Introduction and Welcome

John Stone

Theoretical and Computational Biophysics Group

Beckman Institute for Advanced Science and Technology

University of Illinois at Urbana-Champaign

http://www.ks.uiuc.edu/Research/gpu/

Workshop on GPU Programming for Molecular Modeling , University of Illinois at Urbana-Champaign, August 2, 2013

Announcements and Reminders

- Hotel should provide continental breakfast: make sure you eat before you arrive on Saturday and Sunday mornings.
- Lunch: will be provided on Saturday (Pizza) and Sunday (Jimmy John's)
- Saturday dinner: Ad hoc

Announcements and Reminders (2)

- Please arrive between 8:40am and 9:00am on Sat./Sun. so we can start on time
- Please use the provided ethernet cable or WiFi to access the internet.
- The workshop agenda page will be updated with new links to CUDA and OpenCL example code, and PDFs of relevant papers and book chapters.

Overview of Daily Agenda

- Lectures are intended to be in an informal format. Questions are welcome at any time!
- Participant presentations and discussions are intended to be in an informal "clinic" format, where we learn about each participant's problem and discuss potential GPU solutions.
- You may use scheduled lab time and any other free time to work on your own projects on the GPU supercomputer at Oak Ridge National Laboratory.

Introductions

- Everyone introduce yourself and tell us:
  - Your name and institution
  - Length of time and level of experience you have working with CUDA or OpenCL so far
  - What do you hope to learn from this workshop?
  - Any topics from past GPU classes or workshops that you would like to know more about?

Participant Presentations, Group Discussions, GPU "clinic"

- Friday night (tonight after opening lecture)
- Saturday afternoon
- Sunday afternoon

- Any unused presentation time will be used to allow participants extra "lab" time.

A Brief Introduction to GPU Computing in Molecular Modeling

John Stone

Programmable Graphics Hardware

Groundbreaking research systems:

AT&T Pixel Machine (1989):

82 x DSP32 processors

UNC PixelFlow (1992-98):

64 x (PA-8000 +

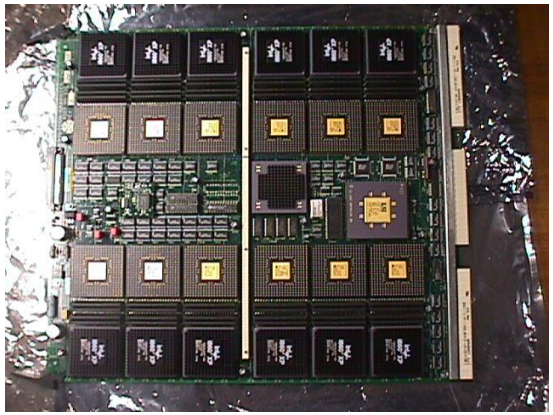
8,192 bit-serial SIMD)

SGI RealityEngine (1990s):

Up to 12 i860-XP processors perform vertex operations (*u*code), fixed-func. fragment hardware

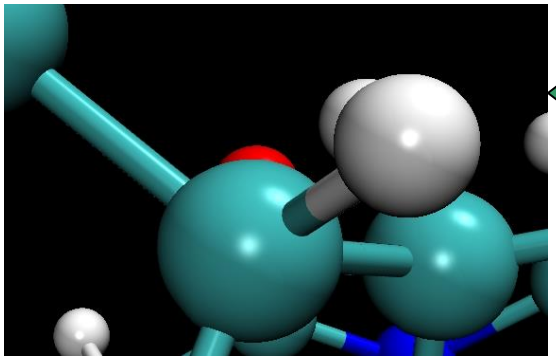
All mainstream GPUs now incorporate fully programmable processors

UNC PixelFlow Rack



SGI Reality Engine i860 Vertex Processors

GLSL Sphere Fragment Shader



- Written in OpenGL Shading Language
- High-level C-like language with vector types and operations
- Compiled dynamically by the graphics driver at *runtime*
- Compiled machine code executes on GPU

```
//
// VMD Sphere Fragment Shader (not for normal geometry)
//
void main(void) {
  vec3 raydir = normalize(V);
  vec3 spheredir = spherepos - rayorigin;

  // Perform ray-sphere intersection tests based on the code in Tachyon
  float b = dot(raydir, spheredir);
  float temp = dot(spheredir, spheredir);
  float disc = b*b + sphereradsq - temp;

  // only calculate the nearest intersection, for speed
  if (disc <= 0.0)
    discard; // ray missed sphere entirely, discard fragment

  // calculate closest intersection
  float tnear = b - sqrt(disc);

  if (tnear < 0.0)
    discard;

  // calculate hit point and resulting surface normal
  vec3 pnt = rayorigin + tnear * raydir;
  vec3 N = normalize(pnt - spherepos);

  // Output the ray-sphere intersection point as the fragment depth
  // rather than the depth of the bounding box polygons.
  // The eye coordinate Z value must be transformed to normalized device
  // coordinates before being assigned as the final fragment depth.
  if (vmdprojectionmode == 1) {
    // perspective projection = 0.5 + (hfpn + (f * n / pnt.z)) / diff
    gl_FragDepth = 0.5 + (vmdprojparms[2] + (vmdprojparms[1] * vmdprojparms[(
3];
  } else {
    // orthographic projection = 0.5 + (-hfpn - pnt.z) / diff
    gl_FragDepth = 0.5 + (-vmdprojparms[2] - pnt.z) / vmdprojparms[3];
  }

#ifdef TEXTURE
  // perform texturing operations for volumetric data
  // The only texturing mode that applies to the sphere shader
```

