Some Myths in High Performance Computing

William D. Gropp
www.mcs.anl.gov/~gropp
Mathematics and Computer Science
Argonne National Laboratory

Some Popular Myths

- Parallel Programming is Hard
  - Harder than what?
  - Have you tried to keep your laptop up?
- Shared-Memory will save the day
  - Correctness of programs?
  - Why have SMP OSes been so troublesome?
- New Programming Languages are Needed
  - Where will the applications come from?
  - Why is this true? (Is Java a new language or a dialect of C/C++?)
- The Grid
  - What is it?
  - Does it work?

Why are These Myths Popular?

- Myths are fun to repeat
  - That’s how they become myths
- Myths fill a need
  - To explain the unknown
  - Particularly capricious and painful events
- Myths reflect a view of reality

- Understanding Myths gives us an advantage

Myth: Parallel Programming is Hard

- Reality:
  - Programming for performance is hard
  - Programming for correctness is hard
- Many parallel computers achieve a low fraction of peak performance
  - Inference: Parallel programming is hard
- Why is programming for performance hard, and how does it relate to parallel computing?

Choosing the Correct Metric

- Classically, numerical analysts have counted floating point operations
  - Flops used to be expensive
  - Goal for algorithms is O(n) work (defined as floating point operations) on O(n) data
    - But this does not reflect actual computational effort
- True costs are now more often related to memory loads/stores
  - BLAS3 advantage over BLAS1,2 is n^3 operations with n^2 load/stores

Myth #1

- Parallel computers achieve a low fraction of peak performance

- Reality: True but not because of parallelism

Sparse Matrix-Vector Product

- Common operation for optimal (in floating-point operations) solution of linear systems
- Sample code:

```
for row=0,n-1
    m   = i[row+1] - i[row];
    sum = 0;
    for k=0,m-1
        sum += *a++ * x[*j++];
    y[i] = sum;
```

- Data structures are a[nnz], j[nnz], i[n], x[n], y[n]

Simple Performance Analysis

- Memory motion:
  - nnz (sizeof(double) + sizeof(int)) +
    n (2*sizeof(double) + sizeof(int))
  - Perfect cache (never load same data twice)
- Computation
  - nnz multiply-add (MA)
- Roughly 12 bytes per MA
- Typical WS node can move 1-4 bytes/MA
  - *Maximum* performance is 8-33% of peak

