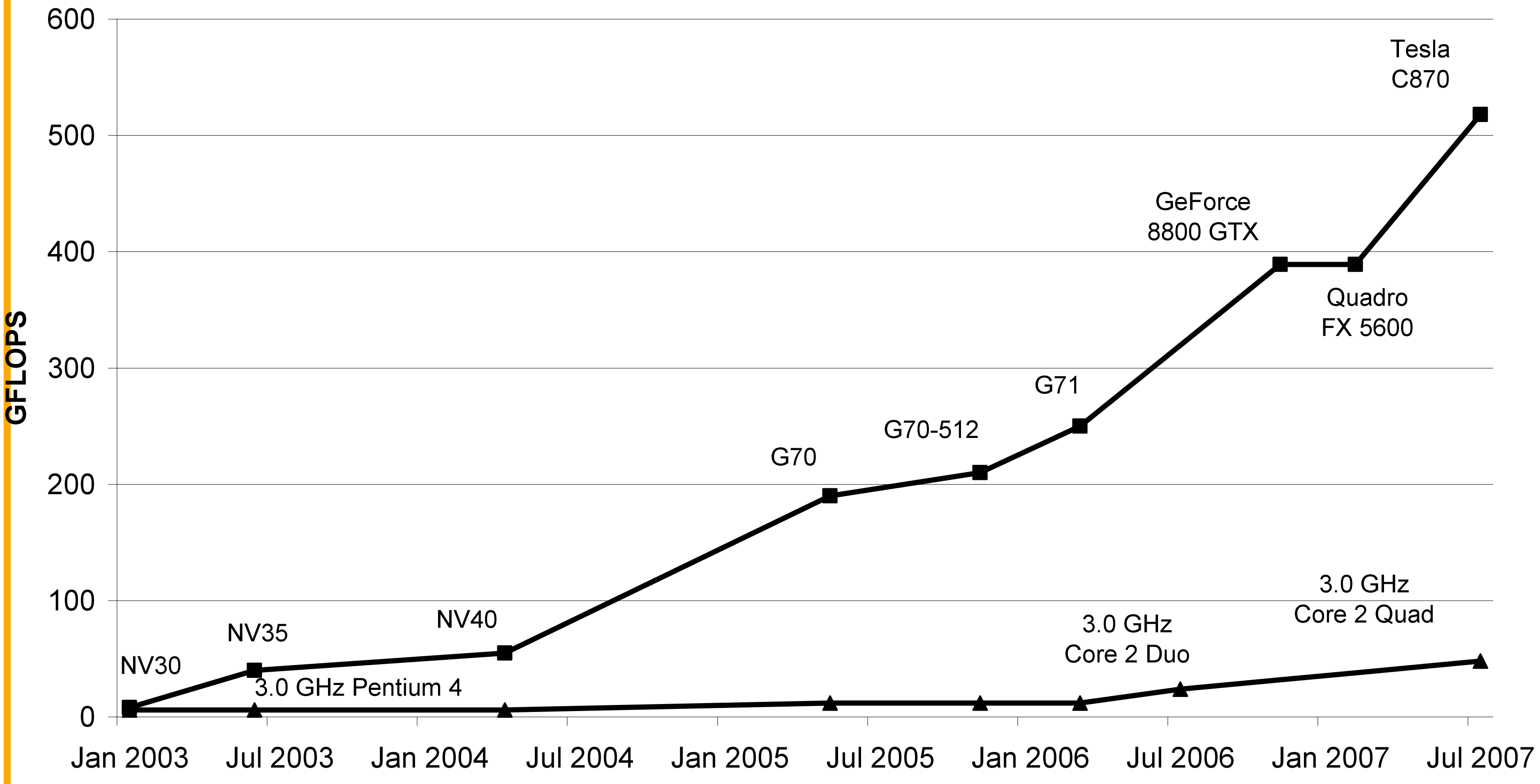
VSCSE Summer School 2008

Accelerators for Science and Engineering Applications: GPUs and Multi-cores

Lecture 1
Introduction and Motivation

Urbana, Illinois, August 18-22, 2008

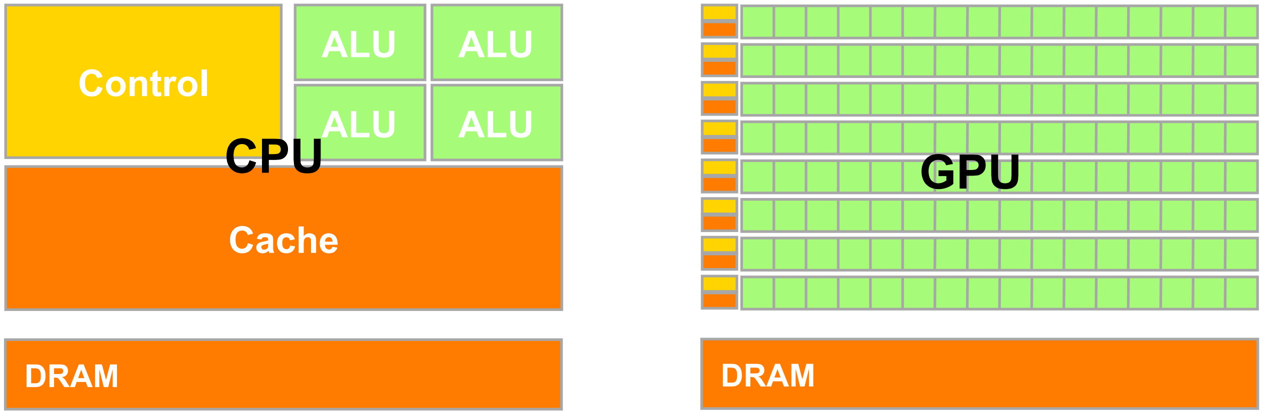
What is driving the many-cores?



