Lincoln/AC Experience

Wen-mei Hwu

University of Illinois, Urbana Champaign

With contributions from

Mike Showerman, Jeremy Enos, Guochun Shi, Volodymyr Kindratenko, Nacho Navarro, John Stone, Jim Phillips, Chris Rodrigues, Issac Gelado, I-Jui Song, and Sara Baghsorkhi

National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign

GPU Clusters at NCSA

- **Lincoln**
  - Production system available via the standard NCSA/TeraGrid HPC allocation

- **AC**
  - Experimental system available for exploring GPU computing

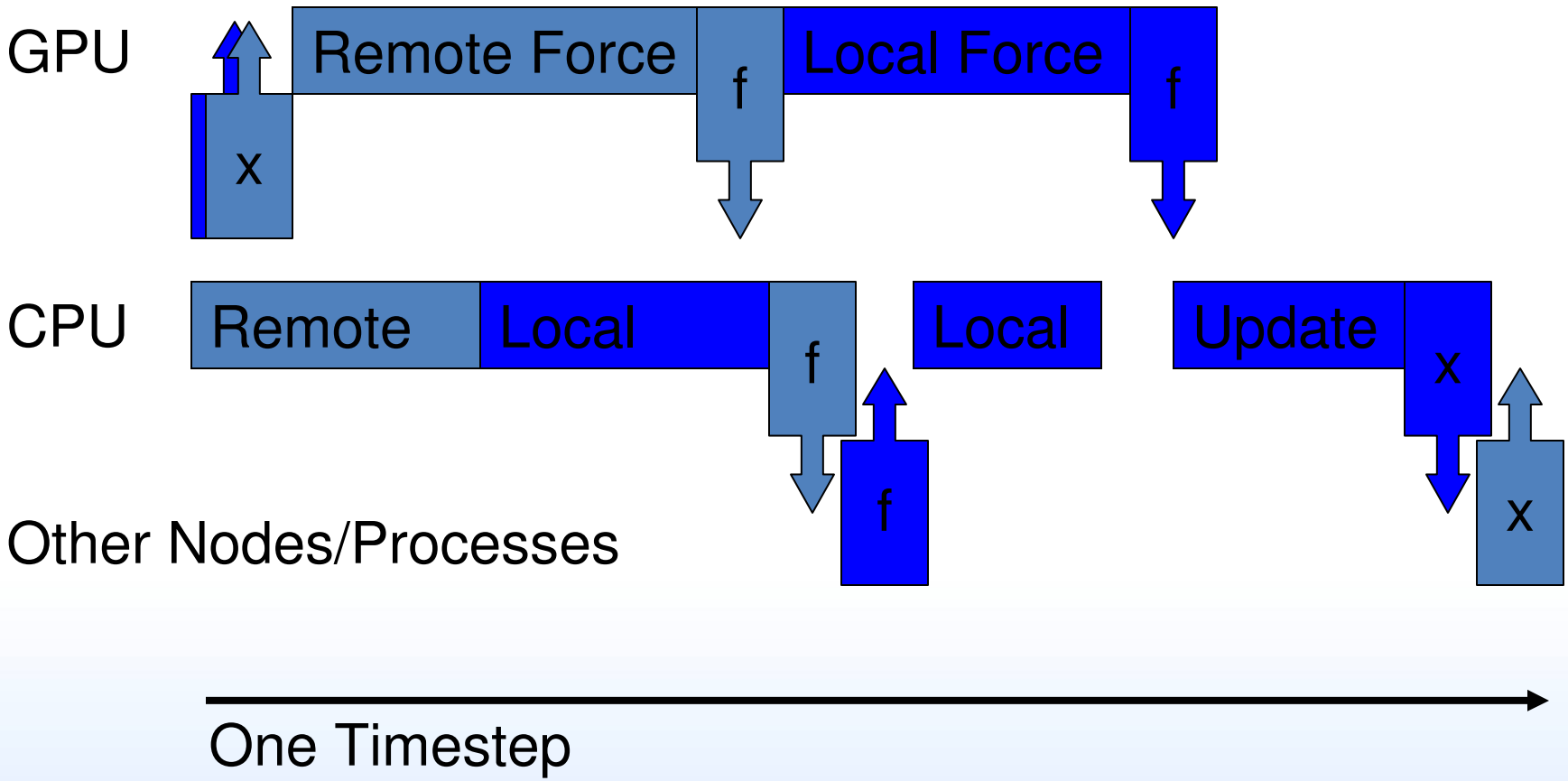
Lincoln vs. AC: Configuration

- **Lincoln (Production)**
  - Compute cores
    - CPU cores: 1536
    - GPU units: 384
    - CPU/GPU ratio: 4
  - Memory
    - Host memory: 16 GB
    - GPU Memory: 8 GB/host
    - Host mem/GPU: 8 GB
  - I/O
    - PCI-E 2.0 (x8)
    - GPU/host bandwidth: 4 GB/s
    - IB bandwidth/host: 8 Gbit/s
- **AC (Experimental)**
  - Compute cores
    - CPU cores: 128
    - GPU units: 128
    - CPU/GPU ratio: 1
  - Memory
    - Host memory: 8 GB
    - GPU Memory: 16 GB/host
    - Host mem/GPU: 2 GB
  - I/O
    - PCI-E 1.0 (x16)
    - GPU/host bandwidth: 4 GB/s
    - IB bandwidth/host: 16 Gbit/s

