# Hardware and Software Environments for Large-Scale Simulation

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

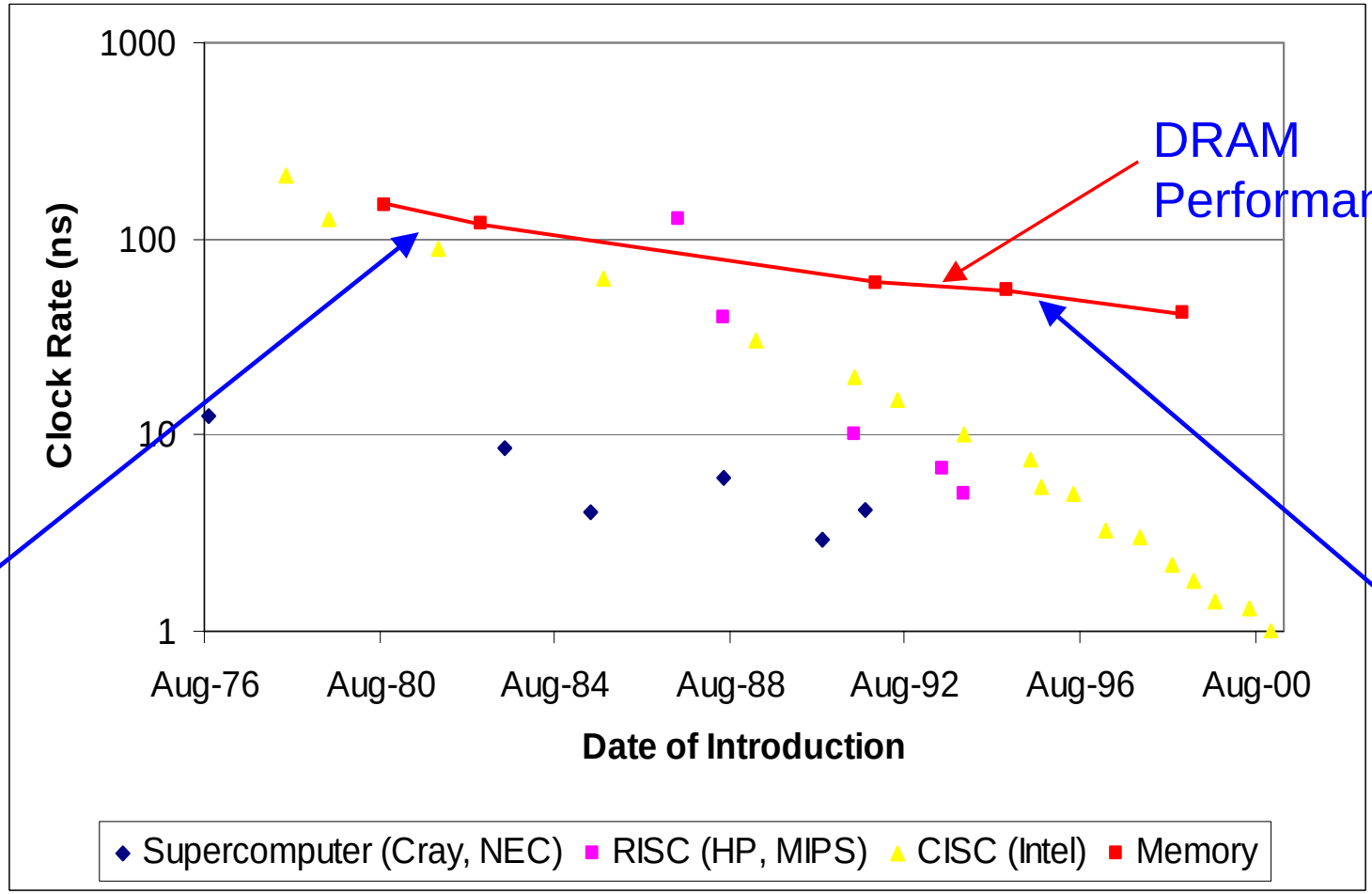
# The Dimensions of a Typical Cluster

- 6.1 m x 2.1 m x 1m
- 1-norm size (airline baggage norm) = 9.2m
- At 2.4Ghz, =
  74 cycles
  (49 x 17 x 8)
- Real distance is greater
  - Routes longer
  - Signals slower than light in a vacuum