[1] Based on slide 7 of S. Green, “GPU Physics,” SIGGRAPH 2007 GPGPU Course. http://www.gpgpu.org/s2007/slides/15-GPGPU-physics.pdf

Design philosophies are different.

- The GPU is specialized for compute-intensive, massively data parallel computation (exactly what graphics rendering is about)
  - So, more transistors can be devoted to data processing rather than **data caching** and **flow control**



- The fast-growing video game industry exerts strong economic pressure for constant innovation

This is not your advisor’s parallel computer!

- Significant application-level speedup over uni-processor execution
  - No more “killer micros”
- Easy entrance
  - An initial, naïve code typically get at least 2-3X speedup
- Wide availability to end users
  - available on laptops, desktops, clusters, super-computers
- Numerical precision and accuracy
  - IEEE floating-point and double precision
- Strong scaling roadmap

GPU Computing Scaling

- Laptops, desktops, workstations, servers, clusters – (cell phones? iPods?)
- UIUC has built a 16-node GPU cluster
  - Peak performance 32.5 TFLOPS (SP)
  - For science and engineering apps
- UIUC is planning a 32-node GPU cluster for Summer 2008
  - Estimated peak performance 130 TFLOPS (SP) and 16 TFLOPS (DP)
- UIUC is planning a 400-GPU upgrade to the NSCA Abe production cluster in Fall 2008

GeForce 8800

Tesla D870

Tesla S870

How much computing power is enough?

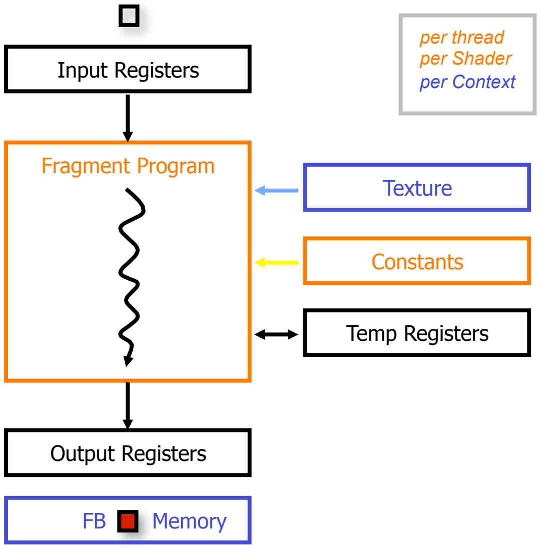
- Each 10X jump in computing power motivates new ways of computing
  - Many apps have approximations or omissions that arose from limitations in computing power
  - Every 10x jump in performance allows app developers to rethink their fundamental assumptions and strategies
  - Example: graphics, medical imaging, physics simulation, etc.
- Each 2-3X allows addition new, innovative features to applications

Historic GPGPU Movement

- General Purpose computation using GPU in applications other than 3D graphics
  - GPU accelerates critical path of application
- Data parallel algorithms leverage GPU attributes
  - Large data arrays, streaming throughput
  - Fine-grain SIMD parallelism
  - Low-latency floating point (FP) computation
- Applications – see //GPGPU.org
  - Game effects (FX) physics, image processing
  - Physical modeling, computational engineering, matrix algebra, convolution, correlation, sorting

Historic GPGPU Constraints

- Dealing with graphics API
  - Working with the corner cases of the graphics API
- Addressing modes
  - Limited texture size/dimension
- Shader capabilities
  - Limited outputs
- Instruction sets
  - Lack of Integer & bit ops
- Communication limited
  - No interaction between pixels
  - No scatter store ability - a[i] = p



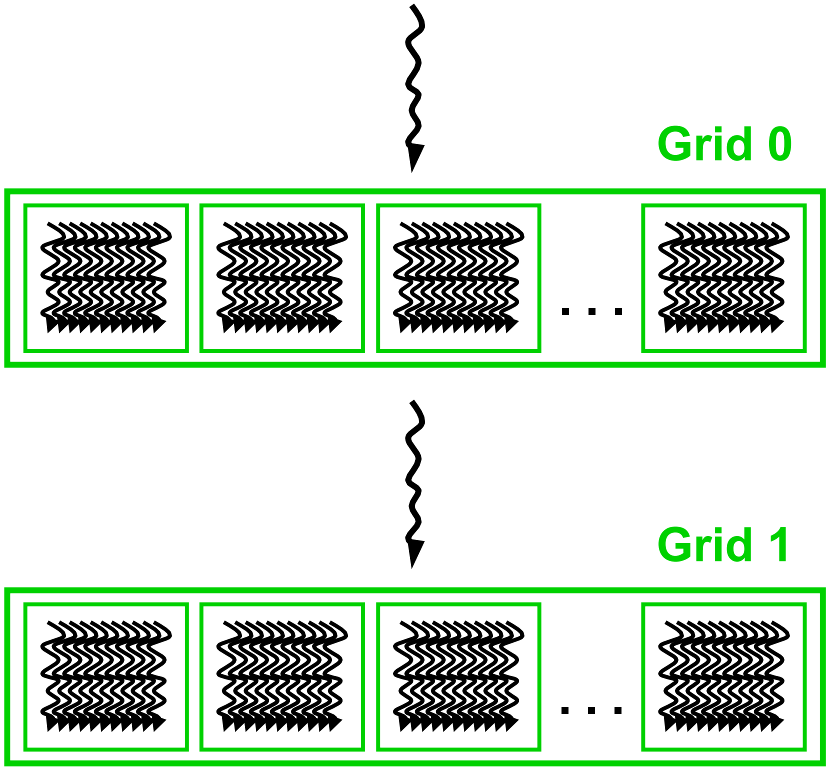
These have all changed with CUDA!

What is the GPU Good at?

- The GPU is good at data-parallel processing
  - The same computation executed on many data elements in parallel – low control flow overhead with **high SP floating point arithmetic intensity**
  - Many calculations per memory access
  - Currently also need high floating point to integer ratio
- High floating-point arithmetic intensity and many data elements mean that memory access latency can be hidden with calculations instead of big data caches – Still need to avoid bandwidth saturation!

