FAS
INFORMATION TECHNOLOGY

Writing MPI Programs for Odyssey

Teresa Kaltz, PhD
Research Computing

FAS IT Research Computing

- Provide compute resources to FAS and SEAS for research purposes
- Leverage FAS IT infrastructure
- Architect and manage RC resources
- Support both shared and dedicated hardware
- Also have expertise on staff
  - Domain level
  - Programming

FAS INFORMATION TECHNOLOGY

What is Odyssey?

- Generic name for RC resources is "odyssey"
- This is the name of the original cluster
  - 4096 core Infiniband cluster
  - Originally listed as #60 on Top500!
- Now it is just the alias for the login pool
- There are many compute and storage resources available
  - 6000+ cores
  - PB+ storage

Using RC Resources

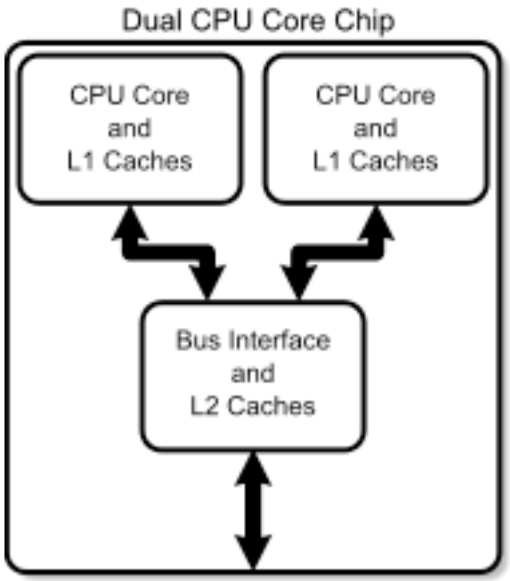
- Users login to access node pool
  - odyssey.fas.harvard.edu
- Compute resources accessed via LSF batch queuing system
- Software environment controlled via modules
- Ability to run parallel jobs
  - Many parallel applications installed on Odyssey
  - You can also run your own parallel code... so let's get programming!

What is parallel computing?

- Doing calculations concurrently (“in parallel”)
- Instruction level parallelism happens “automagically”
  - Intel Harpertown can execute 4 flops/tick
- Thread and process level parallelism must be explicitly programmed
  - Some compilers offer autoparallelism features
- Type of parallel computing available depends on compute infrastructure

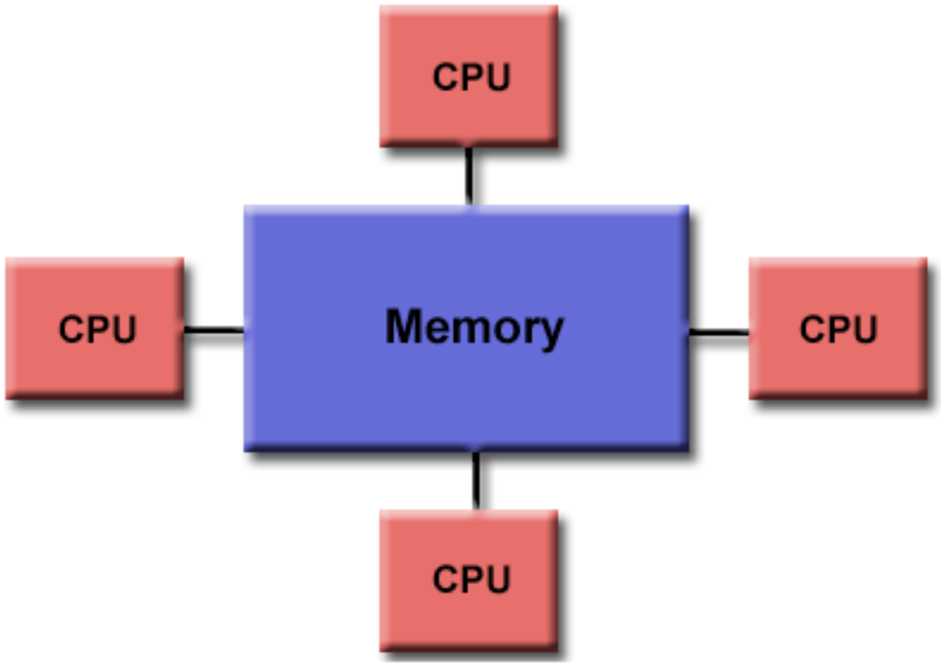
Processing Element (PE)

