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

Challenges for the Message Passing Interface in the Petaflops Era

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

What this Talk is About

- The title talks about MPI
  - Because MPI is the dominant parallel programming model in computational science
- 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)?
  - How can MPI adapt or be replaced in the parallel software ecosystem?
  - Short version of this talk:
    - *The problem with MPI is not with what it has but with what it is missing*
- Lets start with some history …

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

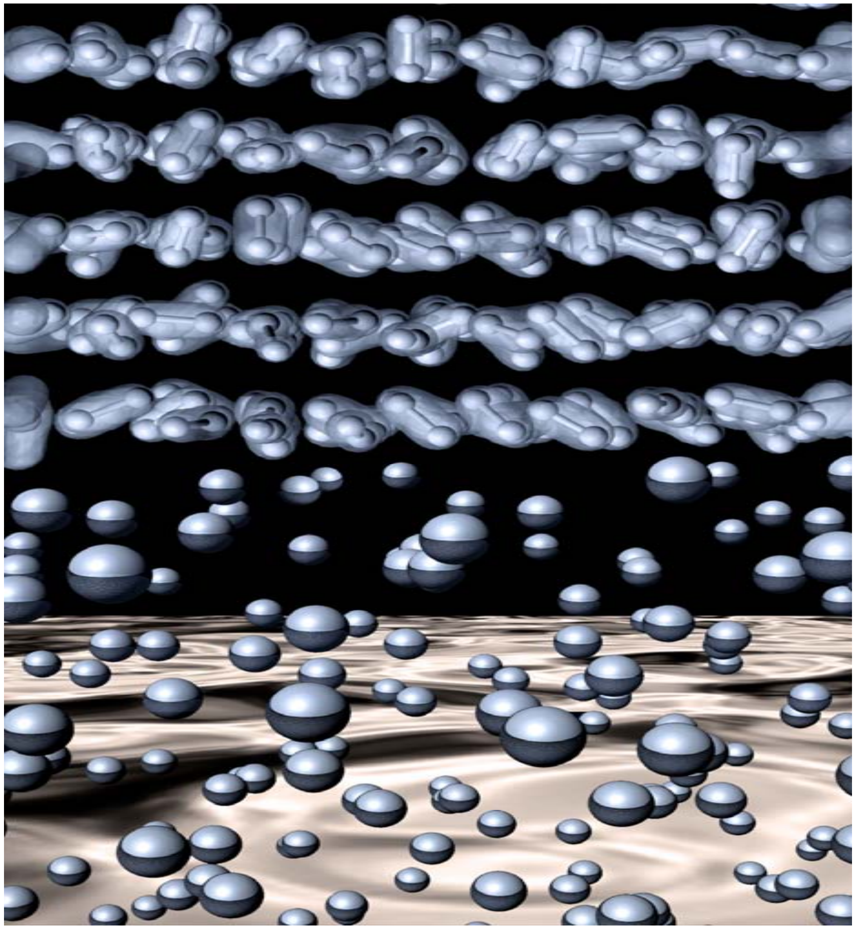
- **"The strongest desire expressed by these users was simply to satisfy the urgent need to get applications codes running on parallel machines as quickly as possible"**
- **In a list of enabling technologies for mathematical software, "Parallel prefix for arbitrary user-defined associative operations 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 2006, the Qbox materials science code achieved a sustained 207.3TF on the BlueGene/L at LLNL.
  - A doubling of achieved sustained performance since November 2005
  - 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 or replacing 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
- For a more complete discussion, see “Learning from the Success of MPI”, http://www.mcs.anl.gov/~gropp/bib/papers/2001/mpi-lessons.pdf

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!*
  - More on “Greatest” versus “Least” at the end of this talk…

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
  - Communication contexts in MPI are an example
    - *Other features, such as communicator attributes, were less successful features*

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 no 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 that require that synchronization only occurs collectively for all processes or tasks
  - Contrast: Models that provide support for a specialized (sub)set of distributed data structures
- 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?*

Issues that are not issues

- MPI and RDMA networks and 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*

