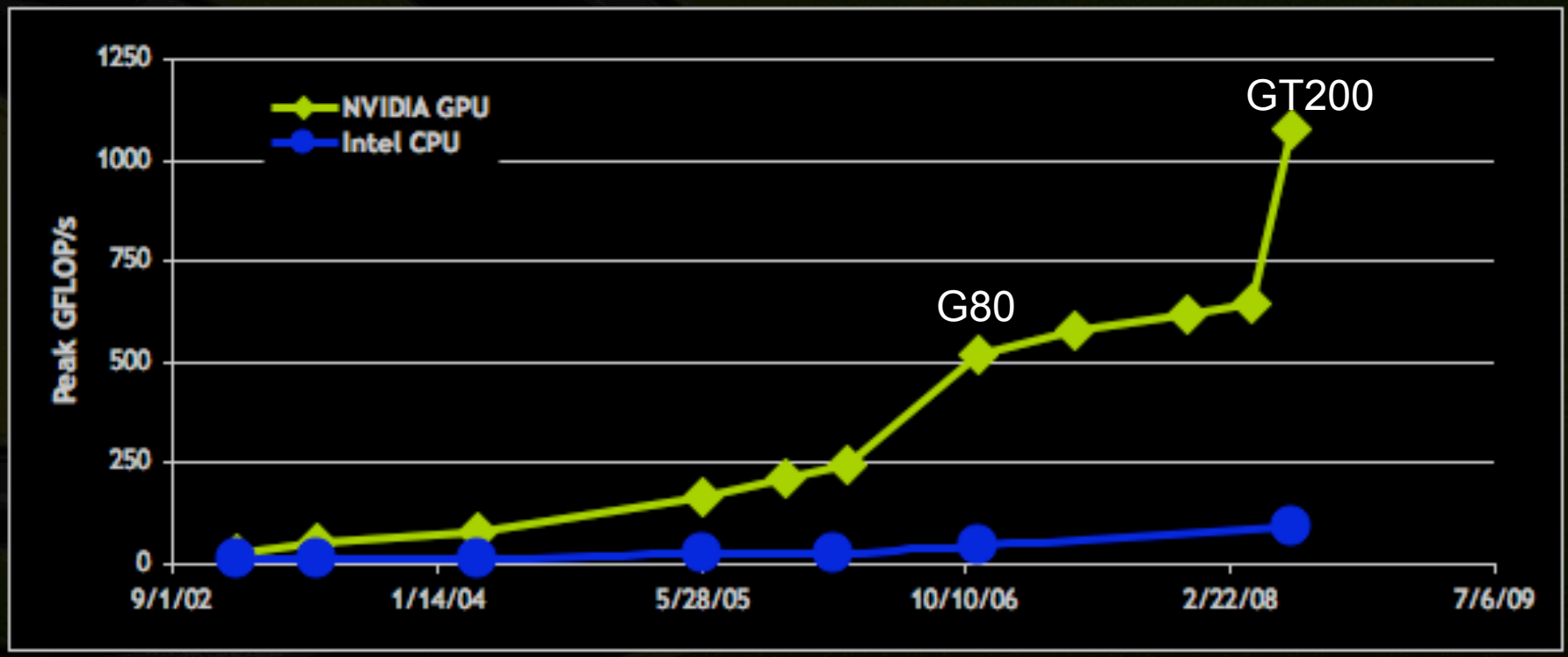

High Performance
Computing with CUDA
Massimiliano Fatica
NVIDIA Corporation
NVIDIA

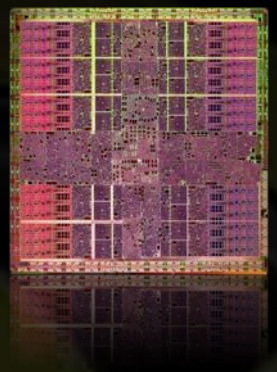
GPU Performance History

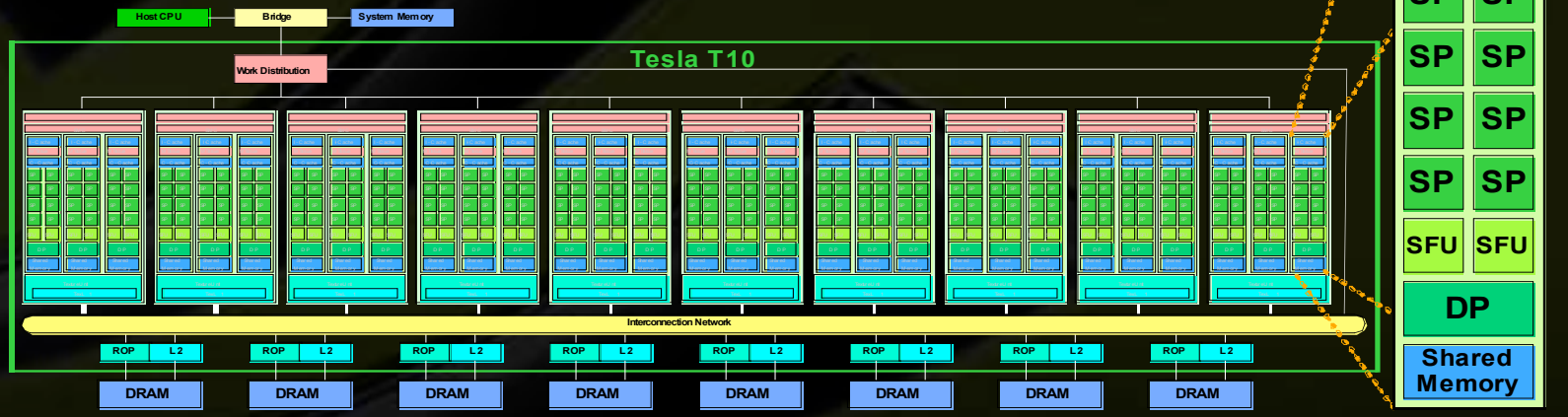
- GPUs are massively multithreaded many-core chips
  - Hundreds of cores, thousands of concurrent threads
  - Huge economies of scale
  - Still on aggressive performance growth
  - High memory bandwidth



