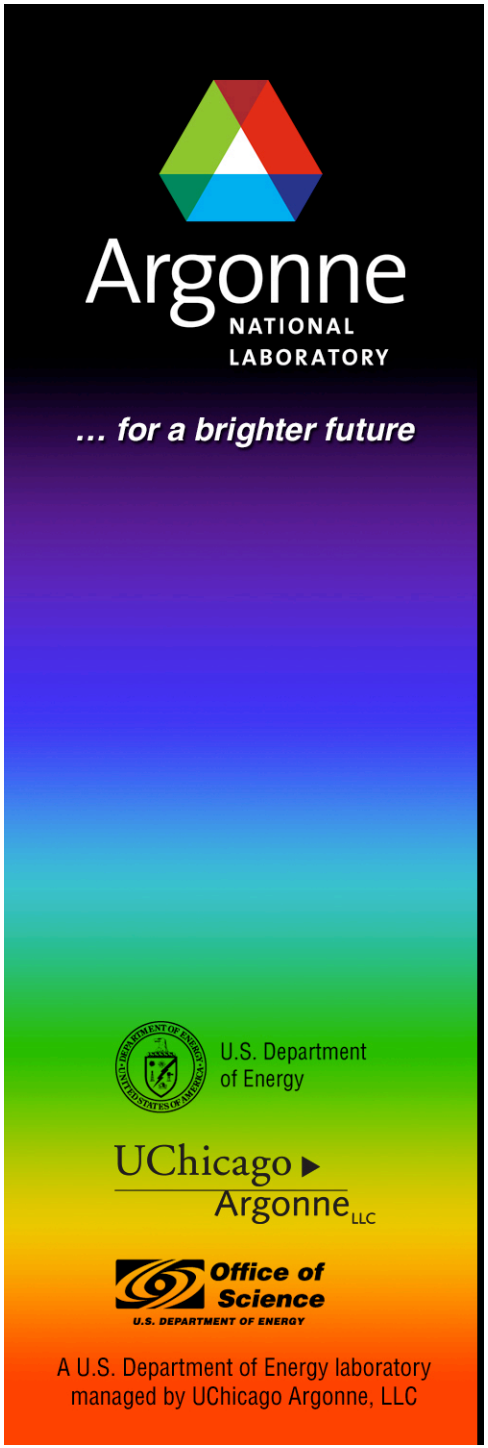


MPI on a Hundred Million Processors…

Why Not?

Rusty Lusk

Mathematics and Computer Science Division

Argonne National Laboratory

Conventional Panic

Aieeeee!

A million processors! Petascale! Exascale! Multicore! MANYcore! We are all DOOMed!

Some have concerns about MPI…

Outline

- Conventional Wisdom about MPI and scalability
- Some non-issues
  - It's might be the algorithm, not the model
- Some real issues
  - MPI specification
  - MPI implementation
  - MPI scope
- Addressing scalability and productivity simultaneously
  - An example library
  - A little fun
  - A serious application
- MPI going forward
  - The MPI Forum
  - MPI Research

Conventional Wisdom

- "MPI will never scale to 100,000 processors"
- Too late.

- "We need new programming models for extreme scale."
- Let's see them. Working. At scale.

- I don't know what we'll be using on Exascale computers, but it certainly won't be MPI."
- I don't know either, but it *will* be MPI.

- "MPI is the assembly language of parallel programming."
- No, it isn't.

- "MPI has failed. What now?" - Patrick Geoffray
- MPI has succeeded. Now what?

What Do Such Comments Really Mean?