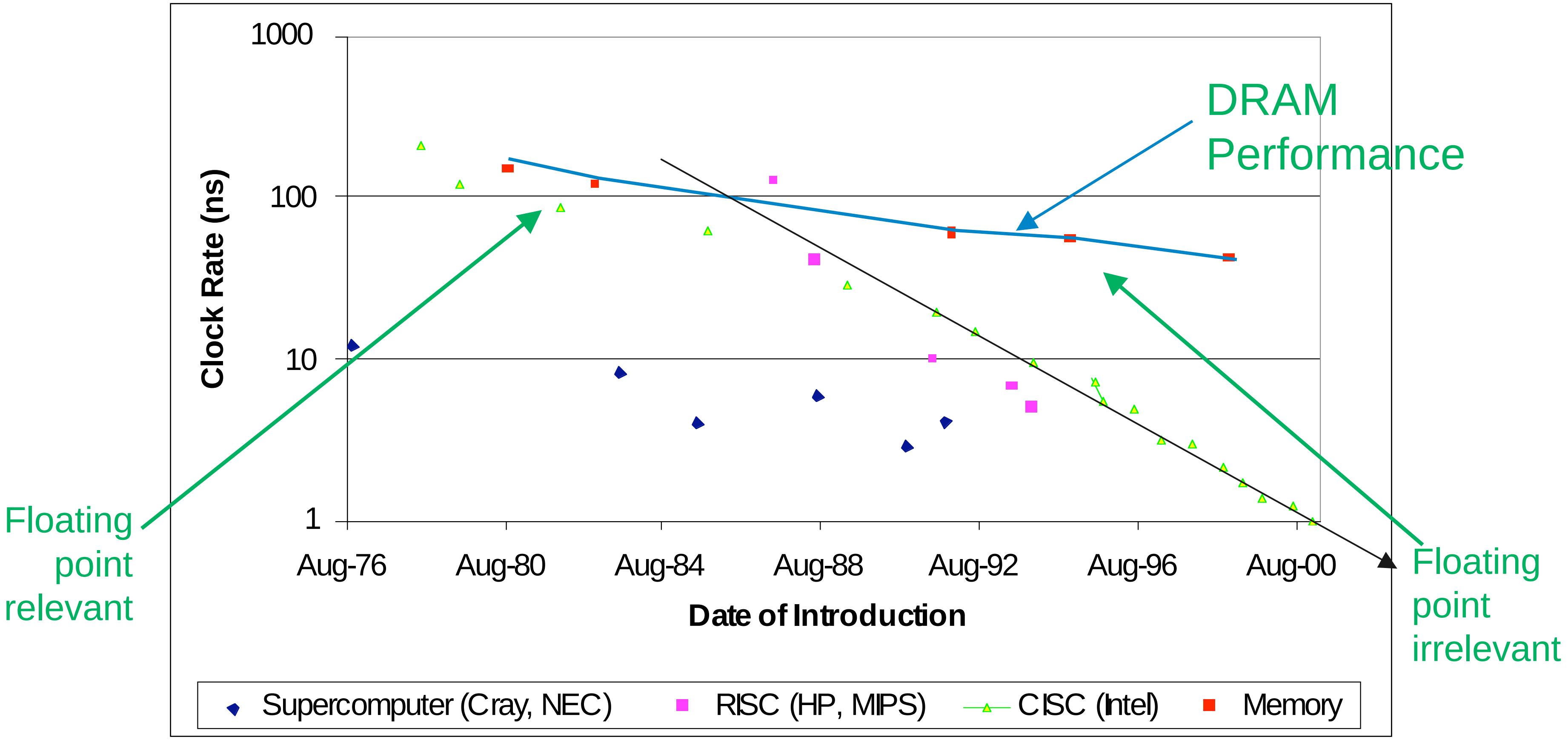
Scalability Issues in the MPI Definition

- 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)

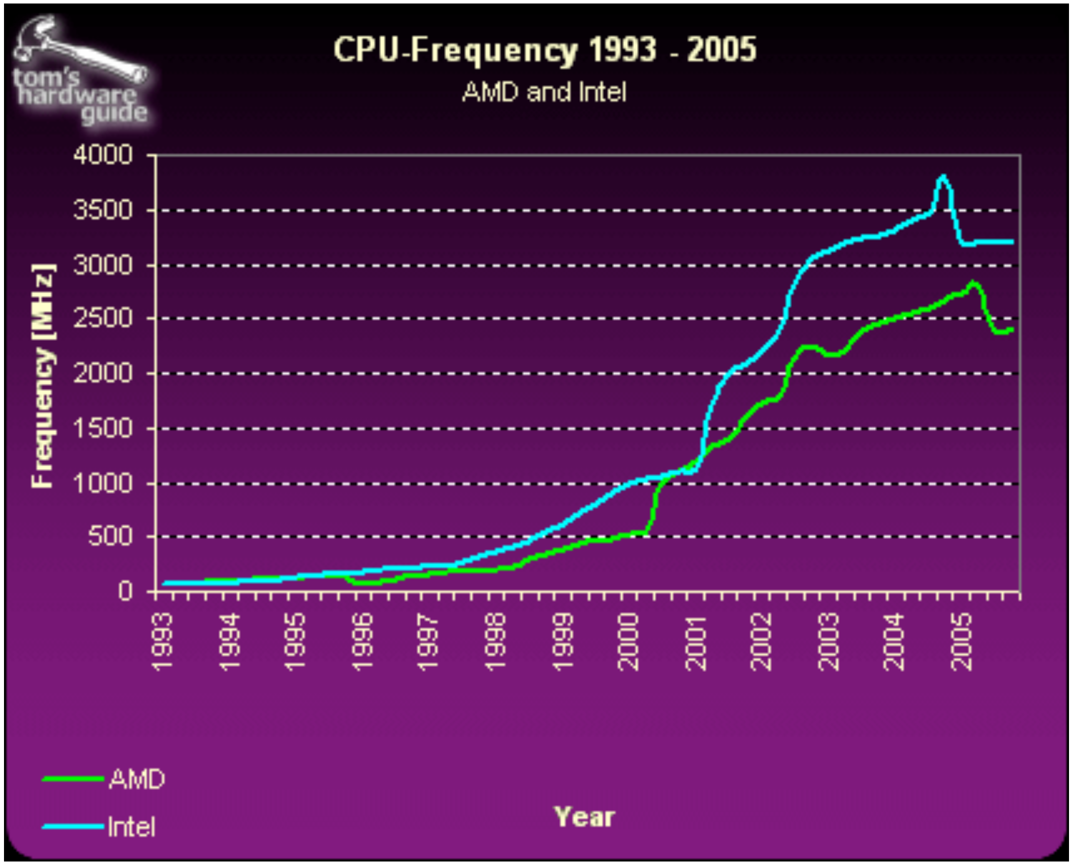
Performance Issues

- 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?



