Adventures in Load Balancing at Scale: Successes, Fizzles, and Next Steps

Rusty Lusk

Mathematics and Computer Science Division
Argonne National Laboratory

Outline

- Introduction
  - Two abstract programming models
  - Load balancing and master/slave algorithms
  - A collaboration on modeling small nuclei
- The Asynchronous, Dynamic, Load-Balancing Library (ADLB)
  - The model
  - The API
  - An implementation
- Results
  - Serious – GFMC: complex Monte Carlo physics application
  - Fun – Sudoku solver
  - Parallel programming for beginners: Parameter sweeps
  - Useful – batcher: running independent jobs
- An interesting alternate implementation that scales less well
- Future directions
  - for the API
  - yet another implementation

Two Classes of Parallel Programming Models

- Data Parallelism
  - Parallelism arises from the fact that physics is largely local
  - Same operations carried out on different data representing different patches of space
  - Communication usually necessary between patches (local)
    - global (collective) communication sometimes also needed
  - Load balancing sometimes needed
- Task Parallelism
  - Work to be done consists of largely independent tasks, perhaps not all of the same type
  - Little or no communication between tasks
  - Traditionally needs a separate "master" task for scheduling
  - Load balancing fundamental

Load Balancing

- Definition: the assignment (scheduling) of tasks (code + data) to processes so as to minimize the total idle times of processes
- Static load balancing
  - all tasks are known in advance and pre-assigned to processes
  - works well if all tasks take the same amount of time
  - requires no coordination process
- Dynamic load balancing
  - tasks are assigned to processes by coordinating process when processes become available
  - Requires communication between manager and worker processes
  - Tasks may create additional tasks
  - Tasks may be quite different from one another

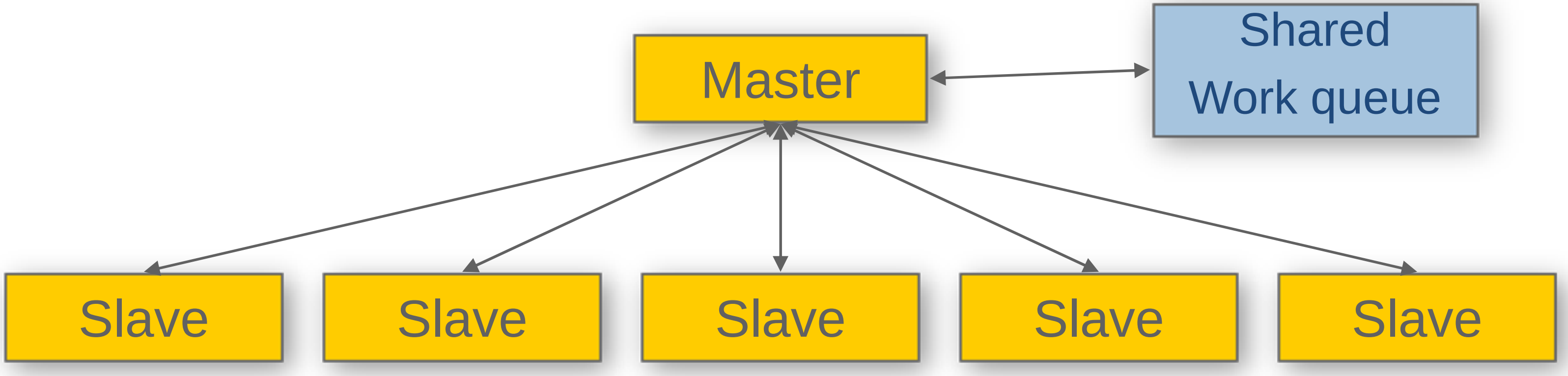
Green's Function Monte Carlo – A Complex Application

- Green's Function Monte Carlo -- the "gold standard" for *ab initio* calculations in nuclear physics at Argonne (Steve Pieper, PHY)
- A non-trivial master/slave algorithm, with assorted work types and priorities; multiple processes create work dynamically; large work units
- Had scaled to 2000 processors on BG/L a little over four years 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 threatened to become even more complex, with more types and dependencies among work units, together with smaller work units
- Wanted to maintain master/slave structure of physics code
- This situation brought forth ADLB
- Achieving scalability has been a multi-step process
  - balancing processing
  - balancing memory
  - balancing communication