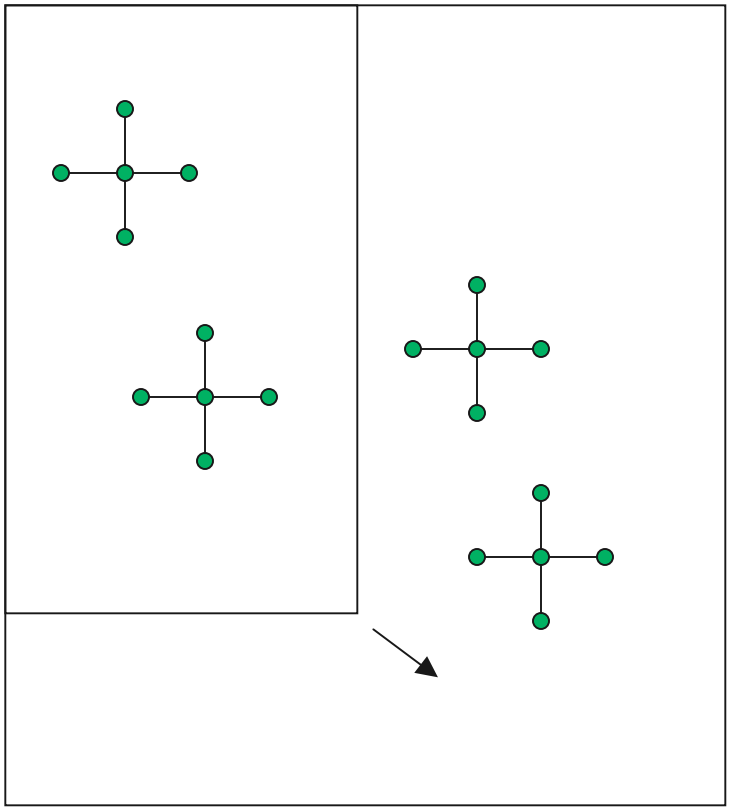
- That scalable algorithms can't be expressed in MPI? (The API itself isn't scalable?)
- That the specification precludes scalable implementation?
- That it won't be capable of extracting performance from hierarchical architectures?
- That it impedes programmer productivity?
  - (For whom? Application writers? Library writers? System programmers? Workflow authors? Webmasters?)

These issues deserve more consideration than they have received.

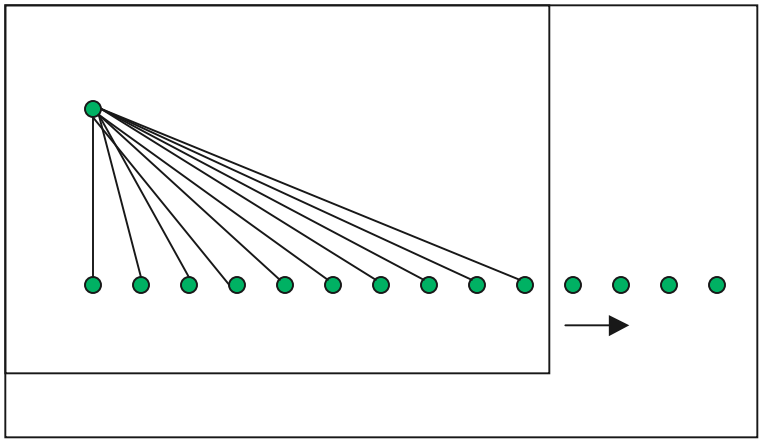
First, Worry About Algorithm Scalability

- Some algorithms obviously scale; others obviously don't. For others, it is not obvious.