- Almost all CPU’s in Odyssey are multicore
- Each core can execute instructions and is called a “processing element” in this presentation



Shared Memory Computer Architecture

- PE's operate independently but share memory resources ("global address space")

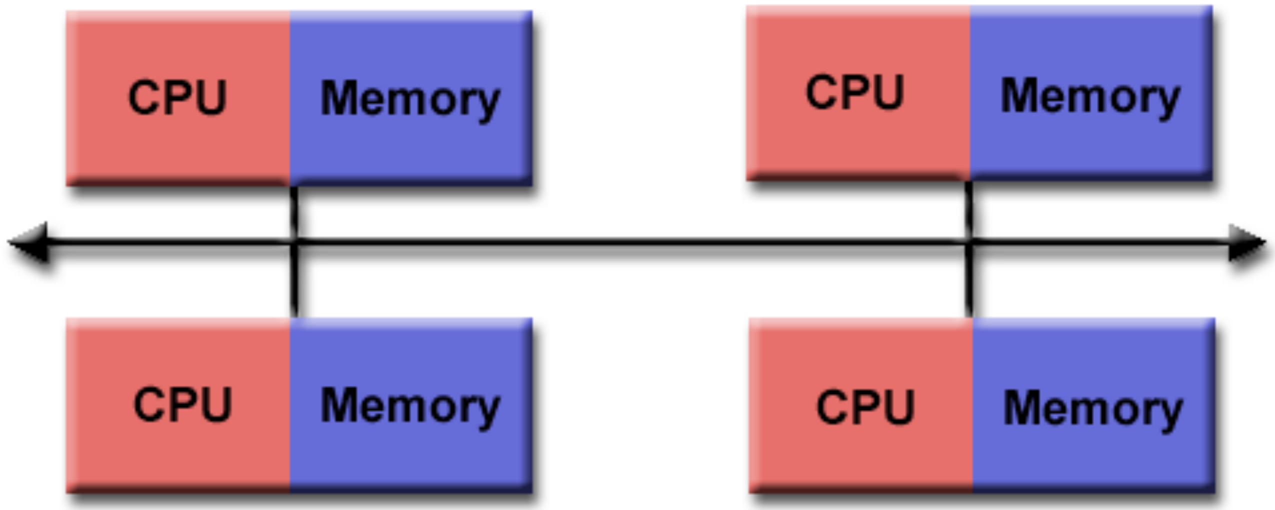


Shared Memory Parallel Programming

- Multiple execution contexts have the same “view” of all or some of the logical address space
- Programs use symmetric multi-processing (SMP) systems like Dell M600 Harpertown blades in Odyssey
  - may have non-uniform memory architecture (NUMA) (some Iliad nodes are Opteron, Nehalems are here!)
- Scientific programmers usually use OpenMP, Posix Threads (Pthreads), or MPI

Distributed Memory Computer

- PE’s have local memory and require a network to communicate with other PE’s



Distributed Memory Parallel Programming

- "Message passing" provided by a library
- Multiple execution contexts with their own address space pertaining to local memory
- Programs are run on any type of system that can communicate over a network
  - MPI jobs on Odyssey use an Infiniband network
- Message passing libraries before MPI
  - Proprietary library from system vendor
  - PVM for network of workstations

What is MPI?

- MPI (Message Passing Interface) is a library containing a set of standardized names and calling sequences enabling programs in Fortran, C or C++ to be run in parallel
- These calling sequences are simply inserted into existing code
- The resulting code is portable and can be run on any parallel compute system
- 1.0 standard in 1994; MPI2 later

MPI Implementations

- Available on Odyssey
  - OpenMPI: http://www.open-mpi.org/
  - MVAPICH: http://mvapich.cse.ohio-state.edu/
- Vendor implementations
  - Intel MPI, HP MPI
  - IBM MPI, Cray MPI
- Original open source
  - MPICH: http://www.mcs.anl.gov/research/projects/mpich2/

Advantages of MPI

- Programmer has ultimate control over how and when messages are passed between PE’s
- Resulting code is portable
- MPI is a standard
- Multiple implementations are available
  - Some are open source
- All HPC interconnects are supported
- Performance is usually good

