How We Crammed a Black Hole, a Star Cluster, & a Turbulent Plasma into a GPU (and lived to talk about it)

William Dorland
Center for Multiscale Plasma Dynamics
Department of Physics
University of Maryland

NVision 08
Aug 26, 2008

In collaboration with

Ramani Duraiswami

Nail Gumerov

Kate Despain

Yuancheng Luo

Amitabh Varshney

George Stantchev

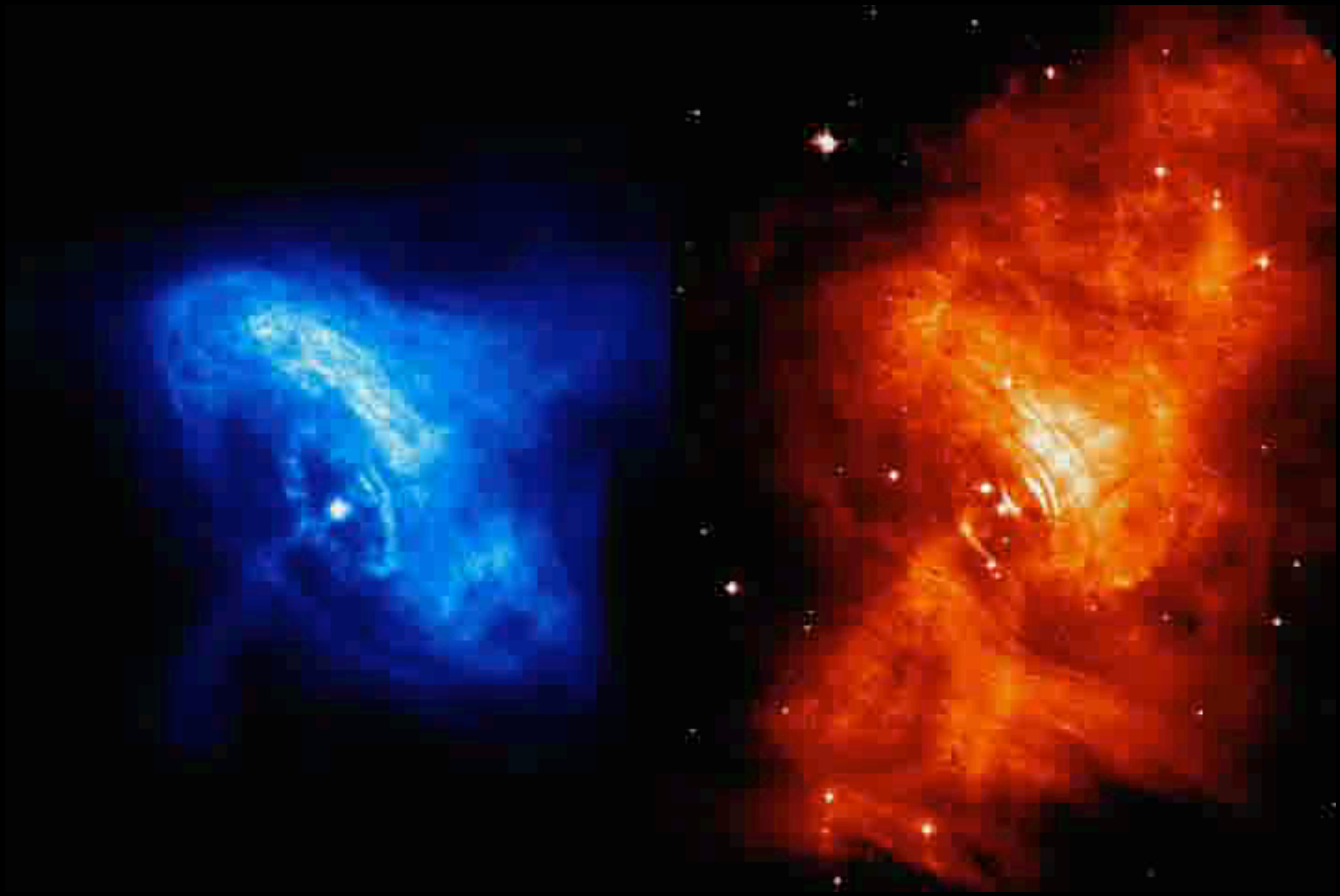
Bill Burns

Manuel Tiglio and Meem Mahmud

Advanced Viz: A Black Hole

Milky Way in X-Rays from Chandra

Pulsar in Crab Nebula: Chandra, Hubble



Everything you can see in these pictures is associated with super-hot, turbulent plasma (ionized gas).

To understand the details, one needs to understand how astrophysical objects turn gravitational energy into light.

This requires high-performance computing, now being accelerated with NVidia GPU's.