CUDA - No more shader functions.

- Integrated CPU+GPU application C program
  - Serial or modestly parallel C code executes on CPU
  - Highly parallel SPMD kernel C code executes on GPU



CPU Serial Code

GPU Parallel Kernel

KernelA<<< nBlk, nTid >>>(args);

CPU Serial Code

GPU Parallel Kernel

KernelB<<< nBlk, nTid >>>(args);

It is about applications!

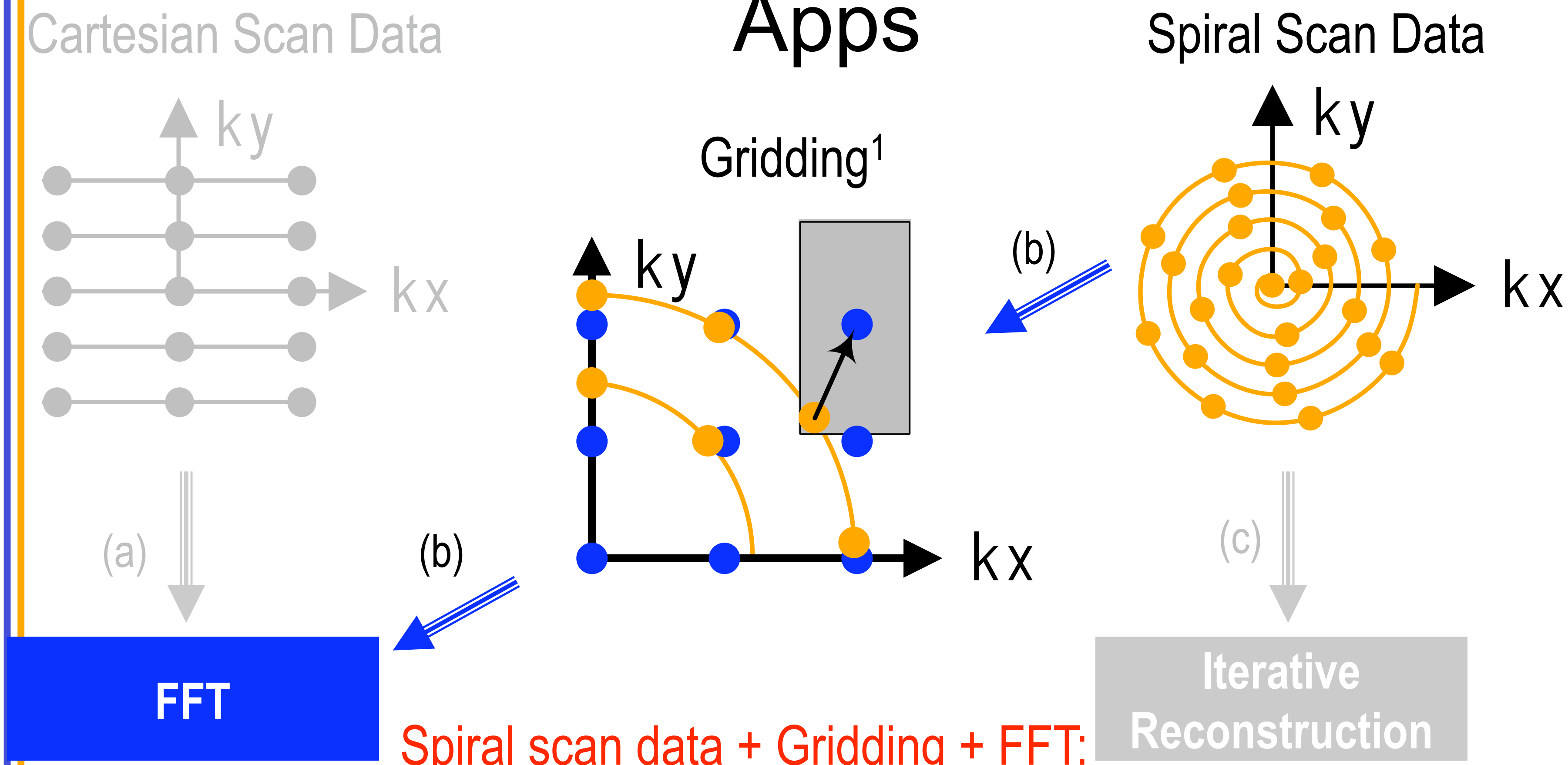
Vision, Imaging, VACE, HCI, Modeling and Simulation…

Science and Engineering Application Speedup

App.	Archit. Bottleneck	Simult. T	Kernel X	App X
H.264	Registers, global memory latency	3,936	20.2	1.5
LBM	Shared memory capacity	3,200	12.5	12.3
RC5-72	Registers	3,072	17.1	11.0
FEM	Global memory bandwidth	4,096	11.0	10.1
RPES	Instruction issue rate	4,096	210.0	79.4
PNS	Global memory capacity	2,048	24.0	23.7
LINPACK	Global memory bandwidth, CPU-GPU data transfer	12,288	19.4	11.8
TRACF	Shared memory capacity	4,096	60.2	21.6
FDTD	Global memory bandwidth	1,365	10.5	1.2
MRI-FHD	Instruction issue rate	8,192	23.0	23.0

[HKR HotChips-2007]