HPC Application Acceleration Work at UIUC

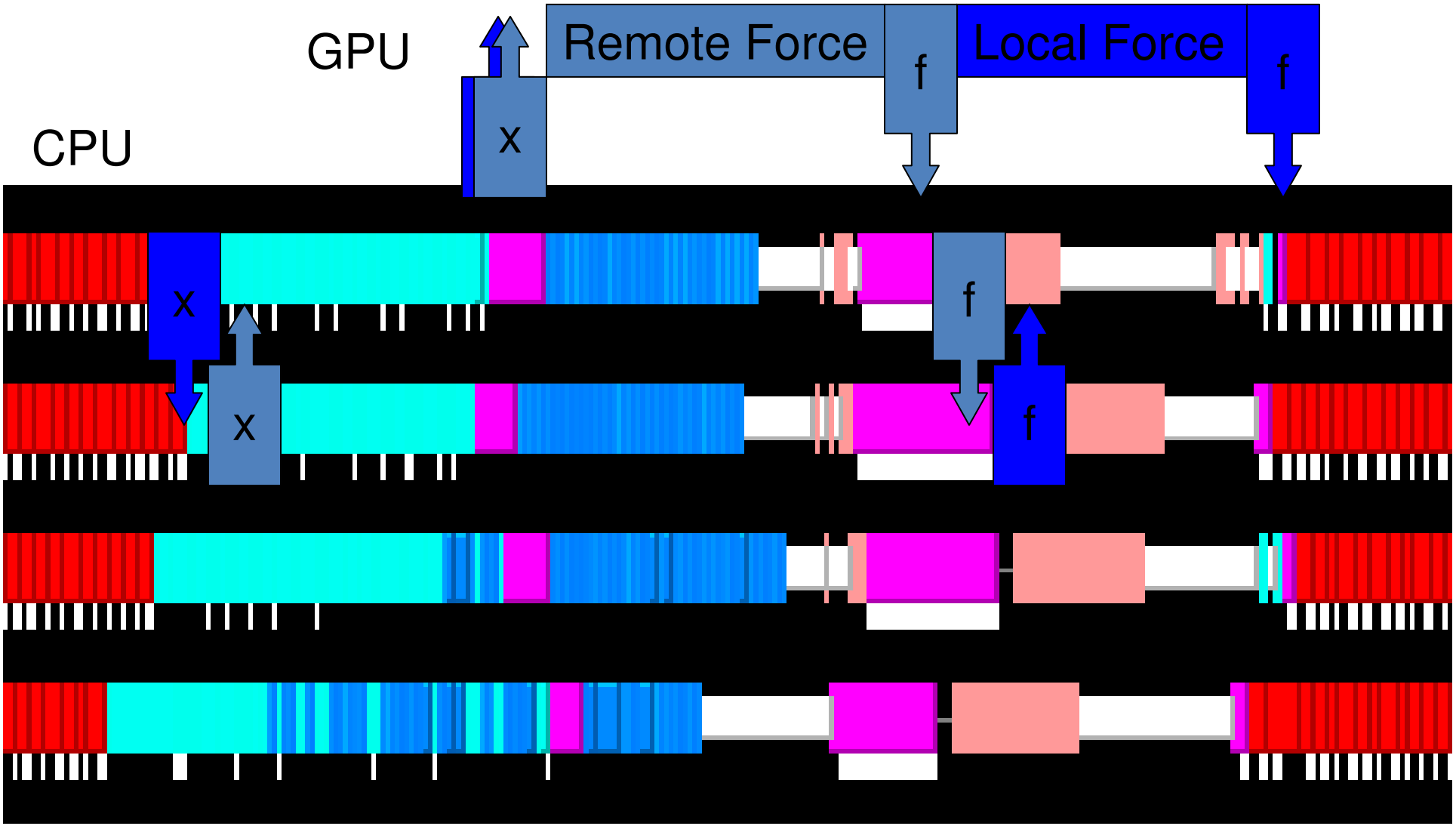
- **Collaborative efforts across campus**
  - IACAT/NCSA, CUDA Center of Excellence, Coordinated Science Lab, Physics, Chemistry, Mechanical, Material, Bioengineering, ECE, CS, …
- **Broad range of applications**
  - Molecular dynamics, CFD, Lattice QCD/MILC, Quantum Chemistry, Weather/Climate, Cosmology, Biomedical Imaging, Genomics…
- **New Programming Tools and Utilities**
  - Performance models and tools, memory optimizations, domain specific code generators, CPU/GPU data comunication