Origins of Computing on GPUs

- Widespread support for programmable shading led researchers to begin experimenting with the use of GPUs for general purpose computation, “GPGPU”
- Early GPGPU efforts used existing graphics APIs to express computation in terms of drawing
- As expected, expressing general computation problems in terms of triangles and pixels and “drawing the answer” is obfuscating and painful to debug…
- Soon researchers began creating dedicated GPU programming tools, starting with Brook and Sh, and ultimately leading to a variety of commercial tools such as RapidMind, CUDA, OpenCL, and others...

GPU Computing

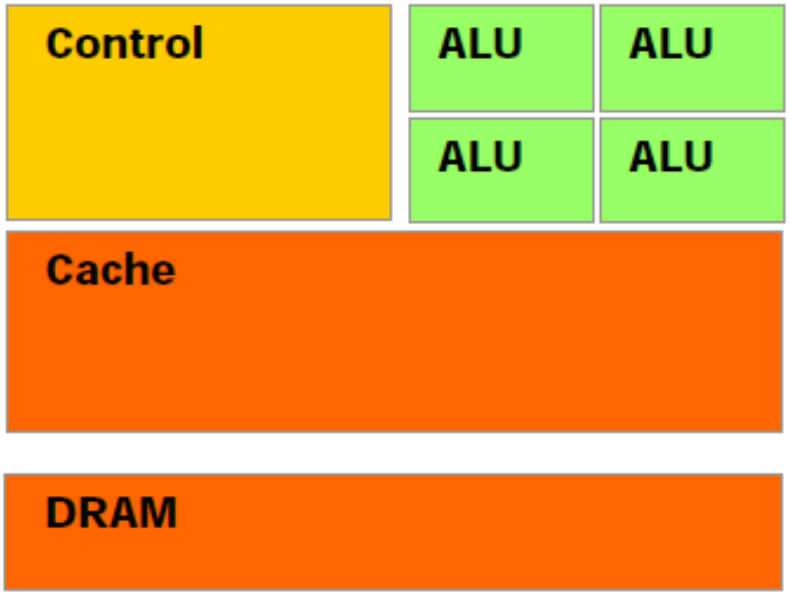
- Commodity devices, omnipresent in modern computers (over a **million** sold per **week**)
- Massively parallel hardware, hundreds of processing units, throughput oriented architecture
- Standard integer and floating point types supported
- Programming tools allow software to be written in dialects of familiar C/C++ and integrated into legacy software
- GPU algorithms are often multicore friendly due to attention paid to data locality and data-parallel work decomposition

What Speedups Can GPUs Achieve?

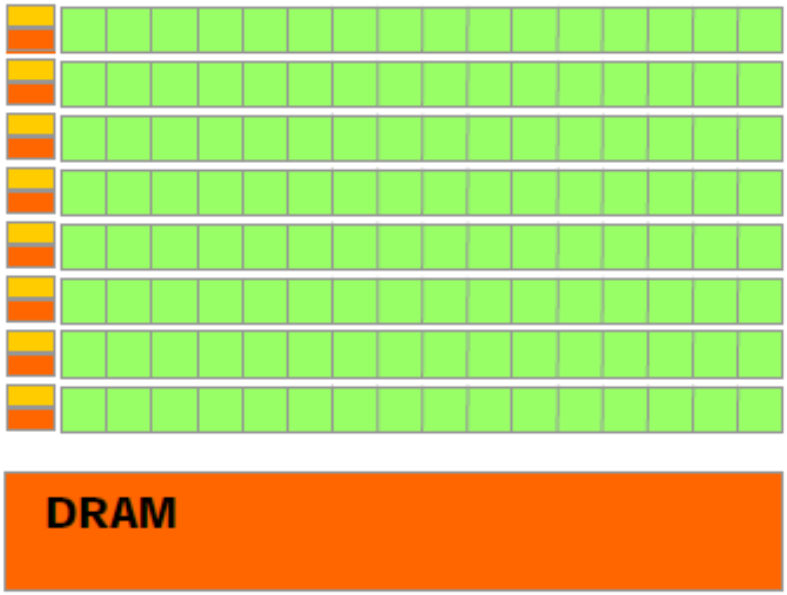
- Single-GPU speedups of **10x** to **30x** vs. one CPU core are common
- Best speedups can reach **100x** or more, attained on codes dominated by floating point arithmetic, especially native GPU machine instructions, e.g. expf(), rsqrtf(), …
- Amdahl's Law can prevent legacy codes from achieving peak speedups with shallow GPU acceleration efforts

