Myths in Parallel Programming for Scientific Applications

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++?)

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

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

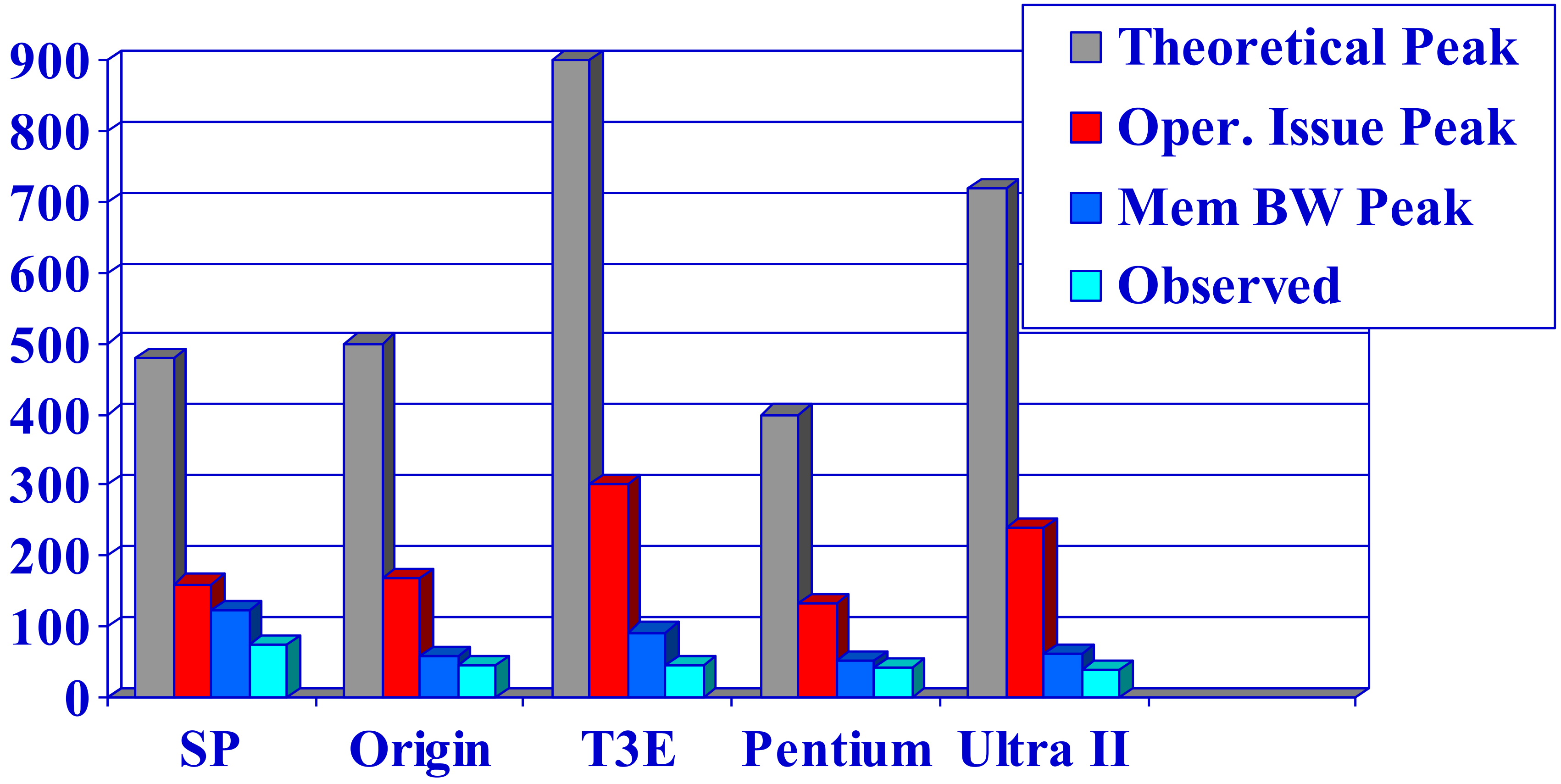
More Performance Analysis

- 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

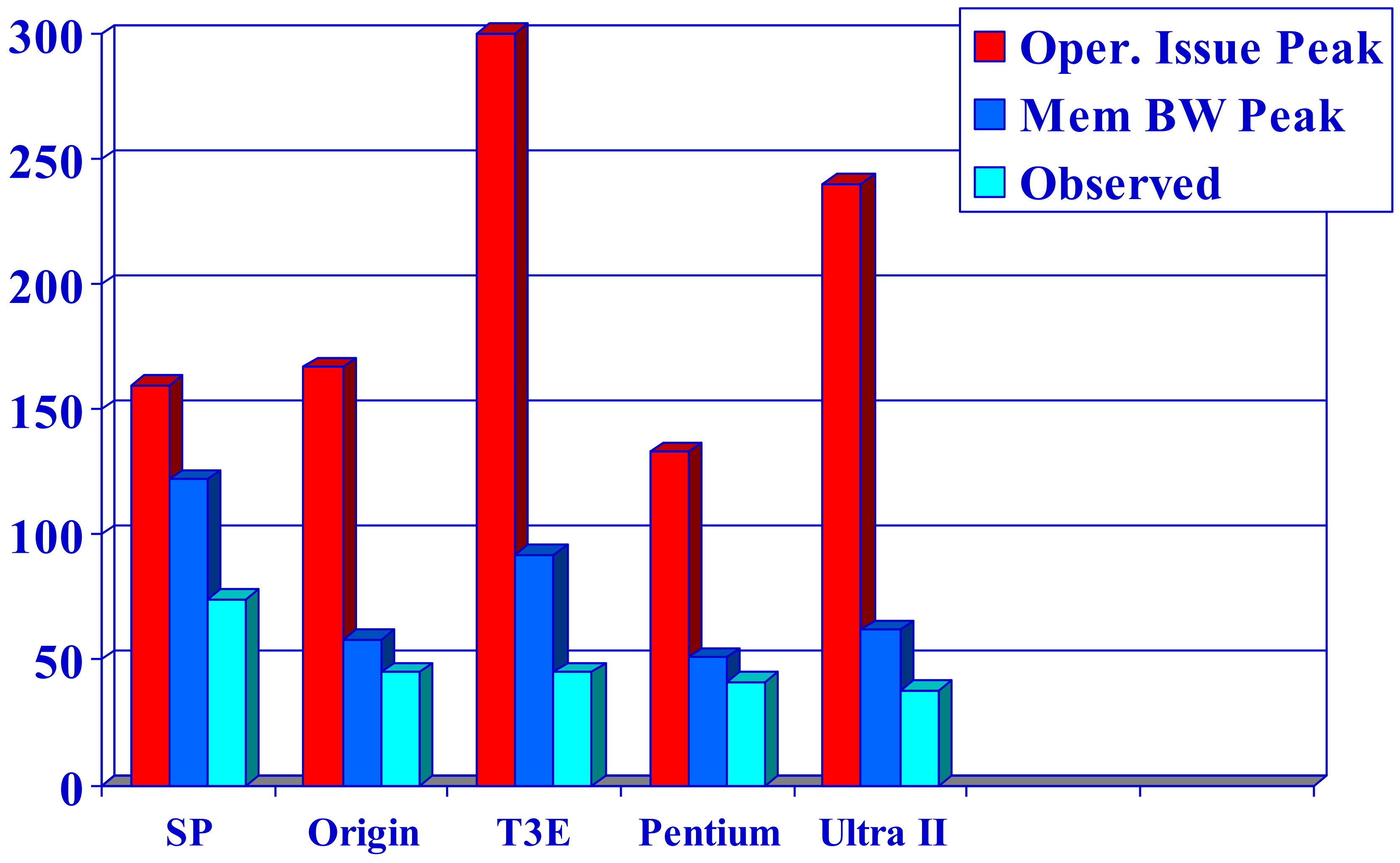
one vector, matrix size, m = 90,708, nonzero entries nz = 5,047,120