NAMD: Overlapping GPU and CPU with Communication

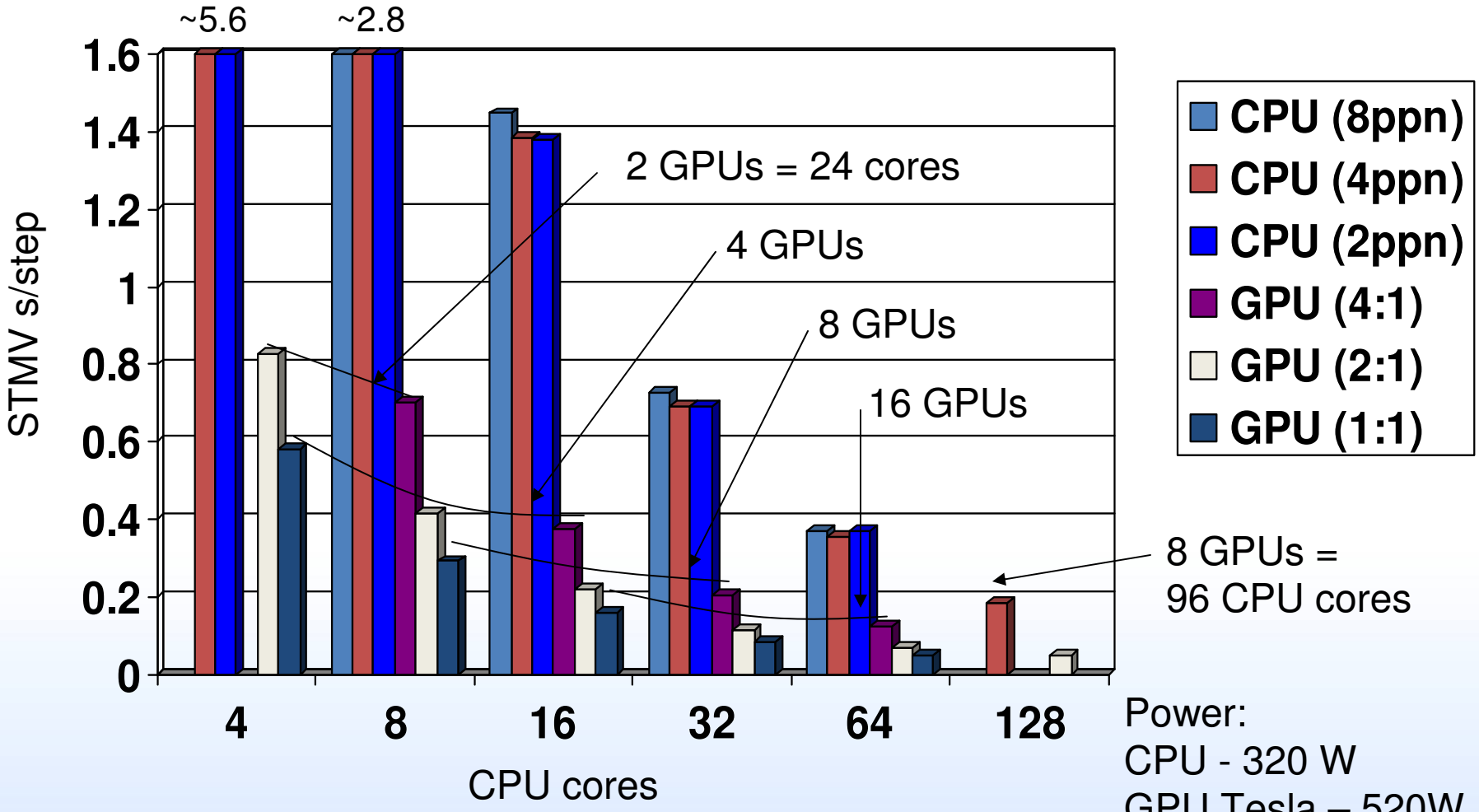


Actual Timelines from NAMD

Generated using Charm++ tool "Projections"



NAMD on Lincoln Cluster Performance
(8 cores and 2 GPUs per node, very early results)



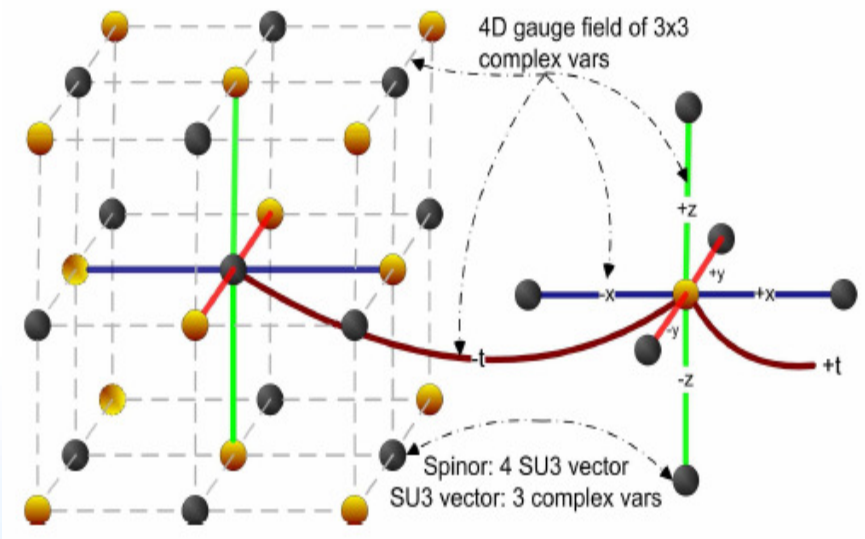
Power:
CPU - 320 W
GPU Tesla – 520W

Courtesy of James Phillips, UIUC

NCSA

Lattice QCD

- **Solving the quantum chromodynamics theory of quarks and gluons in 4-D lattice of space and time**
- **A typical computation involves collecting neighboring links/spinors in 4-D lattice and update the local spinor/momentum.**



