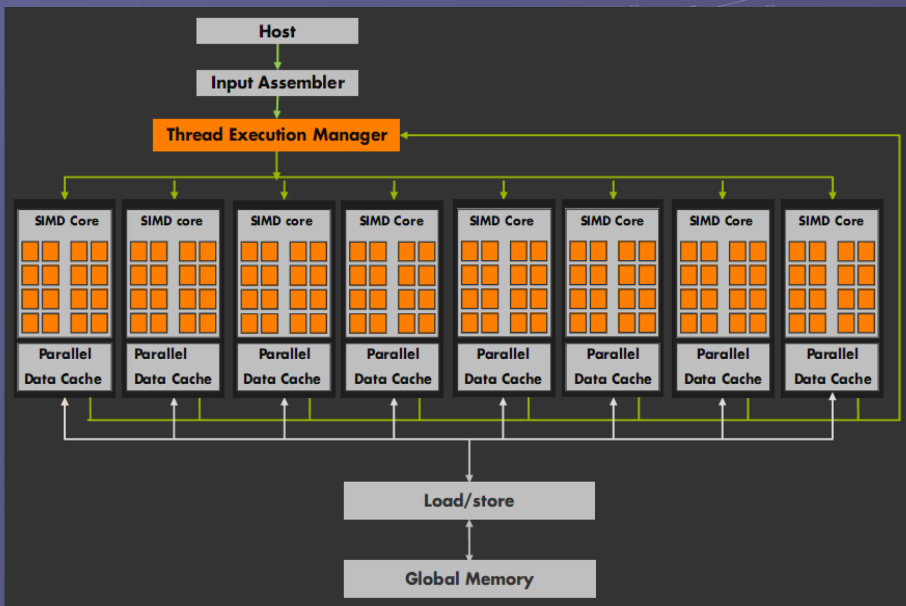
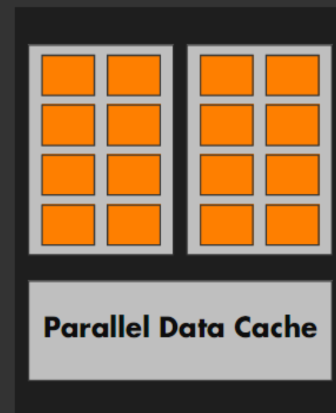
More on GPU

Graphics Processors (GPU) as General Purpose Supercomputers (GPGPU)

2008…
GeForce 9800 GTX, 128 Stream Proc., 512 MB
GeForce 9800 GX2, 256 Stream Proc., 1 GB
GeForce 9800 GT, 64 Stream Proc., 512 MB
[...]
2009: Tesla ~200 Proc., 4GB
2010: Fermi ~400 Proc., 4GB
2013: Kepler K20, ~2500 Procs., 6GB
2016: Kepler K80, ~5000 Procs.
2017/18: Pascal, Volta, Ampere > 5000 Procs., 40 GB

Hardware around 2006





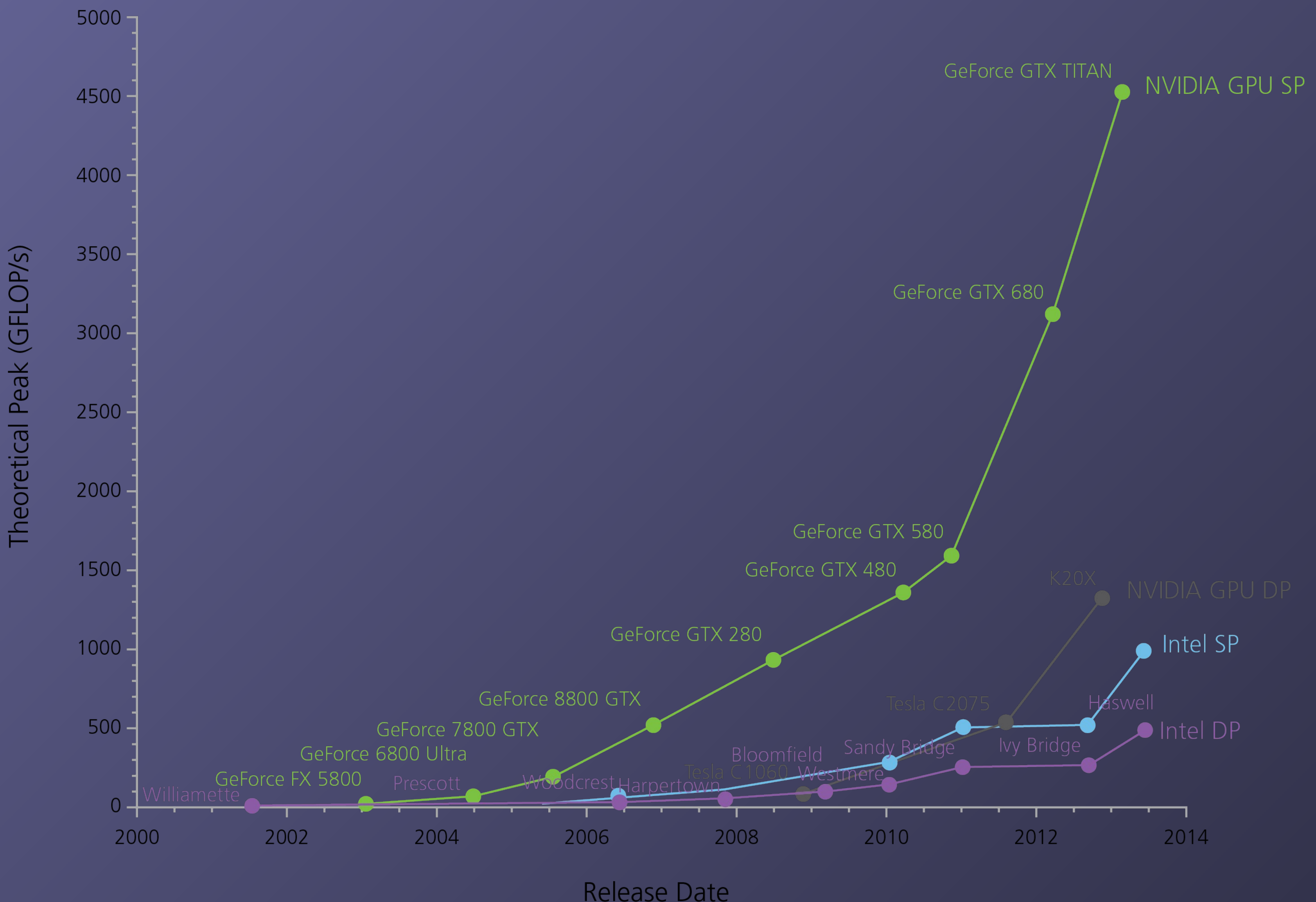
Each core
- 8 functional units
- SIMD 16/32 "warp"
- 8-10 stage pipeline
- Thread scheduler
- 128-512 threads/core
- 16 KB shared memory

