General Purpose Computation on Graphics Processors (GPGPU)

Mike Houston, Stanford University

A little about me

- http://graphics.stanford.edu/~mhouston
- Education:
  - UC San Diego, Computer Science BS
  - Stanford University, Computer Science MS
  - Currently a PhD candidate at Stanford University
- Research
  - Parallel Rendering
  - High performance computing
    - Sequoia
  - Computation on graphics processors (GPGPU)
    - Brook, Folding@Home (GPU)

The world changed over the last year…

- Multiple GPGPU initiatives
  - Vendors without GPGPU talking about it
- A few big apps:
  - Game physics
  - Folding@Home
  - Video processing
  - Finance modeling
  - Biomedical
  - Real-time image processing
- Courses
  - UIUC – ECE 498
  - Supercomputing 2006
  - SIGGRAPH 2006/2007
- Lots of academic research
- Actual GPGPU companies
  - PeakStream
  - RapidMind
  - Accelware
  - …

What can you do on GPUs other than graphics?

- Large matrix/vector operations (BLAS)
- Protein Folding (Molecular Dynamics)
- Finance modeling
- FFT (SETI, signal processing)
- Ray Tracing
- Physics Simulation [cloth, fluid, collision,…]
- Sequence Matching (Hidden Markov Models)
- Speech/Image Recognition (Hidden Markov Models, Neural nets)
- Databases
- Sort/Search
- Medical Imaging (image segmentation, processing)
- And many, many, many more…

GPGPU

http://www.gpgpu.org

Why use GPUs?

- Commodity off the shelf (COTS)
  - In every machine
- Performance
  - Intel Core2 Duo
    - 48 GFLOPS peak
    - 10 GB/s to main memory
  - AMD HD2900XT
    - 475 GFLOPS peak
    - 100 GB/s to video memory
- Lots of Perf/Watt and Perf/$

Rolling your own GPGPU apps

- Lots of information on GPGPU.org
- Use graphics APIs (old way)
- High level languages and systems to make GPGPU easier
  - PeakStream
  - RapidMind
  - Brook
  - CUDA
- Mid-level
  - CTM (CAL)
- Low-level
  - Full control over hardware – CTM (HAL)

GPGPU/Streaming Languages

- Why do you want them?
  - Make programming GPUs easier!
    - Don't need to know OpenGL, DirectX, or ATI/NV extensions
    - Simplify common operations
    - Focus on the algorithm, not on the implementation
- PeakStream
  http://www.peakstreaminc.com
- RapidMind
  Commercial follow-on to Sh
  http://www.rapidmind.net
- Accelerator
  - Microsoft Research
  http://research.microsoft.com/downloads
- Brook
  - Stanford University
  http://brook.sourceforge.net
- CUDA
  - NVIDIA
- CTM
  - AMD

BrookGPU

- History
  - Developed at Stanford University
  - Goal: allow non-graphics users to use GPUs for computation
  - Lots of GPGPU apps written in Brook
- Design
  - C based language with streaming extensions
  - Compiles kernels to DX9, OpenGL, CTM
  - Runtimes (DX9/OpenGL/CTM) handle GPU interaction
- Used for Folding@Home GPU code
  - Large, real application

Streams

- Collection of records requiring similar computation
  - particle positions, voxels, FEM cell, …

```
Ray r<200>;
float3 velocityfield<100,100,100>;
```

- Similar to arrays, but…
  - index operations disallowed: `position[i]`
  - read/write stream operators

```
streamRead (r, r_ptr);
streamWrite (velocityfield, v_ptr);
```

Kernels

- Functions applied to streams
  - similar to for_all construct
  - no dependencies between stream elements

```
kernel void foo (float a<>, float b<>,
                 out float result<>) {
  result = a + b;
}

float a<100>;
float b<100>;
float c<100>;

foo(a,b,c);
```

```
for (i=0; i<100; i++)
      c[i] = a[i]+b[i];
```

Kernels

- Kernel arguments
  - input/output streams

```
kernel void foo (float a<>,
                 float b<>,
                 out float result<>) {
   result = a + b;
}
```

Kernels

- Kernel arguments
  - input/output streams
  - gather streams

```
kernel void foo (..., float array[] ) {
      a = array[i];
}
```

Kernels

- Kernel arguments
  - input/output streams
  - gather streams
  - constant parameters

```
kernel void foo (..., float c ) {
      a = c + b;
}
```

