Argonne NATIONAL LABORATORY

... for a brighter future

U.S. Department of Energy

UChicago Argonne LLC

Office of Science U.S. DEPARTMENT OF ENERGY

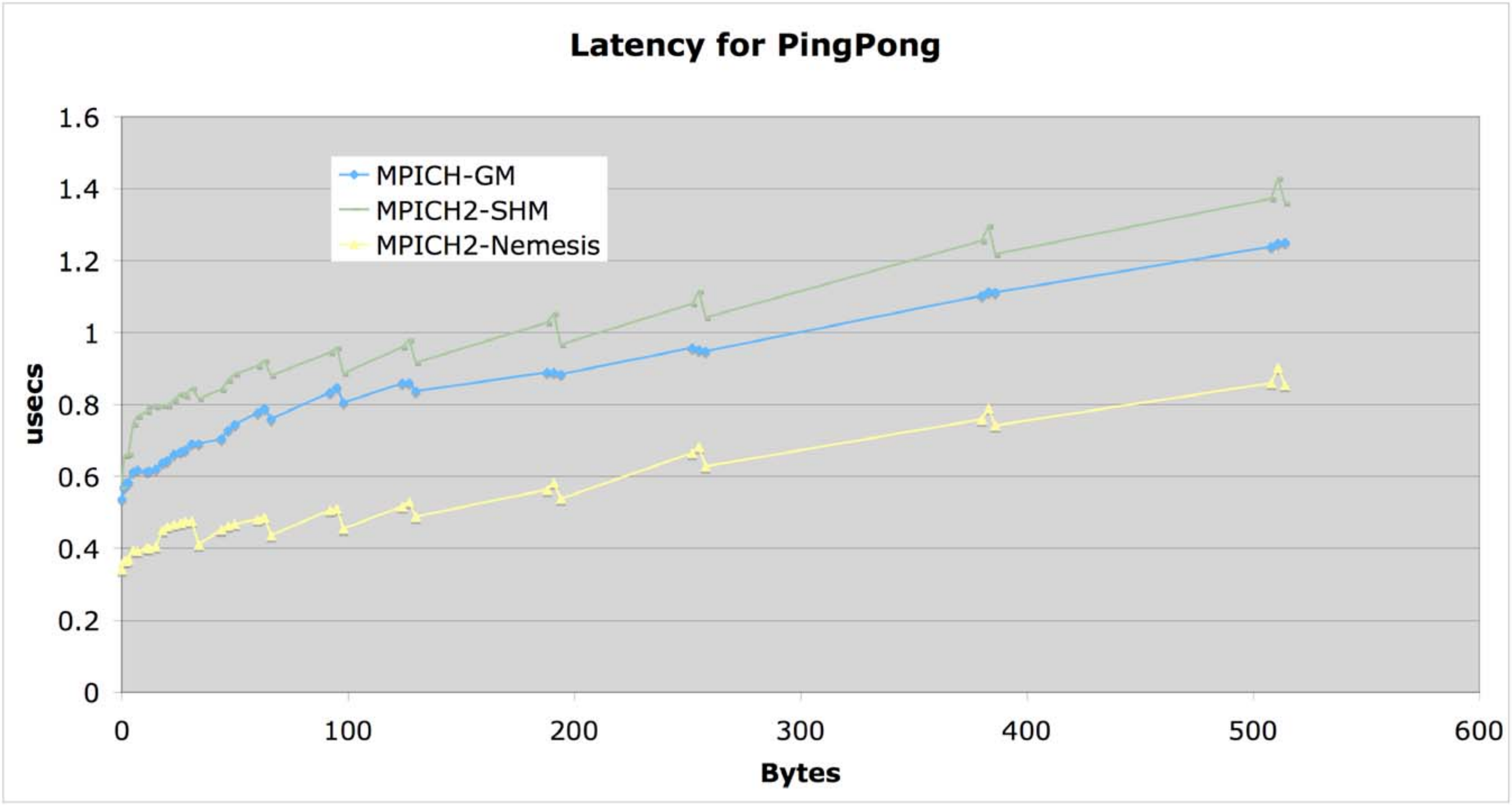
A U.S. Department of Energy laboratory managed by UChicago Argonne, LLC

Half full or half empty?

