General Purpose Programming on Modern Graphics Processors and Accelerators

(a guest lecture for HPC 1 by L. Shawn Matott, Ph.D.)

November 2013

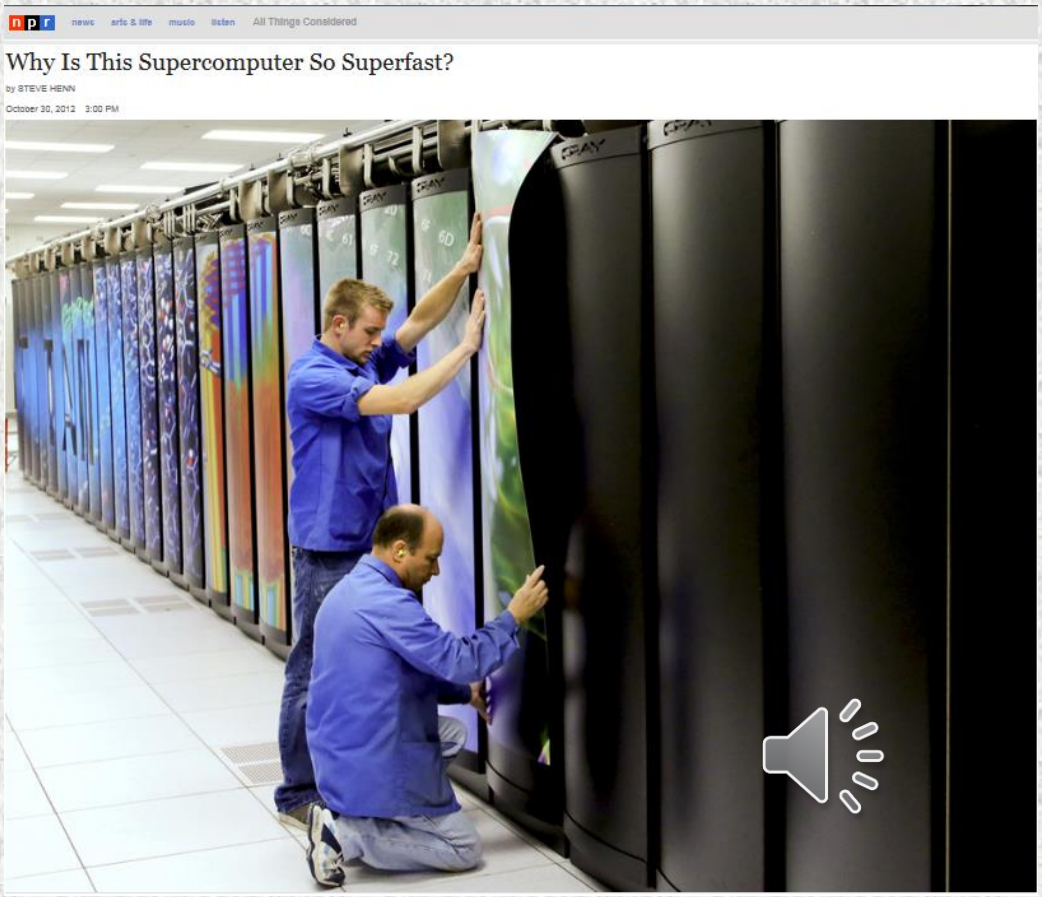
CENTER FOR COMPUTATIONAL RESEARCH
University at Buffalo The State University of New York

Overview

- History of GP GPU
- GPU Hardware
- GPU Software
- GPU at UB CCR
  - GPU Resources
  - GPU Programming
- GPU Performance
- Future Trends

GPU in the News

- From 10/30/2012 NPR "All Things Considered" Broadcast



Recent supercomputing gains largely attributed to graphics devices.

What is a GPU?

❑GPU – Graphics Processing Unit

- ❑ Special-purpose *co-processors (accelerators)* for high-end graphics
  - ❑video games, medical imaging, surveillance
  - ❑Can also be used for general-purpose computation (GP GPU)!
- ❑ Modern GPU architectures are inherently parallel and multi-threaded

Timeline of GPU Development

- 1999 – NVIDIA introduces GeForce 256
  - Transform & Lighting, Cube Environment Mapping
  - Limited Programmability via OpenGL extensions
- 2000 – MS releases DirectX 8
  - Full GPU programmability via ‘shaders’
  - Shader version defines programmability
- 2001 – NVIDIA introduces GeForce 3
- 2001 – GPU used for CFD (Navier-Stokes)
- 2002 – www.GPGPU.org is founded

Timeline of GPU Development

- 2002 – ATI introduces Radeon 8500
- 2002-2006 – ATI vs. NVIDIA
- July 2006 – AMD acquires ATI
- Nov. 2006 – MS releases DirectX 10
  - shader 4.0 – unified shader model
  - shared instruction set for vertex, pixel, and geometry shaders
- 2007 – NVIDIA introduces CUDA
  - Specifically for GP GPU computing

Timeline of GPU Development

- 2006 – 2009 – AMD GP GPU efforts
  - Close to Metal, ATI Stream SDK, OpenCL
- 2009 – MS releases DirectCompute API
- 2009 – NAMD GPU support
- 2010 – MATLAB GPU support
- 2011 – Intel demos MIC architecture
- 2012 – ORNL Debuts Titan Supercomputer
- 2012 – Accelerator APIs (GPU, APU, PHI/MIC, etc.)
  - C++ AMP (Microsoft) ; OpenACC (NVIDIA)

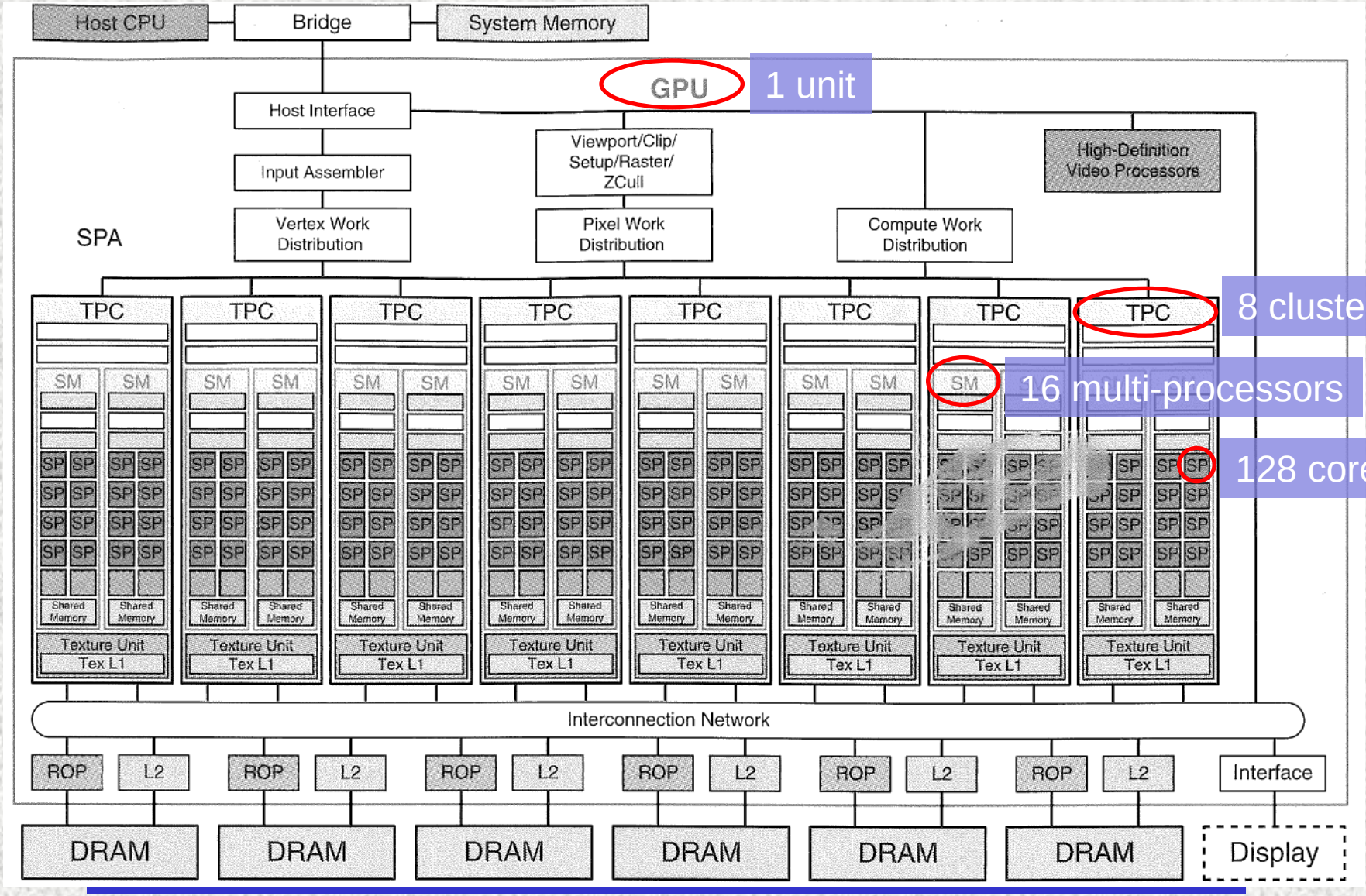
Timeline of GPU Development

- 2013 – Stampede (9.6PF), Tianhe-2 (34 PF)
  - Using Intel Xeon/Phi processor/accelerator
- 2013 – OpenMP 4.0 API Specification
  - Support for accelerators, DSPs (digital signal processors), RTSs (real-time systems)

Overview

- History of GP GPU
- GPU Hardware
- GPU Software
- GPU at UB CCR
  - GPU Resources
  - GPU Programming
- GPU Performance
- Future Trends