CUDA Computing with Tesla T10

- 240 SP processors at 1.44 GHz: 1 TFLOPS peak
- 30 DP processors at 1.44 GHz: 86 GFLOPS peak
- 128 threads per processor: 30,720 threads total





CUDA

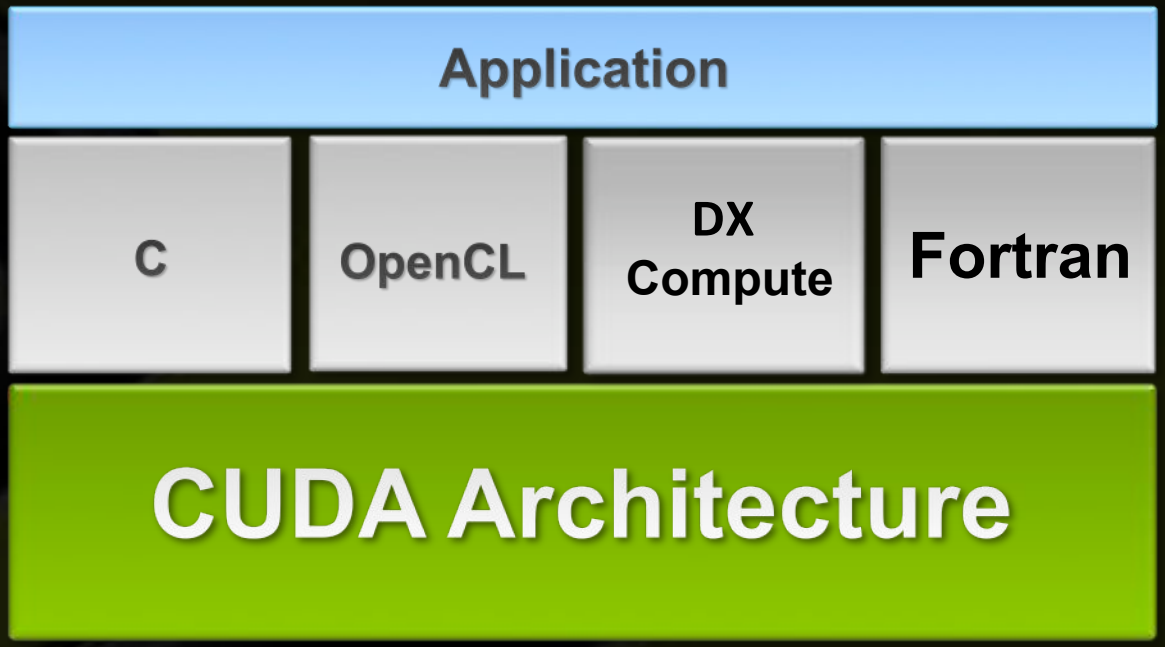
A Parallel Computing Architecture for NVIDIA GPUs

Supports standard languages and APIs

- C
- OpenCL
- DX Compute
- Fortran (PGI)

Supported on common operating systems:

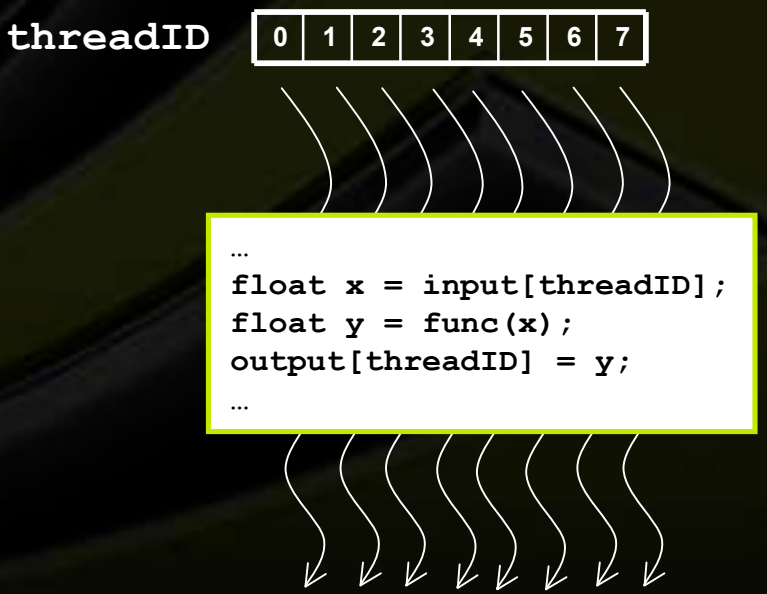
- Windows
- Mac OS X
- Linux



NVIDIA supports any initiative that unleashes the massive power of the GPU

CUDA Programming Model

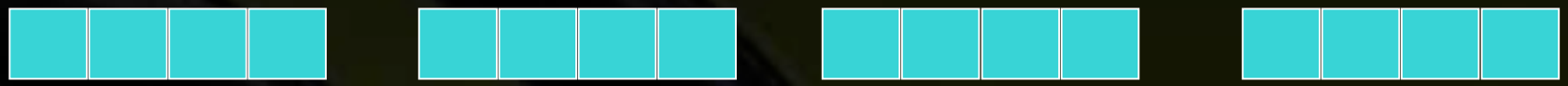
- Parallel code (kernel) is launched and executed on the GPU by many threads
- Parallel code is written for a thread
  - Each thread is free to execute a unique code path
  - Each thread has an ID that it uses to compute memory addresses and make control decisions
- Threads are grouped into thread blocks
- Threads in a block can co-operate



Example: Increment Array Elements

Increment N-element vector a by scalar b

Let's assume N=16, blockDim=4 -> 4 blocks



blockIdx.x=0
blockDim.x=4
threadIdx.x=0,1,2,3
idx=0,1,2,3

blockIdx.x=1
blockDim.x=4
threadIdx.x=0,1,2,3
idx=4,5,6,7

blockIdx.x=2
blockDim.x=4
threadIdx.x=0,1,2,3
idx=8,9,10,11

blockIdx.x=3
blockDim.x=4
threadIdx.x=0,1,2,3
idx=12,13,14,15

```
int idx = blockDim.x * blockId.x + threadIdx.x;
```

will map from local index threadIdx to global index

NB: blockDim should be >= 32 in real code, this is just an example

Example: Increment Array Elements

CPU program

```
void increment_cpu(float *a, float b, int N)
{

        for (int idx = 0; idx<N; idx++)
            a[idx] = a[idx] + b;
}


void main()
{
    .....
        increment_cpu(a, b, N);
}
```

C for CUDA program

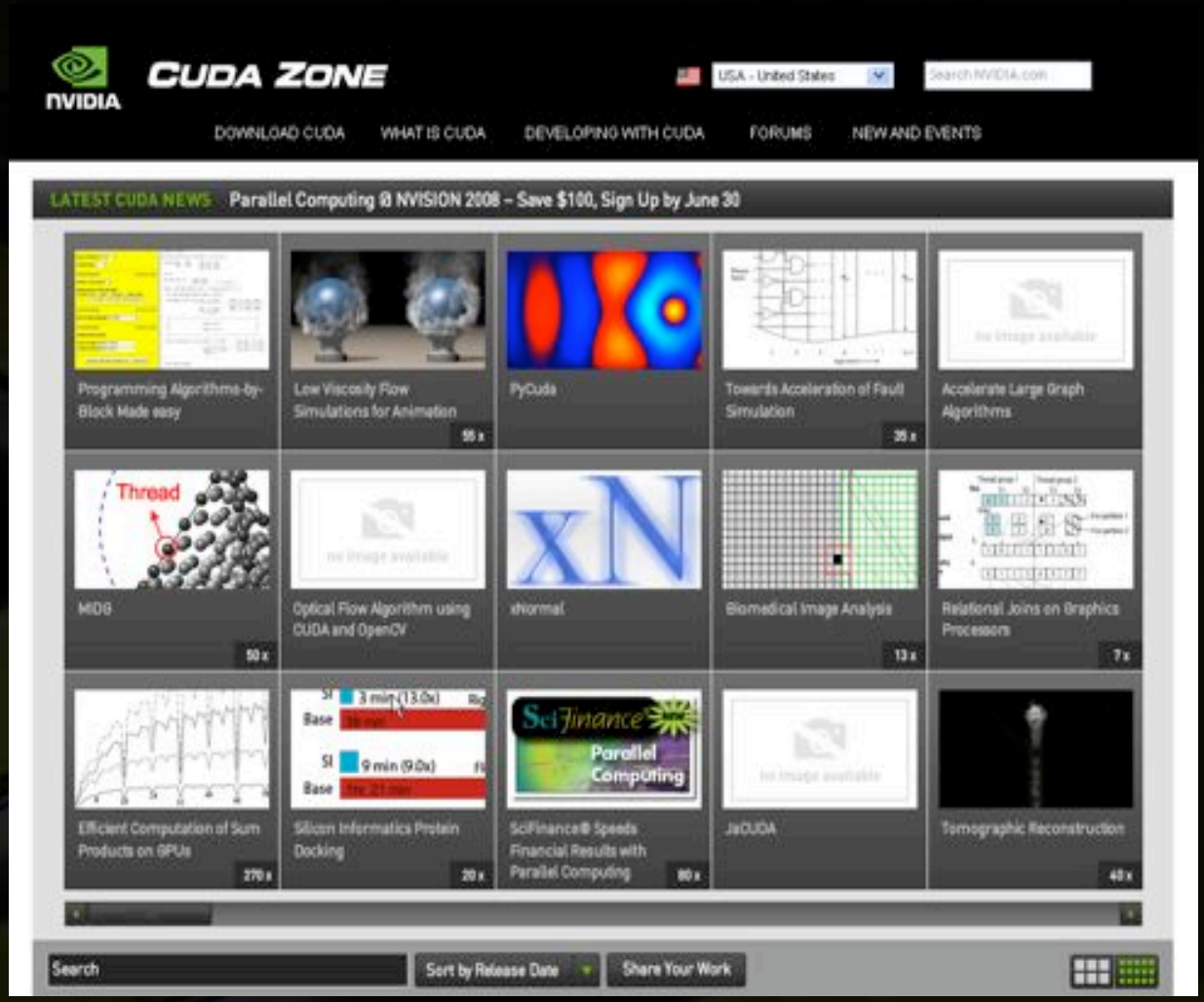
```
__global__ void increment_gpu(float *a, float b, int N)
{
        int idx = blockIdx.x * blockDim.x + threadIdx.x;
        if (idx < N)
            a[idx] = a[idx] + b;
}


void main()
{
    .....
        dim3 dimBlock (blocksize);
        dim3 dimGrid( ceil( N / (float)blocksize)  );
        increment_gpu<<<dimGrid, dimBlock>>>(ad,bd, N);
}
```

