



**Institute for Advanced Computing**  
Applications and Technologies

---

## **How Would We Define High Value in a Software Innovation Institute?**

William Gropp

[www.cs.illinois.edu/~wgropp](http://www.cs.illinois.edu/~wgropp)



The Institute combines research in a host of disciplines at the University of Illinois with the advanced technology capabilities of the National Center for Supercomputing Applications.

# High Value Software

- Software in this context is used to perform and analyze experiments, both computational and other
  - Value is measured by how it enables these tasks
  - Measuring this is hard; like measuring value of research.
  - Total cost of the experiment and analysis includes multiple elements:
    - Cost of people (writing/planning/using)
    - Cost of hardware on which software runs (capital and operating)
    - Cost of developing/adapting/tuning/maintaining software
    - Cost of data acquisition and management
  - There are also constraints
    - Immediate: correctness, completeness
    - Longer term: portable, extensible,
- Added value to what would happen without a software institute
- Life is making choices



# Correct assignment of problem

- Are new ideas/tools needed or better implementations of existing tools?
  - Must avoid building a new tool to work around a limitation in some implementation
    - Example: Parallel I/O libraries that work around limitations of current implementations
      - Instead of illuminating upon and requiring better implementations of existing standards
      - Easier to get credit by creating a new project than improve an existing one

If you thought trying to get a reward for software is hard, try getting a reward for *improving* software



# Who is the direct beneficiary of this institute?

- The rare code experts needed to build key libraries and tools?
- Code developers who are *not* experts?
- The practicing scientists and engineers that need to use computation in their work?
- Students learning to be scientists and engineers?  
Learning to be code experts?
- Most likely: a combination of these
  - But beware mission creep



# On what sort of systems will this code run?

- Leadership systems (e.g., Blue Waters, Sequoia, K, Titan)
- Divisional/Center systems (e.g., iForge)
- Deskside/laptop
- Other (phone, embedded)
- Probably all of the above, but may require different solution and emphasis
  - Is scalability single chip/node (shared memory/ threads) or 10k-100k nodes? (distributed memory; fully distributed control)



# Two Timescales in CS Work

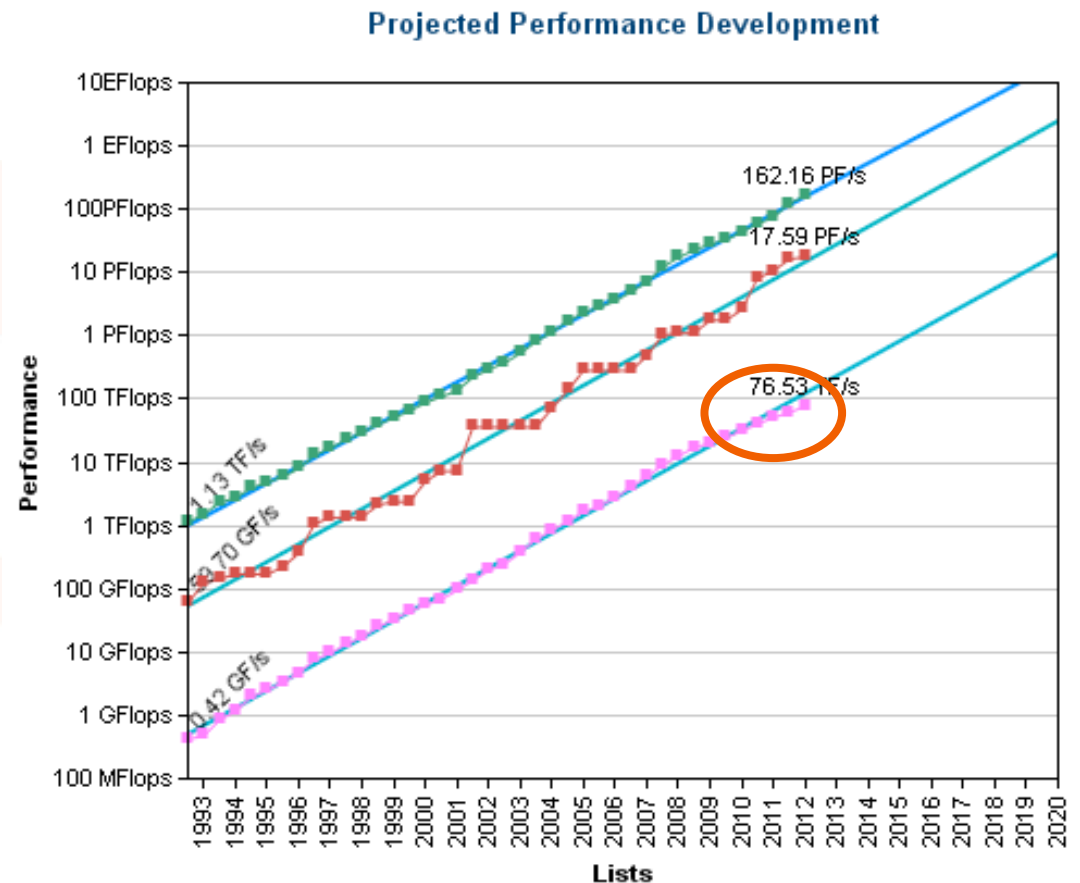
- Computer science has a least two timescales:
  - Very fast: esp. the increase in hardware capability
  - Slow or step changes: everything else
- Sustaining progress requires recognizing the difference between these