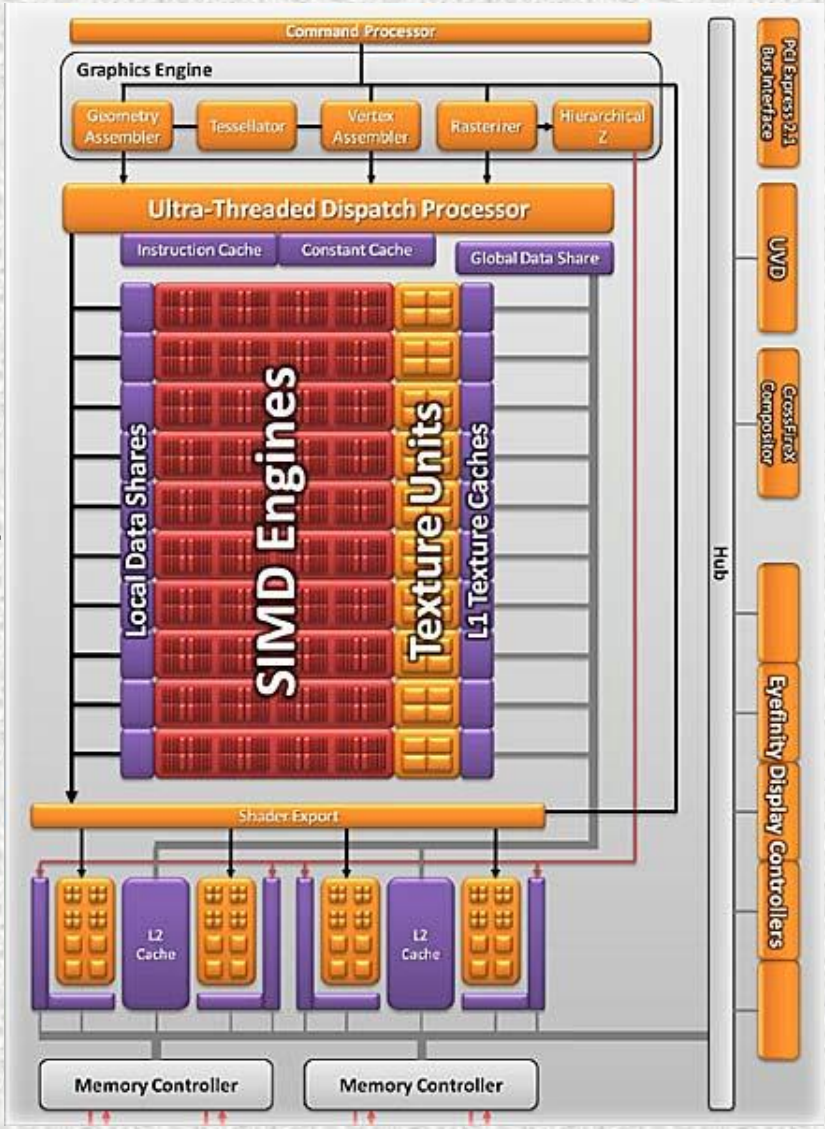
NVIDIA Hardware Architecture



Tesla GeForce 8800 GPU Architecture (Image from Patterson & Hennessy, 2009)

ATI Hardware Architecture

ATI Radeon HD5770 Juniper GPU Video Card

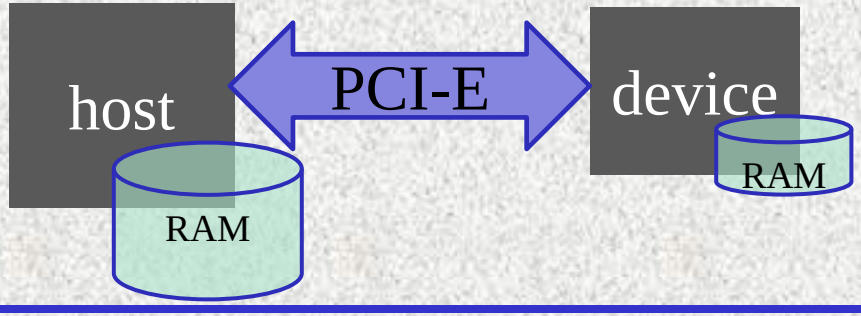


Overview

- History of GP GPU
- GPU Hardware
- GPU Software
- GPU at UB CCR
  - GPU Resources
  - GPU Programming
- GPU Performance
- Future Trends

GPU Software – CUDA

- CUDA - Compute Unified Device Architecture
  - host processor → the CPU
  - device (co-)processor → an NVIDIA GPU
  - Programs are launched on the host
    - specific operations offloaded to the device
  - Generally, host and device memory are completely separate and exclusive
    - Host and device communicate via PCI-express bus
    - Must stage data in/out of GPU memory
    - Newer hardware supports pinned host memory



GPU Software – CUDA

- CUDA C
  - NVIDIA extensions to the C-language
  - nvcc – NVIDIA's compiler for CUDA C
  - New modifiers to support CUDA programming

    `__global__` ← a function called from host, but runs on device

    `__device__` ← a function called from device and runs on device

    `__shared__` ← a device variable is shared among device threads

    `__constant__` ← a device variable is constant
  - GPU memory management functions

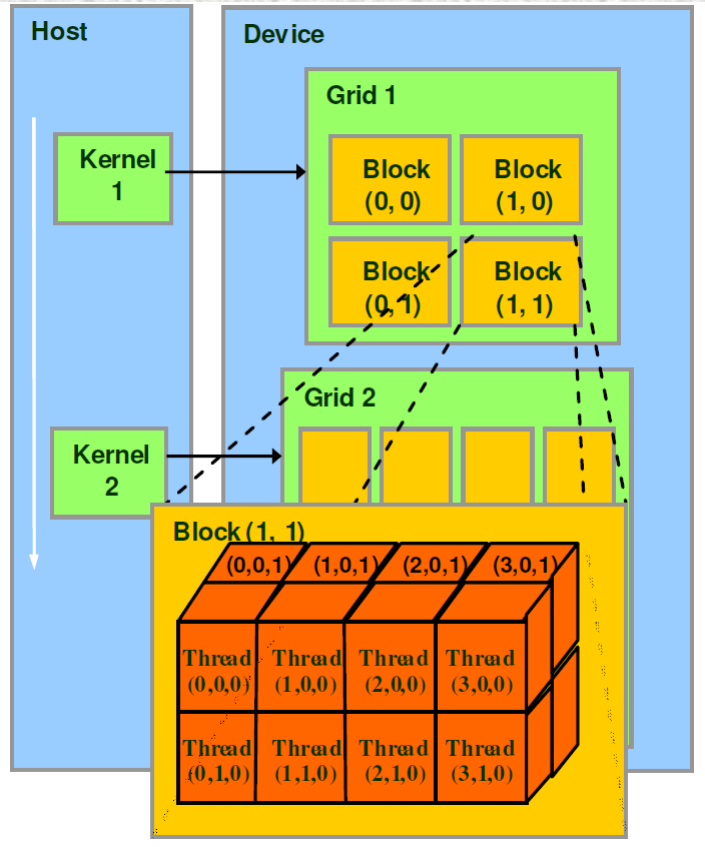
    `cudaMalloc()` ← allocate device memory

    `cudaMemcpy()` ← copy memory to/from device

    `cudaFree()` ← free device memory

GPU Software – CUDA

- CUDA Grids, Blocks, and Threads
  - Grids
    - Collection of independent GPU tasks
    - Each grid can run completely different codes (kernels)
    - Grids are subdivided into blocks
  - Blocks
    - Blocks run in parallel
    - Blocks are subdivided into threads
  - Threads
    - Collection of cooperating tasks
    - All threads in a given block can share memory
  - Blocks and Threads are assigned unique ids (`blockIdx` and `threadIdx`)
    - analogous to MPI rank



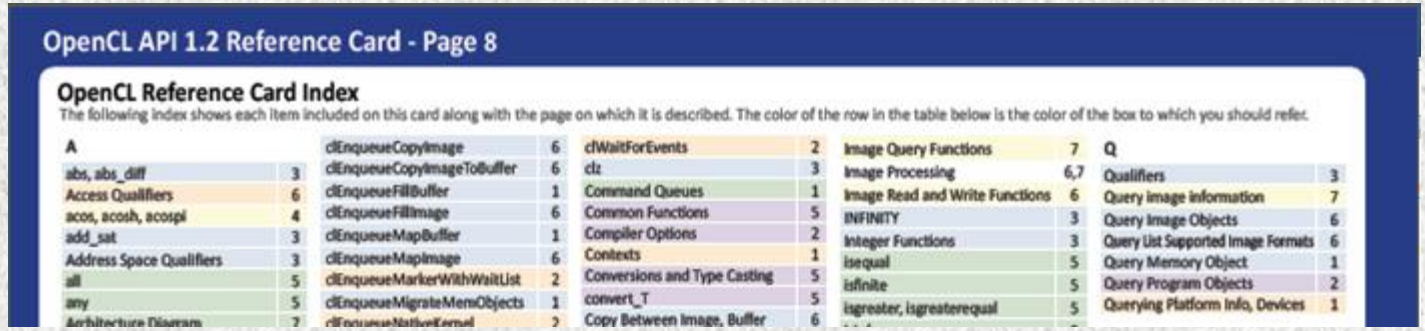
GPU Software – CUDA

- Calling device routines from host
  - Define routine as __global__
  - Embellish calls with <<blocks,threads>>
    - Blocks ← specifies block structure (a 1-, 2- or 3D grid)
    - Threads ← number of threads per block

```
__global__ void dummyKernel( int a )
{
  /* do nothing */
}

int main (void)
{
   dummyKernel<<1,1>>(5);
}
```

Other GPU Software

- OpenCL (Open Computing Language)
  - Currently at version 1.2 (w/ 2.0 in provisional review)
  - Open standard for programming GP GPUs
    - Portable across hardware vendors
    - About 400 functions, including math and vector intrinsics
  - Specs available from www.khronos.org/opencl


OpenCL API 1.2 Reference Card - Page 8

OpenCL Reference Card Index

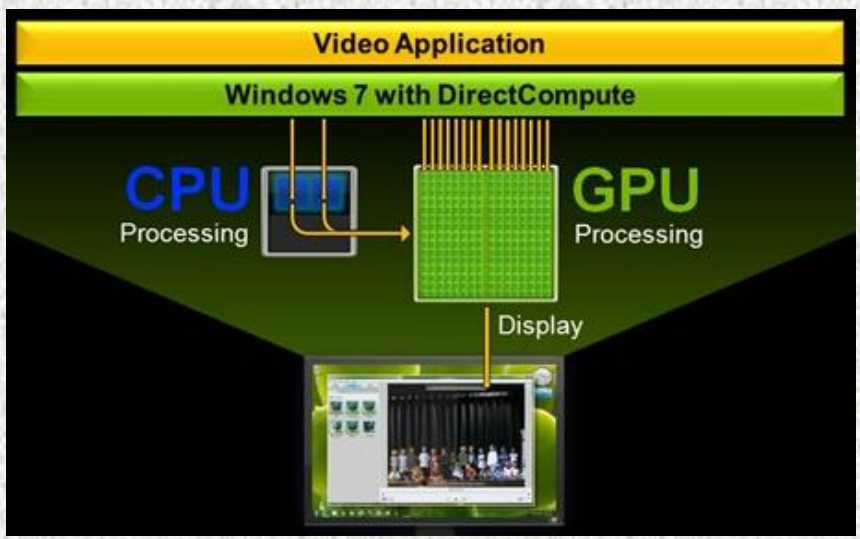
The following index shows each item included on this card along with the page on which it is described. The color of the row in the table below is the color of the box to which you should refer.

