Algorithms and Architecture

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

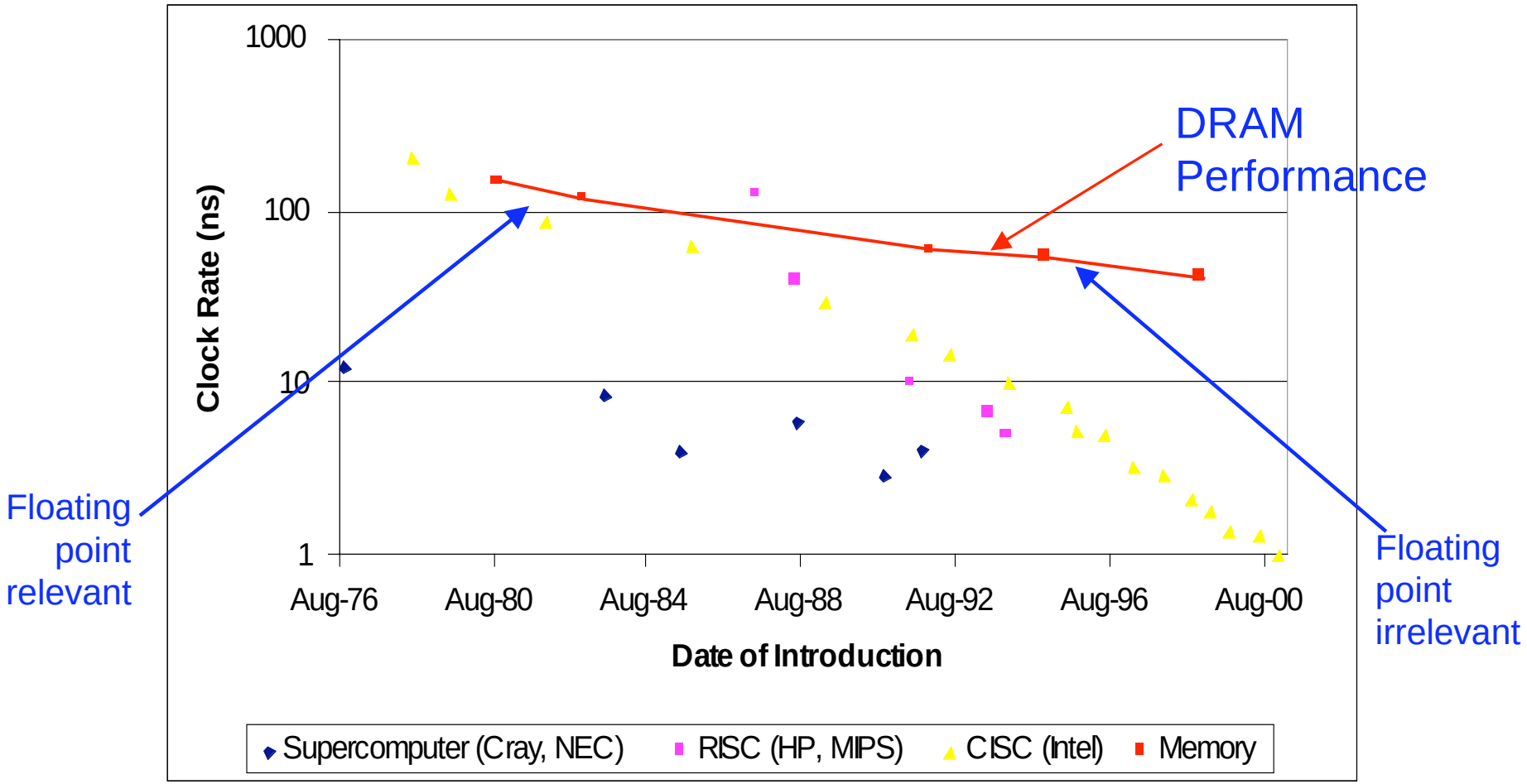
Algorithms

- What is an algorithm?
  - A set of instructions to perform a task
- How do we evaluate an algorithm?
  - Correctness
  - Accuracy
    - Not an absolute
  - Efficiency
    - Relative to current and future machines
- How do we measure efficiency?
  - Often by counting floating point operations
  - Compare to “peak performance”