Brook for GPUs

- Open source - Sourceforge
  - CVS tree *much* more up to date (includes CTM support)
- Project Page
  - http://graphics.stanford.edu/projects/brook
- Source
  - http://www.sourceforge.net/projects/brook
- **Paper:**

Brook for GPUs: Stream Computing on Graphics Hardware

Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, Pat Hanrahan

Fly-fishing fly images from The English Fly Fishing Shop

PeakStream

- http://www.peakstreaminc.com
- C/C++ library based
- Extended operators
  - Array constructs
- Just-in-time compilation/optimization
  - Really amazing vectorization and scheduling
- Fairly easy switch for Fortan/Matlab users
- GPUs and multi-core

RapidMind

- http://www.rapidmind.net/
- Embedded within Standard C++
  - No new tools, compilers, preprocessors, etc.
- Portable core
  - Exposes platform specific functionality to also allow tuning for specific platforms
- Integrates with existing programming models
  - Wrap your compute heavy code in extensions
  - Leave rest of the code alone
- GPUs, multi-core, Cell

CUDA - NVIDIA

- C with extensions/limitations
- Similar to Brook, but better matched to G80 hardware
- Complete toolchain
  - Compiler/profiler/debugger
- Standard provided libraries
  - BLAS, FFT
- Support for G80+
- Programmer's guide and lots of hardware information available

- Exposed hardware traits
  - Shared memory
  - Multiple memory levels
  - Register limits
- Data-parallel + multi-threading
  - Exposes the steaming model as data-parallel at the top
  - Exposes multi-threading at the bottom
    - Threads/Warps/Blocks

AMD CTM

- HAL – low level
  - Streaming processor ISA
  - Command buffer control
  - Full memory layout
    - Tiling
    - Addressing
  - Full processor scheduling
- Basically a low level driver interface

- CAL – mid level
  - Stream processor compilers
    - HLSL/PS3/PS4
  - Command buffer assistance
  - Memory management
    - Allocation
    - Layout
  - Multi-GPU handling
- Similar to DX/GL GPGPU programming but semantics to match computation and without the overheads of graphics APIs

AMD + Stream Processing Future

- Brook
  - AMD actively supporting CTM Brook backend for R5XX/R6XX
- CTM – CAL is open platform
  - http://sourceforge.net/projects/amdctm
  - Sits atop HAL for each target
- Streaming extensions to current programming languages
  - Brook-like at first, then expanding
- Multiple platforms
  - GPUs and multi-core with same abstraction through CAL
- Support multiple abstraction levels
  - High-level – Brook support and streaming extensions to standard languages
  - Mid-level (CAL) – memory management, command buffers, etc
  - Low-level (HAL) – direct access to hardware, including ISA

Sequoia – Programming the memory system

- Hierarchical stream programming
  - Similar to multi-level Brook
  - Performance is about programming the memory system
    - Compute is “free”
    - Fast application have data locality at all levels
- We can run on “difficult” systems
  - Cell/PS3
  - Clusters
  - Multi-core/SMP
  - Cluster of Cells in SMP
- http://sequoia.stanford.edu

GPGPU and the HD 2900XT

GPGPU on the HD 2900XT

- 32-bit floating point
  - Tighter IEEE conformance compared to last generation
  - ½ ULP on MUL/ADD,

    1 ULP on MAD
  - Denorm/underflow treated as 0
  - No exception support
- Integer support
  - Much more natural stream addressing
- Scalar floating-point engines
  - 320 of them
- Long programs
  - No instruction limits (up to available memory)
- Branching and Looping
  - Low overhead branching and looping
  - Fine branch granularity:

    ~64 fragments

GPGPU on the HD 2900XT, cont.

- Advanced memory controller
  - Latency hiding for streaming reads and writes to memory
    - With enough math ops you can hide all memory access!
  - Large bandwidth improvement over previous generation
    - Fetch4 performance without the pain
- Large memory addressing
  - 40-bit address space
- Faster upload/download
  - DMA engine

GPGPU on the HD 2900XT, cont.