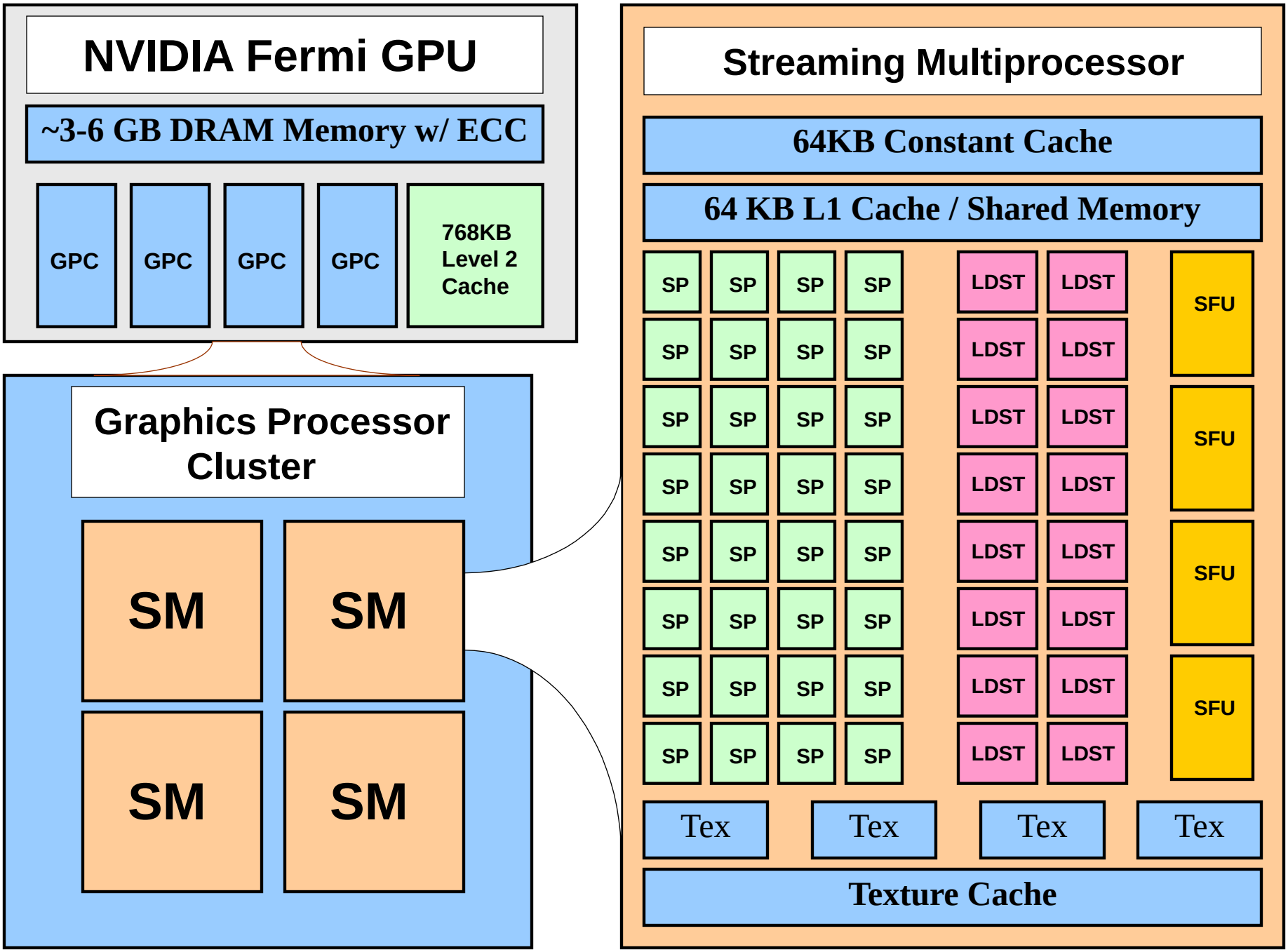
Comparison of CPU and GPU Hardware Architecture

