How Not to Measure Performance: Lessons from Parallel Computing
or
Only Make New Mistakes

William Gropp
www.mcs.anl.gov/~gropp

ARGONNE NATIONAL LABORATORY • UNIVERSITY OF CHICAGO •

Why Measure Performance?

- Publish papers or sell product
- Engineer a solution to performance goals
  - ♦ Predict performance
  - ♦ Tune systems
- Understand limitations of current systems (research into the future)
- Diagnose or predict performance problems
- Compare methods/systems (publish *good* papers)

What Kinds of Measurement?

- Means-based
  - ♦ Measure what you can
- Ends-based
  - ♦ Measure what you need to know
- One way to test:
  - ♦ Does improving the measured value improve the user's experience?
- Examples:
  - ♦ FLOPS are usually means-based
    - Not related to solving a problem
    - Often rewards poor algorithms
  - ♦ Wall clock times are ends-based

Some Characteristics of Good Benchmarks

- Repeatable
  - ♦ Perhaps in a statistical sense
- Easy to use
- Consistent in meaning across systems
- Consistent in result
  - ♦ If the benchmark says “A is better than B” then a user would say the same
  - ♦ Lilja calls this “Reliable”
- Unbiased
  - ♦ Independent of stakeholder interests

What Do We Know?

- Many performance-related books and papers
  - *The Science of Computer Benchmarking*, Roger Hockney, 1996
    - Emphasizes HPC, particularly parallel/vector
  - *Measuring Computer Performance*, David Lilja, 2000
    - Really a book in basis statistics for computer scientists
- Unfortunately, much of the work is of limited use
  - Best captured by a famous short paper by David Bailey

Twelve Ways to Fool the Masses (paraphrased from David Bailey)

1. Quote only 32-bit performance results
2. Present results for an inner kernel and then represent these as the performance of the entire application
3. Quietly employ assembly code
4. Scale problem with the number of processors but omit any mention of this
5. Quote performance results projected to a full system
6. Compare your results against scalar, unoptimized code

Twelve Ways (continued)

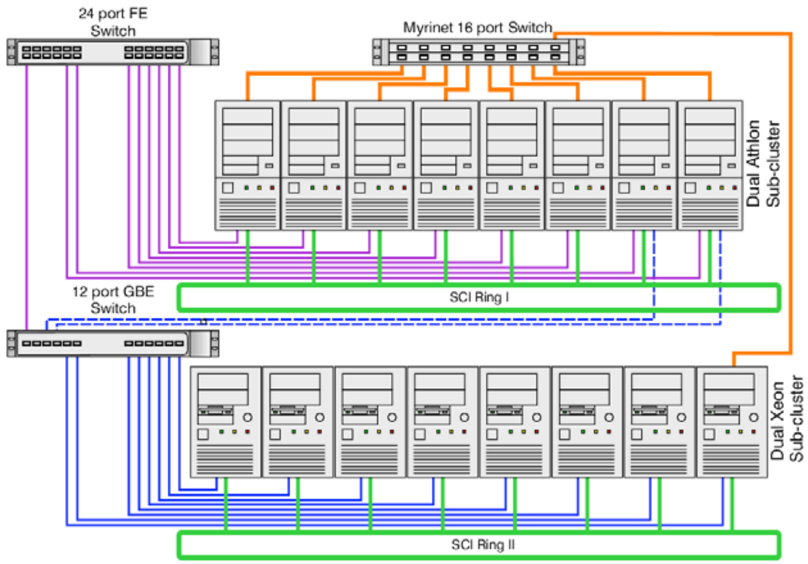
7. When direct run time comparisons are required, compare with an old code on an obsolete system
8. If MFLOPS rates must be quoted, base the operation count on the parallel implementation, not the best sequential implementation
9. Quote performance in terms of processor utilization, parallel speedups, or MFLOPS/$
10. Mutilate the algorithm used in the parallel implementation to match the architecture
11. Measure parallel run times on a dedicated system but measure conventional run times in a busy environment
12. If all else fails, show pretty pictures and animated videos, and don't talk about performance

Why is this relevant?