Scales



Not



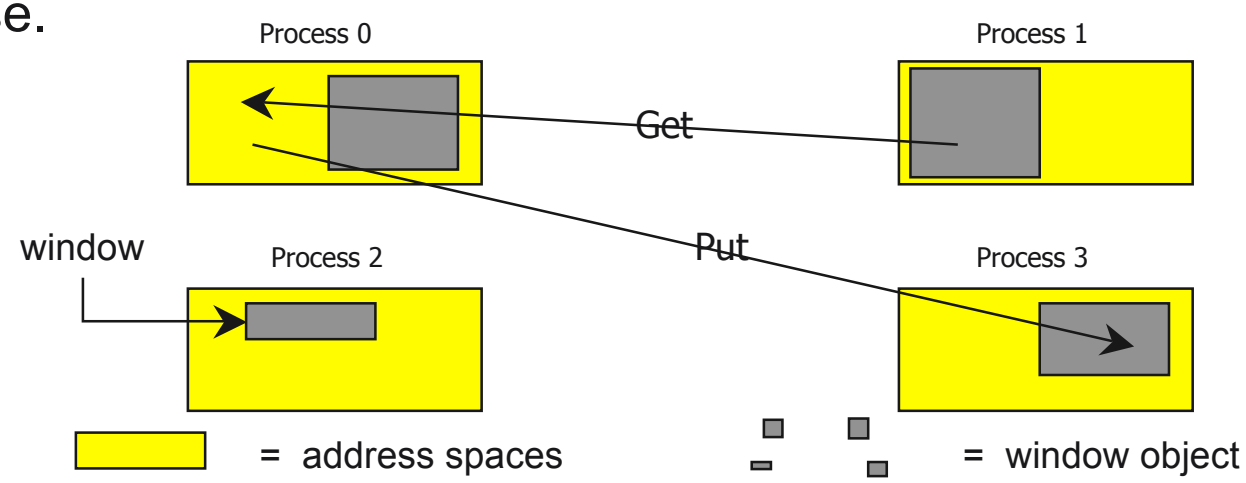
- (Actually, we might be able to scale this algorithm by changing the implementation.
  - More on this later)

Scalability Issues in the MPI Specification

- MPI_Comm_rank returns an int. On most machines, this limits applications to fewer than 2,000,000,000 MPI processes or so.
- Several functions have parameters that scale linearly with the number of processes in the communicator.
  - MPI_Group_create
  - MPI_Graph_create
  - Collective “v” functions, e.g. MPI_Gatherv
    - *Even for paths where no data is actually transferred*
- The topology routines look like they might help, but don’t promise to do what you expect.
- (The MPI Forum is currently working on these issues.)

Scalability Issues in the MPI Implementation

- Need scalable internal data structures
  - E.g. connection table
- And scalable behavior
  - E.g. lazy connection setup and table entry reuse.
- Collective operations
  - Fertile area for research -- many variations
- Difficult to implement MPI_Win creation scalably in non-symmetric case.



MPI Processes

- Most MPI implementations use “Unix” processes for MPI processes, but --
- This is not part of the standard.
  - A lighter weight object could be used, with a little help from the OS and compiler.
  - The NEC SX machines use threads for MPI processes.
- But you might want heavier instead of lighter
  - Some algorithms prefer a smaller number of processes
  - An MPI process could span several nodes (hybrid models with PGAS languages, below)

Scalability Issues in the Scope of the MPI Specification

- Topology routines potentially useful, but need more depth (in the hierarchy sense).
- Fault Tolerance
  - MPI already has features that support the writing of fault-tolerant applications.
    - *Communicators can be used to isolate faults*
    - *Custom error handlers can be used to react to them*
  - Some new objects (like variable-sized process groups) might be useful
    - *MPI Forum currently looking at this.*
- Virtualization
  - Some work done (Charm++/AMPI); more needed

More on the Scope of MPI

- MPI is for moving data between *address spaces*.
  - If you have only one address space, then you don't need MPI (although it might help you manage locality).
    - *But if the parallelism in your program is complex then you will be in debug mode forever, and then begins tuning.*
- The ability to have multiple cores working in the same address space while MPI handles the inter-address-space communication has led to hybrid models.
- Current hybrid models (MPI + OpenMP or pthreads) leave the parallelism within an address space to another programming model.
  - This works because the thread safety specification in MPI permits precise negotiation between the MPI implementation and the application code on the level of thread safety required/provided.
- Alternate hybrid models (think MPI communication among UPC global address spaces, possibly spanning multiple nodes) require similar standardization on mutual commitments.

So What Should We Do?