Massive Speedup can Revolutionize Apps



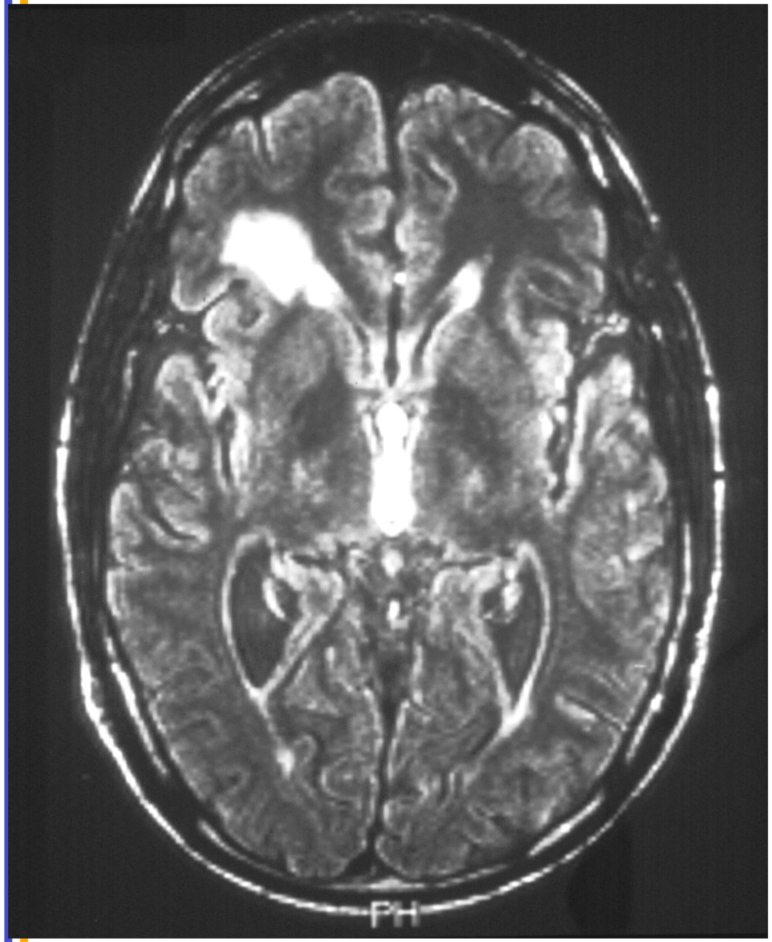
Spiral scan data + Gridding + FFT:

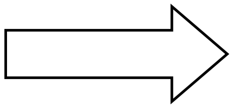
Faster scan reduces artifacts, averaging increases SNR.

Reconstruction requires little computation.

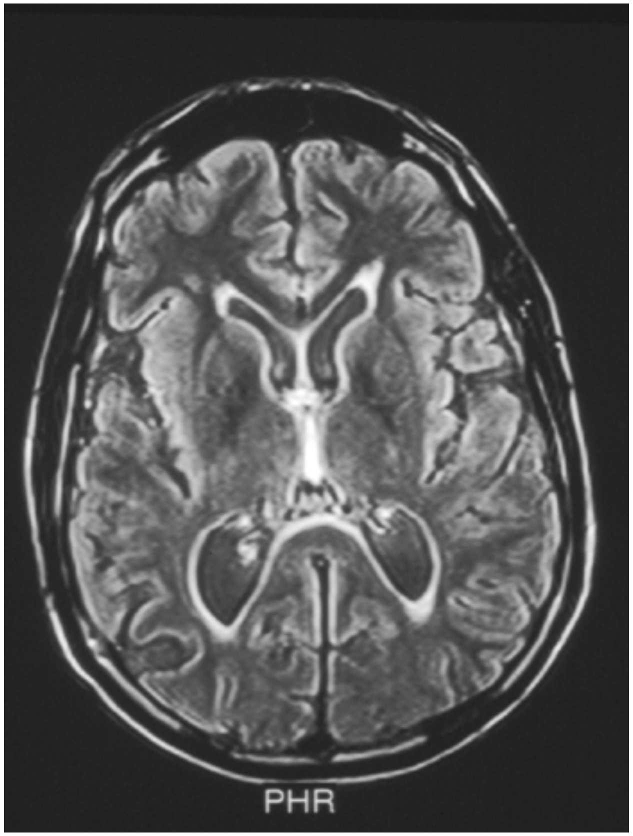
[1] Based on Fig 1 of Lustig et al, Fast Spiral Fourier Transform for Iterative MR Image Reconstruction, IEEE Int'l Symp. on Biomedical Imaging, 2004

