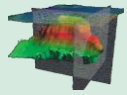
Table of Contents

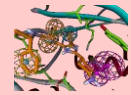
- Motivation & Trends in HPC
- R&D Projects @ PP
- **Mathematical Modeling**
- Numerical Methods used in HPSC
  - Systems of Differential Equations: ODEs & PDEs $m\frac{d^2x(t)}{dt^2} = F(x(t)),$
  - Solving Optimization Problems $\min_{x\in\mathbb{R}} f(x) = e^{-x} + x^2$
  - Solving Nonlinear Equations $f(x) = -e^{-x} + 2x = 0$
  - Basic Linear Algebra, Eigenvalues and Eigenvectors
  - Chaotic systems
- HPSC Program Development/Enhancement: from Prototype to Production
- Profiling, Performance Analysis & Optimization

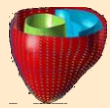
Objective

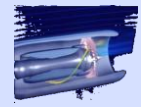
- Building up the ability to translate parallel computing power into science and engineering breakthroughs
  - Identify applications whose computing structures are suitable for
  - These applications can be revolutionized by 100X more computing power
  - You have access to expertise needed to tackle these applications
- Develop algorithm patterns that can result in both better efficiency as well as better HW utilization
  - To share with the community of application developers

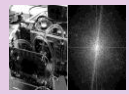
Future Science and Engineering Breakthroughs Hinge on Computing

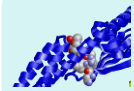

Computational Geoscience

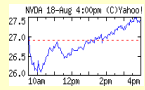

Computational Chemistry


Computational Medicine


Computational Modeling


Computational Physics


Computational Biology



Computational Finance


Image Processing

A major shift of paradigm

- In the 20th Century, we were able to understand, design, and manufacture what we can ***measure***
  - Physical instruments and computing systems allowed us to see farther, capture more, communicate better, understand natural processes, control artificial processes…
- In the 21st Century, we are able to understand, design, create what we can ***compute***
  - Computational models are allowing us to see even farther, going back and forth in time, relate better, test hypothesis that cannot be verified any other way, create safe artificial processes

Examples of Paradigm Shift

20th Century	21st Century
• Small mask patterns and short light waves • Electronic microscope and Crystallography with computational image processing • Anatomic imaging with computational image processing • Teleconference	• Computational optical proximity correction • Computational microscope with initial conditions from Crystallography • Metabolic imaging sees disease before visible anatomic change • Tele-emersion

Faster is not "just Faster"

- 2-3X faster is "just faster"
  - Do a little more, wait a little less
  - Doesn't change how you work
- 5-10x faster is "significant"
  - Worth upgrading
  - Worth re-writing (parts of) the application
- 100x+ faster is "fundamentally different"
  - Worth considering a new platform
  - Worth re-architecting the application
  - Makes new applications possible
  - Drives "time to discovery" and creates fundamental changes in Science

How much computing power is enough?

- Each jump in computing power motivates new ways of computing
  - Many apps have approximations or omissions that arose from limitations in computing power
  - Every 100x jump in performance allows app developers to innovate
  - Example: graphics, medical imaging, physics simulation, etc.

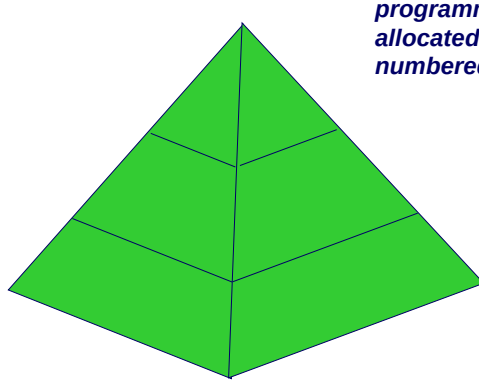
Application developers do not take us seriously until they see real results.

Why didn't this happen earlier?