- There are similarities between parallel and grid computing
  - Multiple processing elements connected by a network
  - Data as well as compute-centric applications
- And there are differences
  - Where to begin?
    - Relevant applications
    - Potentially much higher latencies
    - Security
    - Administration
    - Reliability

What's similar with the Grid

- Many parallel systems are clusters or constellations
- Some grid applications are distributed computing



- Others involve distributed I/O, similar to parallel I/O issues

What's Different About the Grid?

- Goals: Shared resource
  - ♦ No exact reproducibility of experimental conditions
    - Classic MPPs have very good reproducibility
    - Clusters less so
- No central control in operation
- Very complex paths for messaging; multiple transport types
- Very high latency
  - ♦ Latency in the ms (>10^7 operations!)
  - ♦ Leads to asynchronous applications
  - ♦ Performance goals emphasize bulk or realtime performance
- Often a greater software gap between the application and the hardware

What Can We Learn About Benchmarks From the MPP Experience

- What has worked
  - ♦ Kinds of benchmarks
- What has *not* worked
  - ♦ May be more valuable
- What we should have done
  - ♦ How to measure
  - ♦ How to choose

Some Kinds of Benchmarks

- Microbenchmarks
  - Small, easily understood code
  - Runs on (most) systems unchanged
  - Measures a well-defined* quantity
  - But
    - May not address the *reason* for benchmarking
- Application benchmarks
  - Large, awkward, complex code
  - Often uses non-portable constructs
  - Measures an application-defined quantity
    - May be what you want
- Synthetic benchmarks and program kernels
  - Attempt to capture key features of applications benchmarks but without the problems
    - But only valuable to the extent that they are in fact representative

