Argonne NATIONAL LABORATORY

... for a brighter future

U.S. Department of Energy

THE UNIVERSITY OF CHICAGO

Office of Science U.S. DEPARTMENT OF ENERGY

A U.S. Department of Energy laboratory managed by The University of Chicago

Where Does MPI Need To Grow?

William D. Gropp
Mathematics and Computer Science
www.mcs.anl.gov/~gropp

What this Talk is About

- I was asked to talk about
  - “Real or perceived shortcomings of MPI”
  - “The need, if any, for extensions or modifications of the standard”
  - “Pressing implementation issues”
- But the issue is really
  - What are the needs of the parallel software ecosystem?
  - How does MPI fit into that ecosystem?
  - What are the missing parts (not just from MPI)?
- Lets start with some history …

Quotes from "System Software and Tools for High Performance Computing Environments" (1993)

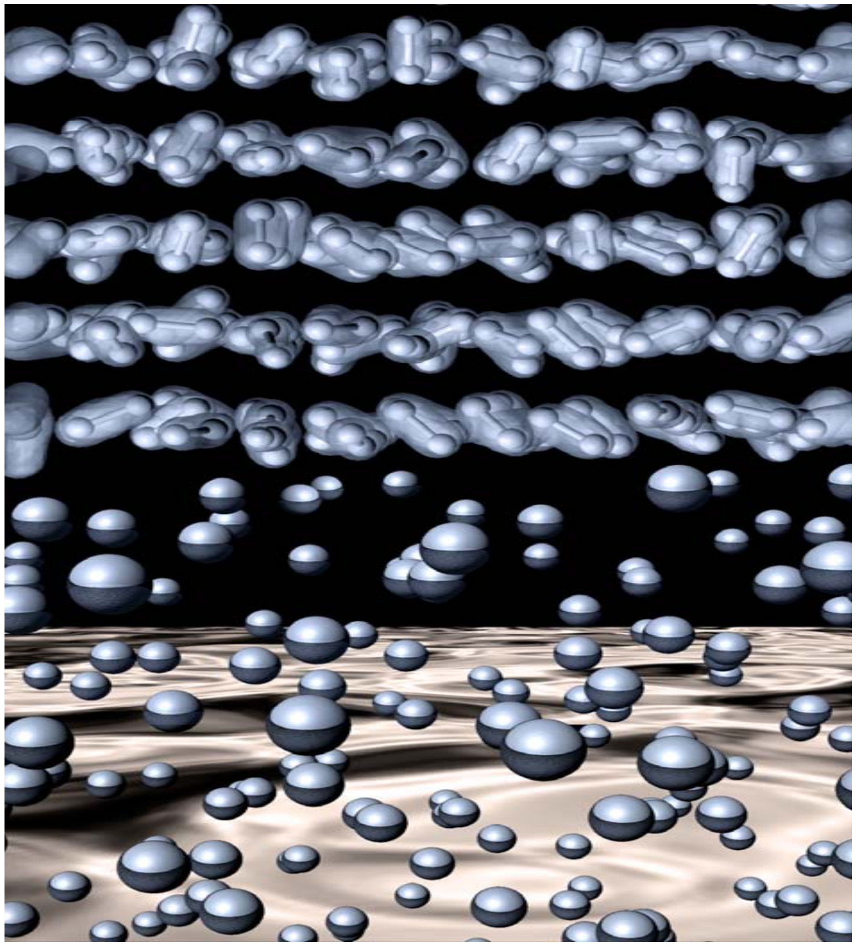
- **"The strongest desire expressed by these users was simply to satisfy the urgent need to get applications codes running on parallel machines as quickly as possible"**
- **In a list of enabling technologies for mathematical software, "Parallel prefix for arbitrary user-defined associative operations 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."**
- **"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)**

How's MPI Doing?

- Great!
- In June of this year, the Qbox materials science code achieved a sustained 207.3TF on the BlueGene/L at LLNL.
  - A doubling of achieved sustained performance since last November
  - Of course, the hard part in this was getting so much performance out of one node, not the MPI part….
- http://www.llnl.gov/PAO/news/news_releases/2006/NR-06-06-07.html
- Before changing MPI, we need to understand why MPI has succeeded



Why Was MPI Successful?

- It address all of the following issues:
  - Portability
  - Performance
  - Simplicity and Symmetry
  - Modularity
  - Composability
  - Completeness

Portability and Performance

- Portability does not require a “lowest common denominator” approach
  - Good design allows the use of special, performance enhancing features without requiring hardware support
  - For example, MPI’s nonblocking message-passing semantics allows but does not require “zero-copy” data transfers
- MPI is really a “Greatest Common Denominator” approach
  - It *is* a “common denominator” approach; this is portability
    - *To fix this, you need to change the hardware (change “common”)*
  - It *is* a (nearly) greatest approach in that, within the design space (which includes a library-based approach), changes don’t improve the approach
    - *Least suggests that it will be easy to improve; by definition,* any *change would improve it.*
    - *Have a suggestion that meets the requirements? Lets talk about it this evening!*

Simplicity and Symmetry

- MPI is organized around a small number of concepts
  - The number of routines is not a good measure of complexity
  - E.g., Fortran
    - *Large number of intrinsic functions*
  - C and Java runtimes are large
  - Development Frameworks
    - *Hundreds to thousands of methods*
  - This doesn’t bother millions of programmers

Symmetry

- Exceptions are hard on users
  - But easy on implementers — less to implement and test
- Example: MPI_Issend
  - MPI provides several send modes:
    - *Regular*
    - *Synchronous*
    - *Receiver Ready*
    - *Buffered*
  - Each send can be blocking or non-blocking
  - MPI provides all combinations (symmetry), including the "Nonblocking Synchronous Send"
    - *Removing this would slightly simplify implementations*
    - *Now users need to remember which routines are provided, rather than only the concepts*
  - It turns out he MPI_Issend is useful in building performance and correctness debugging tools for MPI programs
    - *Used in FPMPI2 to estimate time a rendezvous send may be blocked waiting for the matching receive*
- Some symmetries may not be worth the cost
  - MPI cancel of send

Modularity

- Modern algorithms are hierarchical
  - Do not assume that all operations involve all or only one process
  - Software tools must not limit the user
- Modern software is built from components
  - MPI designed to support libraries
  - Example: communication contexts, static communicators

Composability

- Environments are built from components
  - Compilers, libraries, runtime systems
  - MPI designed to “play well with others”
- MPI exploits newest advancements in compilers
  - … without ever talking to compiler writers
  - OpenMP is an example
    - *MPI (the standard) required <u>no</u> changes to work with OpenMP*
    - *MPI Thread modes provided for performance reasons*
- MPI was designed from the beginning to work within a larger collection of software tools
  - What’s needed to make MPI better? More good tools!

