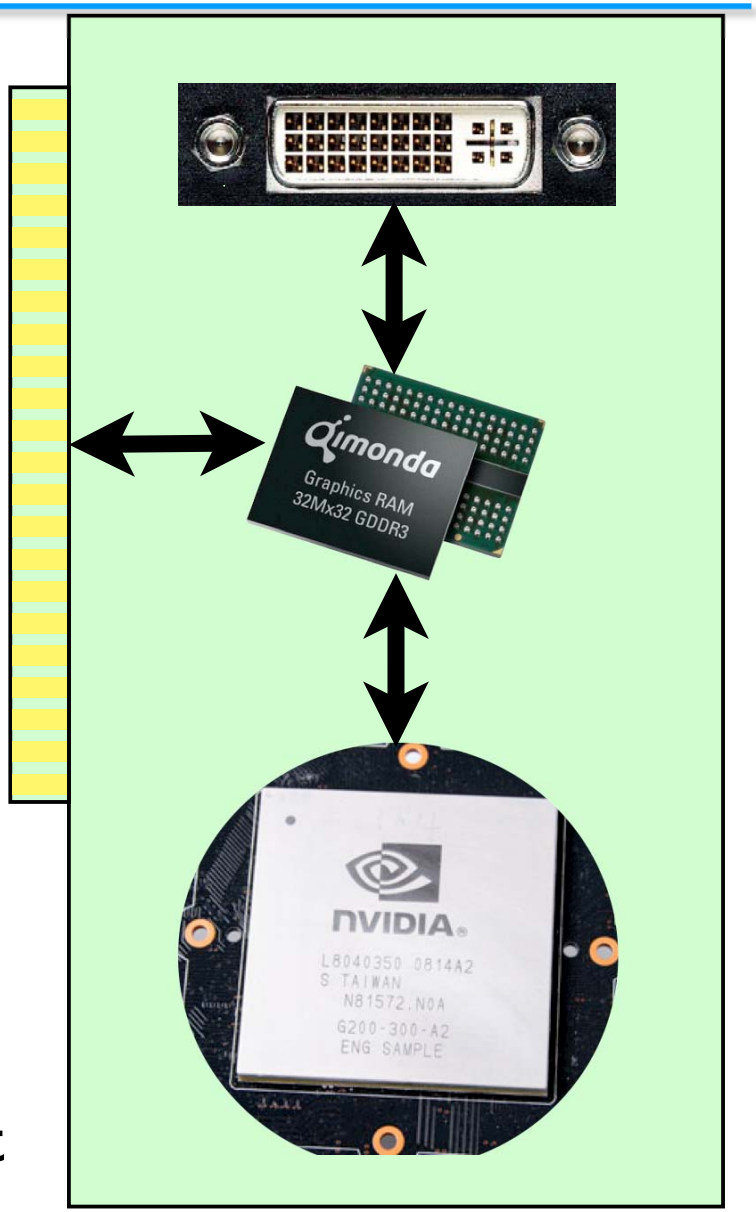


- **PCIe** - Point to point communications link. Used to connect the host to the device.