# Consequences of the Speed of Light for Computer Architecture

- A “load” operation from anywhere in memory may require 148 cycles (on this cluster), just to move the data at the speed of light
  - Does not include the time to access the data
  - Leadership-class machines are physically larger—thus data is even farther away
  - A “flat” memory is *not possible* if performance is also a requirement
- Fixes require changing programming models:
  - Must separate out initiation from completion (cannot wait on data return)
  - Must be careful about requiring every processor’s view of memory to be the same
    - As in, this does not have any meaning any more
  - Many HPC programming models are moving in this direction
    - A natural fit for MPI (nonblocking operations in MPI-1 + MPI-2)
    - CAF, UPC also provide “split” operations for access to remote data
- As if this wasn’t bad enough…

# CPU and Memory Performance



# CPU / Memory BW Performance Ratio

| System | CPUs | CPU Peak | Memory BW | Ratio |
|---|---|---|---|---|
| NEC SX-7 | 1 | 8825.6 | 4415.9 | 2.0 |
| Cray X1 | 1 | 12800 | 3002.9 | 4.3 |
| IBM eServer p690+ | 32 | 217600 | 5133 | 42.4 |
| AMD Opteron 248 | 1 | 4400 | 393 | 11.2 |
| Generic P4-1400 | 1 | 2800 | 196.9 | 14.2 |
| Cray C90 | 1 | 960 | 1187.6 | 0.8 |
| AMD 486DX-50 | 1 | 10 | 2.9 | 3.4 |

http://www.cs.virginia.edu/stream/standard/Balance.html

Note: CPU / Main Memory *latency* can be as important, particularly for programmability

# Node Performance

- Current laptops now have a peak speed (based on clock rate) of over 2 Gflops (20 Cray1s!)
- Observed (sustained) performance is often a small fraction of peak
- Why is the gap between “peak” and “sustained” performance so large?
- Lets look at a simple numerical kernel