Total #threads/chip
16 * 512 = 8K

GeForce 8800 GTX:

575 MHz * 128 processors * 2 flop/inst * 2 inst/clock = 333 Gflops

CPU vs. GPU speedup timeline

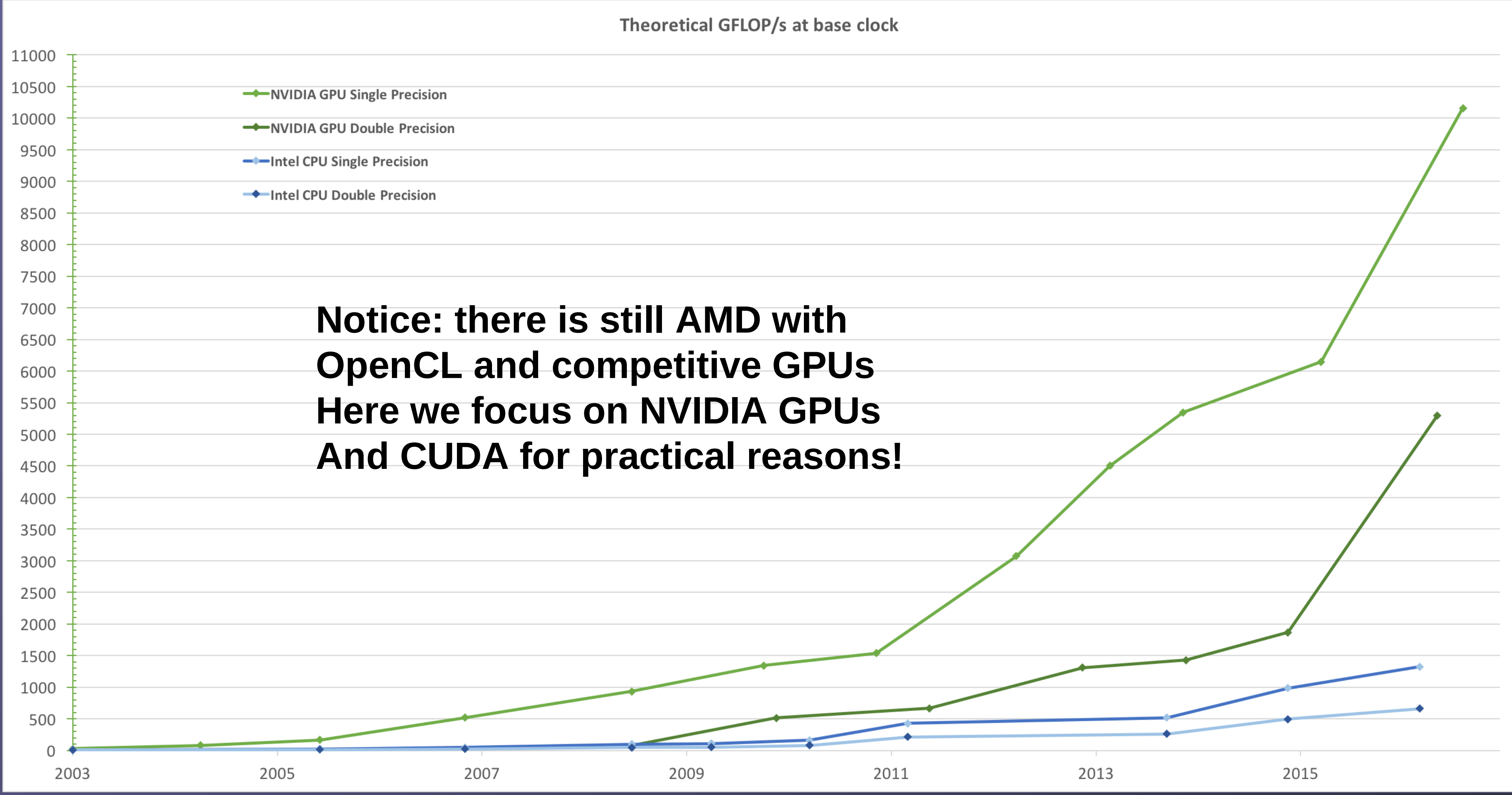


Source: https://www.carestream.com/blog/wp-content/uploads/2015/09/CSH_CPU-GPU_Chart.png