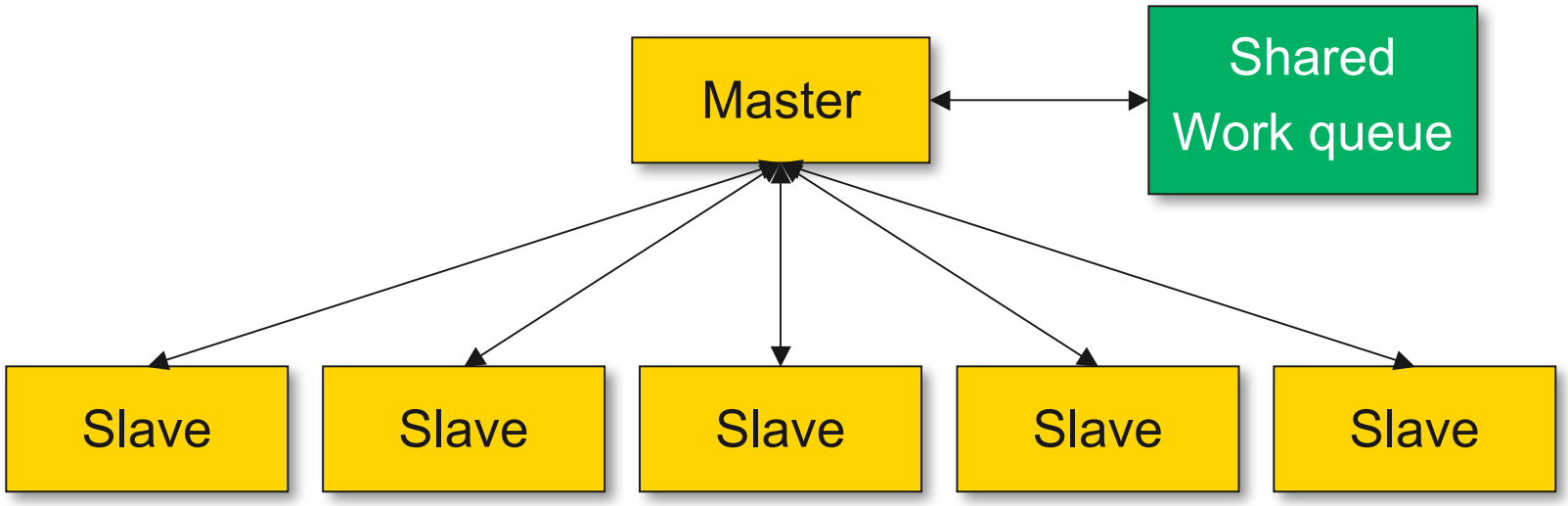
- Continue research into scalable algorithms at the mathematics and application level.
- Continue research into MPI implementation techniques that scale.
- Look for opportunities to improve scalability at the MPI specification level.
- Create libraries, preprocessors, tools, even compilers; MPI will be there to help.

- Especially libraries!
  - Libraries can address scalability and application programmer productivity simultaneously.