William Gropp
Mathematics and Computer Science
www.mcs.anl.gov/~gropp

MPI on Multicore Processors

- Work of Darius Buntinas and Guillaume Mercier
- 340 ns MPI ping/pong latency
- More room for improvement (but will require better software engineering tools)



Everything Doesn't Double

Travel times from New York to Chicago, from 1,000,000BC to 1830

(Further west is infinite - unlikely to survive the trip)

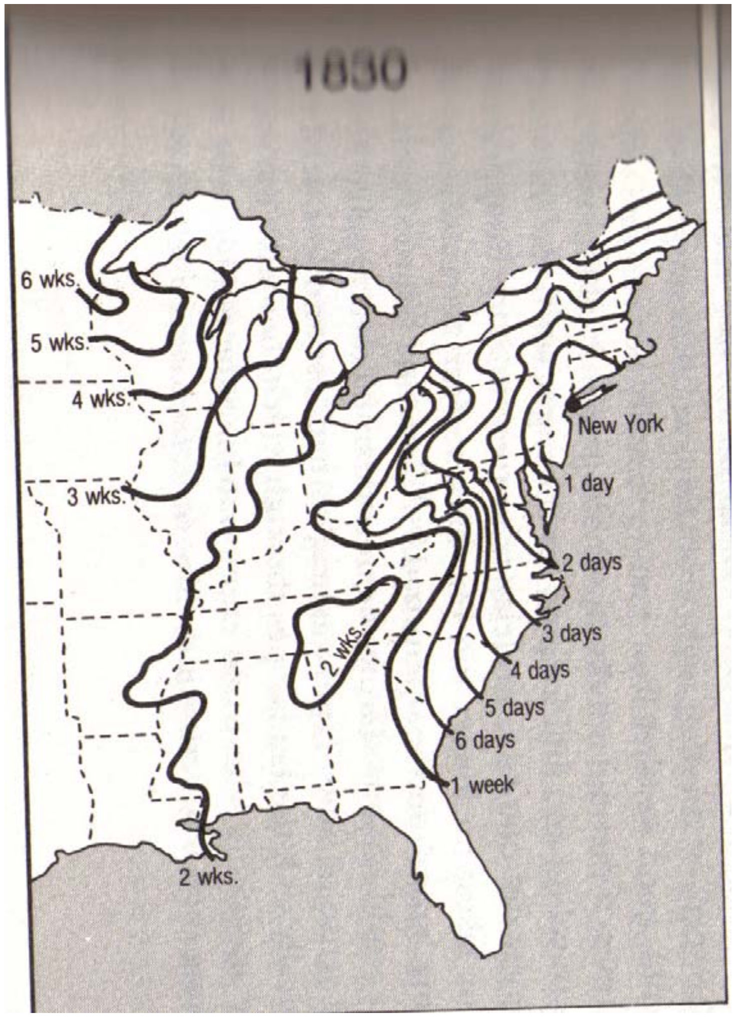
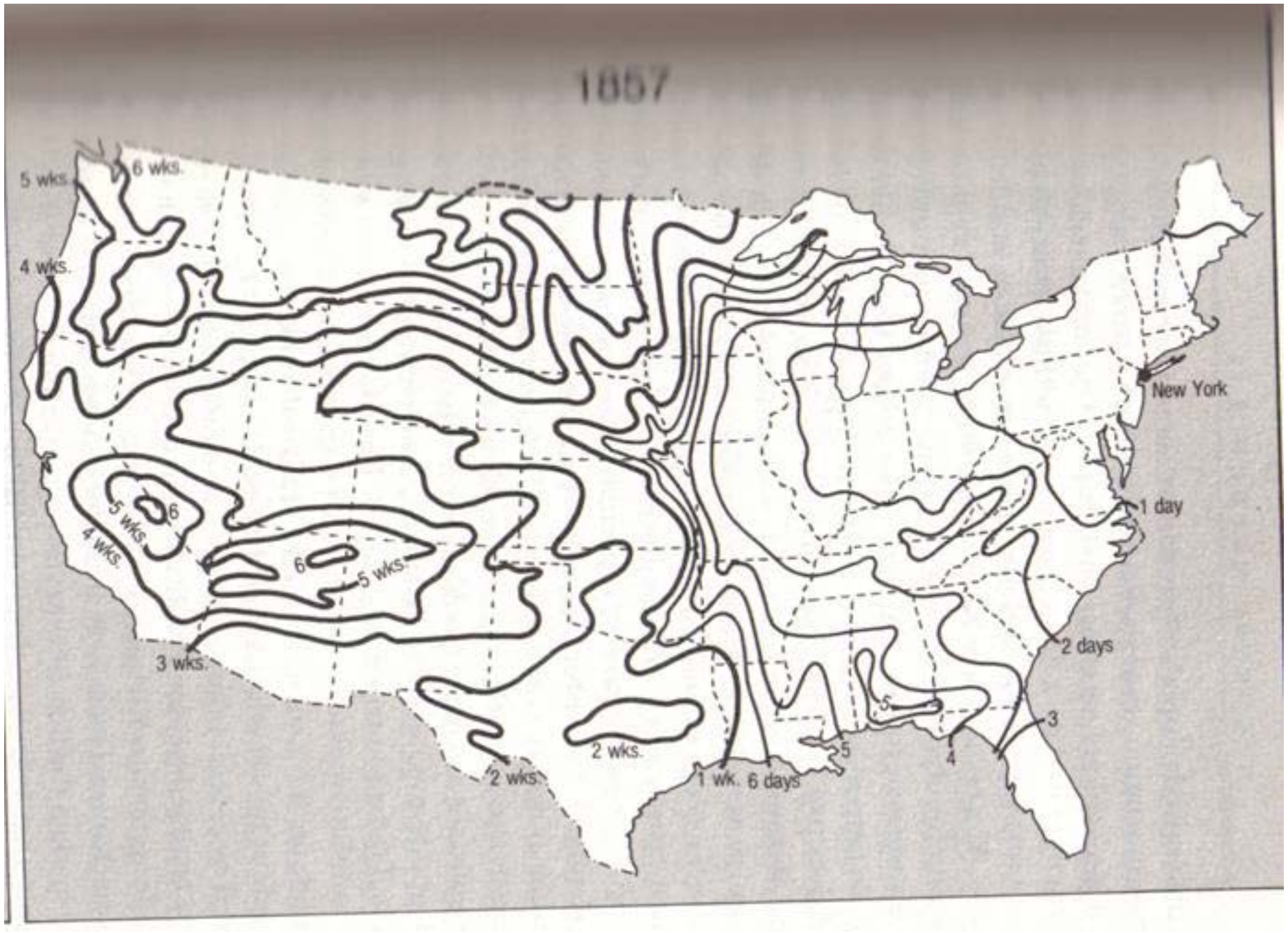