Floating Point Operations per Second for CPU and GPU:

From NVIDIA CUDA Developer Zone at:

http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

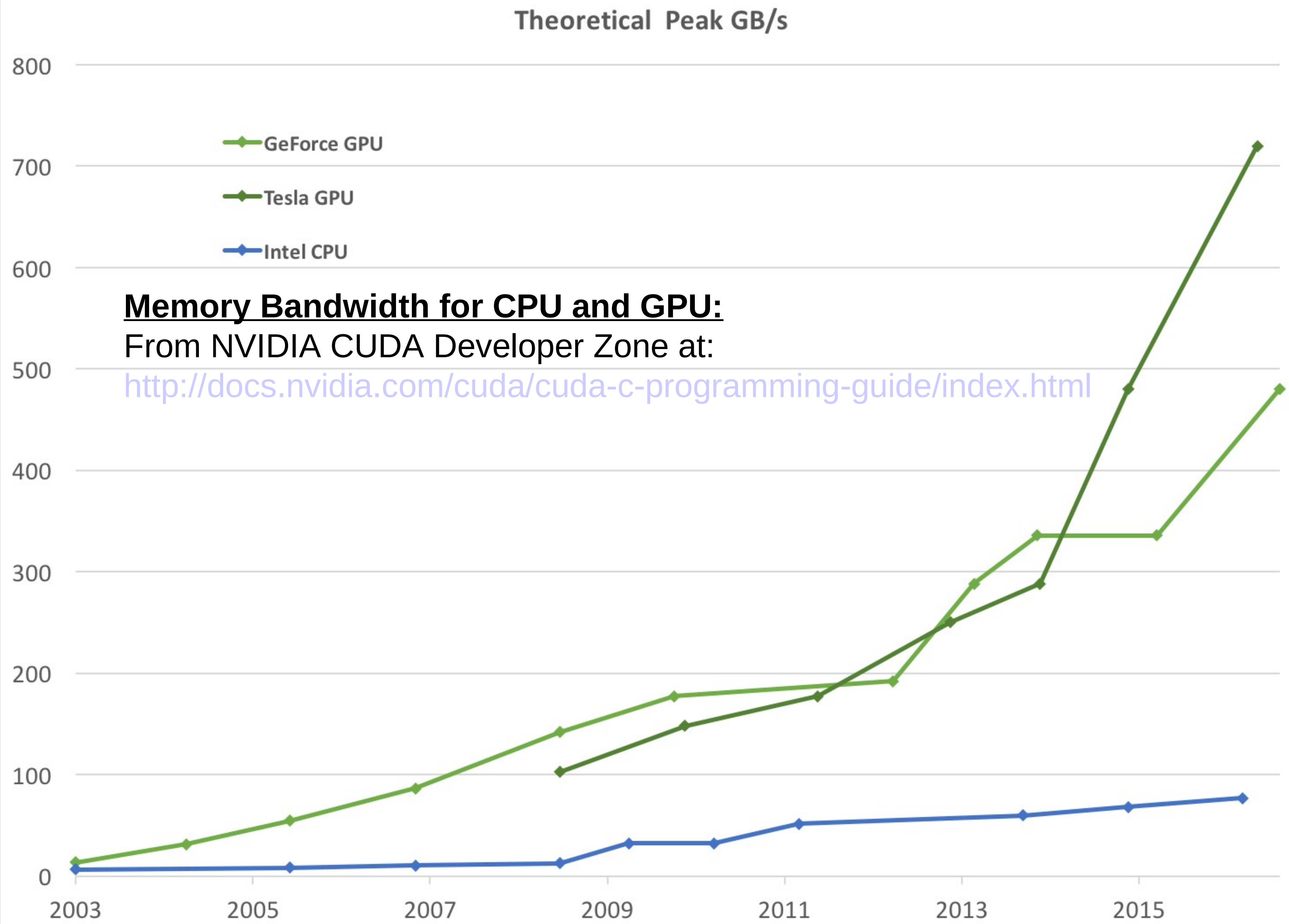


Notice: there is still AMD with OpenCL and competitive GPUs Here we focus on NVIDIA GPUs And CUDA for practical reasons!