Disadvantages of MPI

- Programmer has ultimate control over how and when messages are passed between PE's
  - Burden rests entirely on programmer
- Performance and execution can depend on implementation
- Debugging can be very difficult
  - Scaling issues
  - Synchronization and race conditions

MPI API

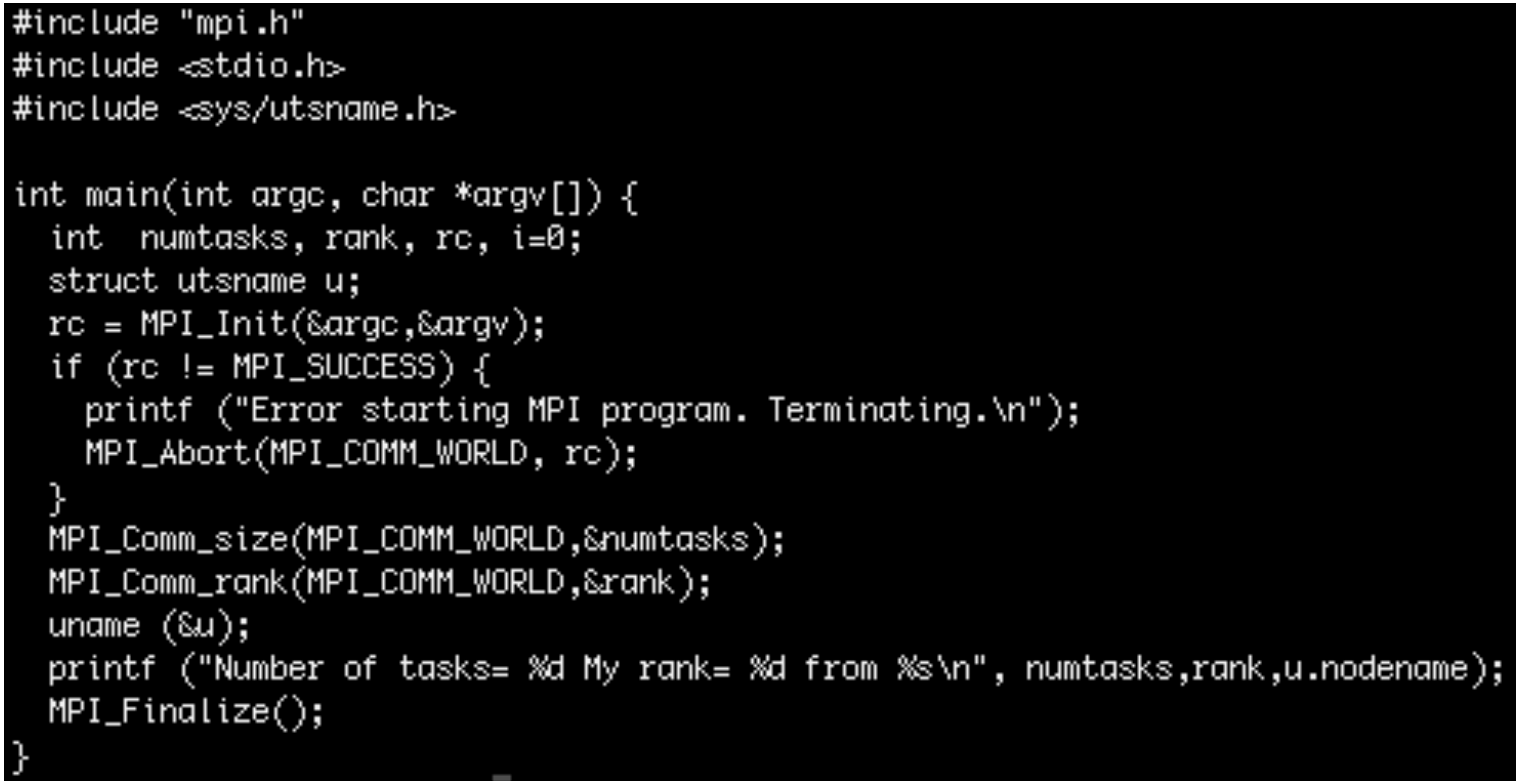
- API has >100 routines
- Most codes use a small subset
- Many books and tutorials explain semantics and provide examples
  - http://www.mcs.anl.gov/research/projects/mpi/
  - https://computing.llnl.gov/tutorials/mpi/
- MPI standard is online
  - http://www.mpi-forum.org/
- Easy to teach yourself (mostly)

