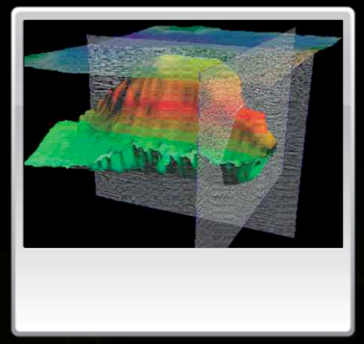
Manycore Parallel Computing with CUDA

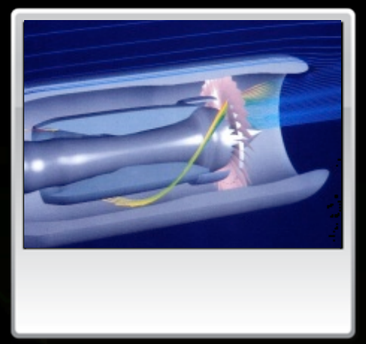
Mark Harris

mharris@nvidia.com

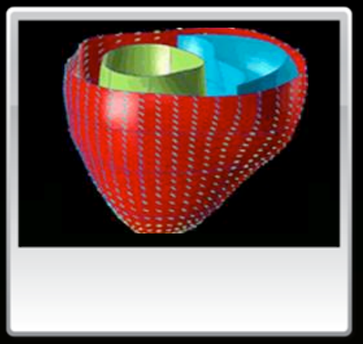
Future Science & Engineering Breakthroughs Hinge on Computing



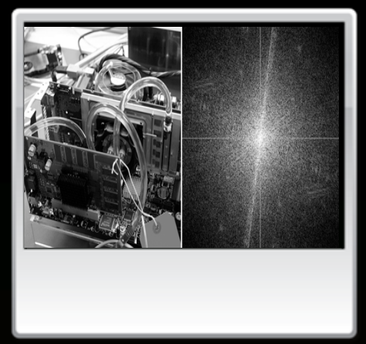
Computational Geoscience



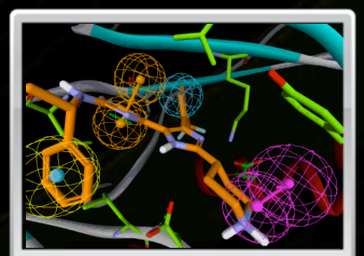
Computational Modeling



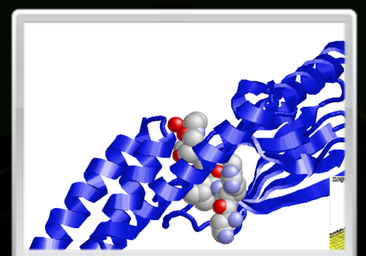
Computational Medicine



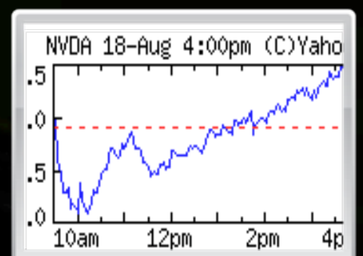
Computational Physics



Computational Chemistry



Computational Biology



Computational Finance

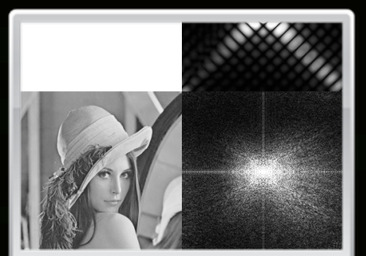


Image Processing

Faster is not “Just faster”

2-3x **“Just faster”**

Do a little more, wait a little less

Doesn’t change how you work

5-10x **“Significant”**

Worth upgrading

Worth rewriting (parts of) your application

100x+ “Fundamentally Different”

Worth considering a new platform

Worth re-architecting your application

Makes new applications possible

Drives down “time to discovery”

Creates fundamental changes in science

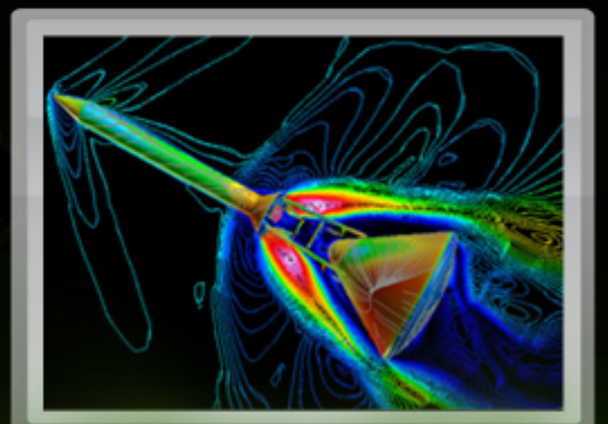
Parallel Computing with CUDA

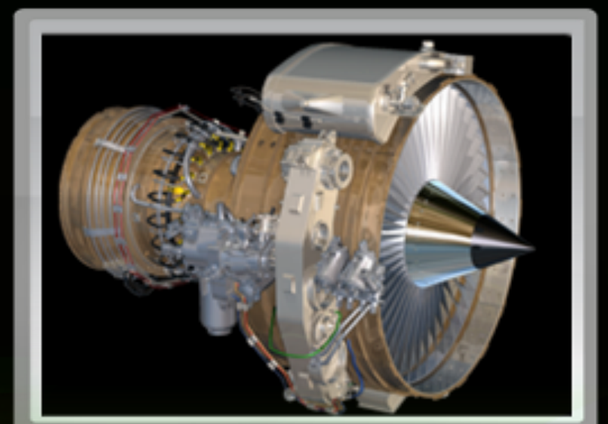
- Enabling new science and engineering
  - By drastically reducing time to discovery
  - Engineering design cycles: from days to minutes, weeks to days
- Enabling new computer science
  - By reinvigorating research in parallel algorithms, programming models, architecture, compilers, and languages

GeForce®
Entertainment

Tesla™
High-Performance Computing

Quadro®
Design & Creation



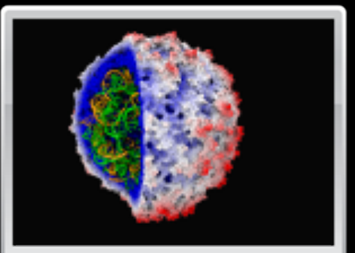


GPU

Wide Developer Acceptance and Success

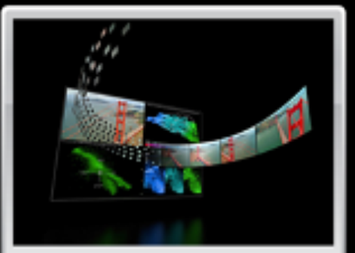
146X

Interactive visualization of volumetric white matter connectivity



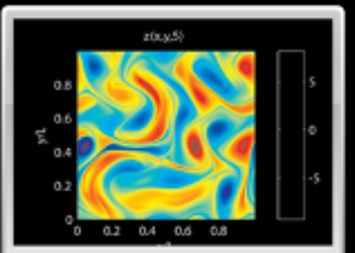
36X

Ion placement for molecular dynamics simulation



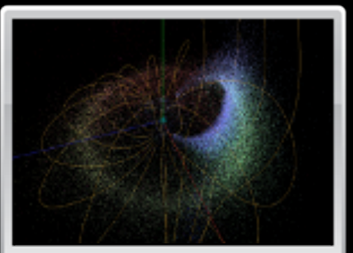
19X

Transcoding HD video stream to H.264



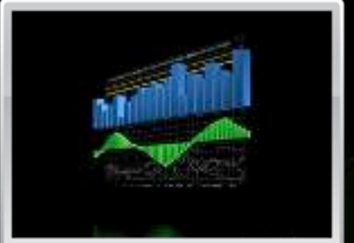
17X

Simulation in Matlab using .mex file CUDA function



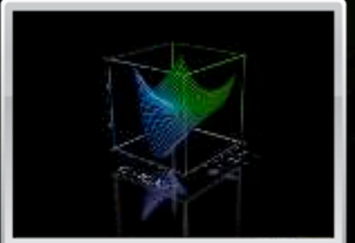
100X

Astrophysics N-body simulation



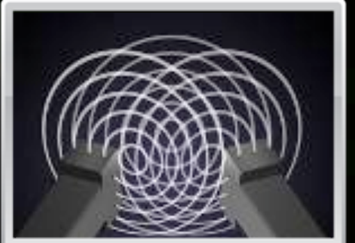
149X

Financial simulation of LIBOR model with swaptions



47X

GLAME@lab: An M-script API for linear Algebra operations on GPU

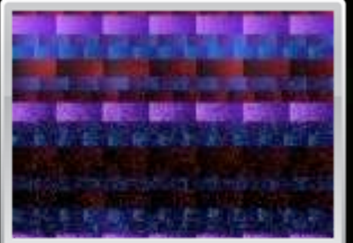


20X

Ultrasound medical imaging for cancer diagnostics

24X

Highly optimized object oriented molecular dynamics



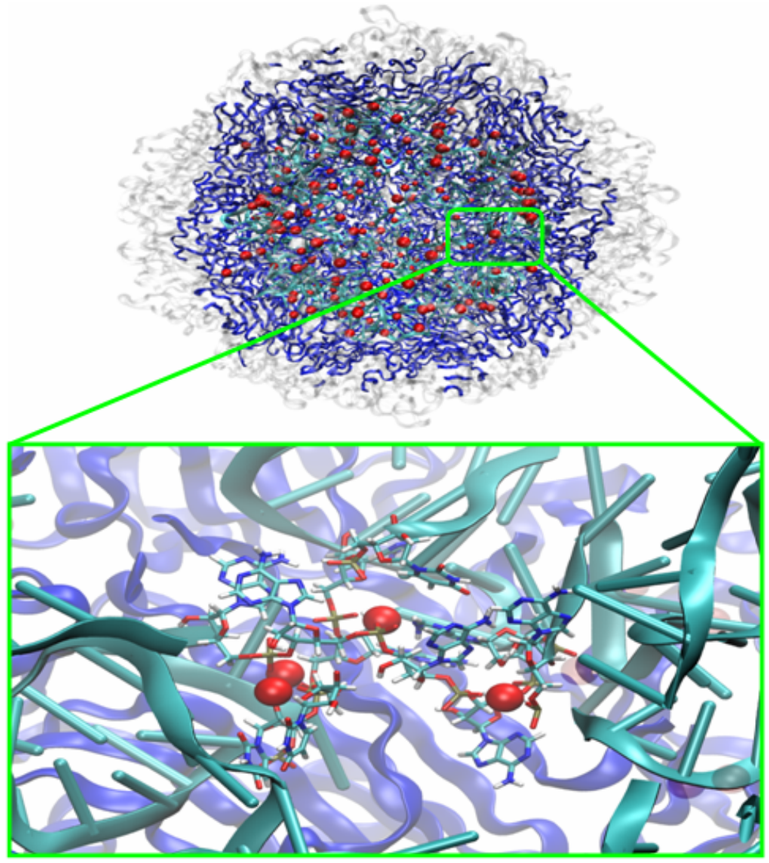
30X

Cmatch exact string matching to find similar proteins and gene sequences

Preparing the Virus for Simulation

John Stone
Beckman Institute, University of Illinois

- Key task: placement of ions inside and around the virus
- 110 CPU-hours on SGI Altix Itanium2
- Larger viruses could require thousands of CPU-hours
- 27 GPU-minutes on G80
- Over 240 times faster - ion placement can now be done on a desktop machine!