Moore's Law ++

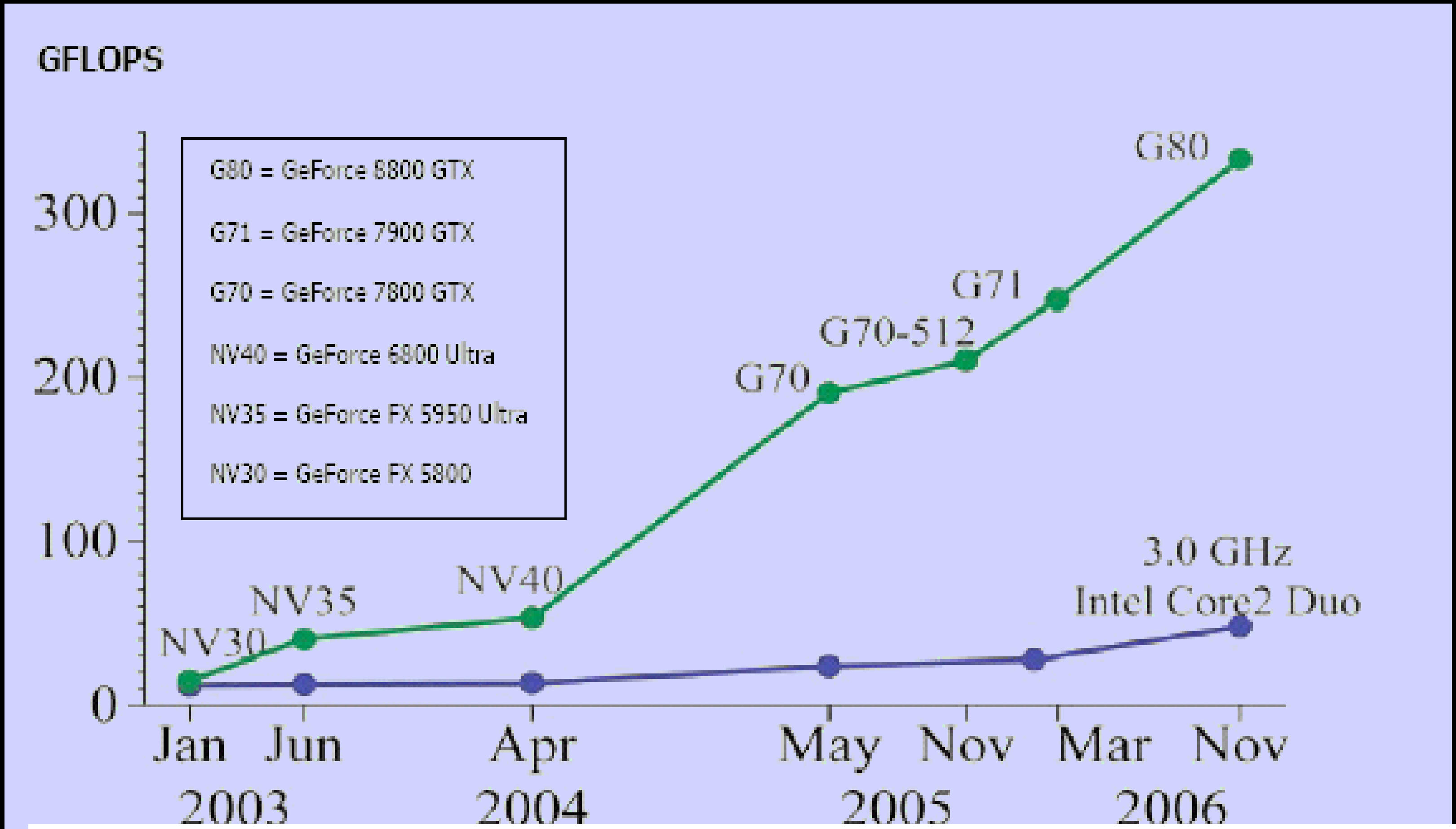
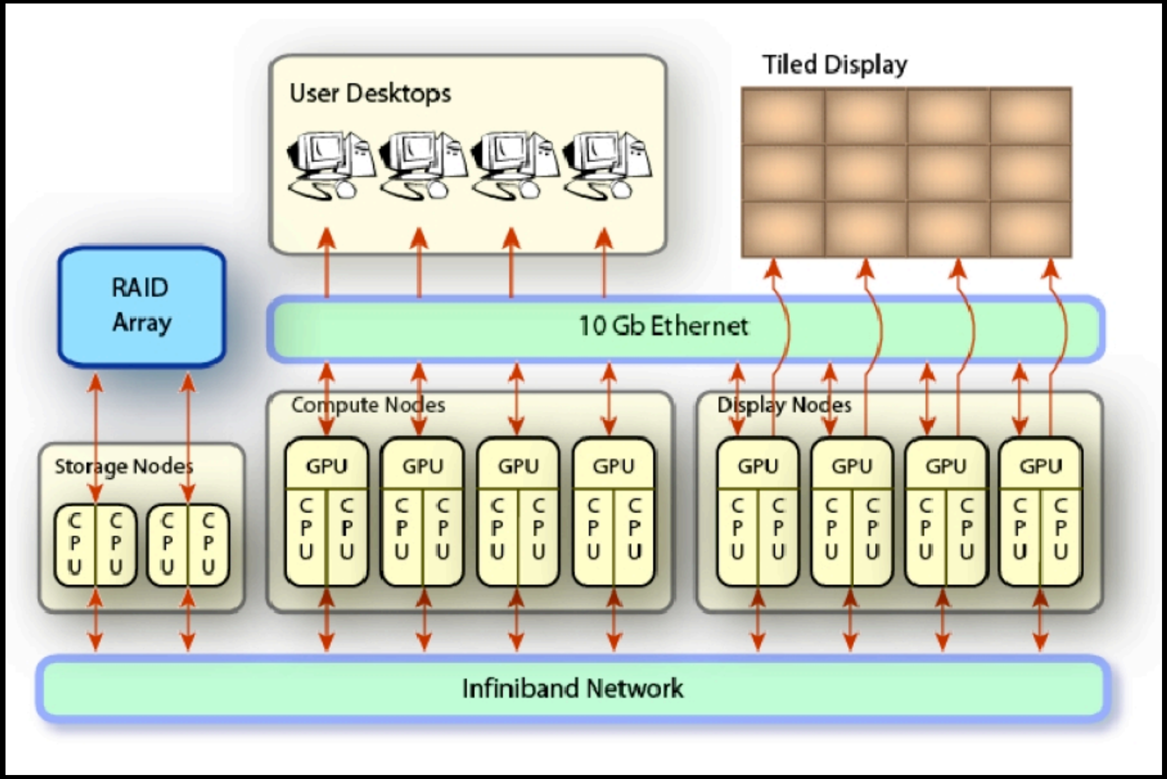


Fig. 1 Relative gigaflop ratings of state of the art Intel CPUs and NVIDIA GPUs

Overview

- Maryland effort: who, what, why
- Working physics algorithms
  1. FMM (N-body, other uses)
  2. Pseudo-spectral turbulence
  3. Particle-in-cell turbulence
  4. Also audio and visual computing on GPU's, optimization, large-scale visualization algorithms, etc
- Results of integration: Middleware library to accelerate development in Fortran 9x: ***Flagon***
- Maryland roadmap

Maryland GP-GPU hardware



NSF-funded CPU/GPU cluster with:

15 display nodes

12 computer nodes

3 storage nodes

1 console node

1 scheduler node

10 TB storage

CPUs: 3 GHz Xeon

GPUs: various nVidia models

Upgrade scheduled

Who is involved at UM?