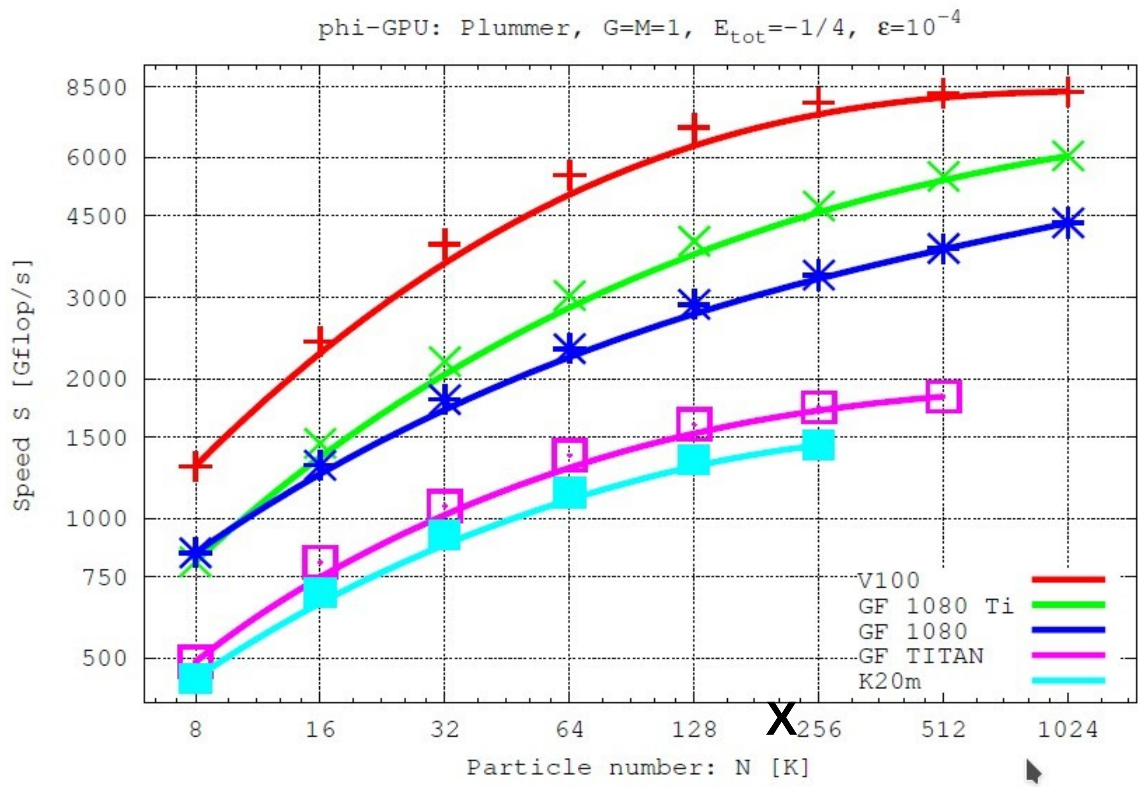
<u>Memory Bandwidth for CPU and GPU:</u>

From NVIDIA CUDA Developer Zone at:

http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html



Kepler, Pascal, Volta, Scaling, it works...



