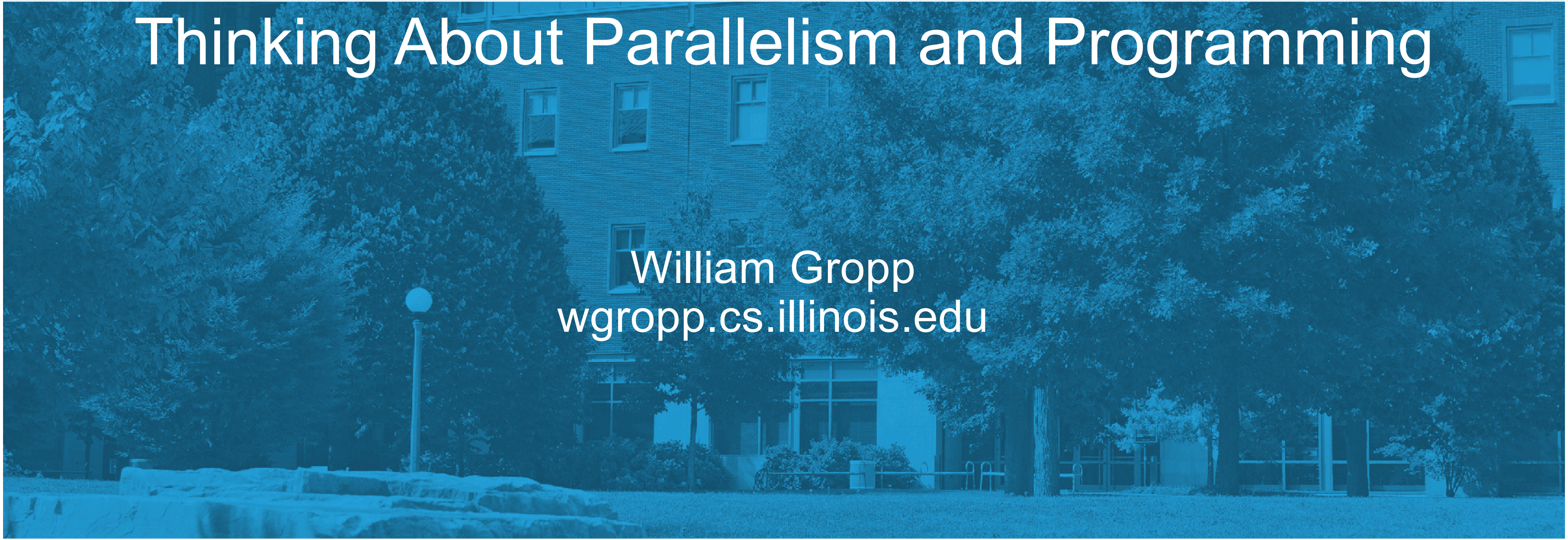


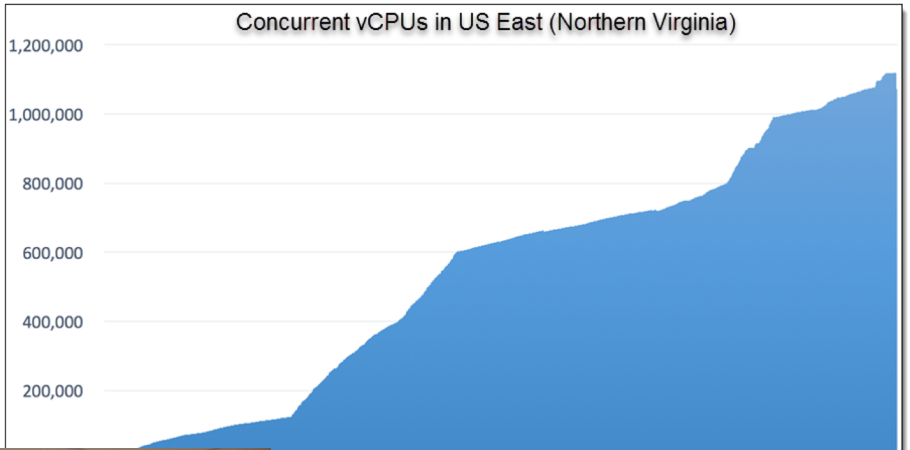
# Thinking About Parallelism and Programming