Chemo Therapy Monitoring

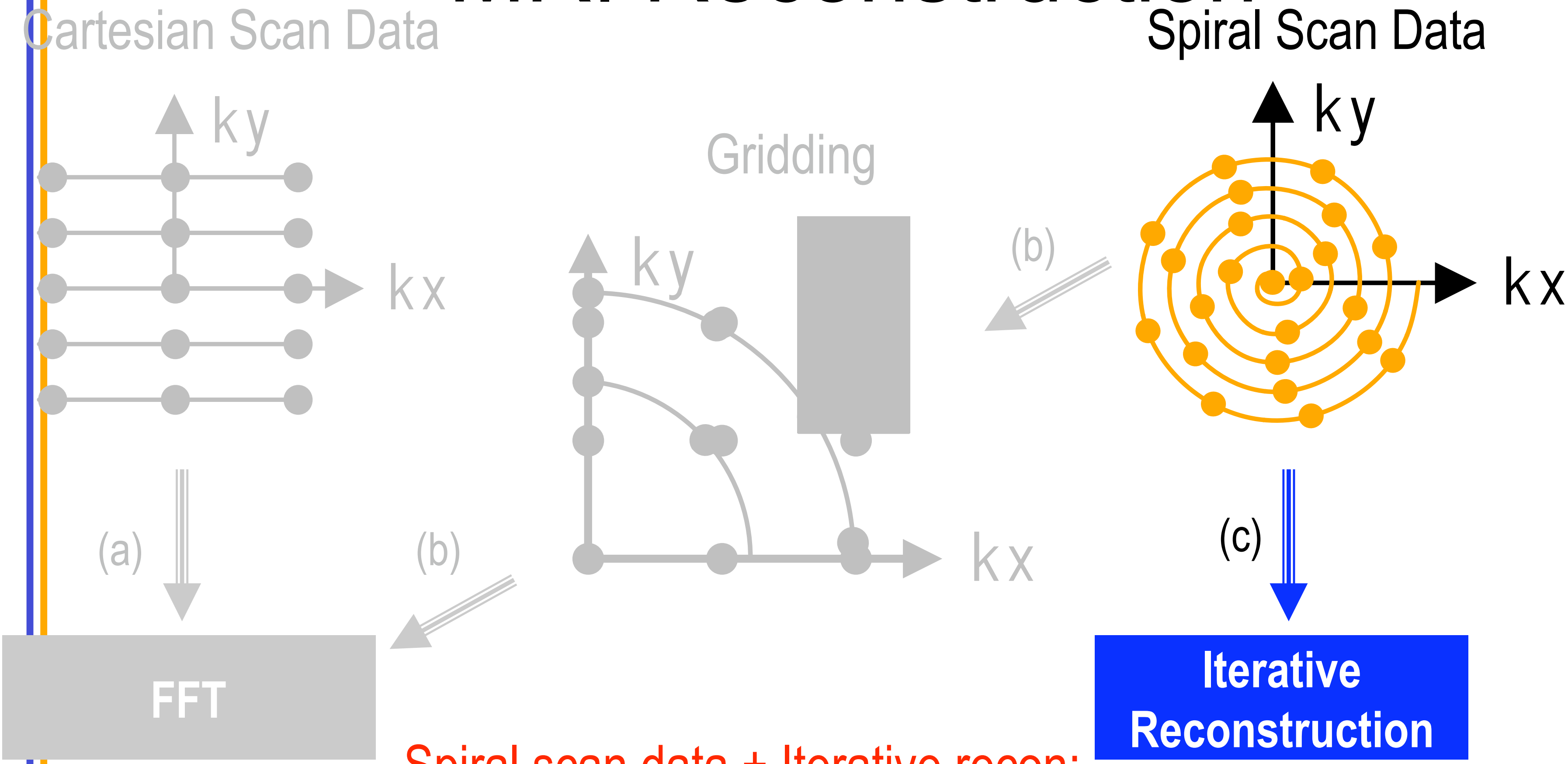




6-12 weeks
(hopefully)



MRI Reconstruction

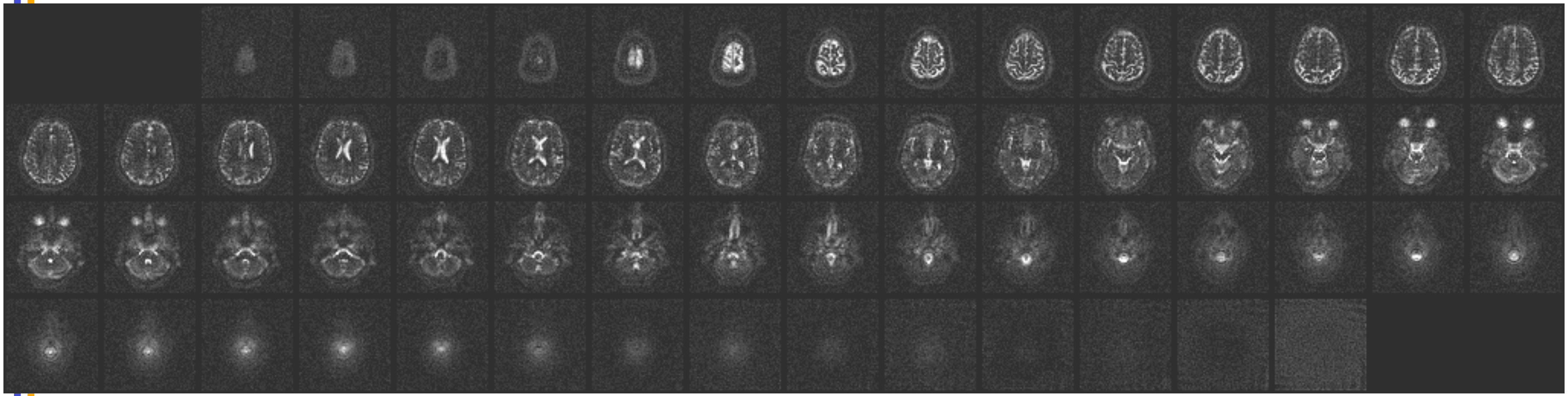


Spiral scan data + Iterative recon:

Fast scan reduces artifacts, iterative reconstruction increases SNR.

Reconstruction requires a lot of computation.

An Exciting Revolution - Sodium Map of



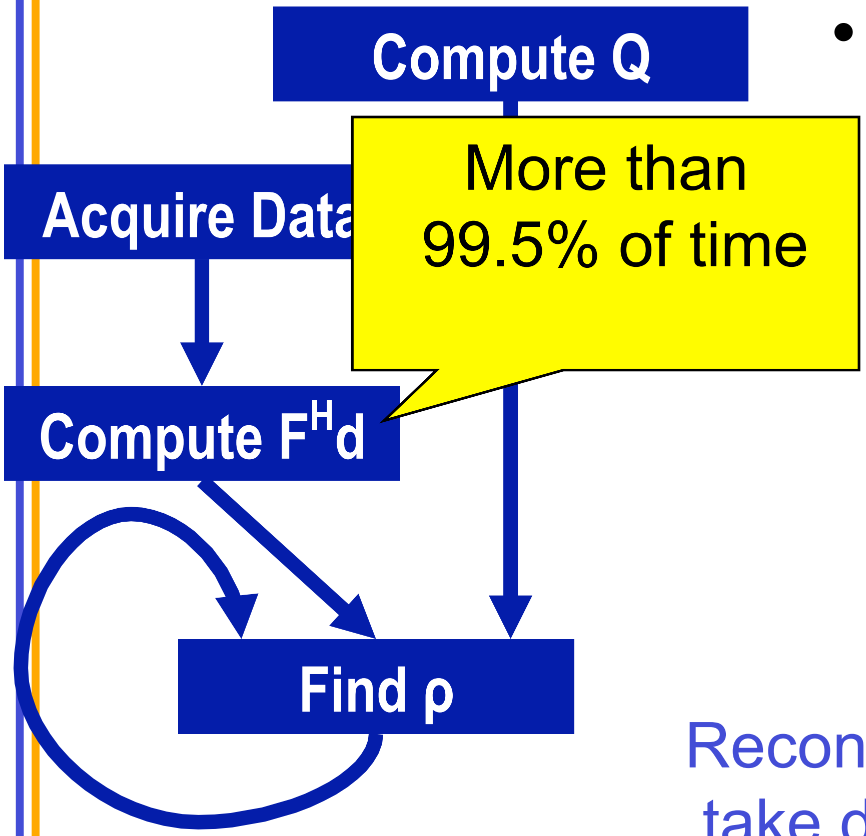
- Images of sodium in the brain
  - Requires powerful scanner (9.4 Tesla)
  - 2000x less abundant than water, the main modality of MRI today
  - Very large number of samples for increased SNR
  - Requires high-quality reconstruction