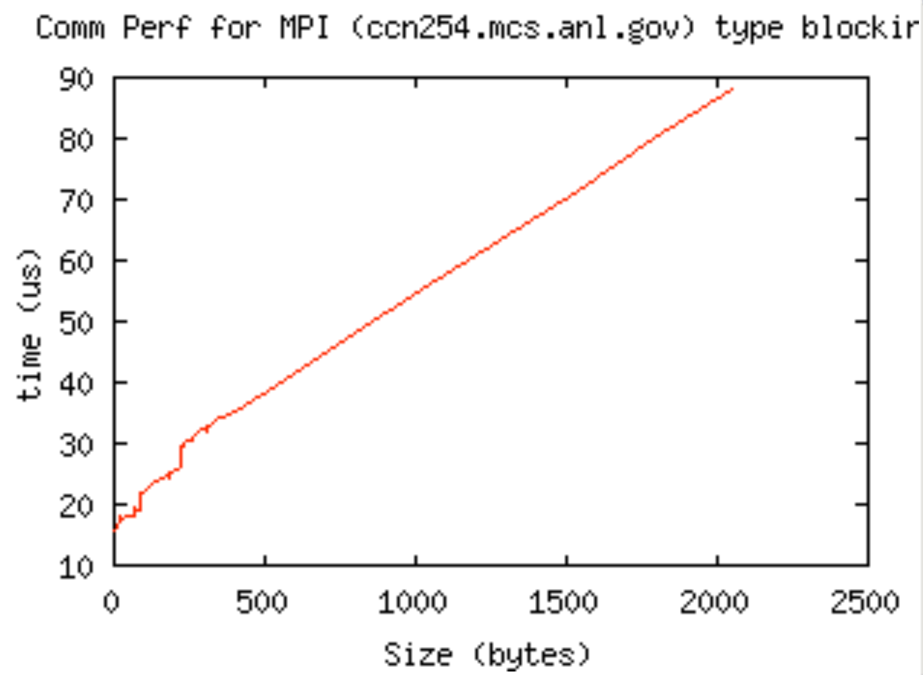
Example from Parallel Computing

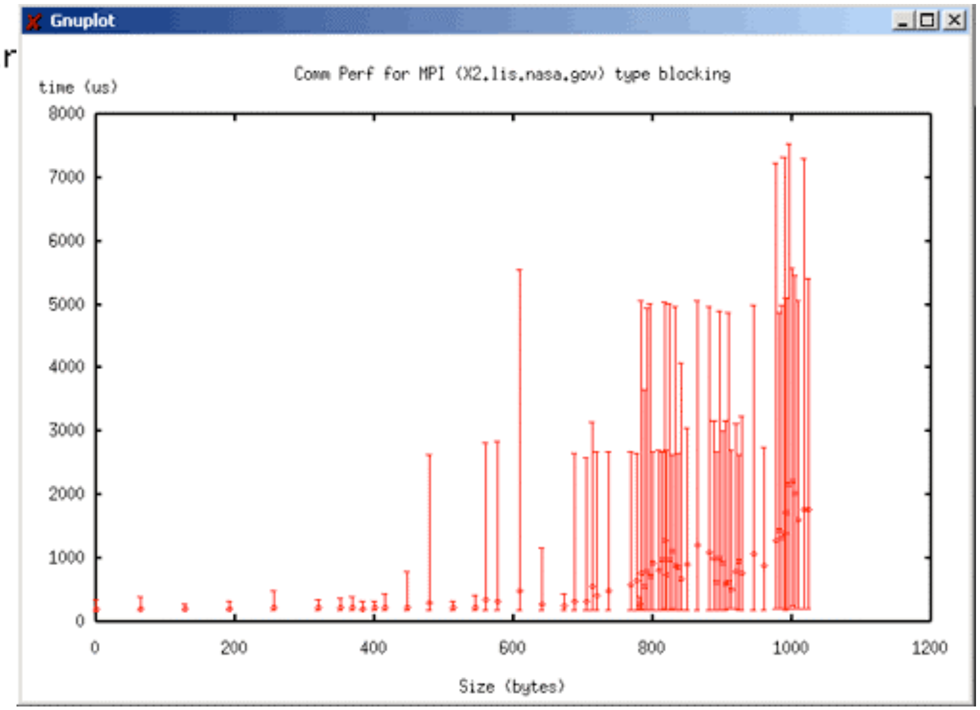
- Basic measurement of communication performance: latency and bandwidth
  - Defined by $T_c = s + rn$
- Problems:
  - Assumes hardware and software match this model
  - Absolute numbers make it difficult to draw conclusions
    - Important quantity is often the *relative* cost, compared to work
    - A better model may be $(s/f) + (r/f)n$, scale by floating point speed
      - Increasing processor speed without increasing network speed increases relative communication cost
- Note that for the grid, latency may be dominated by time-of-flight

Problems With Microbenchmarks

- Simplification may change results by activating special-case code
  - ♦ Is latency the time to send a 0-byte message?
  - ♦ Or is it the coefficient s in the time model $T_c = s + rn$?
    - Which is more useful?
- Reduction may miss important features