Volta V100

Pascal GF1080

Kepler K20m

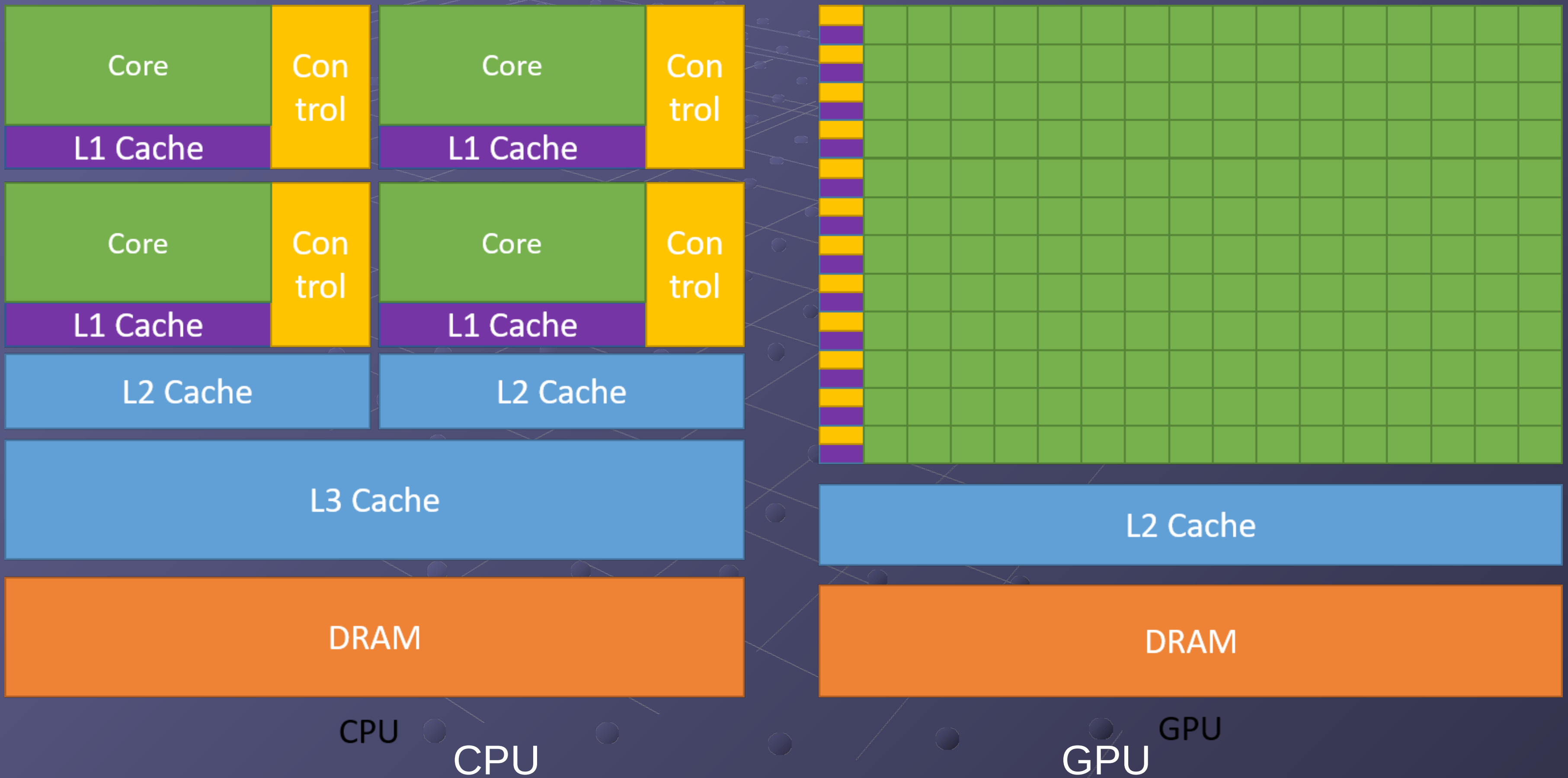
Spurzem, Berczik, et al., 2013, LNCS Supercomputing, 2013, pp. 13-25, Springer. (updated unpublished)

Fig. 4. Here we report a preliminary result from a benchmark test of our code on one Kepler K20 card; we compare with the performance on Fermi C2050 (used in the Mole-8.5 cluster), and the oldest Tesla C1060 GPU (used in the laohu cluster of 2009) - the latter is used as a normalization reference. We plot the speed ratio of our usual benchmarking simulation used in the previous figures, as a function of particle number. From this we see the sustained performance of a Kepler K20 would be about 1.4 - 1.5 Tflop/s.

X = first GPU of laohu 2010

CPU and GPU; from CUDA NVIDIA Developer Zone at

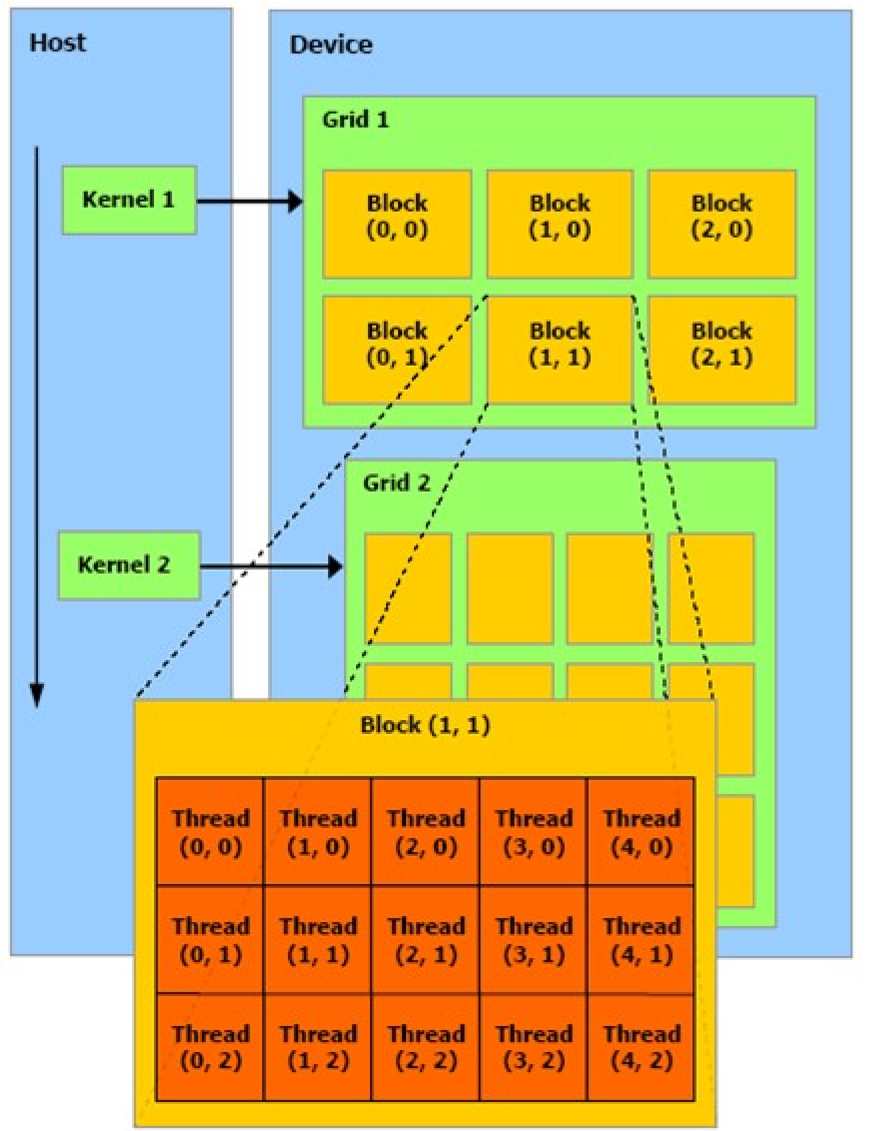
http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html