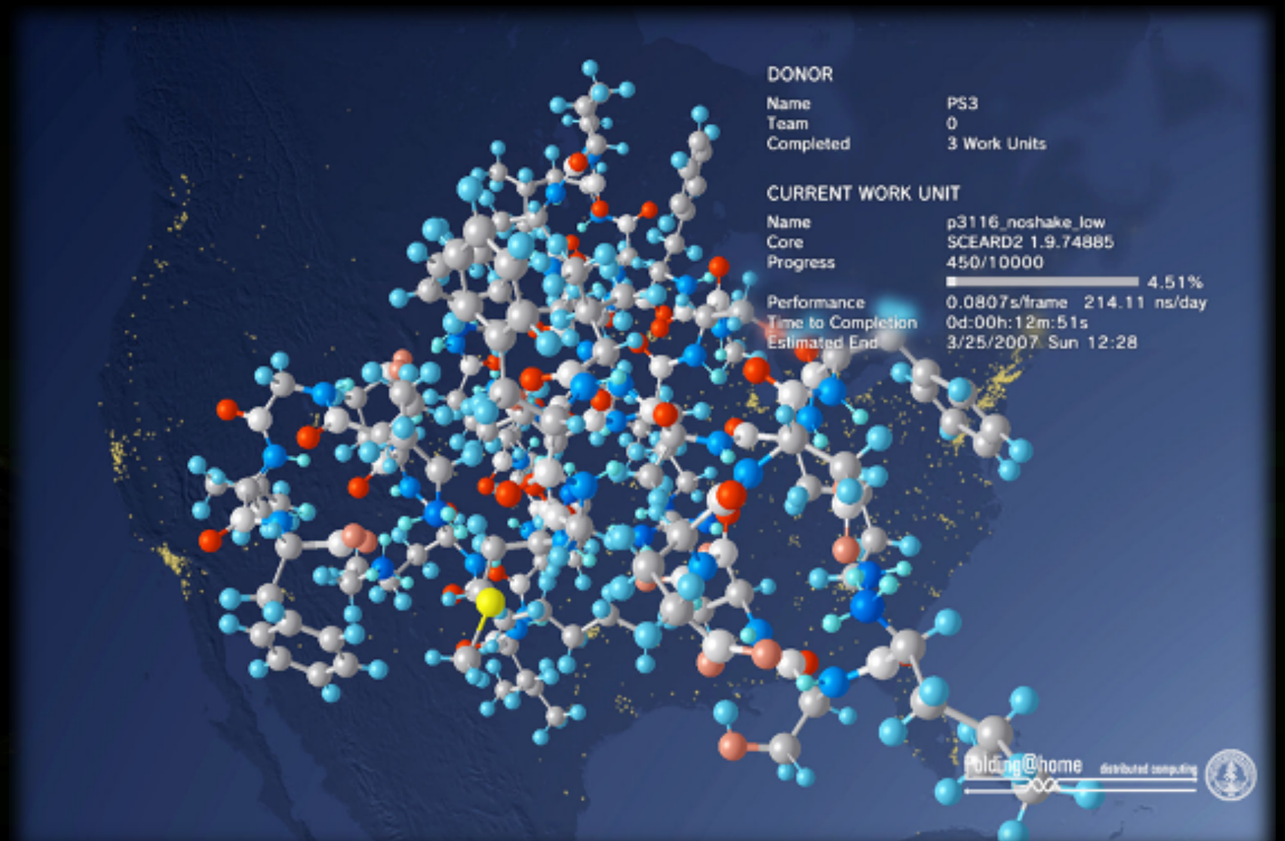
NIH Resource for Macromolecular Modeling and Bioinformatics
http://www.ks.uiuc.edu/

Beckman Institute, UIUC

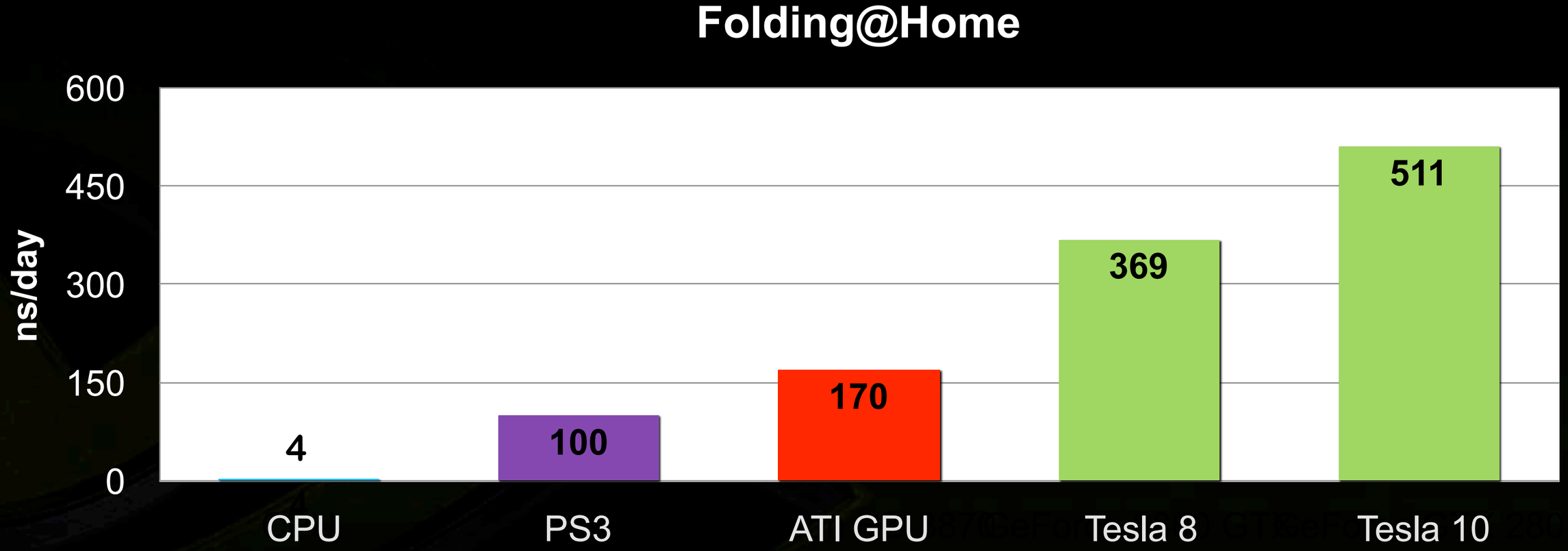
10

Folding@Home

- Distributed computing to study protein folding
  - Alzheimer's Disease
  - Huntington's Disease
  - Cancer
  - Osteogensis imperfecta
  - Parkinson's Disease
  - Antibiotics
- CUDA client available soon!



Molecular Dynamics : GROMACS



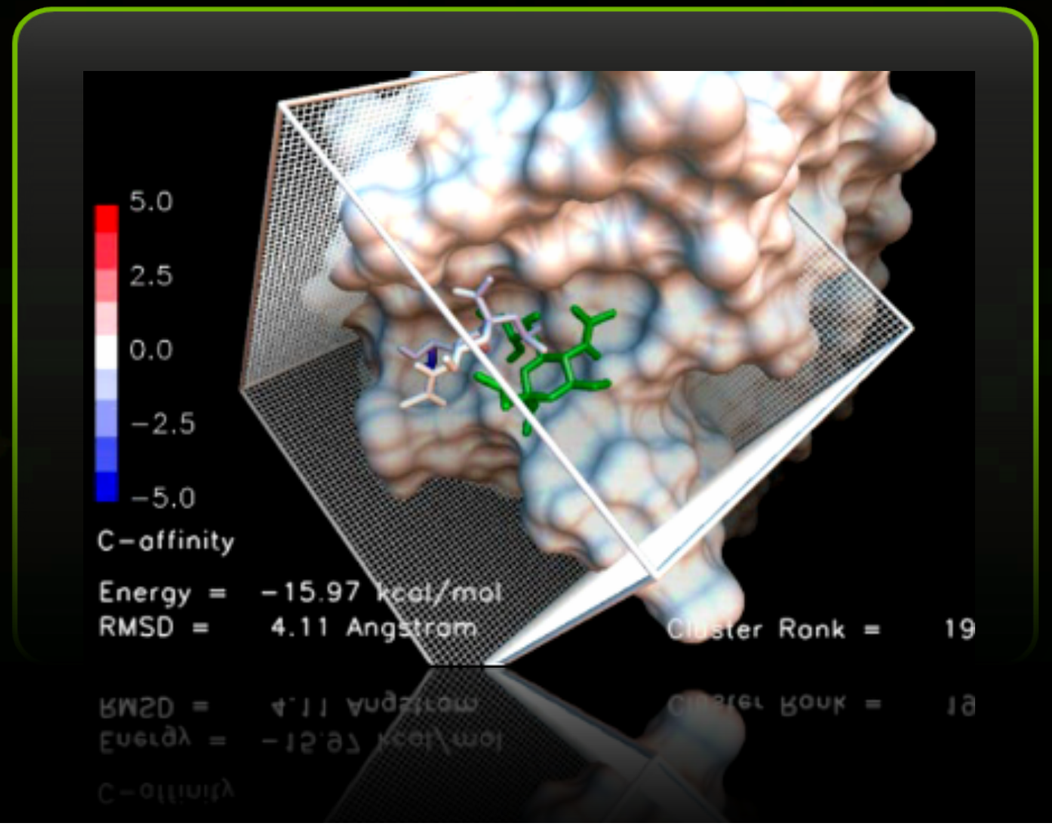
Life Sciences: Autodock for Cancer Research

