The Triumph of Hope over Experience*?

Bill Gropp

ARGONNE NATIONAL LABORATORY • UNIVERSITY OF CHICAGO •

*Samuel Johnson

Where Was Parallel I/O?

- ***No*** parallel I/O results (independent I/O by processes in the same parallel “job” doesn’t count) were presented
- No application wants to generate a zillion files, proportional to the number of processes or nodes
  - ♦ They do it because they’ve given up getting what they want
- “Only one parallel file system works and I can’t buy it”
  - ♦ This is bad?
- Avoid I/O?
  - ♦ But you need to output something (why else compute?)

Realistic (Application-centric) I/O Benchmarks

- POSIX I/O essentially specifies sequential consistency
  - Understandable choice for the masses
  - Disastrous choice for parallel performance
- Solution is not “no consistency” (e.g., NFS); it is a precise, relaxed consistency model
  - Such definitions exist:
    - MPI-I/O provides an API
    - Parallel file systems (incl PVFS) provide implementations that work today
- We have a parallel I/O benchmark suite
  - It is in pretty rough shape
  - But any working parallel file system (with MPI-IO) should be able to run it ☺

Predictions

- SOS9 – *Parallel* (not independent) I/O results
- (out on a limb) POSIX I/O will stop being a requirement for high-performance file systems
- Parallel I/O will be required of any programming model/environment
  - ♦ Just kidding!

Programming Models

- You can always create a special-purpose language for your application
  - It will (given enough effort) be superior to all other languages for your application
  - It may be a *disaster* for others
  - It might be a disaster for *you* when the next machine comes out

Why Was MPI Successful?

- It address *all* of the following issues:
  - Portability
  - Performance
  - Simplicity and Symmetry
  - Modularity
  - Composability
  - Completeness
- Most current proposals for new languages address only a subset
  - Two features often missing: modularity (e.g., library support) and completeness (e.g., I/O)

Predictions

- We will be complaining about MPI
  - That’s ok. People are still complaining about Fortran
  - We’ll be complaining because most of the important applications are using MPI
- We will be complaining about memory latency and bandwidth
  - The ratio of memory latency to CPU cycle will continue to increase
  - The ratio of MPI latency to memory latency will be smaller, both to now and to the main/CPU ratio
- Applications will begin to use MPI Put/Get

MPI Put/Get

- Part of MPI-2 (MPI ≡ MPI-1 + MPI-2)
- Can be implemented efficiently on a wide range of platforms (e.g., does not require cache coherence wrt network DMA)
- Designed to fit into MPI model (with full generality)
- Q: I’ve heard that the rules are too complex
  - A: There are simple rules that are sufficient for most programmers.
- Q: I’ve heard that the RMA is slower than point to point
  - A: Sadly, few implementations have gotten this right. MPICH2 demonstrates how to do it; shame will do the rest
- Q: But what about xxx?
  - A: Some xxx are a misunderstanding of the standard. And some xxx are in fact limitations of the standard. The right fix though is to improve the standard, not start over.