An Example Application Whose Scalability Has Been Enabled by a Library

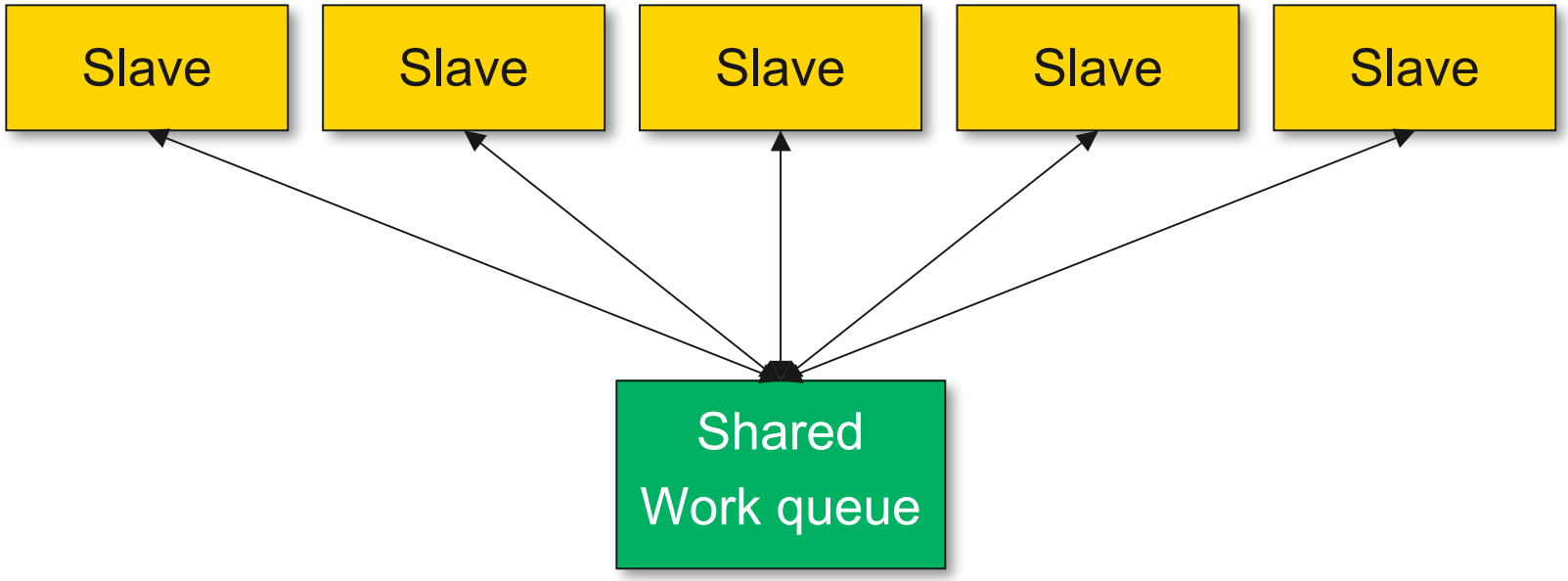
- Green's Function Monte Carlo -- the "gold standard" for *ab initio* calculations in nuclear physics (Steve Pieper at Argonne)
- A non-trivial master/slave algorithm, with assorted work types and priorities; multiple processes create work; large work units
- Has scaled to 2000 processors on BG/L a little over a year ago, then hit scalability wall.
- Need to get to 10's of thousands of processors at least, in order to carry out calculations on ^{12}C, an explicit goal of the UNEDF SciDAC project.
- The algorithm has had to become more complex, with more types and dependencies among work units, together with smaller work units
- Overall master/slave structure to be maintained
- This situation called for the invention of a new library -- ADLB, the Asynchronous Dynamic Load Balancing Library.

Master/Slave Algorithms and Load Balancing



- Advantages
  - Automatic load balancing
- Disadvantages
  - Scalability - master can become bottleneck
- Wrinkles
  - Slaves may create new work
  - Multiple work types and priorities that impose ordering

The ADLB Model (no master)