FAS INFORMATION TECHNOLOGY

MPI Programs

- MPI programs on Odyssey use a unique process for each MPI rank
- Each rank will run on one PE
- MPI processes are launched out of band over GbE
  - ssh "fan-out" that scales to large node count
  - daemons provide environment, etc
- Running multiple copies of executable across all PE's

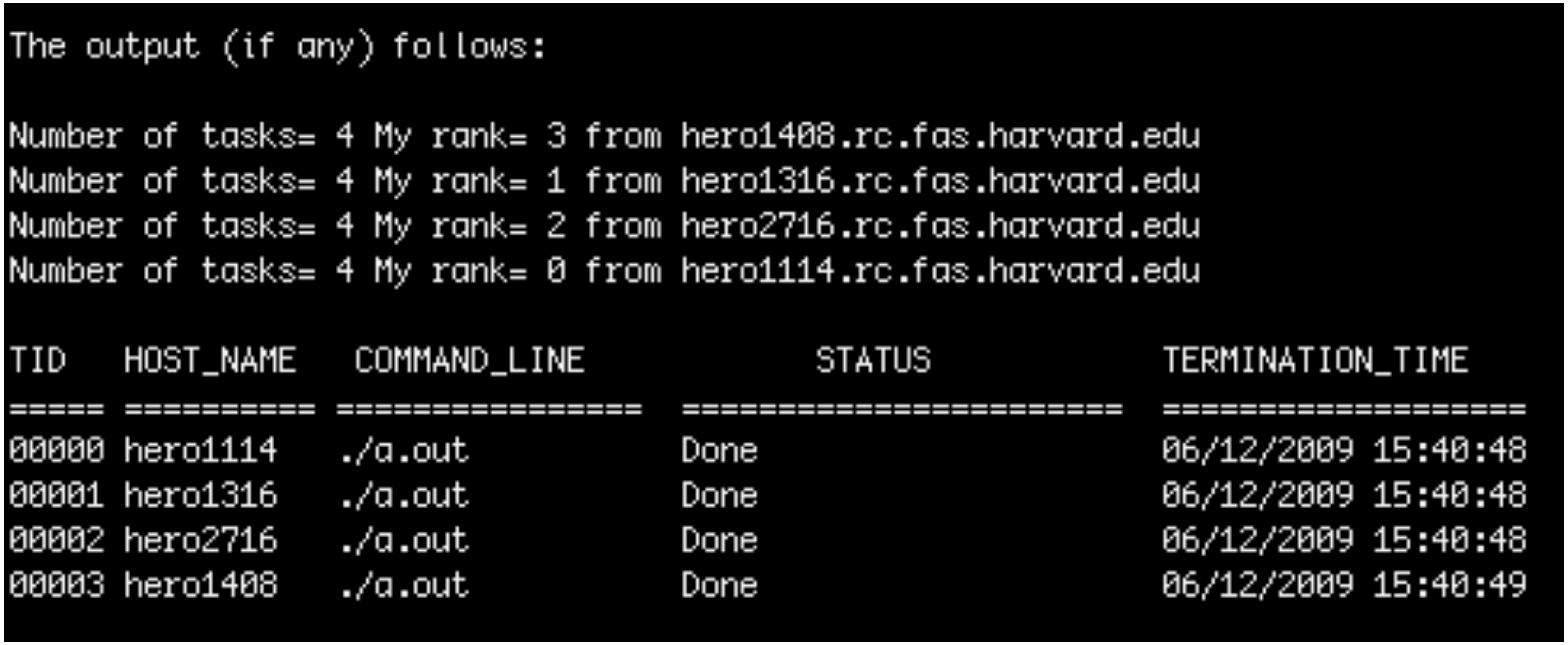
“Hello World” MPI Code



```
#include "mpi.h"
#include <stdio.h>
#include <sys/utsname.h>

int main(int argc, char *argv[]) {
  int  numtasks, rank, rc, i=0;
  struct utsname u;
  rc = MPI_Init(&argc,&argv);
  if (rc != MPI_SUCCESS) {
    printf ("Error starting MPI program. Terminating.\n");
    MPI_Abort(MPI_COMM_WORLD, rc);
  }
  MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  uname (&u);
  printf ("Number of tasks= %d My rank= %d from %s\n", numtasks,rank,u.nodename);
  MPI_Finalize();
}
```