Two Examples

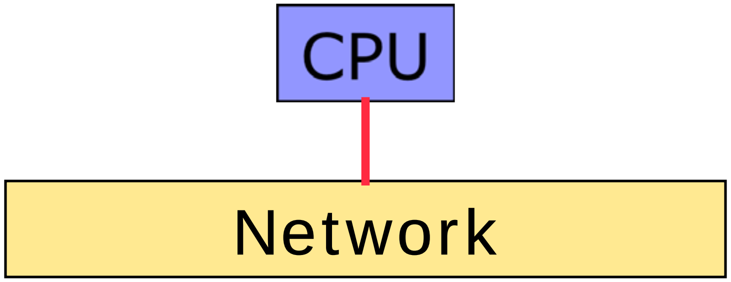




- Sometimes $T_c = s + rn$ is a good fit…

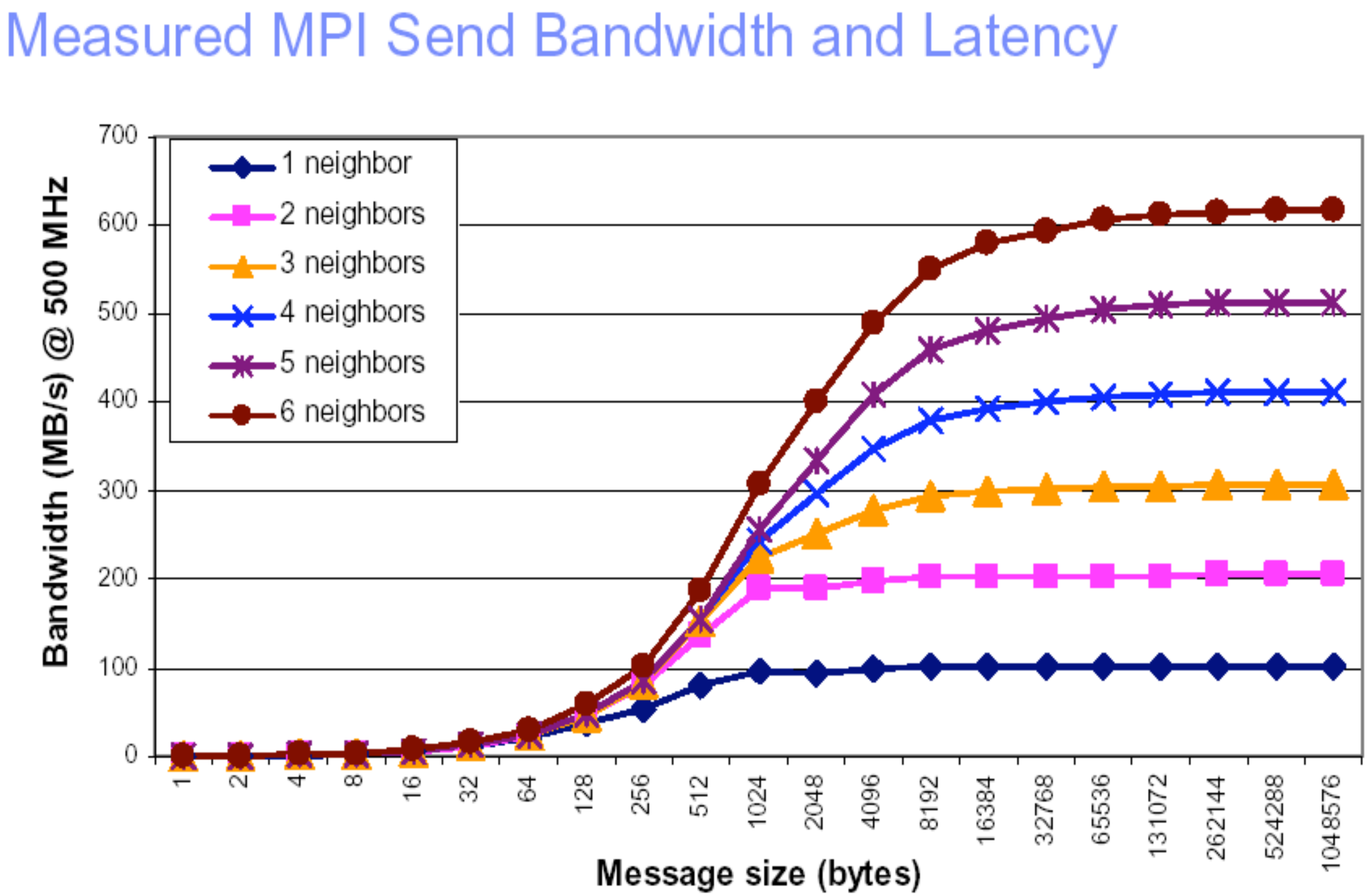
Wrong Communication Model

- The typical model is with a single connection into the network



- Simple and true for many systems, particularly Beowulfs and I/O-peripheral-based networks
- Procurements often based on this model
- But…
    - ♦ Few applications *require* only communicating with a single partner

Measuring the Wrong Quantity



Wrong Benchmark Code

- As benchmarks move away from microbenchmarks, the code becomes more subject to design decisions that may not reflect the environment
  - ♦ NPB avoided use of MPI topology routines even though much evidence that proper layout is important
  - ♦ “Halo exchange” tests may use less efficient code
    - Blocking calls may serialize
    - Misordered nonblocking calls may add overhead
    - Alternatives (e.g., Alltoall) may be semantically equivalent and faster on some platforms
- What are you measuring?