The Plan

- Design a library that would:
  - allow GFMC to retain its basic master/slave structure
  - eliminate visibility of MPI in the application, thus simplifying the programming model
  - scale to the largest machines

Generic Master/Slave Algorithm

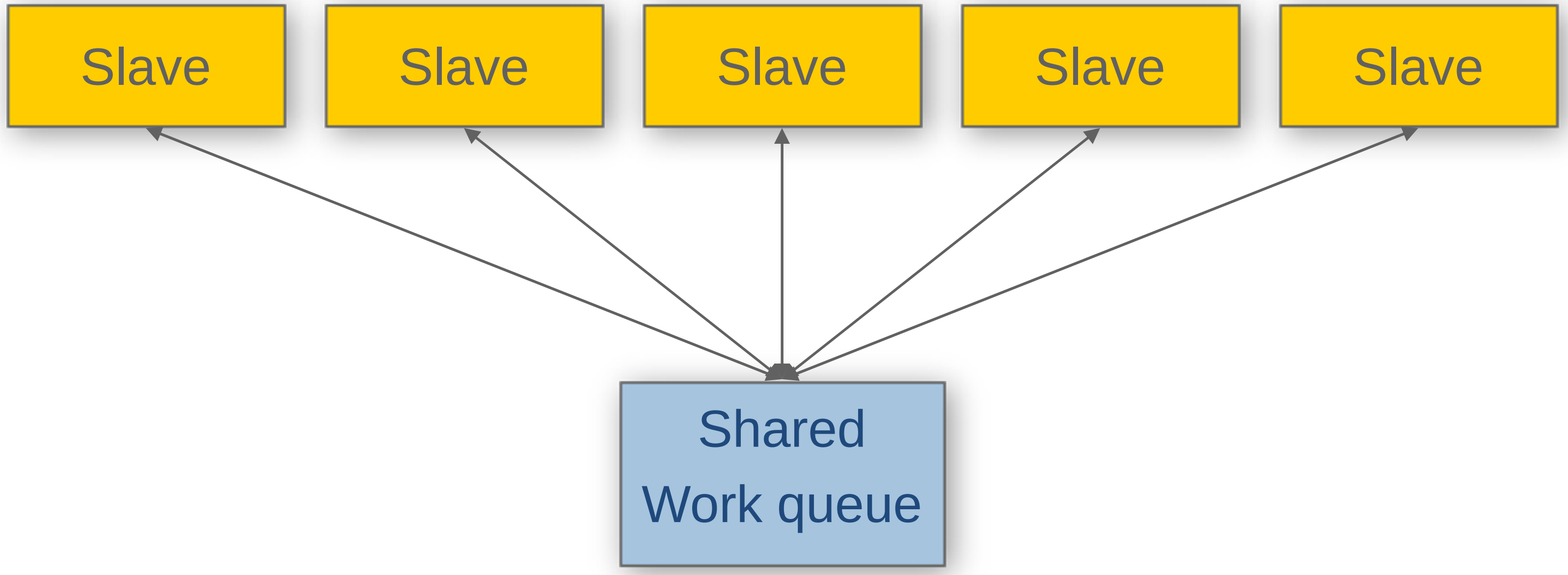


- Easily implemented in MPI
- Solves some problems
  - implements dynamic load balancing
  - termination
  - dynamic task creation
  - can implement workflow structure of tasks
- Scalability problems
  - Master can become a communication bottleneck (granularity dependent)
  - Memory can become a bottleneck (depends on task description size)

The ADLB Vision

- No explicit master for load balancing; slaves make calls to ADLB library; those subroutines access local and remote data structures (remote ones via MPI).
- Simple Put/Get interface from application code to distributed work queue hides MPI calls
  - Advantage: multiple applications may benefit
  - Wrinkle: variable-size work units, in Fortran, introduce some complexity in memory management
- Proactive load balancing in background
  - Advantage: application never delayed by search for work from other slaves
  - Wrinkle: scalable work-stealing algorithms not obvious