- Study of brain-cell viability before anatomic changes occur in stroke and cancer treatment – within days!

Courtesy of Keith Thulborn and Ian Atkinson, Center for MR Research, University of Illinois at Chicago

Advanced MRI Reconstruction

$$(F^H F + \lambda W^H W)\rho = F^H d$$



More than 99.5% of time

- Q: partial $F^H F$ and depends only on scanner setup

  $F^H d$ depends on scan data

  ρ found using linear solver
  - $F^H F$ computed once per iteration; depends on Q, $F^H d$
  - $\lambda W^H W$ incorporates anatomical constraints

Reconstruction of a 64^3 image used to take days using MatLab!

Haldar, et al, “Anatomically-constrained reconstruction from noisy data,” MR in Medicine.

Code

```
for (p = 0; p < numP; p++) {
  for (d = 0; d < numD; d++) {
    exp = 2*PI*(kx[d] * x[p] +
                ky[d] * y[p] +
                kz[d] * z[p]);
    cArg = cos(exp);
    sArg = sin(exp);
    rFhD[p] += rRho[d]*cArg –
               iRho[d]*sArg;
    iFhD[p] += iRho[d]*cArg +
               rRho[d]*sArg;
  }
}
```

Traditional C

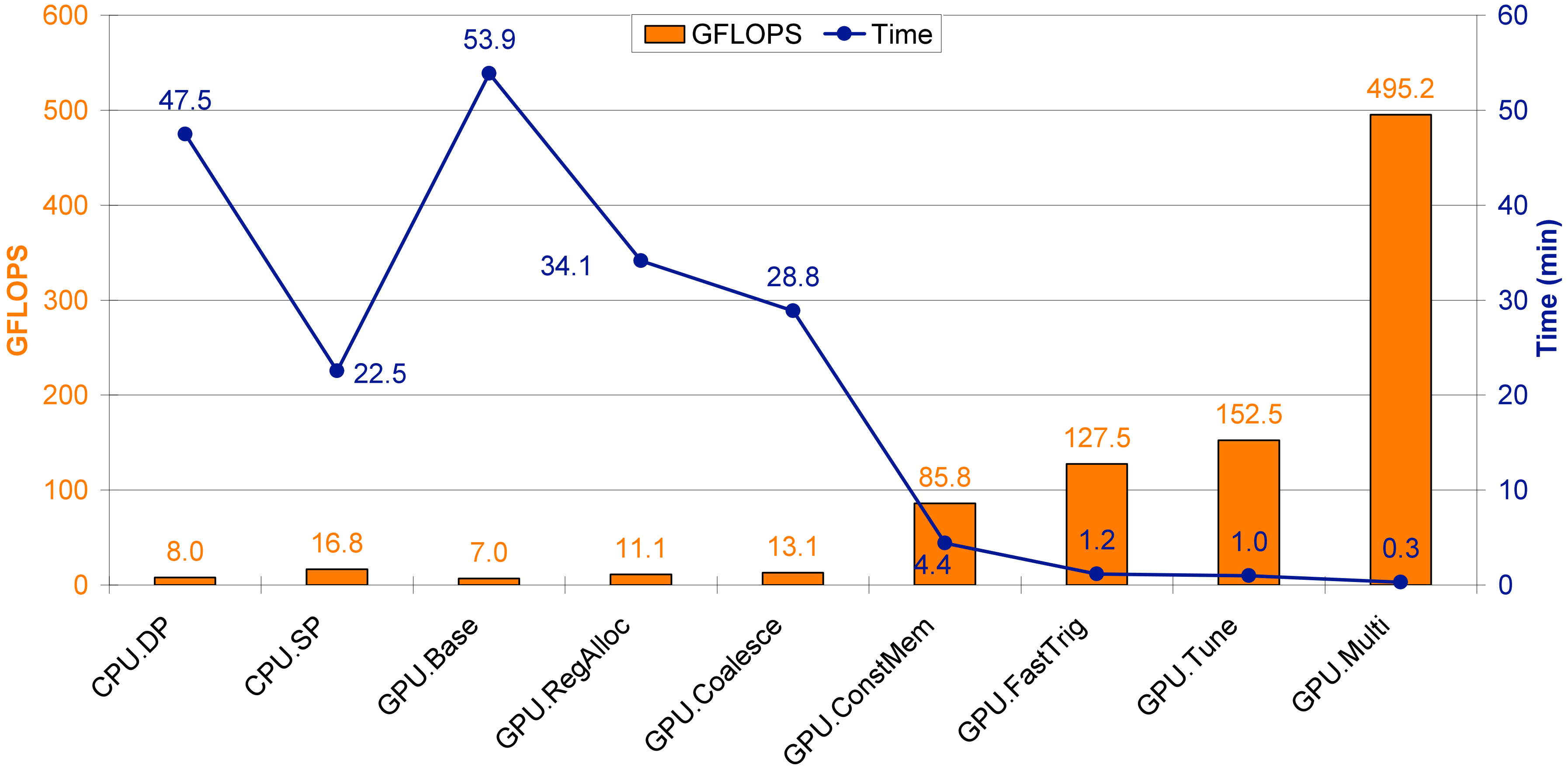
```
__global__ void
cmpFhD(float* gx, gy, gz, grFhD, giFhD) {
  int p = blockIdx.x * THREADS_PB + threadIdx.x;

  // register allocate image-space inputs & outputs
  x = gx[p];  y = gy[p];  z = gz[p];
  rFhD = grFhD[p];  iFhD = giFhD[p];

  for (int d = 0; d < SCAN_PTS_PER_TILE; d++) {
    // s (scan data) is held in constant memory
    float exp = 2 * PI * (s[d].kx * x +
                          s[d].ky * y +
                          s[d].kz * z);
    cArg = cos(exp);  sArg = sin(exp);
    rFhD += s[d].rRho*cArg – s[d].iRho*sArg;
    iFhD += s[d].iRho*cArg + s[d].rRho*sArg;
  }
  grFhD[p] = rFhD;  giFhD[p] = iFhD;
}
```