Courtesy of K. Z. Ibrahim

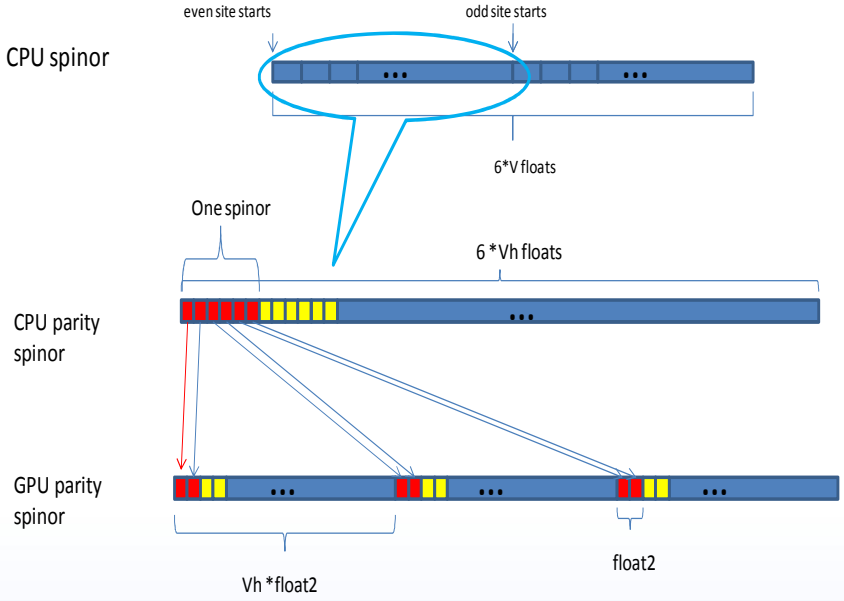
Time distribution for a run on 2048 XT3 (BigBen) cpus using a $40^3 \times 96$ grid ($5 \times 10^2 \times 6$ per cpu) with $m_l = 0.1 m_s$

Activity	time(s)	MF/cpu	per cent
CG	2987	530	58.5
FF	1125	579	22.0
GF	489	469	9.5
Fat	442	627	8.7
Long	24	340	<1
Input config.	41		<1
total above	5108		
unaccounted	104		1.9
wallclock	5212		

CPU time in MILC program for one typical data set

Data layout in CPU and GPU

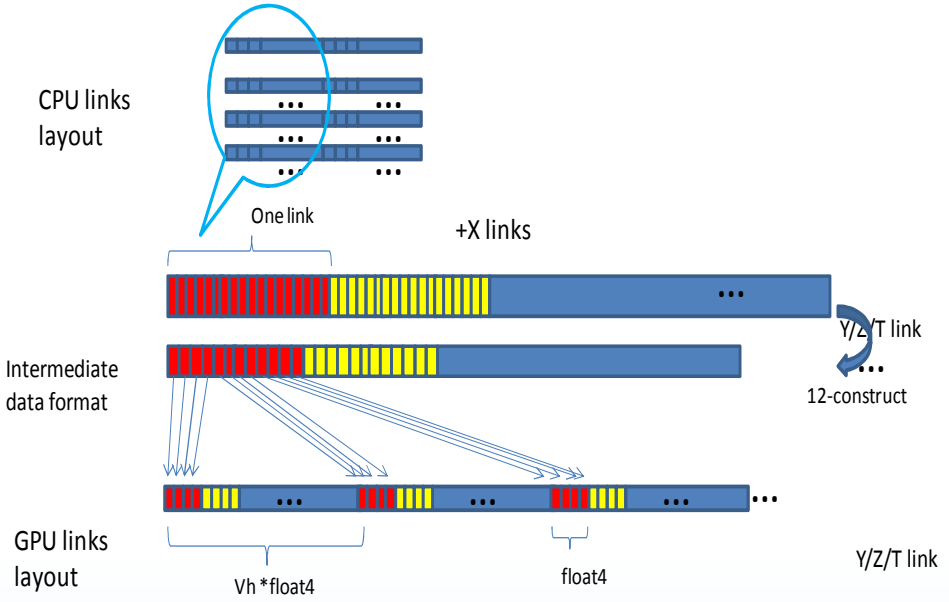
- All the four major components (CG, FF, GF, Fat) of MILC program are bandwidth bound
- Data layout in GPU is arranged so that we can do coalesce read and write



GPU kernel code to read spinor

```
#define READ_SPINOR_SINGLE(spinor)                  \
  float2 I0 = tex1Dfetch((spinor), sp_idx + 0*Vh);  \
  float2 I1 = tex1Dfetch((spinor), sp_idx + 1*Vh);  \
  float2 I2 = tex1Dfetch((spinor), sp_idx + 2*Vh);
```

Spinor data layout



GPU kernel code to read link

```
#define READ_FAT_MATRIX_12_SINGLE(gauge, dir, idx)                  \
  float4 FAT0 = tex1Dfetch((gauge), idx + ((dir/2)*3+0)*Vh);    \
  float4 FAT1 = tex1Dfetch((gauge), idx + ((dir/2)*3+1)*Vh);    \
  float4 FAT2 = tex1Dfetch((gauge), idx + ((dir/2)*3+2)*Vh);    \
  float4 FAT3 = make_float4(0,0,0,0);                           \
  float4 FAT4 = make_float4(0,0,0,0);
```