# Sparse Matrix-Vector Product

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

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

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

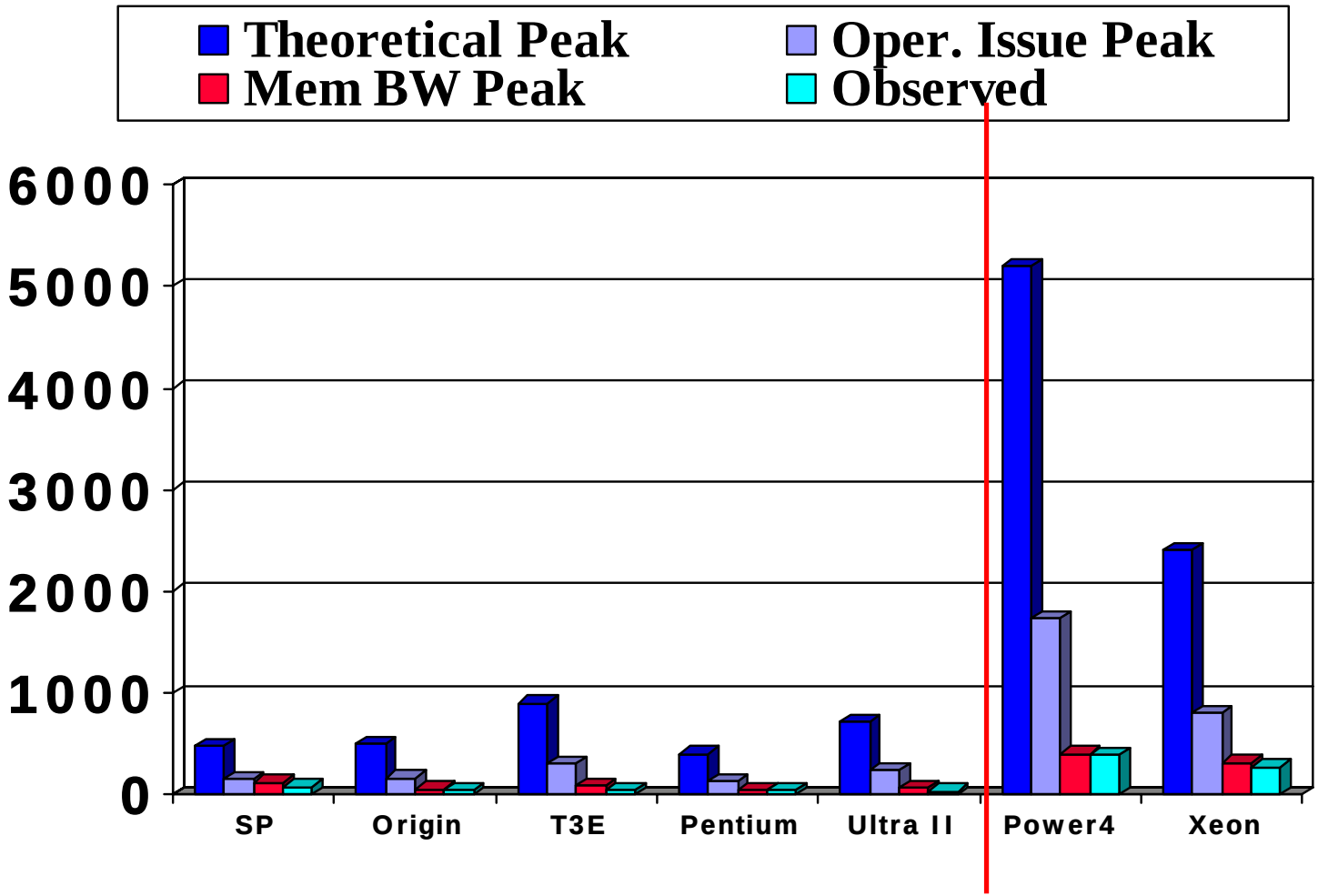
# Simple Performance Analysis

- Memory motion:
  - nnz (sizeof(double) + sizeof(int)) + n (2*sizeof(double) + sizeof(int))
  - Assume a 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

# Realistic Measures of Peak Performance

## Sparse Matrix Vector Product

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



# What About CPU-Bound Operations?

- Dense Matrix-Matrix Product
  - Most studied numerical program by compiler writers
  - Core of some important applications
  - More importantly, the core operation in High Performance Linpack
    - Benchmark used to “rate” the top 500 fastest systems
  - Should give optimal performance...

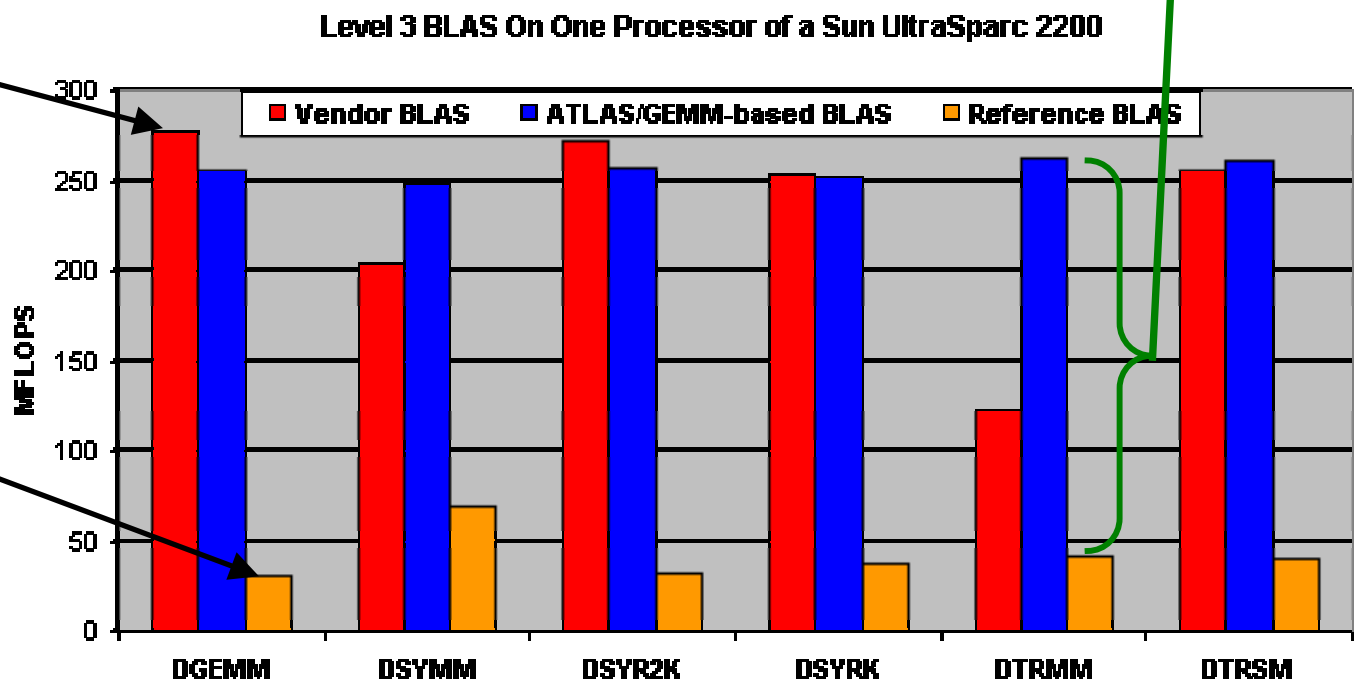
# The Compiler Will Handle It (?)