Wrong Design or Use

- Some benchmarks are designed to compare two or more systems. Many are flawed (to be fair, this is a *very* hard problem)
  - SPEC
    - Uses geometric mean to produce a single number that is meaningless (but avoids strong effect from outliers)
  - TPC
    - Artificial set can be gamed
  - Linpack, STREAM
    - Provide precise measurements that are frequently misused

Missing Measurements

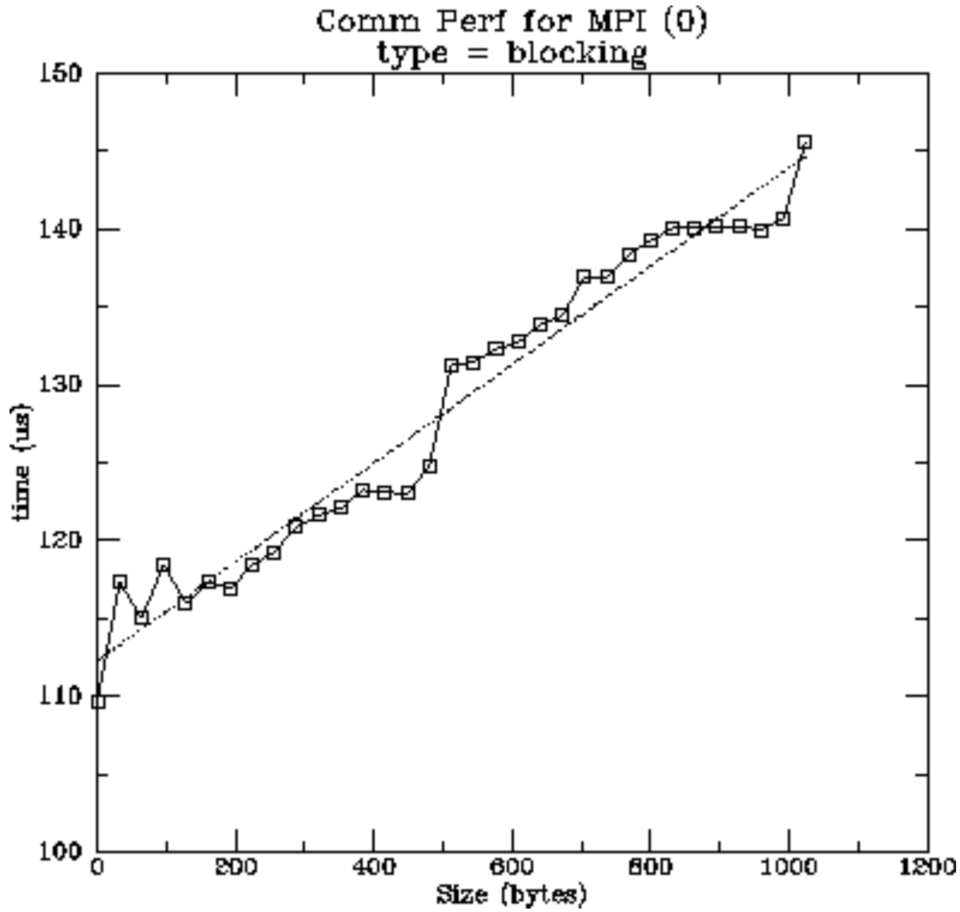
- Failure in grid applications that require multiple co-scheduled resources is common
  - ♦ Increases the cost to applications:
  - ♦ Assume an application requires time T, and that the probability that resource i completes is p_i, and that the application is charged T whether or not the computation completes.
  - ♦ The expected cost is $T(1-p)(1+2p+3p^2+4p^3+...) = T(1+p+p^2+p^3+...) = T(1/(1-p))$ where $p=1-\Pi p_i$; $T_{total} \rightarrow \infty$ as $p \rightarrow 0$

Can We Measure On th Grid?

- Concern: The grid is different because it is constantly changing
  - Thus it is impossible to conduct reproducible experiments
- True (the grid is changing) but irrelevant
  - Scientists often measure in the field
    - Statistical methods used to design and interpret the experiment
  - The user sees the "real" grid. Measurements in an artificial environment have limited usefulness
- MPP benchmarks are not as clean as you might think…