Completeness

- MPI provides a complete parallel programming model and avoids simplifications that limit the model
  - Contrast: Models which require that synchronization only occurs collectively for all processes or tasks
- Make sure that the functionality is there when the user needs it
  - Don’t force the user to start over with a new programming model when a new feature is needed

Conclusions: Lessons From MPI

- A successful parallel programming model must enable more than the simple problems
  - It is nice that those are easy, but those weren’t that hard to begin with
- Scalability is essential
  - Why bother with limited parallelism?
  - Just wait a few months for the next generation of hardware
- Performance is equally important
  - But not at the cost of the other items
- It must also fit into the Software Ecosystem
  - MPI did not replace the languages
  - MPI did not dictate particular process or resource management
  - MPI defined a way to build tools by replacing MPI calls (the profiling interface)
  - (later) Other interfaces, such as debugging interface, also let MPI interoperate with other tools

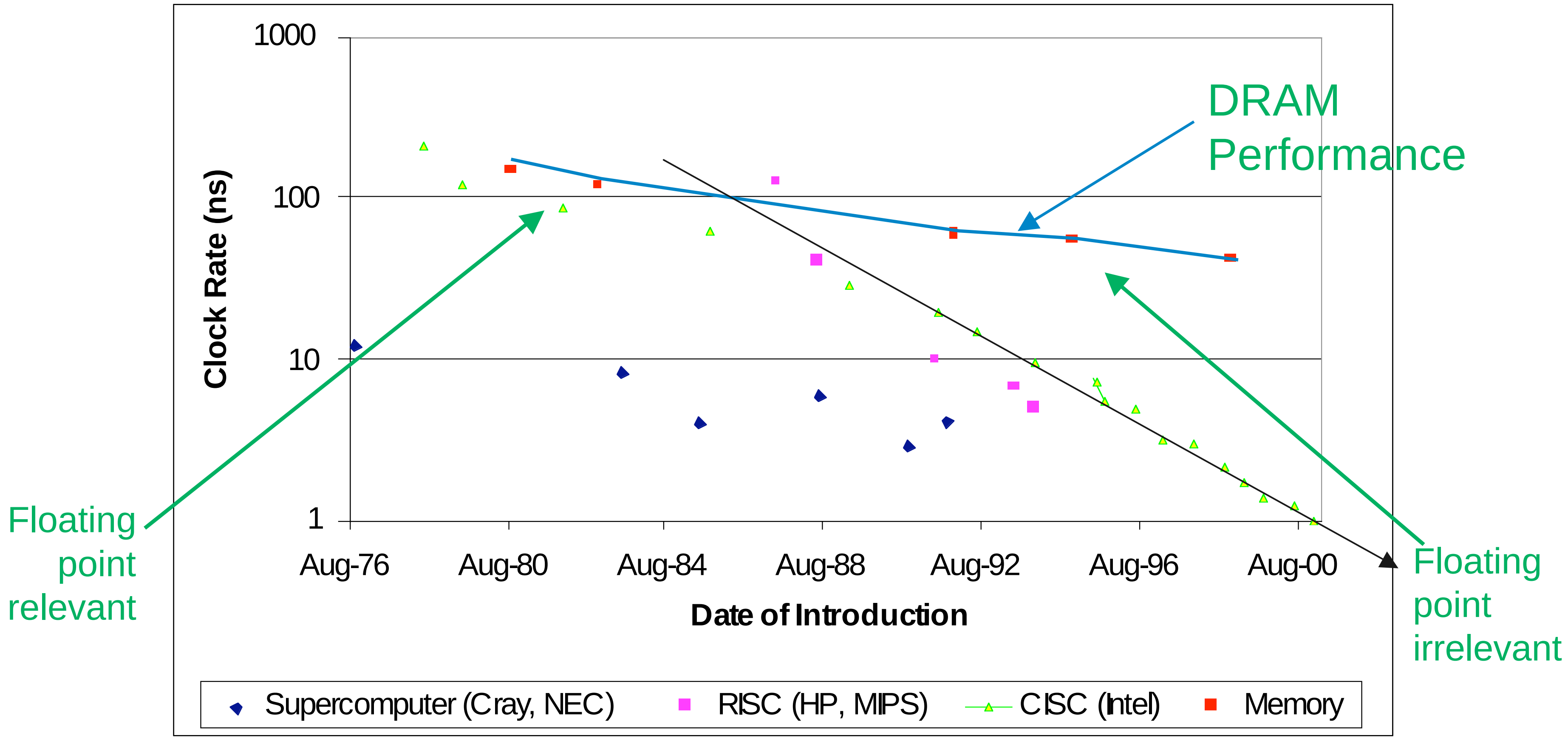
Some Weaknesses in MPI

- Easy to write code that performs and scales poorly
  - Using blocking sends and receives
    - *The attractiveness of the blocking model suggests a mismatch between the user's model and the MPI model of parallel computing*
  - The right fix for this is better performance tuning tools
    - *Don't change MPI, improve the environment*
    - *The same problem exists for C, Fortran, etc.*
- No compile-time optimizations
  - Only MPI_Wtime, MPI_Wtick, and the handler conversion functions may be macros.
  - Sophisticated analysis allows inlining
  - Does it make sense to optimize for important special cases
    - *Short messages? Contiguous Messages? Are there lessons from the optimizations used in MPI implementations?*