Large gap between natural code and specialized code

Hand-tuned

Compiler

From Atlas




Enormous effort required to get good performance

# Trends in Computer Architecture I

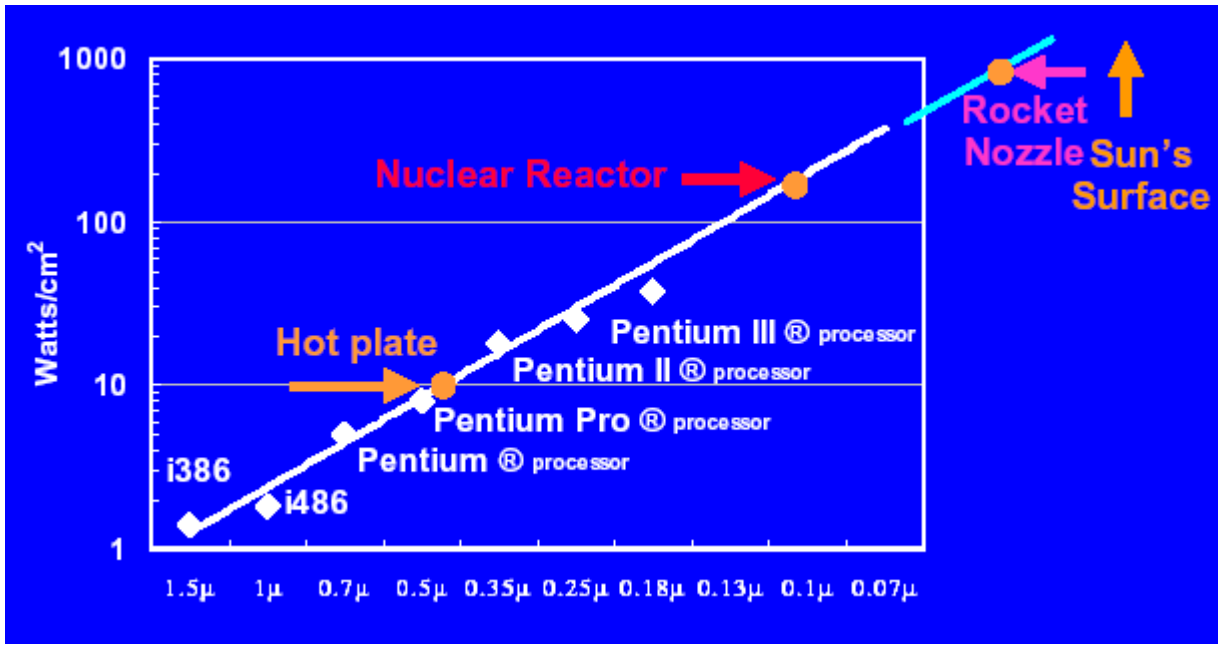
- Latency to memory will continue to grow relative to CPU speed
  - Latency hiding techniques require finding increasing amounts of independent work: Little's law implies
    - Number of concurrent memory references = Latency * rate
    - For 1 reference per cycle, this is already 100–1000 concurrent references

# Trends in Computer Architecture II

- Clock speeds will continue to increase
  - The rate of clock rate increase has increased recently
  - Light travels 3 cm (in a vacuum) in one cycle of a 10 GHz clock
    - CPU chips won't be causally connected within a single clock cycle, i.e., a signal will not cross the chip in a single clock cycle
    - Processors will be parallel!

# Trends in Computer Architecture III

- Power dissipation problems will force more changes
  - Current trends imply chips with energy densities greater than a nuclear reactor
  - Already a problem: Last year, Consumer Reports looked at the likelihood of getting a serious burn from your laptop!
  - Will force new ways to get performance, such as extensive parallelism, even in laptops



# Trends in Computer Architecture IV

- Three branches:
  - Commodity
    - Exploit consumer computing and economies of scale
    - Beowulf clusters,
  - "Commodity Process"
    - Tune commodity for special needs
      - Exploits commodity advantages but tunes for HPC needs
      - Preserve investment in CPU design, compilers
    - IBM BlueGene
  - Custom
    - Still use commodity fab
    - Parallel Vector machines; alternative highly parallel architectures
- Each is a tradeoff
- Are we doomed?