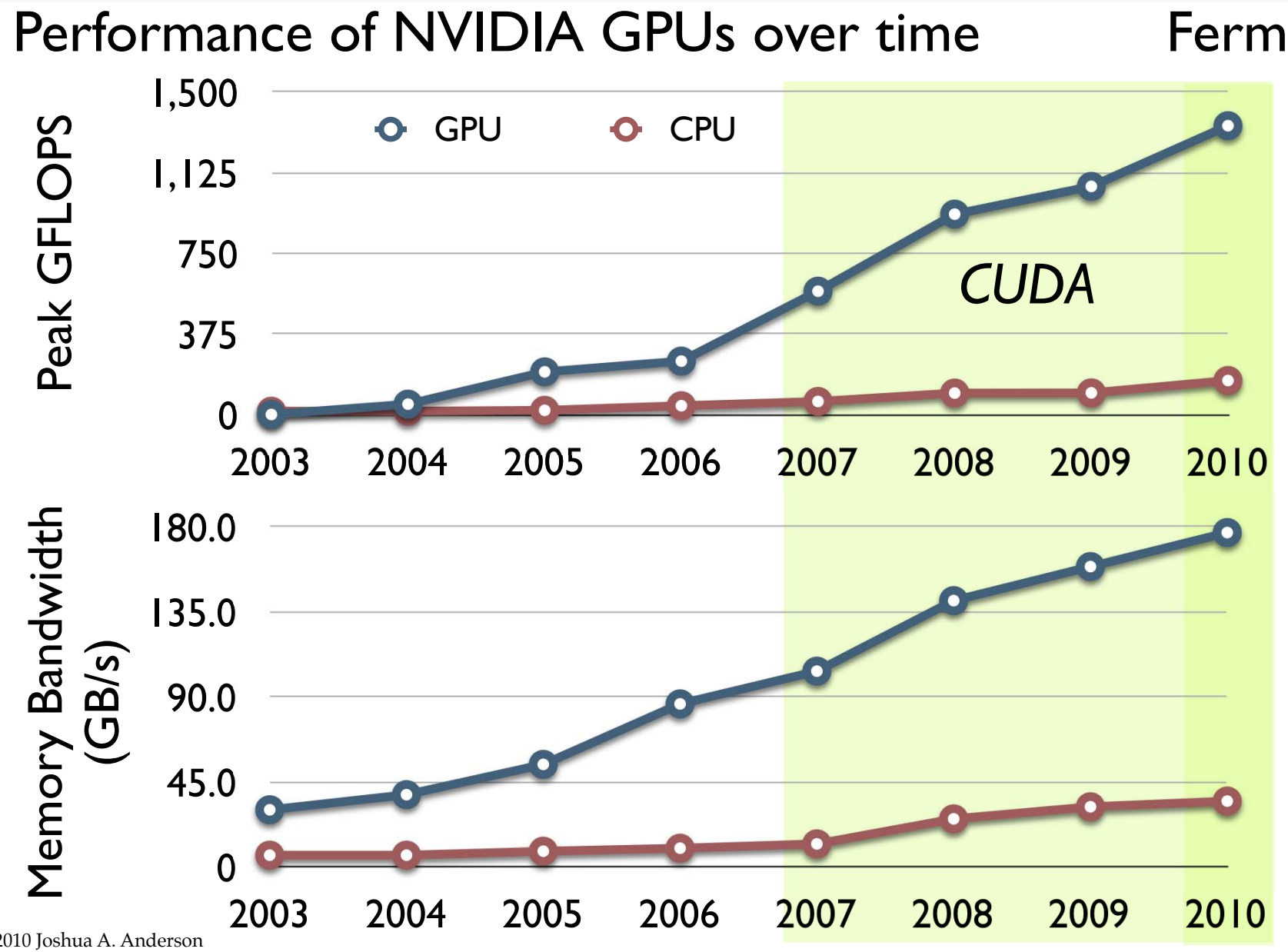
1997: 3DFX - the GPU is born





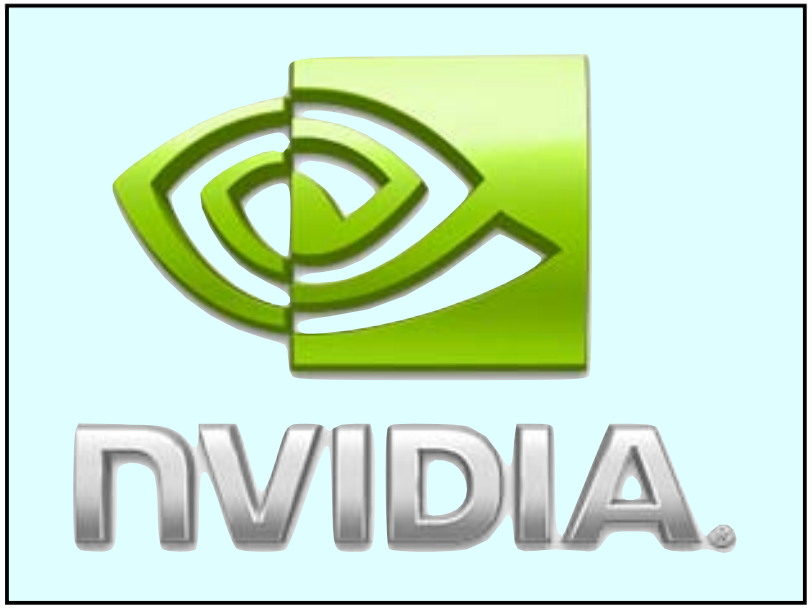
- **GPU** - graphics processing unit

GPU performance in recent history