Experimental Performance

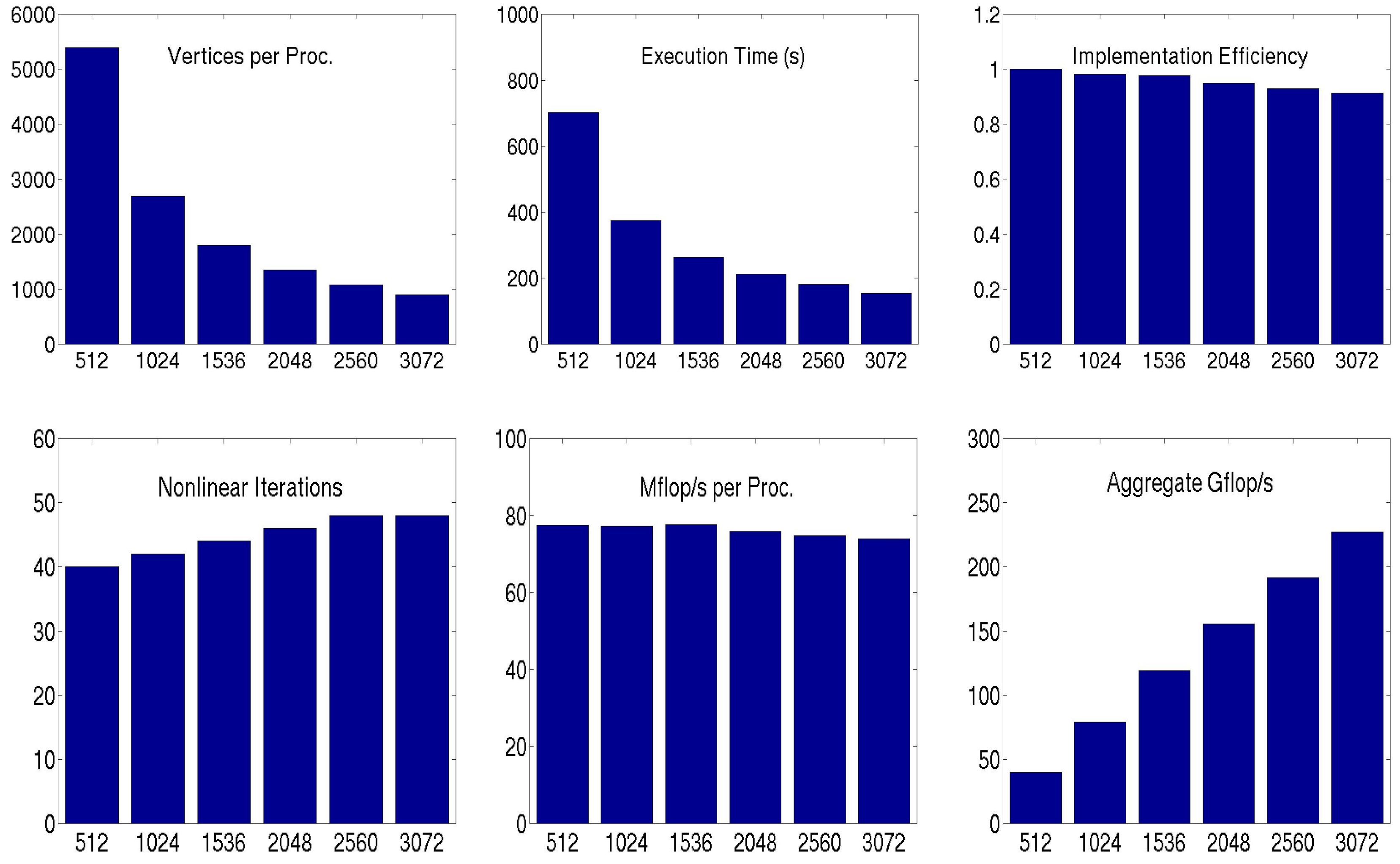
Sparse Matrix Vector Product

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

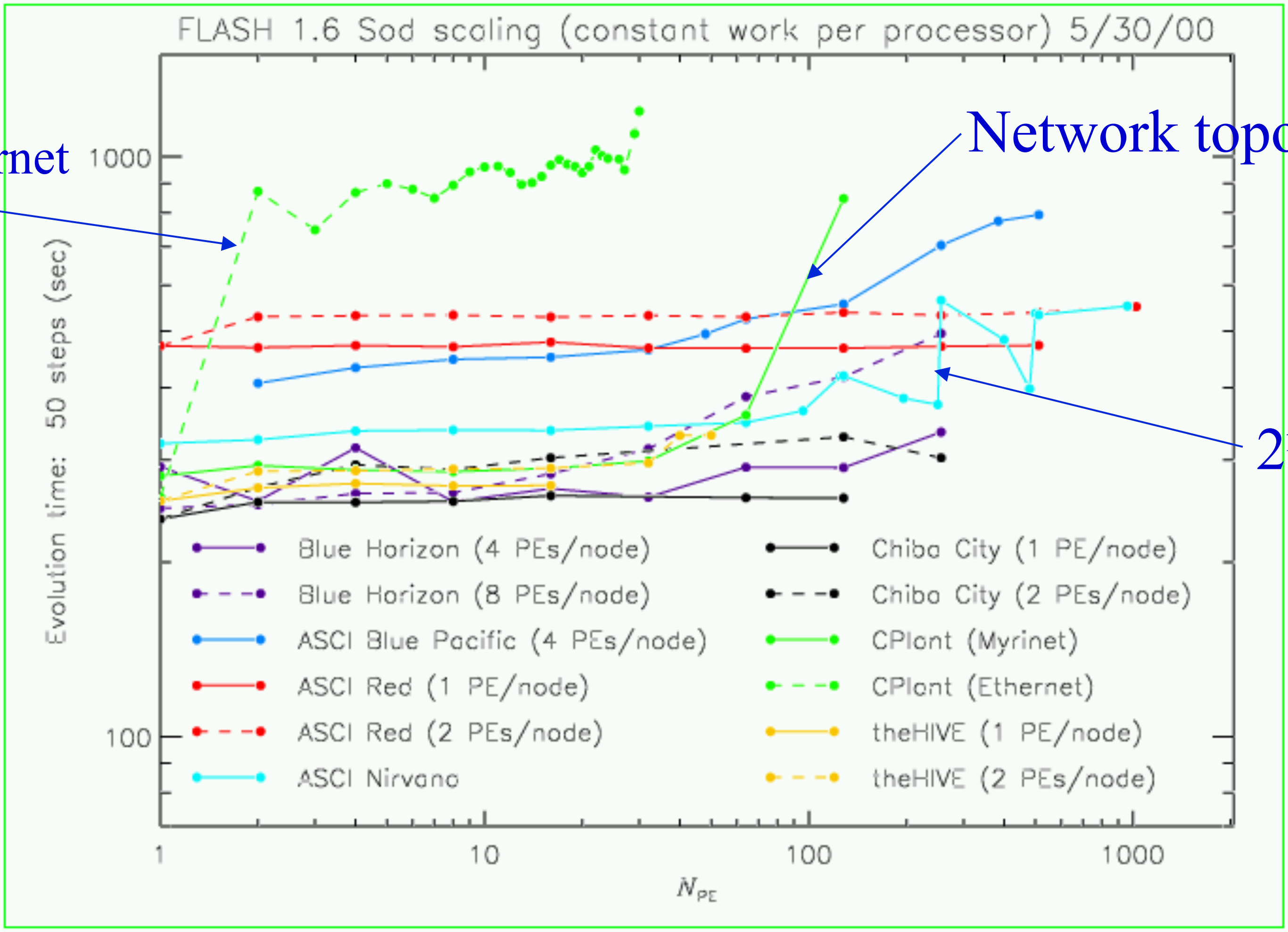


Parallel Scaling Results on ASCI Red

ONERA M6 Wing Test Case, Tetrahedral grid of 2.8 million vertices (about 11 million unknowns) on up to 3072 ASCI Red Nodes (each with dual Pentium Pro 333 MHz processors)



FLASH Scaling Runs



Bad Ethernet switch?

Network topology?