Training available at SC'13

8:30AM - 5:00PM	Tutorials	OpenCL: A Hands-On Introduction

Other GPU Software

- DirectCompute
  - Part of MS DirectX 11+
  - Introduces a general-purpose "compute" shader
  - Based on HLSL
    - High Level Shading Language
    - Similar to C/C++ but MS proprietary
    - Familiar to DirectX developers, not others
  - Tied to Windows OS



Other GPU Software

- C++ AMP
  - AMP = Accelerated Massive Parallelism
  - Adds a “restrict” language feature – compiler will check functions for GPU compatibility.
  - Adds STL-like “concurrency” namespace
  - Provides “parallel_for_each” function to support parallel for loops.
  - Currently tied to Windows OS, but some efforts to port to Linux via OpenCL
  - Debugger and profiler support within MS Visual Studio IDE

```
void AddArrays(int n, int m, int * pA, int * pB, int * pSum) {
    concurrency::array_view<int,2> a(n, m, pA), b(n, m, pB), sum(n, m, pSum);
    concurrency::parallel_for_each(sum.extent, [=](concurrency::index<2> i) restrict(amp)
    {
          sum[i] = a[i] + b[i];
    });
}
```

Other GPU Software

- OpenACC
  - NVIDIAs pragma-based approach to programming accelerators (i.e. GPU, MIC, etc.)
  - Similar to OpenMP approach
    - Plans in place to merge with OpenMP standard
  - Provides a few API functions as well:
    - acc_init(), acc_shutdown(), acc_get_device_num(), etc.
  - Uses a "gangs of workers" concept to organize work on the target device.
  - A partial list of pragmas:

```
#pragma acc parallel
#pragma acc kernels
#pragma acc data
#pragma acc loop
#pragma acc cache
#pragma acc update
#pragma acc declare
#pragma acc wait
```

Training available at SC'13

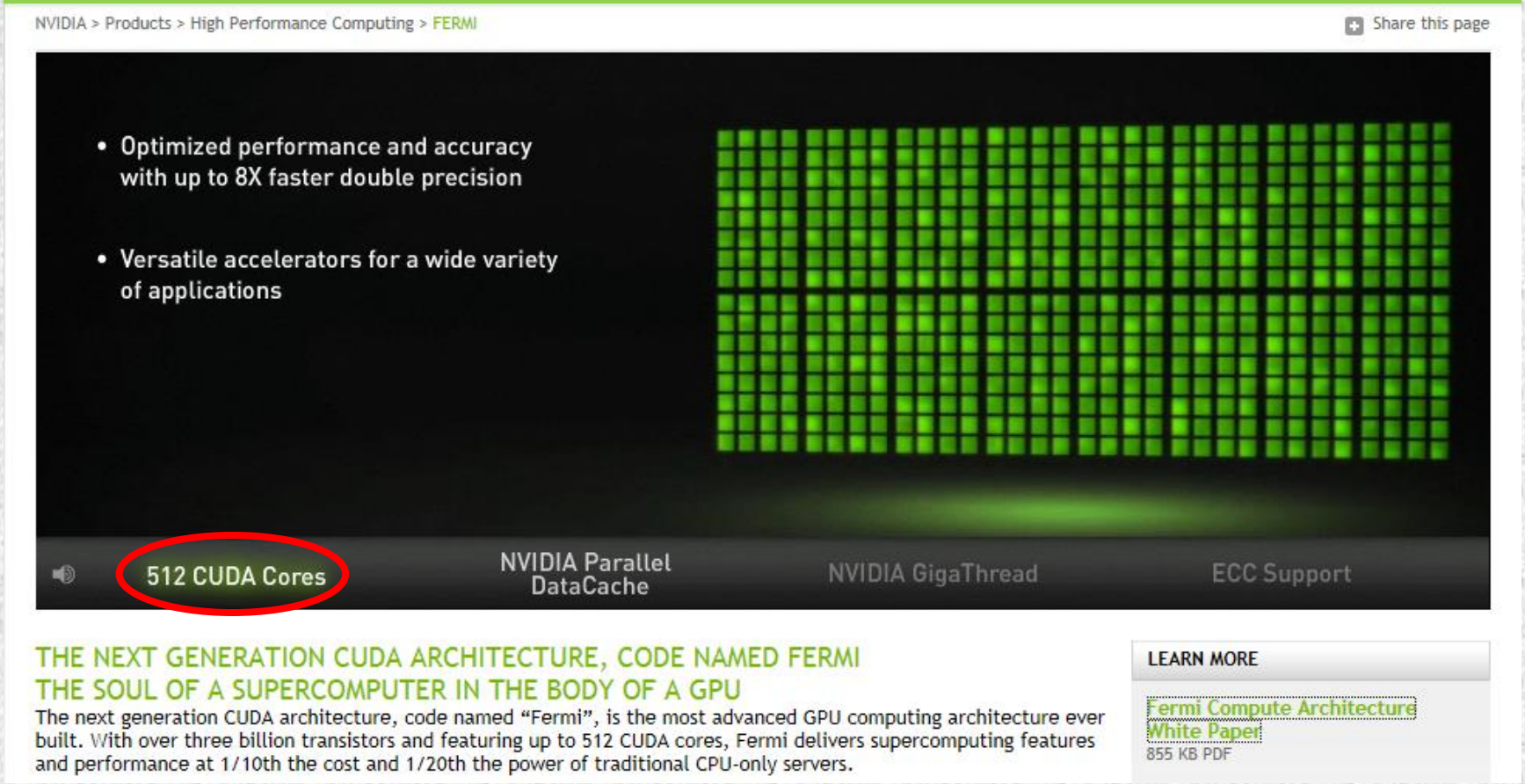
8:30AM - 5:00PM	Tutorials	OpenACC: Productive, Portable Performance on Hybrid Systems Using High-Level Compilers and Tools

Overview

- History of GP GPU
- GPU Hardware
- GPU Software
- GPU at UB CCR
  - GPU Resources
  - GPU Programming
- GPU Performance
- Future Trends

GP GPU at UB CCR

NVIDIA Fermi GPU Architecture (Image from www.nvidia.com, 2011)



- CCR GPU partition has 64 Fermi units → 32,768 cores!

GP GPU at UB CCR

- CUDA (NVIDIA GPU programming)
  - nvcc
  - Versions: 3.2, 4.0, 4.1, 4.2
- MATLAB
  - Latest version is R2013b, see /util/matlab/gpu-example
- NAMD (Not Another Molecular Dynamics)
  - Latest version is 2.9 (but 2.8-CUDA is faster for some)
- Python – pyCUDA module
- OpenCV – gpu module
- MAGMA – "LAPACK for multicore+GPU systems"

GP GPU at UB CCR

- Requesting GPU resources
  - Use --gres=gpu:1 or --gres=gpu:2 to request GPU resources for your SLURM job
    - `#SBATCH --gres=gpu:1` (in an SBATCH script)
    - `$fisbatch --gres=gpu:1` (in a fisbatch interactive job)
  - Use --partition=gpu instead of --partition=general-compute to obtain priority access to GPU nodes on the main general-compute partition
    - Gives significant priority boost over non-GPU jobs
    - You must actually use GPU resources! If not, your access to the GPU partition can be revoked by CCR admins.

GP GPU at UB CCR

```
[lsmatott@rush:~]$ snodes all gpu
HOSTNAMES  STATE    CPUS S:C:T    CPUS(A/I/O/T)     CPU_LOAD MEMORY  GRES     PARTITION
k05n11s01  alloc    12   2:6:1    12/0/0/12         7.61     48000   gpu:2    gpu
k05n11s02  alloc    12   2:6:1    12/0/0/12         12.06    48000   gpu:2    gpu
k05n12s01  alloc    12   2:6:1    12/0/0/12         12.06    48000   gpu:2    gpu
k05n12s02  alloc    12   2:6:1    12/0/0/12         12.00    48000   gpu:2    gpu
k05n20s01  alloc    12   2:6:1    12/0/0/12         12.06    48000   gpu:2    gpu
k05n20s02  alloc    12   2:6:1    12/0/0/12         12.00    48000   gpu:2    gpu
k05n21s01  alloc    12   2:6:1    12/0/0/12         12.01    48000   gpu:2    gpu
k05n21s02  alloc    12   2:6:1    12/0/0/12         12.02    48000   gpu:2    gpu
k05n30s01  alloc    12   2:6:1    12/0/0/12         12.16    48000   gpu:2    gpu
k05n30s02  alloc    12   2:6:1    12/0/0/12         12.05    48000   gpu:2    gpu
k05n31s01  alloc    12   2:6:1    12/0/0/12         12.00    48000   gpu:2    gpu
k05n31s02  alloc    12   2:6:1    12/0/0/12         12.02    48000   gpu:2    gpu
k05n39s01  alloc    12   2:6:1    12/0/0/12         12.03    48000   gpu:2    gpu
k05n39s02  alloc    12   2:6:1    12/0/0/12         12.01    48000   gpu:2    gpu
k05n40s01  alloc    12   2:6:1    12/0/0/12         12.01    48000   gpu:2    gpu
k05n40s02  down*    12   2:6:1    0/0/12/12         N/A      48000   gpu:2    gpu
k06n11s01  down*    12   2:6:1    0/0/12/12         N/A      48000   gpu:2    gpu
k06n11s02  alloc    12   2:6:1    12/0/0/12         12.00    48000   gpu:2    gpu
k06n12s01  drain    12   2:6:1    0/0/12/12         0.04     48000   gpu:2    gpu
k06n12s02  mix      12   2:6:1    8/4/0/12          1.56     48000   gpu:2    gpu
k06n20s01  alloc    12   2:6:1    12/0/0/12         12.07    48000   gpu:2    gpu
k06n20s02  drain    12   2:6:1    0/0/12/12         0.00     48000   gpu:2    gpu
k06n21s01  alloc    12   2:6:1    12/0/0/12         12.05    48000   gpu:2    gpu
k06n21s02  alloc    12   2:6:1    12/0/0/12         12.06    48000   gpu:2    gpu
k06n30s01  alloc    12   2:6:1    12/0/0/12         12.06    48000   gpu:2    gpu
k06n30s02  alloc    12   2:6:1    12/0/0/12         12.00    48000   gpu:2    gpu
k06n31s01  alloc    12   2:6:1    12/0/0/12         12.08    48000   gpu:2    gpu
k06n31s02  alloc    12   2:6:1    12/0/0/12         12.00    48000   gpu:2    gpu
k06n39s01  alloc    12   2:6:1    12/0/0/12         12.00    48000   gpu:2    gpu
k06n39s02  alloc    12   2:6:1    12/0/0/12         12.02    48000   gpu:2    gpu
k06n40s01  idle     12   2:6:1    0/12/0/12         0.00     48000   gpu:2    gpu
k06n40s02  drain*   12   2:6:1    0/0/12/12         N/A      48000   gpu:2    gpu
```