- Effort began in UMIACS (UM Inst for Advanced Comp Studies)
- Focus broadened to include physics, astrophysics
- Weekly meetings of 5-20 with discussions, informal talks
- Undergraduates, graduate students and post-docs from outside UMIACS now more deeply involved.
- Funded research in GPU's from NSF, DOE, NASA, DARPA
- Current astrophysics-GPU projects: Turbulent heating, MHD turbulence, Numerical relativity

Key Questions Addressed

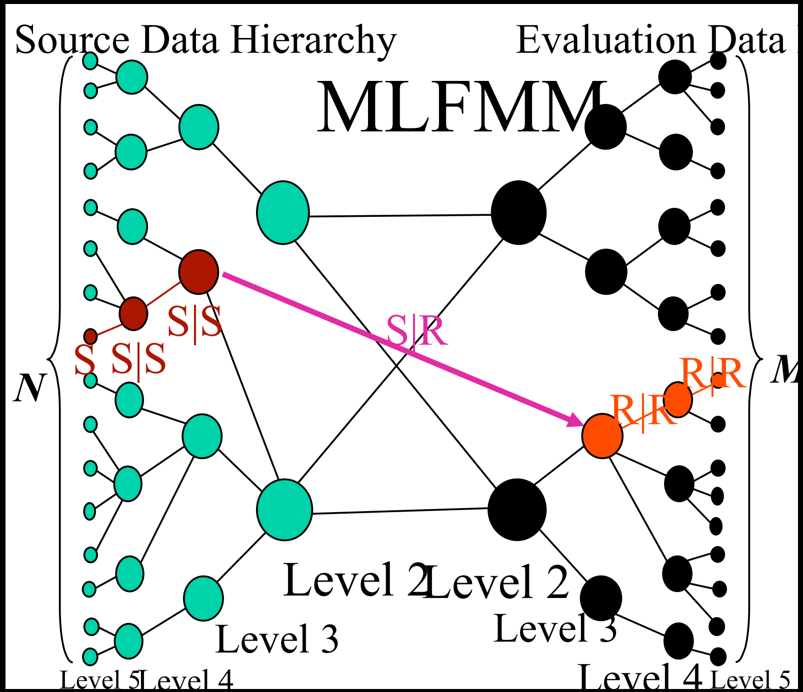
- How stable is the hardware + software platform?
- Which major algorithms can successfully exploit GPU hardware?
- What is the development and maintenance path for porting large-scale applications to this platform? Is there a net benefit? Or is the programming model too cumbersome?
- Which algorithms can exploit clusters of GPU's?
- Aiming for a $1-5M purchase in 2008-09 timeframe. CPU? GPU? Which vendor?

Refuting the Moore's law argument

- Argument ~ Wait for processor speed to solve complex problems, without algorithm improvement
- Is this true?
- Yes, for algorithms with linear asymptotic complexity
- No!! For algorithms with different aymptotic complexity
- Most scientific algorithms ~ N^2 or N^3
- For a million variales, we would need about 16 generations of Moore's law before an N^2 algorithm was comparable with an *O(N)* algorithm.
- Implies need for sophisticated algorithms, but are they programmable on a GPU?

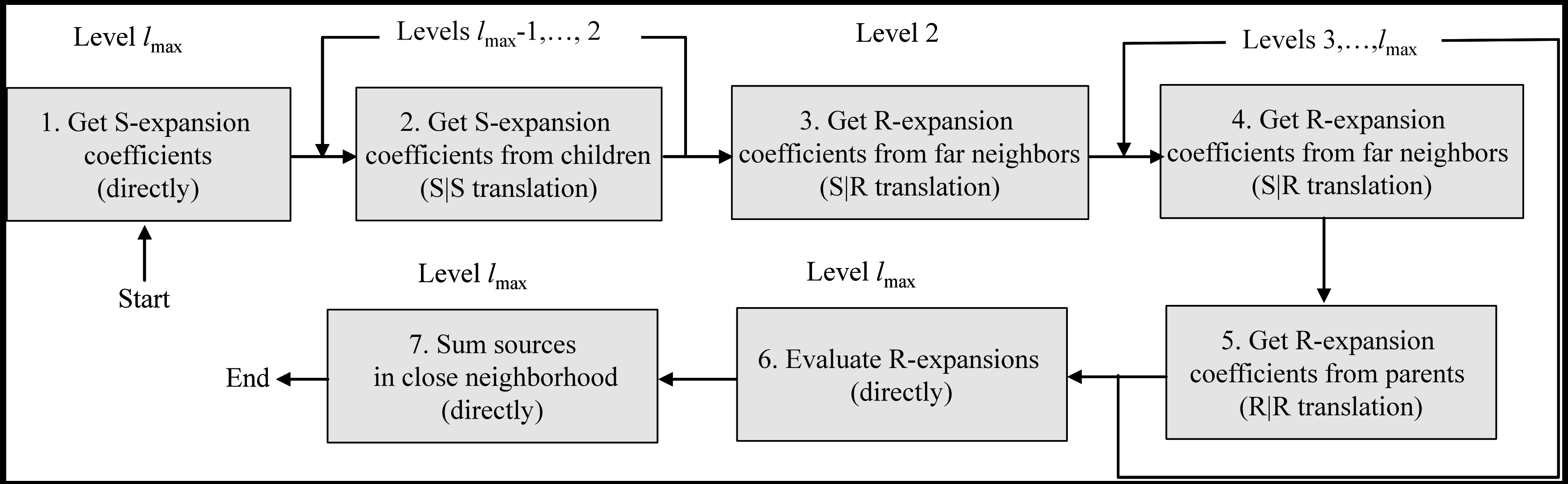
Fast Multipole Methods

- Follows from seminal work of Rokhlin and Greengard (1987)
- General method for accelerating large classes of dense matrix vector products
- Reduce computational/memory complexity from O(N*N) to O(N)
- Allow reduction of O(N*N) and O(N*N*N) operations to linear order
- Involves intricate programming with sophisticated data structures
- Recursive tree transverses



Fast Multipole Methods

- Follows from seminal work of Rokhlin and Greengard (1987)
- General method for accelerating large classes of dense matrix vector products
- Reduce computational/memory complexity from O(N*N) to O(N)
- Allow reduction of O(N*N) and O(N*N*N) operations to linear order
- Involves intricate programming with sophisticated data structures
- Recursive tree transverses



Challenges

- Complex FMM data structure
- No simply mapping to SPMD architecture
  - non-uniformity of data causes problems with efficient work load (taking into account large number of threads)
  - serial algorithms use recursive computations
  - existing libraries (CUBLAS) and middleware approach only get us part of the way
  - high performing FMM functions should be redesigned and written in CU
- Low amount of fast (shared/constant) local memory for efficient implementation of translation
- Absence of good debugging tools for GPU