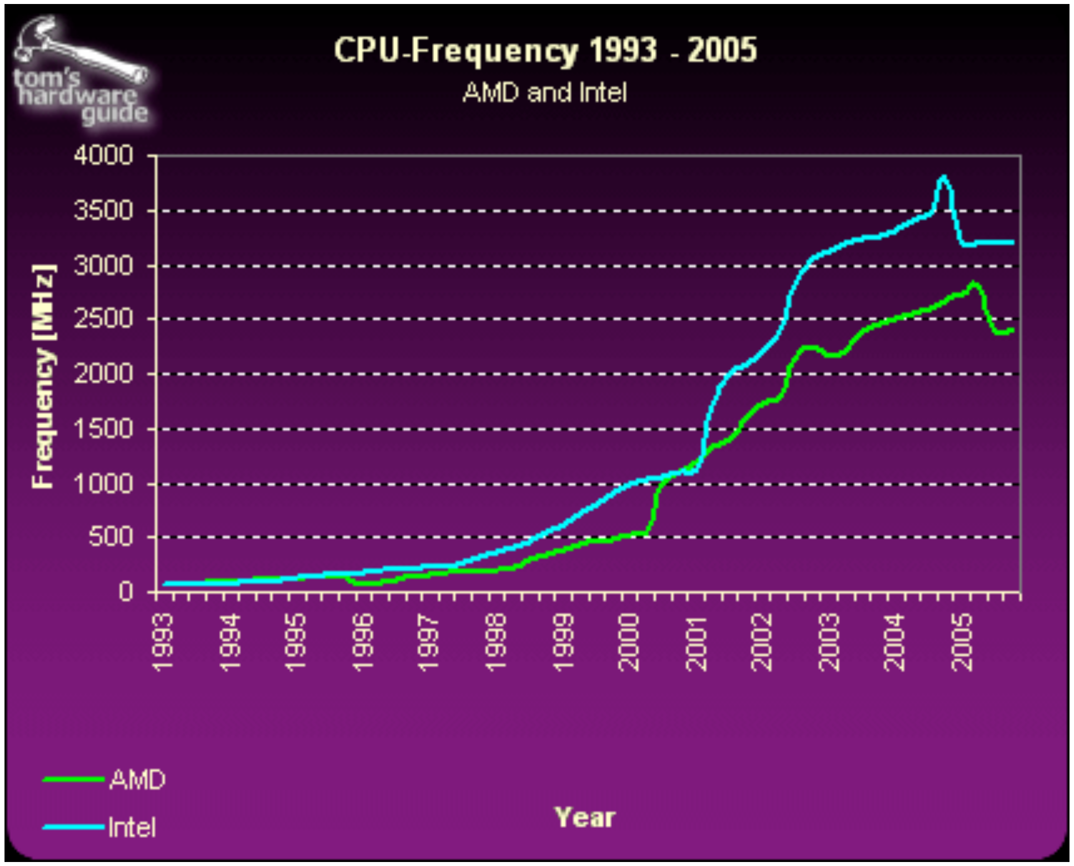
Image from "Nature's Metropolis"