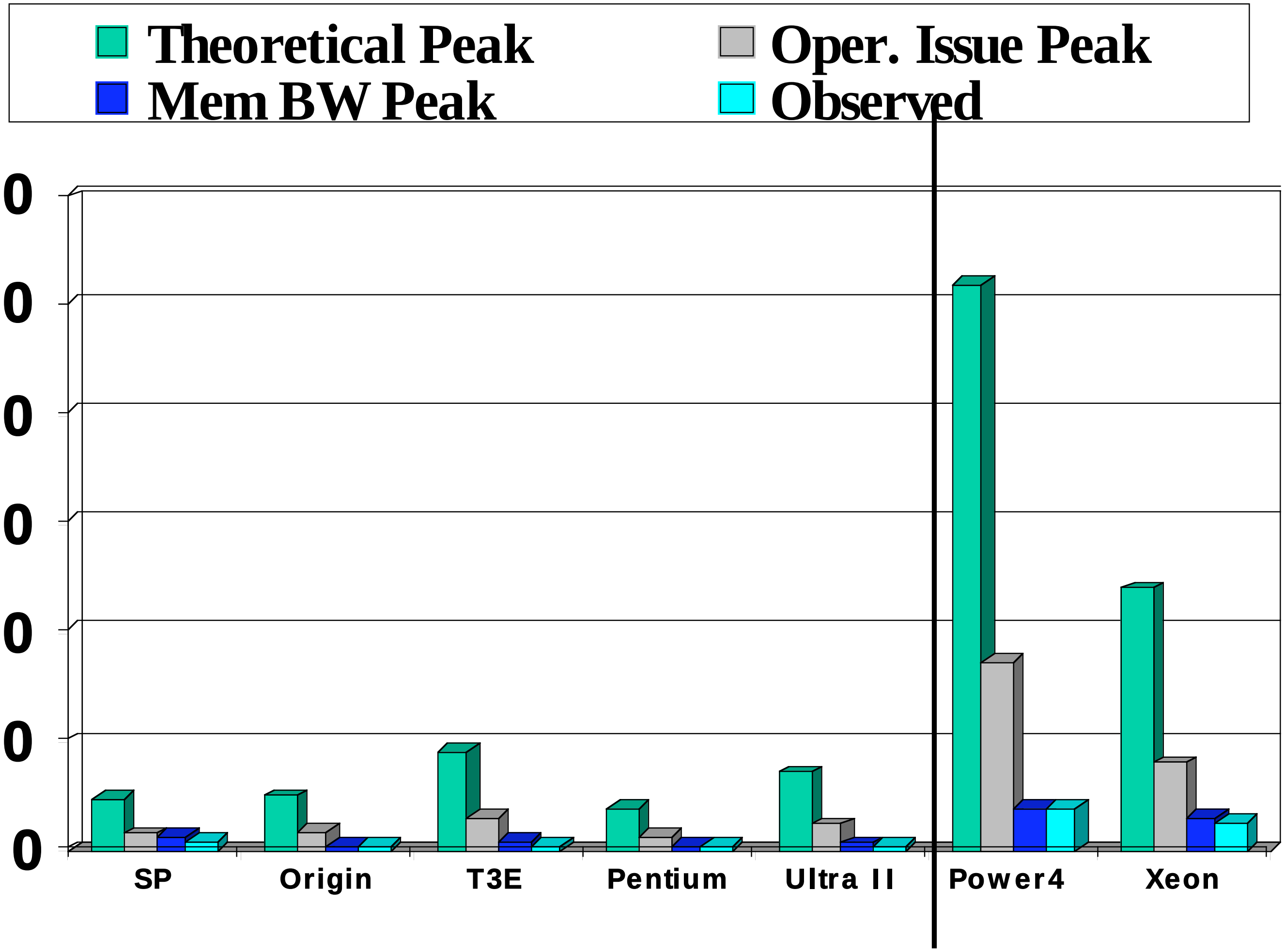
More Performance Analysis

- Instruction Counts:
  - nnz (2*load-double + load-int + mult-add) +
    n (load-int + store-double)
- Roughly 4 instructions per MA
- Maximum performance is 25% of peak (33% if MA overlaps one load/store)
- Changing matrix data structure (e.g., exploit small block structure) allows reuse of data in register, eliminating some loads (x and j)
- Implementation improvements (tricks) cannot improve on these limits