2nd SMP

Myth #2

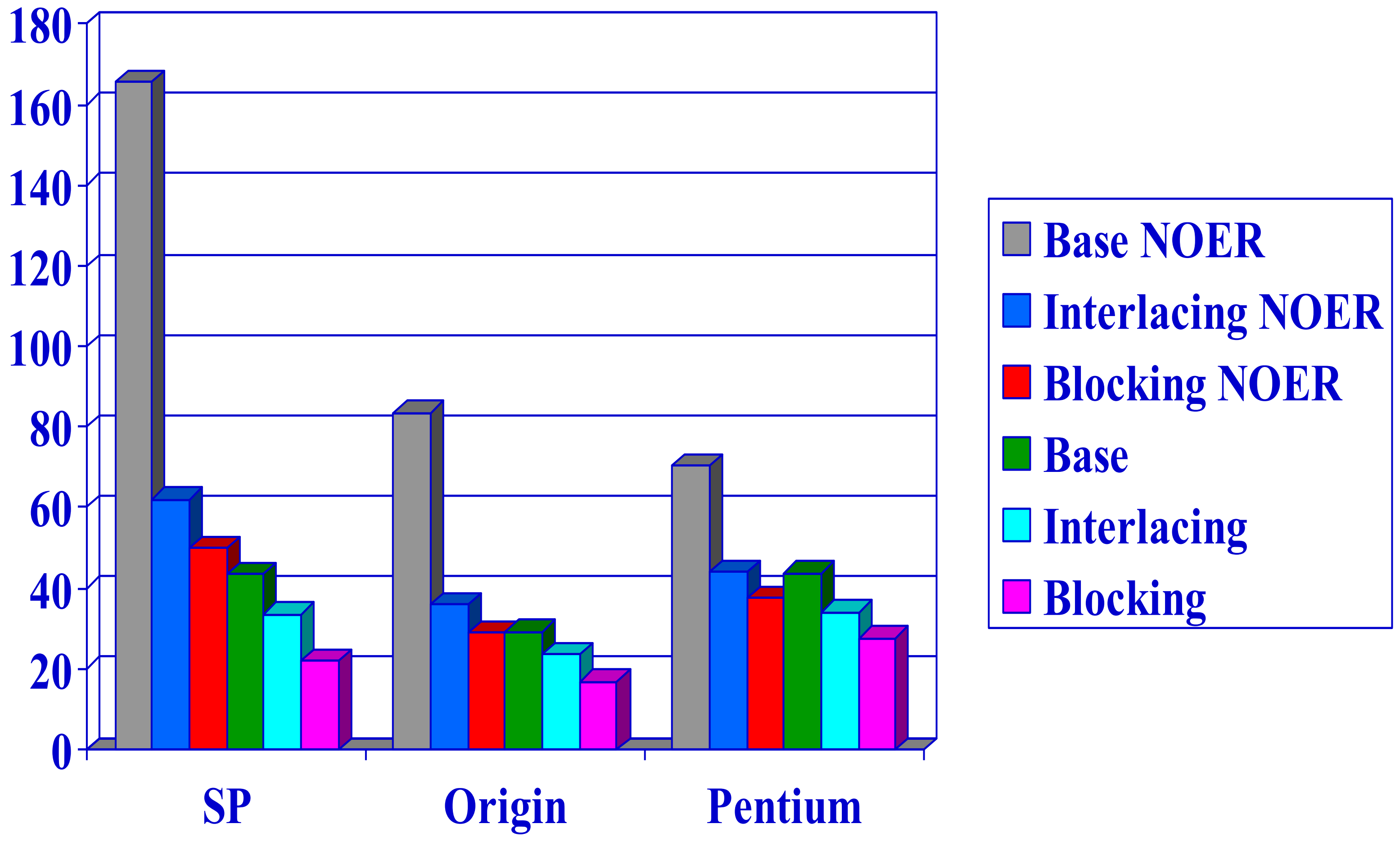
- Parallel computers are hard to program
- Reality: Relative to uni-processors, the difficulty is comparable
  - Even easier

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 ns	Latency cycles	Bandwidth GB/sec
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

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

SGI O2000 (300MHz) 101 cycles to L2

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

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***

Hope for the Future

- New Algorithmic Directions
  - Match directions in memory heirarchies
- New Computing Systems
  - Bring back (affordable) memory bandwidth
    - Then we can complain about latency
  - Exotic solutions
    - Exotic + small benefit = **dead**
- Mixed Memory-Model Systems
  - Augment commodity nodes with smart memory

Conclusions

- Parallel Programming is *easy*
  - compared to uni-processor programming
- Algorithms need to be developed to reflect hardware trends
  - These are not new trends
- Programming models need to support performance-oriented applications
  - Particularly memory locality and access
- More radical (willing to fail) approaches are needed