Rate of Travel from NY – 1857



What's the time today? 1hr to airport, 2hrs at airport, 2hrs in air, 1 hr to destination = 6hrs; a factor of only six from 1857.

CPU Speeds Stagnant



Doubling happens until it doesn't

- From http://www.tomshardware.com/2005/11/21/the_mother_of_all_cpu charts_2005/

What will multicore look like in 6 years?

- Already 8 cores in some chips; if double every 1.5 years, we'll have 128 cores in a chip.
- Will these be cache coherent?
  - Maybe, but most likely by making accesses to other caches relatively expensive, or …
  - "When [a] request for data from Core 1 results in a L1 cache miss, the request is sent to the L2 cache. If this request hits a modified line in the L1 data cache of Core 2, certain internal conditions may cause incorrect data to be returned to the Core 1."
  - What do you think this is going to be like with 64 cores? 1024?
- How well will prefetch work in hiding memory latency with 128 concurrent yet different memory references?

Memory Access

- Consider the balance of computation to memory access
  - We look at latency and bandwidth now
  - Consider latency: Number of cycles to access main (off chip) memory
  - For any part that of the code that is not parallelized, for a multicore chip, the effective latency is really the sum(over cores) of the cycles to access main memory, since that is the lost (or overlappable) work while waiting for memory
    - *Multicore doesn’t just stress bandwidth, it increases the need for <u>perfectly</u> parallel algorithms*
- All systems will look like attached processors - high latency, low (relative) bandwidth to main memory