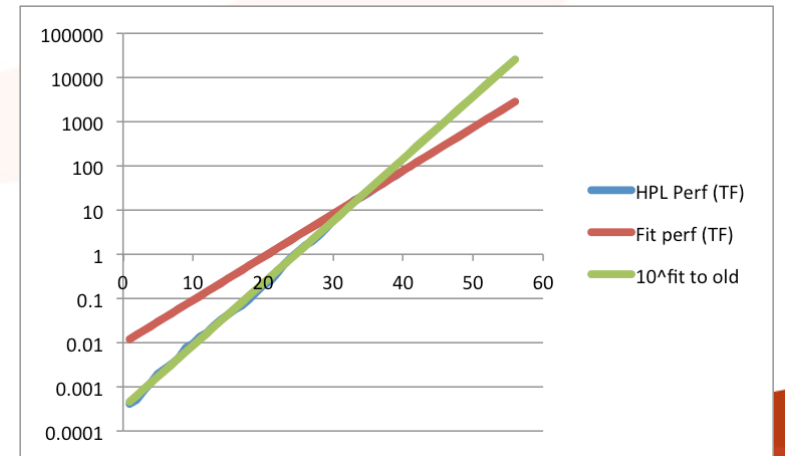
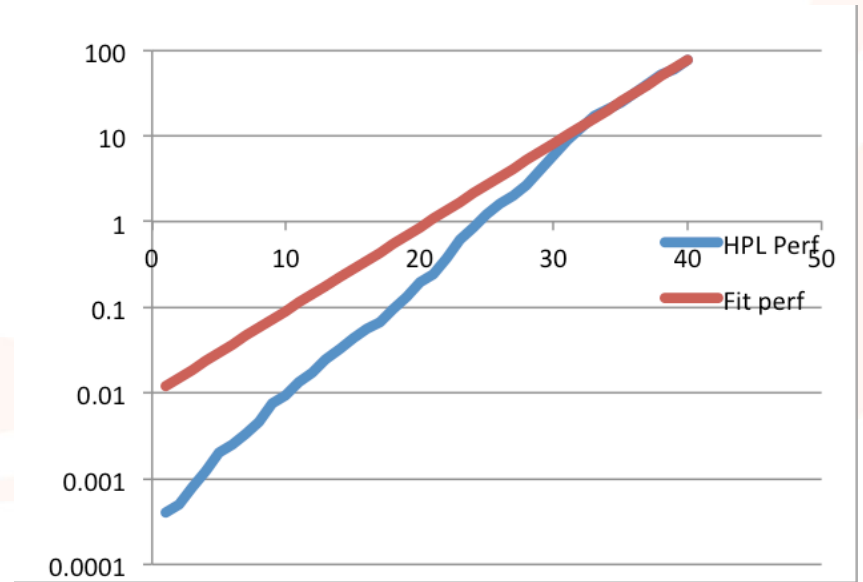
# Amazing Increase in Computing Power

- Exponential increase in performance for several *decades*
- Five orders of magnitude since the Top500 was started only 20 years ago
- But not everything has changed that fast...