Peak CPU speeds are stable

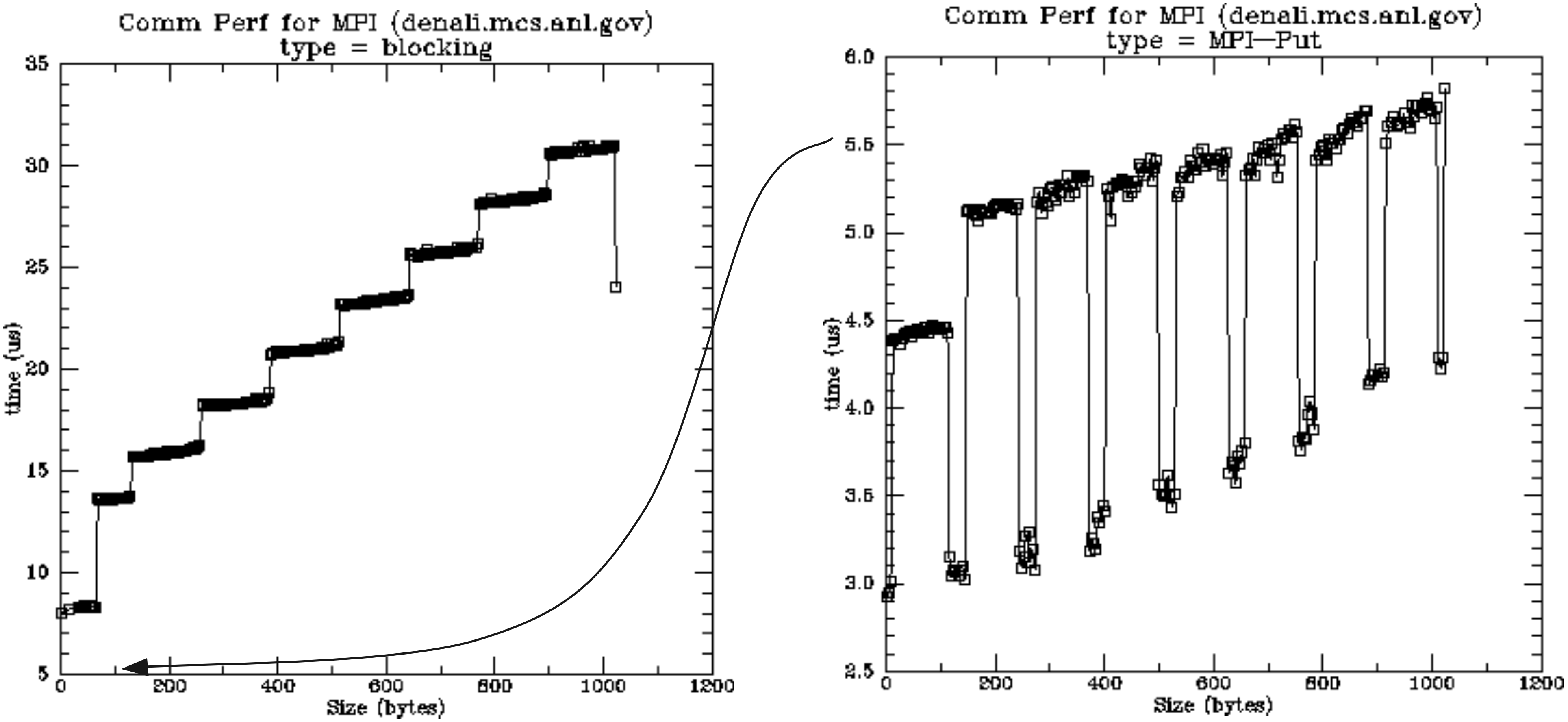


- From http://www.tomshardware.com/2005/11/21/the_mother_of_all_cpu_charts_2005/

Performance Issues (2)