Performance

N=1,048,576 (potential only)

	serial CPU	GPU	Ratio
p=4	22.25 s	0.683 s	33
p=8	51.17 s	0.908 s	56
p=12	66.56 s	1.395 s	48

N=1,048,576 (potential+forces (gradient))

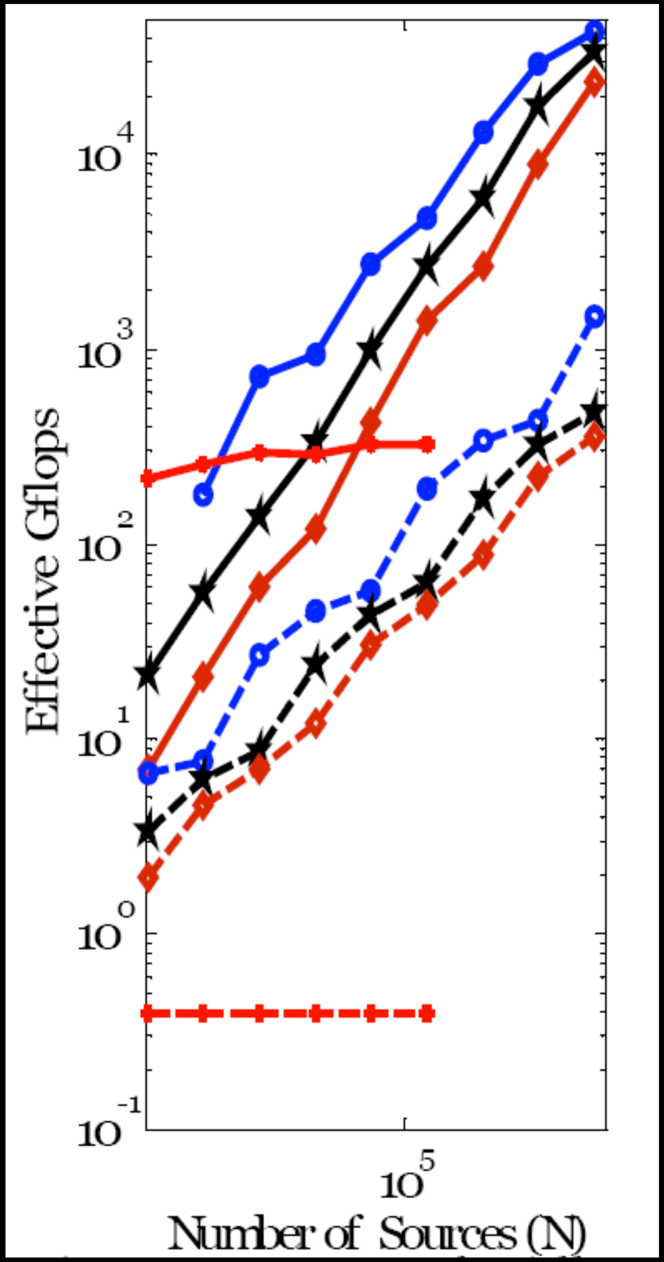
	serial CPU	GPU	Ratio
p=4	28.37 s	0.979 s	29
p=8	88.09 s	1.227 s	72
p=12	116.1 s	1.761 s	66

Performance

Computations of the potential and forces:

Peak performance of GPU for direct summation 290 Gigaflops, while for the FMM on GPU effective rates in range 25-50 Teraflops are observed (following the citation below).

M.S. Warren, J.K. Salmon, D.J. Becker, M.P. Goda, T. Sterling & G.S. Winckelmans. “Pentium Pro inside: I. a treecode at 430 Gigaflops on ASCI Red,” Bell price winning paper at SC’97, 1997.

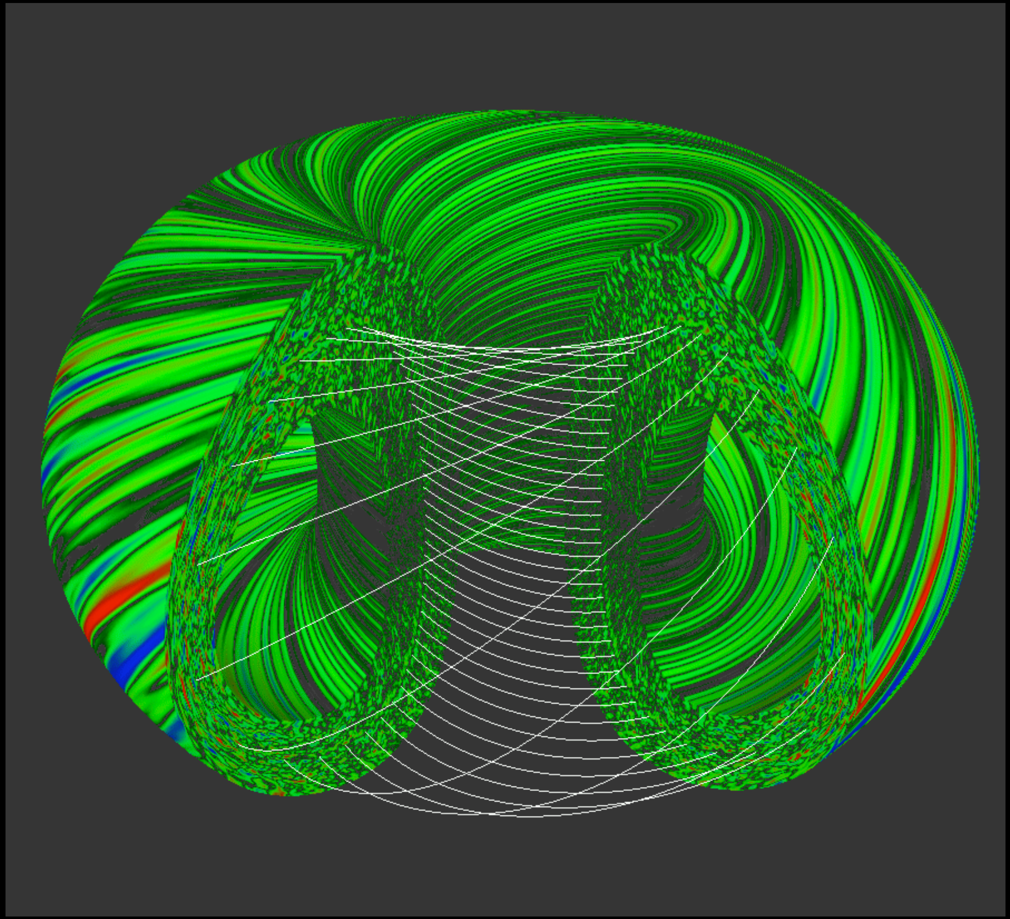


First conclusion: GPU can handle sophisticated algorithms

really well!