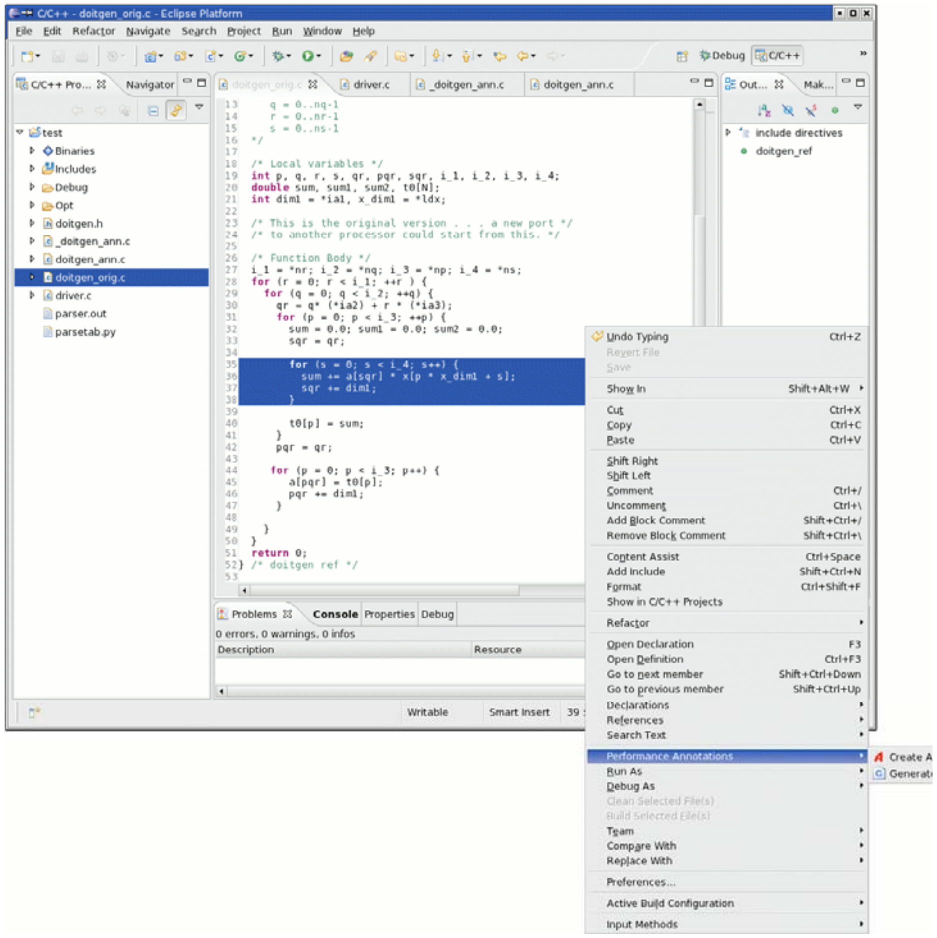
Breaking the Assumptions

- Don’t have any off-chip memory
  - Consequence: Need algorithms, programming models, and software tools to work in more limited memory (a few GB)
- Have off-chip memory, but manage it more effectively
  - Consequence: Need to find a true, general-purpose hardware/software model
- Overlap latency with split operations
  - Consequence: Need to find massive amounts of concurrency; need to manage the programming challenges of split operations (these are hard for programmers to use correctly - may be an opportunity for formal methods)

Three Ways to Make Multicore Work

- **Number 3:**
- Software Engineering: Better ways to restructure codes
  - E.g. Loop fusion (vs the more maintainable and understandable to the computational scientist approach of using separate loops). Need to present the computational scientist with the best code to maintain and change, while efficiently managing the creation of more memory-bandwidth-friendly codes. Must manage the issues mentioned by Ken
  - Library routine fusion (telescoping languages)
    - *While libraries provide good abstractions and often better implementations, those very abstractions can introduce extra memory motion*
  - Tools to manage locality
    - *Compile time (local/global?) and Runtime (memory views, perhaps similar to file views in parallel file systems)*

Source code transformation tool for performance annotations, thanks to Boyanna Norris



```
   q = 0..nq-1
   r = 0..nr-1
   s = 0..ns-1
*/

/* Local variables */
int p, q, r, s, qr, pqr, sqr, i_1, i_2, i_3, i_4;
double sum, sum1, sum2, t0[N];
int dim1 = *ia1, x_dim1 = *ldx;

/* This is the original version . . . a new port */
/* to another processor could start from this. */

/* Function Body */
i_1 = *nr; i_2 = *nq; i_3 = *np; i_4 = *ns;
for (r = 0; r < i_1; ++r ) {
  for (q = 0; q < i_2; ++q) {
    qr = q* (*ia2) + r * (*ia3);
    for (p = 0; p < i_3; ++p) {
      sum = 0.0; sum1 = 0.0; sum2 = 0.0;
      sqr = qr;

      for (s = 0; s < i_4; s++) {
        sum += a[sqr] * x[p * x_dim1 + s];
        sqr += dim1;
      }

      t0[p] = sum;
    }
    pqr = qr;

   for (p = 0; p < i_3; p++) {
      a[pqr] = t0[p];
      pqr += dim1;
    }

  }
}
return 0;
} /* doitgen ref */
```

0 errors, 0 warnings, 0 infos

Three Ways to Make Multicore Work

- **Number 2:**
- Programming Models: Work with the system to coordinate data motion
  - Vectors, Streams, Scatter/Gather, …
  - Provide better compile and runtime abstractions about reuse and locality of data
  - Stop pretending that we can provide an efficient, single-clock-cycle-to-memory, programming model and help programmers express what really happens (but maintaining an abstraction so that codes are not instance-specific)
  - I didn't say programming languages
  - I didn't say threads
    - *See, e.g., Edward A. Lee, "The Problem with Threads," Computer, vol. 39, no. 5, pp. 33-42, May, 2006.*
    - *"Night of the Living Threads", http://weblogs.mozillazine.org/roc/archives/2005/12/night_of_the_living_threads.html, 2005*
    - *"Why Threads Are A Bad Idea (for most purposes)" John Ousterhout (~2004)*
    - *"If I were king: A proposal for fixing the Java programming language's threading problems" http://www-128.ibm.com/developerworks/library/j-king.html, 2000*

Three Ways to Make Multicore Work

- **Number 1:**
- Mathematics: Do more computational work with less data motion
  - E.g., Higher-order methods
    - *Trades memory motion for more operations per word, producing an accurate answer in less elapsed time than lower-order methods*
  - Different problem decompositions (no stratified solvers)
    - *The mathematical equivalent of loop fusion*
    - *E.g., nonlinear Schwarz methods*
  - Ensemble calculations
    - *Compute ensemble values directly*
  - It is time (really past time) to rethink algorithms for memory locality and latency tolerance