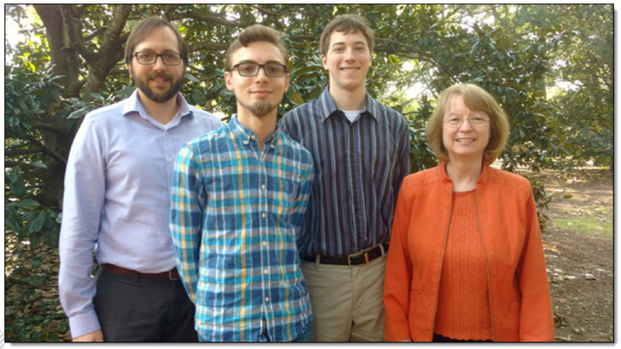
William Gropp
wgropp.cs.illinois.edu

NCSA

# Parallelism is Everywhere

- And at all scales
- Students can begin working with parallel systems early
  - My son Chris ran on Jaguar at ORNL, then the Top500 #1, after his freshman year
  - More recently, ran on nearly 1.2M cores (more than I ever have!)
  - And he also found scalability bugs…





# Training students to think precisely and quantitatively about parallel computing

- Parallel programming is *hard*
- We need to be scientific about solving these problems
- We would all like parallel programming to be eaiser and more fun, but to accomplish that, we need to focus on the real problems
- And we must set a good example for our students.
  - What follows are some examples of fuzzy thinking that we, as a community, must strive to improve
  - We can do this by insisting that our students be rigorous and to follow the scientific method
  - We must remind each other to separate opinion from fact …

# Quotes from "Enabling Technologies for Petaflops Computing" (MIT Press 1995)

- **"The software for the current generation of 100 GF machines is not adequate to be scaled to a TF…"**
- "The Petaflops computer is achievable at reasonable cost with technology available in about 20 years [2014]."
  - (estimated clock speed in 2004 — 700MHz
- "Software technology for MPP's must ev that is portable across a wide variety of can the small but important MPP sector leverage the massive investment that is being applied to commercial software for the business and commodity computer market."
- "To address the inadequate state of software productivity, there is a need to develop language systems able to integrate software components that use different paradigms and language dialects."

Except that software was adequate not only for TeraFLOP but for PetaFLOP and almost certainly ExaFLOP

# Quotes from “Enabling Technologies for Petaflops Computing” (MIT Press 1995)

- “The software for the current generation of 100 GF machines is not adequate to be scaled to a TF…”
- **“The Petaflops computer is achievable at reasonable cost with technology available in about 20 years [2014].”**
  - **(estimated clock speed in 2004 — 700MHz)**
- “Software technology for MPP’s e
  that is portable across a wide va en
  can the small but important MPP t
  leverage the massive investment ware
  for the business and commodity
- “To address the inadequate state l to
  develop language systems able t se
  different paradigms and language dialects.”

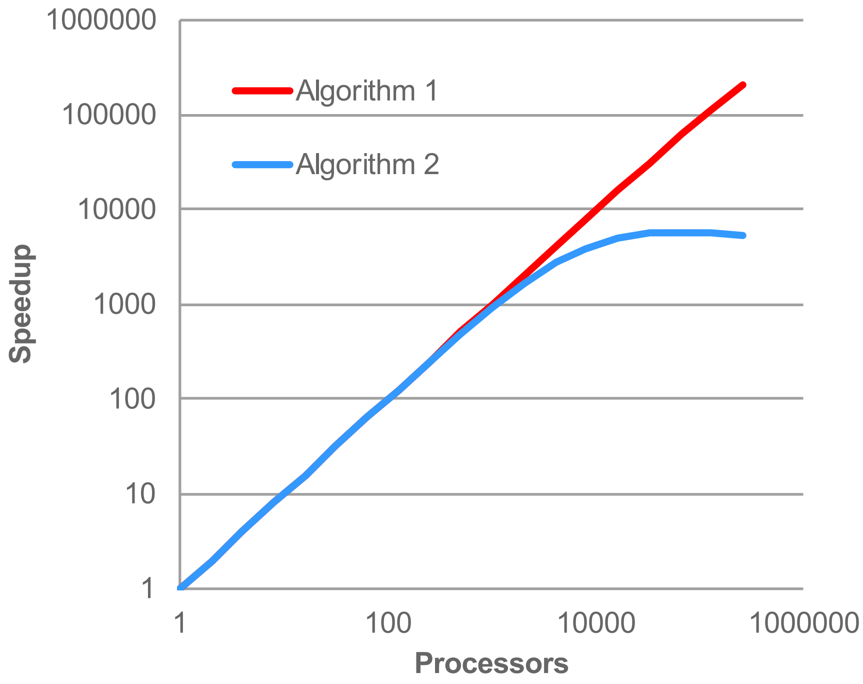
This clock speed estimate was dead on target – IBM BG/L ran at this speed, and this prediction was based on quantitative projections