Link data layout

Preliminary results

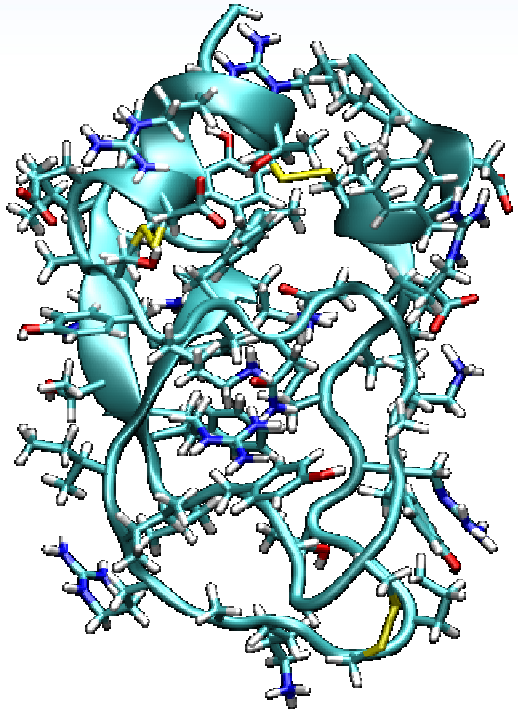
	description	status	Results* (Gflops)
CG	**Update spinors using the neighboring links and spinors through Conjugate Gradient process**	**Fully implemented (12 and 8-reconstruct, SP, DP and half precision, mixed precisions)**	**28.7 (DP)** **86.1 (SP)** **120 (HP)**
Fat	**Update fatlink using the neighboring links**	**Implemented the 12-reconstruct, single precision case**	**156 (SP)**
GF	**Update the momentum using the neighoring links**	**Work in process**	**x**
FF	**Update the momenum using the neighboring links and spinors**	**Not started yet**	**x**

** The results is obtained in a single gtx280*

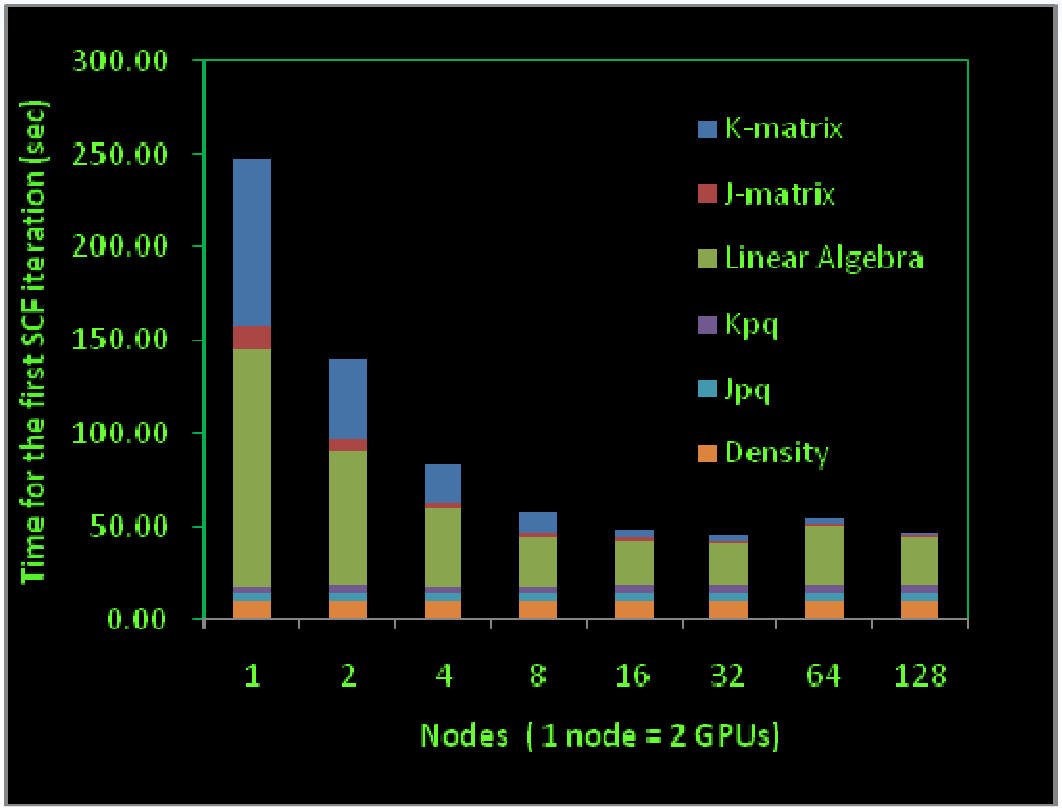
CPU DP
0.53 GFLOPS

New CUDA (OpenCL) CG and Sparse package/framework from UIUC.