- Read/Write cache
  - 8KB of on-chip storage
  - Spill to memory
  - Lots of interesting uses
    - Reduction buffer
    - Unlimited writes
    - Register spill
    - …
- Flexibility
  - Unlimited texture reads
  - Scalar engines
  - ***Lots*** of latency hiding
  - Heavy register usage without penalty
    - Fewer threads in flight, but you should have more math

Performance basics for GPGPU – HD 2900XT

- Compute (DX/CTM)
  - 470 GFLOPS (MAD – observed)
    - No games, just a giant MAD kernel
- Offload to GPU (DX)
  - Readback (GPU→CPU): 1.4 GB/s
  - Download (CPU→GPU): 2.4GB/s
- Branch granularity (DX/CTM)
  - ~64 threads

- Memory (CTM)
  - 180 GB/s cache bandwidth
  - 60 GB/s streaming bandwidth
  - 8 GB/s random access
  - 1 cycle latency for a float4 fetch (cache hit)

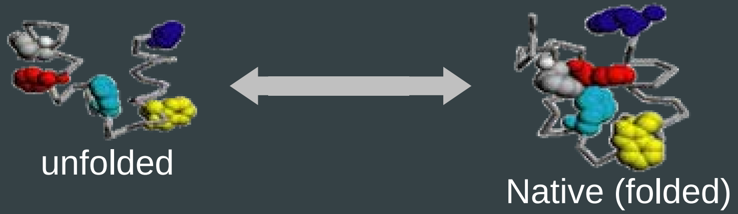
    4 cycle latency for a float4 fetch (streaming)

    1 cycle = 5 four-way MADs

Folding@Home

What is protein folding

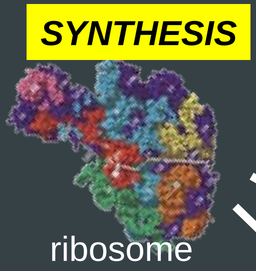
Protein folding: a fundamental biophysical problem

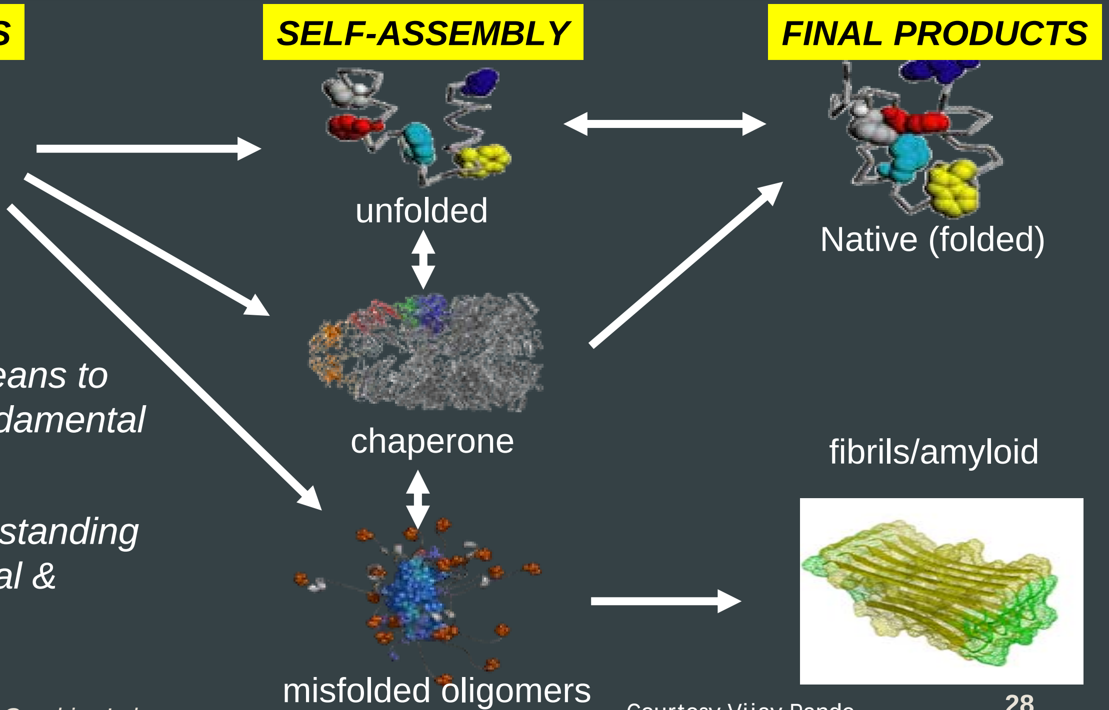


- What are proteins and why do they “fold”?
  - Proteins are molecules in the body which carries out many important functions, such as enzymes and antibodies
  - Before proteins can carryout their function, they must first assemble themselves or “fold”