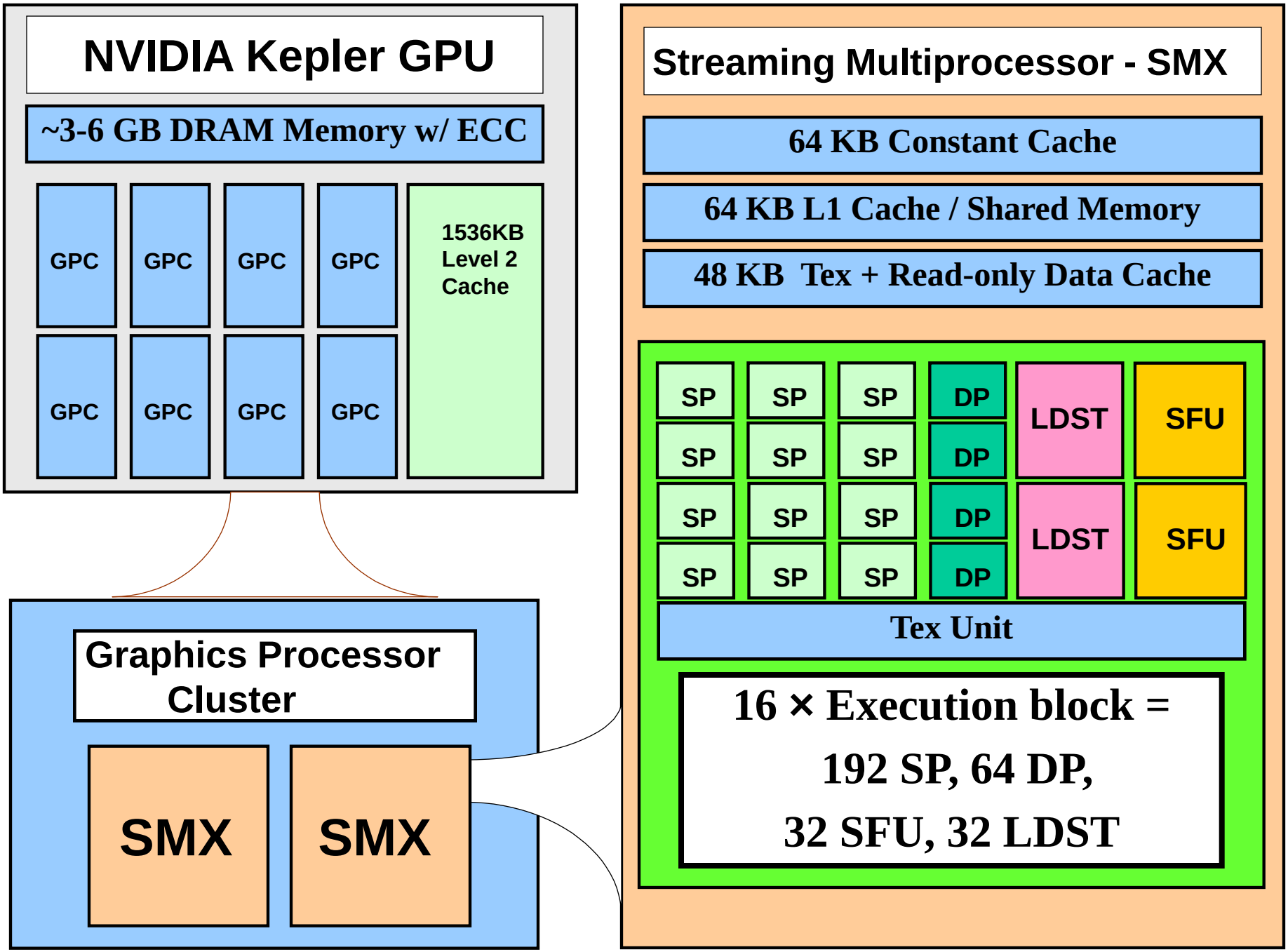
CPU: Cache heavy, focused on individual thread performance