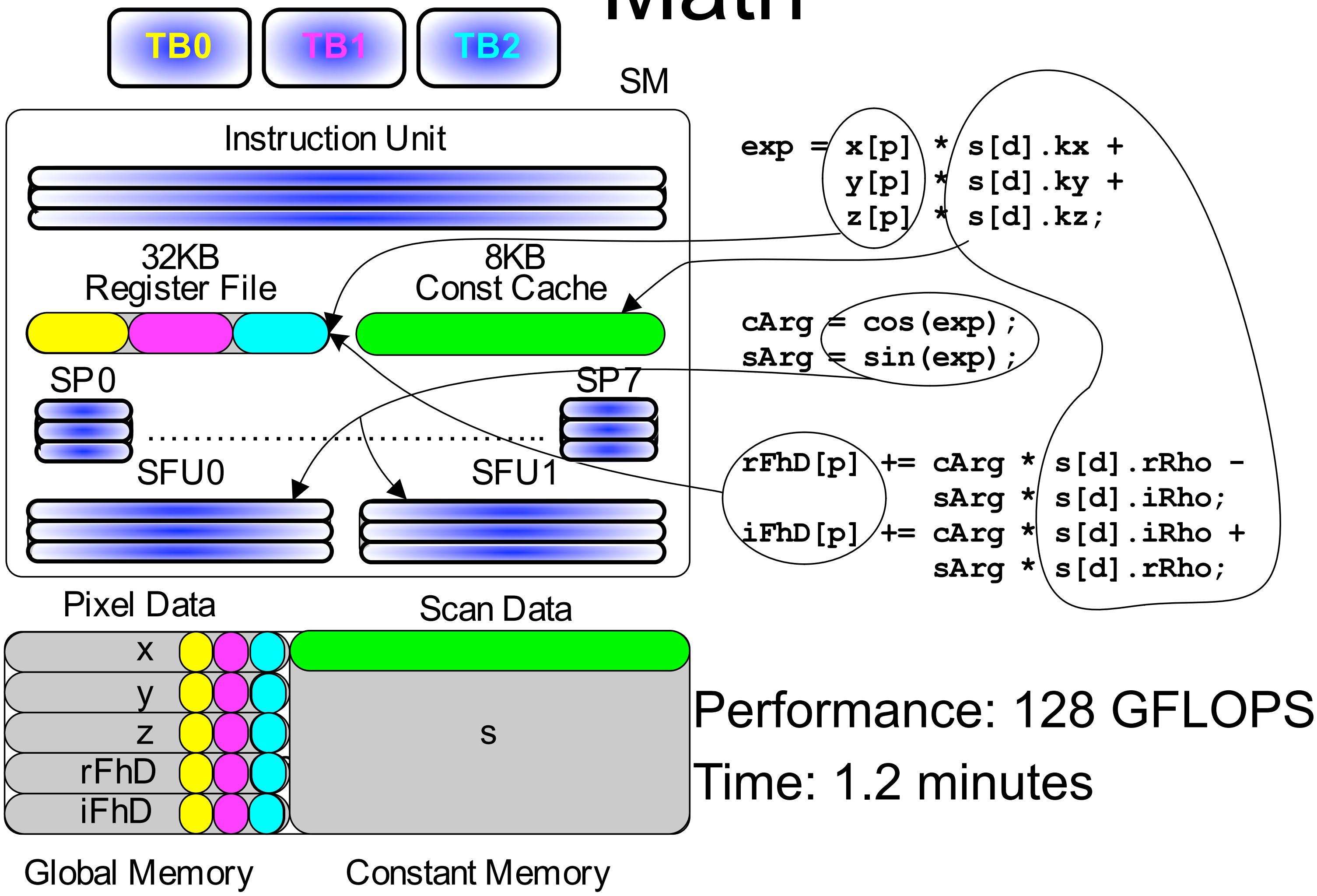
CUDA Kernel

Performance of FhD Computation



S.S. Stone, et al, “Accelerating Advanced MRI Reconstruction using GPUs,” ACM Computing Frontier Conference 2008, Italy, May 2008.

Final Data Arrangement and Fast Math



```
exp = x[p] * s[d].kx +
      y[p] * s[d].ky +
      z[p] * s[d].kz;

cArg = cos(exp);
sArg = sin(exp);

rFhD[p] += cArg * s[d].rRho -
           sArg * s[d].iRho;

iFhD[p] += cArg * s[d].iRho +
           sArg * s[d].rRho;
```

Performance: 128 GFLOPS

Time: 1.2 minutes

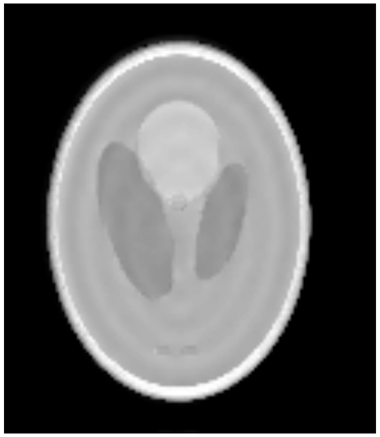
Results must be validated by domain experts.