- Potential impact
  - Design of novel proteins & protein-like heteropolymers
  - Understanding protein misfolding related diseases

Protein folding in the cell





Our goals:

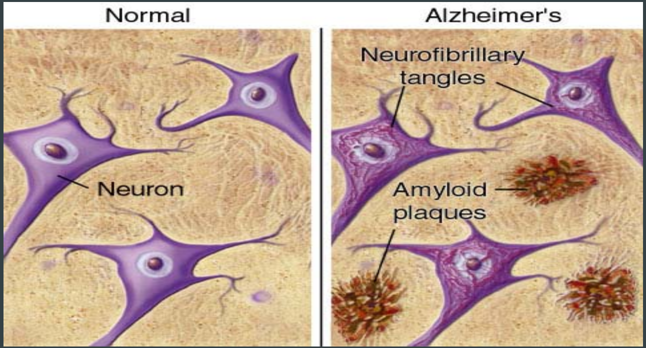
(1) To develop new means to characterize these fundamental biophysical processes

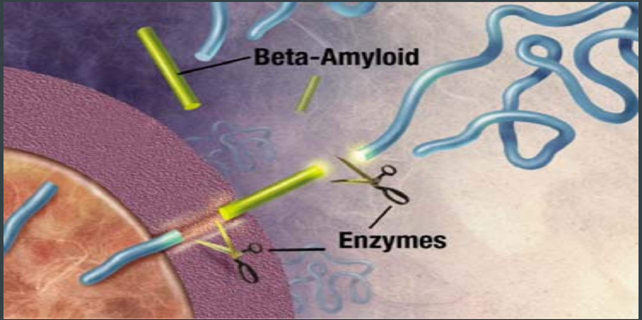
(2) To apply this understanding to challenging biological & biomedical problems

Courtesy Vijay Pande

Molecular nature of Alzheimer’s Disease

- Molecular nature
  - Brain tissue contains deposits of beta-Amyloid peptide (Aβ)
  - amphipathic 39-42 residue fragment of a membrane protein cleaved by secretases
  - Aggregate first into oligomers, then fibrils
- Intrinsically unstructured protein
  - Important new class of proteins
  - Common issue in many protein misfolding related diseases
  - Very different paradigm for structural study

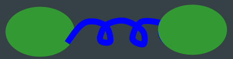




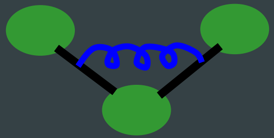
Courtesy Vijay Pande

The 25+ year old dream: use the laws of physics to study biomolecules

- Biomolecules obey the laws of physics
  - Use physical chemistry theory and simulation techniques to study biomolecules
  - All atom model with physical interactions
- Physics-based model: Molecular Dynamics
  - Numerically integrate Newton's equations
  - Choose δt to match timescale ($\delta t \sim 10^{-15}$ sec)
  - Classical model, with parameters to match experiment and/or quantum mechanics calculation
- Comparison to informatic approaches
  - Not limited by protein experiment: examine structures not in PDB (eg aggregates)
  - Possibility to include non-protein molecules (eg small molecules)
  - **<u>BUT, very, very computationally demanding</u>**
  -



bond stretching



angle bending

van der Waals

electrostatics

The 25 year old nightmare- ***The challenge of long timescales***

- Fundamental problem for simulation
  - Proteins fold in micro- to milliseconds
  - Computers can simulate nanoseconds
  - **How can we break this impasse?**

Natural idea: Use multiple processors

- Natural first idea: use multiple CPU's
  - Typical calculation needed requires 1,000,000 days on 1 fast processor
  - How about running on 100,000 CPU's?
  - "Linear scaling" would suggest this would take just 10 days with 100,000 CPU's

- Life isn't that easy
  - Most codes can't scale this way
  - In 2000, we suggested a method to solve this problem
  - Newest version (2005), allows for a very efficient use of 100K CPU's

Can 2000 grad students complete a PhD's worth of research in 1 day?