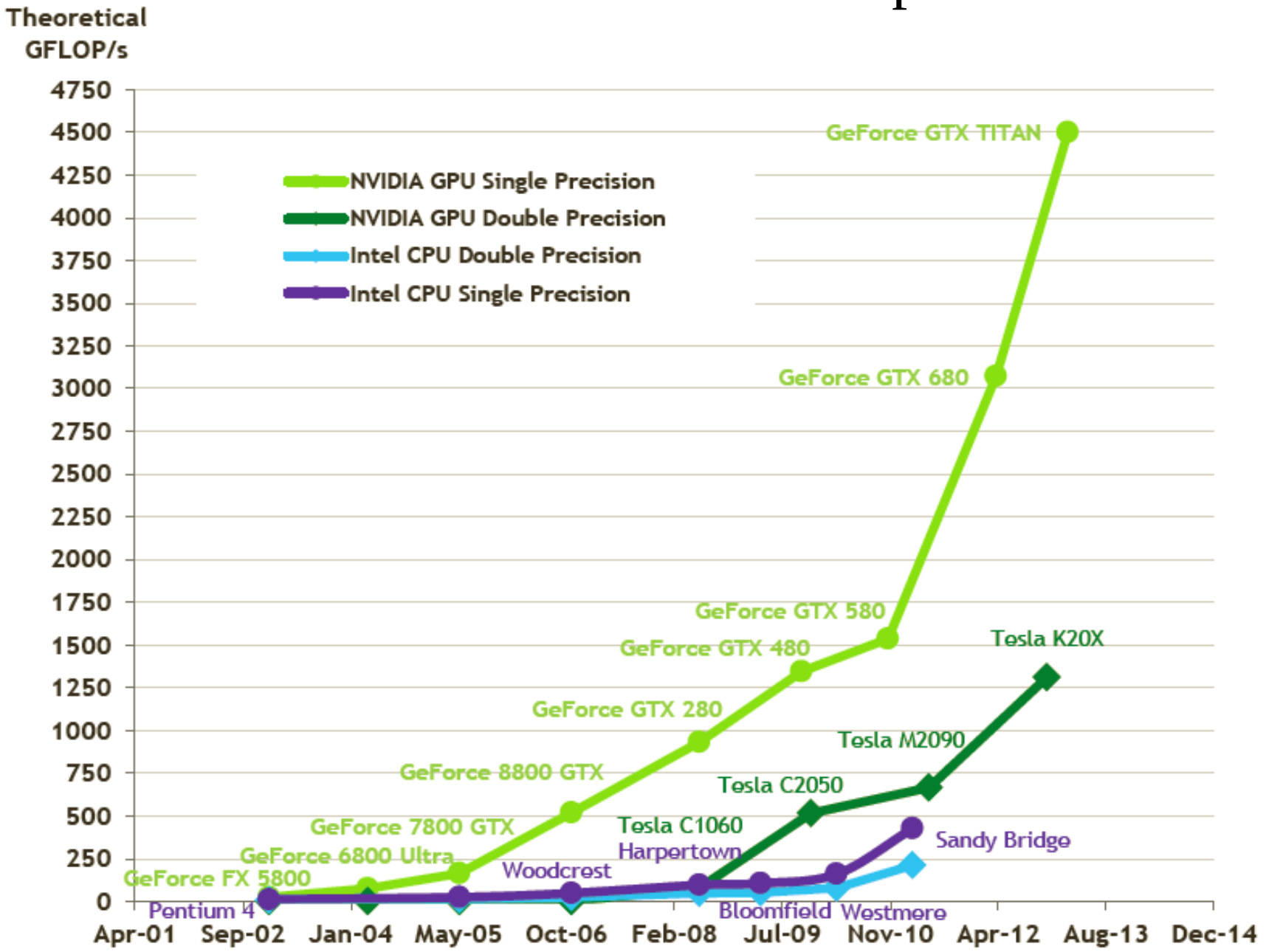
GPU: ALU heavy, massively parallel, throughput oriented