GP GPU at UB CCR

- Requesting GPU resources
  - One GPU node is also available in the debug partition (--partition=debug --gres=gpu:1[or 2]).
    - Useful for testing code before a 'production' run.

```
[lsmatott@rush:~]$ snodes all debug
HOSTNAMES   STATE    CPUS S:C:T    CPUS(A/I/O/T)    CPU_LOAD MEMORY   GRES      PARTITION
d07n33s01   mix      8    2:4:1    2/6/0/8          0.00     24000    (null)    debug
d07n33s02   drain    8    2:4:1    0/0/8/8          0.02     24000    (null)    debug
d16n02      alloc    8    2:4:1    8/0/0/8          0.00     24000    (null)    debug
d16n03      alloc    8    2:4:1    8/0/0/8          0.08     24000    (null)    debug
k05n26      mix      16   2:8:1    12/4/0/16        0.49     128000   gpu:2     debug
k08n41s01   idle     12   2:6:1    0/12/0/12        0.01     48000    (null)    debug
k08n41s02   idle     12   2:6:1    0/12/0/12        0.09     48000    (null)    debug
```

GPU Programming

❑ Fractal Example – the Julia Set

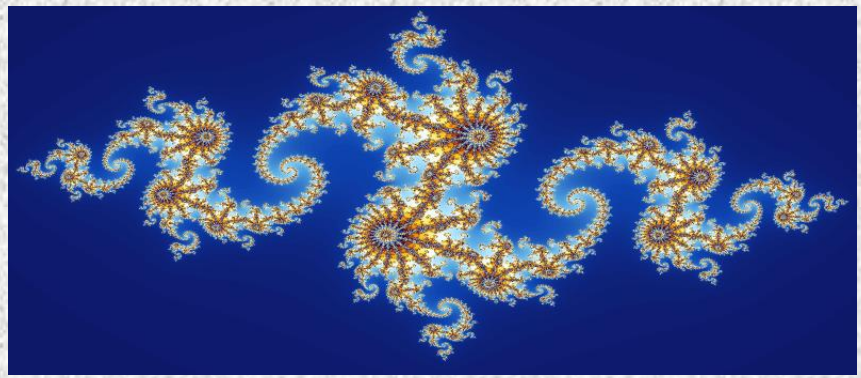
Pick a point (Z_p) in the complex plane:

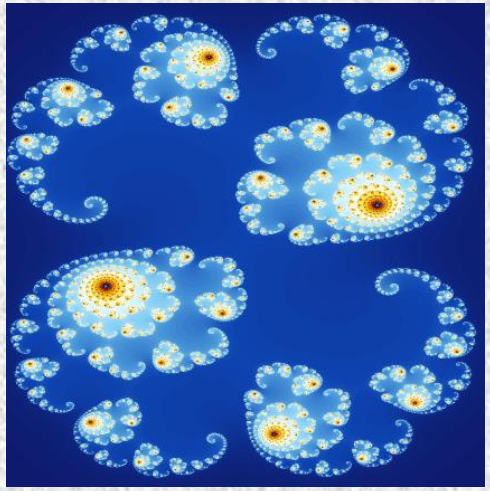
$$Z_0 = Z_p$$

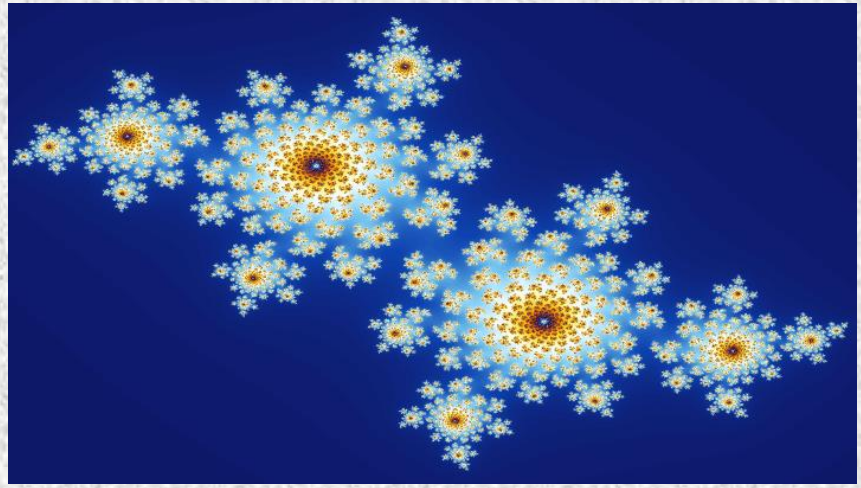
Iterate according to:

$$Z_{n+1} = Z_n^{\,2} + C$$

If Z_{n+1} diverges, discard point
Else, plot point



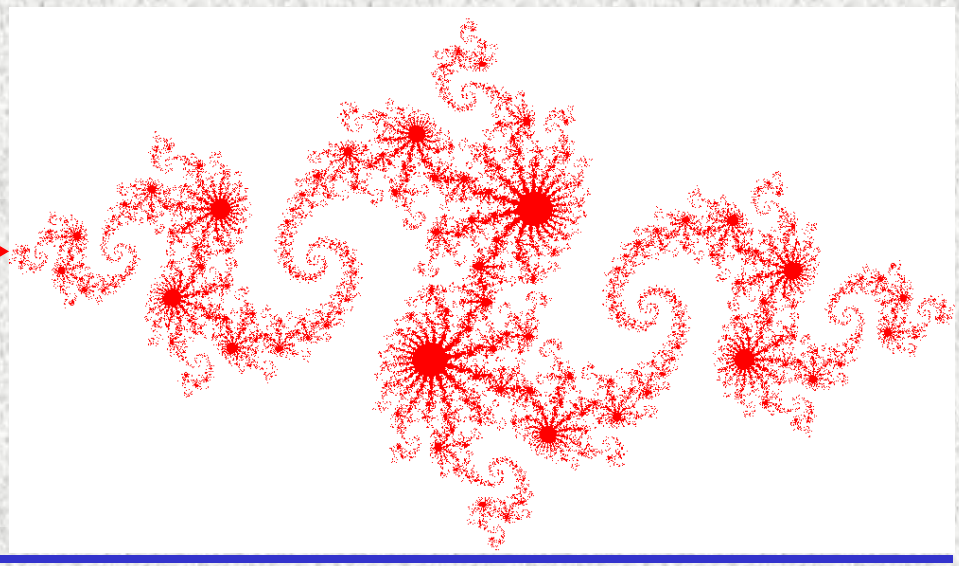




GPU Programming

- /projects/ccrstaff/lsmatott/cuda
  - julia_cpu.c ← serial CPU version (plain old C/C++)

```
$ module load cuda/4.2.9
$ g++ -o julia_cpu -L$CUDA_LIB_PATH -lcudart \
      -lglut -I$CUDA_INC_PATH -DCUDA_STUB \
      julia_cpu.c
$ export LIBGL_ALWAYS_INDIRECT=y
$ ./julia_cpu
```



GPU Programming

- /panasas/scratch/lsmatott/cuda
  - julia_gpu.cu ← parallel version, runs on GPU (CUDA C)

```
$ module load cuda/4.2.9
$ nvcc -o julia_gpu -lglut *.cu
$ export LIBGL_ALWAYS_INDIRECT=y
$ ./julia_gpu
  CUDA driver version is insufficient for CUDA
  runtime version in julia_gpu.cu at line 79
```

(no GPU on front-end, so launch interactive job on GPU partition)

```
$ fisbatch --gres=gpu:1 --nodes=1 --ntasks=1 \
           --time=00:10:00 --partition=gpu
```

-- wait a bit for the interactive job to launch

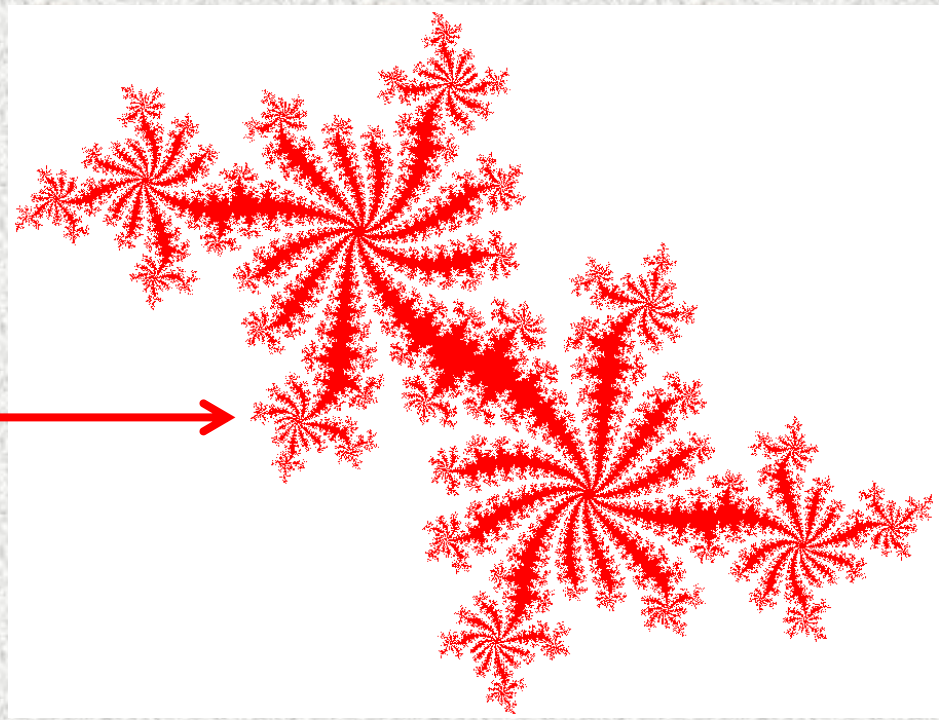
-- will eventually get a shell prompt on a GPU node

```
[lsmatott@k5n11s01 lsmatott]$
```