Pervasive Parallel Computing with CUDA

- CUDA brings data-parallel computing to the masses
  - Over 100M CUDA-capable GPUs deployed since Nov 2006
- Wide developer acceptance:
  - Download CUDA from www.nvidia.com/cuda
  - Over 60K CUDA developer downloads
  - Thousands of papers, presentations, codes
- Data-parallel supercomputers are everywhere!
  - CUDA makes this power readily accessible
  - Enables rapid innovations in data-parallel computing
- Parallel computing rides the commodity technology wave

CUDA Zone: www.nvidia.com/cuda



- Resources, examples, and pointers for CUDA developers

C for CUDA

Driver: required component to run CUDA applications

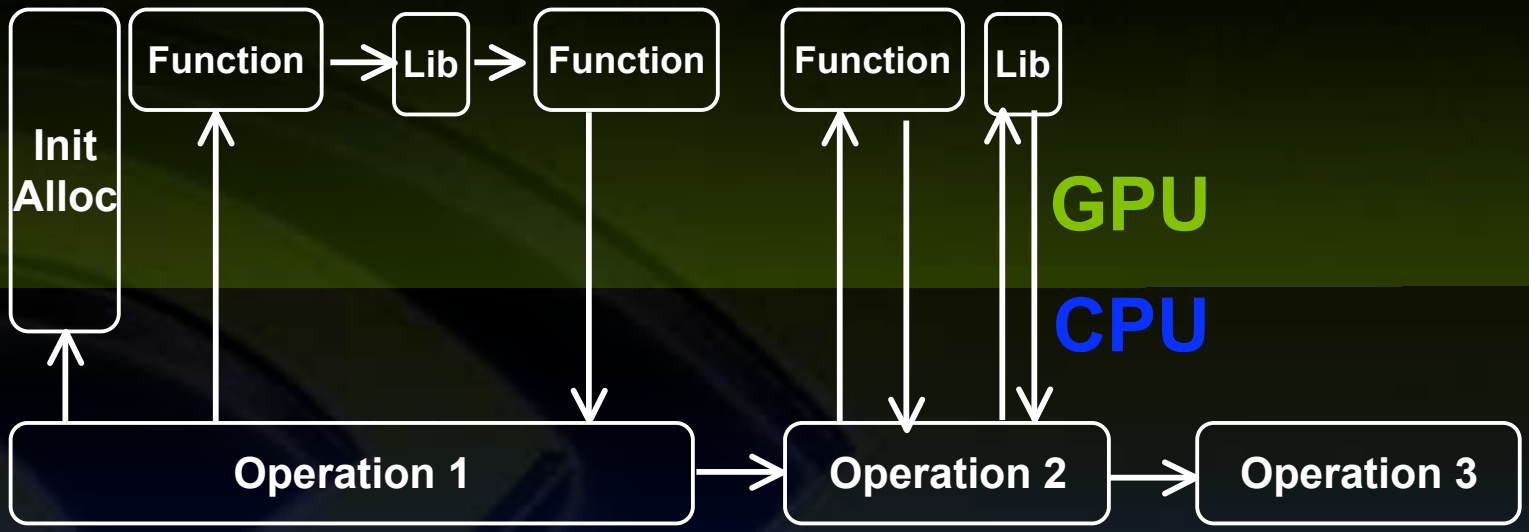
Toolkit: compiler, CUBLAS and CUFFT, profiler, debugger

(required for development)