The ADLB Model (no master)



- Doesn’t really change algorithms in slaves
- Not a new idea (e.g. Linda)
- But need scalable, portable, distributed implementation of shared work queue
  - MPI complexity hidden here

API for a Simple Programming Model

- Basic calls
  - ADLB_Init(num_servers, am_server, app_comm)
  - ADLB_Server()
  - ADLB_Put(type, priority, len, buf, target_rank, 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)

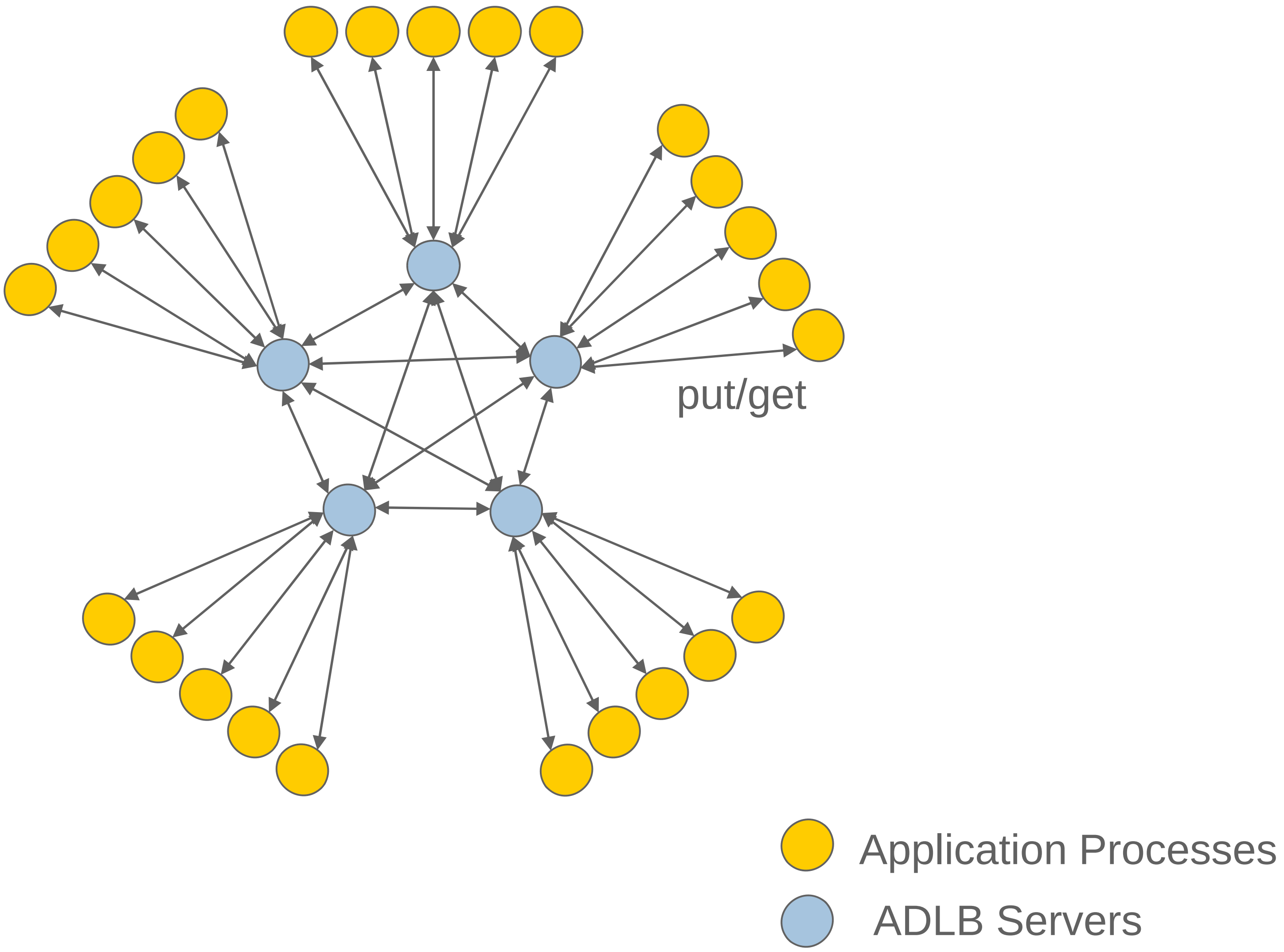
API Notes

- Return codes (defined constants)
  - ADLB_SUCCESS
  - ADLB_NO_MORE_WORK
  - ADLB_DONE_BY_EXHAUSTION
  - ADLB_NO_CURRENT_WORK (for ADLB_Ireserve)
- Batch puts are for inserting work units that share a large proportion of their data