- Computational experimentation is just reaching critical mass
  - Simulate large enough systems
  - Simulate long enough system time
  - Simulate enough details
- Computational instrumentation is also just reaching critical mass
  - Reaching high enough accuracy
  - Cover enough observations

A Great Opportunity for Many

- New massively parallel computing is enabling
  - 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
- This is once-in-a-career opportunity for many!



The Pyramid of Parallel Programming

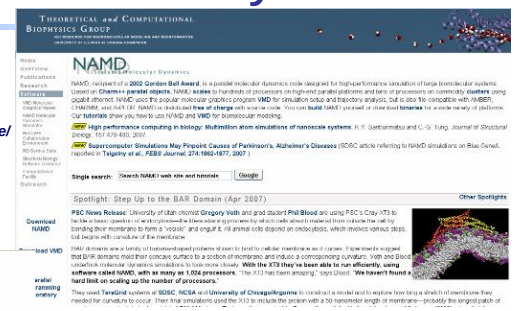
Thousand-node systems with MPI-style programming, >100 TFLOPS, $M, allocated machine time (programmers numbered in hundreds)

Hundred-core systems with CUDA-style programming, 1-5 TFLOPS, $K, machines widely availability (programmers numbered in 10s of thousands)

Hundred-core systems with MatLab-style programming, 10-50 GFLOPS, $K, machines widely available (programmers numbered in millions)

VMD/NAMD Molecular Dynamics

http://www.ks.uiuc.edu/Research/vmd/projects/ece498/lecture/



Parallel GPUs with Multithreading:
705 GFLOPS /w 3 GPUs

- One host thread is created for each CUDA GPU
- Threads are spawned and attach to their GPU based on their host thread ID
  - First CUDA call binds that thread's CUDA context to that GPU for life
  - Handling error conditions within child threads is dependent on the thread library and, makes dealing with any CUDA errors somewhat tricky, left as an exercise to the reader…. ☺
- Map slices are computed cyclically by the GPUs
- Want to avoid false sharing on the host memory system
  - map slices are usually much bigger than the host memory page size, so this is usually not a problem for this application
- Performance of 3 GPUs is stunning!
- Power: 3 GPU test box consumes 700 watts running flat out

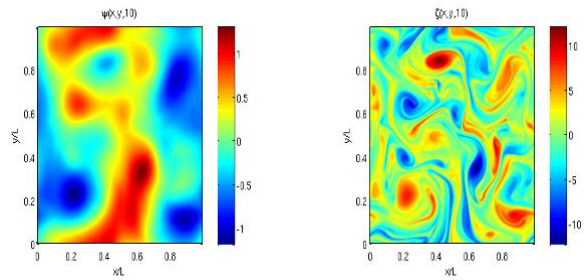
© David Kirk/NVIDIA and Wen-mei W. Hwu, 2007
ECE 498AL, University of Illinois, Urbana-Champaign
21