National Cancer Institute reports 12x speedup

Wait for results reduced from 2 hours to 10 minutes

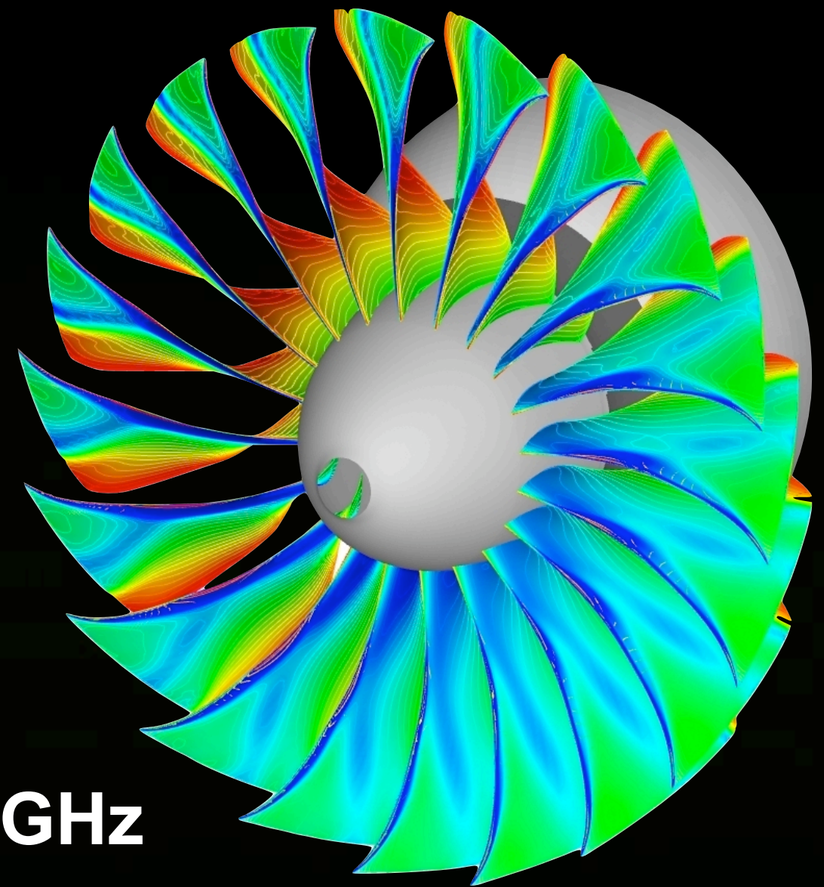
"We can only hope that in the long run, Silicon Informatics' efforts will accelerate the discovery of new drugs to treat a wide range of diseases, from cancer to Alzheimer's, HIV to malaria."

Dr. Garrett Morris, Scripps, Author of AutoDock



Cambridge University Turbomachinery CFD

- Partial differential equations on structured grids
- NVIDIA GPUs enable faster design cycles and new science
- Up to 10 million points on a single GPU
- Scales to many GPUs using MPI
- 10x – 20x speedup Intel Core2 Quad 2.33 GHz



UNIVERSITY OF CAMBRIDGE

WHITTLE LABORATORY

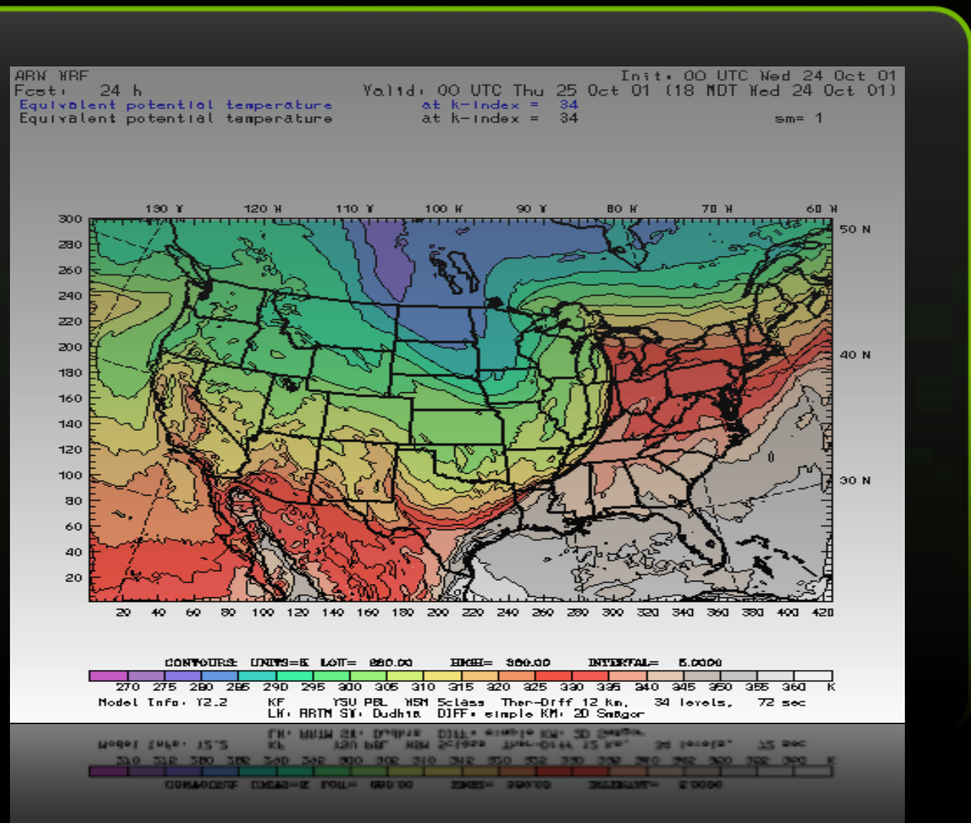
CFD analysis of a jet engine fan, courtesy of Vicente Jerez Fidalgo, Whittle Lab

Slide courtesy of Tobias Brandvik and Graham Pullan, Whittle Lab

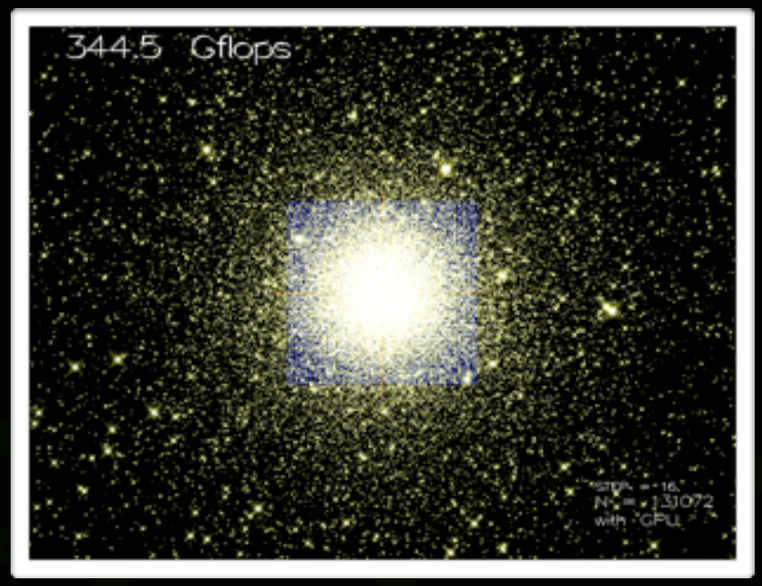
National Center for Atmospheric Research

- Weather Research and Forecast (WRF) model
  - 4000+ registered users worldwide
  - First-ever release with GPU acceleration
- Adapted 1% of WRF code to CUDA
- Resulted in 20% overall speedup
- Ongoing work to adapt more of WRF to CUDA

12km CONUS WRF benchmark
Running on NCSA CUDA cluster



Astrophysics N-Body Simulation



http://progrape.jp/cs/

- 12+ billion body-body interactions per second
- 300 GFLOP/s+ on GeForce 8800 Ultra
  - 1-5 GFLOP/s on single-core CPU
  - Faster than custom GRAPE-6Af n-body computer
- http://www.astrogpu.org/

Finance: Real-time Options Valuation

Hanweck Associates Volera real-time option valuation engine
Value the entire U.S. listed options market in real-time using 3 NVIDIA Tesla S870's

	GPUs	CPUs	Savings
Processors	12	600	
Rack Space	6U	54U	9x
Hardware Cost	$42,000	$262,000	6x
Annual Cost	$140,000	$1,200,000	9x

Figures assume:

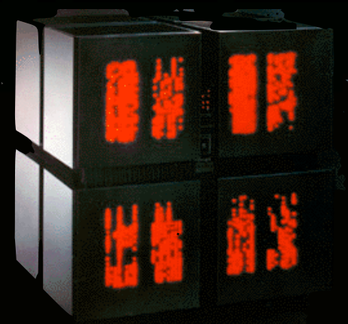
- NVIDIA Tesla S870s with one 8-core host server per unit
- CPUs are 8-core blade servers; 10 blades per 7U
- $1,800/U/month rack and power charges
- 5-year depreciation

Massively Parallel Computing with CUDA

Parallel Computing's Golden Age

- 1980s, early 1990s
  - Particularly data-parallel computing
- Architectures
  - Connection Machine, MasPar, Cray
  - True supercomputers: incredibly exotic, powerful, expensive
- Algorithms, languages, & programming models
  - Solved a wide variety of problems
  - Various parallel algorithmic models developed
  - P-RAM, V-RAM, circuit, hypercube, etc.

Cray X-MP (1982)



Thinking Machines CM-1 (1984)

MasPar MP-1 (1990)

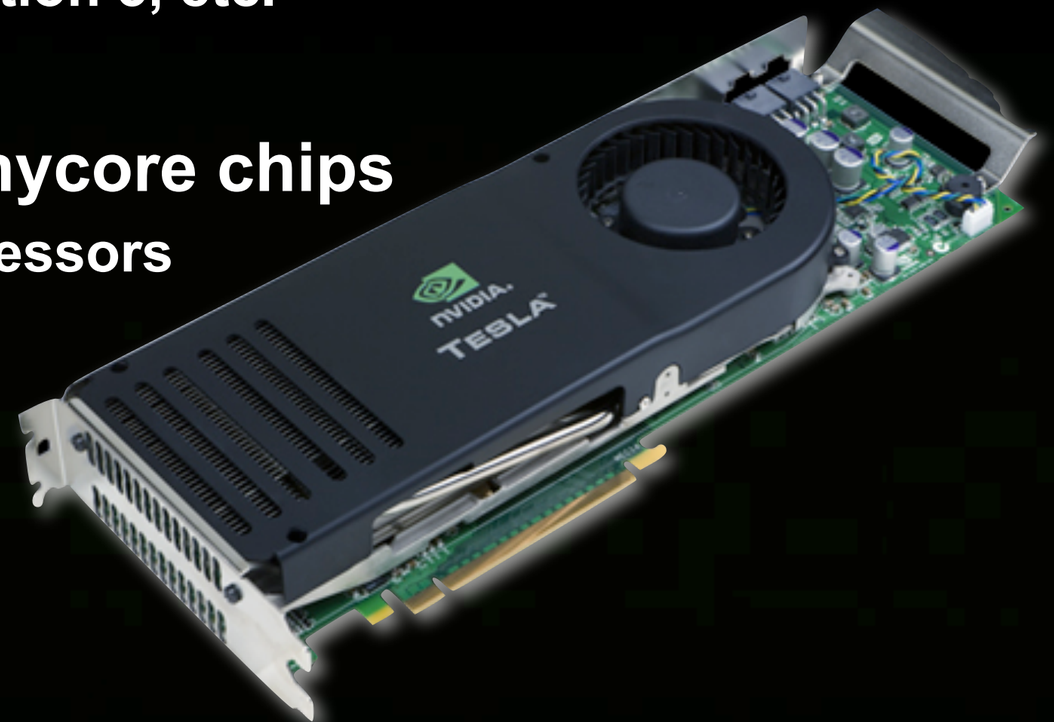
Parallel Computing's Dark Age

- But…impact of data-parallel computing limited
  - Thinking Machines sold 7 CM-1s (100s of systems total)
  - MasPar sold ~200 systems
- Commercial and research activity subsided
  - Massively-parallel machines replaced by clusters of ever more powerful commodity microprocessors
  - Beowulf, Legion, grid computing, …