# What is Scalability?

- Is it
  - Using all cores on a chip efficiently
  - Using all nodes in a distributed memory system efficiently
  - Using many virtual CPUs in ”the cloud” efficiently (and what is “many”)
  - Ability to handle larger data
  - Ability to handle more users/requests
  - Running faster than a single core/chip/node?
- At least strong and weak scaling (when discussed together) are usually interpreted as #1 or #2
- Preciseness in language is important!

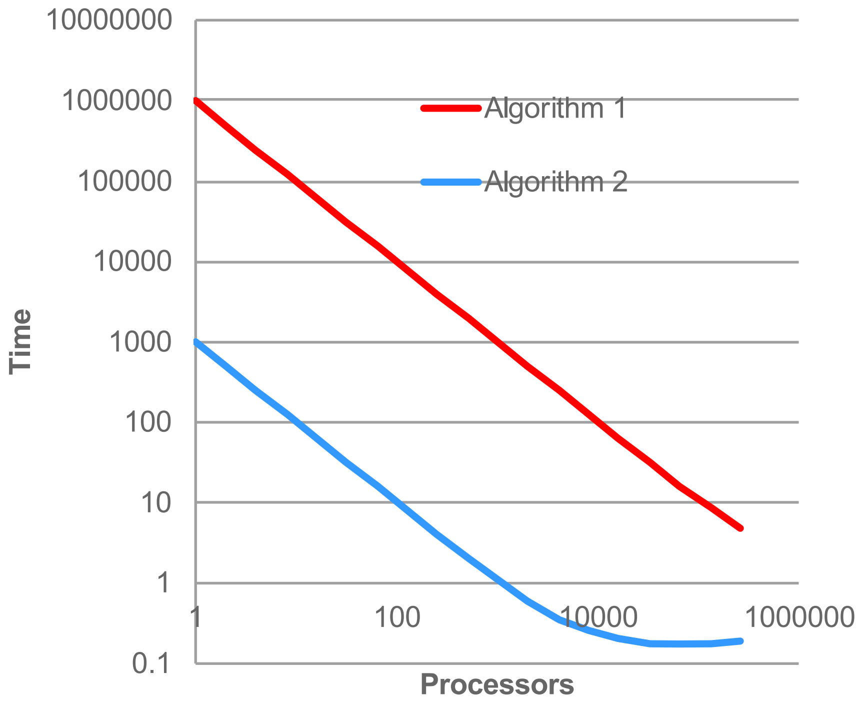
# Scalability – What does it tell you?

- Should I use algorithm 1 or algorithm 2?



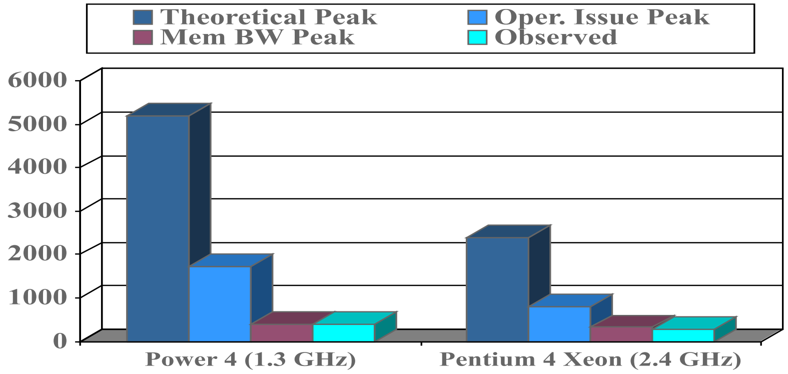
# Scalability – What does it tell you?