Realistic Measures of Peak Performance

Sparse Matrix Vector Product

one vector, matrix size, m = 90,708, nonzero entries nz = 5,047,120

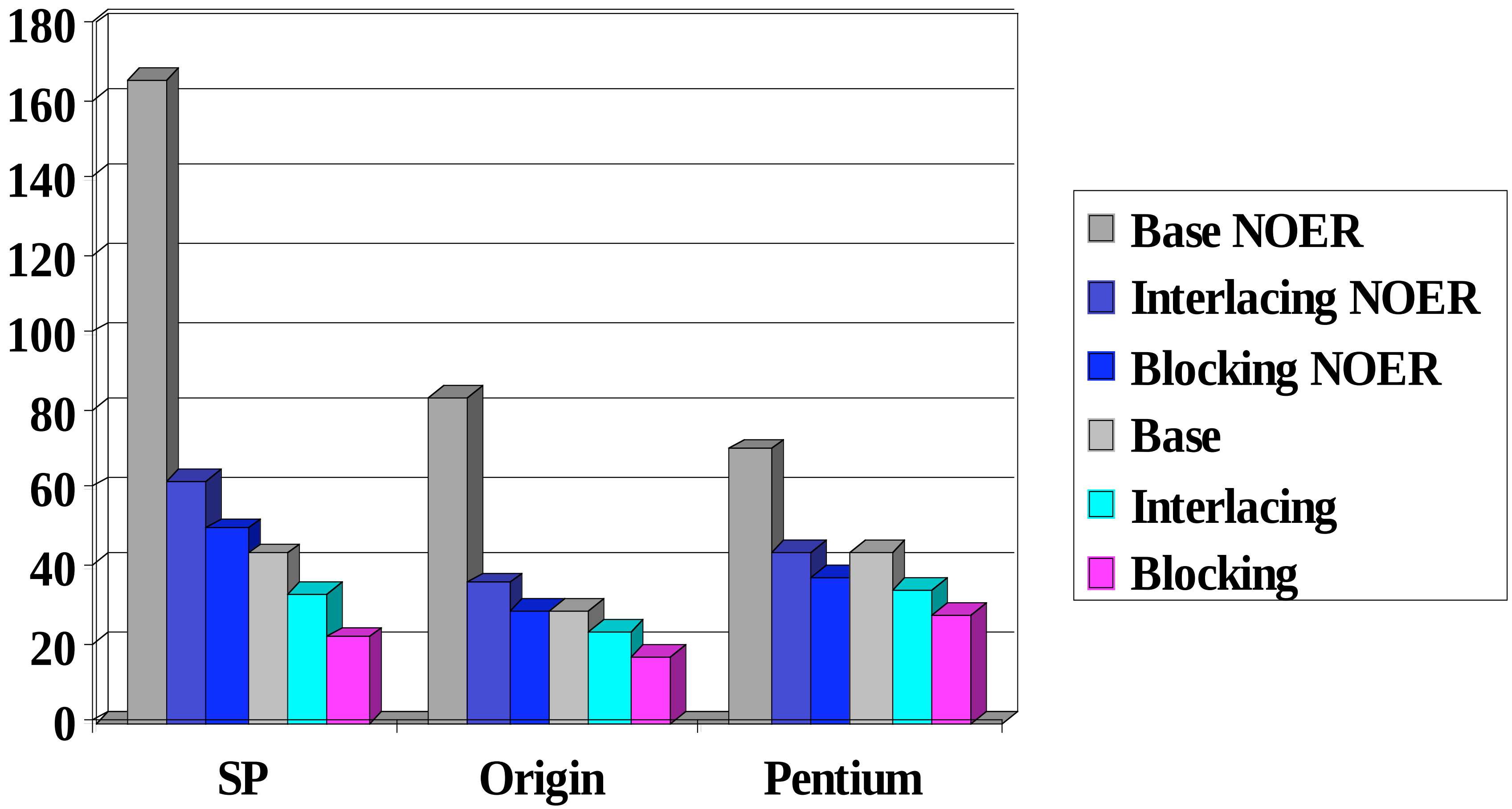


Myth #2

- Parallel computers are hard to program
- Reality: Relative to uni-processors, the difficulty is comparable
  - Even easier
  - More time is (often) spent on per-processor tuning than on parallelism
    - Fun3d — 1999 Gordan Bell winner (special)
    - QMC — Nuclear structure code on BG/L

Sequential Performance—Time/iteration

SP: IBM P2SC ("thin"), 120 MHz, cache: 128 KB data and 32 KB instr
Origin: MIPS R10000, 250 MHz, cache 32 KB data/32KB instr/4MB L2
Pentium: Intel Pentium II, 400 MHz, cache: 16KBdata/16KB instr/512 KB L2



Myth #3

- Shared memory architectures (hardware) will save the day (for software)

- Reality: A system with uniform memory access time *might* save the day, but the laws of physics make that unlikely

Hardware Realities

- Performance is determined by memory performance

  (Well, it is a major contributor)
- Memory system design for performance makes system performance less predictable
- Fast memories possible, but
  - Expensive ($,£,¥,€)
  - Large
  - Power hungry
- Programming models and algorithms we develop that don’t take these realities into account may be irrelevant

Uniprocessor Memory Performance

- AlphaServer 8200 read latencies (3.33ns clock)

Memory Level	Latency		Bandwidth GB/sec
	ns	cycles	
Cache	6.7	2	4.8
L2 Cache	20	6	4.8
L3 Cache	26	8	0.96
Main	253	76	1.2
DRAM	60	18	.03-.1

Note that `a[i] = b[i] * c[i]` requires 7.2 GB/sec

Parallel Processor Memory Performance

- Average read latency

CPUs	AlphaServer		Origin2000	
MHz	300		195	
	ns	cycles	ns	cycles
1	176	53		
2	190	57	313	61
4	220	66	405	79
8	299	117	528	103
16			641	125
32			710	138
64			796	155
128			903	176

… and worse (cluster and cluster-like scalable systems)

More recent measurements:
21264 (500MHz): 82 cycles just to L2

SGI O2000 (300MHz)
101 cycles to L2

SunFire 6800
(900 MHz)
198—252 cycles on L2 miss)