Massively parallel computing lost momentum to the inexorable advance of commodity technology

Enter the GPU

- GPU = *Graphics Processing Unit*
  - Processor in computer video cards, PlayStation 3, etc.
- GPUs are massively multithreaded manycore chips
  - NVIDIA Tesla products have 128 scalar processors
  - Over 470 GFLOPS sustained performance
  - Over 12,000 concurrent threads



Enter CUDA

- ***CUDA*** **is a scalable parallel programming model and software environment for parallel computing**
- **NVIDIA *TESLA* GPU architecture accelerates CUDA**
  - **Hardware and software designed together for computing**
  - **Expose the computational horsepower of NVIDIA GPUs**
  - **Enable general-purpose *GPU computing***

The Democratization of Parallel Computing

- GPUs and CUDA bring parallel computing to the masses
  - Over 70M CUDA-capable GPUs sold to date
  - 60K CUDA developers
  - A “developer kit” costs ~$200 (for 500 GFLOPS)
- Data-parallel supercomputers are everywhere!
  - CUDA makes this power accessible
  - We’re already seeing innovations in data-parallel computing

Massively parallel computing has become a commodity technology!

CUDA

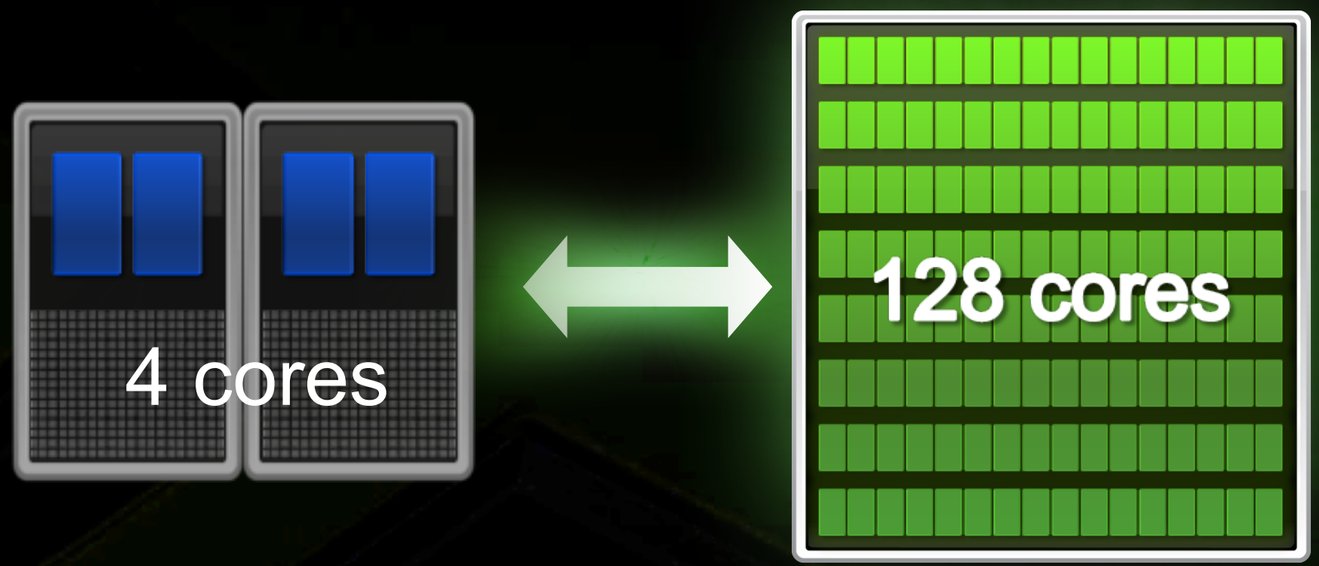
CUDA

A scalable parallel programming model and software environment for parallel computing

Minimal extensions to familiar C/C++ environment

Heterogeneous serial-parallel programming model

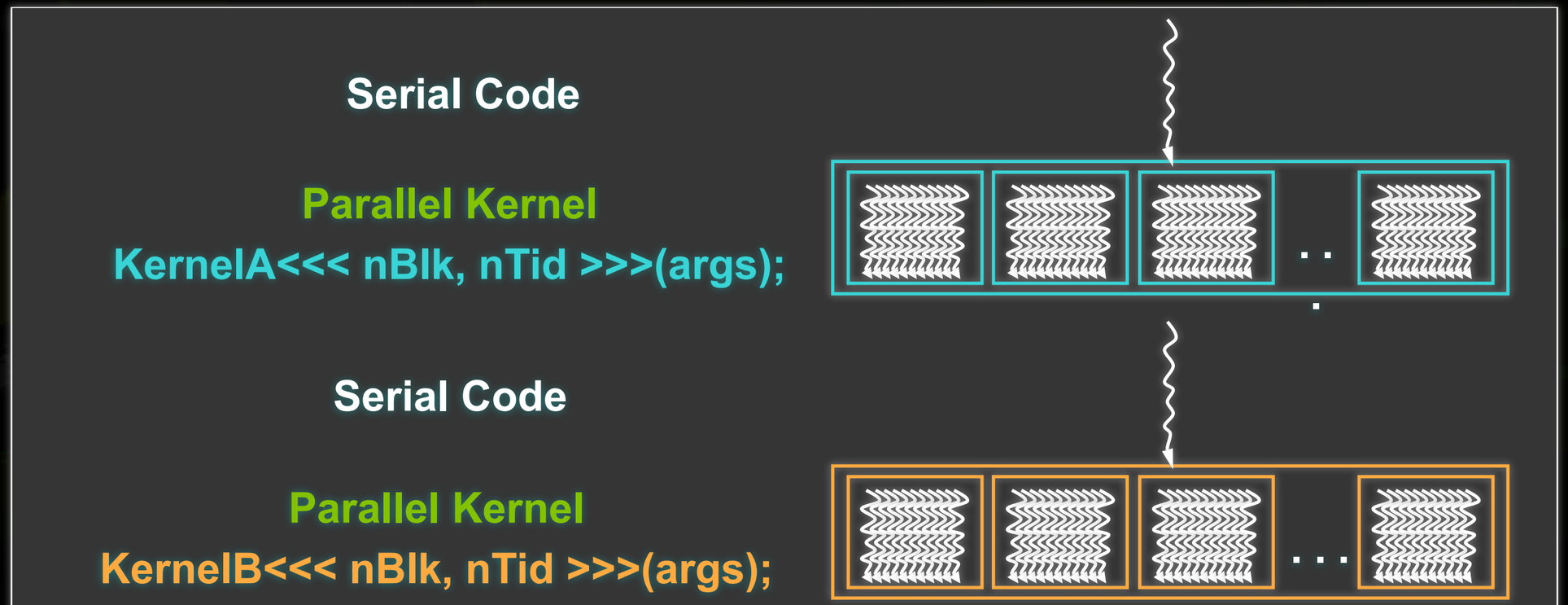
Heterogeneous Computing



Multi-Core ***plus*** **Many-Cores**

Heterogeneous Programming

- CUDA = serial program with parallel kernels, all in C
  - Serial C code executed by a CPU thread
  - Parallel kernel C code executed by *thread blocks* across multiple processing elements

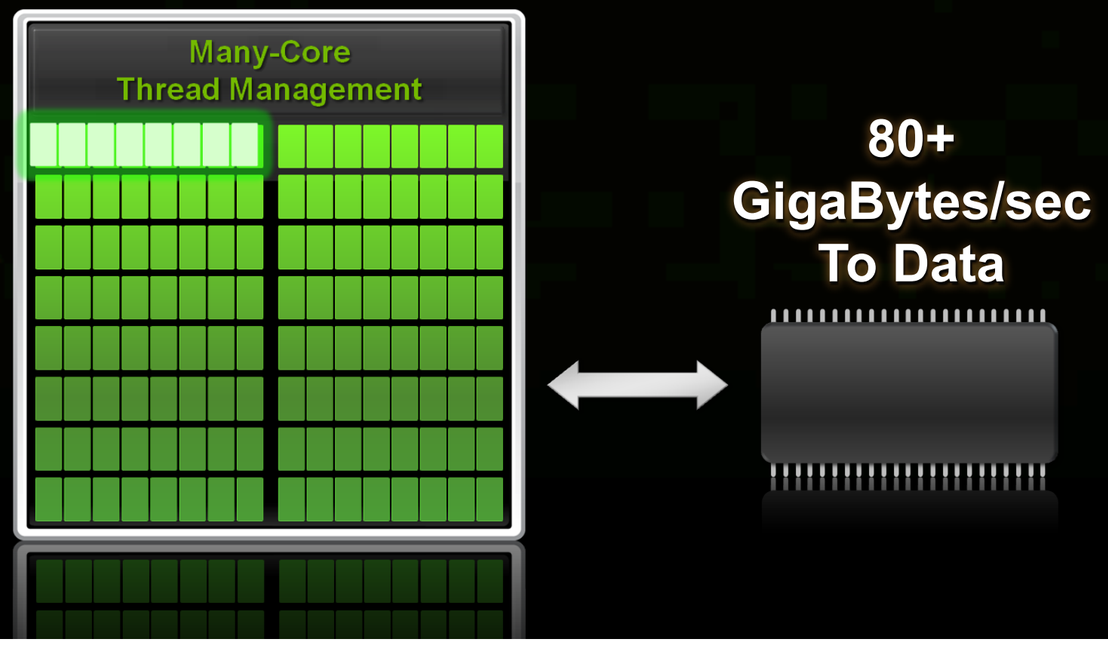


A Highly Multithreaded Coprocessor

- The GPU is a highly parallel compute coprocessor
  - serves as a coprocessor for the host CPU
  - has its own device memory with high bandwidth interconnect
  - executes many threads in parallel