- Should I use algorithm 1 or algorithm 2?
- Here's the same data, only showing time instead of speedup
- Its time to solution that is important
- You can always improve scalability by:
  - Decreasing per-core performance
  - Using a more computationally intensive algorithm



# How should you evaluate speedup?

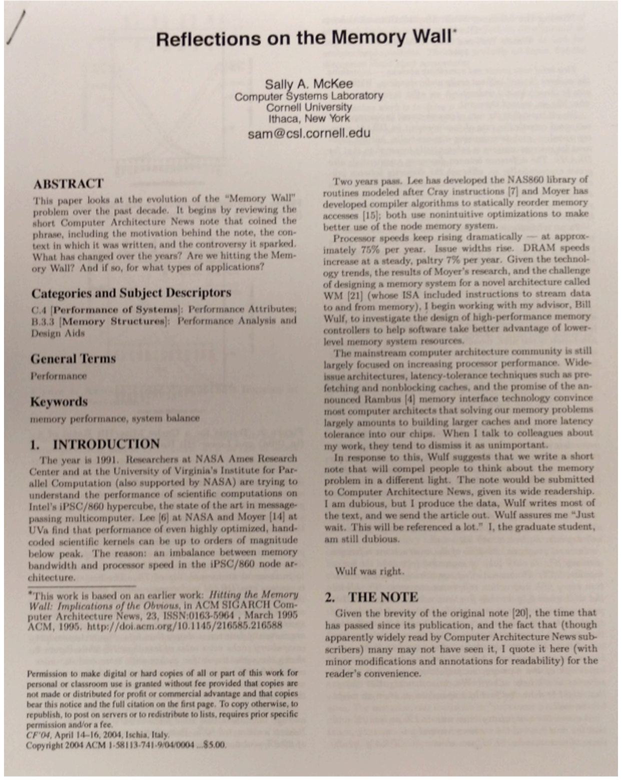
- When measured, you need to
  - Understand the algorithm and options
    - Some are not obvious – accurate n-body force calculations can be done in O(n), not O($n^2$) time; Broadcast of n words takes O(n), not O(n log p)
  - Evaluate implementation quality
    - Can use simple quantitative performance model
    - For many apps, STREAM is appropriate



W. K. Anderson, William D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith. *Achieving high sustained performance in an unstructured mesh CFD application*, SC99 (Gordon Bell Prize)

# What Parameters Are Important?

- Raw rates (clock speed, FLOPS, memory bus bandwidth) are rarely useful by themselves
    - I once saw a paper that claimed memory bandwidth wasn't a useful indicator of performance
        - What they were looking at was the memory bus bandwidth – which was *far* higher than the sustained memory performance seen by applications or STREAM
        - Memory bus bandwidth *was* useless, but *sustained bandwidth* as measured by STREAM was useful
- Complex interactions of latency, bandwidth, instruction hazards, etc. make simple comparisons suspect
    - Students often latch onto the hype around raw performance numbers.



## Reflections on the Memory Wall*

Sally A. McKee
Computer Systems Laboratory
Cornell University
Ithaca, New York
sam@csl.cornell.edu

### ABSTRACT

This paper looks at the evolution of the "Memory Wall" problem over the past decade. It begins by reviewing the short Computer Architecture News note that coined the phrase, including the motivation behind the note, the context in which it was written, and the controversy it sparked. What has changed over the years? Are we hitting the Memory Wall? And if so, for what types of applications?

### Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Performance Attributes; B.3.3 [**Memory Structures**]: Performance Analysis and Design Aids

### General Terms

Performance

### Keywords

memory performance, system balance

### 1. INTRODUCTION

The year is 1991. Researchers at NASA Ames Research Center and at the University of Virginia's Institute for Parallel Computation (also supported by NASA) are trying to understand the performance of scientific computations on Intel's iPSC/860 hypercube, the state of the art in message-passing multicomputer. Lee [6] at NASA and Moyer [14] at UVa find that performance of even highly optimized, hand-coded scientific kernels can be up to orders of magnitude below peak. The reason: an imbalance between memory bandwidth and processor speed in the iPSC/860 node architecture.

Two years pass. Lee has developed the NAS860 library of routines modeled after Cray instructions [7] and Moyer has developed compiler algorithms to statically reorder memory accesses [15]; both use nonintuitive optimizations to make better use of the node memory system.

Processor speeds keep rising dramatically — at approximately 75% per year. Issue widths rise. DRAM speeds increase at a steady, paltry 7% per year. Given the technology trends, the results of Moyer's research, and the challenge of designing a memory system for a novel architecture called WM [21] (whose ISA included instructions to stream data to and from memory), I begin working with my advisor, Bill Wulf, to investigate the design of high-performance memory controllers to help software take better advantage of lower-level memory system resources.