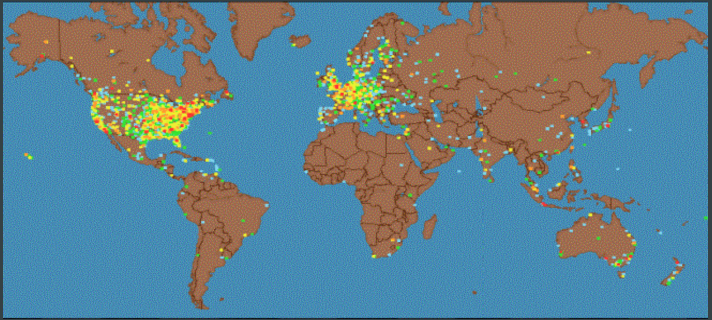
Most likely not! One must design novel ways to utilize large scale resources in efficient ways.

Courtesy Vijay Pande

Folding@Home: Computational resources from the future, realized today

- Very powerful computational resource
  - ~700 Teraflops sustained performance
  - >2,000,000 total processors; ~250,000 active
- Big impact
  - Break the micro- to millisecond barrier
  - What used to take 1M days, now takes a week or two
  - Enables key research previously impossible
  - These new methods are also starting to be used by other groups for folding study

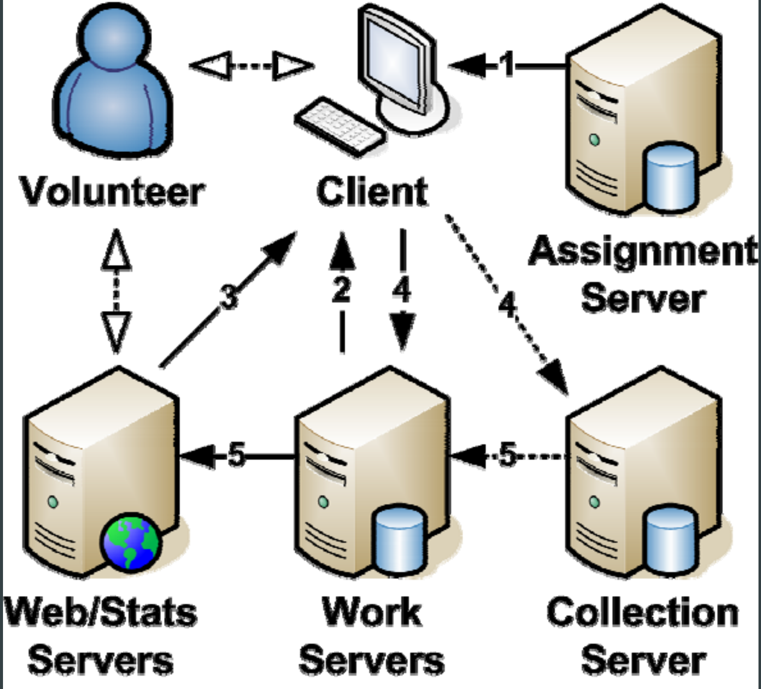
Folding@home distributed computing



>250,000 active CPUs over the world (CPU locations from IP address)

Courtesy Vijay Pande

How does this all work?

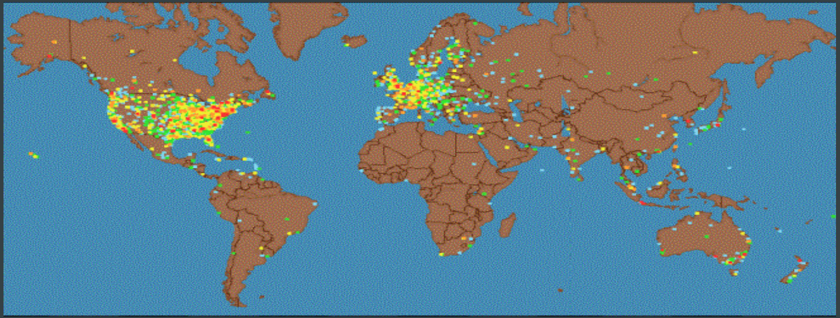


Courtesy Vijay Pande

Who’s Folding?