SDK: collection of examples and documentation

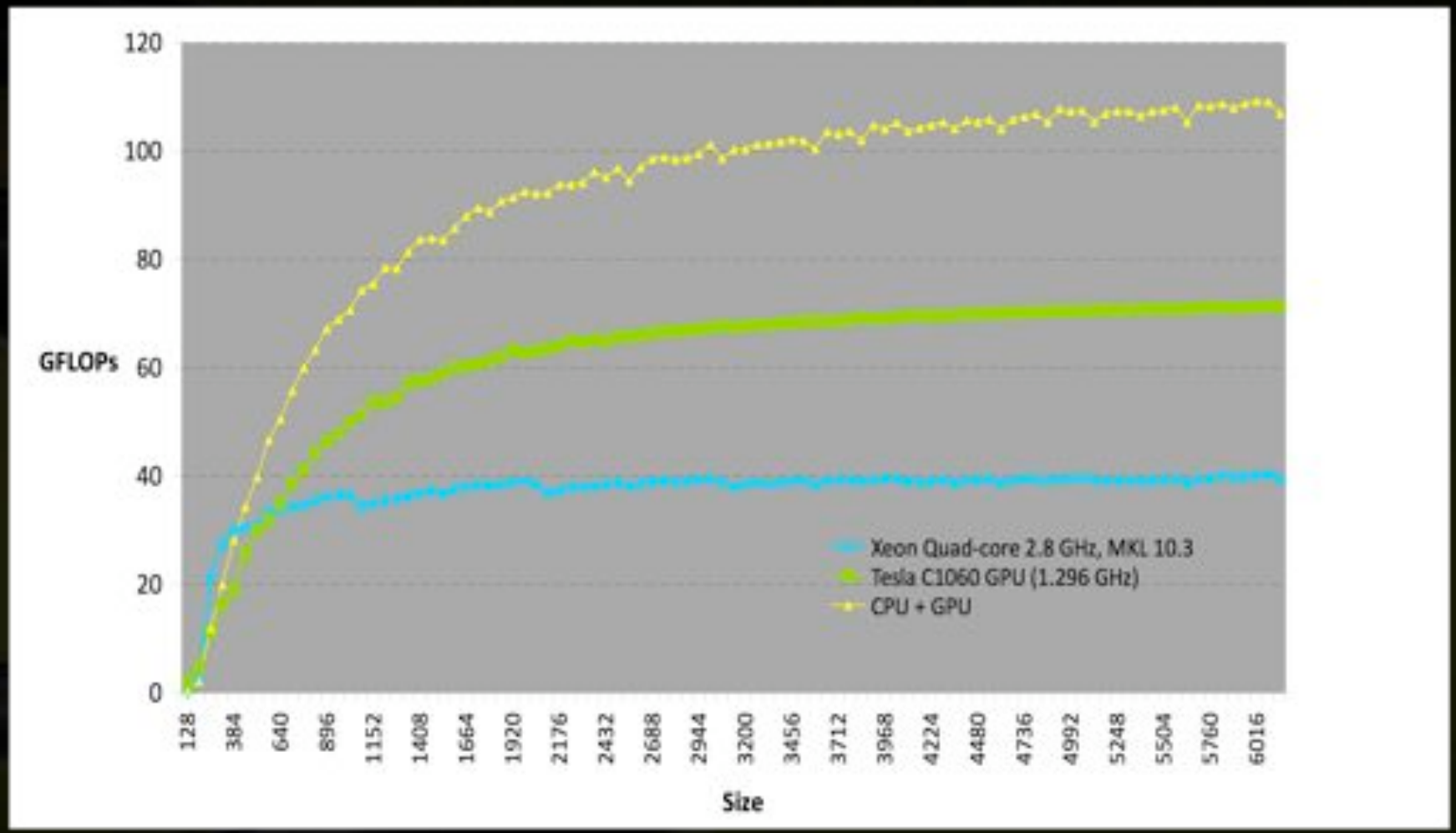
Support for Linux (32 and 64 bit), Windows XP and Vista (32 and 64 bit), MacOSX 10.5