The mainstream computer architecture community is still largely focused on increasing processor performance. Wide-issue architectures, latency-tolerance techniques such as prefetching and nonblocking caches, and the promise of the announced Rambus [4] memory interface technology convince most computer architects that solving our memory problems largely amounts to building larger caches and more latency tolerance into our chips. When I talk to colleagues about my work, they tend to dismiss it as unimportant.

In response to this, Wulf suggests that we write a short note that will compel people to think about the memory problem in a different light. The note would be submitted to Computer Architecture News, given its wide readership. I am dubious, but I produce the data, Wulf writes most of the text, and we send the article out. Wulf assures me "Just wait. This will be referenced a lot." I, the graduate student, am still dubious.

Wulf was right.

### 2. THE NOTE

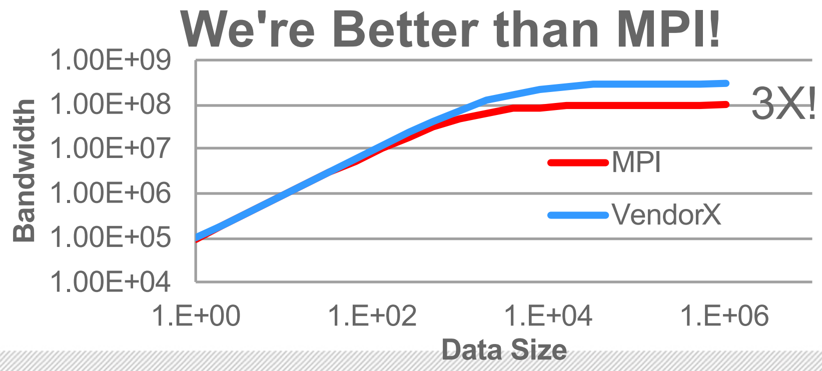
Given the brevity of the original note [20], the time that has passed since its publication, and the fact that (though apparently widely read by Computer Architecture News subscribers) many may not have seen it, I quote it here (with minor modifications and annotations for readability) for the reader's convenience.

*This work is based on an earlier work: *Hitting the Memory Wall: Implications of the Obvious*, in ACM SIGARCH Computer Architecture News, 23, ISSN:0163-5964 , March 1995 ACM, 1995. http://doi.acm.org/10.1145/216585.216588

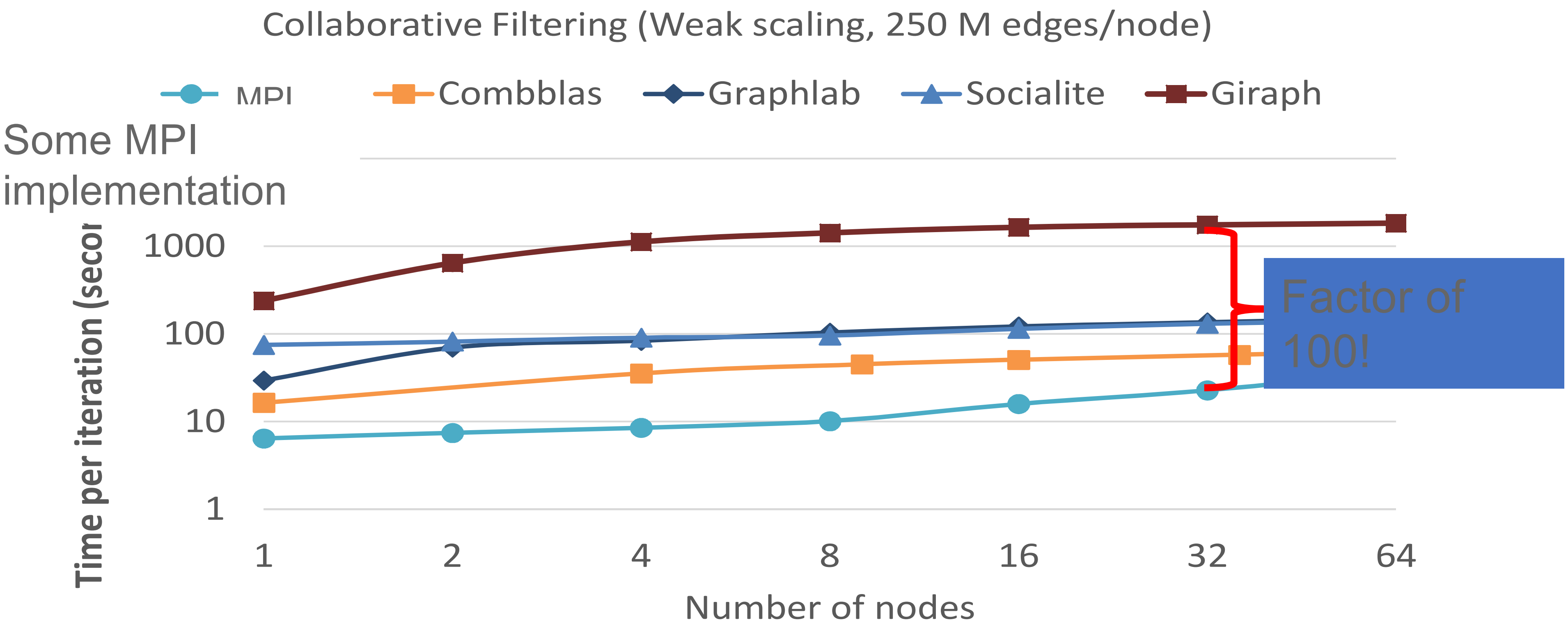
*CF'04*, April 14–16, 2004, Ischia, Italy.