- Types, answer_rank, target_rank can be used to implement some common patterns
  - Sending a message
  - Decomposing a task into subtasks
  - Maybe should be built into API

More API Notes

- If some parameters are allowed to default, this becomes a simple, high-level, work-stealing API
  - examples follow
- Use of the “fancy” parameters on Puts and Reserve-Gets allows variations that allow more elaborate patterns to be constructed
- This allows ADLB to be used as a low-level execution engine for higher-level models
  - API’s being considered as part of other projects

How It Works



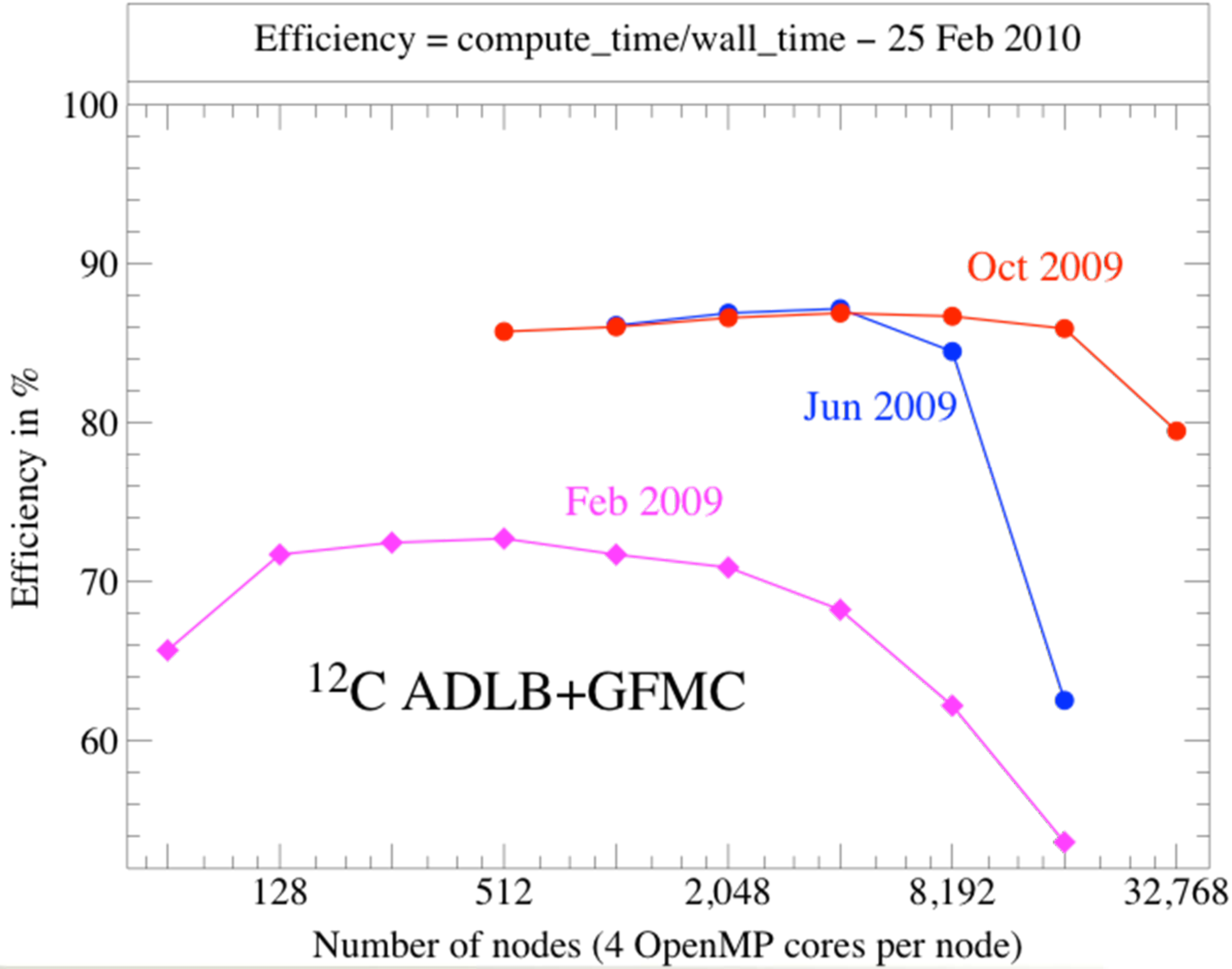
Early Experiments with GFMC/ADLB on BG/P