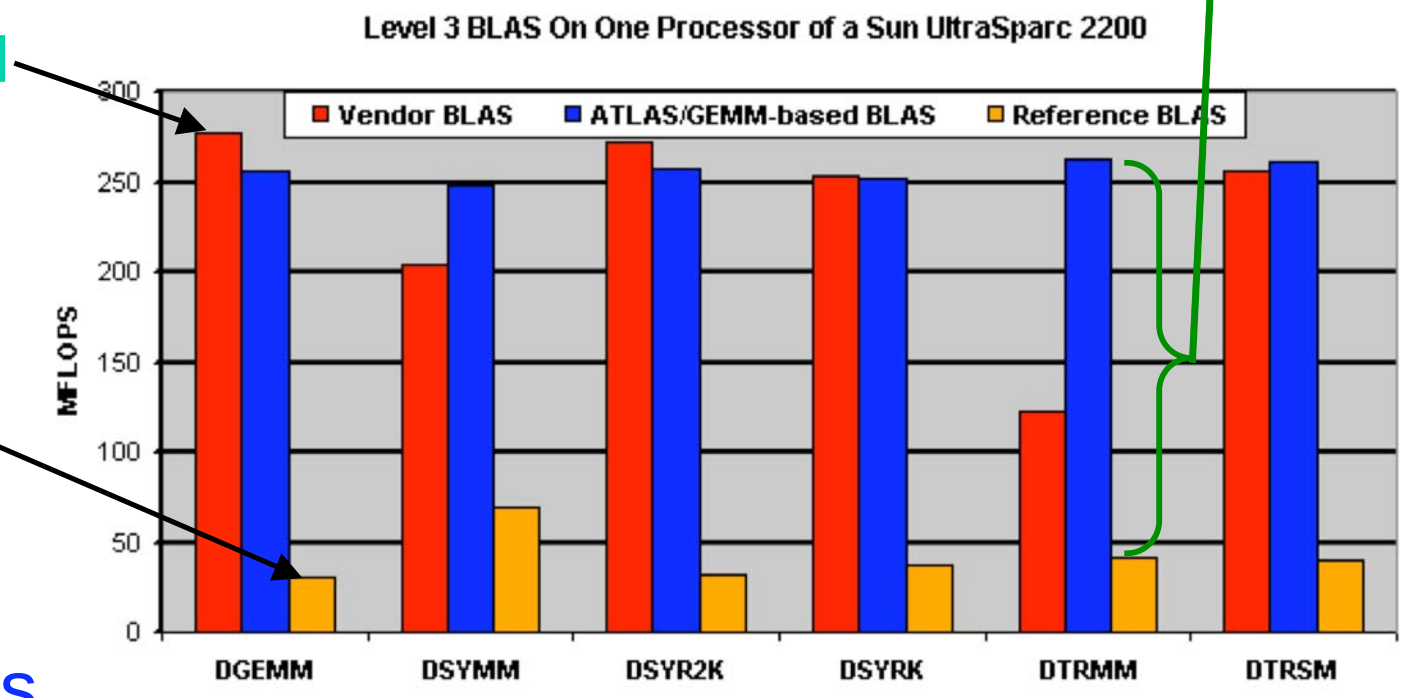
Massively Parallel Computing and Performance

- Poor *per processor performance* (relative to peak) is a common argument *against* massively parallel computing
  - Just get better performance and massively parallel computing isn’t necessary
- The *source* of poor per processor performance is the difficulty of making effective use of the memory system. This problem only gets worse in parallel systems
  - But complexity of problem argues that a common solution must be found

Other Myths

- Compilers will solve the parallel programming problem
  - Pro: no new algorithms needed
  - Con: compilers still can’t handle dense matrix-matrix multiply
- SMPs and shared memory will make performance programming easier
  - 1998 Gordon Bell Prize winners were uniprocessors; 3 of 4 winners in 1999 were uniprocessors
  - MPI remains the most effective programming model for managing data placement, locality, and access (Eeek!)
- Multithreaded architectures will save the day
  - Large latencies require enormous numbers of threads
- ***Denial is not a solution***

The Compiler Will Handle It (?)