Closely Coupled CPU-GPU

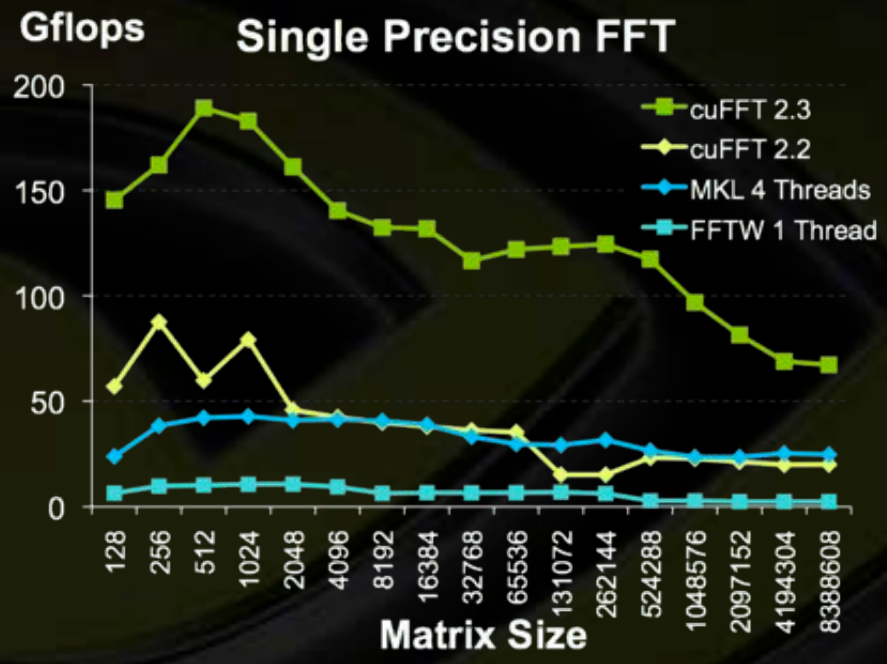


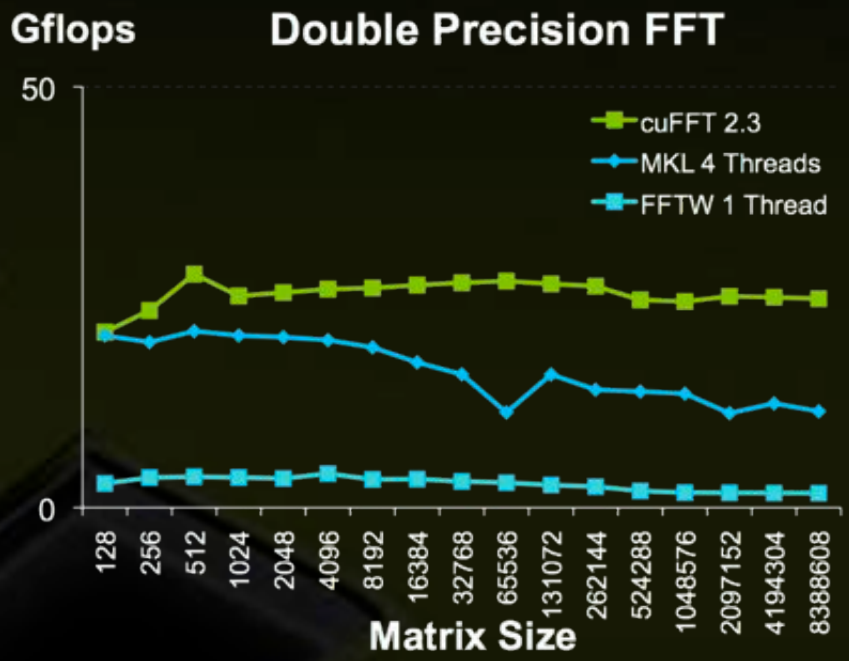
- Integrated programming model
- High speed data transfer – up to 5.5GB/sec
- Asynchronous data transfer, overlap
- Large GPU memory systems

DGEMM Performance (CUBLAS)