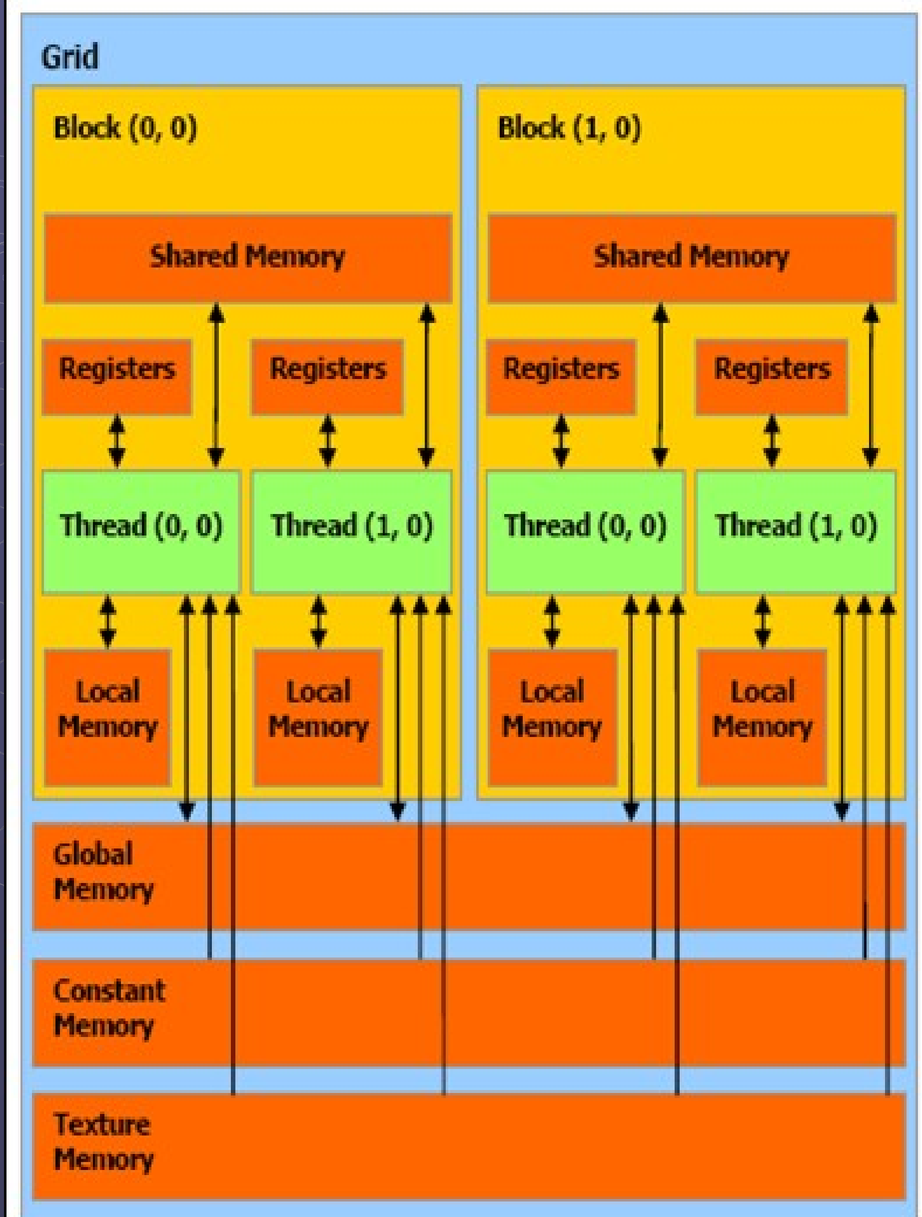
"The GPU devotes more transistors to computing"
"favours data parallel operations"