Real and Idealized Computer Architectures

- Any algorithm assumes an idealized architecture
  - Common choice:
    - Floating point work costs time
    - Data movement is free
  - Real systems:
    - Floating point is free (fully overlapped with other operations)
    - Data movement costs time…a *lot* of time
- Classical complexity analysis for numerical algorithms is *no longer correct* (more precisely, no longer *relevant*)
  - Known since at least BLAS2 and BLAS3

CPU and Memory Performance



Trends in Computer Architecture I

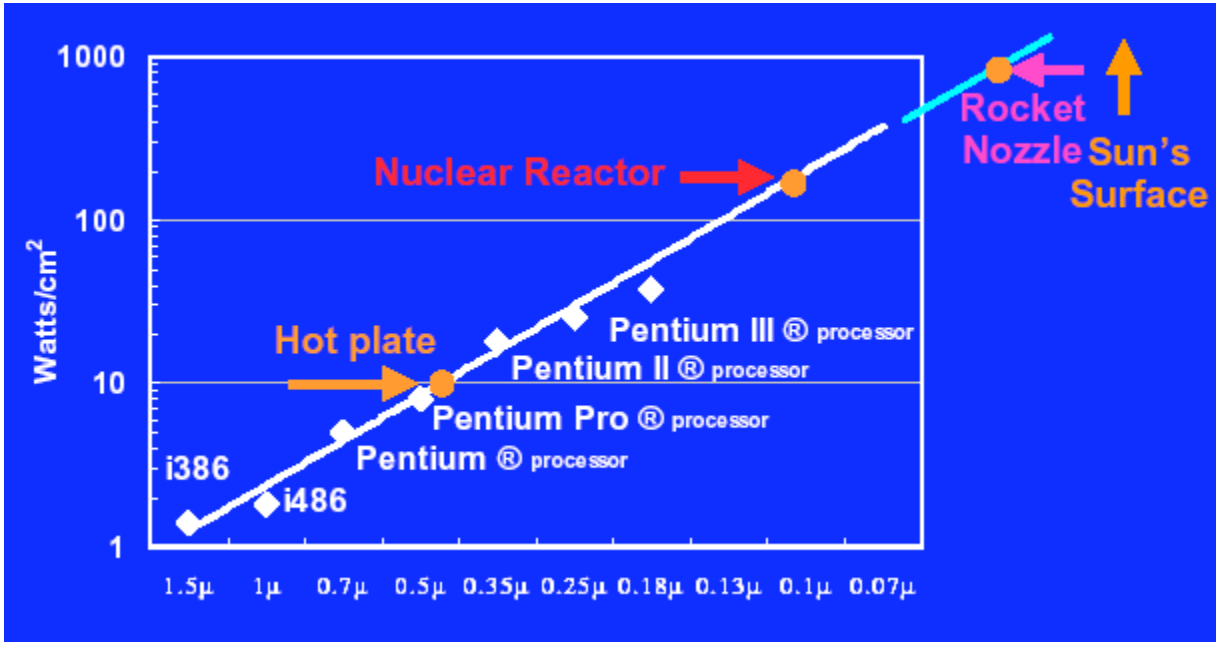
- Latency to memory will continue to grow relative to CPU speed
  - Latency hiding techniques require finding increasing amounts of independent work: Little's law implies
    - Number of concurrent memory references = Latency * rate
    - For 1 reference per cycle, this is already 100–1000 concurrent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