Folding@Home

~100,000 CPUs over the world (CPU locations from IP address)



Earth@Night

Electricity as distributed in the world (from NASA satellite data)

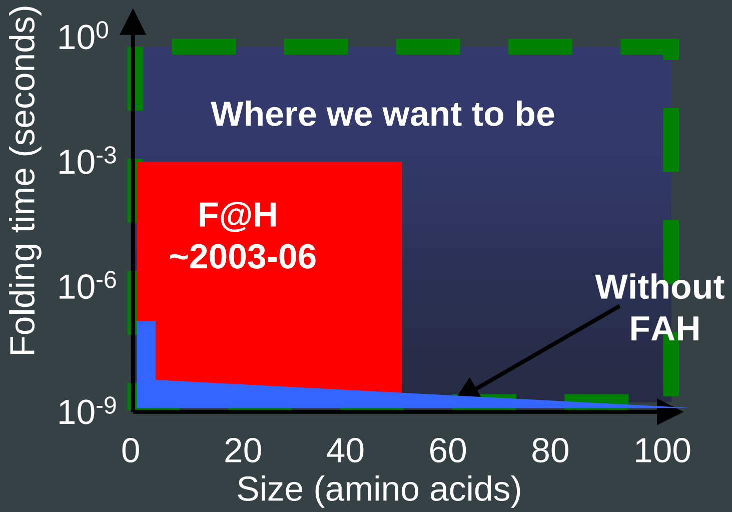
Courtesy Vijay Pande

Folding@Home communities – small subset

Next steps: new challenges

- What we can do
  - Somewhat long timescales (microseconds to milliseconds)
  - Somewhat large peptides & proteins (30-50 residues)
  - Allows us to directly test the relevant chemical detail in these biophysical problems

- Where we want to be
  - Longer timescales (seconds?)
  - Larger systems (~100 residues)
  - Yet not sacrificing details



Streaming processors on FAH

- New processors
  - Originally special purpose (eg for graphics, games)
  - High flops: ~500GFLOPS peak programmable, 32-bit FLOPs

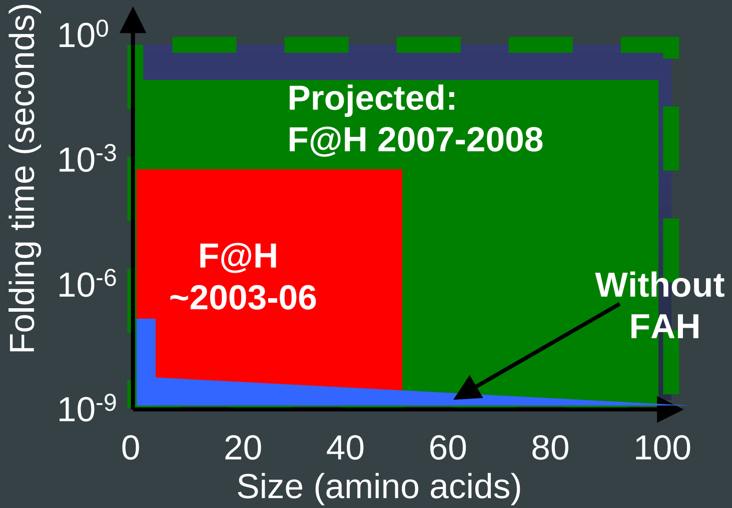
- Utilize for scientific computing
  - We have ported our MD code to GPUs & Cell (PS3)
  - Partnership with Sony and AMD
  - 20x to 40x speed increase

- Impact on calculations
  - Critical for MSM's: *longer trajectories, not just more of them*
  - Tackle more complex systems

Courtesy Vijay Pande

Folding@Home's next steps

- New methods
  - New processors and new advances in implicit solvation models allow for a ~20x to 500x speedup
  - Achieve simulations on the second timescale
- Impact: sampling may soon not be a problem
  - "@Home" not needed to simulate sub-millisecond timescale
  - More complex problems would be in reach with distributed computing
  - Turn our attention to further improving models (experiment key for these next steps)



With these new capabilities, we would be able to simulate the folding of a significant fraction of protein domains

Courtesy Vijay Pande

GPUs, SMPs, PS3s (oh my!)