# Many Efforts

- Commercial efforts (most focused on commodity market)
- DARPA HPCS (hardware/software)
- DOE SC SciDACs, particularly PERC, SSS, SDM (software), others for algorithms
- Many new architecture projects in the “small”
  - TRIPS, STREAMS, QCDoC, Blue Gene, Strider, ...
  - Many fine-grain projects (MIND, IRAM, SHAMROCK,...)
- Others in technology areas (memory, networking, ... )

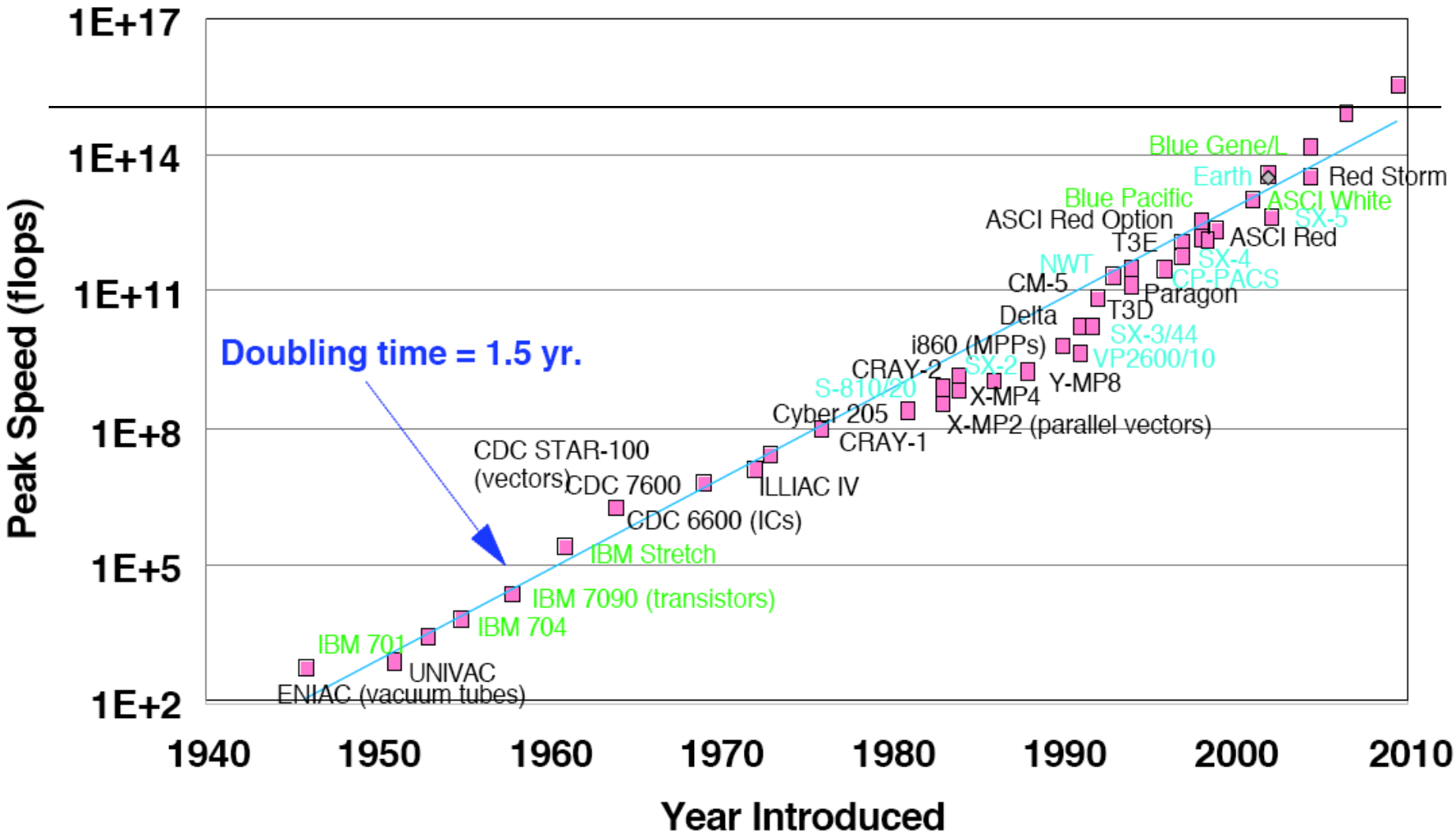
# Solutions to the Performance Problem

- More raw speed
  - Improvement in clock rates will slow down, so raw speed will come from increasing parallelism
  - Requires finding enough concurrent work
- Handling latency
  - Split memory operations into init and complete
    - Multithreading can help hide latency
      - Limited by Little's law (nt = Latency/clock cycle)
  - Vector operations describe many memory references with a single instruction
    - Limited by structure of memory references and Little's law (short vectors good but Little's law => increasing vector length)

# More Solutions to the Performance Problem

- Reduce Heat
  - Slower clock (increase parallelism to compensate)
  - Simpler operations (fewer gates per op)
  - Lower voltage (see clock)
- More Bandwidth
  - Most amenable to engineering (e.g., optical)
  - Maintaining high bisection bandwidth for significant parallelism is expensive in hardware and latency
    - But bisection bandwidth is min(all pairs); easy to use as a bound but often stricter than required by applications
- Rethink architecture
  - Several projects placing computing in or near memory
    - Reduces power
    - High *bandwidth* to local memory much cheaper
    - Trades simpler processor for greater numbers of processors (more ops per gate, but fewer ops/sec/processor)

# Supercomputer Peak Performance



# Caveats

- Peak Performance a poor metric
  - No fast machine achieves a significant fraction of peak on *all* codes
- Fraction of peak is an irrelevant metric
  - How fast does your car go? How often do you drive it that fast? Why did you buy an engine that powerful if you aren't using at least 80% of it all the time?
- Cost per delivered performance is more relevant
  - Cost should be fully burdened — machine + operation + porting + tuning + software maintenance + ...
  - No machine uniformly good here either
    - Vector machines excellent for algorithms and codes with sufficient structure
    - Superscalar machines (particularly those with high memory bandwidths/processor) excellent (by cost/delivered performance) for codes that are highly adaptive (e.g., sparse matrix, adaptive mesh) or that have high memory locality