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

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%

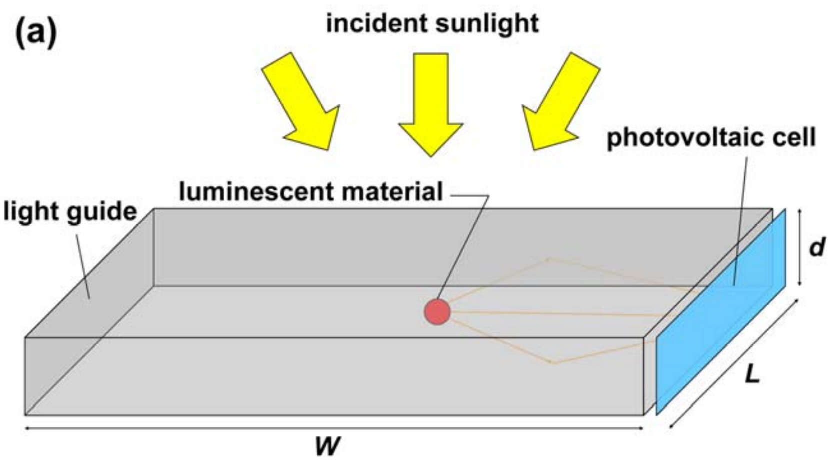
- Recent work: “micro-parallelization” needed for ^{12}C, OpenMP in GFMC.
  - a successful example of hybrid programming, with ADLB + MPI + OpenMP

Progress with GFMC



Another Physics Application – Parameter Sweep

- Luminescent solar concentrators
  - Stationary, no moving parts
  - Operate efficiently under diffuse light conditions (northern climates)
- Inexpensive collector, concentrate light on high-performance solar cell
- In this case, the authors never learned any parallel programming approach before ADLB



The “Batcher”

- Simple but potentially useful
- Input is a file of Unix command lines
- ADLB worker processes execute each one with the Unix “system” call