True

Gridded

CPU.DP

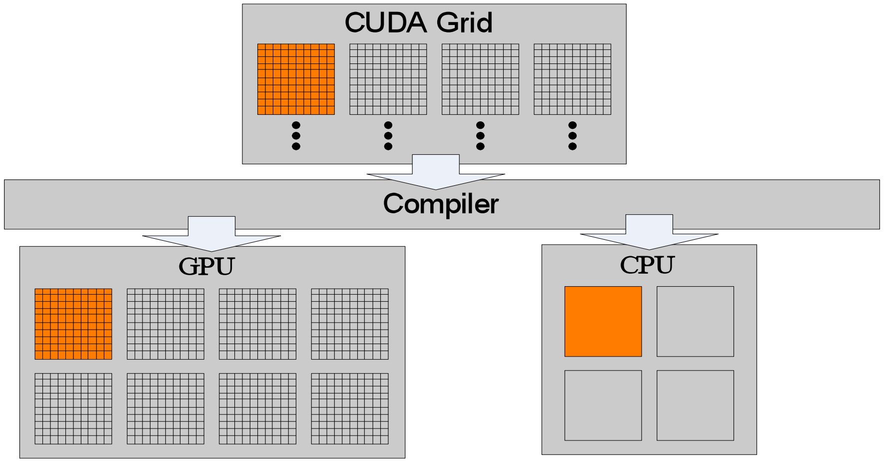
CPU.SP



GPU.Tune

CUDA for Multi-Core CPU

- A single GPU thread is too small for a CPU Thread
  - CUDA emulation does this and performs poorly
- CPU cores designed for ILP, SIMD
  - Optimizing compilers work well with iterative loops
- Turn GPU thread blocks from CUDA into iterative CPU loops



Bigger Picture Performance Results

- Consistent speed-up over hand-tuned single-thread code
- Best optimizations for GPU and CPU not always the same

Application	C on single core CPU Time	CUDA on 4-core CPU Time	Speedup*	CUDA on G80 Time
MRI-FHD	**~1000s**	**230s**	**~4x**	**8.5s**
CP	180s	45s	4x	.28s
SAD	42.5ms	25.6ms	1.66x	4.75ms
MM (4Kx4K)	7.84s**	15.5s	3.69x	1.12s

*Over hand-optimized CPU

**Intel MKL, multi-core execution

A Great Opportunity for Many

- GPU parallel computing allows
  - Drastic reduction in “time to discovery”
  - 1st principle-based simulation at meaningful scale
  - New, 3rd paradigm for research: computational experimentation
- The “democratization” of power to discover
  - $2,000/Teraflop SPFP in personal computers today
  - $5,000,000/Petaflops DPFP in clusters in two years
  - HW cost will no longer be the main barrier for big science
  - You will make the difference!

Course Objective

- To learn high-performance parallel programming
  - Computational thinking – formulating domain problems into computational models
  - Understanding hardware strength and limitation
  - Understand optimizations
- To maintain reliability and supportability
  - Using simple and disciplined parallel execution models
- To achieve scalability
  - Achieving high-performance on current and future hardware platforms with the same code

Agenda

Monday, August 18:

11:00 AM – 1:00 PM	Registration, Lunch
12:50-1:00 PM	Welcome
1:00 – 2:15 PM	Introduction
2:15 – 2:30 PM	Break
2:30 – 3:45 PM	CUDA Basics
4:00 – 5:00 PM	Multidisciplinary Panel (NCSA Auditorium)
5:30 – 6:30 PM	Reception (NCSA Lobby)

Agenda

Tuesday, August 19:

8:00 – 9:00 AM	Breakfast
9:00 – 10:15 AM	CUDA Threading Model
10:15 – 10:30 AM	Break
10:30 – 11:45 AM	CUDA Memory Model
12:00 – 1:00 PM	Lunch
1:00 – 3:45 PM	Hands-on Lab
4:00 – 5:00 PM	Keynote (NCSA Auditorium)
5:30 – 6:30 PM	Reception (NCSA Lobby)

Agenda

<u>Wednesday, August 20:</u>

- 8:00 – 9:00 AM Breakfast
- 9:00 – 10:15 AM Performance Considerations
- 10:15 – 10:30 AM Break
- 10:30 – 11:45 AM Floating-Point Considerations
- 12:00 – 1:00 PM Lunch
- 1:00 – 3:45 PM Hands-on Lab
- 4:00 – 5:00 PM Keynote (NCSA Auditorium)
- 5:30 – 6:30 PM Reception (NCSA Lobby)

Agenda

Thursday, August 21:

- 8:00 – 9:00 AM Breakfast
- 9:00 – 10:15 AM Case Study: Quantitative MRI
- 10:15 – 10:30 AM Break
- 10:30 – 11:45 AM Case Study: Molecular Dynamics
- 12:00 – 1:00 PM Lunch
- 1:00 – 3:45 PM Hands-on Lab
- 4:00 – 5:00 PM Keynote (NCSA Auditorium)
- 5:30 – 6:30 PM Reception (NCSA Lobby)

Agenda

Friday, August 22:

8:00 – 9:00 AM	Breakfast
8:30 – 9:45 AM	Wrap up and next steps
9:45-10:15	Student Feedback
10:15-10:30	Break
10:30 AM – 12:30 PM	Individual and/or group sharing of projects or ideas (quick and informal)
12:30	Box Lunch, Adjourn