# That “kink” in #500 is Real

- Extrapolation of recent data gives ~1PF HPL in 2018 on the #500 system
- Extrapolation of older data gives ~1PF in 2015, ~7PF in 2018
- The #500 *may* be a better predictor of trends





# Everything Else Changes Slowly

- Programming, libraries
- Standards, software, languages
  - Ken Kennedy said it takes at least 10 years for a new programming language to “take”
  - MPI and MPICH illustrate both (see later)
- Somewhere in the middle
  - Are Applications here? What do you think?
- “Punctuated Equilibrium” may be a better model
  - Combined with slow change
  - Can argue that accelerators are another step change in hardware (look at the *top* of the top500)
- To predict the future it is useful to look at the past...




## Quotes from “System Software and Tools for High Performance Computing Environments” (1993)

- “The strongest desire expressed by these users was simply to satisfy the urgent need to get applications codes running on parallel machines as quickly as possible”
- In a list of enabling technologies for mathematical software, “Parallel prefix for arbitrary user-defined associative operations should be supported. Conflicts between system and library (e.g., in message types) should be automatically avoided.”
  - Note that MPI-1 provided both
- Immediate Goals for Computing Environments:
  - Parallel computer support environment
  - Standards for same
  - Standard for parallel I/O
  - Standard for message passing on distributed memory machines
- **“The single greatest hindrance** to significant penetration of MPP technology in scientific computing is **the absence of common programming interfaces** across various parallel computing systems”



# Quotes from “Enabling Technologies for Petaflops Computing” (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 evolve new ways to design software that is portable across a wide variety of computer architectures. Only then can the small but important MPP sector of the computer hardware market leverage the massive investment that is being applied to commercial software for the business and commodity computer market.”  Trickle up
- “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.”
- (9 overlapping programming models, including shared memory, message passing, data parallel, distributed shared memory, functional programming, O-O programming, and evolution of existing languages)