- Performance
  - GPU > PS3 > SMP > Single-core
- Flexibility
  - Single-core > SMP > PS3 > GPU

- GPUs are ***really*** fast
  - 20-40x a single core P4
- But they can only run a subset of the research
  - Implicit water models

- Points are awarded with this in consideration

Current status – April 20, 2007

http://fah-web.stanford.edu/cgi-bin/main.py?qtype=osstats

Client type	Current TFLOPS*	Active Processors
Windows	177	185870
Mac OS X/PPC	8	10394
Mac OS X/Intel	14	4600
Linux	47	27373
GPU	56	947
Playstation3	409	31217
Total	711	260401

*TFLOPS is actual flops from software cores, **not** peak values

For More Information

- Main site
  - http://folding.stanford.edu
- GPU information
  - http://folding.stanford.edu/FAQ-ATI.html
- Petaflop Initiative
  - Stream processors
    - GPUs
    - PS3 (Cell)
  - http://folding.stanford.edu/FAQ-FPI.html

Folding on HD 2900XT

- Support will ship with next release
- Force calculation 2.2X(!) faster than R580
- Overall application ~45% faster
  - Amdahl's law strikes again
    - Supporting kernels unoptimized
    - CPU overhead
  - Little tuning time so far on R6XX
- We can now explore more complex algorithms
  - Larger register requirements
  - Integer support – random number generation
  - More bandwidth

Future Folding@Home GPGPU

- Improvements
  - Lower CPU overhead
    - Brook DX9 backend + core changes
  - CTM through Brook
    - Lower overhead/finer tuning
    - Linux support much easier
    - Better stability – users can upgrade drivers for games without breaking anything
  - More GPUs supported
- New cores
  - Back-port some of what we learned from Cell ports
  - Several new models
  - Many more work units
  - Revisit other models running well on SMP/Cell

Acknowledgements – Folding@Home research group

Current members:
Adam Beberg
Relly Brandman
Kim Branson
Jeremy England
Dan Ensign
Guha Jayachandran
Rajdas Jaykumar
Peter Kasson
Nick Kelley
Del Lucent
Edgar Luttmann
Sanghyun Park
Paula Petrone
Alex Robertson
Nina Singhal
Eric Sorin
Vishal Vaidyananthan

Former members:
Ian Baker
Jim Caldwell
Lillian Chong
Sidney Elmer
Mark Engelhardt
Amit Garg
Siraj Khaliq
Stefan Larson
Sung Joo Lee
Bradley Nakatani
Young Min Rhee
Michael Shirts
Chris Snow
Abhay Sukumaran
Bojan Zagrovic

Coauthors (collaborators):
Steve Boxer (Stanford)
Axel Brunger (Stanford)
John Desjarlais (Xencor)
Seb Doniach (Stanford)
Hide Fujutani (Fujitsu)
Feng Gai (U. Penn.)
Martin Gruelebe (UIUC)
Leo Guibas (Stanford)
Steve Hagen (U. Florida)
Dan Herschlag (Stanford)
Pat Hanrahan (Stanford)
Teri Klein (Stanford)
Ron Kopito (Stanford)
Grant Krafft (Acumen)
Erik Lindahl (Stockholm)
Jed Pitera (IBM)
Kevin Plaxco (UCSB)
Guillermo Sapiro (U. Minn.)
Tobin Sosnick (U. Chicago)
Bill Swope (IBM)

***Funding**: Folding@Home donors, NSF, NIH, Dreyfus, ACS PRF, Acumen Pharmaceuticals, Intel, Google, Apple, Terman*

Making GPGPU easier

What GPGPU needs from vendors – what we got

- More information
  - Shader ISA ★
  - Latency information
  - GPGPU Programming guide (floating point) ★
- Direct access to the hardware/Compute APIs ★
- Fast transfer to and from GPU
  - Non-blocking
- Consistent graphics drivers ★
  - Some optimizations for games hurt GPGPU performance

What GPGPU needs from vendors

- More information
  - Latency information
  - Memory system information
- Fast transfer to and from GPU
  - Non-blocking
  - DMA driven
- Consistent drivers
  - Stable drivers for games AND computation
- High-level <u>and</u> low-level access
  - High-level: get more people using GPUs, make it easy to approach
  - Low-level: let us tune to our heart's content!
    - Remove shader compiler and game optimizations from breaking our code
- More software companies
  - PeakStream and RapidMind a good start

What GPGPU needs from the community

- Data Parallel programming languages
  - Need more exploration
- “GCC” for GPUs/streaming processors
  - We almost have enough information for this
- Parallel data structures
  - Still really hard…
- More applications
  - What will make the average user care about GPGPU?
    - Folding@Home! ;-)
  - What else can we make data parallel and run fast?

Questions?

- I'll also be around after the talk
- Email: mhouston@stanford.edu
- Web: http://graphics.stanford.edu/~mhouston

- For lots of great GPGPU information:
  - GPGPU.org (http://www.gpgpu.org)