# How do you compare two programming models?

- First – are you comparing programming *models*, programming *systems*, or *implementations* of programming systems?
  - Answer – Almost always implementations
  - Implication – No paper should be accepted that claims to compare X to Y when all it does is compare an implementation of X on Z to an implementation of Y on Z
- Second – what conclusions can you draw from the comparison?
  - Example (drawn from a real vendor document)
  - Intended message: use our non-standard method – its faster!
  - Actual message: We have no clue how to implement MPI correctly
  - Good Test Question: Why do I make this claim?



# A comparison of implementations



Some MPI implementation

Factor of 100!

**Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets**
Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Jiwon Seo, Jongsoo Park, M. Amber Hassaan, Shubho Sengupta, Zhaoming Yin, and Pradeep Dubey

# Reproducibility and significant digits

- A keystone of science is the reproducible experiment
  - Part of this is measuring and reporting only what is reproducible
- There are many challenges in making experiments reproducible
  - Complete description of the apparatus (e.g., system, compiler version/options, source code, …)
- But some are (relatively easy)
  - Don’t report overly precise measurements
    - Or, don’t use output from %e in your paper; even %.2e is often better
  - Describe your test environment
  - Make the code and data available

# What’s in a name?

- In science, rigor and precision are important
- Example:
  - “Programming exascale systems requires moving beyond the BSP programming model; we need to replace MPI with <some new thing>.”
- What is wrong with this statement?
  - “…exascale systems requires moving beyond BSP”
  - This is a hypothesis.  How would you test or demonstrate it?
  - But there is a more subtle error…