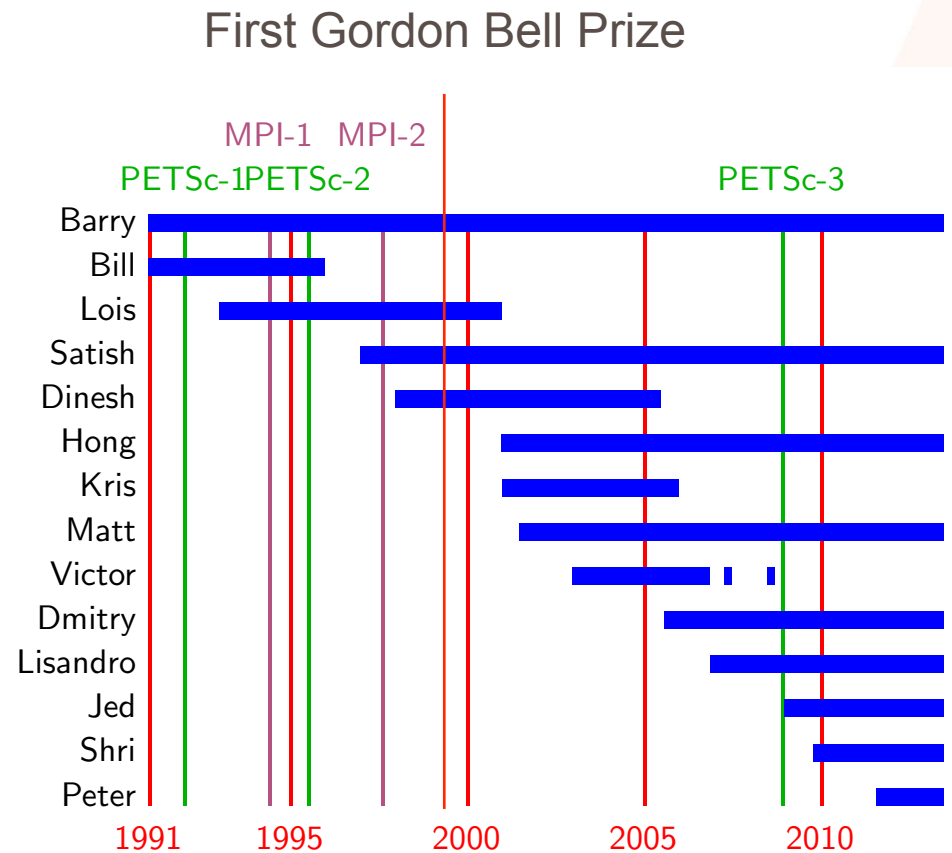
# Why This Matters

- Performance gains from hardware are slowing
  - Some features, such as frequency scaling, ended years ago
  - We need to change intuition about hardware performance and impact on algorithms and software
- Expectations of rapid change diverts attention from the need to sustain development in software and algorithms
  - Change is a step – but the step only succeeds if it is nurtured



# PETSc Timeline

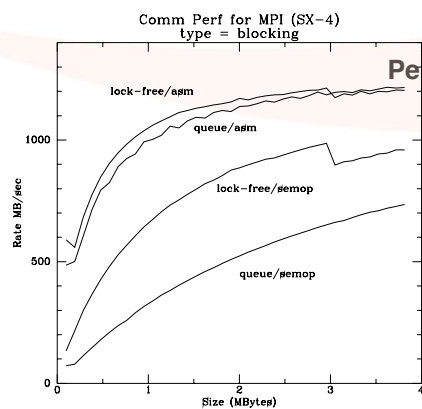
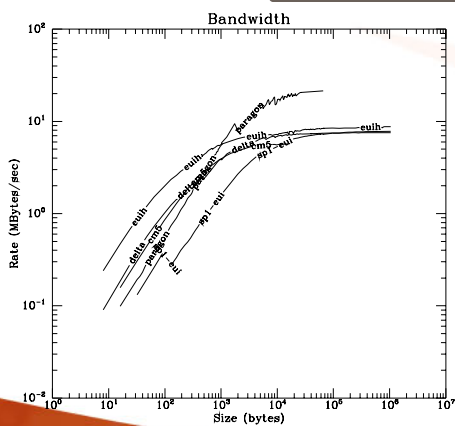
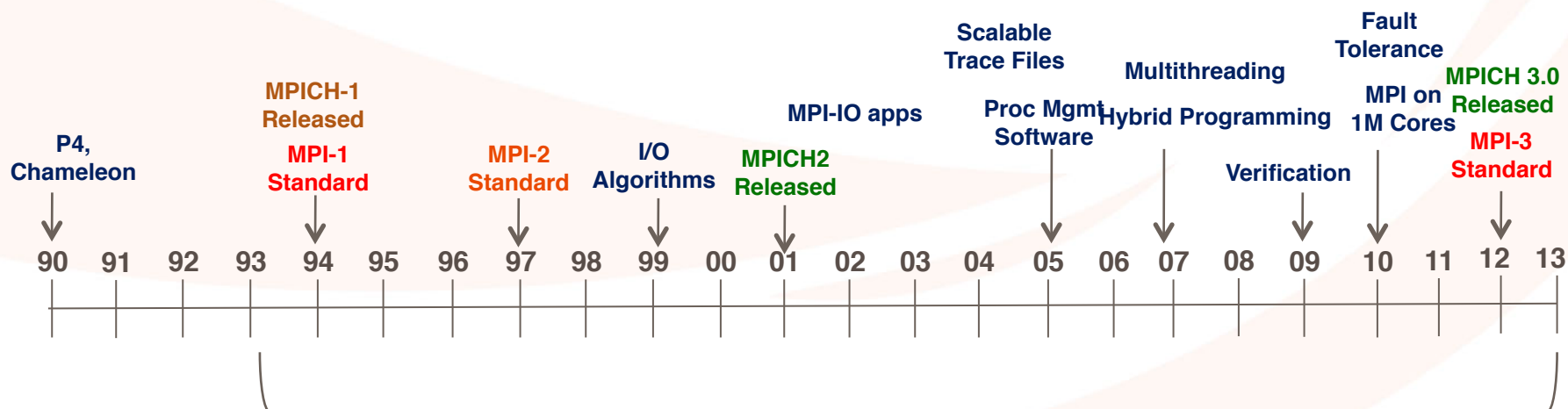
- Two very distinct timescales
  - Fast: New way of looking at organization
  - Slow: Work of implementation, tuning, extension
- Requires sustained effort to provide end-to-end support, extend to new application needs