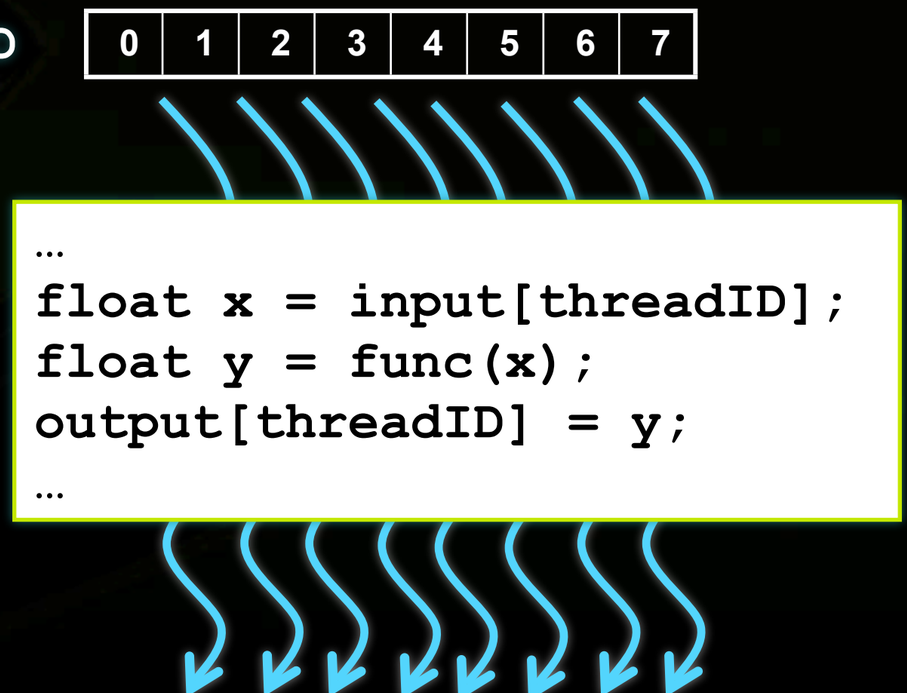
- Parallel portions of an application are executed as kernels
  - Many threads execute each kernel

- CUDA threads
  - extremely lightweight
    - Very little creation overhead,
    - Instant switching
  - GPU uses 1000s of threads for efficiency



Arrays of Parallel Threads

- A CUDA kernel is executed by an array of threads
  - All threads run the same code
  - Each thread uses its ID to compute addresses and make control decisions



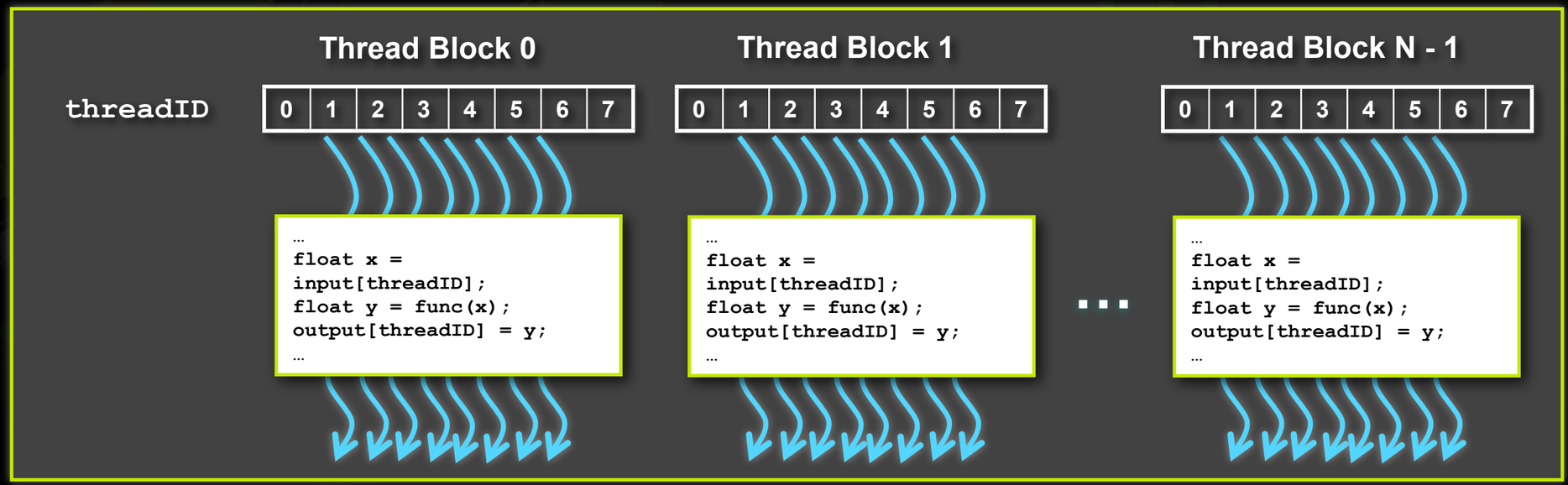
Thread Cooperation

- Share results to save computation
- Share memory accesses for drastic bandwidth reduction

- Thread cooperation is a powerful feature of CUDA
- Threads can cooperate via on-chip shared memory and synchronization

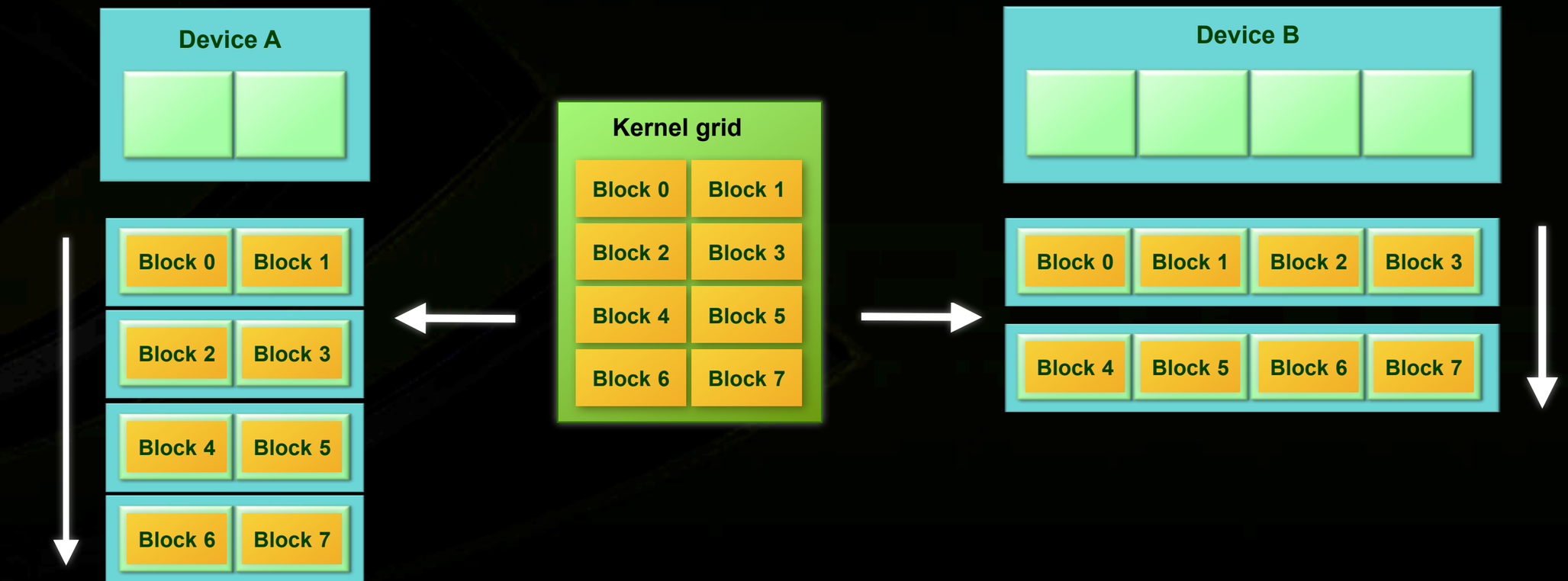
Thread Blocks: Scalable Cooperation

- Divide monolithic thread array into multiple blocks
  - Threads within a block cooperate via shared memory
  - Threads in different blocks cannot cooperate
- Enables programs to transparently scale to any number of processors!



Transparent Scalability

- Hardware is free to schedule thread blocks on any processor
  - Kernels scale to any number of parallel multiprocessors



Simple “C” Extensions to Express Parallelism

Standard C Code

```
void
saxpy_serial(int n, float a,
             float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}
// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

CUDA C Code

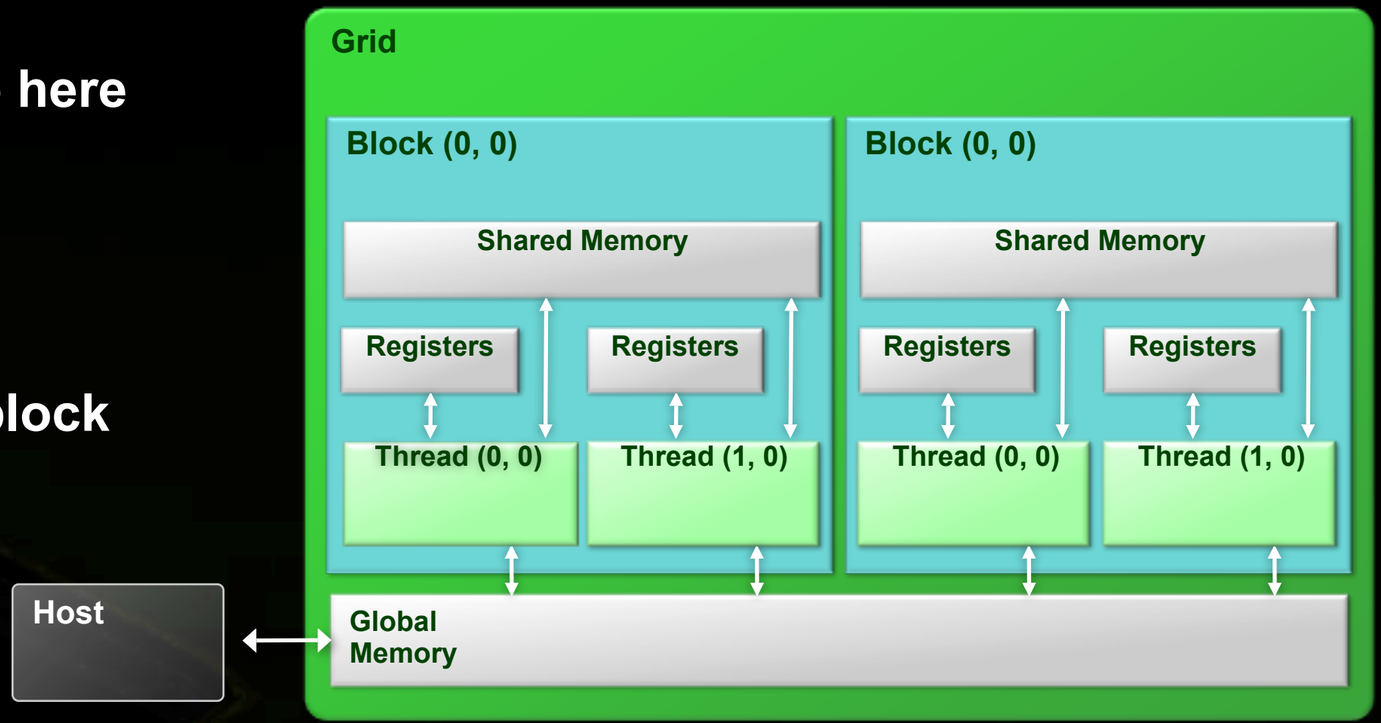
```
__global__ void
saxpy_parallel(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x +
            threadIdx.x;
    if (i < n)  y[i] = a*x[i] + y[i];
}
// Invoke parallel SAXPY kernel with
// 256 threads/block
int nblocks = (n + 255) / 256;

saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```


32

Kernel Memory Access

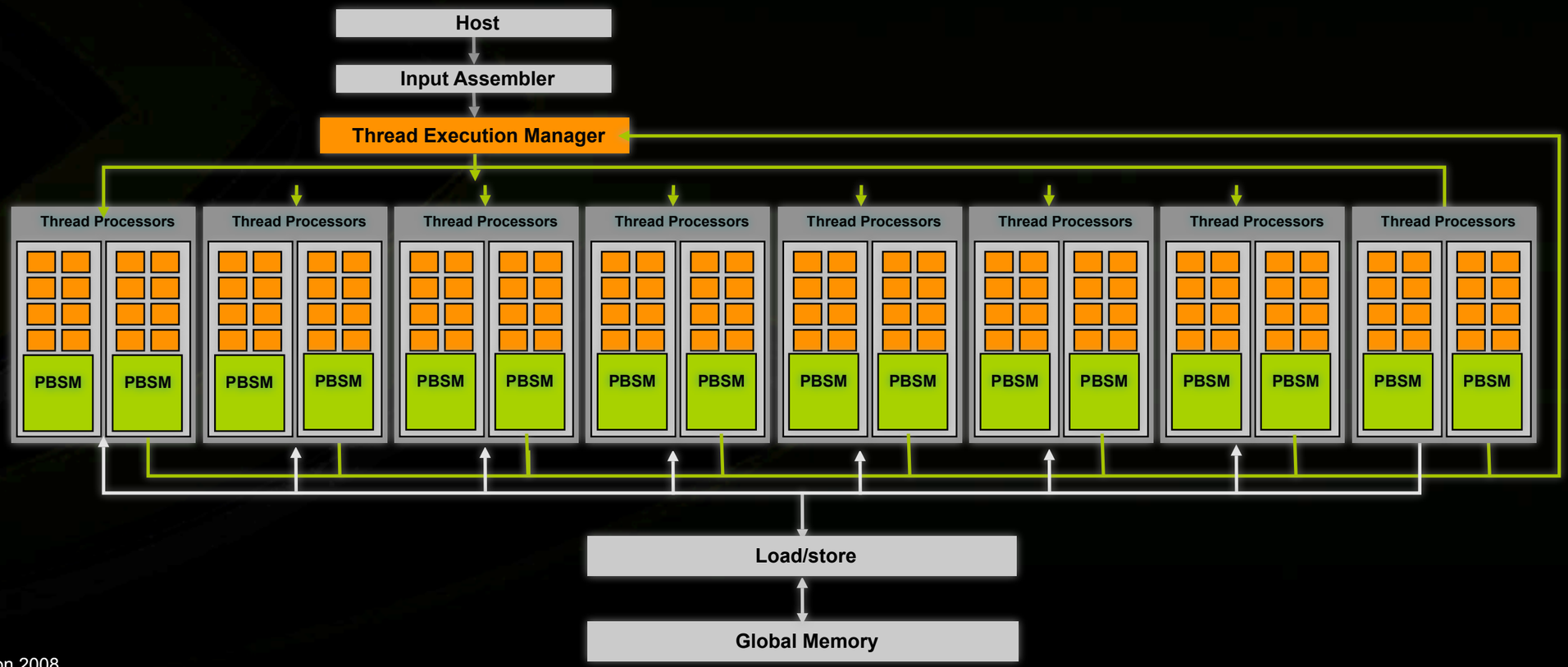
- Registers
- Global Memory
  - Kernel input and output data reside here
  - Off-chip, large
  - Uncached
- Shared Memory
  - Shared among threads in a single block
  - On-chip, small
  - As fast as registers



- The host can read & write global memory but not shared memory

Manycore GPU – Block Diagram

- G80 (launched Nov 2006 – GeForce 8800 GTX)
- 128 Thread Processors (16 multiprocessors) execute kernel threads
- Up to 12,288 parallel threads active
- Per-block shared memory (PBSM) accelerates processing

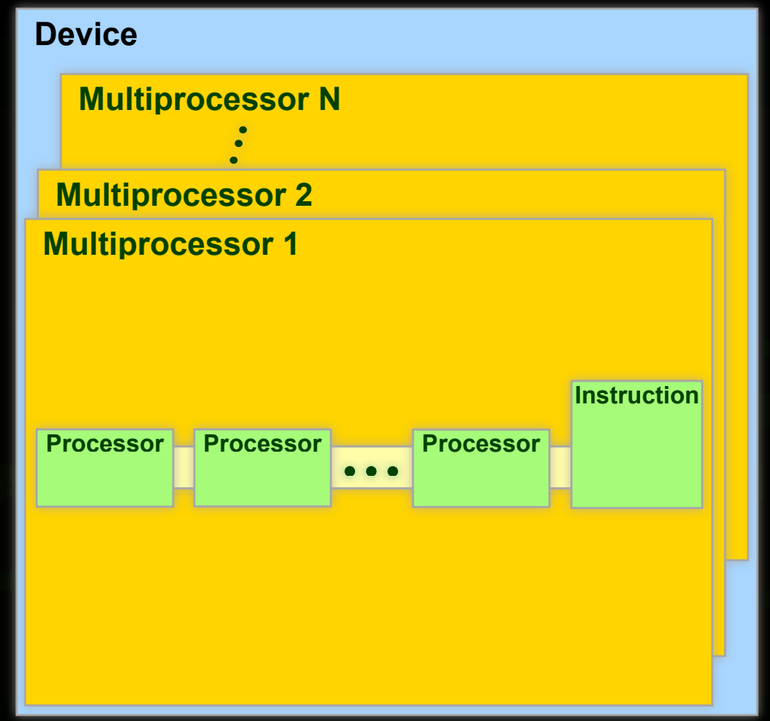


Hardware Implementation: Collection of SIMT Multiprocessors

- Each multiprocessor is a set of SIMT thread processors
  - Single Instruction Multiple Thread

- Each thread processor has:
  - program counter, register file, etc.
  - scalar data path
  - read/write memory access

- Unit of SIMT execution: warp
  - execute same instruction/clock
  - Hardware handles thread scheduling and divergence *transparently*



- Warps enable a friendly data-parallel programming model

The Keys to GPU Computing Performance

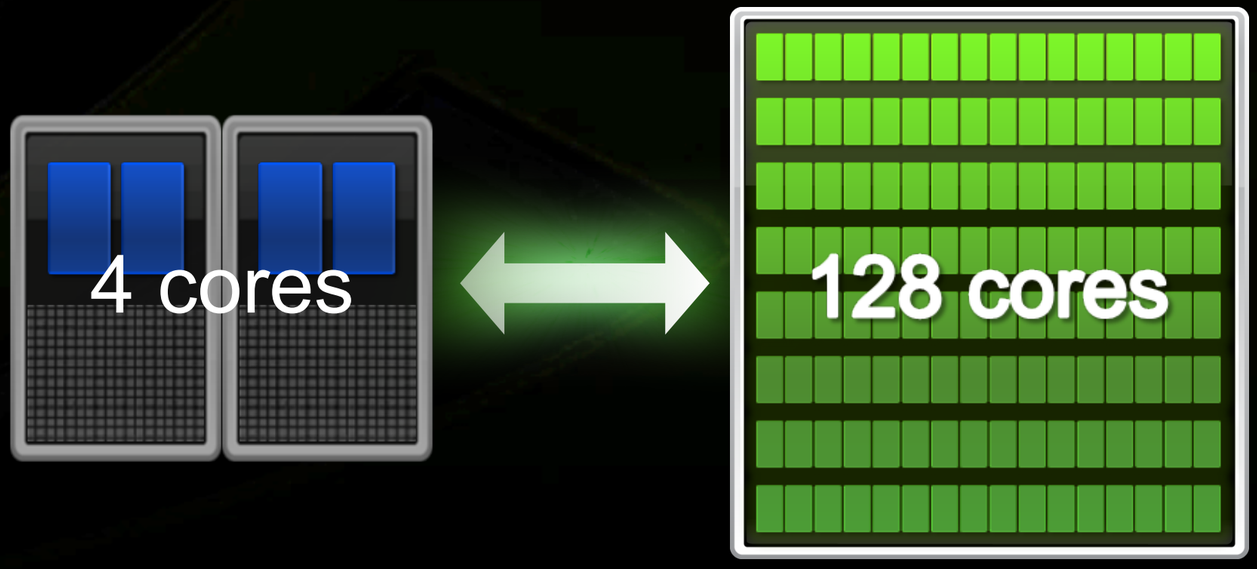
- **Hardware Thread Management**
  - Thousands of lightweight concurrent threads
  - No switching overhead
  - Hide instruction and memory latency
- **On-Chip Shared memory**
  - User-managed data cache
  - Thread communication / cooperation within blocks
- **Random access to global memory**
  - Any thread can read/write any location(s)
  - Direct host access