TeraChem



Bovine pancreatic trypsin inhibitor (BPTI) 3-21G, 875 atoms, 4893 basis functions

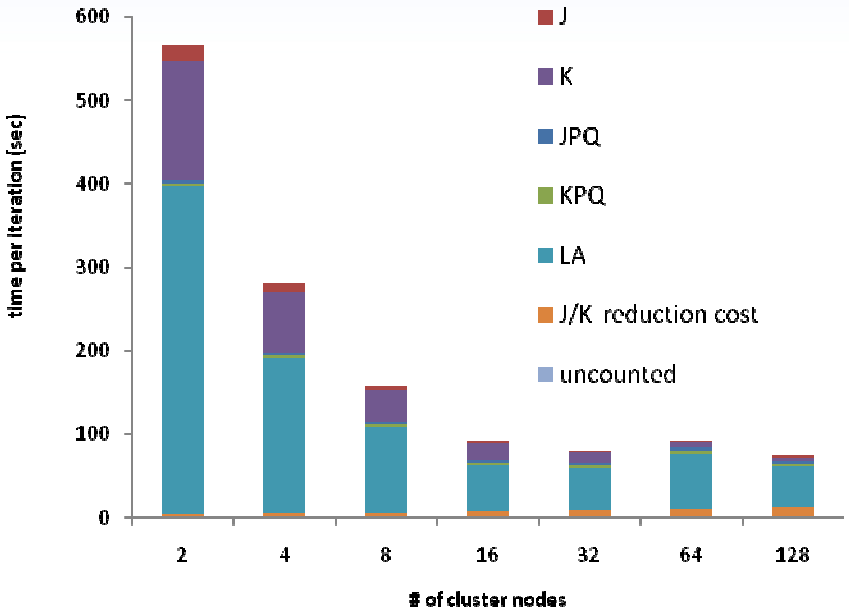


MPI timings and scalability

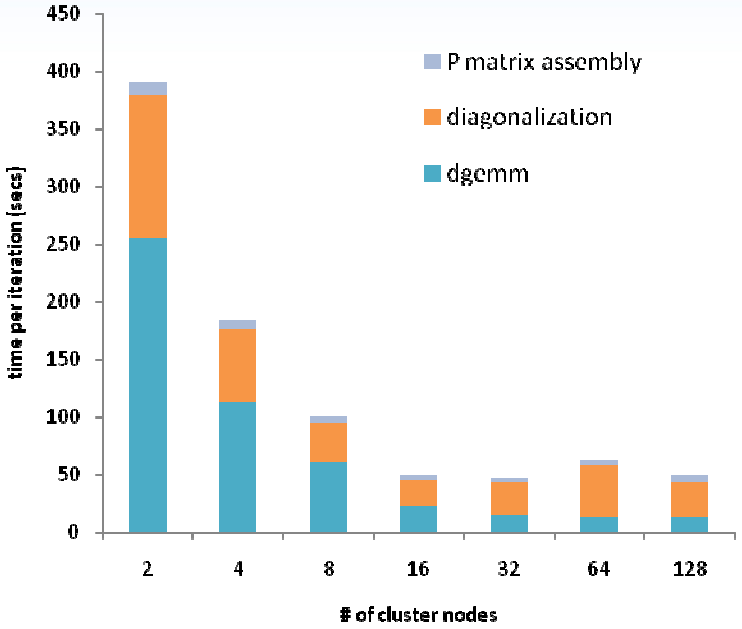
GPU (computed with SP): K-matrix, J-matrix
CPU (computed with DP): the rest

Courtesy of Ivan Ufimtsev, Stanford

Performance in Lincoln cluster



(a) Execution time of CspA molecule (1732 atoms) as a function of the number of GPU cluster nodes used to perform the calculations.



(b) Linear algebra execution time breakdown CspA molecule as a function of the number of GPU cluster nodes used

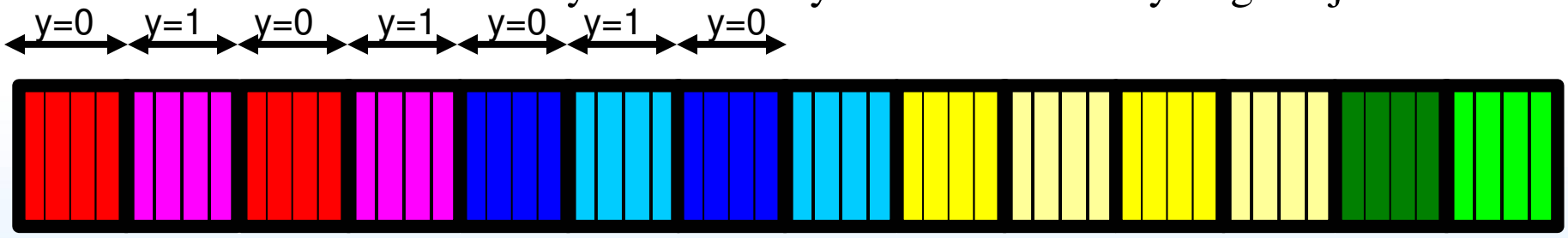
- J and K scales well as node number increases
- Linear Algebra (LA) can only scale to 16 nodes
- Among LA, the diagonalization has the worst scalability
- CPU and communications are the bottleneck