A Tutorial Example: 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

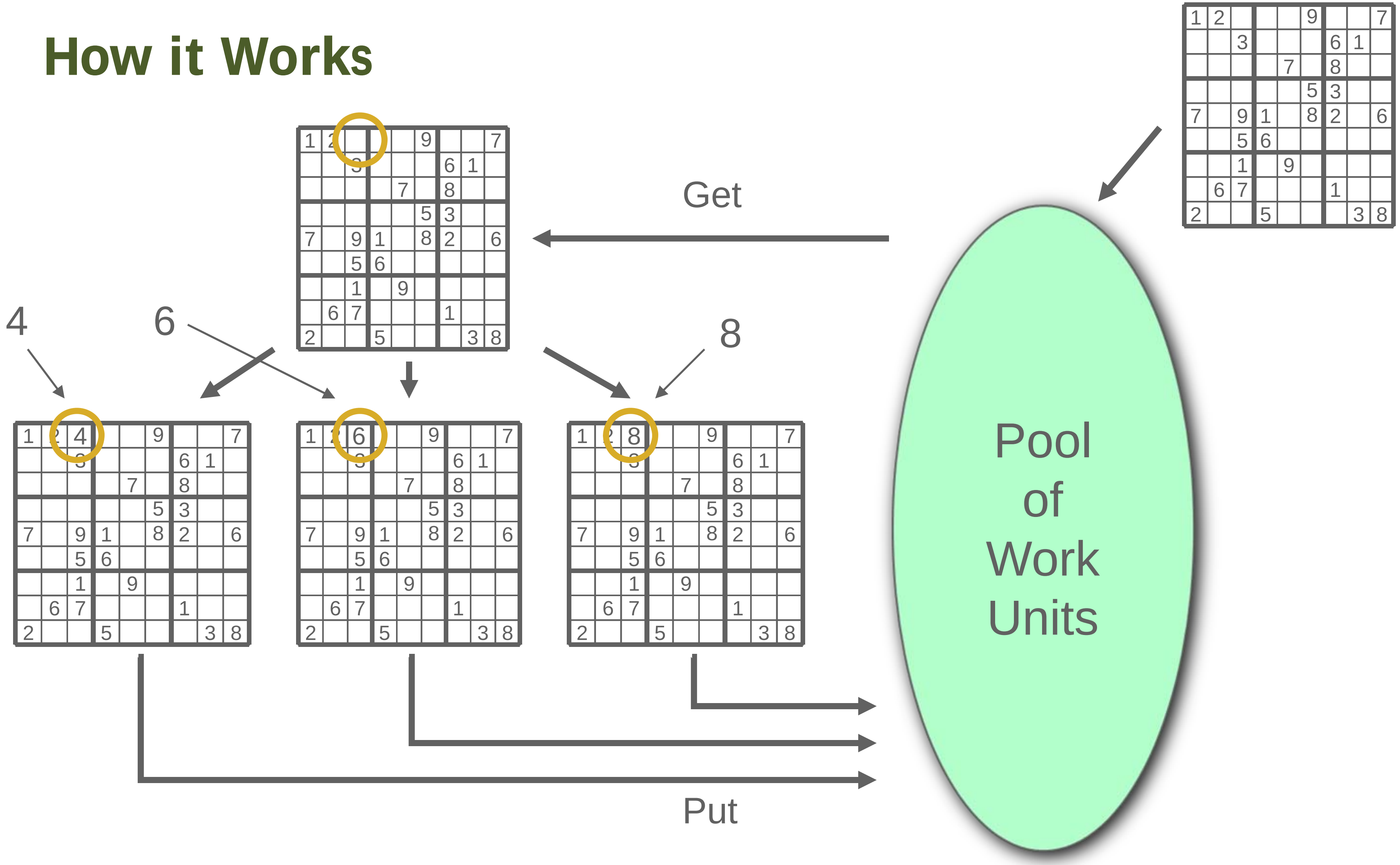
Work unit =

partially completed "board"

Program:

```
    if (rank = 0)
        ADLB_Put initial board
    ADLB_Get board (Reserve+Get)
    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
```

How it Works



- After initial Put, all processes execute same loop (no master)

Optimizing Within the ADLB Framework

- Can embed smarter strategies in this algorithm
  - ooh = "optional optimization here", to fill in more squares
  - Even so, potentially a *lot* of work units 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., proof that there is only one solution, or the case of an invalid input board)

The ADLB Server Logic

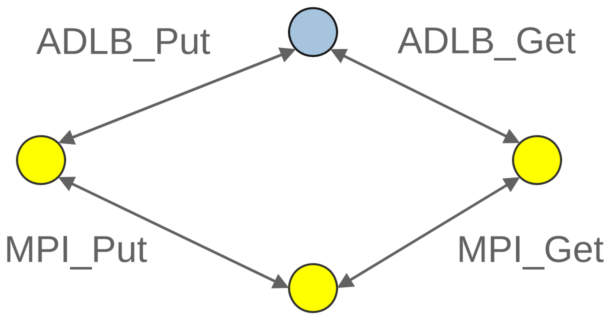
- Main loop:
  - MPI_Iprobe for message in busy loop
  - MPI_Recv message
  - Process according to type
    - Update status vector of work stored on remote servers
    - Manage work queue and request queue
    - (may involve posting MPI_Isends to isend queue)
  - MPI_Test all requests in isend queue
  - Return to top of loop
- The status vector replaces single master or shared memory
  - Circulates every .1 second at high priority
  - Multiple ways to achieve priority

ADLB Uses Multiple 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_Testany. No MPI_Wait.
- 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.