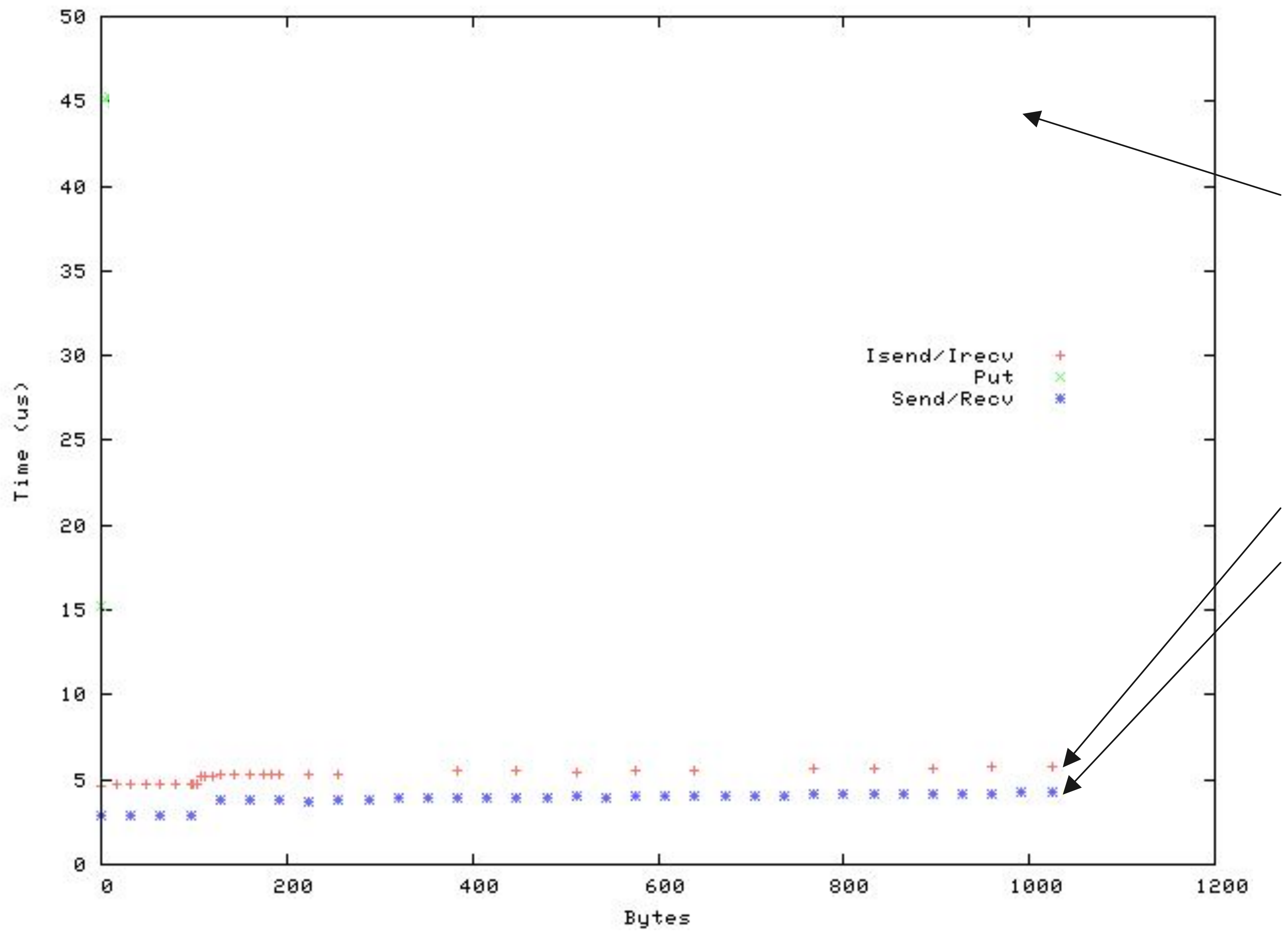
- MPI-2 RMA design supports non-cache-coherent systems
  - Good for portability to systems of the time
  - Complex rules for memory model (confuses users)
    - *But note that the rules are precise and the same on* all *platforms*
  - Performance consequences
    - *Memory synchronization model*
    - *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 ways?
  - Split operations introduce “programmer hazards”
    - *Common to initiate a split operation, such as a get, and then use the “result” before completing the operation*