GPU Structure

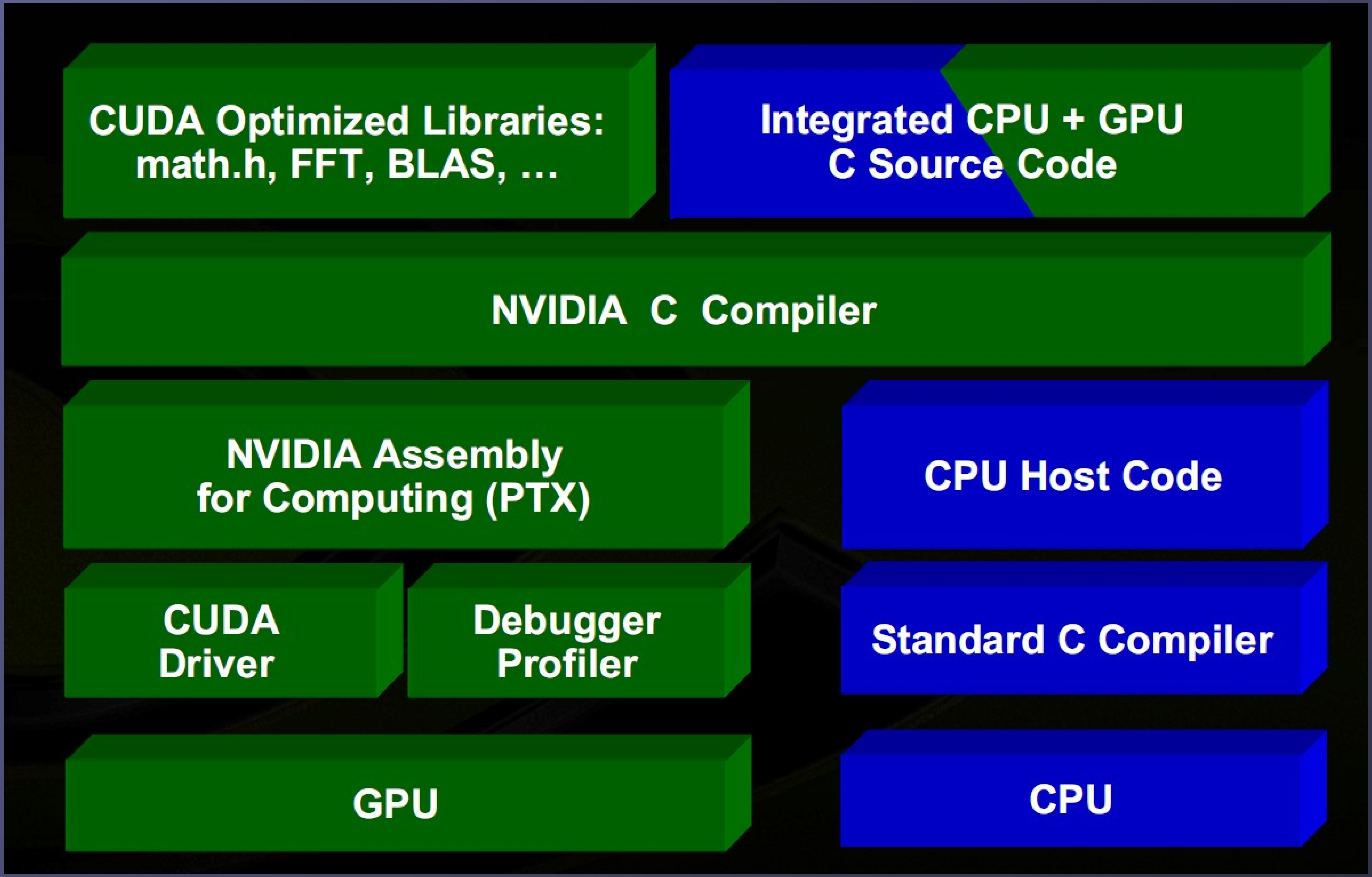
https://docs.nvidia.com/cuda/parallel-thread-execution/index.html



The host issues a succession of kernel invocations to the device. Each kernel is executed as a batch of threads organized as a grid of thread blocks



CUDA



CUDA Optimized Libraries:
math.h, FFT, BLAS, …
Integrated CPU + GPU
C Source Code
NVIDIA C Compiler
NVIDIA Assembly
for Computing (PTX)
CPU Host Code
CUDA
Driver
Debugger
Profiler
Standard C Compiler
GPU
CPU

GPU Computing Applications

Libraries and Middleware

cuDNN TensorRT	cuFFT cuBLAS cuRAND cuSPARSE	CULA MAGMA	Thrust NPP	VSIPL SVM OpenCurrent	PhysX OptiX iRay	MATLAB Mathematica

Programming Languages

C	C++	Fortran	Java Python Wrappers	DirectCompute	Directives (e.g. OpenACC)