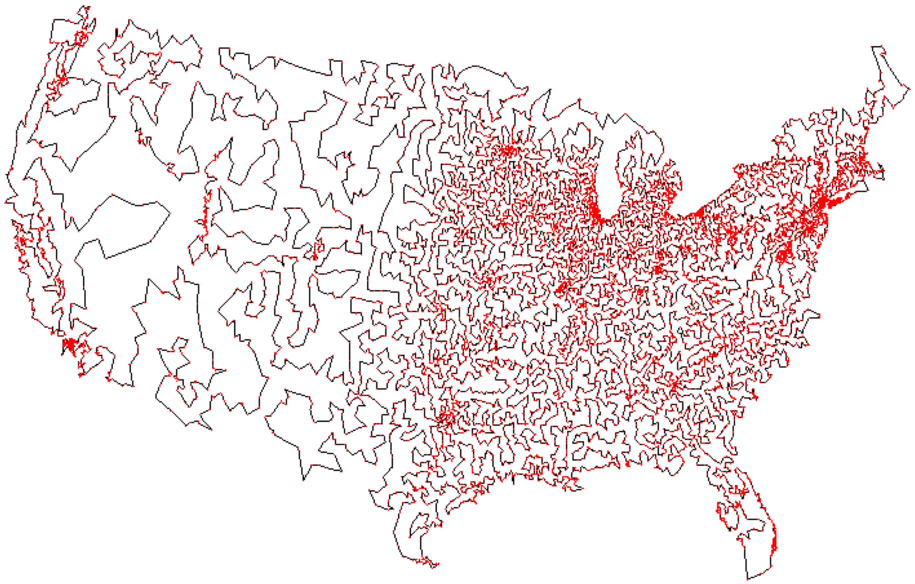
An Alternate Implementation of the Same API

- Motivation for 1-sided, single-server version
  - Eliminate multiple views of "shared" queue data structure and the effort required to keep them (almost) coherent)
  - Free up more processors for application calculations by eliminating most servers.
  - Use larger client memory to store work packages
- Relies on "passive target" MPI-2 remote memory operations
- Single master proved to be a scalability bottleneck at 32,000 processors (8K nodes on BG/P) not because of processing capability but because of network congestion.
- Have not yet experimented with hybrid version



Getting ADLB

- Web site is http://www.cs.mtsu.edu/~rbutler/adlb
- To download adlb:
  - svn co http://svn.cs.mtsu.edu/svn/adlbm/trunk adlbm
- What you get:
  - source code (both versions)
  - configure script and Makefile
  - README, with API documentation
  - Examples
    - Sudoku
    - Batcher
      - Batcher README
    - Traveling Salesman Problem



- To run your application
  - configure, make to build ADLB library
  - Compile your application with mpicc, use Makefile as example
  - Run with mpiexec
- Problems/complaints/kudos to {lusk,rbutler}@mcs.anl.gov

Future Directions

- API design
  - Some higher-level function calls might be useful
  - User community will generate these
- Implementations
  - The one-sided version
    - implemented
    - single server to coordinate matching of requests to work units
    - stores work units on client processes
    - Uses MPI_Put/Get (passive target) to move work
    - Hit scalability wall for GFMC at about 8000 processes
  - The thread version
    - uses separate thread on each client; no servers
    - the original plan
    - maybe for BG/Q, where there are more threads per node
    - not re-implemented (yet)

Conclusions

- The Philosophical Accomplishment: Scalability need not come at the expense of complexity

- The Practical Accomplishment: Multiple uses
  - As high-level library to make simple applications scalable
  - As execution engine for
    - complicated applications (like GFMC)
    - higher-level “many-task” programming models

The End