GPU Programming

- Commands for interactive job

```
$ module load cuda/4.2.9
$ export LIBGL_ALWAYS_INDIRECT=y
$ ./julia_gpu
```



GPU Programming

Online Resources

- http://ccr.buffalo.edu
  - User Support → Software Resources → CUDA
  - User Support → Software Resources → MATLAB → GPU
- NVIDIA Developer Zone
- http://developer.nvidia.com/cuda-downloads

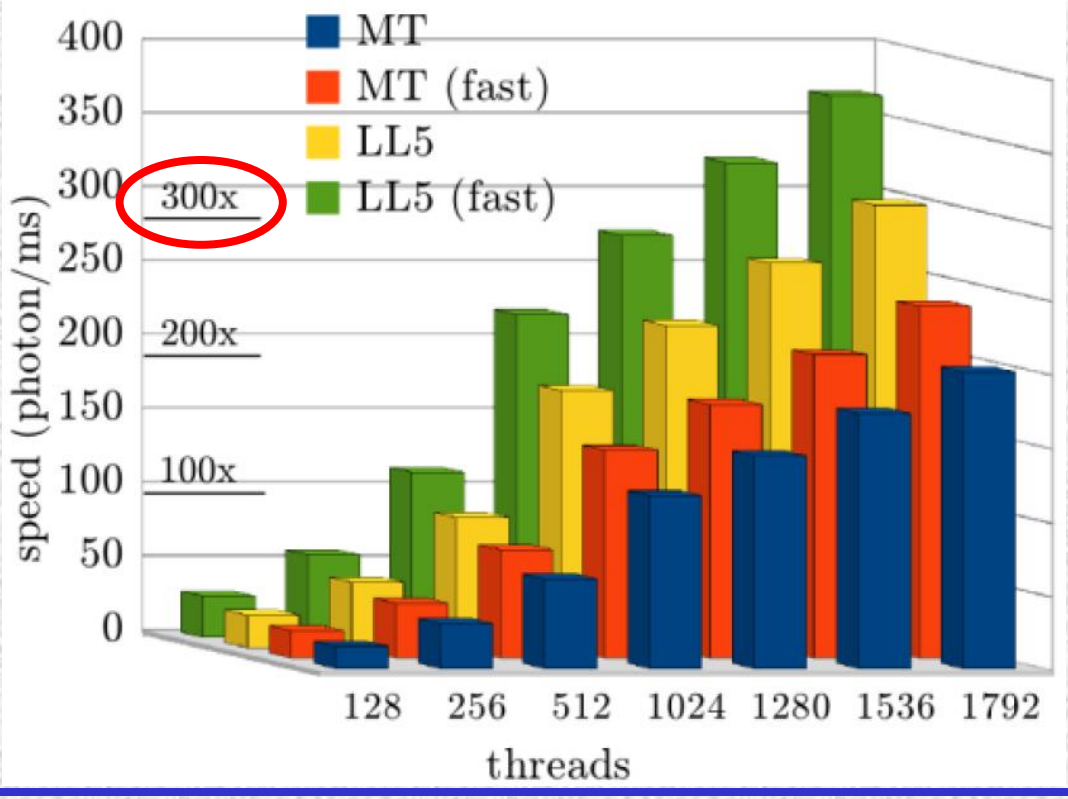
Overview

- History of GP GPU
- GPU Hardware
- GPU Software
- GPU at UB CCR
  - GPU Resources
  - GPU Programming
- GPU Performance
- Future Trends

GPU vs. CPU Performance

Monte Carlo simulation of photon migration in 3D turbid media accelerated by graphics processing units

Qianqian Fang* and David A. Boas



GPU vs. CPU Performance

Towards Flow Cytometry Data Clustering on Graphics Processing Units

Jeremy Espenshade[1], Doug Roberts[1], James Cavenaugh[2], and Gregor von Laszweski[1]

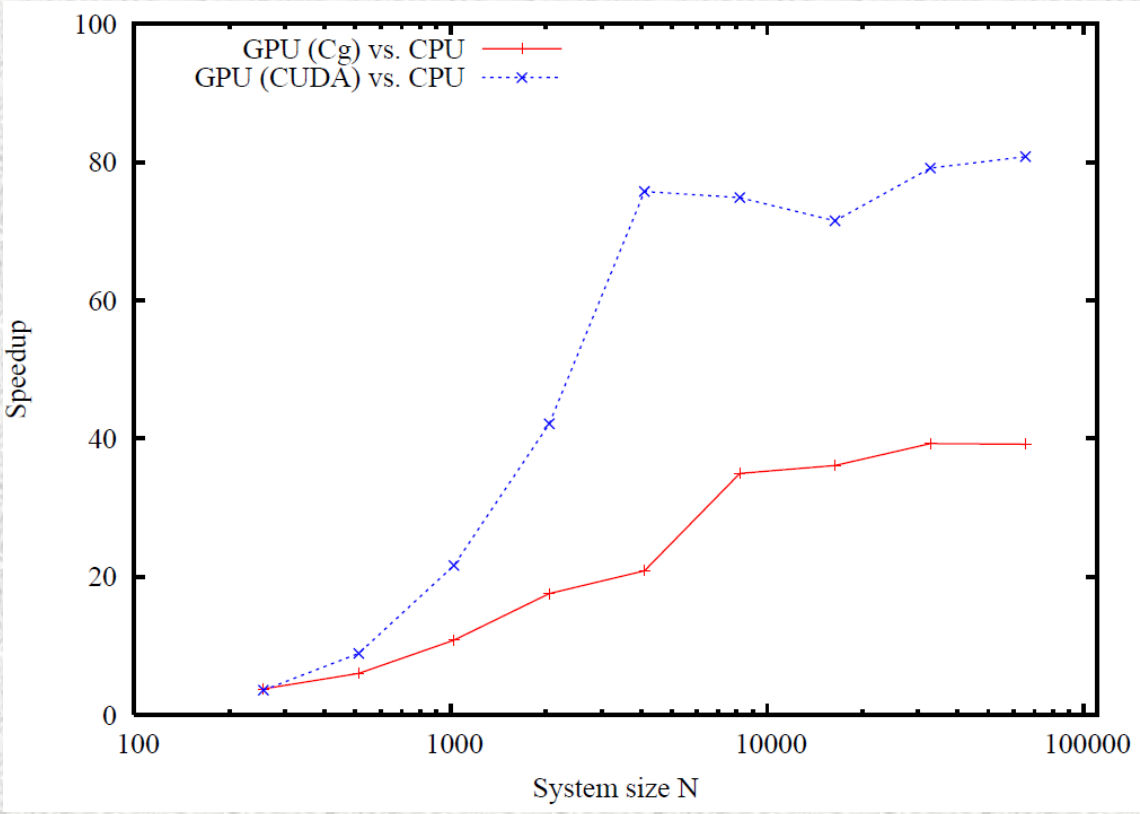
K-Medoids Performance Summary

	Sequential (ms)	CUDA (ms)	Speed Up
2	471	39.53	13.18
4	1199.5	44.67	27.77
8	3559.5	72.06	52.72
16	11772	140.47	98.14
32	42616	313.8	159.68

GPU vs. CPU Performance

Harvesting graphics power for MD simulations

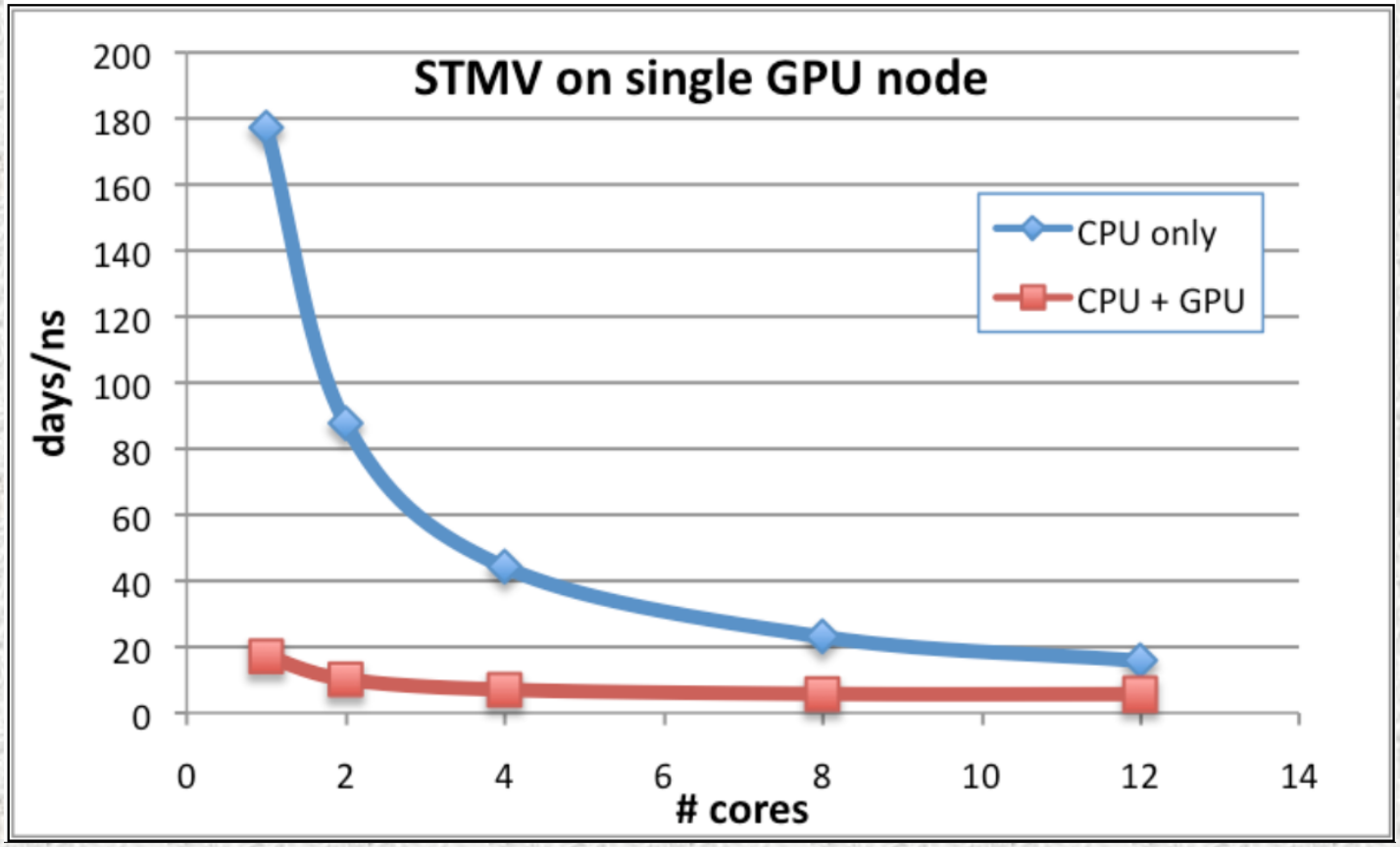
J.A. van Meel *, A. Arnold *, D. Frenkel *,
S.F. Portegies Zwart †‡, R.G. Belleman‡