FFT Performance: CPU vs GPU





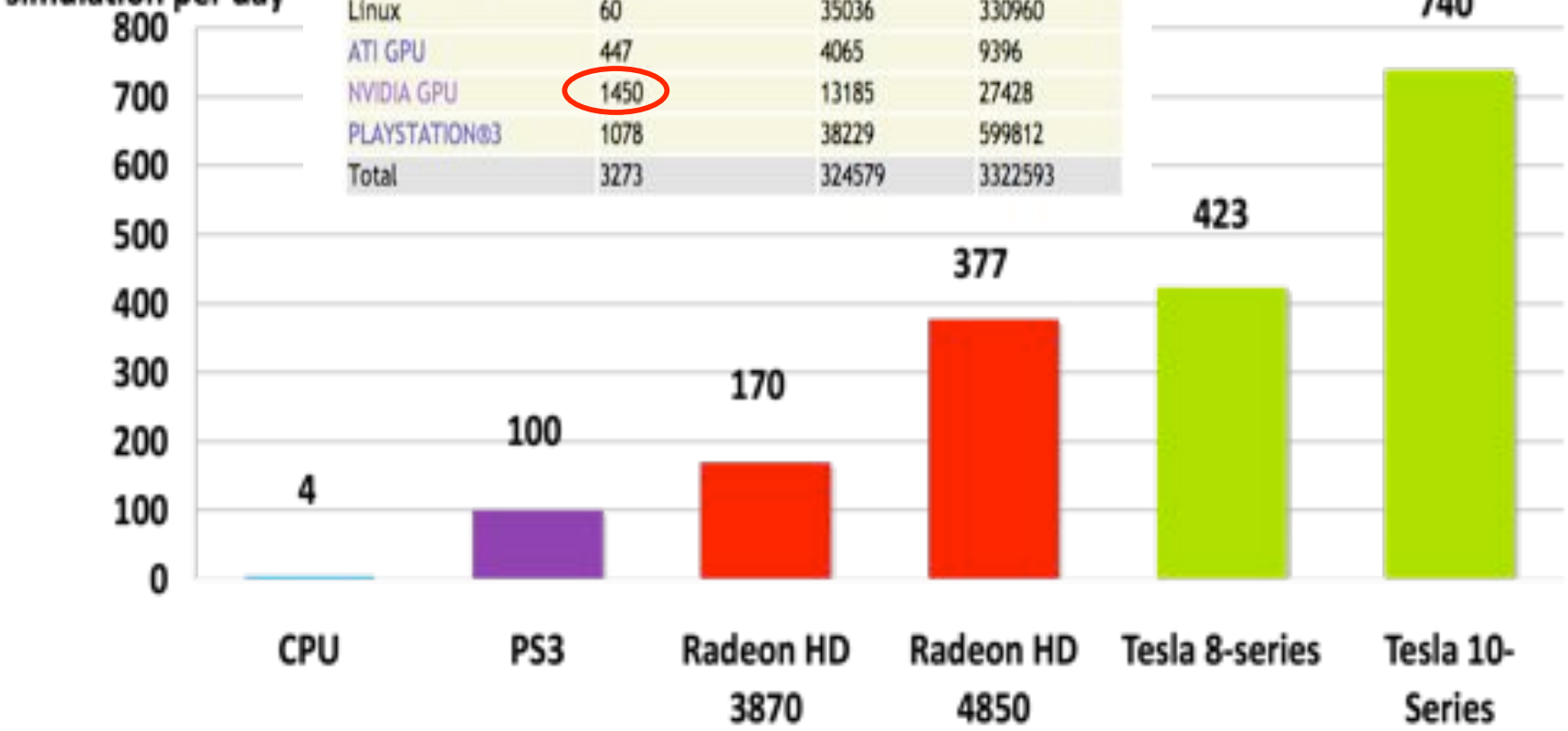
cuFFT 2.3 beta: NVIDIA Tesla C1060 GPU

MKL 10.1r1: Quad-Core Intel Core i7 (Nehalem) 3.2GHz


NVIDIA
Applications

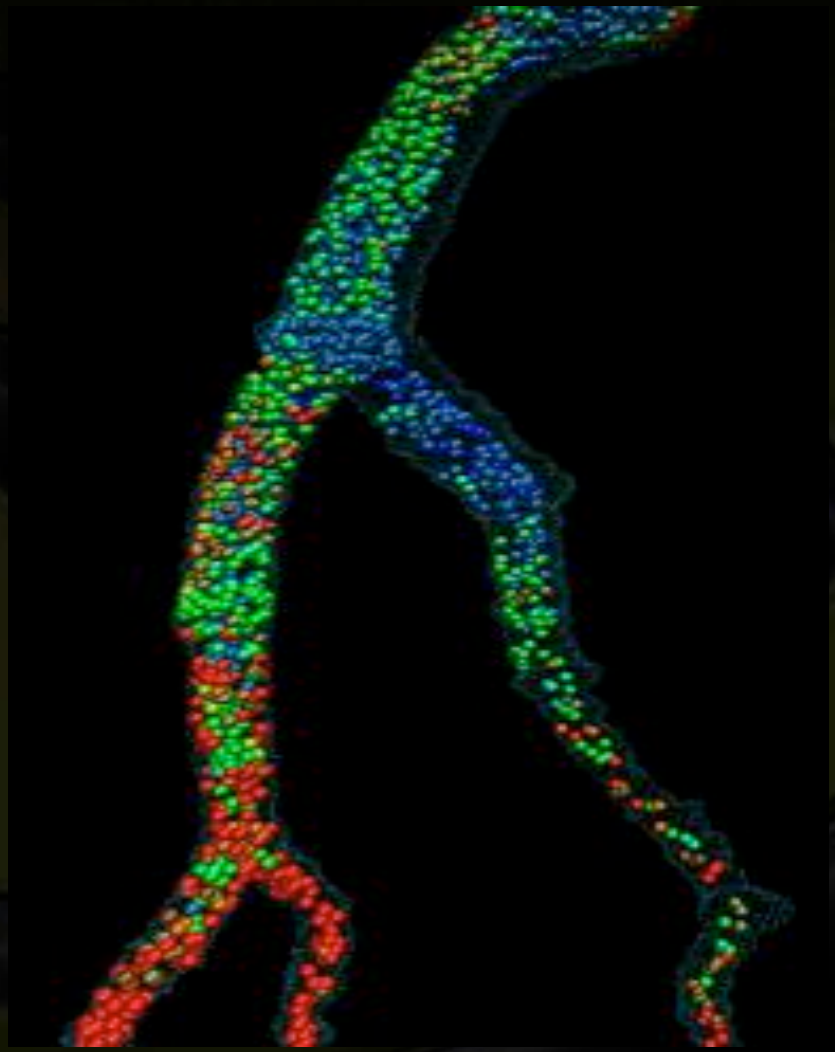
Folding@home Performance Comparison

OS Type	Current TFLOPS*	Active CPUs	Total CPUs
Windows	208	218169	2172051
Mac OS X/PowerPC	7	8563	120066
Mac OS X/Intel	23	7332	62880
Linux	60	35036	330960
ATI GPU	447	4065	9396
NVIDIA GPU	1450	13185	27428
PLAYSTATION®3	1078	38229	599812
Total	3273	324579	3322593



F@H kernel based on GROMACS code

Lattice Boltzmann



1000 iterations on a 256x128x128 domain

Cluster with 8 GPUs: 7.5 sec

Blue Gene/L 512 nodes: 21 sec

10000 iterations on irregular 1057x692x1446 domain with 4M fluid nodes

1 C870	**760 s**	**53 MLUPS**
2 C1060	**159 s**	**252 MLUPS**
8 C1060	**42 s**	**955 MLUPS**

Blood flow pattern in a human coronary artery, Bernaschi et al.

CUDA accelerated Linpack

Standard HPL code, with library that intercepts DGEMM and DTRSM calls and executes them simultaneously on the GPUs and CPU cores. Library is implemented with CUBLAS.

Cluster with 8 nodes:

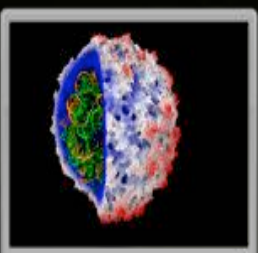
- -Each node has 2 Intel Xeon E5462 (2.8Ghz), 16GB of memory and 2 Tesla GPUs (1.44Ghz clock).
- -The nodes are connected with SDR Infiniband.