FAS INFORMATION TECHNOLOGY

Running "Hello World" MPI Code



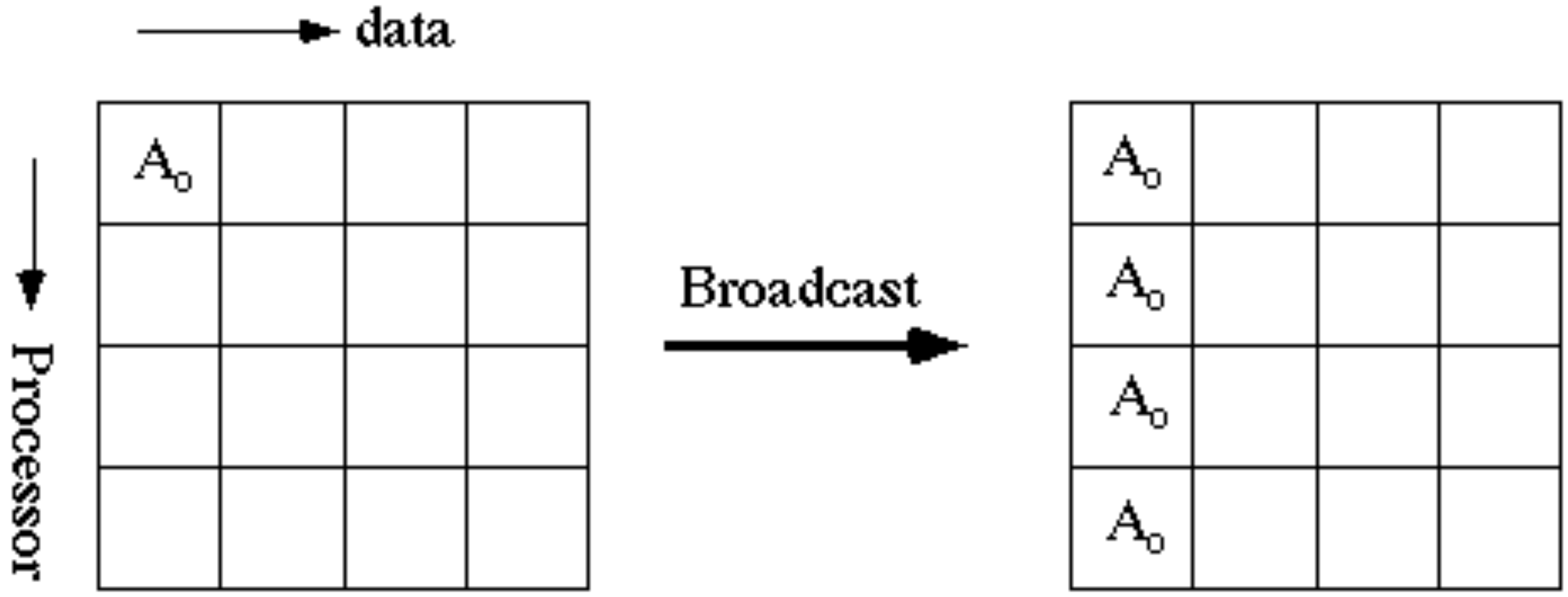
```
The output (if any) follows:

Number of tasks= 4 My rank= 3 from hero1408.rc.fas.harvard.edu
Number of tasks= 4 My rank= 1 from hero1316.rc.fas.harvard.edu
Number of tasks= 4 My rank= 2 from hero2716.rc.fas.harvard.edu
Number of tasks= 4 My rank= 0 from hero1114.rc.fas.harvard.edu

TID   HOST_NAME  COMMAND_LINE              STATUS           TERMINATION_TIME
===== ========== ================ ======================= ===================
00000 hero1114   ./a.out          Done                    06/12/2009 15:40:48
00001 hero1316   ./a.out          Done                    06/12/2009 15:40:48
00002 hero2716   ./a.out          Done                    06/12/2009 15:40:48
00003 hero1408   ./a.out          Done                    06/12/2009 15:40:49
```