GPU vs. Cluster Performance

biowulf
AT THE NIH

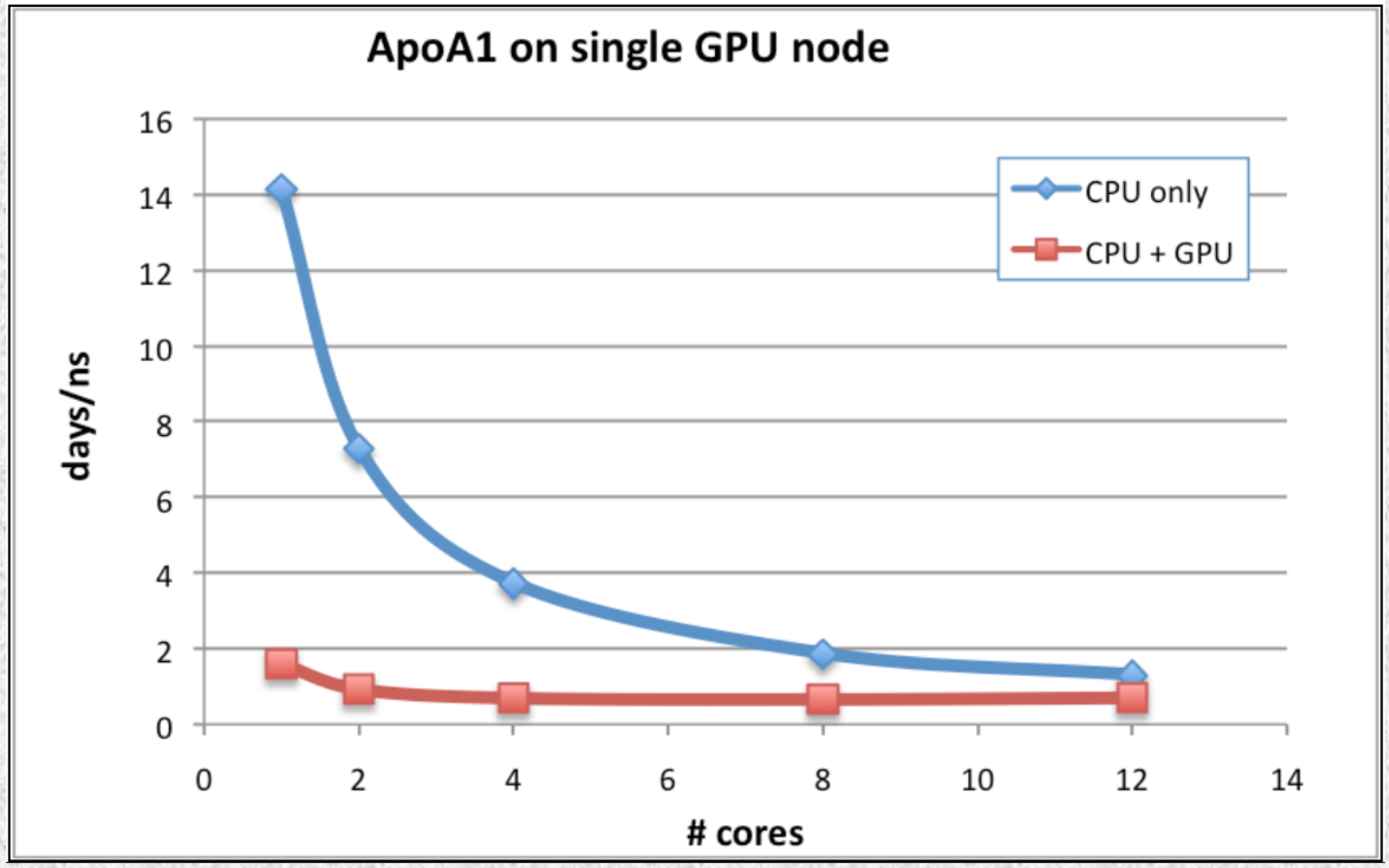
NAMD on Biowulf GPU nodes



GPU vs. Cluster Performance

biowulf AT THE NIH

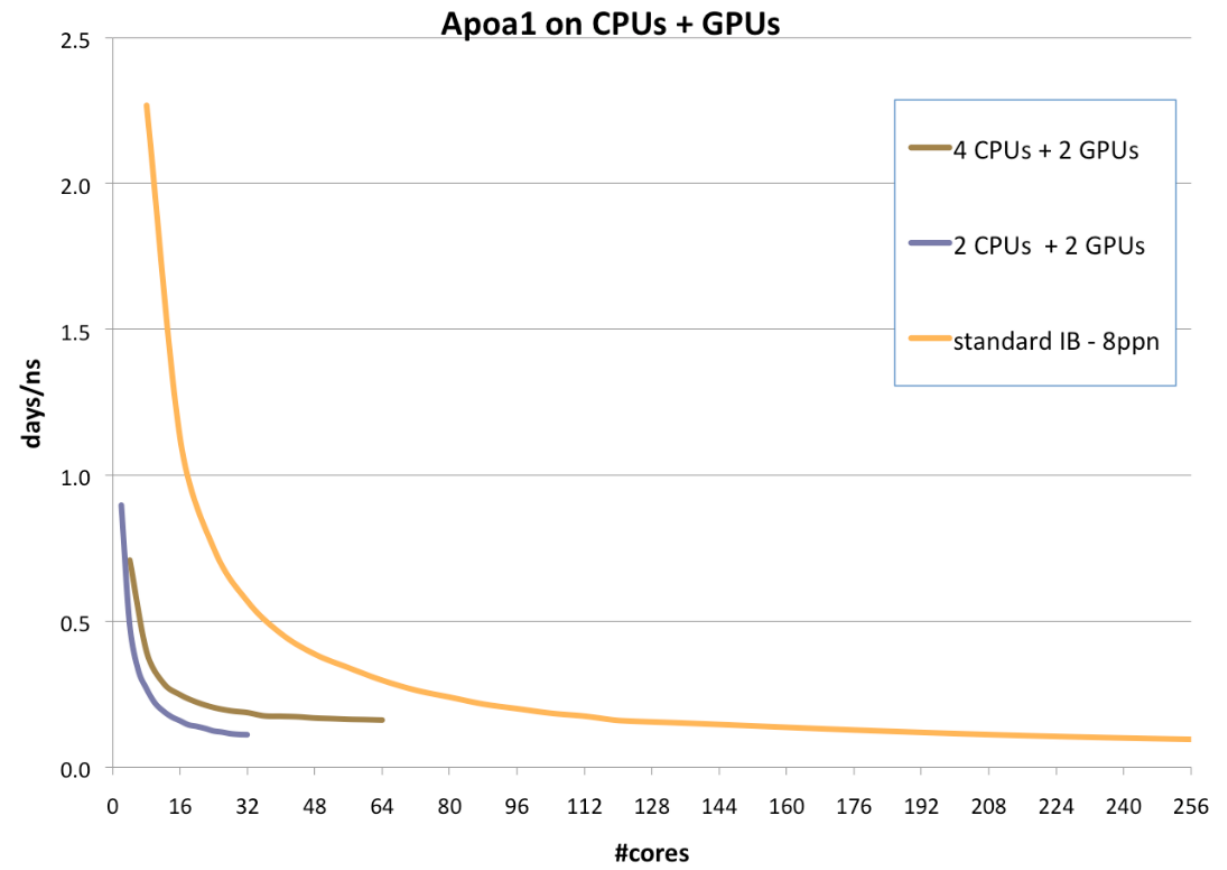
NAMD on Biowulf GPU nodes



Hybrid Performance

biowulf AT THE NIH

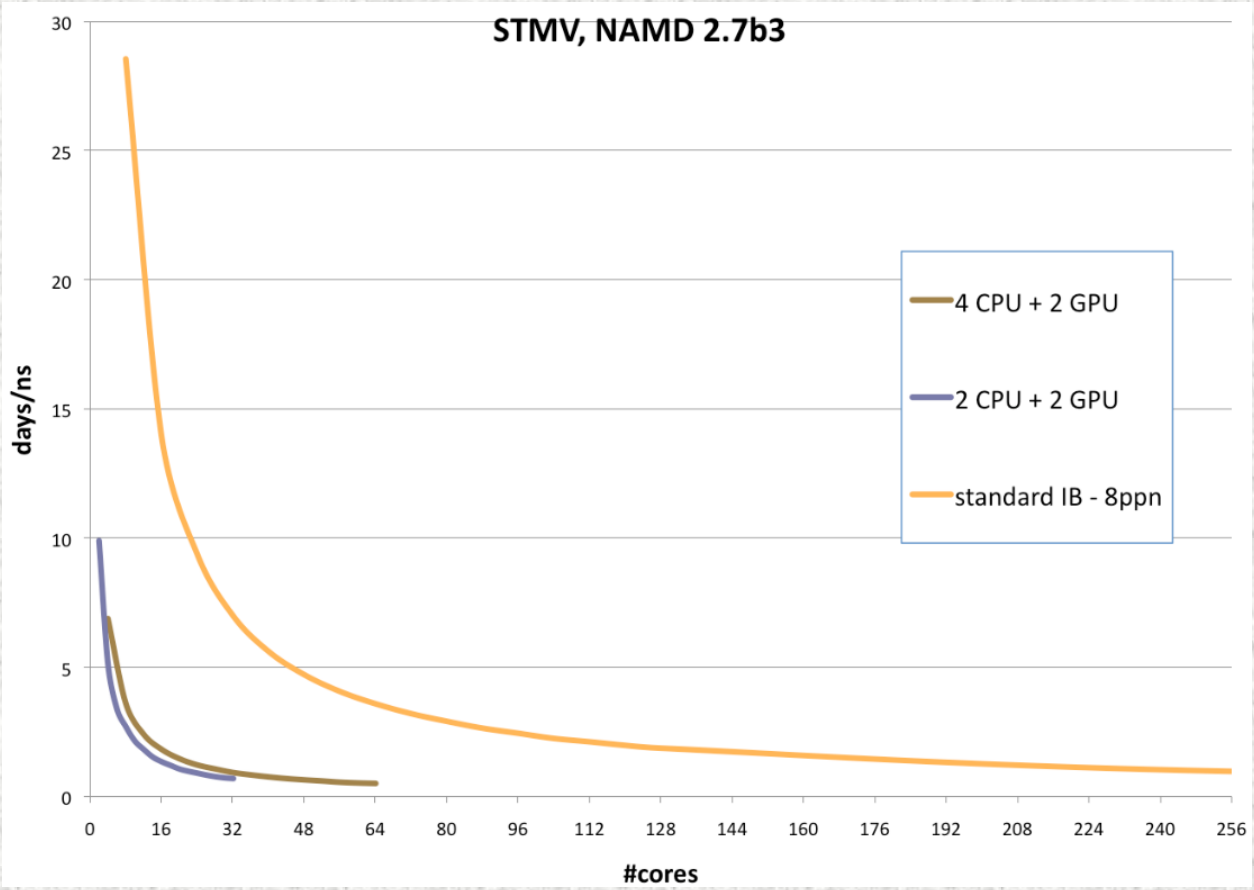
NAMD on Biowulf GPU nodes



Hybrid Performance

biowulf AT THE NIH

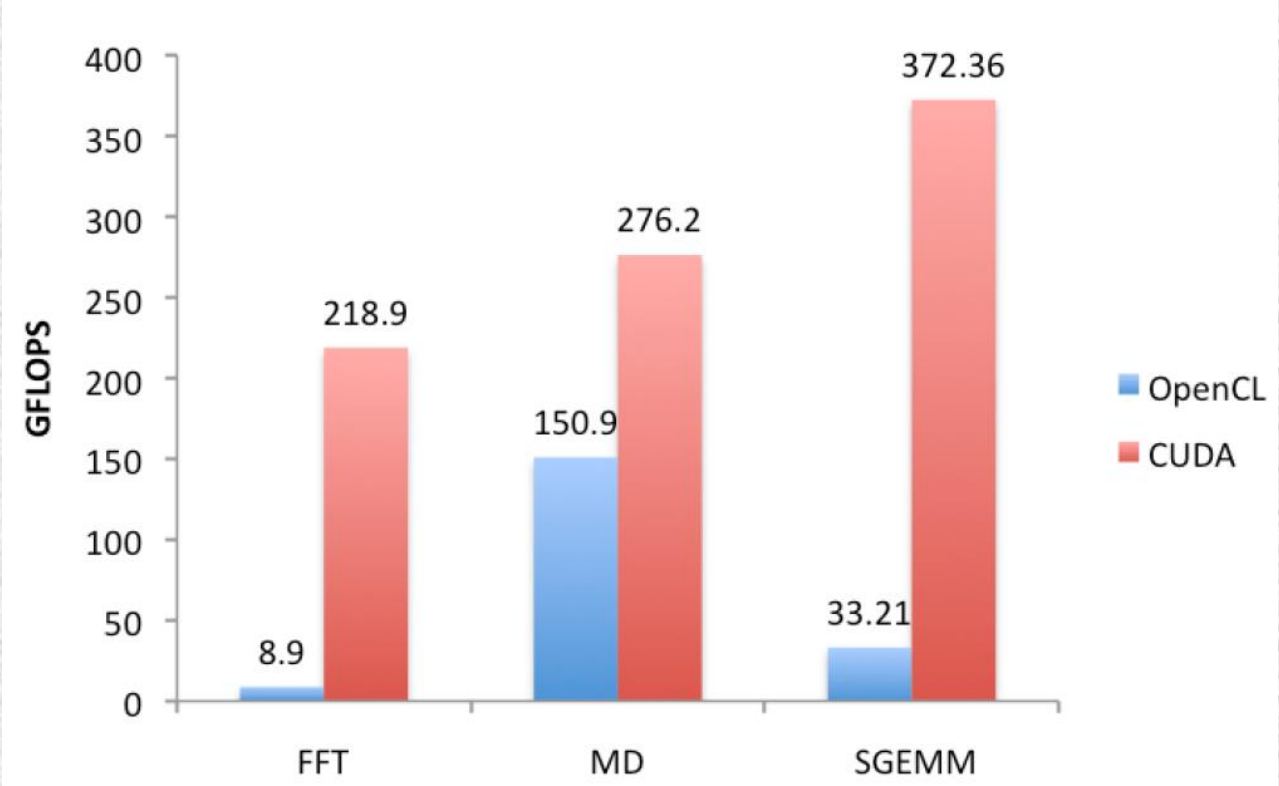
NAMD on Biowulf GPU nodes



CUDA vs. OpenCL Performance