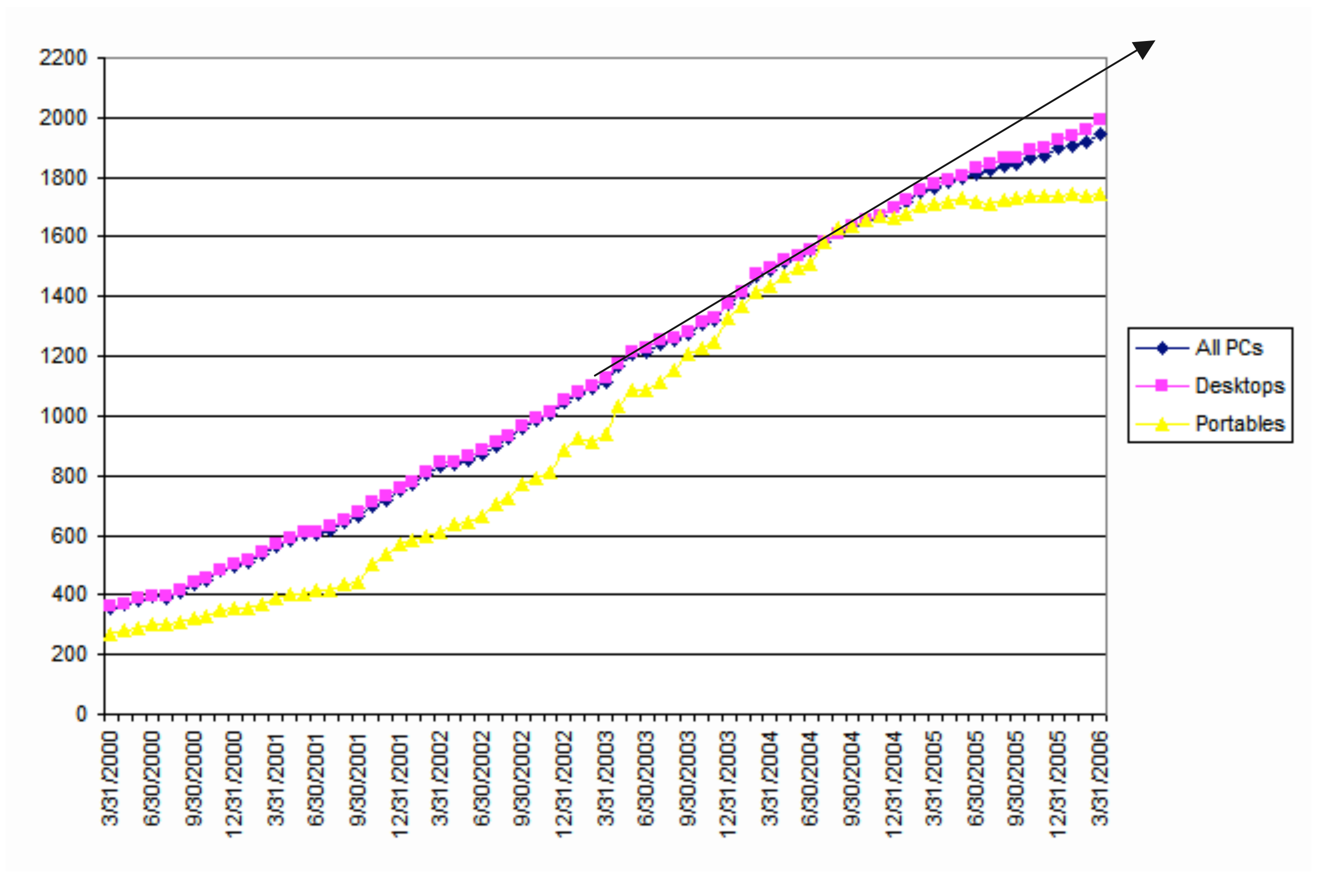
Performance Issues

- Library interface introduces overhead
  - ~200 instructions ?
- Hard (though not impossible) to “short cut” the MPI implementation for common cases
  - Many arguments to MPI routines
  - These are due to the attempt to limit the number of basic routines
    - *You can’t win --- either you have many routines (too complicated) or too few (too inefficient)*
    - *Is MPI for users? Library developers? Compiler writers?*
- Computer hardware has changed since MPI was designed (1992 - e.g., DEC announces Alpha)
  - SMPs are more common
  - Cache-coherence (within a node) almost universal
    - *MPI RMA Epochs provided (in part) to support non-coherent memory*
  - Interconnect networks
  - CPU/Memory/Interconnect speed ratios
  - Note that MPI is often blamed for the poor fraction of peak performance achieved by parallel programs. But is MPI the culprit?

Why is achieved performance on a single node so poor?