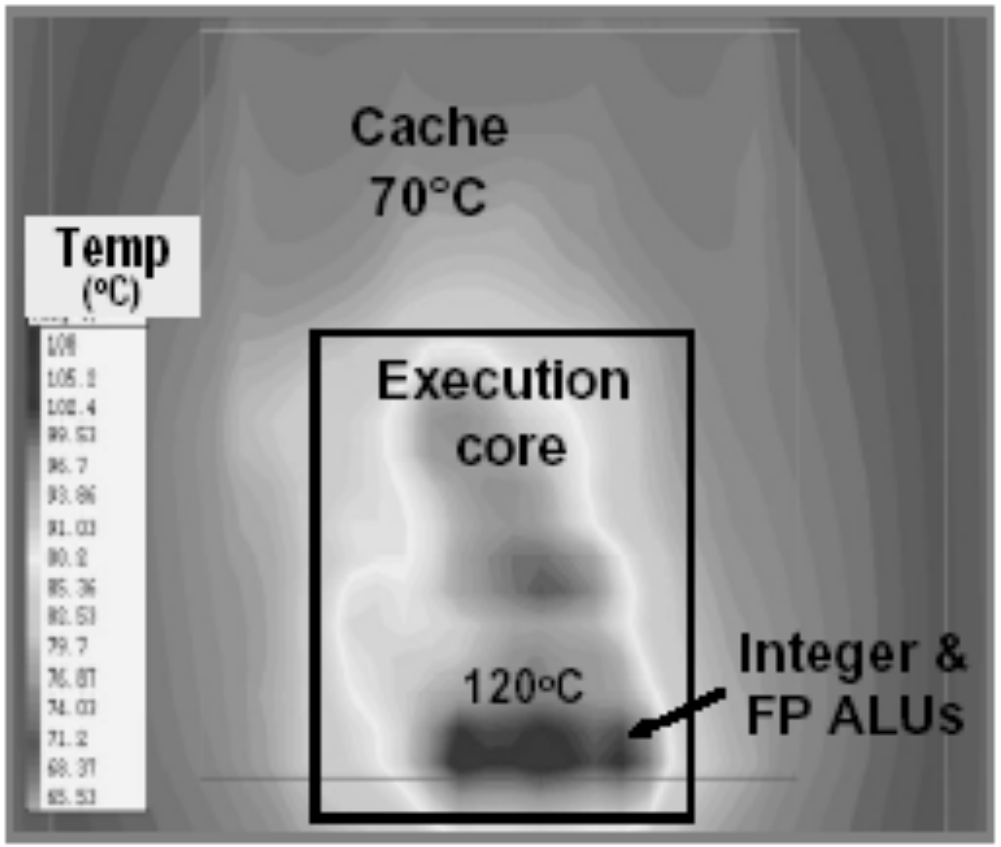
Trends in Computer Architecture III

- Power dissipation problems will force more changes
  - Current trends imply chips with energy densities greater than a nuclear reactor
  - Already a problem: The current issue of consumer reports looks at the likelihood of getting a serious burn from your laptop!
  - Will force new ways to get performance, such as extensive parallelism



Itanium Power Dissipation

- Power is not uniformly distributed across chip
- Peak power densities growing even faster



Consequences

- Gap between memory and processor performance will continue to grow
- Data motion will dominate the cost of many (most) calculations
- The key is to find a computational cost abstraction that is as simple as possible *but no simpler*

Architecture Invariants

- Performance is determined by memory performance
- Memory system design for performance makes system performance less predictable
- Fast memories possible, but
  - Expensive ($)
  - Large (meters^3)
  - Power hungry (Watts)
- Algorithms that don't take these realities into account may be irrelevant

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]