The Scalable HeterOgeneous Computing (SHOC) Benchmark Suite

Anthony Danalis[‡†] Gabriel Marin[†] Collin McCurdy[†] Jeremy S. Meredith[†]
Philip C. Roth[†] Kyle Spafford[†] Vinod Tipparaju[†] Jeffrey S. Vetter[†]



shoc module now available

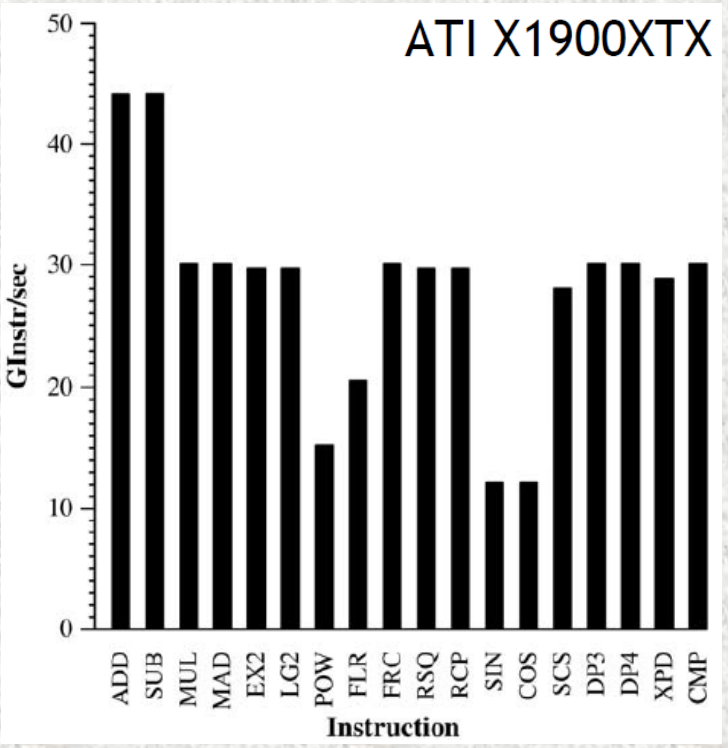
```
[lsmatott@rush:~]$ module avail shoc

------------------------------------

shoc/1.1.5
[lsmatott@rush:~]$
```

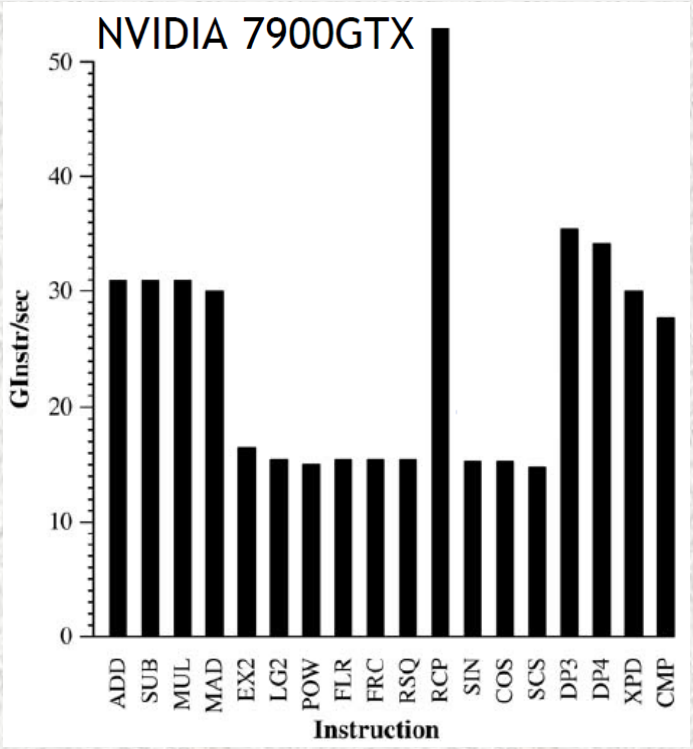
ATI vs. NVIDIA Performance

GPUBENCH

How much does your GPU bench?