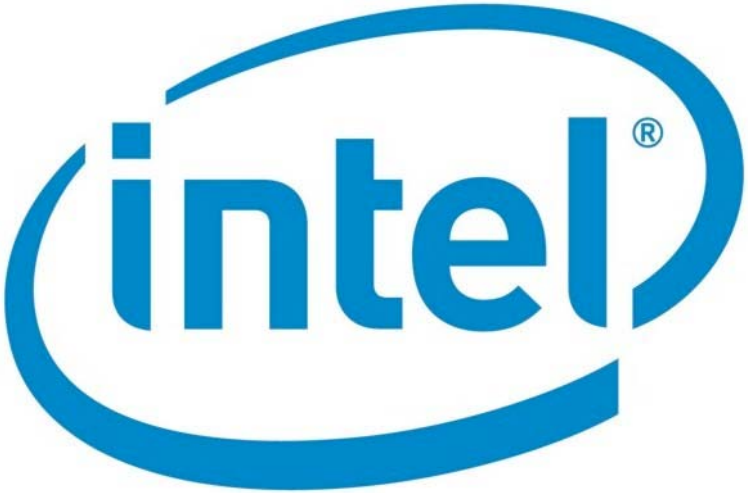


Recorded for the Virtual School of Computational Science and Engineering

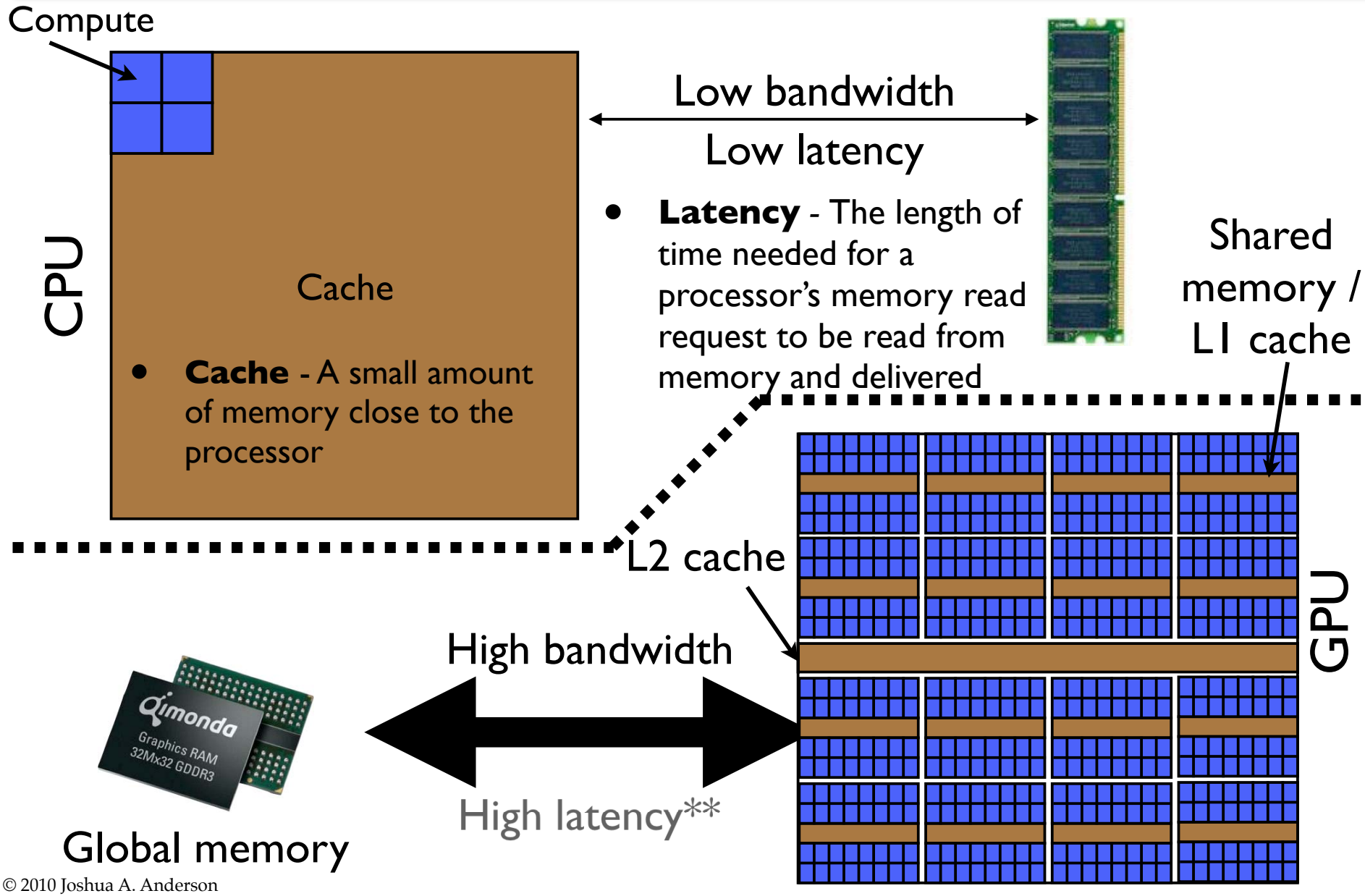
The industry competitors today



- **74.9 million** GPUs sold in Q1 2009
- Three big players
- Healthy competition and large demand => GPUs are **inexpensive** and **widely supported**