Implementation Issues

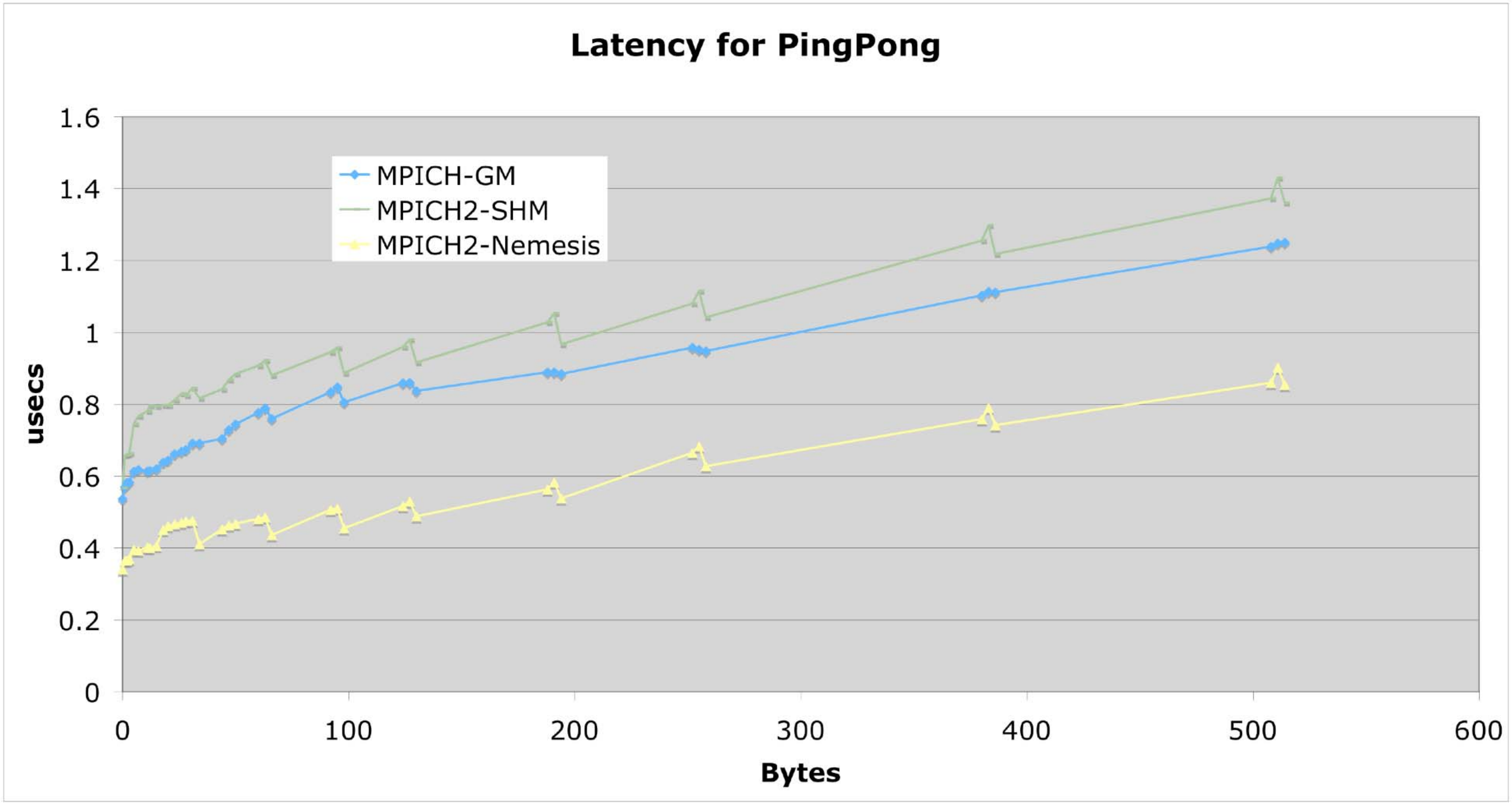
- Intel's surveyed its customers for "MPI needs":
  - Application performance (overnight turnaround)
  - Network independence
  - Thread safety for mixed mode programming
  - Communication/computation overlap
  - ABI stability between product versions
  - Failover support
  - Job schedule integration
  - 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
- And there's multicore …

Is it time to Panic on Multicore?

- "The way people write parallel programs now, with threads and locks, is intellectually challenging," said Microsoft Research lab manager Roy Levin. "People get it wrong because parts of the program are not in sync or the synchronization is too coarse, leading to poor performance."
  EE Times (03/12/07)
- Unless a simple, inexpensive multicore chip programming technique is found, innovation within the market will dry up, warns MIT electrical engineer Saman Amarasinghe.
  New Scientist (03/10/07) Vol. 193, No. 2594, P. 26

MPI on Multicore Processors

- Work of Darius Buntinas and Guillaume Mercier
- 340 ns MPI ping/pong latency
- More room for improvement (but will require better software engineering tools)



What will multicore look like in 6 years?

- Already 8 cores in some chips (80 in demos); if double every 1.5 years, we'll have 128 cores in a commodity chip.
- Will these be cache coherent?
  - Maybe, but most likely by making accesses to other caches relatively expensive (or remote data uncacheable), or …
  - "When [a] request for data from Core 1 results in a L1 cache miss, the request is sent to the L2 cache. If this request hits a modified line in the L1 data cache of Core 2, certain internal conditions may cause incorrect data to be returned to the Core 1."
  - What do you think this is going to be like with 64 cores?   1024?
- How well will prefetch work in hiding memory latency with 128 concurrent yet different memory references?
- How do we adapt the software to this environment? This is the productivity problem…