Large gap between natural code and specialized code

Hand-tuned

Compiler

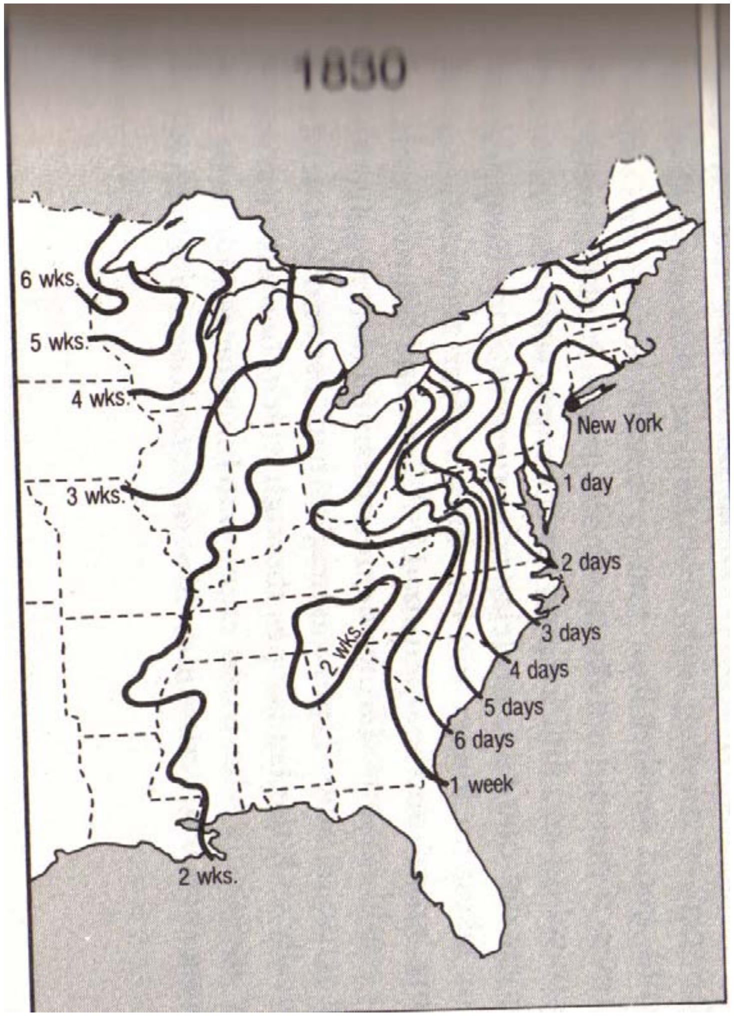
From Atlas

Enormous effort required to get good performance

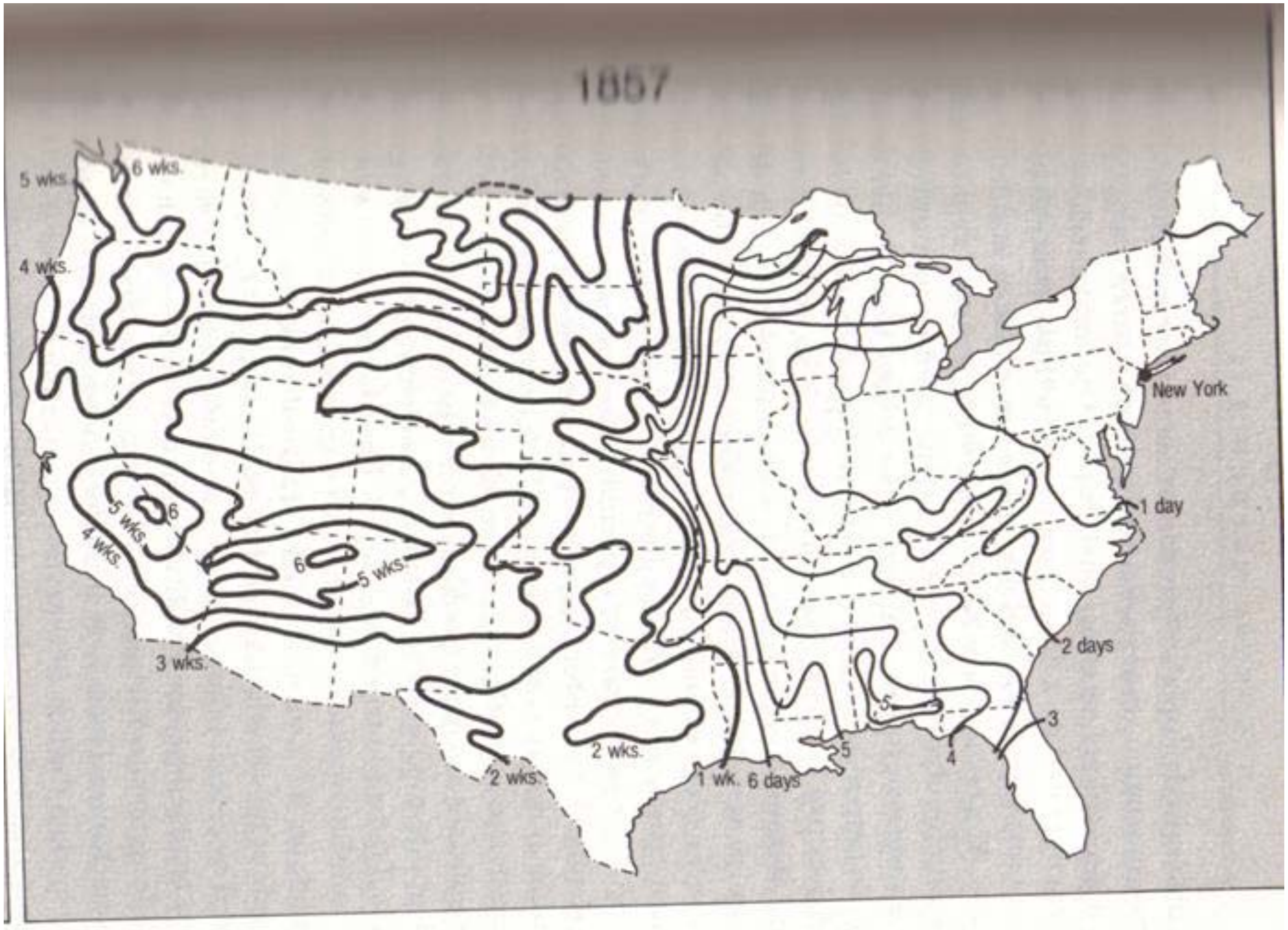
Myth #4: The Grid

- What is the Grid for?
  - Metacomputing?
  - Virtual organizations?
  - Data sharing?
  - Collaboration?
- What is the real problem?
  - My view: Collaboration and information sharing
  - What have been the transitions in collaboration?

Rate of Travel from NY — 10000BC to 1830AD



Rate of Travel from NY — 1857



Travel Times Today

- Chicago – NY
  - 2 hours to ORD, 2 hours flight, 1 hour from LGA to hotel
  - 5 hours
    - Doesn’t could 1 day weather delay
- Chicago – Beijing
  - 2 hours to ORD, 13 hour flight, 1 hour to hotel
  - 16 hours
- And in the future
  - “Allow 6 hours for takeoff and 6 for landing, and assuming no traffic snarls near the world of destination, and we have a ship which can go anywhere in 13 and nowhere in less than 12…”
    - Regional president of General Products, to Beowulf Shaeffer, “At the core”
- The point is
  - The next qualitative change in how we “meet” (baring teleportation) must be virtual

Is the Grid like the Power Grid?

- Similarities
  - Commodity resources
  - Resource can’t be stored (cycles are lost if not used)
- But
  - Wrong direction
    - Power comes into my home, to use as I wish and control
    - Computing tasks (including my data) go out of my home, to where someone else controls them
  - Not standardized
    - Executables won’t run as is on other platforms
  - Its not just cycles
    - It’s the data
  - No single parameter measure of resource (no counterpart to the Watt)
    - Makes resources less fungible
  - Emphasizes consumption of resources

Is the Grid more like the air travel network?

- Airway analogy (railways are so 20th century)
- A better fit to real grid use?