Application Software
Industry Standard C Language

Libraries

cuFFT | cuBLAS | cuDPP

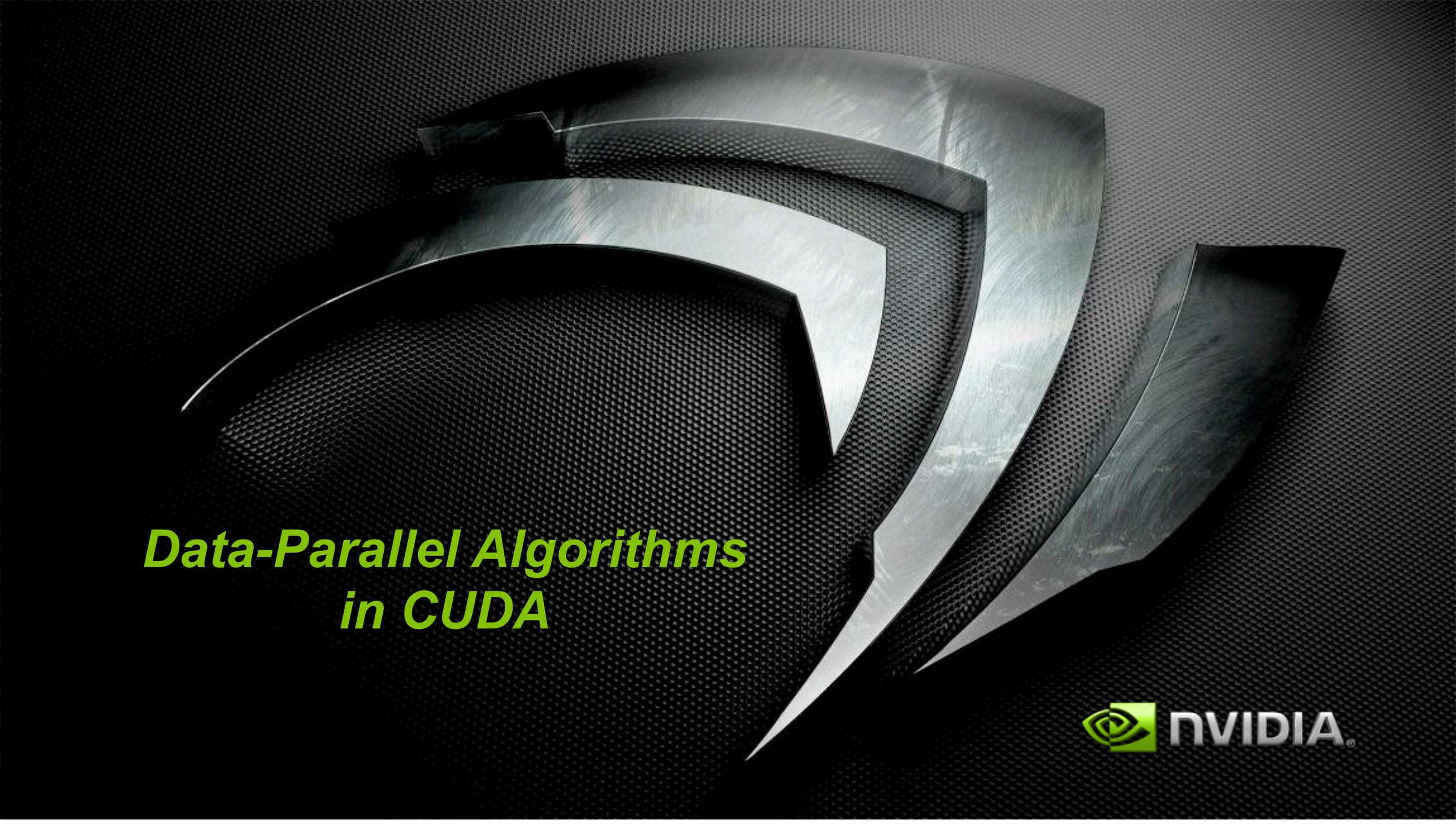
System	CUDA Compiler	CUDA Tools
1U PCI-E Switch	**C** **Fortran**	Debugger Profiler



A New Platform: Tesla

- HPC-oriented product line
  - C870: board (1 GPU)
  - D870: deskside unit (2 GPUs)
  - S870: 1u server unit (4 GPUs)

Data-Parallel Algorithms in CUDA



Common Situations in Parallel Computation

- Many parallel threads need to generate a single result value
  - Reduce
- Many parallel threads that need to partition data
  - Split
- Many parallel threads and variable output per thread
  - Compact / Expand / Allocate

Split Operation

Given an array of true and false elements (and payloads)

Flag	T	F	F	T	F	F	T	F
Payload	3	1	7	0	4	1	6	3

Split Operation

Given an array of true and false elements (and payloads)

Flag	T	F	F	T	F	F	T	F
Payload	3	1	7	0	4	1	6	3

Return an array with all true elements at the beginning

T	T	T	F	F	F	F	F
3	0	6	1	7	4	1	3

Split Operation

Given an array of true and false elements (and payloads)

Flag	T	F	F	T	F	F	T	F
Payload	3	1	7	0	4	1	6	3

Return an array with all true elements at the beginning

T	T	T	F	F	F	F	F
3	0	6	1	7	4	1	3

Examples: sorting, building trees

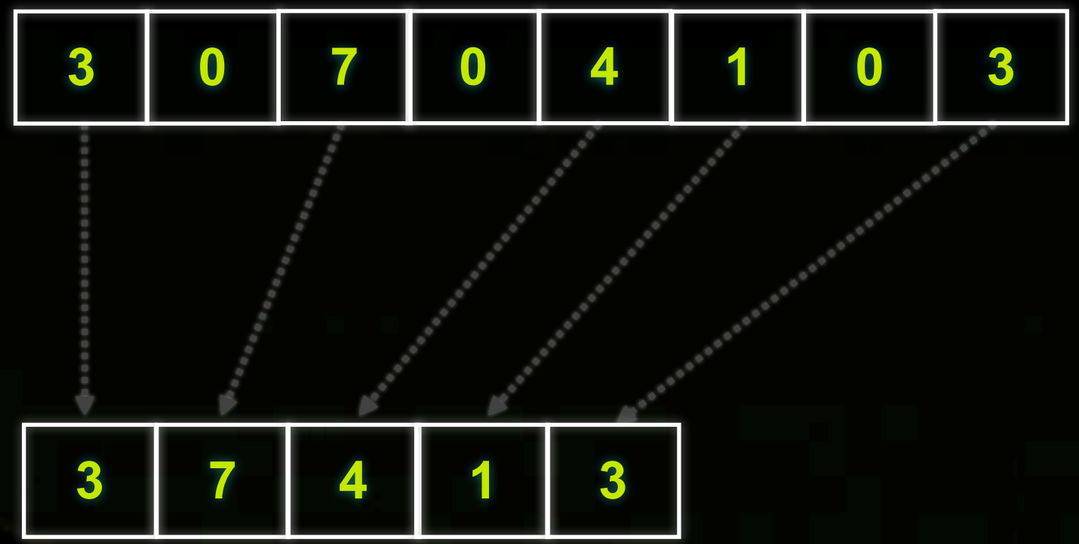
Variable Output Per Thread: Compact

Remove null elements

Example: collision detection

Variable Output Per Thread: Compact

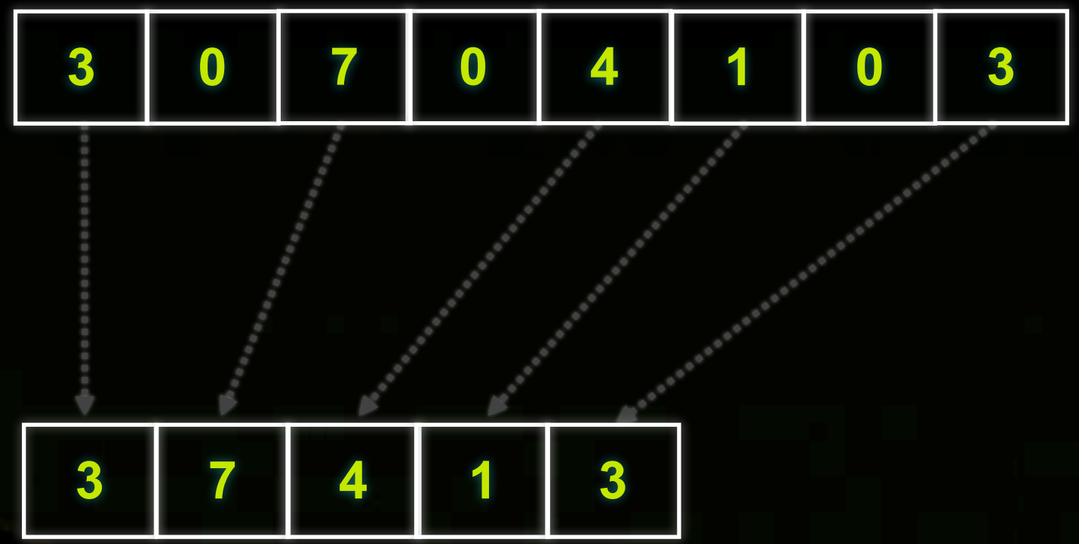
Remove null elements



Example: collision detection

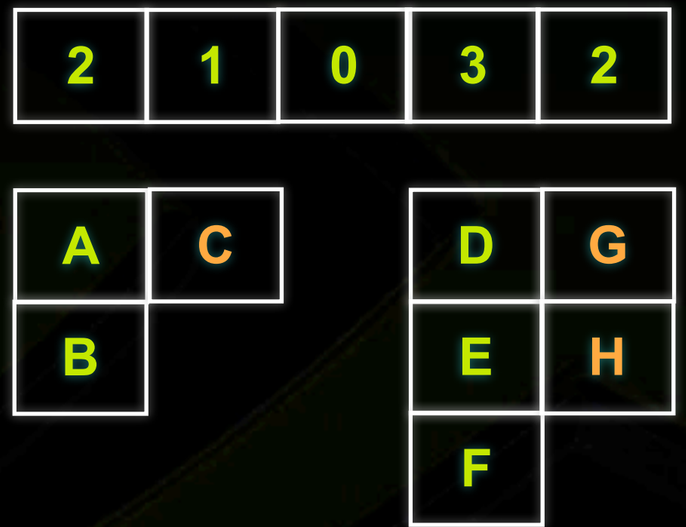
Variable Output Per Thread: Compact

Remove null elements



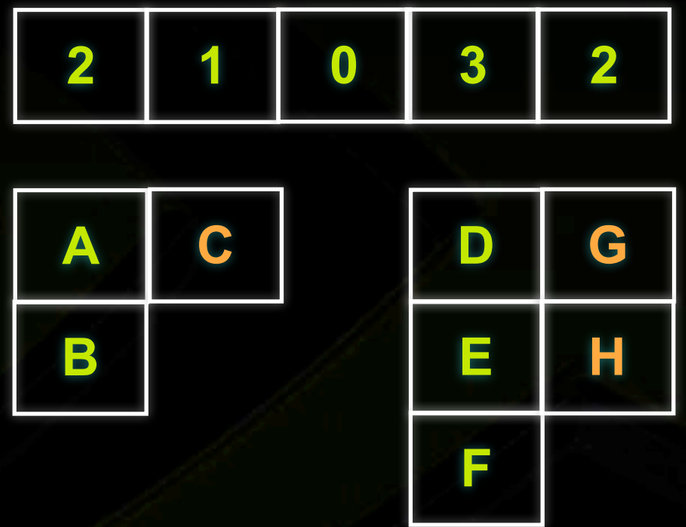
Example: collision detection

Variable Output Per Thread: General Case



Variable Output Per Thread: General Case

Allocate Variable Storage Per Thread



Variable Output Per Thread: General Case

Allocate Variable Storage Per Thread

Variable Output Per Thread: General Case

Allocate Variable Storage Per Thread

Examples: marching cubes, geometry generation

“Where do I write my output?”