Productivity Problems with MPI

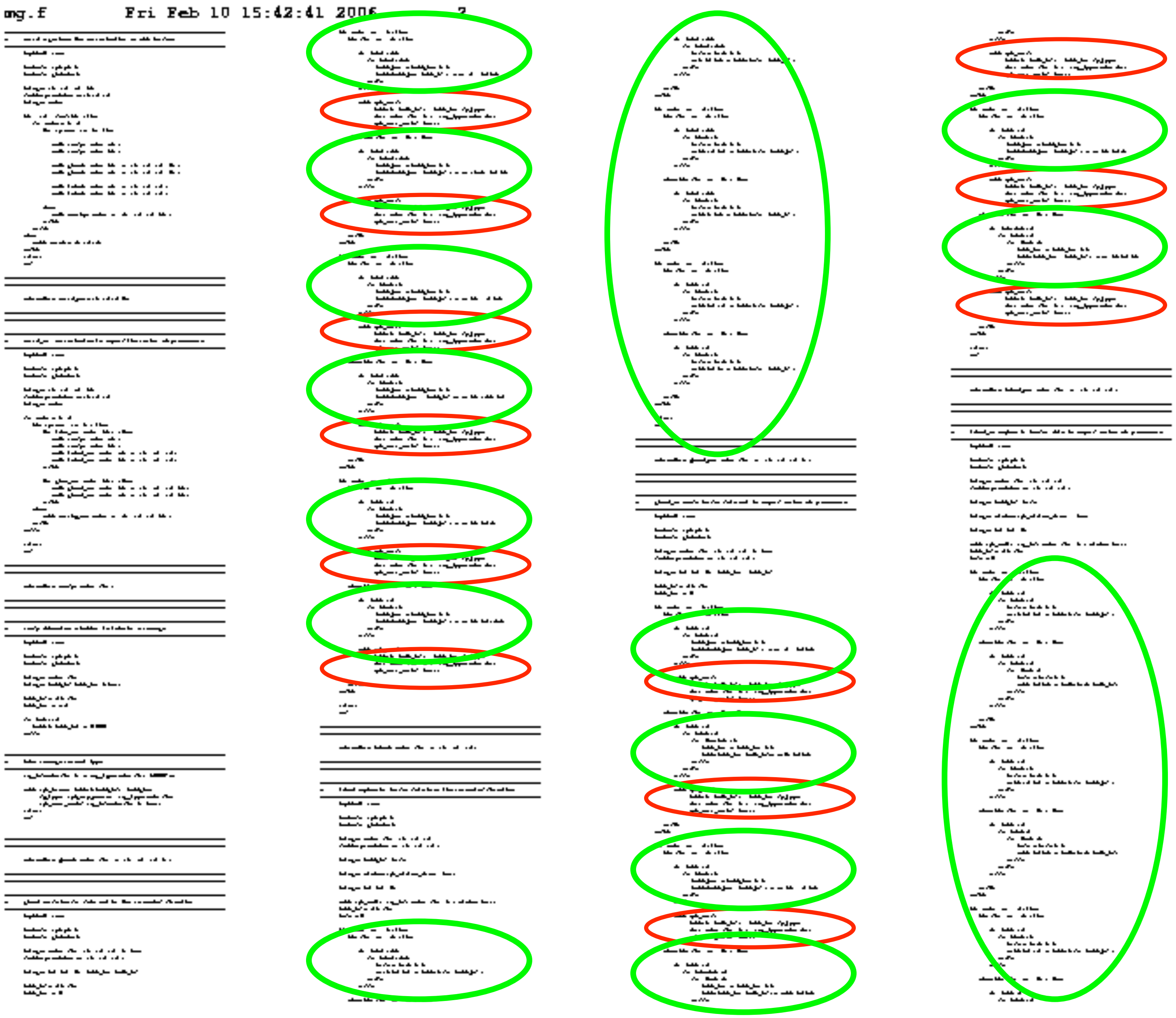
- No support for distributed data structures
  - Other than MPI_Type_create_darray and MPI_Type_create_subarray
    - *Examples of the difficulties of a library-based approach*
    - *(The Problem is that past matrix and simple grids, this gets very difficult to do efficiently and generally)*
- Action 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

What is Needed To Achieve Real High Productivity Programming

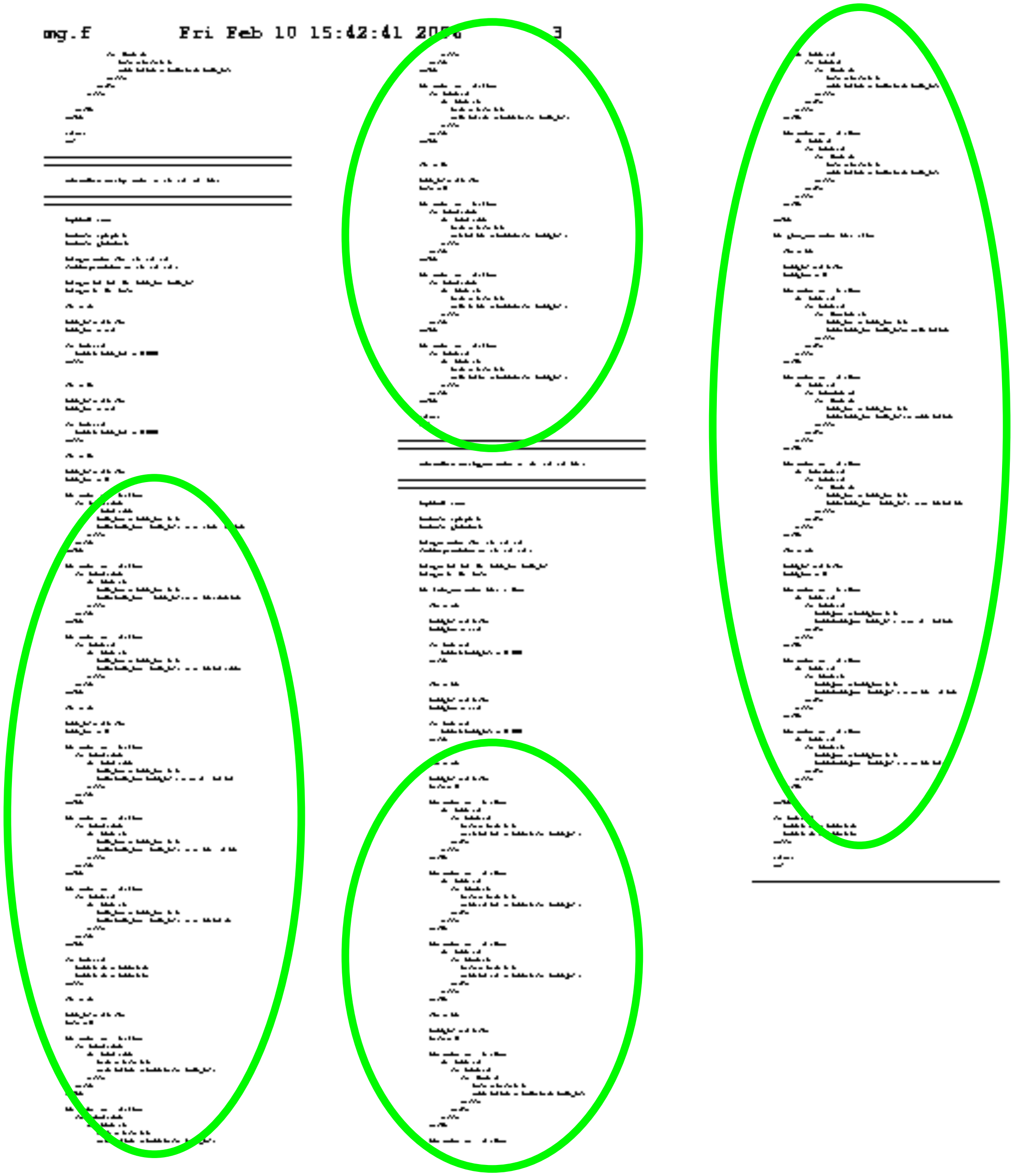
- Simplify the construction of correct, high-performance applications
- Managing Data Decompositions
  - Necessary for both parallel and uniprocessor applications
  - Many levels must be managed
  - Strong dependence on problem domain (e.g., halos, load-balanced decompositions, dynamic vs. static)
- Possible approaches include
  - 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
- Consider a simple distributed data structure …