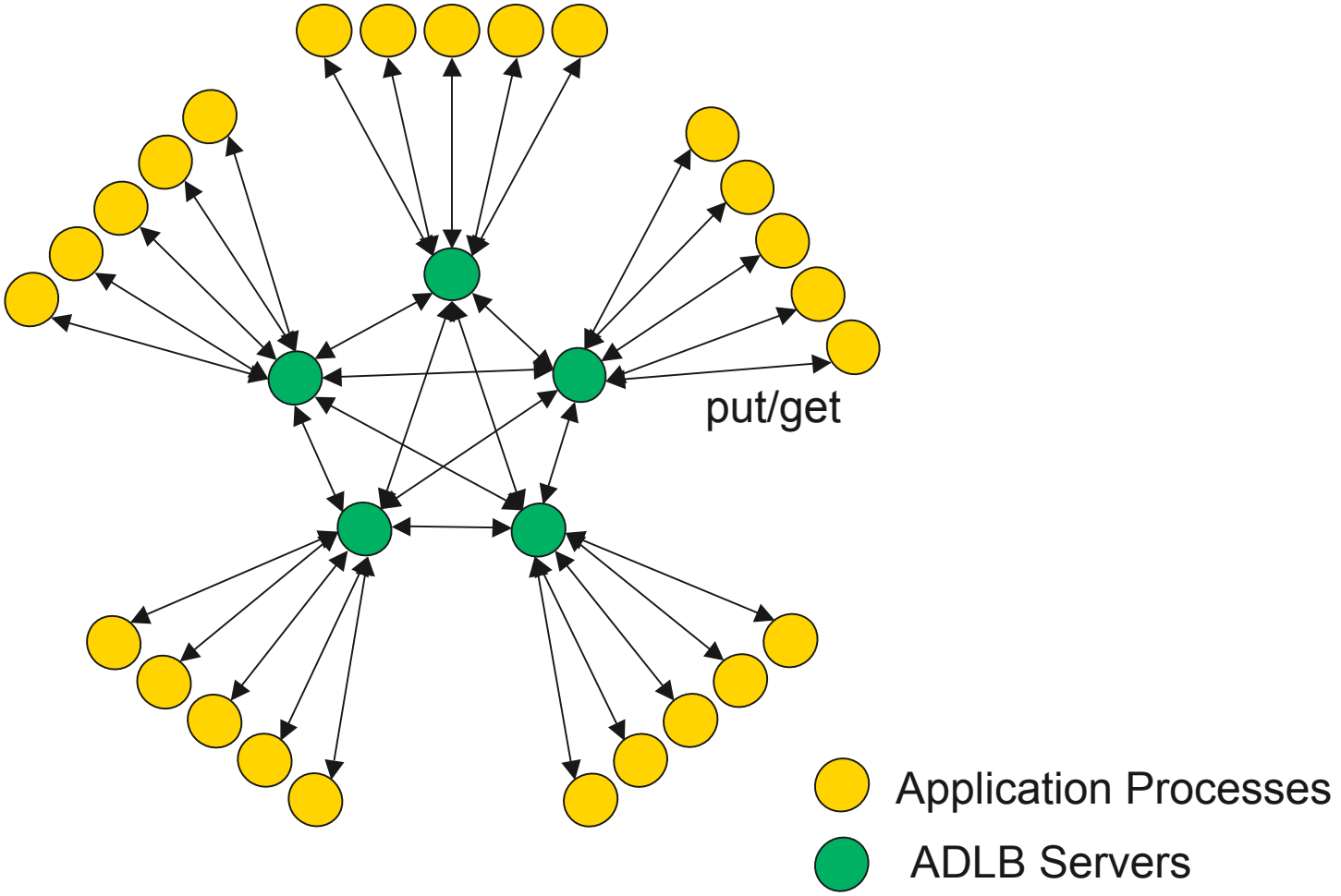


- Doesn't really change algorithms in slaves
- But need distributed implementation of shared work queue for scalability

The API

- Basic calls
  - ADLB_Init(num_servers, am_server, app_comm)
  - ADLB_Server()
  - ADLB_Put(type, priority, len, buf, answer_dest)
  - ADLB_Reserve(req_types, handle, len, type, prio, answer_dest)
  - ADLB_Ireserve(…)
  - ADLB_Get_Reserved(handle, buffer)
  - ADLB_Set_Done()
  - ADLB_Finalize()
- A few others, for tuning and debugging
  - ADLB_{Begin,End}_Batch_Put()
  - Getting performance statistics with ADLB_Get_info(key)

Behind the Scenes



- To warm up, we look at the next DOE Grand Challenge app…

Sudoku!

1	2				9			7
		3				6	1	
				7		8		
					5	3		
7		9	1		8	2		6
		5	6					
		1		9				
	6	7				1		
2			5				3	8

Parallel Sudoku Solver with ADLB

1	2				9			7
		3				6	1	
				7		8		
					5	3		
7		9	1		8	2		6
		5	6					
		1		9				
	6	7				1		
2			5				3	8

Work unit =

partially completed “board”

Program:

```
if (rank = 0)
    ADLB_Put initial board
ADLB_Get board
while success  (else done)
    ooh
    find first blank square
    if failure  (problem solved!)
        print solution
        ADLB_Set_Done
    else
        for each valid value
            set blank square to value
            ADLB_Put new board
        ADLB_Get board
end while
```