Turbulence theory is guided by computation/simulation

- Properties of plasma turbulence are important in laboratory experiments (such as fusion research), in space physics (solar wind), and in astrophysics (ISM, accretion flow luminosity)
- Most computations from Maryland carried out at national supercomputing facilities, on hundreds to thousands of processors.

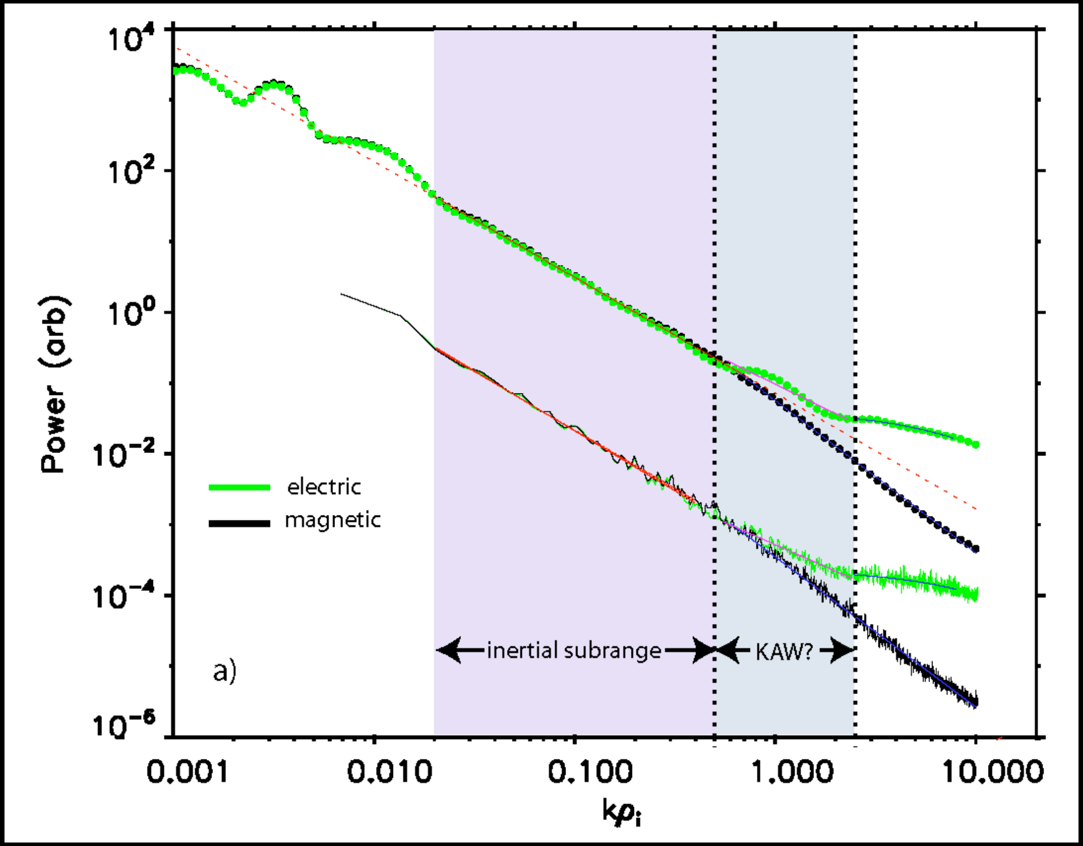


Fusion turbulence

Turbulence theory is guided by computation/simulation

- Properties of plasma turbulence are important in laboratory experiments (such as fusion research), in space physics (solar wind), and in astrophysics (ISM, accretion flow luminosity)

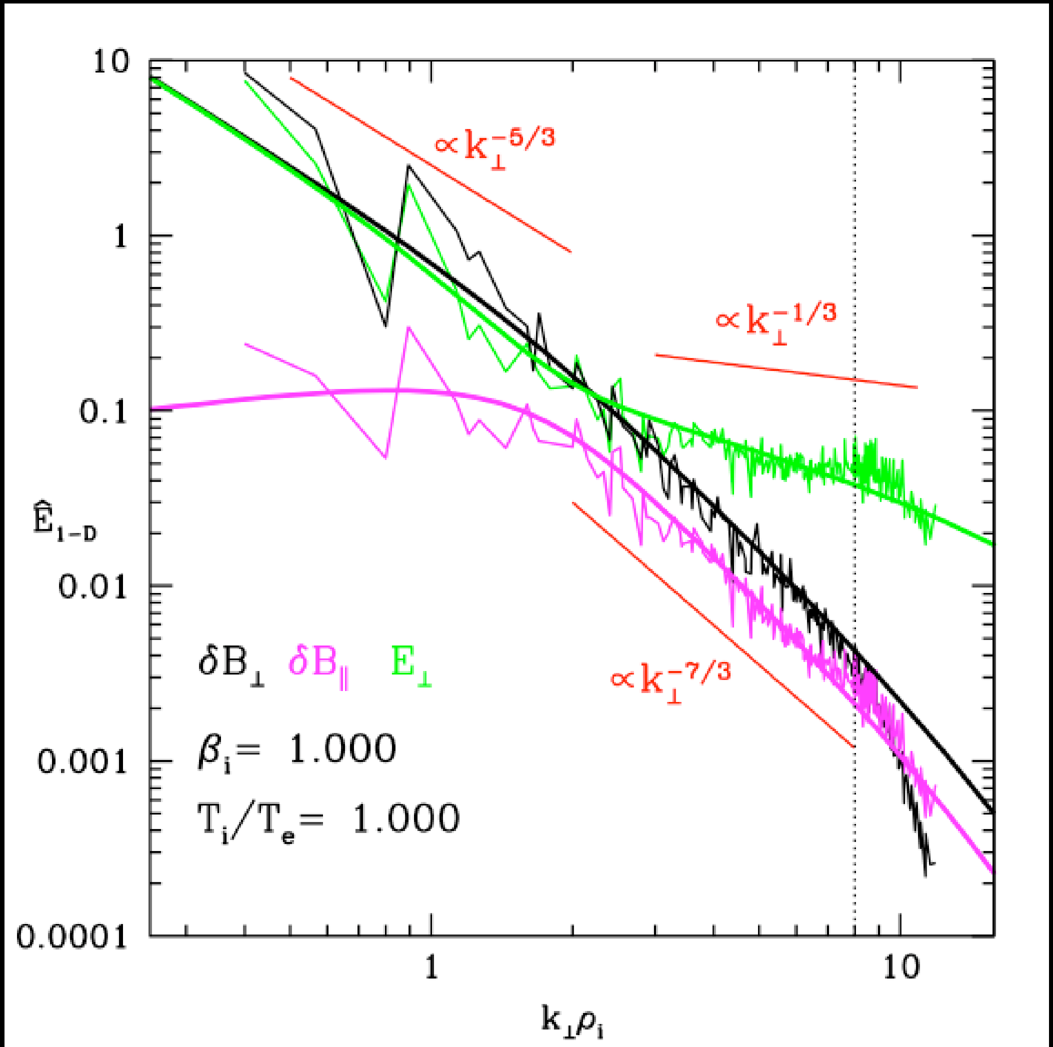
- Most computations from Maryland carried out at national supercomputing facilities, on hundreds to thousands of processors.