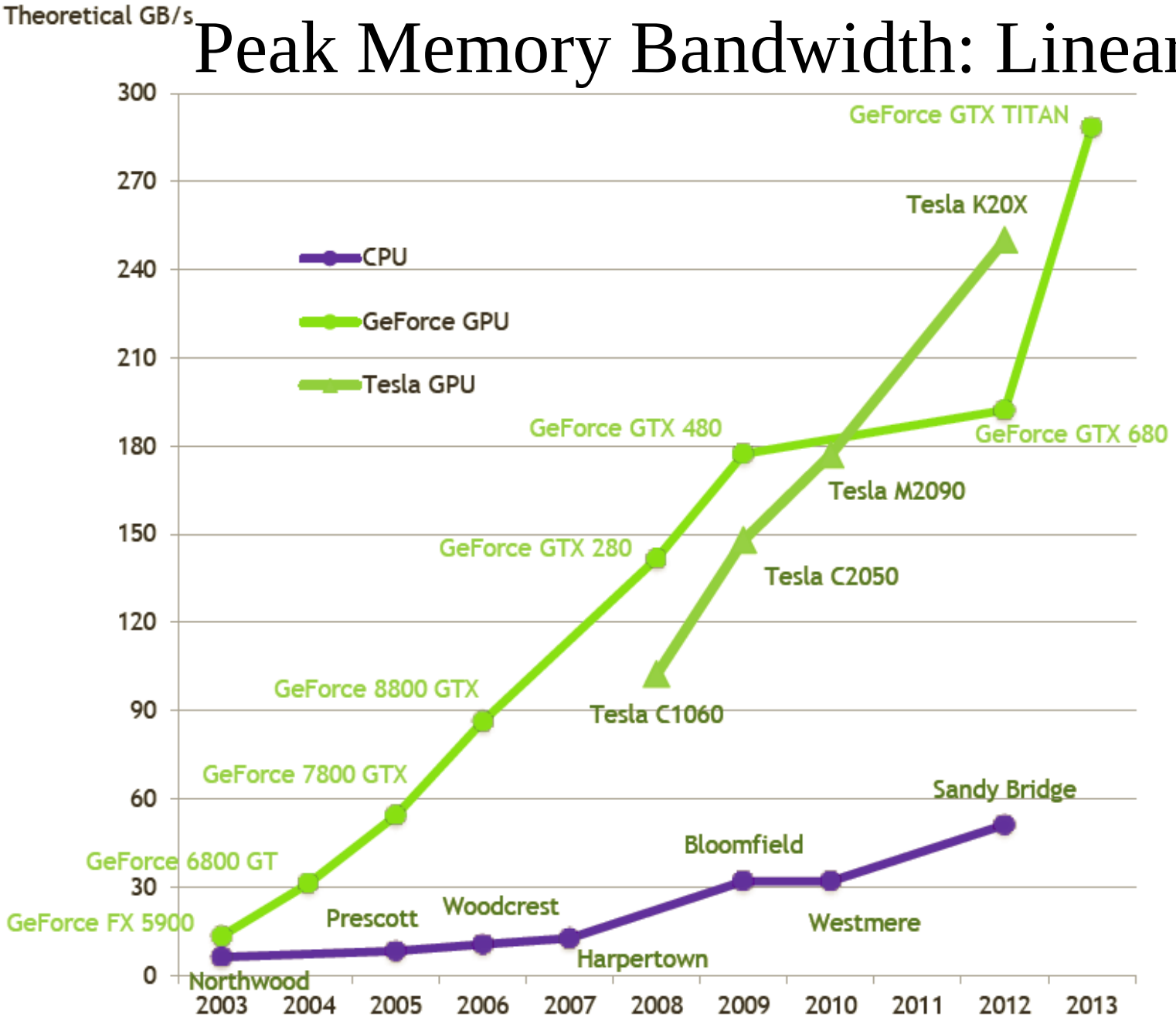

NVIDIA Fermi GPU
~3-6 GB DRAM Memory w/ ECC
GPC
GPC
GPC
GPC
768KB Level 2 Cache
Graphics Processor Cluster
SM
SM
SM
SM
Streaming Multiprocessor
64KB Constant Cache
64 KB L1 Cache / Shared Memory
SP
LDST
SFU
Tex
Texture Cache


NVIDIA Kepler GPU
~3-6 GB DRAM Memory w/ ECC
GPC
GPC
GPC
GPC
GPC
GPC
GPC
GPC
GPC
1536KB Level 2 Cache
Graphics Processor Cluster
SMX
SMX
Streaming Multiprocessor - SMX
64 KB Constant Cache
64 KB L1 Cache / Shared Memory
48 KB Tex + Read-only Data Cache
SP
DP
LDST
SFU
Tex Unit
16 × Execution block =
192 SP, 64 DP,
32 SFU, 32 LDST

Peak Arithmetic Performance: Exponential Trend



Peak Memory Bandwidth: Linear Trend

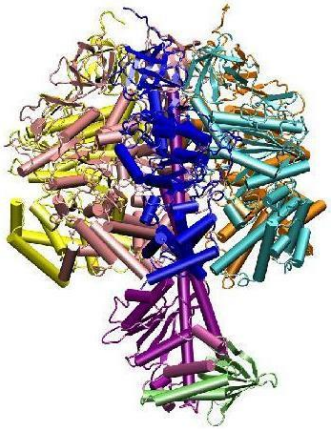


Examples of GPU Accelerated Molecular Modeling Applications

- Molecular dynamics
- Quantum chemistry
- Docking
- Molecular visualization
- MD trajectory analysis
- Simulation preparation
- Many more, see mini-review:
  "GPU-accelerated molecular modeling coming of age"
  (posted on workshop web site)