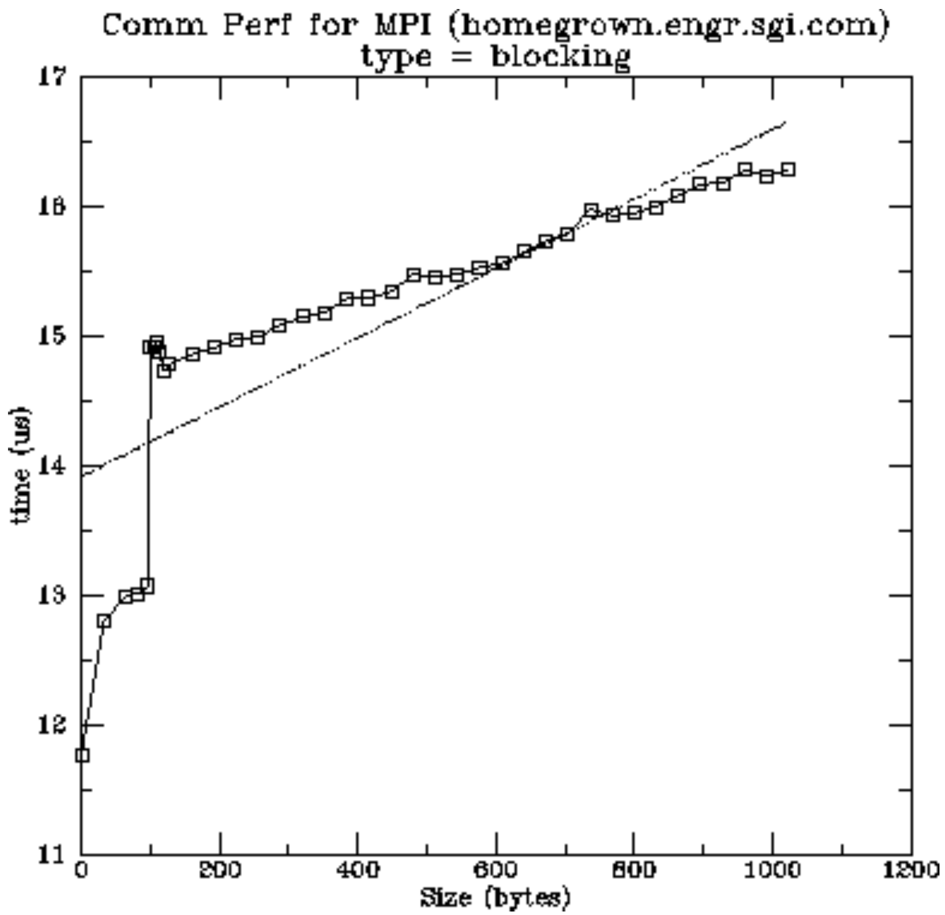
Latency and Bandwidth

- This example shows a reasonable match to the standard formal T_c=s + rn
- Still a few anomalies (512 and 1024 bytes) but not too serious.
- Suggests that we report only s and r



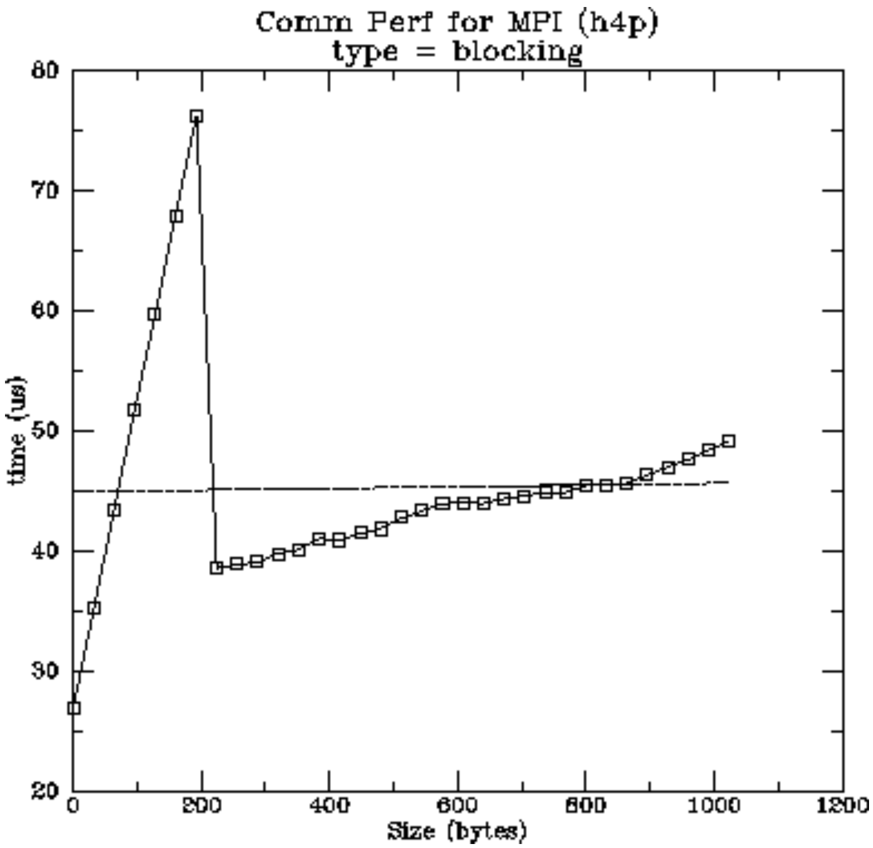
Is Latency the 0-byte Time?