Solar wind turbulence *[Bale, et al, 2005]*

Turbulence theory is guided by computation/simulation

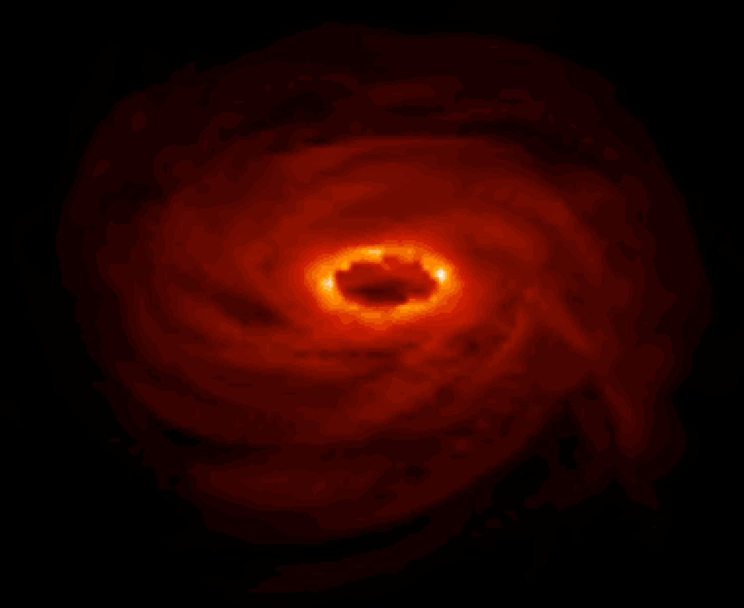
- Properties of plasma turbulence are important in laboratory experiments (such as fusion research), in space physics (solar wind), and in astrophysics (ISM, accretion flow luminosity)
- Most computations from Maryland carried out at national supercomputing facilities, on hundreds to thousands of processors.



First-principles simulations *[Howes, et al, 2007]*

Turbulence theory is guided by computation/simulation

- Properties of plasma turbulence are important in laboratory experiments (such as fusion research), in space physics (solar wind), and in astrophysics (ISM, accretion flow luminosity)
- Most computations from Maryland carried out at national supercomputing facilities, on hundreds to thousands of processors.