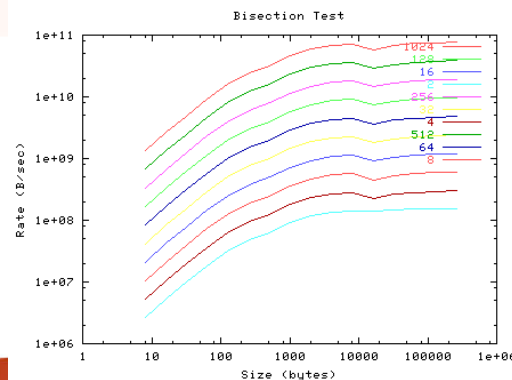
“Why we couldn't use numerical libraries for PETSc,” Proceedings of the IFIP TC2/WG2.5 Working Conference on the Quality of Numerical Software, Assessment and Enhancement, 1997.



# MPI and MPICH Timeline



## Performance research





# What is the goal of a software institute?

- Create new software?
- Improve current software?
- “Harden” research prototypes?
- Create a better process for community software?



# Create software that doesn't exist yet?

- Why? Who will use? How big is the user base? What is the value WRT development effort?
- Example: Why we created PETSc
  - Problem: Performing research into parallel domain decomposition algorithms
    - Divide domain into parts, solve on parts, put back together
    - May want to recurse (solve by applying domain decomposition)
  - Most numerical libraries of the time unusable
    - Global state: Can't nest library calls; some have high overhead for initialization
    - No routines to solve problems – only routines to apply a specific algorithm
    - Often “unnatural” data structures (designed for algorithm, not problem)
    - No parallelism
  - PETSc created with a very specific user in mind – me!
    - Not as a numerical library for others – if we had, it would have been another of many such projects that has since disappeared



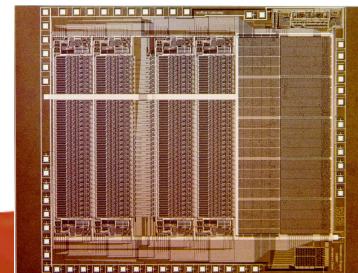
# Improve current software?

- Why? What quantitative improvement is needed?
- Recall claims from petascale studies that “The software for the current generation of 100 GF machines is not adequate to be scaled to a TF...”
- That statement based on *no* evidence
  - Just a feeling that “It has to be better than this”
- Any claims must be specific both to what limitations exist and why they can’t be solved with known methods
  - Productivity arguments talk about simple codes – but MPI’s strength is its support for programming in the large, not the support for short, simple programs
- Yes, using/extending known tools may involve paying for software. Would an institute be more cost effective? Why?



# Create usable software from research prototypes?

- Nurture an existing prototype software to build a user base (middle age support for software)
  - Why? How do you select? What is the quantitative value?
  - Can you commit to the length of time often needed to build a user base?
  - Q: When was the GPU introduced by NVIDIA?
    - Founded 1993, GeForce in 1999, CUDA 2006
    - Q: when did SGI introduce a graphics processing chip?
      - 1982: Geometry Engine



# Create a stronger ecosystem for software development?

- Meta support – Support others in the development of high value software
  - Why? What is needed? Many open source tools exist (e.g., testing, documentation, code transformation, open compiler frameworks). Why are these not adequate?
  - How? Code development notoriously personal and immune to rational discussion (e.g., language wars)
  - Who? Current coders and development teams, or students before they are corrupted?



# Challenges for Current Software

- Obvious:
  - Code performance
    - Core performance
    - Node performance
    - Scalability
  - Correctness and Testing
  - Portability
    - Especially in a era of rapidly changing hardware, compute architecture, and constraints such as power
  - Productivity
    - Whatever that really means
- Maybe less obvious
  - Enabling/encouraging new algorithms
  - More quantitative approach to development/evaluation
  - Performance irregularity in all elements
  - Data-centric applications and workflows
- And so on...