- What is the right definition of latency?
  - To be used in engineering applications, it is the parameter s in T = s + rn, not the time to send a zero byte message
  - How many applications depend on rapidly sending zero-byte messages?
- (Note the poor fit, even though from 128 bytes, the performance is nearly linear)



Perils of Fitting the Wrong Model

- Latency number is not far off
- Bandwidth is essentially infinite
- Two algorithm method
  - Clearly the switch should happen earlier, at least for this test
  - The benchmark implicitly assumes a single algorithm implementation

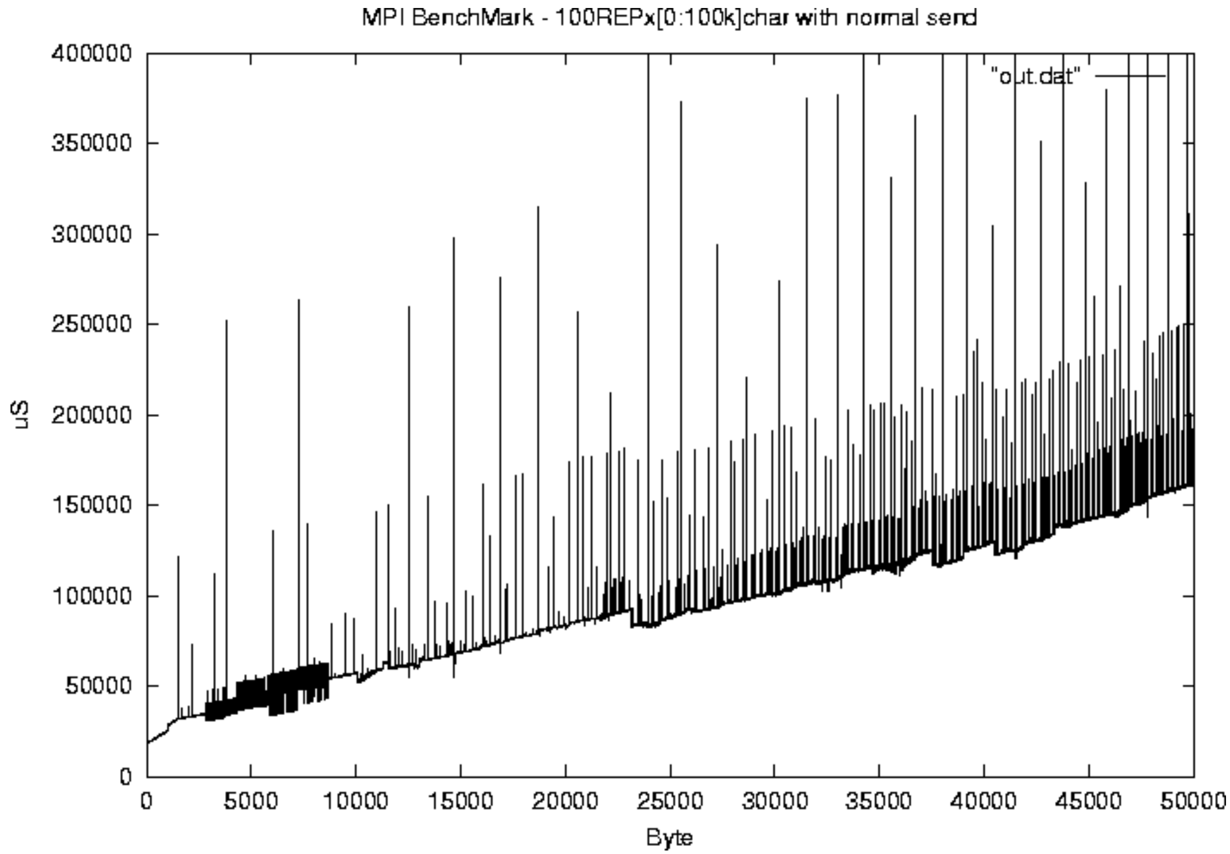


Comments on Benchmark Results

- Single number benchmarks are very convenient — easy to discuss and relatively easy to analyze
- But:
  - ♦ Single number benchmarks rely heavily on a well-defined number that fits the calculation and environment
  - ♦ If you don't know for sure, you need to confirm the model
    - Almost always good to collect more data

Even Simple Parallel Computing Benchmarks can Give Complex Data

- Typical cluster benchmark
  - Note minimum time is (reasonably) well behaved
  - Wide variations
- For purposes of tuning *best* case performance, the minimum times are useful
- But the user sees some average (depending on application)



Grid Challenges

- End-to-End Benchmarks
  - Strong effect from “last meter” (poor I/O to disk; saturated memory system; misconfigured interior network)
  - Must isolate effects
    - Lets corrections can be made
    - Proper experimental design can help
    - Must include interaction effects (multifactor experiments)
- Lack of Reproducibility
  - Benchmarks on the grid are experiments in the field
    - Impossible to control all factors
    - Experiments must have a valid *statistical* design
    - No “instant gratification” benchmarks
- Lack of Established Applications
  - What should we measure?

What Can We Conclude About Grid Benchmarks

- Determine critical application classes and how performance impacts their success.
  - Derive benchmark needs from these
  - Measurements and predictions must include uncertainty
- Grid simulations are needed for reproducible, controlled experiments
  - Understand effects and provides a way to evaluate new methods
  - Counterpart to lab experiments
- "Live" Grid performance measurements based on good experimental design
  - Must use good statistical design
  - CS curriculum needs a course in statistics; Lilja's book is a good place to start