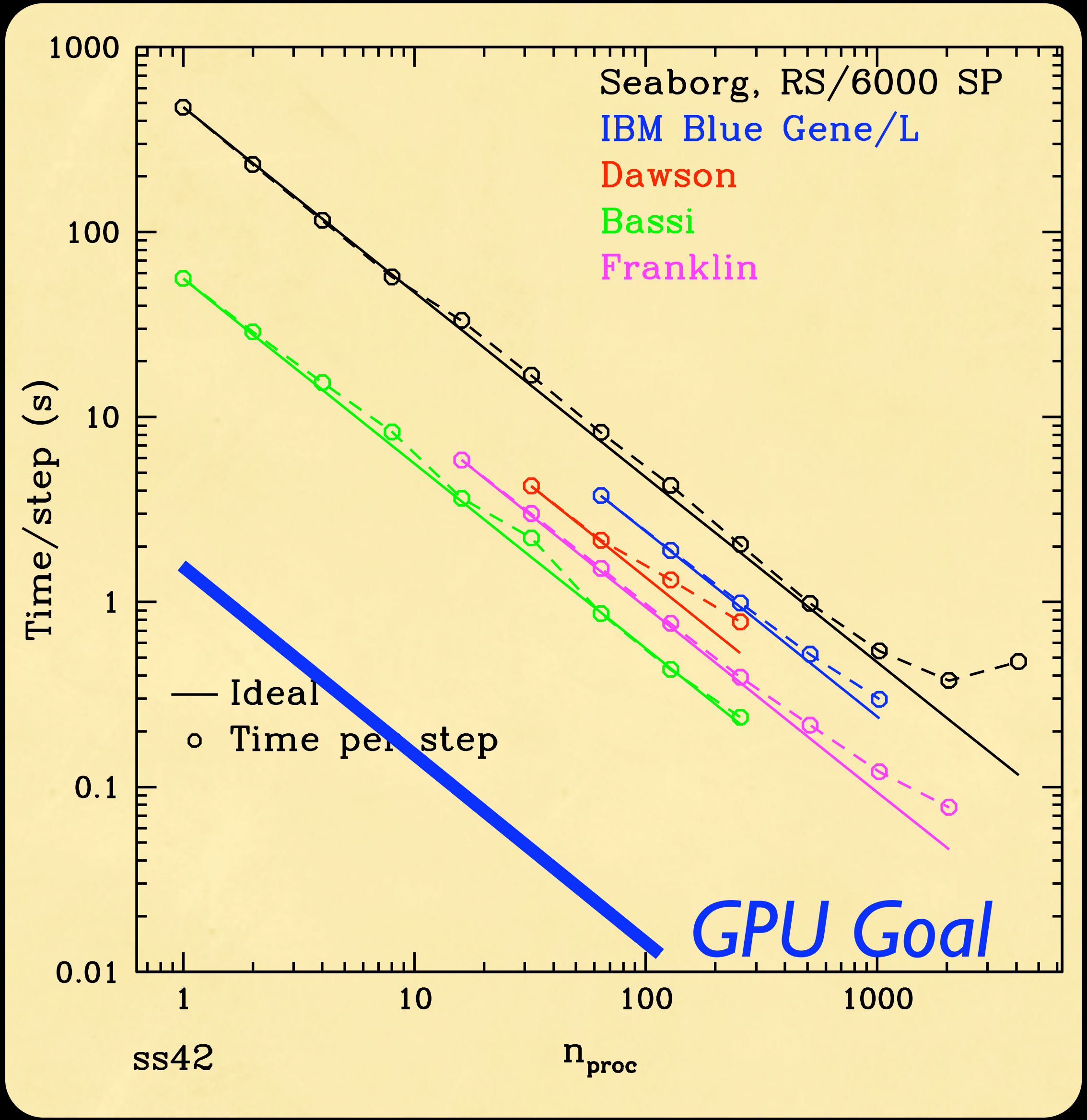


Accretion flow luminosity
[Goldston, Quataert, Igumenshchev, 2005]

Turbulence theory is guided by computation/simulation

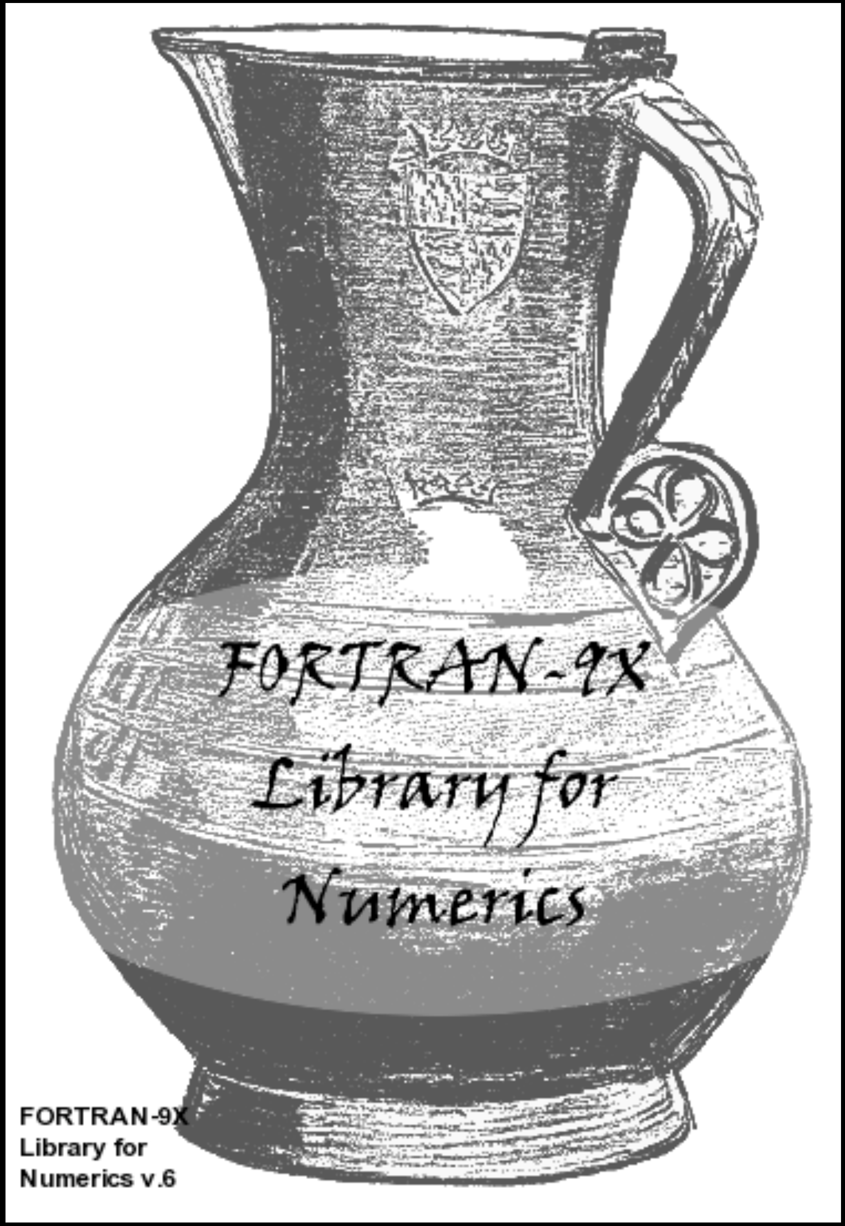
- Properties of plasma turbulence are important in laboratory experiments (such as fusion research), in space physics (solar wind), and in astrophysics (ISM, accretion flow luminosity)

- Most computations from Maryland carried out at national supercomputing facilities, on hundreds to thousands of processors.



Conversion of legacy Fortran 9x code important to scientific community

- Maintainability, portability
- Developed **Flagon**, an F9x wrapper to ease transition to CUDA
- Flagon available at Sourceforge as Open Source project