New Programming Tools from UIUC

- **ADAPT (PPoPP 2010, Baghsorkhi, et al)**
  - Where and how CUDA kernels spend there cycles through source code analysis
- **MCUDA (CGO 2010, Stratton, et al)**
  - Generating efficient CPU SSE friendly code from CUDA
- **CUDA-to-OpenCL (Nandakumar, et al)**
  - Automatic conversion of CUDA to OpenCL GPU code (soon CPU code)
- **Upcoming**
  - Gluon/Pyon, automatic memory optimization for gridded applications, automatic conversion from scatter to gather

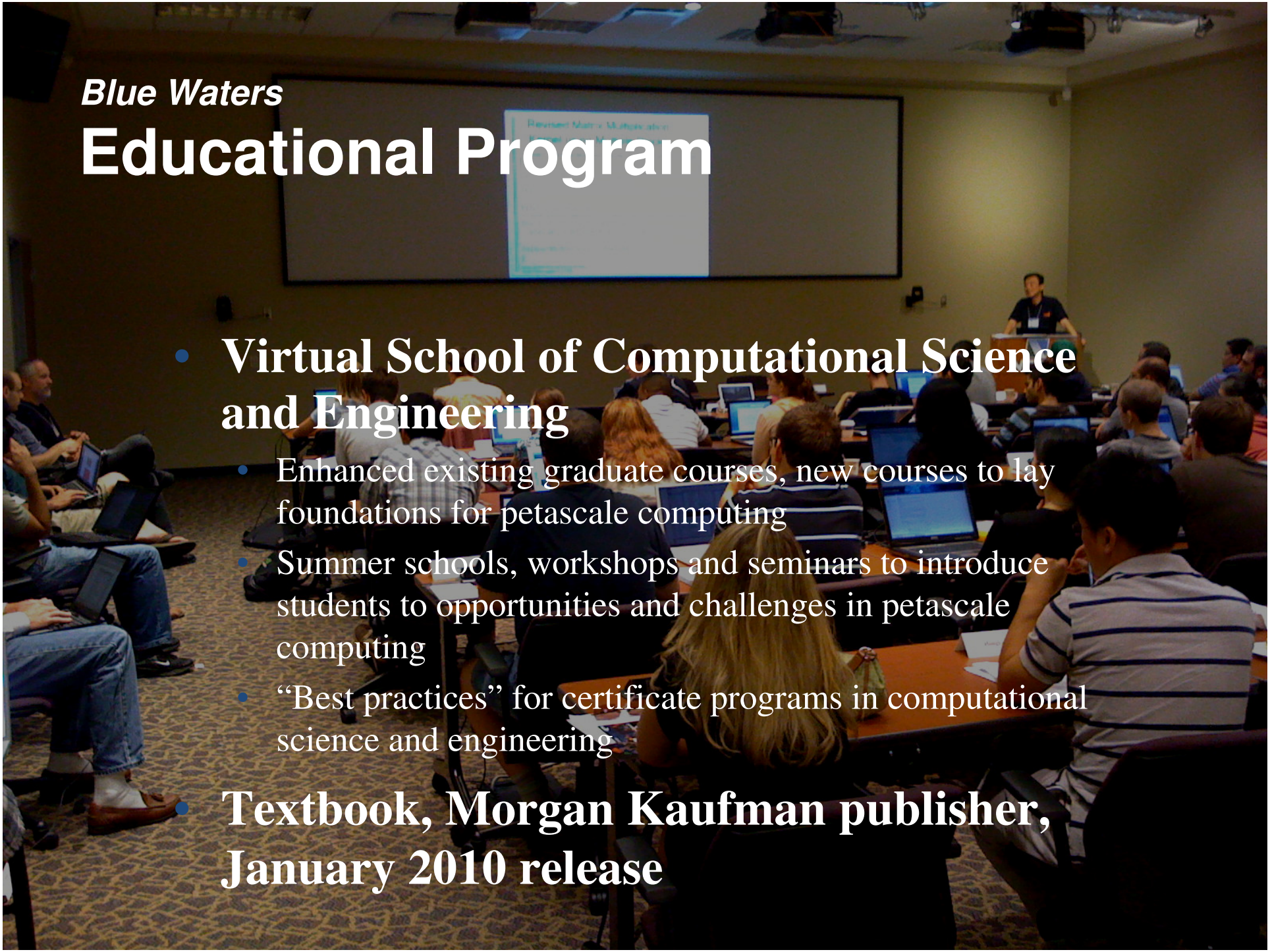
LBM: The best layout is neither SoA nor AoS

- **Tiled Array of Structure, using lower bits in x and y indices, i.e. $x_{3:0}$ and $y_{3:0}$ as lowest dimensions: [z][$y_{31:4}$][$x_{31:4}$][e][$y_{3:0}$][$x_{3:0}$]**
  - $F(z, y, x, e) = z * \lceil |Y|/2^4 \rceil * \lceil |X|/2^4 \rceil * |E| * 2^4 * 2^4 + y_{31:4} * \lceil |X|/2^4 \rceil * |E| * 2^4 * 2^4 + x_{31:4} * |E| * 2^4 * 2^4 + e * 2^4 * 2^4 + y_{3:0} * 2^4 + x_{3:0}$
- **6.4X faster than AoS, 1.6X faster than SoA on GTX280:**
  - Better utilization of data by neighboring cells
  - This is a scalable layout: same layout works for very large objects.