  - A different level of personal and organizational interactions
  - Reduce time to interact and to move commodity (data)
- Multi-dimensional service metrics
  - seats, schedules, aircraft, cost vs. priority
- Even lost luggage
  - Data sizes and transfers can exceed TCP checksum
- And threat if someone falsifies identity
  - We continue to need better security models; even rental cars require a drivers license (id) *and* a credit card (guarantee of payment; basically a second, independent authorization)
- Emphasizes connectivity

Opportunities Overview

- Enabling Computational Science
  - High-end computing
    - Programming models
    - Parallel I/O (and reuse of latency-tolerant concepts to distributed data)
    - Code transformations for legacy software and software evolution
    - Ensuring that applications are ready for the *next* generation of machines
- Enabling Collaborations
  - Interaction tools
  - Using ETF to connect HPC facilities
  - Making tools transparent
    - Match needs of users (scientists and engineers)

Opportunities

- Data Sharing
  - Not just read-only data
    - Some grid I/O proposals are syntax only – no semantics (!!!)
  - But update-rarely and write-once data are important
    - One interesting file system concept — Immutable files
    - Don’t forget the most common data write-once, read-never
  - Many other capabilities will be enabled by robust, semantically-clean interfaces
    - Federated data, discovery, serendipity, …
- Virtual Meetings
  - What needs to be done to make these as easy to schedule as a local conference room?
  - Better (and easier!) then teleconferences?
  - Always-on AG (mini)nodes?

Conclusions

- Orient HPC towards scientists needs
- Many opportunities
  - High-end applications
    - Next-generation parallel architectures
    - Software tools
    - Programming models
  - Collaborations
    - Transparent tools
    - UK eScience examples