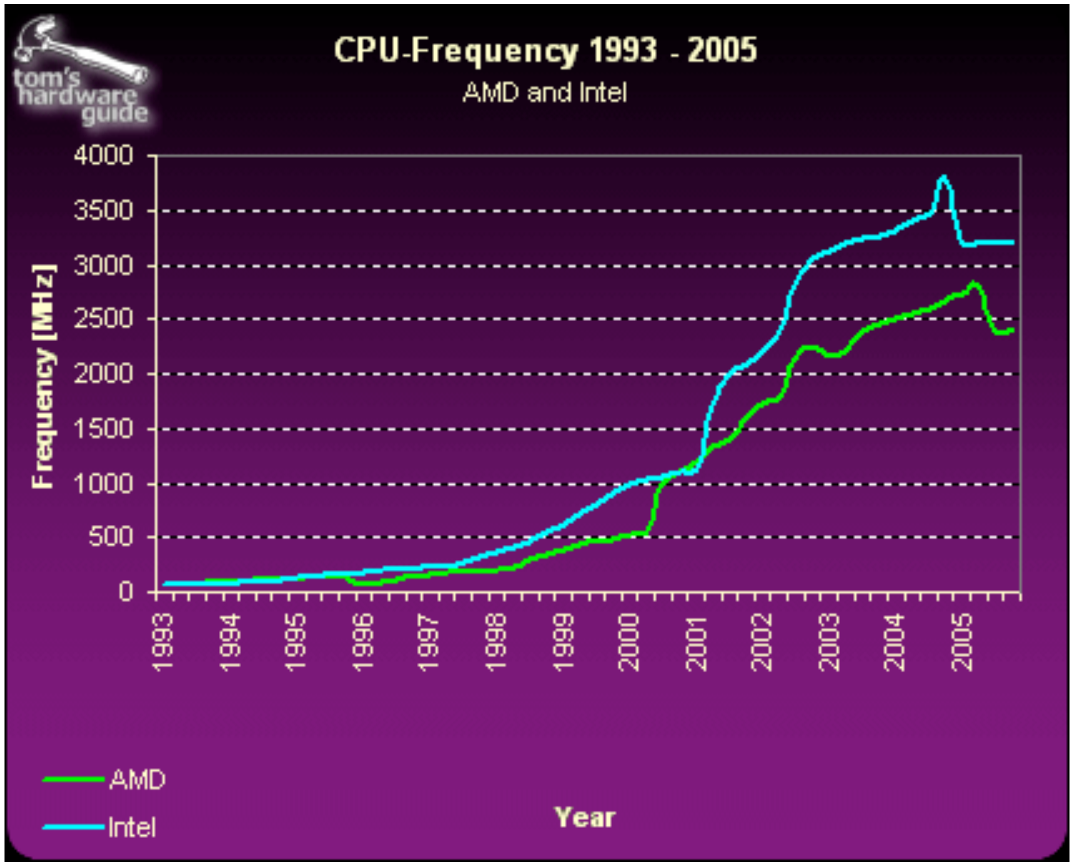


CPU performance is not doubling any more



- Average CPU clock speeds (from http://www.pcpitstop.com/research/cpu.asp)

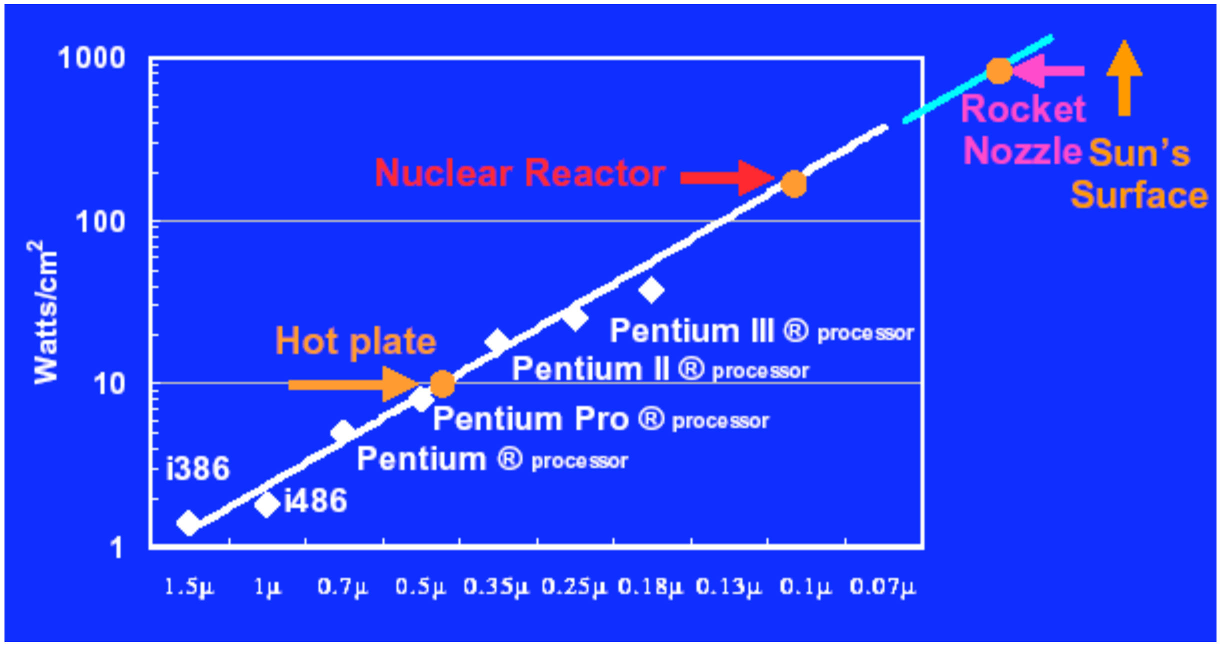
Peak CPU speeds are stable



- From http://www.tomshardware.com/2005/11/21/the_mother_of_all_cpu_charts_2005/

Why are CPUs not getting faster?

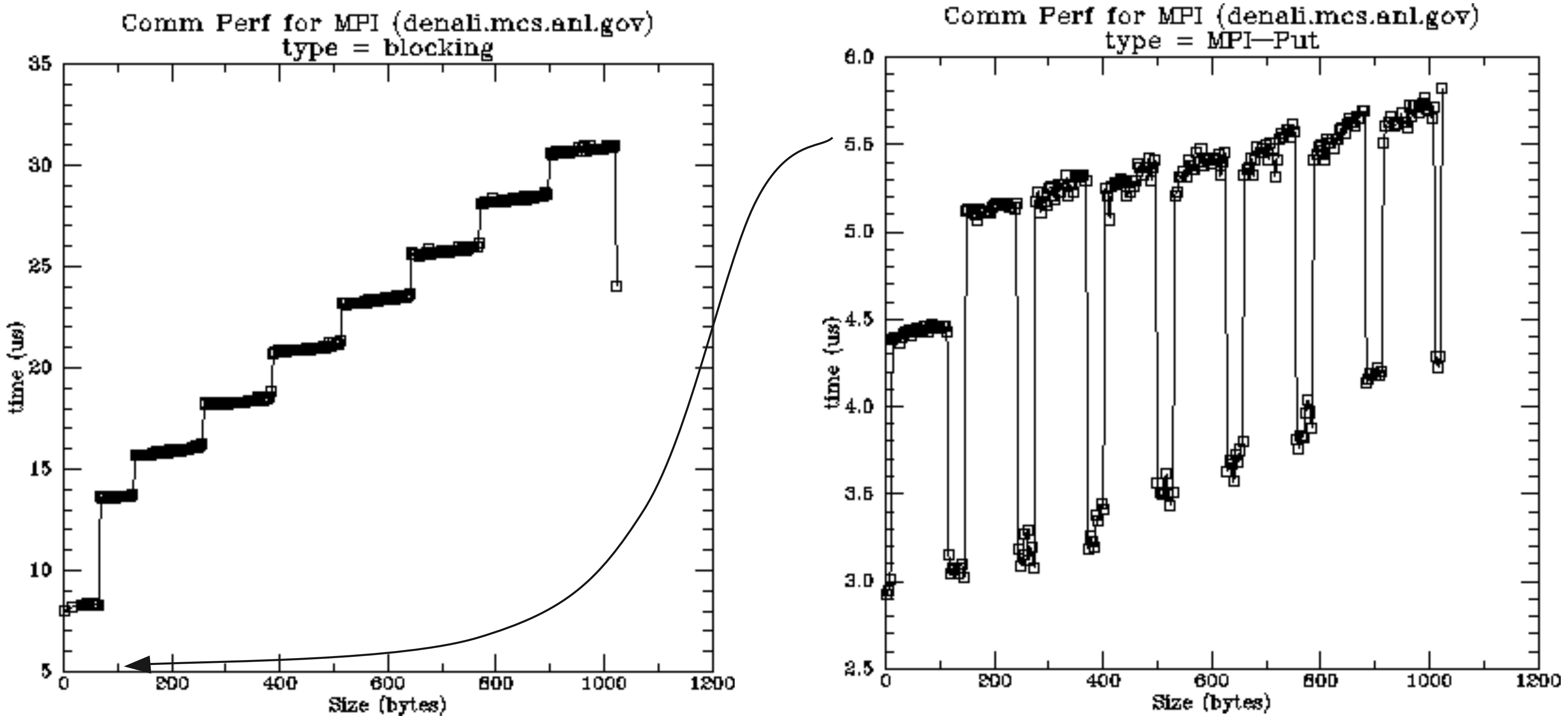
- Power dissipation problems will force more changes
  - Current trends imply chips with energy densities greater than a nuclear reactor
  - Already a problem: Exploding laptops.
  - Will force new ways to get performance, such as extensive parallelism
  - Source of the drive to multicore