New Runtime Utilities from UIUC

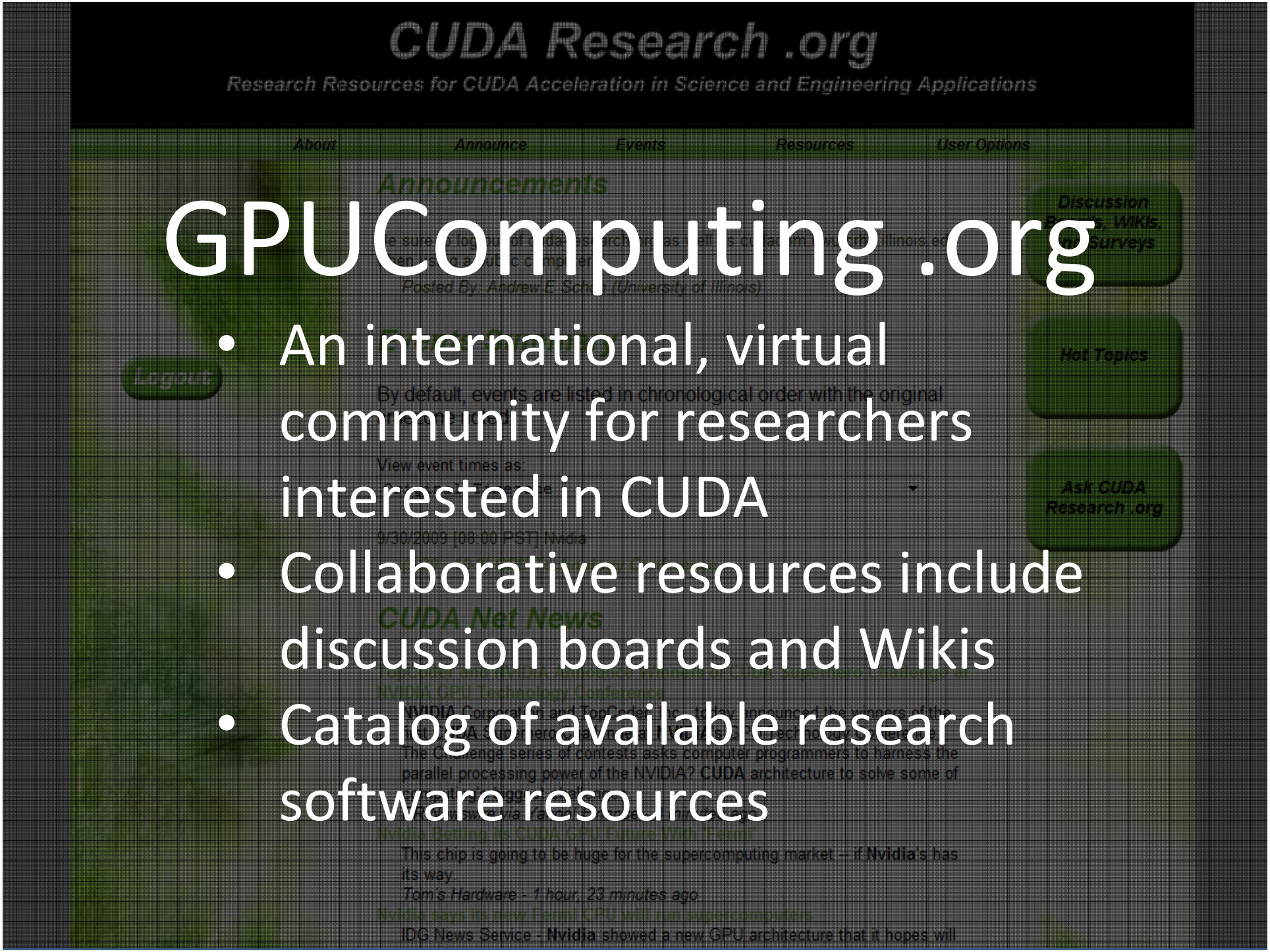
- **GMAC (ASPLOS 2010, Gelado, et al, UIUC/UPC)**
  - Asymmetric Distributed Shared Memory for CPU/GPU, legacy code and I/O library support, multi GPU and peer I/O (with Fermi)
- **CUDA memtest**
  - For both hard and soft memory errors
- **CUDA/OpenCL wrapper**
  - NUMA affinity mapping, GPU device virtualization, device rotation, memory scrubber
- **GPU-aware Cluster management utilities**



Blue Waters

Educational Program

- **Virtual School of Computational Science and Engineering**
  - Enhanced existing graduate courses, new courses to lay foundations for petascale computing
  - Summer schools, workshops and seminars to introduce students to opportunities and challenges in petascale computing
  - "Best practices" for certificate programs in computational science and engineering
- **Textbook, Morgan Kaufman publisher, January 2010 release**



GPUComputing .org

- An international, virtual community for researchers interested in CUDA
- Collaborative resources include discussion boards and Wikis
- Catalog of available research software resources

Conclusions

- **GPU acceleration producing real results**
- **System tools becoming GPU aware, but still some gaps to fill**
- **Heterogeneous parallel programming hard, but better HPC programming tools coming.**
- **GPUComputing.org will help accelerate research progress**

Wen-mei Hwu **w-hwu@uiuc.edu**

Thank you.