# MPI is not a BSP system

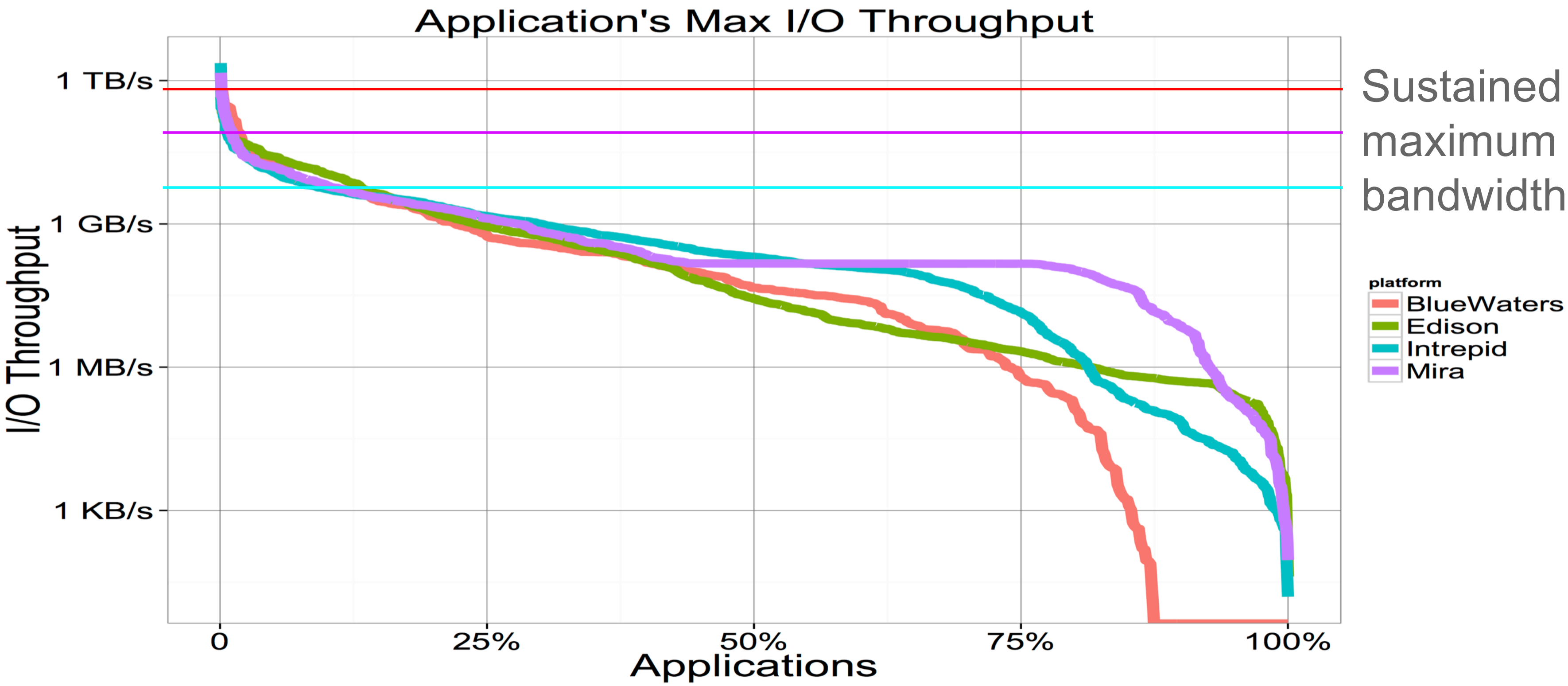
- BSP = Bulk Synchronous Parallel
  - Programmers **like** the BSP model, adopting it even when not necessary (see FIB)
  - Unlike most programming models, *designed* with a performance model to encourage *quantitative* design in programs
- MPI makes it easy to emulate a BSP system
  - Rich set of collectives, barriers, blocking operations
- MPI (even MPI-1) sufficient for dynamic adaptive programming
  - The main issues are performance and “progress”
  - Improving implementations and better HW support for integrated CPU/NIC coordination more effective at supporting need
  - (This is not to say that there aren’t issues – see Marc Snir’s talk: “MPI is too High Level/MPI is too Low Level” at http://www.mcs.anl.gov/mpi-symposium/slides/marc_snir_25yrsmpi.pdf )

Leslie G. Valiant, A bridging model for parallel computation, Communications of the ACM, Volume 33 Issue 8, Aug. 1990

# Understand what you ask for

- Current I/O performance is often appallingly poor
  - Even relative to what current systems can achieve
  - Part of the problem is the I/O interface semantics
- Many applications need to rethink their approach to I/O
  - Not sufficient to “fix” current I/O implementations
- HPC Centers have been complicit in causing this problem
  - By asking users the wrong question
  - By using their response as an excuse to keep doing the same thing
  - What is that question that causes so much trouble?
    - Do you want/need POSIX I/O?