Performance Issues (2)

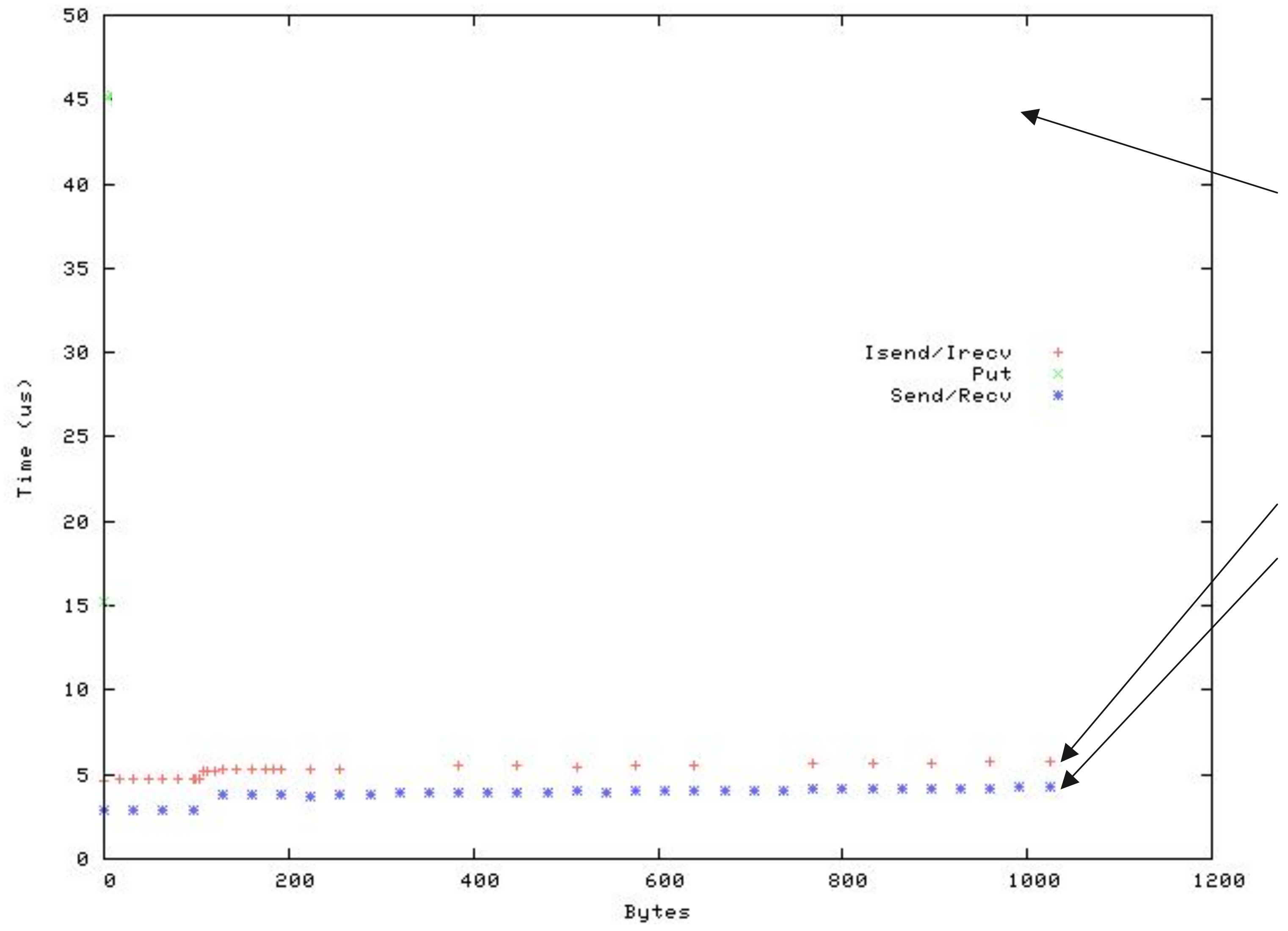
- MPI-2 RMA design supports non-cache-coherent systems
  - Good for portability to systems of the time
  - Complex rules for memory model (confuses users)
  - Performance consequences
    - *Memory synchronization model*
    - *One example: Put requires an ack from the target process*
- Missing operations
  - No Read-Modify-Write operations
  - Very difficult to implement even fetch-and-increment
    - *Requires indexed datatypes to get scalable performance(!)*
    - *We've found bugs in vendor MPI RMA implementations when testing this algorithm*
  - Challenge for any programming model
    - *What operations are provided?*
    - *Are there building blocks, akin to the load-link/store-conditional approach to processor atomic operations?*

Performance of RMA



Caveats: On this SGI implementation, MPI_Put uses specially allocated memory

RMA and Send/Recv Performance Comparison



Huge Penalty

Why the large penalty?