T/V	N	NB	P	Q	Time	Gflops
WR11R2L2	118144	960	4	4	874.26	1.258e+03
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=					0.0031157 ..... PASSED	

Applications in several fields

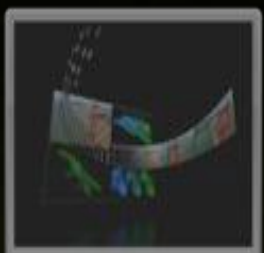
146X

Interactive visualization of volumetric white matter connectivity



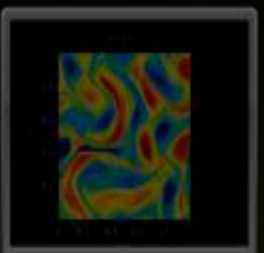
36X

Ionic placement for molecular dynamics simulation on GPU



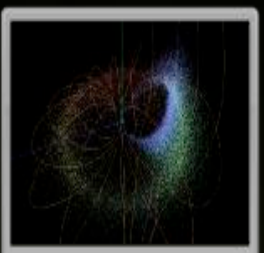
19X

Transcoding HD video stream to H.264



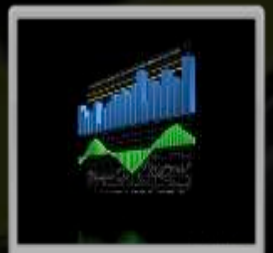
17X

Simulation in Matlab using .mex file CUDA function



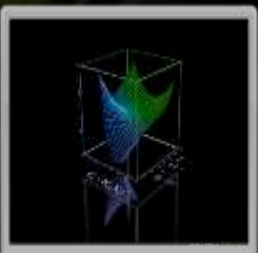
100X

Astrophysics N-body simulation



149X

Financial simulation of LIBOR model with swaptions



47X

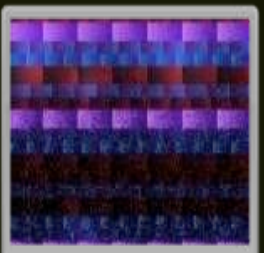
GLAME@lab: An M-script API for linear Algebra operations on GPU

20X

Ultrasound medical imaging for cancer diagnostics

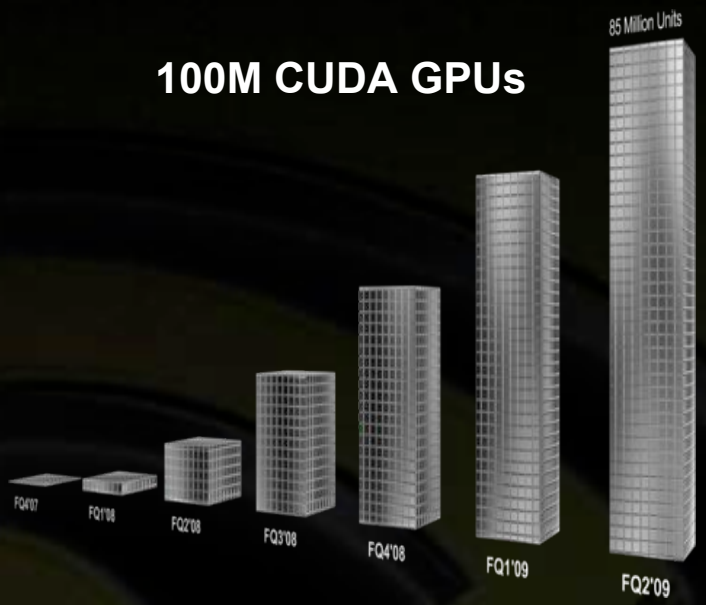
24X

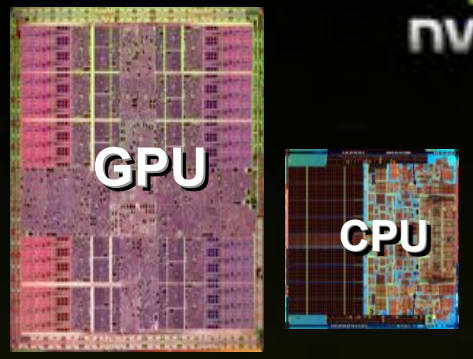
Highly optimized object oriented molecular dynamics



30X

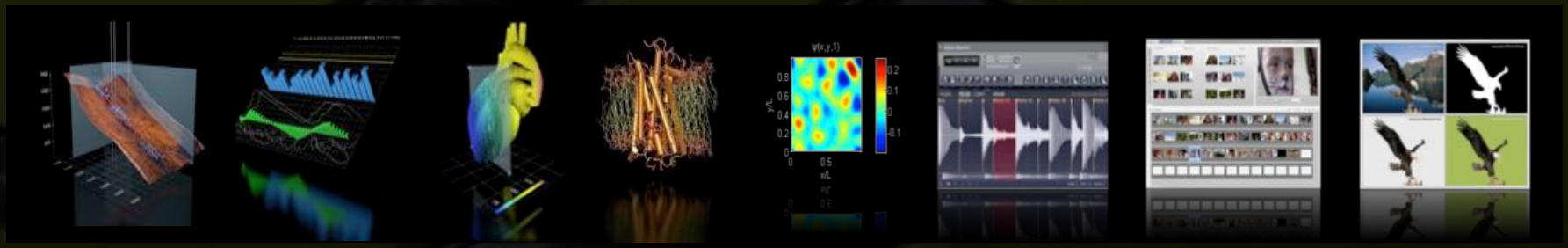
Cmatch exact string matching to find similar proteins and gene sequences





Heterogeneous Computing

CUDA



Oil & Gas | Finance | Medical | Biophysics | Numerics | Audio | Video | Imaging