- 240X speedup
- Computational biology

Matlab: Language of Science

15X with MATLAB CPU+GPU

http://developer.nvidia.com/object/matlab_cuda.html

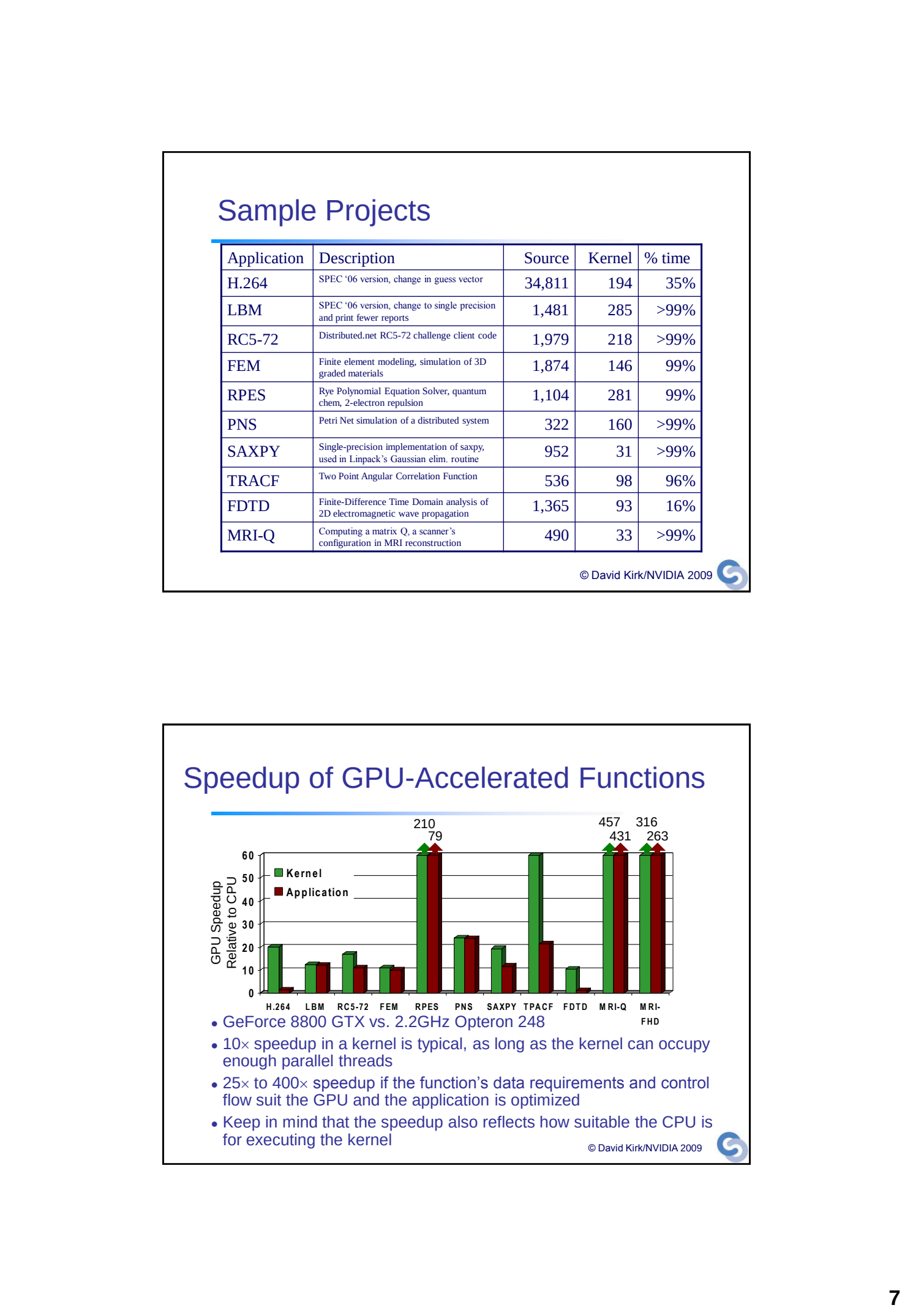


Pseudo-spectral simulation of 2D Isotropic turbulence

http://www.amath.washington.edu/courses/571-winter-2006/matlab/FS_2Dturb.m

Sample Projects

Application	Description	Source	Kernel	% time
H.264	SPEC '06 version, change in guess vector	34,811	194	35%
LBM	SPEC '06 version, change to single precision and print fewer reports	1,481	285	>99%
RC5-72	Distributed.net RC5-72 challenge client code	1,979	218	>99%
FEM	Finite element modeling, simulation of 3D graded materials	1,874	146	99%
RPES	Rye Polynomial Equation Solver, quantum chem, 2-electron repulsion	1,104	281	99%
PNS	Petri Net simulation of a distributed system	322	160	>99%
SAXPY	Single-precision implementation of saxpy, used in Linpack's Gaussian elim. routine	952	31	>99%
TRACF	Two Point Angular Correlation Function	536	98	96%
FDTD	Finite-Difference Time Domain analysis of 2D electromagnetic wave propagation	1,365	93	16%
MRI-Q	Computing a matrix Q, a scanner's configuration in MRI reconstruction	490	33	>99%