Data taken September, 2006

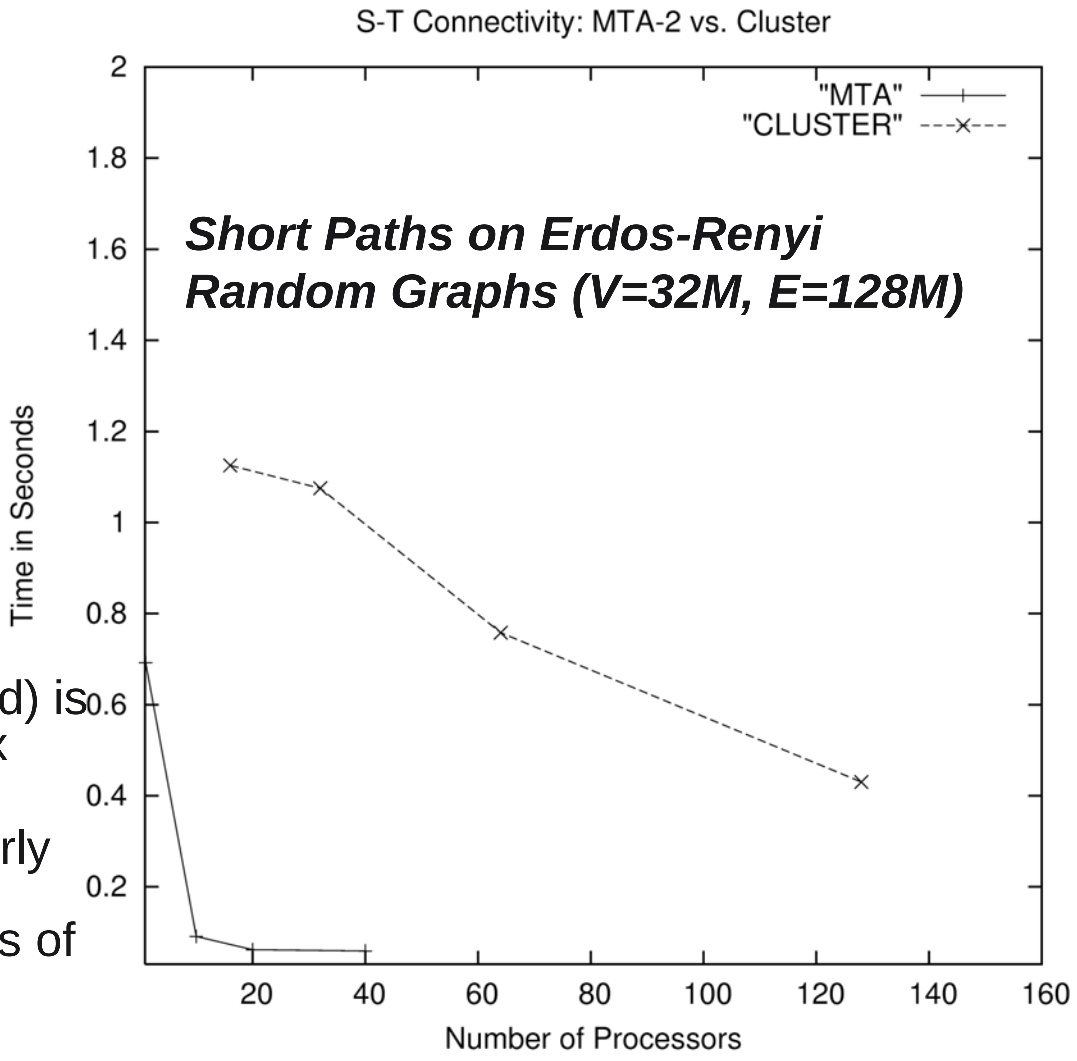
Performance Issues (3)

- Split (nonblocking) operations
  - Necessary for latency hiding
  - MPI supports split operations (e.g., MPI_Isend, MPI_Put) but extra overhead of library calls (in point to point) and awkward synchronization model (e.g., one-sided with get) eliminates some of the benefit
  - How much of CPU/Memory latency gap can MPI hide, and are there better was?

Factors of 100 or more?

- From
“Parallel Graph Algorithms: Architectural Demands of Pathological Applications”
Bruce Hendrickson
Jonathan Berry
Keith Underwood
Sandia National Labs,
Richard Murphy
Notre Dame University

- MTA (with 1/10 of clock speed) is 10x the performance - a 100x potential improvement
- Source of improvement - nearly complete overlap of memory access latency, which is 100’s of clock cycles



Productivity

- No support for distributed data structures
  - Other than MPI_Type_create_darray and MPI_Type_create_subarray
    - *Examples of the difficulties of a library-based approach*
    - *(problem is that past matrix and simple grids, this gets very difficult to do efficiently and generally) - examples from previous year*
- Action at a distance
  - No built-in support for easily identifying matching send/receive, put/load, or store/get operations
- Poor integration with the host language
  - Datatypes for structure definitions
  - Datatypes with dynamic fields
    - *E.g., a single MPI datatype for a struct iov, where each entry is length, pointer to storage of that length*
  - Note that this was an important part of MPI's success
    - *Independent from compilers meant MPI programs could take advantage of the best compilers*
    - *It would be nice if there was a better way to provide language info to libraries*
      - Example: Fortran 90 interface really requires dynamic creation based on use in application program

Scalability Issues in the MPI Definition

- How should you define scalable?
  - Independent of number of processes
- Some routines do not have scalable arguments
  - E.g., MPI_Graph_create
- Some routines require O(p) arrays
  - E.g., MPI_Group_incl, MPI_Alltoall
- Group construction is explicit (no MPI_Group_split)
- Implementation challenges
  - MPI_Win definition, if you wish to use a remote memory operation by address, requires each process to have the address of each remote processes local memory window (O(p) data at each process).
  - Various ways to recover scalability, but only at additional overhead and complexity
    - *Some parallel approaches require "symmetric allocation"*
    - *Many require Single Program Multiple Data (SPMD)*
  - Representations of Communicators other than MPI_COMM_WORLD (may be represented implicitly on highly scalable systems)

Issues that are not issues (Red Herrings)

- MPI and RDMA programming models
  - How do you signal completion at the target?
  - Cray SHMEM succeeded because of SHMEM_Barrier - an easy and efficiently implemented (with special hardware) way to indicate completion of RDMA operations
- Latency
  - Users often confuse Memory access times and CPU times; expect to see remote memory access times on the order of register access
  - Without overlapped access, a single memory reference is 100’s to 1000’s of cycles
  - A load-store model for reasoning about program performance isn’t enough
    - *Don’t forget memory consistency issues*
- Fault Tolerance (as an MPI problem)
  - Fault Tolerance is a property of the application; there is no magic solution
  - MPI *implementations* can support fault tolerance
  - MPI *intended* implementations to continue through faults when possible
    - *That’s why there is a sophisticated error reporting mechanism*
    - *What is needed is a higher standard of MPI* implementation*, not a change to the MPI standard*
  - But - Some algorithms do need a more convenient way to manage a collection of processes that may change dynamically
    - *This is not a communicator*

Where Does MPI Need to Change?