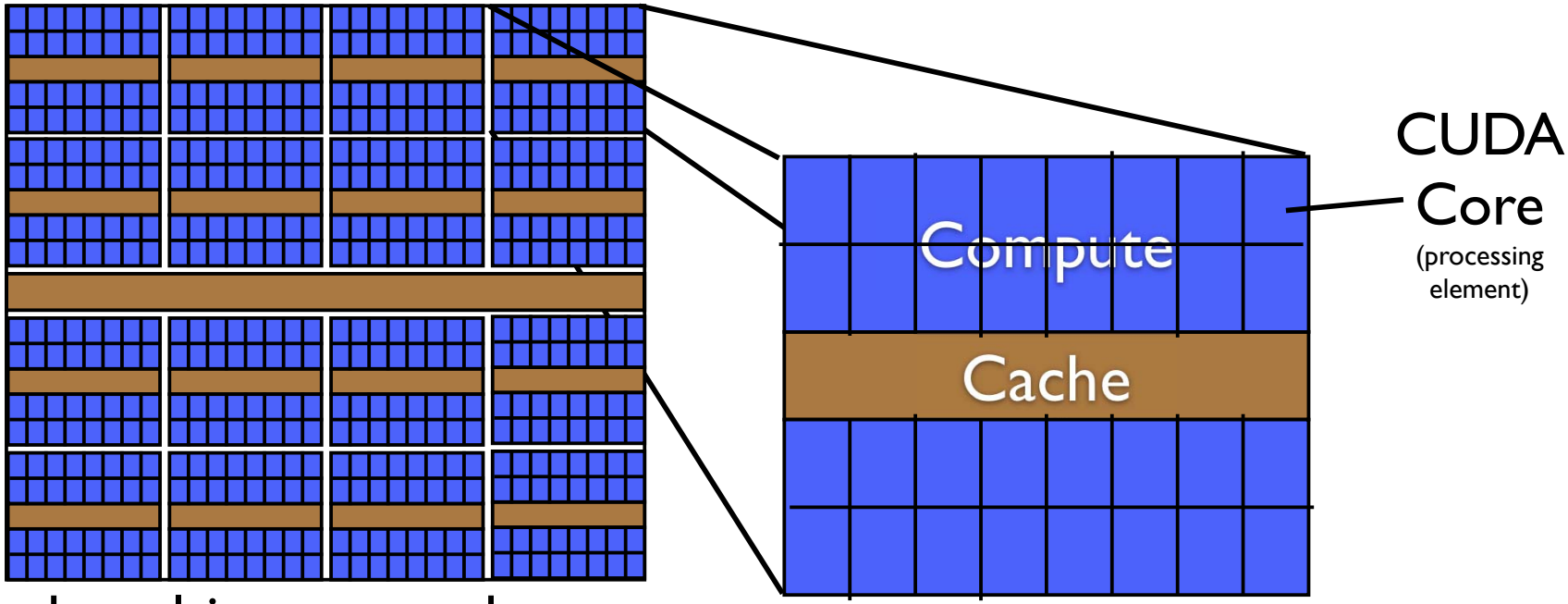


GPU architecture comparison



- **Cache** - A small amount of memory close to the processor
- **Latency** - The length of time needed for a processor's memory read request to be read from memory and delivered

Multiprocessor capabilities



Each multiprocessor has:

CUDA cores	32
Total cache	64 kb
Registers	32,768
Floating point	IEEE-754 double
Number of active threads	1536

- **thread** - An independent sequence of operations (i.e. add, multiply, load, store, compare, branch...)
- **Register** - Temporary storage for the inputs and outputs of compute instructions.

Wait, did you say *1536 active threads!?*

- Instructions are processed in ***warps*** of 32 threads
- any pattern of branches is handled by the **hardware**
- A multiprocessor is a huge **latency hiding** engine
- 16,384 registers => **no overhead** swapping contexts
- **SIMT**, not SIMD

Example instruction stream

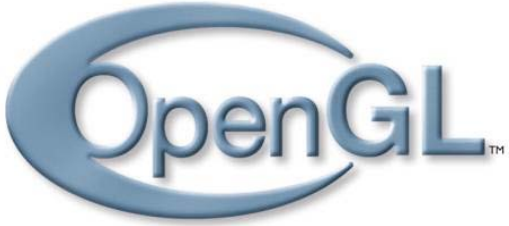
• • •
• • •

warp 1	*memory load - sleeps*
warp 4	multiply
warp 2	add
warp 3	*memory load - sleeps*
warp 2	subtract
warp 1	*wakes up* - multiply
warp 4	subtract
warp 3	*wakes up* - subtract
warp 1	add

• • •
• • •

GPU programming environments

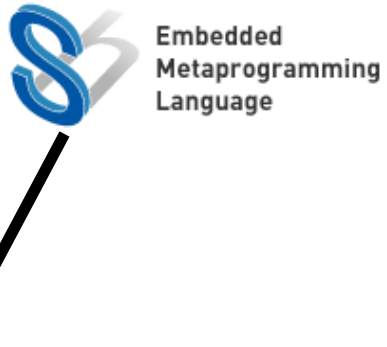
Graphics "only"