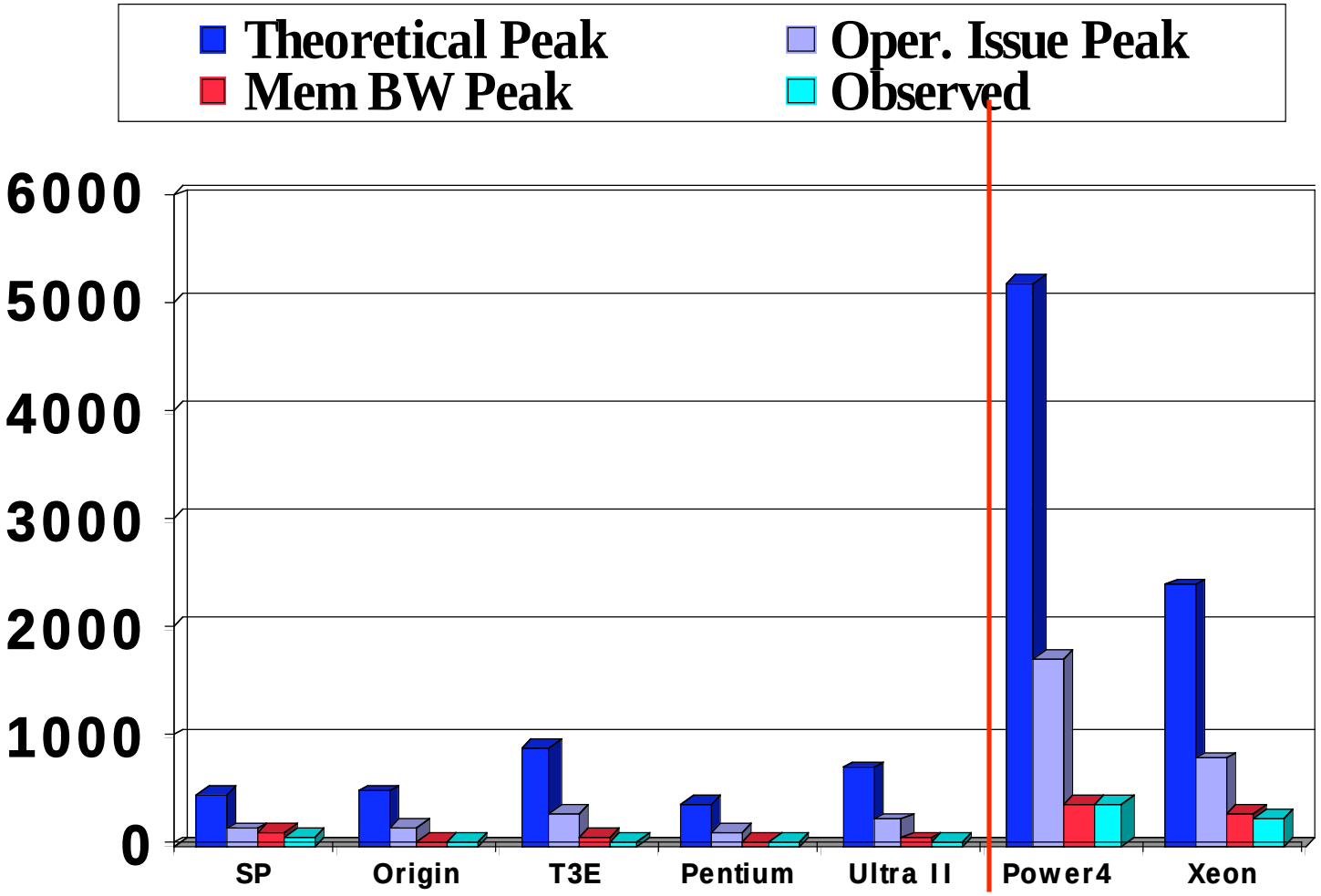
Simple Performance Analysis

- 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…

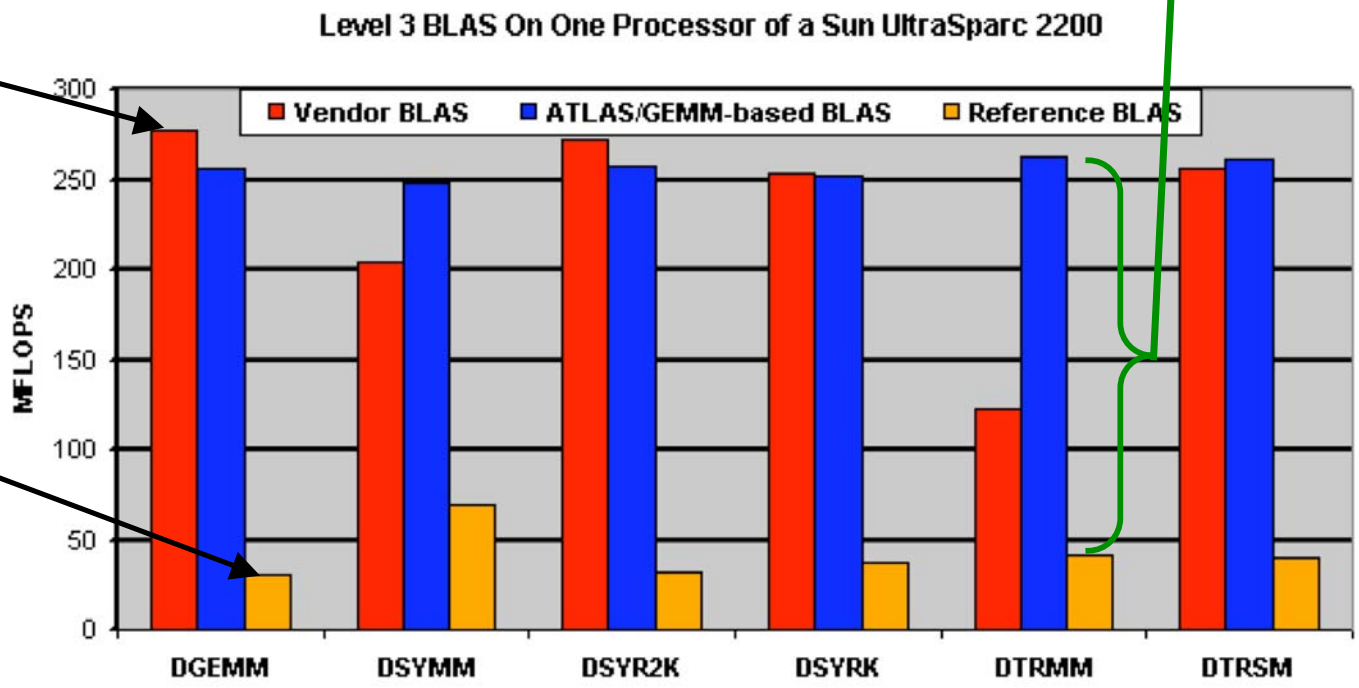
The Compiler Will Handle It (?)