Instruction Set Speeds





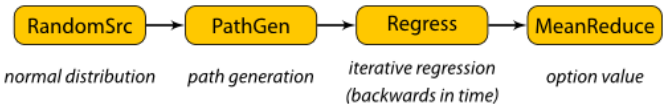
Intel Phi vs. NVIDIA GPU

Accelerators battle for compute-intensive analytics in Finance

Monte-Carlo LIBOR Swaption Portfolio Pricer

Paths	Sequential	Sandy-Bridge CPU[1,2]	Xeon Phi[1,2]	Tesla GPU[2]
128K	13,062ms	694ms	603ms	146ms
256K	26,106ms	1,399ms	795ms	280ms
512K	52,223ms	2,771ms	1,200ms	543ms

Monte-Carlo Pricing of American Options



Paths	Ivy-Bridge CPU[1,2]	Xeon Phi[1,2]	Tesla GPU[2,3]
128K	41.8x	20.9x	52.9x
256K	48.5x	31.0x	71.8x
512K	45.2x	39.7x	86.6x

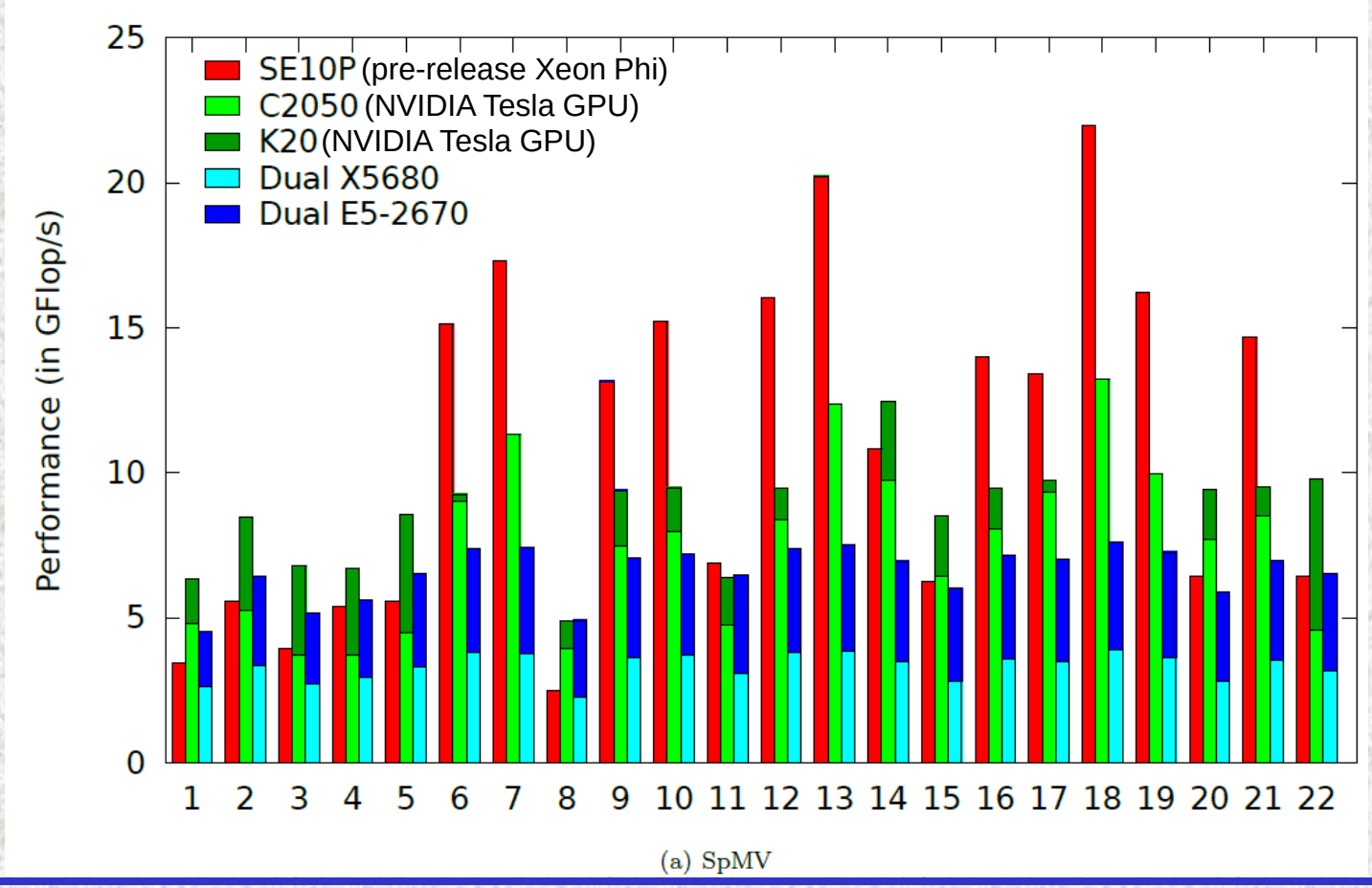
> *"We've seen that there is one processor that needs to be added to the picture — the commodity multi-core CPU. This is already a part of many server configurations, and for some applications, e.g., Monte-Carlo pricing of American options, it can give better or comparable performance than an accelerator processor when optimized correctly. Between NVIDIA's Kepler GPUs and Xeon Phi, the GPU wins for both of our test applications."*

Source: http://blog.xcelerit.com/intel-xeon-phi-vs-nvidia-tesla-gpu/

Intel Phi vs. NVIDIA GPU

Performance Evaluation of Sparse Matrix Multiplication Kernels on Intel Xeon Phi

Erik Saule , Kamer Kaya , and Umit V. Çatalyürek
The Ohio State University



(a) SpMV

Overview

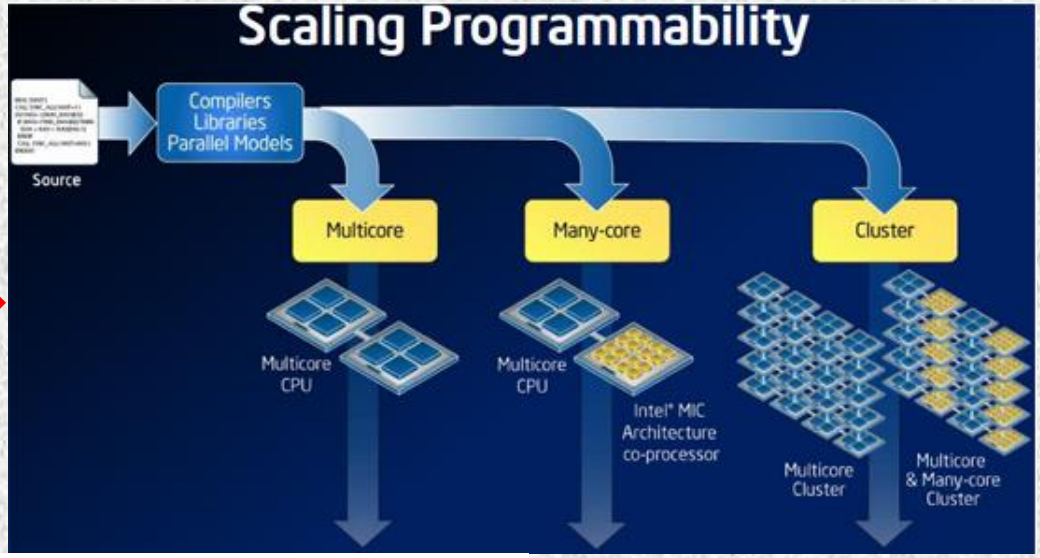
- History of GP GPU
- GPU Hardware
- GPU Software
- GPU at UB CCR
  - GPU Resources
  - GPU Programming
- GPU Performance
- Future Trends

Future Trends

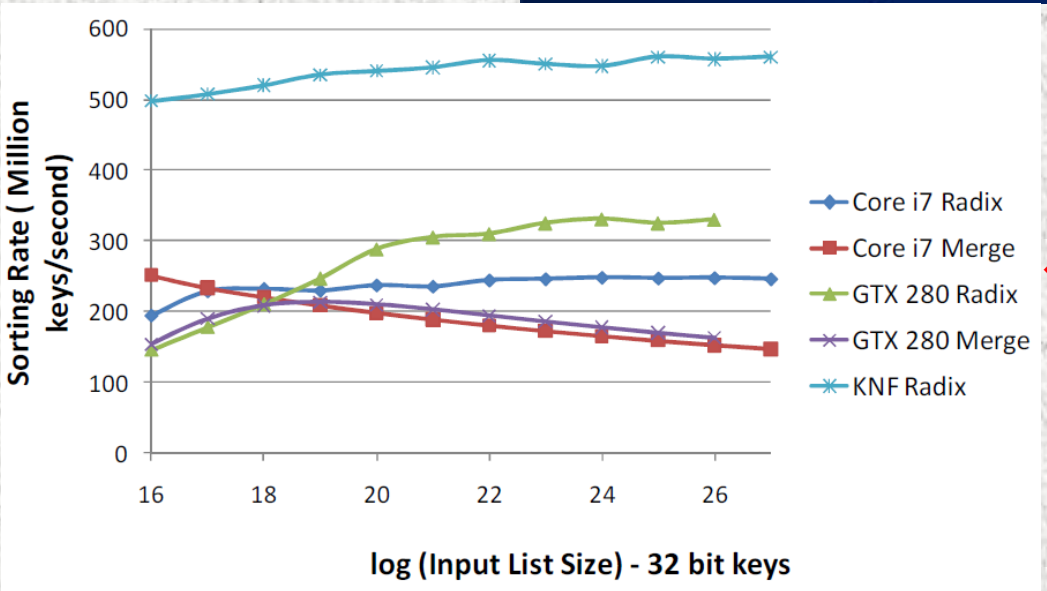
- NVIDIA
  - Currently well-established in GP GPU
  - Extensive support for CUDA
  - Recent emphasis on OpenACC
- AMD/ATI
  - APU products (CPU + GPU on a single chip)
  - Support OpenCL and C++ AMP
- Intel
  - Making inroads with MIC architecture
  - Intel compilers to provide seamless support
  - 10 Petaflop Stampede cluster at U. Texas
- Intel and AMD
  - Moving away from graphics-oriented GP GPU legacy
  - Processing accelerators vs. graphics accelerators

Future Trends

Is Intel MIC (Xeon Phi) a game-changer?



Easy code migration →



← Superior performance

Future Trends

- UB CCR has Intel Xeon Phi node
- Working on installation and integration with SLURM scheduler