Challenges and Issues in Benchmarking MPI

a.k.a. Benchmarks Lie

Keith D. Underwood
(Ron Brightwell)

EuroPVM/MPI 2006
Introduction

• Most important applications are a pain
  – Want to wait 3 weeks for an application to build?
  – How long can we wait for the application to run?
  – Have a classified test cluster handy?

• Benchmarks are frequently too simplistic
  – Nominal objective: isolation of network performance
  – How many applications have NO computation?
  – How many applications send messages in the order they are expected?
  – How many benchmarks include load imbalance?
Benchmarks Lie

• Not a new revelation
• Vendors work hard to keep it this way
  – They don’t really want you to know how similar their products are
  – They have to optimize to “the same test everyone else does”
• That does not mean it is easy to do benchmarking well
  – Hardware optimizations interact with software optimizations to yield interesting effects
  – Each network has different interactions to make a “fair” comparison hard
Overview of Issues

- Optimized host processor
  - Big cache (holds all of benchmark)
  - Prefetcher (works unnaturally well on benchmark)
- Unfriendly compilers
  - Tend to create terrible code without optimization
  - Tend to eliminate “dead” code with optimization – all of a benchmark is “dead” code 😊
- “Good” software optimizations
  - The “right thing to do” in the software stack can interact with the hardware to make good benchmarking hard
- No interaction of “computation” with “communication”
  - Load imbalance?
  - RogueOS effect?
Quadrics Elan4

• Connected by PCI-X bus
• Offload of MPI matching semantics to NIC processor
  – Provides low host overhead
  – Provides true “independent progress”
• Primary issue: “good” software optimization
  – Use of PCI-X bus is lousy for “on node” communications
  – The “right thing to do” is to segregate “local” posted receive list from “remote” posted receive list
Software Optimization Impact

Graph showing the impact of software optimization on effective latency (microseconds) versus queue items traversed. The graph compares Baseline Benchmark and Adjusted Benchmark. The baseline shows a linear increase, while the adjusted benchmark shows a steep increase as queue items traversed increase.
Infinipath

- HyperTransport connected custom IB NIC
- Extreme message rates
- All work done on the host processor
- Primary issue: big, fat, optimized processor
  - Processor has aggressive benchmark optimizations (caches, prefetching)
  - Such resources should be polluted by application (means benchmark behavior is poorly matched to application behavior)
Flushing Cache – Latency
Flush Cache – Message Rate

Effective Bandwidth (MB/s)

Message Size (Bytes)

Elan4, Flushed
Elan4, Baseline
Myrinet, Flushed
Myrinet, Baseline
InfiniPath, Flushed
InfiniPath, Baseline
InfiniBand, Flushed
InfiniBand, Baseline
Flushed the Cache Not Enough

• Modern latency tolerance techniques go beyond caching
  – Techniques now include aggressive prefetching
  – Prefetching should make very little difference in MPI processing
• Unfortunately, good software optimizations interact with prefetching to yield a large benefit
  – MPI data structures are allocated from a slab cache
  – Most benchmarks cause these to be allocated and then freed sequentially
  – Result: a linked list that is stride-1 in memory
Randomizing Slab Cache

![Graph showing latency vs. queue items traversed for different cache configurations.](image)
A Case Study: HPCC RandomAccess

- HPCC RandomAccess benchmark was limited by message rate
  - Implemented a new version that uses “hypercube” style algorithm
  - Optimized using aggressive overlap and leveraging independent progress

- Expected bottlenecks
  - Memory bandwidth (many random accesses)
  - Network latency (messages of 8KB)
  - Network bandwidth (comm. time = latency + size/bw)

- Additional bottleneck: time to post a receive ?!??
  - Never appears in “microbenchmarks” as critical factor
Posting Receives takes HOW LONG?
You must have done SOMETHING wrong…
(Eliminate Contention)
Try MPICH-1.2.6
Comparison of Impacts

- Eliminate Contention
- Baseline
- MPICH-1.2.6
Conclusions

• Benchmarks are hard to do “right”
  – Good software optimizations have too much impact on the typical benchmark case
  – Software optimizations can interact with hardware optimizations to have unrealistically positive impact in the benchmark case

• Typical benchmarks make it impossible to predict performance
  – The goal of a good benchmark should be enabling performance prediction
  – Real application impacts make things like posting a receive take much longer than the benchmark case