# Just how bad Is current I/O performance?

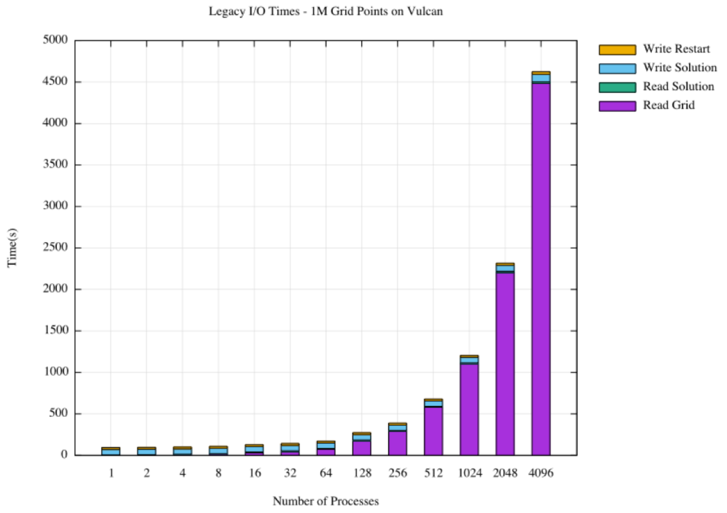


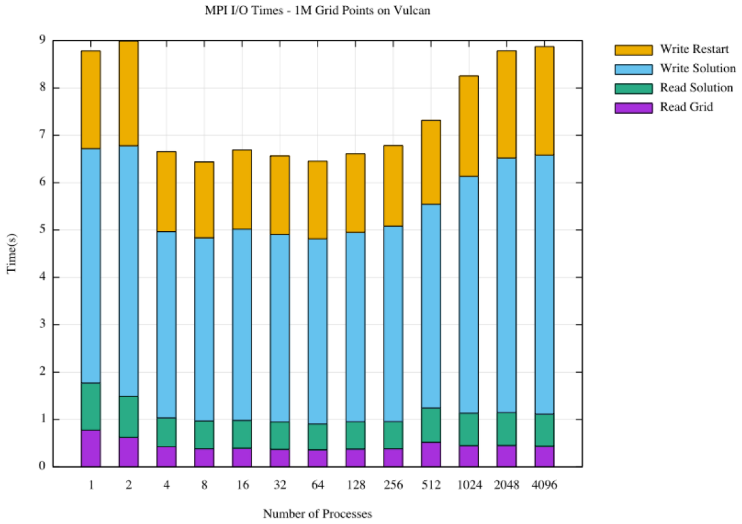
Sustained maximum I/O bandwidth

"*A Multiplatform Study of I/O Behavior on Petascale Supercomputers*," Huong Luu, Marianne Winslett, William Gropp, Robert Ross, Philip Carns, Kevin Harms, Prabhat, Suren Byna, and Yushu Yao, proceedings of HPDC'15.

# One Example in Parallel I/O

- Original code used a simple I/O strategy for reading input grid file
- Typical science run was many hours (6-24). I/O less than 10-20% of time
- But debugging runs are a few seconds – after input loaded
- Understanding the achievable performance, and using the appropriate I/O semantics (not POSIX), gives more than ***1000X*** speedup (MeshIO library of Ed Karrels)





# What are some of the problems?

- POSIX I/O has a strong consistency model
  - Extremely hard to cache effectively
  - Applications need to transfer block-aligned and sized data to achieve performance
  - Complexity adds to fragility of file system, the major cause of failures on large scale HPC systems
- Files as I/O objects add metadata “choke points”
  - Serialize operations, even with “independent” files
- Burst buffers will not fix <u>these</u> problems – must change the semantics of the operations

# Other communities have matched their data systems to their needs

- “Big Data” file systems have very different consistency models and metadata structures, designed for their application needs
  - Why doesn’t HPC?
    - There have been some efforts, such as PVFS, but the requirement for POSIX has held up progress

# Why is the question “wrong”?

- To almost all application developers, asking if they need POSIX I/O means (to them)
  - Do you need open/seek/read/write/close?
- And if they answer “no”, the implication is
  - They will need to rewrite their application and reformat their files
- With this interpretation, **no-one** would ever answer “no” to “Do you need POSIX I/O?”
  - Questions to which only one answer is reasonable don’t tell you anything

# No science application code needs POSIX I/O semantics

- Many are single reader or single writer
  - Eventual consistency is fine
- Some are disjoint reader or writer
  - Eventual consistency is fine, but must handle non-block-aligned writes
- Some applications use the file system as a simple data base
  - Use a data base – we know how to make these fast and reliable
- Some applications use the file system to implement interprocess mutex
  - Use a mutex service – even MPI point-to-point
- A few use the file system as a bulletin board
  - Most likely better off using RDMA
  - Only need release or eventual consistency
- *Correct* Fortran codes do not require POSIX in any form
  - Standard requires unique open, enabling correct and aggressive client and/or server-side caching
- MPI-IO would be better off without POSIX

# Where's the hope?

- I firmly believe that a rigorous, scientific approach is the only way to solve the great challenges facing us in making better, more productive use of parallel computing
- And look where it has gotten us already
  - Computing power in a single system has increased one ***billion*** fold over my career. Data capacity and networking bandwidths have seen similar increases
  - We have been able to program these systems
    - Often the real challenge is in single core or single node performance
- We can accelerate this progress by (training our students in) challenging conventional wisdom and being rigorous in applying the scientific method

# Thanks!

- For funding from
  - National Science Foundation
  - Department of Energy
  - ExxonMobil and JumpLabs
  - State of Illinois
- My co-workers at Yale, Argonne, and Illinois
- My colleagues around the world
- My students
- And especially Ken Kennedy, for his contributions and for the example he set as a scholar and gentleman