Speedup of GPU-Accelerated Functions

- GeForce 8800 GTX vs. 2.2GHz Opteron 248
- 10× speedup in a kernel is typical, as long as the kernel can occupy enough parallel threads
- 25× to 400× speedup if the function's data requirements and control flow suit the GPU and the application is optimized
- Keep in mind that the speedup also reflects how suitable the CPU is for executing the kernel

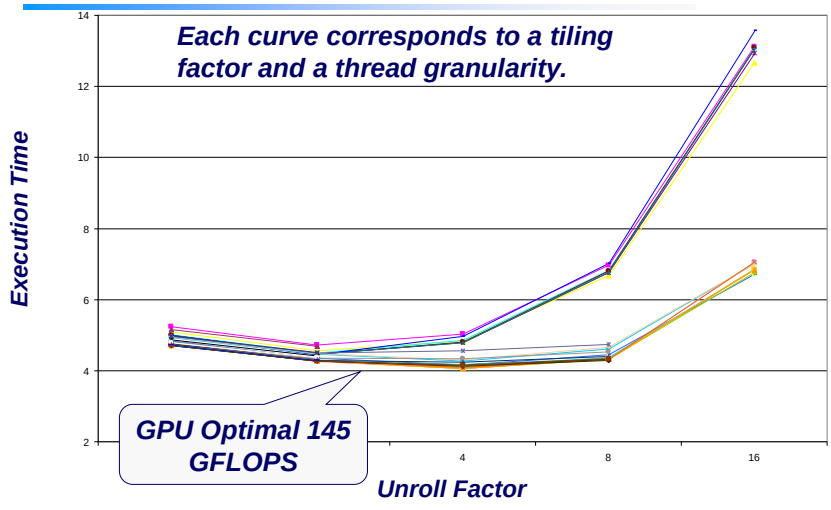
Sample Programs Scaling

App.	Archit. Bottleneck	Simult. T	Kernel X	App X
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-Q	Instruction issue rate	8,192	457.0	431.0

Magnetic Resonance Imaging

- 3D MRI image reconstruction from non-Cartesian scan data is very accurate, but compute-intensive
- >200× speedup in MRI-Q
  - CPU – Athlon 64 2800+ with fast math library
- MRI code runs efficiently on the GeForce 8800
  - High-floating point operation throughput, including trigonometric functions
  - Fast memory subsystems
    - Larger register file
    - Threads simultaneously load same value from constant memory
    - Access coalescing to produce < 1 memory access per thread, per loop iteration

MRI Optimization Space Search



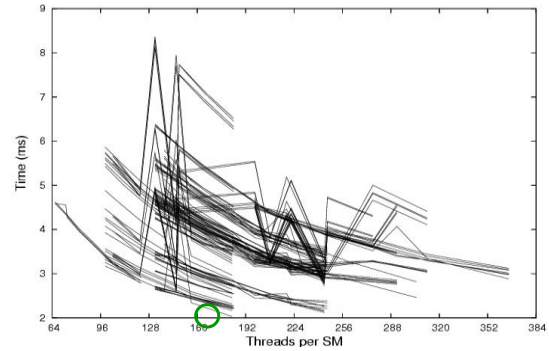
H.264 Video Encoding (from SPEC)

- GPU kernel implements sum-of-absolute difference computation
  - Compute-intensive part of motion estimation
  - Compares many pairs of small images to estimate how closely they match
  - An optimized CPU version is 35% of execution time
  - GPU version limited by data movement to/from GPU, not compute
- Loop optimizations remove instruction overhead and redundant loads
- …and increase register pressure, reducing the number of threads that can run concurrently, exposing texture cache latency

Large and Discontinuous Spaces

Sum of Absolute Differences

- ***More complex than matrix multiplication, but still relatively small (hundreds of lines of code)***
- ***Many performance discontinuities***
- ***To be presented at GPGPU '07***