- Include measures of tool usability
  - Anyone remember Veronica? Gopher?

12 Ways to Fool the Masses

1. Quote UDP with now contention control against TCP
   - Only a few people are using the grid at any time
2. Quote performance figures for basic operations, not the entire application
   - Its hard to run a full grid application
3. Quietly employ experts to build and run the grid application
   - It is difficult to work around the many system problems, so use experts. Don't alarm the audience, however
4. Adjust the problem size to achieve the best performance
   - Performance may vary with size
5. Quote performance results projected to a full size grid
   - Few if any labs can afford a grid testbed, and it is very difficult to arrange coordinated access to grid resources

12 Ways (continued)

6. Compare against unoptimized code on a fast machines connected with the fastest network
   - It's impressive to show that your code is faster than results on a bigger, faster grid
7. For direct comparison, compare against an old code on an obsolete system
   - A parameter study on a grid can easily run 100 x faster than a Beowulf cluster (if that is the original, 486-based cluster)
8. When counting operation rates (e.g., data rates), count all data moved anywhere instead of using the most efficient algorithm
   - There's nothing like a few extra GB of data to improve grid utilization
9. Quote performance in terms of utilization, speedup, or ops/{$£¥€}
   - These sound better and they're easy to achieve by adding extra work
10. Mutilate the application and algorithm to fit the grid architecture
    - Avoid latency issues by recomputing. Make massive copies of data.

12 Ways (continued)

11. Measure times on a dedicated grid testbed with security turned off and compare against measurements in a busy, shared environment with security
12. If all else fails, show pretty pictures and animated videos, and don’t talk about performance

4 More Ways

1. Ignore all cases that failed
   - Users just want to know what succeeded
2. Quote results for an API, even if different implementations of the API have different semantics (and hence perform different operations)
   - Common for I/O operations elsewhere, why should we be any different?
3. Confound intra- and inter-grid experiments
   - Who needs those inter (or intra) grid systems?
4. Quote results with no estimate of experimental error
   - No one else does, why should we?

Measurement is Essential for Progress