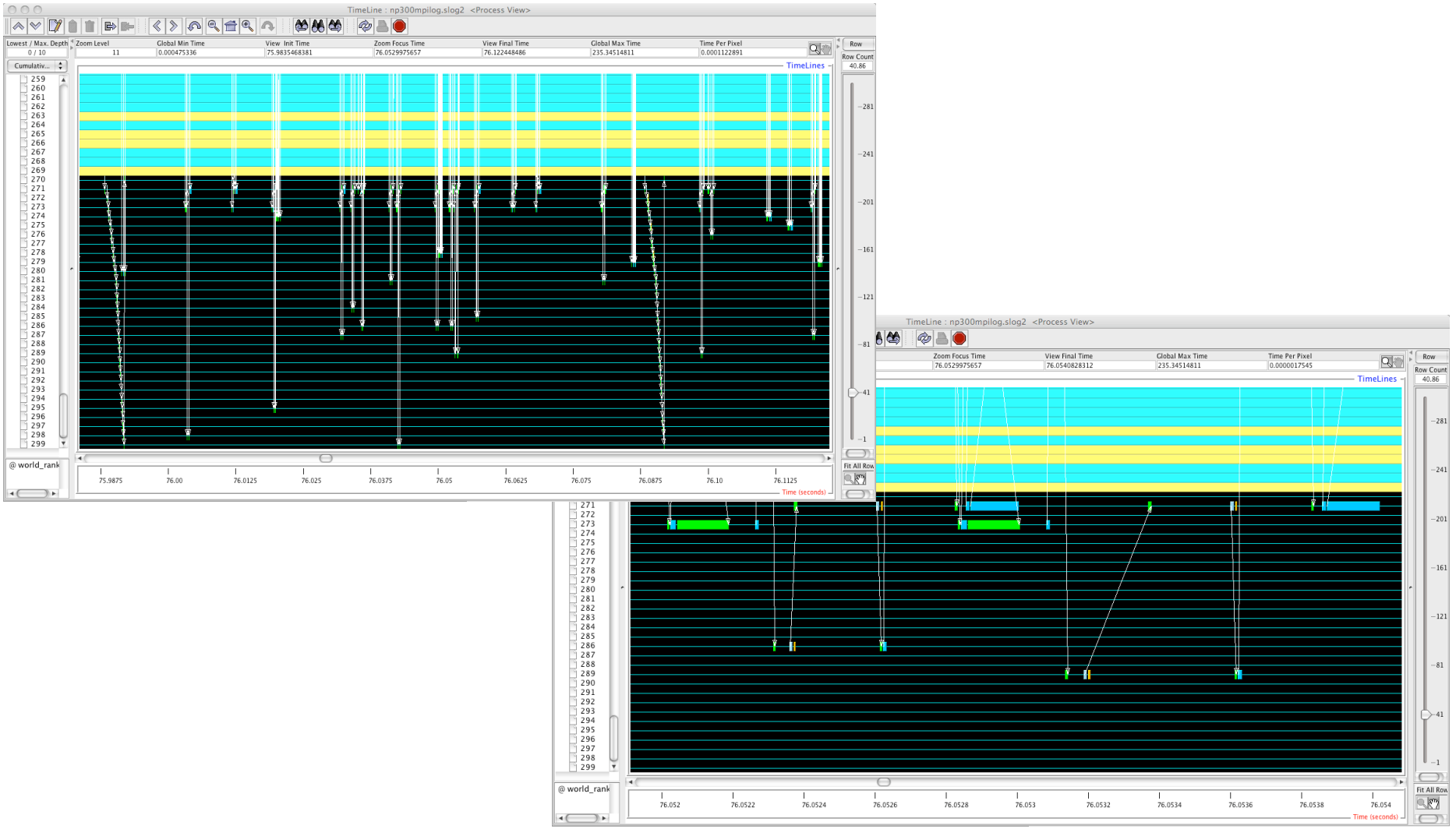
Optimizing Within the ADLB Framework

- Can embed smarter strategies in this algorithm
  - ooh = optional optimization here
  - Even so, potentially a *lot* of work packages for ADLB to manage
- Can use priorities to address this problem
  - On ADLB_Put, set priority to the number of filled squares
  - This will guide depth-first search while ensuring that there is enough work to go around
    - *How one would do it sequentially*
- Exhaustion automatically detected by ADLB (e.g., case of invalid input board)