- Nowhere
  - There are many MPI legacy applications
  - MPI has added routines to address problems rather than changing them
  - For example, to address problems with the Fortran binding and 64-bit machines, MPI-2 added MPI_Get_address and MPI_Type_create_xxx and deprecated (but did not change or remove) MPI_Address and MPI_Type_xxx.
- Where does MPI need to add routines and deprecate others?
  - One Sided
    - *Designed to support non-coherent memory on a node, allow execution in network interfaces, and nonblocking memory motion*
    - *Put requires ack (to enforce ordering)*
    - *Lock/put/unlock model very heavy-weight for small updates*
    - *Generality of access makes passive-target RMA difficult to implement efficiently (exploiting RDMA hardware)*
    - *Users often believe MPI_Put and MPI_Get are blocking (poor choice of name, should have been MPI_Iput and MPI_Iget).*
  - Various routines with “int” arguments for “count”
    - *In a world of 64 bit machines and multiGB laptops, 32-bit ints are no longer large enough*

Extensions

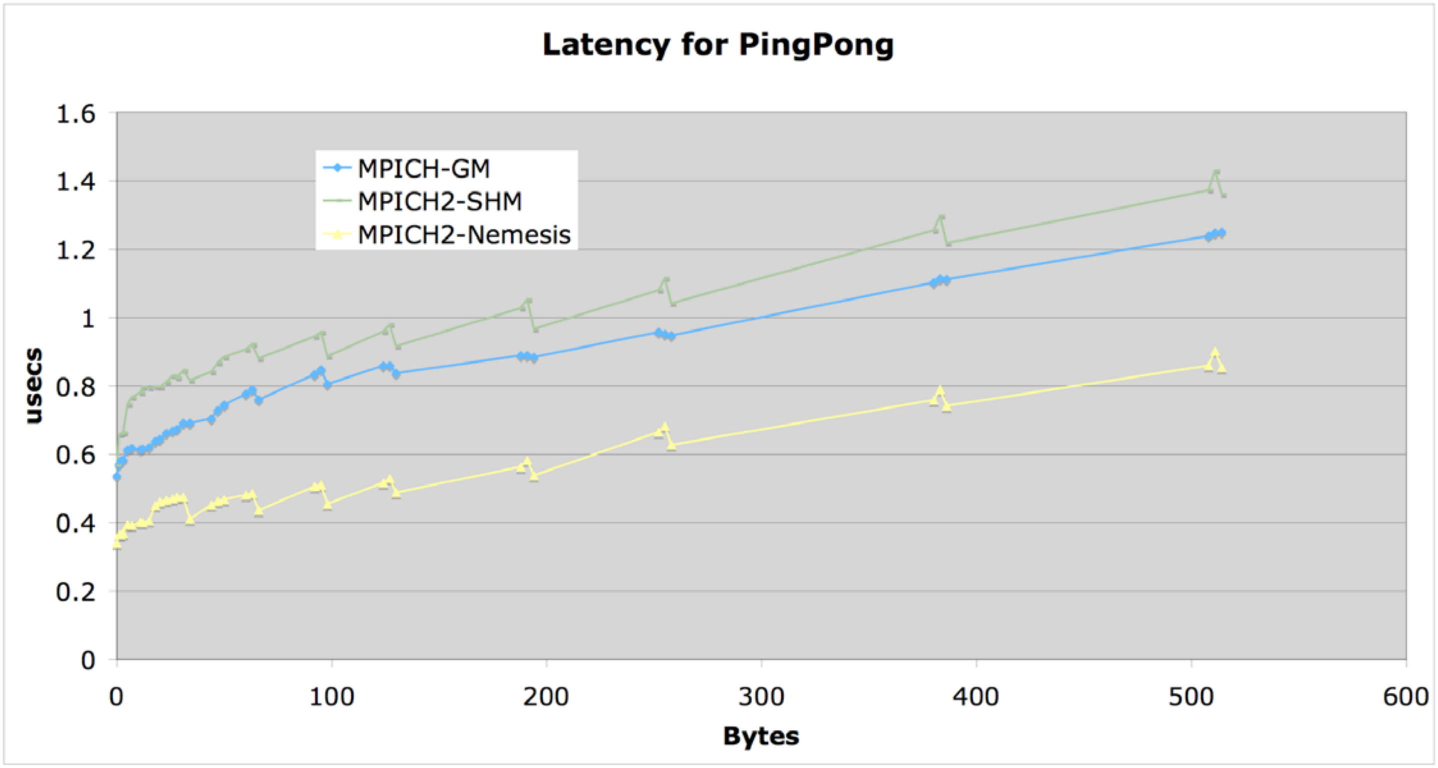
- What does MPI need that it doesn't have?
- Don't start with that question. Instead ask
  - What tool do I need? Is there something that MPI needs to work well with that tool (that it doesn't already have)?
- Example: Debugging
  - Rather than define an MPI debugger, develop a thin and simple interface to allow any MPI implementation to interact with any debugger
- Candidates for this kind of extension
  - Interactions with process managers
    - *Thread co-existance (MPIT discussions)*
    - *Choice of resources (e.g., placement of processes with Spawn) (why - Cera et al; Leopold et al)*
  - Interactions with Integrated Development Environments (IDE)
  - Tools to create and manage MPI datatypes (why - Byna et al; Kimpe et al)
  - Tools to create and manage distributed data structures
    - *A feature of the HPCS languages*