# Software for HPC

- Software components
  - Provides better ways to inject algorithmic improvements
  - Handle on (semi)automatic generation of more efficient (tuned) code
  - Handle on performance and correctness contracts
- Hierarchical view of software
  - Can be a better match to hierarchical nature of hardware *and* problems
  - Allows evolutionary adaptation to changing hardware, software, and algorithms

# Neglected Excitement

- Interactivity
  - Scalable interactive tools (R, Matlab, etc.)
- Integration with desktop tools
  - Backend servers that can work in concert with desktop systems and room oriented systems (smart spaces)
- Persistent Data
  - High-performance persistent data APIs
  - Scalable data infrastructure
- Visual Output
  - Integrating scalable systems with visualization devices

# Designing Algorithms for Real Hardware

- Dense matrix-matrix example shows that even for well-studied, compute-bound kernels, compiler-generated code achieves only a small fraction of available performance
  - "Fortran" code uses "natural" loops, i.e., what a user would write for most code
  - Others use multi-level blocking, careful instruction scheduling etc.
- Algorithms design also needs to take into account the capabilities of the *system*, not just the hardware
  - Example: Cache-Oblivious Algorithms (http://supertech.lcs.mit.edu/cilk/papers/abstracts/abstract4.html)
  - Example: Vector algorithms

# Is Performance Everything?

"In August 1991, the Sleipner A, an oil and gas platform built in Norway for operation in the North Sea, sank during construction. The total economic loss amounted to about $700 million. After investigation, it was found that the failure of the walls of the support structure resulted from a serious error in the finite element analysis of the linear elastic model." (http://www.ima.umn.edu/~arnold/disasters/sleipner.html)

# Conclusions

- Computers will continue to have multiple levels of memory hierarchy
  - Algorithms must *exploit* them
- Computers will be parallel
  - Algorithms can make effective use of greater adaptivity to give better time-to-solution and accuracy
- Computational power will continue to double roughly every 18 months
  - Not forever, but over the next 5–10 years
  - Much of that power will come from increased parallelism, even in consumer products
- Software *can* evolve to match hardware
  - But it won't spontaneously
- Denial is not a solution
  - The old days are gone for good