Ampere and Volta:
Tensor Cores/NVLink

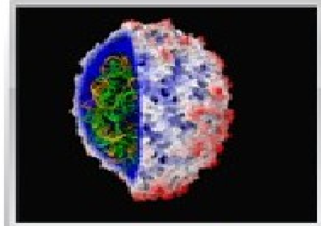
CUDA-Enabled NVIDIA GPUs

NVIDIA Ampere Architecture (compute capabilities 8.x)				Tesla A Series
NVIDIA Turing Architecture (compute capabilities 7.x)		GeForce 2000 Series	Quadro RTX Series	Tesla T Series
NVIDIA Volta Architecture (compute capabilities 7.x)	DRIVE/JETSON AGX Xavier		Quadro GV Series	Tesla V Series
NVIDIA Pascal Architecture (compute capabilities 6.x) Kepler (3.x)	Tegra X2 Tegra K1	GeForce 1000 Series GeForce 700/800	Quadro P Series Quadro K	Tesla P Series Tesla K
	 	 	 	

Speedups using GPU vs. CPU

146X

Interactive visualization of volumetric white matter connectivity[1]

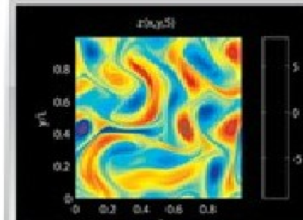


36X

Ionic placement for molecular dynamics simulation on GPU[2]

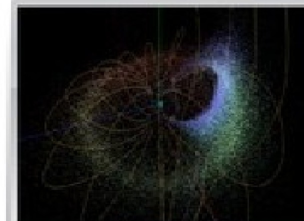
18X

Transcoding HD video stream to H.264 for portable video[3]



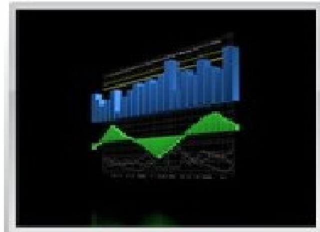
17X

Simulation in Matlab using .mex file CUDA function[4]



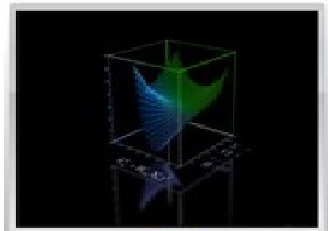
100X

Astrophysics N-body simulation[5]



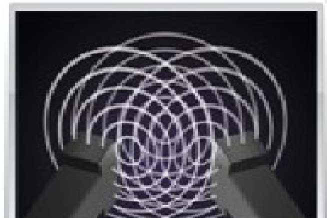
149X

Financial simulation of LIBOR model with swaptions[6]



47X

GLAME@lab: M-script API for linear Algebra operations on GPU[7]

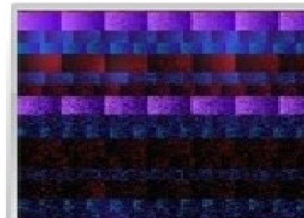


20X

Ultrasound medical imaging for cancer diagnostics[8]

24X

Highly optimized object oriented molecular dynamics[9]



30X

Cmatch exact string matching - find similar proteins & gene sequences[10]

Towards Peta-Scale Green Computation

— applications of the GPU supercomputers in CAS

http://www.nvidia.com/gtc2010-content

GPU TECHNOLOGY CONFERENCE

GTC 2010 | Sept 20-23, 2010

San Jose Convention Center, San Jose, California

Watch the Keynote Recordings

- Algorithms & Numerical Techniques
- Astronomy & Astrophysics
- Audio Processing
- Cloud Computing
- Computational Fluid Dynamics
- Computer Graphics
- Computer Vision
- Databases & Data Mining
- Digital Content Creation
- Embedded & Automotive
- Energy Exploration
- Film
- Finance
- General Interest
- GPU Accelerated Internet
- High Performance Computing
- Imaging
- Life Sciences
- Machine Learning & Artificial Intelligence
- Medical Imaging & Visualization
- Mobile & Tablet & Phone
- Molecular Dynamics
- Neuroscience
- Physics Simulation
- Programming Languages & Techniques
- Quantum Chemistry
- Ray Tracing
- Signal Processing
- Stereoscopic 3D
- Tools & Libraries
- Video Processing

Wei Ge
Xiaowei Wang
Inst. of Proc. Eng.

Yunquan Zhang
Inst. of Software

Rainer Spurzem
Nat. Astro. Obs. Chn.

Long Wang
SC Center

Simple CUDA example

CPU C program

```
void addMatrix(float *a, float *b,
               float *c, int N)
{
  int i, j, index;
  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
      index = i + j * N;
      c[index]=a[index] + b[index];
    }
  }
}

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

CUDA C program

```
__global__ void addMatrix(float *a,float *b,
                          float *c, int N)
{
  int i=blockIdx.x*blockDim.x+threadIdx.x;
  int j=blockIdx.y*blockDim.y+threadIdx.y;
  int index = i + j * N;
  if ( i < N && j < N)
    c[index]= a[index] + b[index];
}

void main()
{
  ..... // allocate & transfer data to GPU
  dim3 dimBlk (blocksize, blocksize);
  dim3 dimGrd (N/dimBlk.x, N/dimBlk.y);
  addMatrix<<<dimGrd,dimBlk>>>(a, b, c,N);
}
```