Large gap between natural code and specialized code

Hand-tuned

Compiler

From Atlas



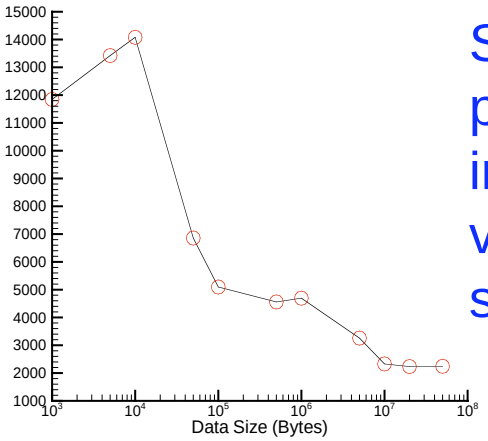
Enormous effort required to get good performance

Performance for Real Applications

- 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)

Challenges in Creating a Performance Model Based on Memory Accesses

- Different levels of the memory hierarchies have significantly different performance
- Cache behavior sensitive to details of data layout
- Still no good calculus for predicting performance



STREAM performance in MB/s versus data size

Interleaved data causes data to be displaced while still needed for later steps

Is Performance Everything?

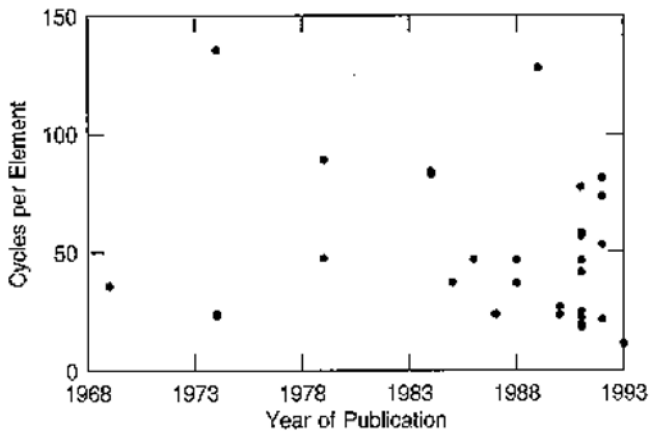
"In August 1991, the Sleipner A, an oil and gas platform built in Norway for operation in the North Sea, sank during construction. The total economic loss amounted to about $700 million. After investigation, it was found that the failure of the walls of the support structure resulted from a serious error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

Correctness and Accuracy

- Many current algorithms designed to balance performance and accuracy
- These choices often made when computers were 10^6 times *slower* than they are now
  - Is it time to re-examine these choices, particularly for applications that are now done on laptops?

Algorithms

- Exploit problem behavior at different scales
  - ♦ Multigrid
  - ♦ Domain Decomposition
    - Generalizes multigrid (or multigrid generalizes DD)
    - Provides a spectrum of robust, optimal methods for a wide range of problems
    - **Nonlinear versions hold great promise**
  - ♦ Continuation
  - ♦ Divide and conquer
  - ♦ Multipole and Wavelets
- Cache-sensitive algorithms
  - ♦ See Karp in SIAM Review 1996
  - ♦ Even the Mathematicians know about this now (McGeoch, AMS Notices March 2001)



Conclusions

- Performance models should count data motion, not flops
- Computers will continue to have multiple levels of memory hierarchy
  - Algorithms should *exploit* them
- Computers will be parallel
  - Algorithms can make effective use of greater adaptivity to give better time-to-solution and accuracy
- Denial is not a solution