NAMD: Parallel Molecular Dynamics

2002 Gordon Bell Award



ATP synthase

PSC Lemieux

34,000 Users, 1200 Citations

Computational Biophysics Summer School

Blue Waters Target Application

Illinois Petascale Computing Facility

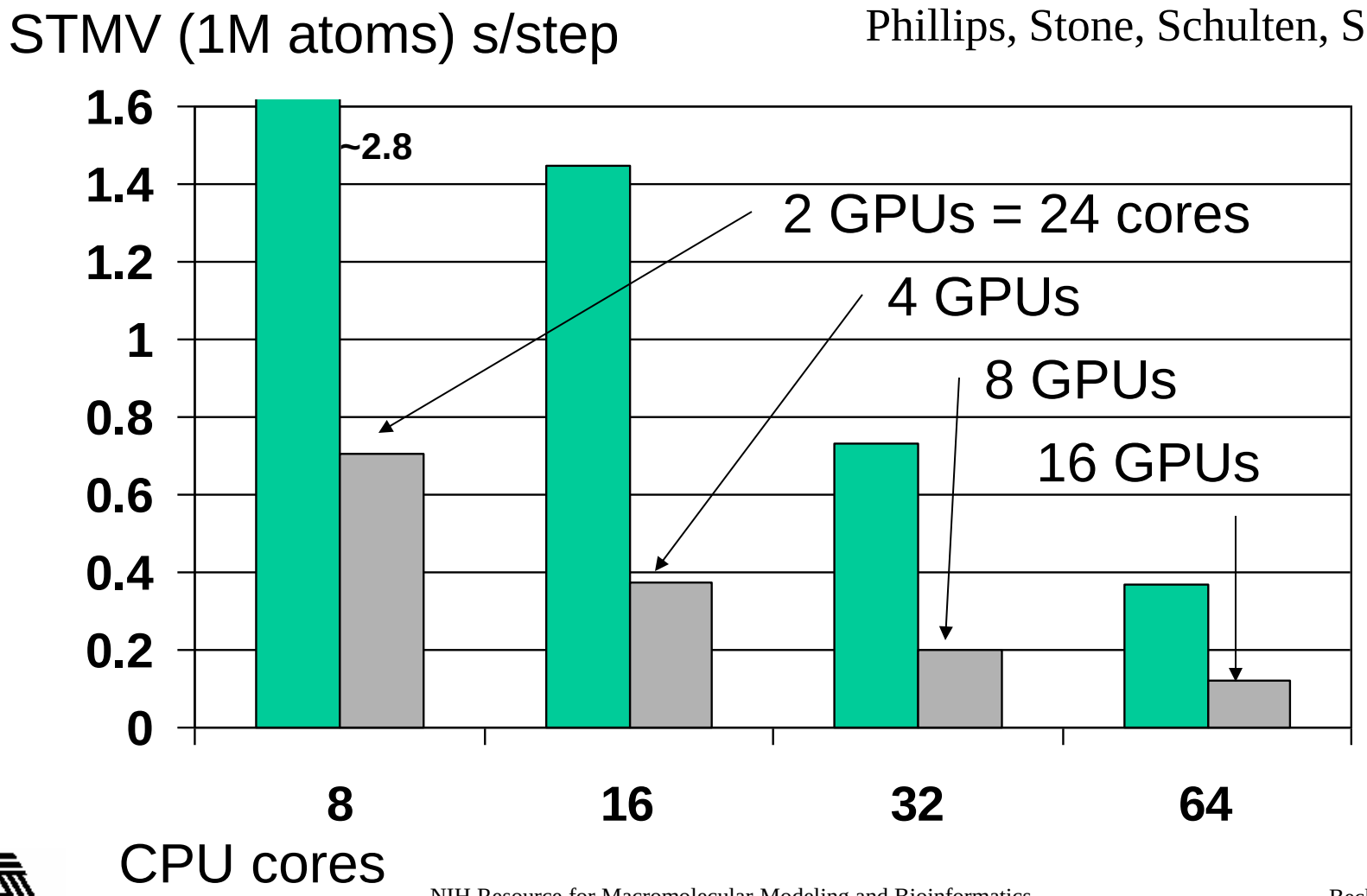
GPU Acceleration

NVIDIA Tesla

NCSA Lincoln

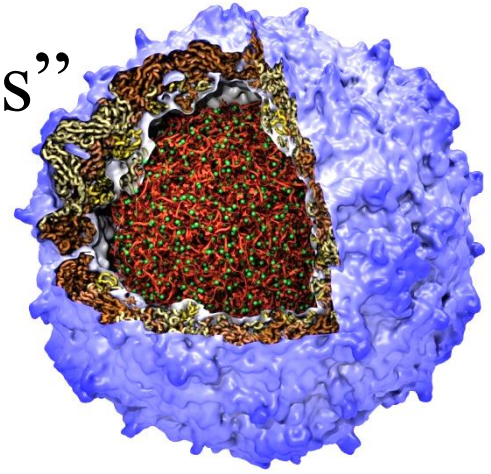
NCSA Lincoln Cluster Performance

(8 Intel cores and 2 NVIDIA Telsa GPUs per node)

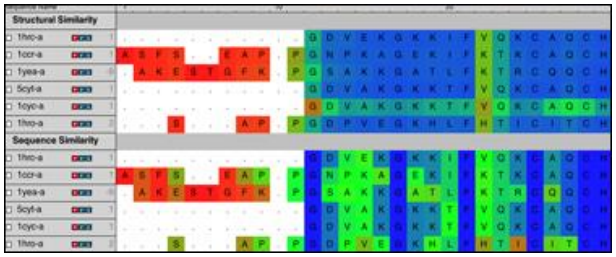


VMD – "Visual Molecular Dynamics"