ADLB Uses MPI Features

- ADLB_Init returns separate application communicator, so application can use MPI for its own purposes if it needs to.
- Servers are in MPI_Iprobe loop for responsiveness.
- MPI_Datatypes for some complex, structured messages (status)
- Servers use nonblocking sends and receives, maintain queue of active MPI_Request objects.
- Queue is traversed and each request kicked with MPI_Test each time through loop; could use MPI_Test_any.
- Client side uses MPI_Ssend to implement ADLB_Put in order to conserve memory on servers, MPI_Send for other actions.
- Servers respond to requests with MPI_Rsend since MPI_Irecvs are known to be posted by clients before requests.
- MPI provides portability: laptop, Linux cluster, SiCortex, BG/P
- MPI profiling library is used to understand application-ADLB behavior.

Looking at GFMC/ADLB with Jumpshot



Recent Experiments with GFMC/ADLB on BG/P

- Using GFMC to compute the binding energy of 14 neutrons in an artificial well (“neutron drop” = teeny-weeny neutron star)
- Weak scaling

BG/P cores	ADLB Servers	Configs	Time (min.)	Efficiency (incl. serv.)
4K	130	20	38.1	93.8%
8K	230	40	38.2	93.7%
16K	455	80	39.6	89.8%
32K	905	160	44.2	80.4%

- Next steps: “micro-parallelization” needed for ^{12}C.

MPI Forum Activities

- MPI 1.0 - 1995
  - Basic communication with buffer management
  - Collective operations
  - New “high-level” ideas: communicators, derived datatypes, …
- MPI 2.0 - 1997
  - Dynamic process management
  - Parallel I/O
  - Remote memory access
- Adopted minor fixes (errata) for several years
- MPI 2.1 - September 2008
  - Merge both official documents and errata
- MPI 2.2 - in progress
  - Additions and clarification that don’t require application changes
    - *E.g. a function to send very long messages (length not an int)*
- MPI 3.0 - simultaneously in progress
  - Could require some changes to applications

Some Possible MPI-3 Areas

- New signatures for old functions
  - E.g. MPI_Send(…,MPI_Count,…)
- Details
  - Fortran binding issues..
- New features
  - MPI_Process_Group and related functions for fault tolerance
  - New topology routines aware of more hierarchy levels
  - Non-blocking collective operations
  - A simpler one-sided communication interface
  - More scalable versions of the “v” collectives
  - …
- See http://www.mpi-forum.org for details of working groups

MPI Research

- The Annual EuroPVM/MPI Conference is the major outlet for computer science research related to MPI.
- A random sample from EuroPVM/MPI #15 (2008) in Dublin
  - "Toward efficient support for multi-threaded MPI communication"
  - "Sparse non-blocking collectives in quantum mechanical calculations"
  - "Self consistent MPI-I/O performance requirements and expectations
  - "Architecture of the component collective message-passing interface"
  - "A simple pipelined algorithm for large irregular allgather problems"
  - "MPI support for multicore architectures: optimizing shared-memory collectives"
  - "Performance issues of synchronization in the MPI-2 one-sided communication API"
  - "Implementing efficient dynamic formal verification methods for MPI programs"

Conclusions

- MPI will not only be possible on 100 million processors; it will be *necessary*.
- Libraries can be the key to scalability and productivity both.
- Vigorous research activities and MPI Forum-guided evolution of the standard will keep MPI fresh.
- Don’t panic!

The End

MPI Codes running on > 130,000 cores on Endeavor (40-rack IBM BG/P at Argonne)

- FLASH: astrophysics/hydrodynamics
- MILC: Quantum Chromodynamics
- CPS: QCD
- Chroma: QCD
- NEK: fluid dynamics
- GTC: fusion plasma
- DOCK5+DOCK6
- QBOX: 1st principles MD
- MGDC: quantum MD
- RXFF: semi-classical MD
- GMD: classical MD
- DNS3D: turbulence
- HYPO4D: lattice Boltzmann
- PLB: lattice Boltzmann
- LAMMPS: molecular dynamics
- CACTUS: problem-solving environment