General purpose compute

OpenCL

Stream programming



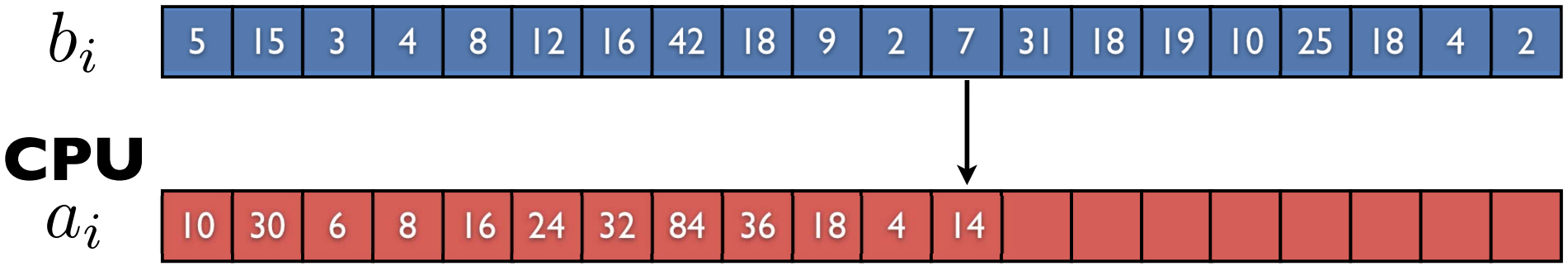
RAPIDMIND

- CUDA
  - Program in **C/C++**
  - **Fortran** support is available
  - **Very** easy to learn and use
  - Close enough to the hardware to obtain near peak performance
  - No limitations on scatter
  - Very good documentation and learning material
- OpenCL = CUDA-like programming model for a variety of different hardware platforms

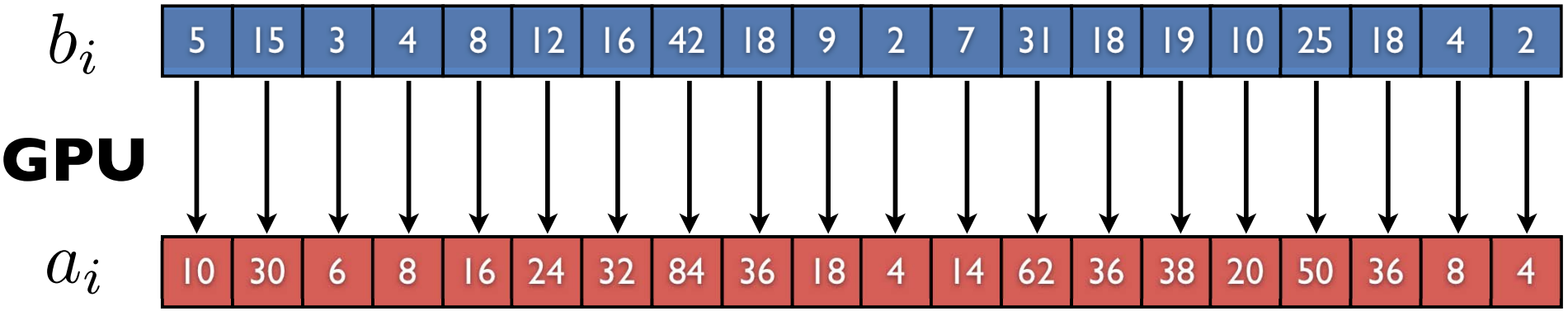
Data-parallel execution model

Example:

$$\textbf{for } i \text{ from } 1 \text{ to } N$$
$$a_i = 2 \cdot b_i$$



data-parallel: $a_i = 2 \cdot b_i$ –run N threads



CUDA tools

- **Compiler:** nvcc
  - C/C++ compiler with a few special directives for programming the GPU
- **Debugger:** cuda-gdb (*linux*) and nexus (*windows*)
  - Full hardware debugger, capable of setting breakpoints and stepping through code
- **Memory checker:** cuda-memcheck (linux & mac)
- **Profiler:** cudaprof
  - Accesses numerous performance counters on the GPU
- **CUBLAS:** A full BLAS implementation on the GPU
- **CUFFT:** General purpose FFT library that runs on the GPU (API is like FFTW)

Additional resources to learn CUDA

- CUDA **programming guide** and **best practices guide**:
  http://developer.nvidia.com/object/cuda_3_0_downloads.html
  Read them cover to cover
- CUDA SDK (a set of good examples of CUDA applications
  - http://developer.nvidia.com/object/cuda_sdk_samples.html
- The CUDA Forums:
  http://forums.nvidia.com/index.php?showforum=62
- Course at UIUC:
  http://courses.ece.illinois.edu/ece498/al/
- Many-core summer school, Virtual School of Computational Science & Engineering:
  http://www.greatlakesconsortium.org
- Many more links:
  http://www.nvidia.com/object/cuda_education.html

Conclusions

- Many-core processors
  - Widely available and inexpensive GPUs
  - Provide 10-100x performance boosts
  - Drastically decrease the time to discovery
  - Architecture differs from standard CPUs
  - Necessitates a new programming model - *more on this in the next lectures*