- In each case, every thread must answer this simple question
- The answer is:
  - “That depends (on how much the other threads write)!”
- “Scan” is an efficient way to answer this question in parallel

Parallel Prefix Sum (Scan)

- Given a sequence $A = [a_0, a_1, \ldots, a_{n-1}]$
  and a binary associative operator $\oplus$ with identity I,

$$\text{scan}(A) = [I, a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \ldots \oplus a_{n-2})]$$

- Example: if $\oplus$ is addition, then scan on the sequence

[3 1 7 0 4 1 6 3]

returns the sequence

[0 3 4 11 11 15 16 22]

Applications of Scan

- Scan is a simple and useful parallel building block for many parallel algorithms:
  - radix sort
  - quicksort (segmented scan)
  - string comparison
  - lexical analysis
  - stream compaction
  - run-length encoding
  - line of sight
  - polynomial evaluation
  - solving recurrences
  - tree operations
  - histograms
  - allocation
  - graph operations
  - summed area tables
  - etc.
- Fascinating, since scan is unnecessary in sequential computing!

Segmented Scan

Segment Head Flags	0	0	1	0	0	1	0	0
Input Data Array	3	1	7	0	4	1	6	3

Segmented scan	0	3	0	7	7	0	1	7

Segmented Scan

Segment Head Flags	0	0	1	0	0	1	0	0
Input Data Array	3	1	7	0	4	1	6	3

Segmented scan	0	3	0	7	7	0	1	7

- Segmented scan enables another class of parallel algorithms
  - Parallel quicksort, sparse matrix-vector multiply, hierarchical data structures

Segmented Scan

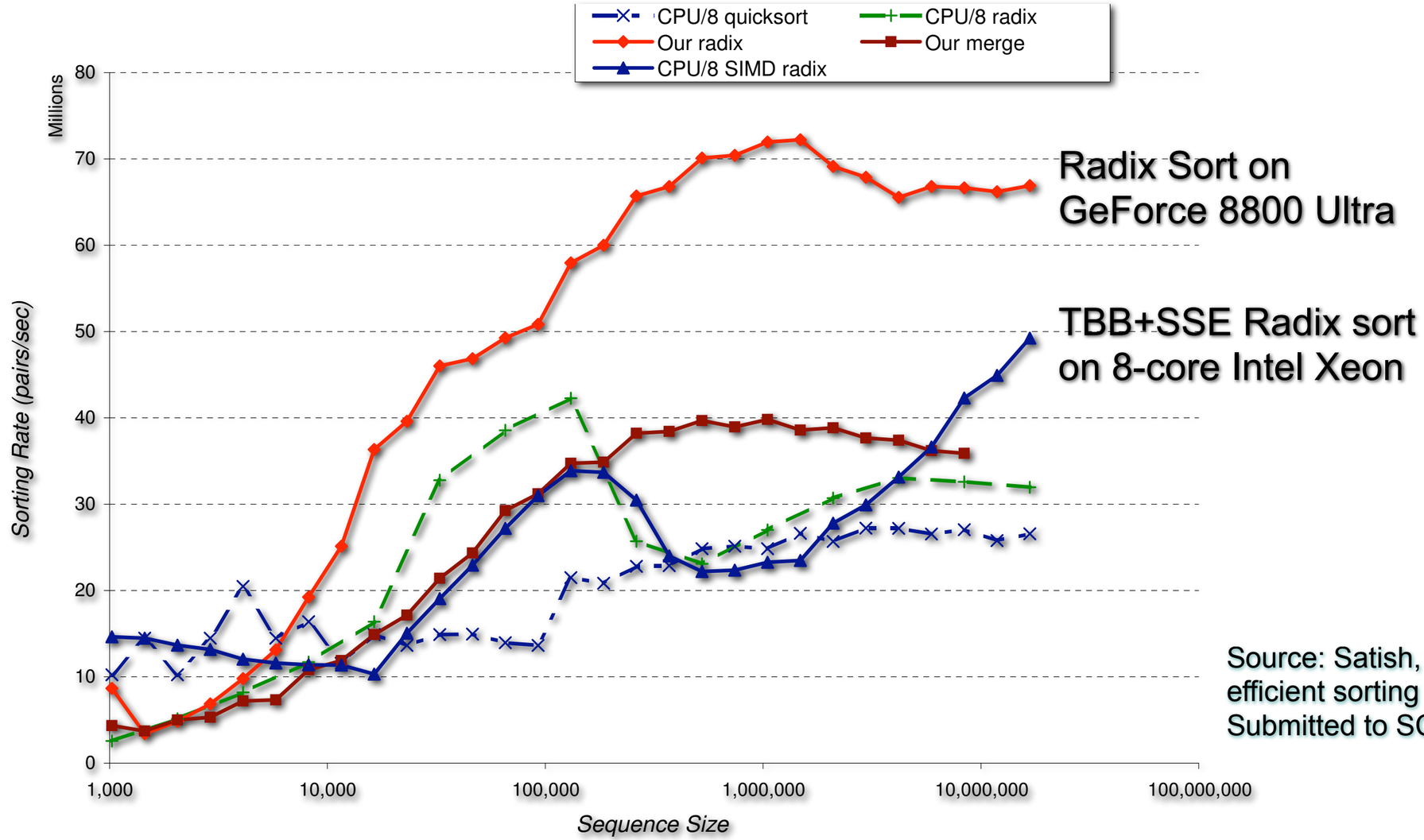
Segment Head Flags	0	0	1	0	0	1	0	0
Input Data Array	3	1	7	0	4	1	6	3

Segmented scan	0	3	0	7	7	0	1	7

- Segmented scan enables another class of parallel algorithms
  - Parallel quicksort, sparse matrix-vector multiply, hierarchical data structures

- Sengupta, Harris, Zhang, Owens. "Scan Primitives for GPU Computing". *Proceedings of Graphics Hardware 2007*
- Sengupta, Harris, Garland. "Data-Parallel GPU Computing". *Under review*
- Dotsenko, Govindaraju, Sloan, Boyd, Manferdelli. "Fast Scan Algorithms on Graphics Processors". *Proceedings of ICS 2008.*

Sorting in CUDA: Radix and Merge Sort



Source: Satish, Harris, and Garland. “Designing efficient sorting algorithms for manycore GPUs”. Submitted to SC08

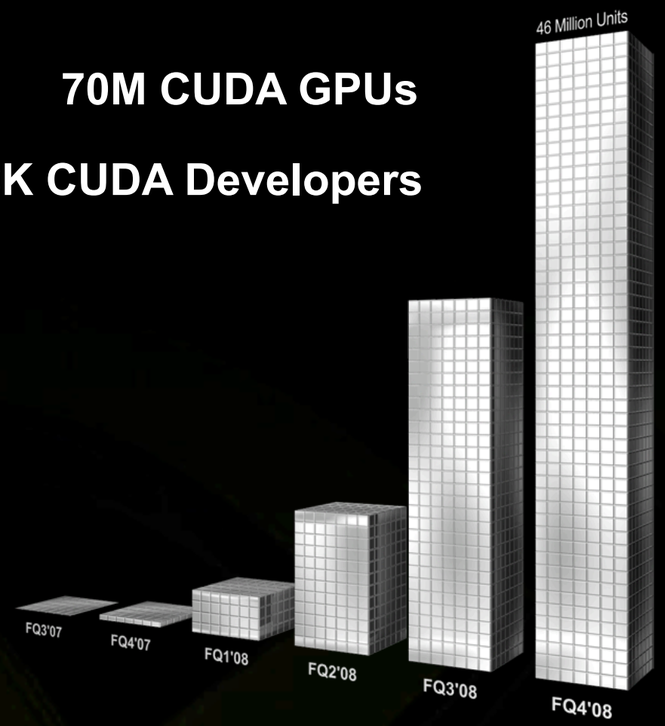
Figure 11. Sorting rates for our GPU sorts compared with an 8-core Intel Xeon system.

cuDPP: CUDA Data-Parallel Primitives Library

- Provide easy access to important data-parallel algorithms in CUDA
  - Scan
  - Segmented Scan
  - Radix sort, merge sort
  - Sparse matrix-vector multiply
  - Coming soon: parallel reduction, faster radix and merge sorts, quicksort
- Open source collaboration between NVIDIA and UC Davis
  - Shubho Sengupta, John Owens, Mark Harris
- http://www.gpgpu.org/developer/cudpp

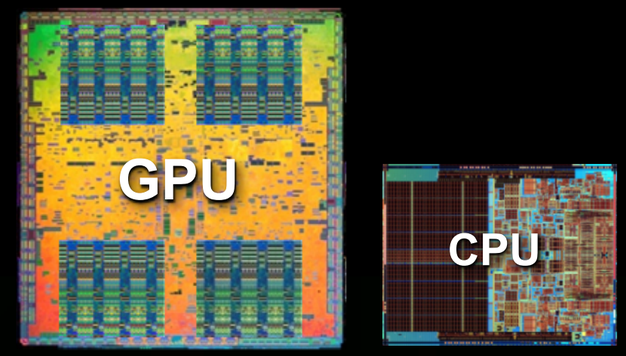
70M CUDA GPUs

60K CUDA Developers

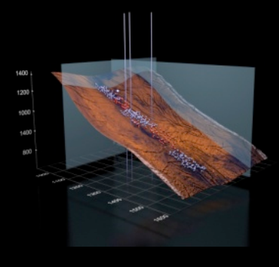


Questions?

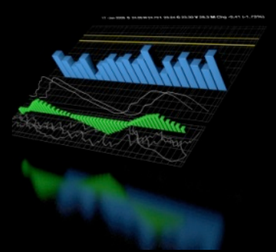
CUDA



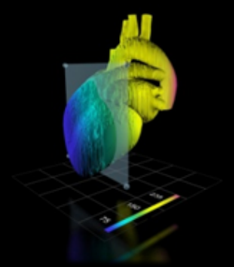
Heterogeneous Computing



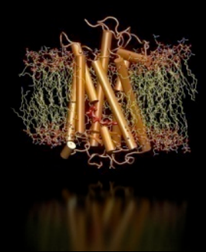
Oil & Gas



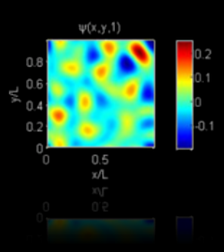
Finance



Medical



Biophysics



Numerics

Audio

Video

Imaging