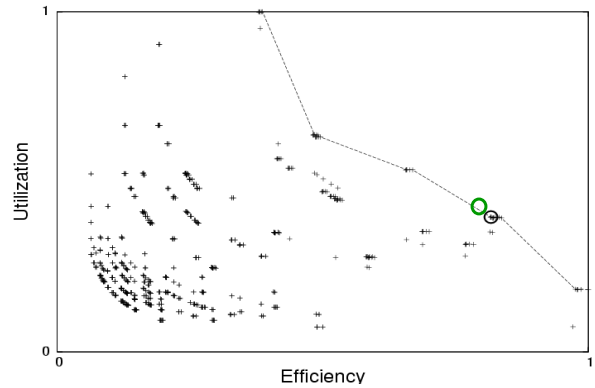


- Search spaces can be huge! Exhaustive search with smaller-than usual data sets still takes significant time.

Analytical Models Help Reduce Search Space.

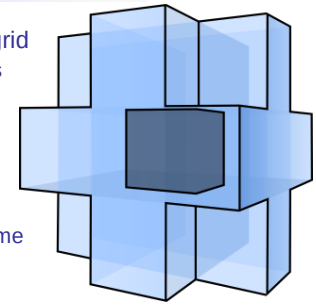
Sum of Absolute Differences

By selecting only Pareto-optimal points, we pruned the search space by 98% and still found the optimal configuration



LBM Fluid Simulation (from SPEC)

- Simulation of fluid flow in a volume divided into a grid
  - It's a stencil computation: A cell's state at time *t+1* is computed from the cell and its neighbors at time *t*
- Synchronization is required after each timestep – achieved by running the kernel once per timestep
- Local memories on SMs are emptied after each kernel invocation
  - Entire data set moves in and out of SMs for every time step
  - High demand on bandwidth
- Reduce bandwidth usage with software-managed caching
  - Memory limits 200 grid cells/threads per SM
  - Not enough threads to completely cover global memory latency



Flow through a cell (dark blue) is updated based on its flow and the flow in 18 neighboring cells (light blue).

Prevalent Performance Limits

Some microarchitectural limits appear repeatedly across the benchmark suite:

- Global memory bandwidth saturation
  - Tasks with intrinsically low data reuse, e.g. vector-scalar addition or vector dot product
  - Computation with frequent global synchronization
    - Converted to short-lived kernels with low data reuse
    - Common in simulation programs
- Thread-level optimization vs. latency tolerance
  - Since hardware resources are divided among threads, low per-thread resource use is necessary to furnish enough simultaneously-active threads to tolerate long-latency operations
  - Making individual threads faster generally increases register and/or shared memory requirements
  - Optimizations trade off single-thread speed for exposed latency

What you will likely need to hit hard

- Parallelism extraction requires global understanding
  - Most programmers only understand parts of an application
- Algorithms need to be re-designed
  - Programmers benefit from clear view of the algorithmic effect on parallelism
- Real but rare dependencies often need to be ignored
  - Error checking code, etc., parallel code is often not equivalent to sequential code
- Getting more than a small speedup over sequential code is very tricky
  - ~20 versions typically experimented for each application to move away from architecture bottlenecks

Call to action on Project Work

- One page project description (pdf + trac) due 7.11.2010
  - **Introduction:** A one paragraph description of the significance of the application.
  - **Description:** A one to two paragraph brief description of what the application really does
  - **Objective:** A sentence on what I would like to accomplish with the team on the application – we have to agree on this at the lab.
  - **Background :** Outline the technical skills (type of Math, Physics, Chemistry, etc) that one needs to understand and work on the application.
  - **Resources:** A list of web and traditional resources that students can draw for technical background, general information and building blocks. Give URL or trac/svn links. Only use our trac/svn.
  - **Contact Information:** Name, e-mail, group and master program the team members are part of.
- The labs on 26.10, 2.11 are dedicated to presentation of project ideas by you and me and used to recruit teammates