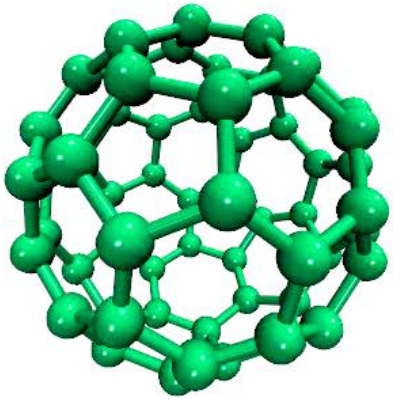
- Visualization and analysis of:
  - molecular dynamics simulations
  - quantum chemistry calculations
  - particle systems and whole cells
  - sequence data
- User extensible w/ scripting and plugins
- http://www.ks.uiuc.edu/Research/vmd/



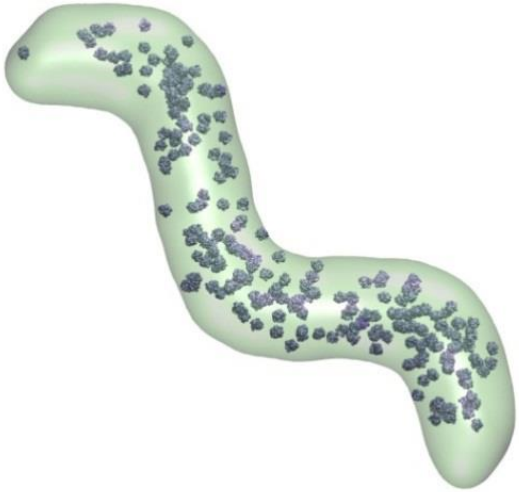
Poliovirus



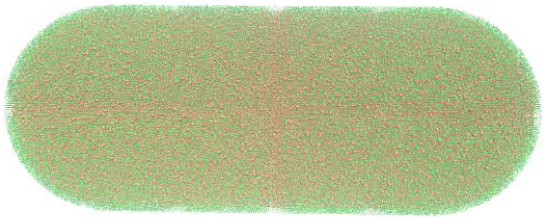
Ribosome Sequences



Electrons in
Vibrating Buckyball



Cellular Tomography,
Cryo-electron Microscopy



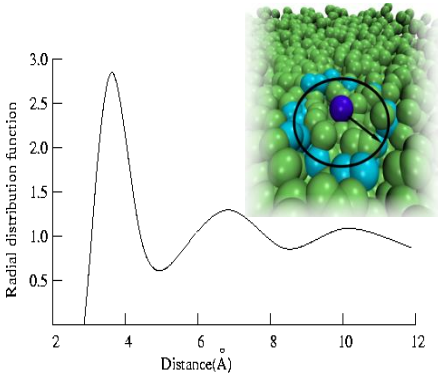
Whole Cell Simulations

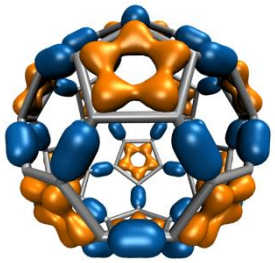
Motivation for GPU Acceleration in VMD

- Increases in supercomputing resources at NSF centers such as NCSA enable increased simulation complexity, fidelity, and longer time scales…
- Drives need for more visualization and analysis capability at the desktop and on clusters
- Desktop use is the most compute-limited scenario, where **GPUs can make a big impact**…
- GPU acceleration provides an opportunity to make some **slow, or batch** calculations capable of being run **interactively, or on-demand…**

GPU Accelerated Trajectory Analysis and Visualization in VMD

GPU-Accelerated Feature	Peak speedup vs. single CPU core
Molecular orbital display	120x
Radial distribution function	92x
Electrostatic field calculation	44x
Molecular surface display	40x
Ion placement	26x
MDFF density map synthesis	26x
Implicit ligand sampling	25x
Root mean squared fluctuation	25x
Radius of gyration	21x
Close contact determination	20x
Dipole moment calculation	15x





Trajectory Analysis on GPU Cluster with MPI-enabled VMD

- Short time-averaged electrostatic field test case (few hundred frames, 700,000 atoms)
- 1:1 CPU/GPU ratio
- Power measured on a single node w/ NCSA monitoring tools
- CPUs-only: 1465 sec, 299 watts
- CPUs+GPUs: 57 sec, 742 watts
- Speedup 25.5 x
- Power efficiency gain: 10.5 x

NCSA Tweet-a-watt power monitoring device

Acknowledgements

- Theoretical and Computational Biophysics Group, University of Illinois at Urbana-Champaign
- Wen-mei Hwu and the IMPACT group at University of Illinois at Urbana-Champaign
- NVIDIA CUDA Center of Excellence, University of Illinois at Urbana-Champaign
- NCSA Innovative Systems Lab
- The CUDA team at NVIDIA
- NIH support: 9P41GM104601