Distributed Memory code

- Single node performance is clearly a problem.
- What about parallel performance?
  - Many successes at scale (e.g., Gordon Bell Prizes for >200TF on 64K BG nodes
  - Some difficulties with load-balancing, designing code and algorithms for latency, but skilled programmers and applications scientists have been remarkably successful
- Is there a problem?
  - There is the issue of productivity.
  - It isn’t just Message-passing vs shared memory
    - *Message passing codes can take longer to write but bugs are often deterministic (program hangs). Explicit memory locality simplifies fixing performance bugs*
    - *Shared memory codes can be written quickly but bugs due to races are difficult to find; performance bugs can be harder to identify and fix*
  - It isn’t just the way in which you move data
    - *Consider the NAS parallel benchmark code for Multigrid (mg.f):*



mg.f Fri Feb 10 15:42:41 2006 2



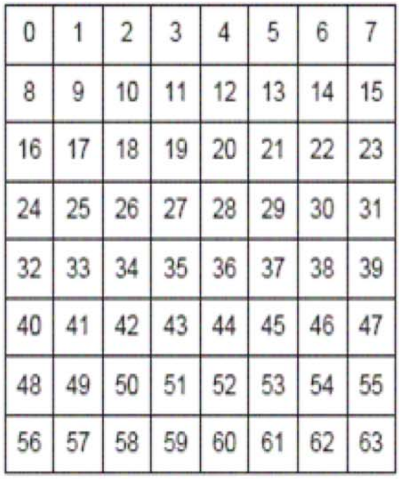
```
mg.f          Fri Feb 10 15:42:41 2006          3
```

What is the problem?
The user is responsible for all steps in the decomposition of the data structures across the processors

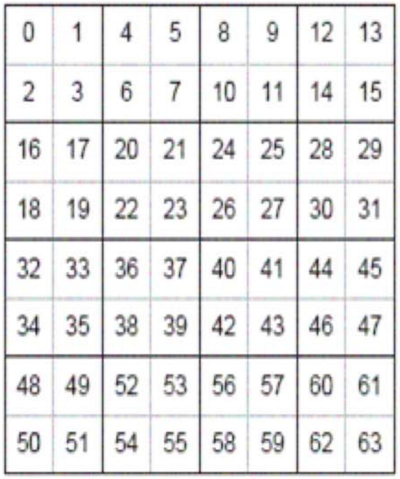
Note that this does give the user (or someone) a great deal of flexibility, as the data structure can be distributed in arbitrary ways across arbitrary sets of processors

Another example…

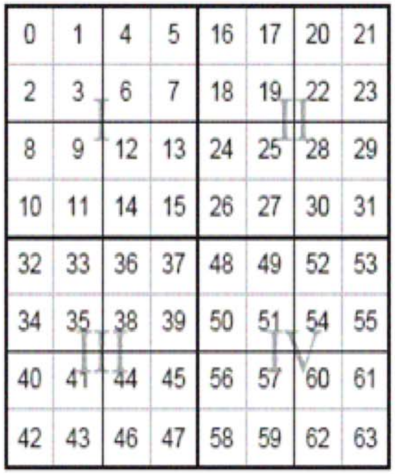
Manual Decomposition of Data Structures



0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63



0	1	4	5	8	9	12	13
2	3	6	7	10	11	14	15
16	17	20	21	24	25	28	29
18	19	22	23	26	27	30	31
32	33	36	37	40	41	44	45
34	35	38	39	42	43	46	47
48	49	52	53	56	57	60	61
50	51	54	55	58	59	62	63



0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	9	12	13	24	25	28	29
10	11	14	15	26	27	30	31
32	33	36	37	48	49	52	53
34	35	38	39	50	51	54	55
40	41	44	45	56	57	60	61
42	43	46	47	58	59	62	63