Conversion of legacy Fortran 9x code important to scientific community

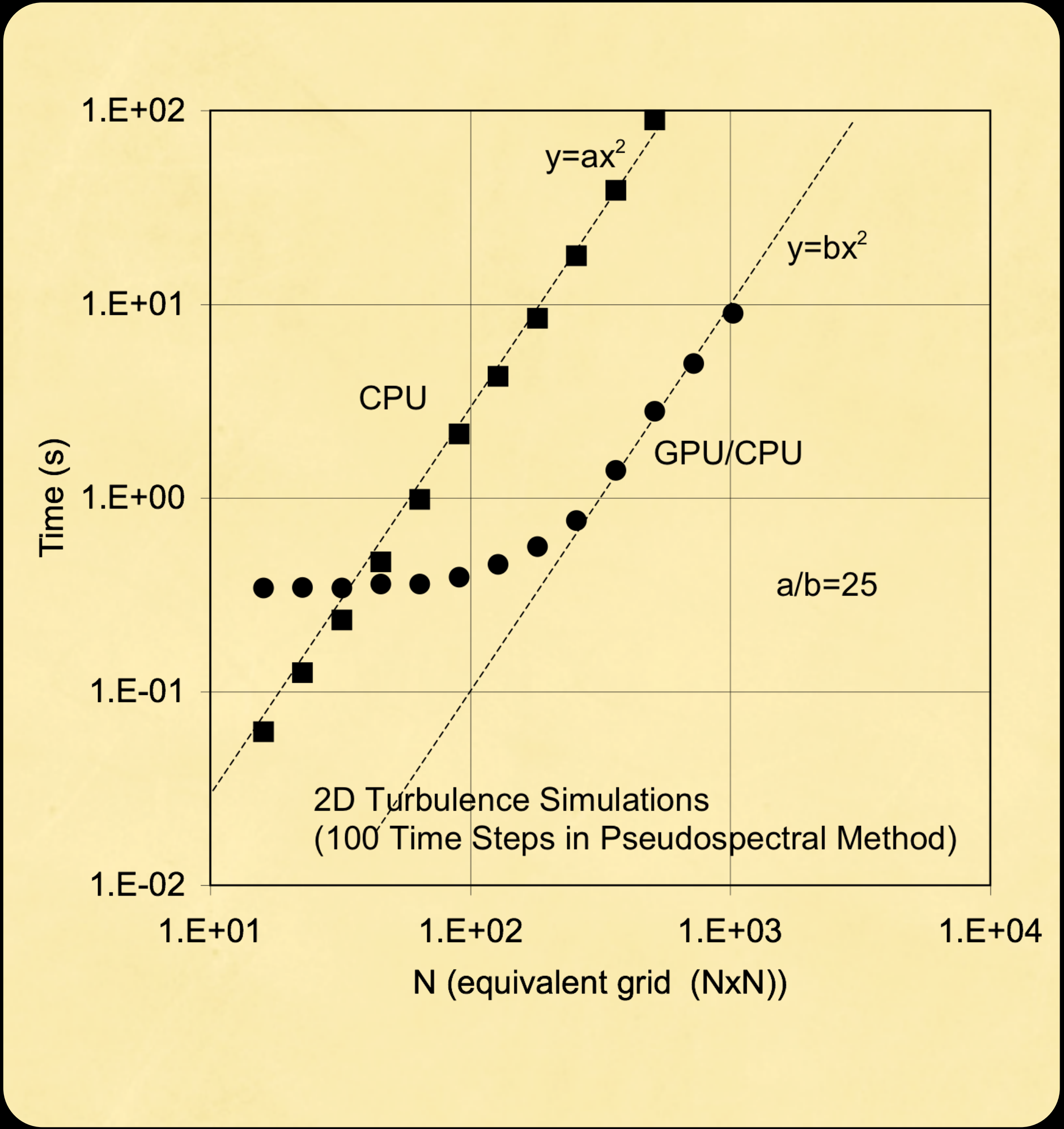
Original Fortran Code

```
subroutine nlterms(isave,a,b,nl)

use com_module


complex,dimension(:,:),intent(in)  ::a,b
complex,dimension(:,:),intent(out)::  nl
integer ::  isave

if(isave /= 1) then
     kxa2=ikx*a
     kya2=iky*a
     call fftwnd_f77_one(plan_f,kxa2,kxa2)
     call fftwnd_f77_one(plan_f,kya2,kya2)
endif

if(isave /= 2) then
     kxb2=ikx*b
     kyb2=iky*b
     call fftwnd_f77_one(plan_f,kxb2,kxb2)
     call fftwnd_f77_one(plan_f,kyb2,kyb2)
end if

nl=kxa2*kyb2-kxb2*kya2
call fftwnd_f77_one(plan_b,nl,nl)
nl=nl*scale

end subroutine nlterms
```

Modified Fortran Code

```
subroutine dev_nlterms(isave,dv_a,dv_b,dv_nl)

use dv_com_module
use mod_devObject

type(devVar),intent(in)  ::  dv_a,dv_b
type(devVar),intent(out)::  dv_nl
integer ::  isave

if(isave /= 1) then
     call devf_mul1(dv_kxa2,dv_kya2,dv_ikx,dv_iky,a)
     call devf_fft(dv_kxa2,fftplan)
     call devf_fft(dv_kya2,fftplan)
endif


if(isave /= 2) then
     call devf_mul1(dv_kxb2,dv_kyb2,dv_ikx,dv_iky,b)
     call devf_fft(dv_kxb2,fftplan)
     call devf_fft(dv_kyb2,fftplan)
endif


call devf_mul2(dv_nl,dv_kxa2,dv_kyb2,dv_kya2,dv_kxb2)
call devf_ifft(dv_nl,fftplan)
call devf_mul(dv_nl,dv_nl,dv_scale)

end subroutine dv_nlterms
```

GPU Performance Achievable by Non-Expert



Plasma turbulence code (Orzag-Tang problem) ported from existing Fortran 95 code in one day achieves 25x speedup

GPU non-expert (me)

Expert (Despain) then upped to 32x

Second conclusion: GPU can handle Fortran 9x and spectral/global algorithms

really well!

Summary of Performance Results

- MHD: 25x speed-up (nonlinear PDE's) -- recently upped to 32x (Despain)
  - FFT's, data-parallel. Very easy to exploit
- N-body force calculation with FMM on GPU: 1M masses in 1 second
  - Memory hierarchy, custom algorithm. Successfully exploited.
- Particle-in-cell: 7-20x speedup.
  - Need coalesced writes to get best performance; without sorting, can achieve ~ 100x speedup, but sorting is required to avoid memory bottleneck

Key early finding: Needed another layer to support easy porting, debugging and maintenance of Fortran 9x legacy codes. Developed Flagon. Happy to talk more about this at Astrophysics Roundtable, tomorrow AM.

Roadmap

- Now working on numerical relativity (with M Tiglio and M Mahmud)
- Upgrading current cluster
- Major questions remaining to be answered:

1. Multi-GPU computing necessary for astro apps we care about
    - Current and continuing focus
2. Other vendors may "win" -- don't want to be stuck with a Connection Machine...
    - Cell BE, AMD/ATI, Larrabee
3. Is development platform rich enough for GPU non-experts?