Implementation Issues

- Intel's "MPI needs" are a good survey:
  - Application performance (overnight turnaround)
  - Network independence
  - Thread safety for mixed mode programming
  - Communication/computation overlap
  - ABI stability between product versions
  - Failover support
  - Job schedule integration
  - Linux and Windows CCS support
  - Ease of use and manageability
- Most of these are challenges internal to the MPI implementation
- The last few involve better interactions with other parts of the parallel software environment

Implementations Issues from the MPICH2 Perspective

- "MPI takes 1200 instructions for send-receive"
  - MPICH2 down to ~500 instructions, 340ns on dual Opteron
  - More room for improvement
  - Unlikely to go below 200 instructions



Implementations Issues from the MPICH2 Perspective (2)

- "MPI RMA is slow"
  - Better implementation approaches give much better performance (presented at previous EuroPVMMPI meetings)
  - There are still some issues with MPI RMA definition
- Job Startup
  - MPICH2 jobs use a *process management interface* that provides an *interface* for scalable, robust startup
  - The same "a.out" can be run with any parallel process manager that implements the interface
- Efficient Thread Safety
- Still need to improve performance of collectives, topologies, datatypes, I/O
  - Optimizations need information about the system (interconnect topology, memory hierarchy, I/O system structure, etc.)
  - Optimizations also need information from the application (available memory, processing priorities, etc.)
  - A common abstraction (or better yet, interface) would benefit many components

Expanding the Ecosystem

- Center for the Improvements of Fault Tolerance in Systems (CIFTS) or FOBAWS
  - ANL, LBNL, ORNL, Indiana Univ., Ohio State Univ., and Univ. Tennessee
  - Developing a “fault tolerance backplane” to allow components to work together to predict, detect, and handle faults in the system
  - Includes exploring interfaces for MPI implementations to talk to the fault tolerant backplane
- Can there be a Common Communication Layer?
  - Provide an implementation for PGAS languages and MPI
  - Allow interoperation of PGAS and MPI codes
    - *Incremental porting of MPI codes to PGAS*
  - Many subtle issues (progress :))
- Parallel Environments …

Improving Parallel Programming

- How can we make the programming of real applications easier?
- Problems with the Message-Passing Model
  - User’s responsibility for data decomposition
  - “Action at a distance”
    - *Matching sends and receives*
    - *Remote memory access*
  - Performance costs of a library (no compile-time optimizations)
  - Need to choose a particular set of calls to match the hardware
- In summary, the lack of abstractions that match the applications

Challenges

- Must avoid the traps:
  - The challenge is not to make easy programs easier. The challenge is to make hard programs possible.
  - We need a "well-posedness" concept for programming tasks
    - *Small changes in the requirements should only require small changes in the code*
    - *Rarely a property of "high productivity" languages*
      - Abstractions that make easy programs easier don't solve the problem
  - Latency hiding is not the same as low latency
    - *Need "Support for aggregate operations on large collections"*
- An even harder challenge: make it hard to write incorrect programs.
  - OpenMP is not a step in the (entirely) right direction
  - In general, current shared memory programming models are very dangerous.
    - *They also perform action at a distance*
    - *They require a kind of user-managed data decomposition to preserve performance without the cost of locks/memory atomic operations*
  - Deterministic algorithms should have provably deterministic implementations
    - *Its great to see formal methods applied to MPI!*

What is Needed To Achieve Real High Productivity Programming

- Simplify the construction of correct, high-performance applications
- Managing Data Decompositions
  - Necessary for both parallel and uniprocessor applications
  - Many levels must be managed
  - Strong dependence on problem domain (e.g., halos, load-balanced decompositions, dynamic vs. static)
- Possible approaches
  - Language-based
    - *Limited by predefined decompositions*
      - Some are more powerful than others; Divacon provided a built-in divided and conquer
  - Library-based
    - *Overhead of library (incl. lack of compile-time optimizations), tradeoffs between number of routines, performance, and generality*
  - Domain-specific languages …

Domain-specific languages

- A possible solution, particularly when mixed with adaptable runtimes
- Exploit composition of software (e.g., work with existing compilers, don't try to duplicate/replace them)
- Example: mesh handling
  - Standard rules can define mesh
    - *Including "new" meshes, such as C-grids*
  - Alternate mappings easily applied (e.g., Morton orderings)
  - Careful source-to-source methods can preserve human-readable code
  - In the longer term, debuggers could learn to handle programs built with language composition (they already handle 2 languages – assembly and C/Fortran/…)
- Provides a single "user abstraction" whose implementation may use the composition of hierarchical models
  - Also provides a good way to integrate performance engineering into the application

Some Answers to the Original Questions

- Real or perceived shortcomings
  - Library overhead
  - Lack of distributed data structure support
  - RMA model
  - Scalability (perceived)
  - Fault Tolerance (perceived)
- Extensions or changes
  - ABI (not until g++ is stable)
  - RMA changes or additions
    - *Cache coherent*
    - *RMW*
  - Symmetric addressing (scalability)
  - Little fixes
    - *Ints for counts*
- Pressing Implementation Issues
  - Cost of wild card operations
  - Cost of message matching
  - Complexity of send cancel
    - *Evils of symmetry*
- See the session later today to discuss MPI 2+

Some Answers to the Revised Questions

- What are the needs of the parallel software ecosystem?
  - More support for tools to work together
- How does MPI fit into that ecosystem?
  - It’s the center (for now)
  - In most cases, MPI should be complemented with new, independent tools, rather than adding new groups of features within MPI
- What are the missing parts (not just from MPI)?
  - Support for higher-level programming
    - *Can but need not require new languages*

Conclusions

- MPI is alive and well
- There is still much to do to improve implementations
  - This conference continues to showcase the best research into better implementation approaches
  - Some parts of MPI's design, particularly for RMA, may need adjustment to achieve the intended performance
- MPI does *not* need new features
  - The parallel computing software ecosystem needs to work together to complement each tool
  - MPI has led the way
    - *With support for libraries*
    - *With the profiling and debugger interfaces*
- MPI (the standard) will continue to evolve
  - http://www-unix.mcs.anl.gov/~gropp/projects/parallel/MPI/mpi-errata/index.html
  - Open Forum, s1/s2 today at 6:15