- Trick!
  - This is from a paper on dense matrix tiling for uniprocessors!
- This suggests that managing data decompositions is a common problem for real machines, whether they are parallel or not
  - *Not just an artifact of MPI-style programming*
  - Aiding programmers in data structure decomposition is an important part of solving the productivity puzzle

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

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) Interactions with Integrated Development Environments (IDE)*
  - Tools to create and manage MPI datatypes
  - Tools to create and manage distributed data structures
    - *A feature of the HPCS languages*

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**

How to Replace MPI

- Retain MPI’s strengths
  - Performance from matching programming model to the realities of underlying hardware
  - Ability to compose with other software (libraries, compilers, debuggers)
  - Determinism (without MPI_ANY_{TAG,SOURCE})
  - Run-everywhere portability
- Add to what MPI is missing, such as
  - Distributed data structures (not just a few popular ones)
  - Low overhead remote operations; better latency hiding/management; overlap with computation (not just latency; MPI can be implemented in a few hundred instructions, so overhead is roughly the same as remote memory reference (memory wall))
  - Dynamic load balancing for dynamic, distributed data structures
  - Unified method for treating multicores, remote processors, other resources
- Enable the transition from MPI programs
  - Build component-friendly solutions
    - *There is no one, true language*

Is MPI the Least Common Denominator Approach?

- “Least common denominator”
  - Not the correct term
  - It is “Greatest Common Denominator”! (Ask any Mathematician)
  - This is critical, because it changes the way you make improvements
- If it is “Least” then improvements can be made by picking a better approach. I.e., anything better than “the least”.
- If it is “Greatest” then improvements require changing the rules (either the “Denominator,” the scope (“Common”), or the goals (how “Greatest” is evaluated)
- Where can we change the rules for MPI?

Changing the Common

- Give up on ubiquity/portability and aim for a subset of architectures
  - Vector computing was an example (and a cautionary tale)
  - Possible niches include
    - *SMT for latency hiding*
    - *Reconfigurable computing; FPGA*
    - *Stream processors*
    - *GPUs*
    - *Etc.*
- Not necessarily a bad thing (if you are willing to accept being on the fringe)
  - Risk: Keeping up with the commodity curve (remember vectors)

Changing the Denominator

- This means changing the features that are assumed present in every system on which the programming model must run
- Some changes since MPI was designed:
  - RDMA Networks
    - *Best for bulk transfers*
    - *Evolution of these may provide useful signaling for shorter transfers*
  - Cache-coherent SMPs (more precisely, lack of many non-cache-coherent SMP nodes)
  - Exponentially increasing gap between memory and CPU performance
  - Better support for source-to-source transformation
    - *Enables practical language solutions*
- If DARPA HPCS is successful at changing the “base” HPC systems, we may also see
  - Remote load/store, remote simple ops
  - Hardware support for hiding memory latency

Changing the Goals

- Change the space of features
  - That is, change the problem definition so that there is room to expand (or contract) the meaning of “greatest”
- Some possibilities
  - Integrated support for concurrent activities
    - *Not threads:*
      - See, e.g., Edward A. Lee, "The Problem with Threads," Computer, vol. 39, no. 5, pp. 33-42, May, 2006.
      - “Night of the Living Threads”, http://weblogs.mozillazine.org/roc/archives/2005/12/night_of_the_living_threads.html, 2005
      - “Why Threads Are A Bad Idea (for most purposes)” John Ousterhout (~2004)
      - “If I were king: A proposal for fixing the Java programming language's threading problems” http://www-128.ibm.com/developerworks/library/j-king.html, 2000
  - Support for (specialized or general) distributed data structures

Issues for MPI in the Petascale Era

- Complement MPI with support for
  - Distributed (possibly dynamic) data structures
  - Improved node performance (including multicore)
    - *May include tighter integration, such as MPI+OpenMP with compiler and runtime awareness of both*
    - *Must be coupled with latency tolerant and memory hierarchy sensitive algorithms*
  - Fault tolerance
  - Load balancing
- Address the real memory wall - latency
  - Likely to need hardware support + programming models to handle memory consistency model
- MPI RMA model needs updating
  - To match locally cache-coherent hardware designs
  - Add better atomic remote op support
- Parallel I/O model needs more support
  - For optimal productivity of the computational scientist, data files should be processor-count independent (canonical form)

Conclusions

- MPI is a successful “Greatest Common Denominator” parallel programming model
- There is still much to do to improve implementations
  - Some parts of MPI’s design, particularly for RMA, may need adjustment to achieve the intended performance
- Adding new features to MPI is (mostly) not necessary:
  - 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

MPI 2.1

- The MPI 2.1 web page is available at http://www.mpi-forum.org/mpi2_1/index.htm . This page contains a link to the current errata discussion (please use this link through the forum page, as we may move this page in the future). You will find on this page many issues that have been raised and are still open, as well as a draft ballot for the third round of errata items.
- All items are open for discussion and new items may be submitted to mpi-21@mpi-forum.org or to mpi-comments@mpi-forum.org .
- Discussions of the technical issues can start at any time (in fact, the page of issues has been available since before the original errata discussions). Send mail about these issues to mpi-21@mpi-forum.org .