FAS INFORMATION TECHNOLOGY

Communicating Using MPI: Collectives

- MPI has an extensive number of routines for sending data between all ranks in a communicator
- These are called “collective” routines
- These are much easier to program than routines communicating between only two ranks
- Routines have been optimized
- Will not scale to large # of PE’s (ranks)
- Not appropriate for large memory jobs

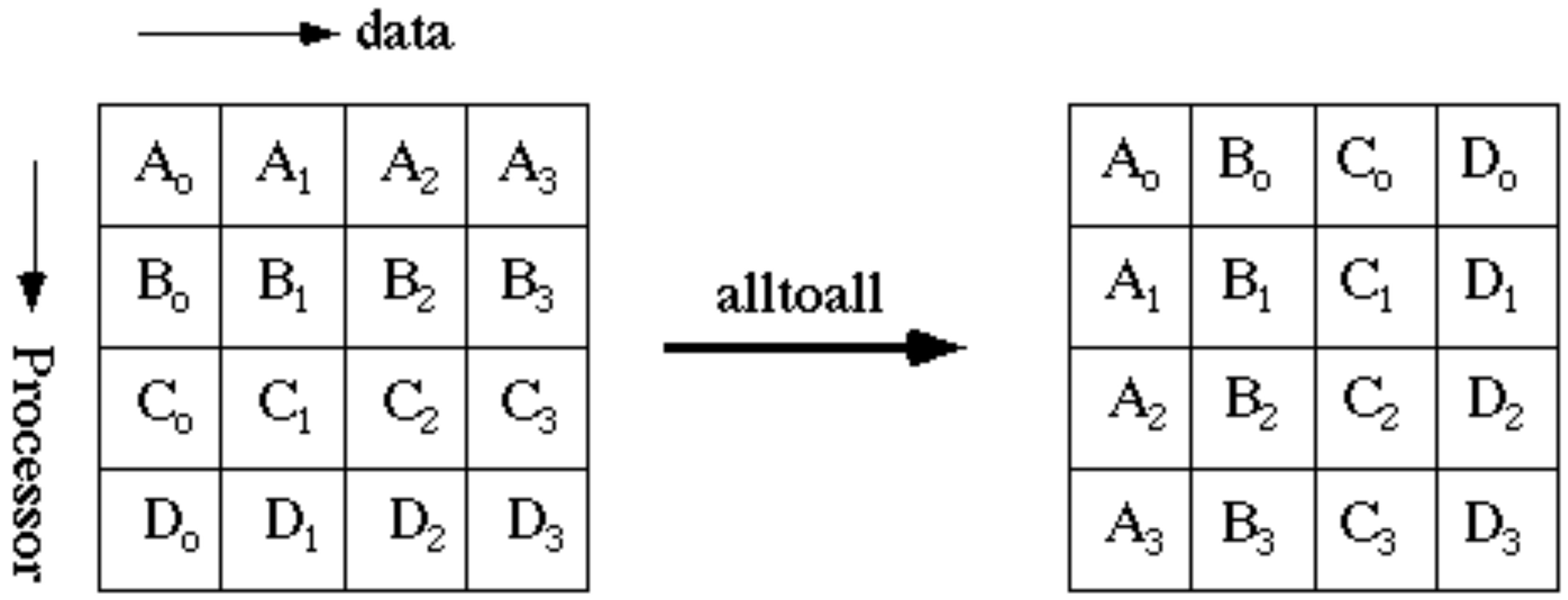
MPI_Bcast(buffer,count,type,source)



Collective Routines

- Reductions, scatter, gather, barrier
- Can account for striding, various data types
- Buffer does not have to be divisible by rank count
- Does not have to be between all ranks
  - Can create subsets of ranks using custom communicators
- MPI_Barrier will synchronize all ranks
  - You will rarely need to do this!

MPI_Alltoall()



Collectives Summary

- Substitute for more complex sequence of sends and receives
  - MPI does the work, not the programmer
- No message tags needed
  - MPI keeps track of messages, ensures progress
- Calls block until they are locally complete
- Routines may or may not synchronize across ranks
  - NOT equivalent to MPI_Barrier

Point to Point Communication

- Message sent from one rank to another rank
- Also called "send and receives"
- 8 basic types of sends
  - 4 communication modes
    - standard, buffered, synchronous, ready
  - blocking vs non-blocking
- "One-sided" communication in MPI2
  - More on this later

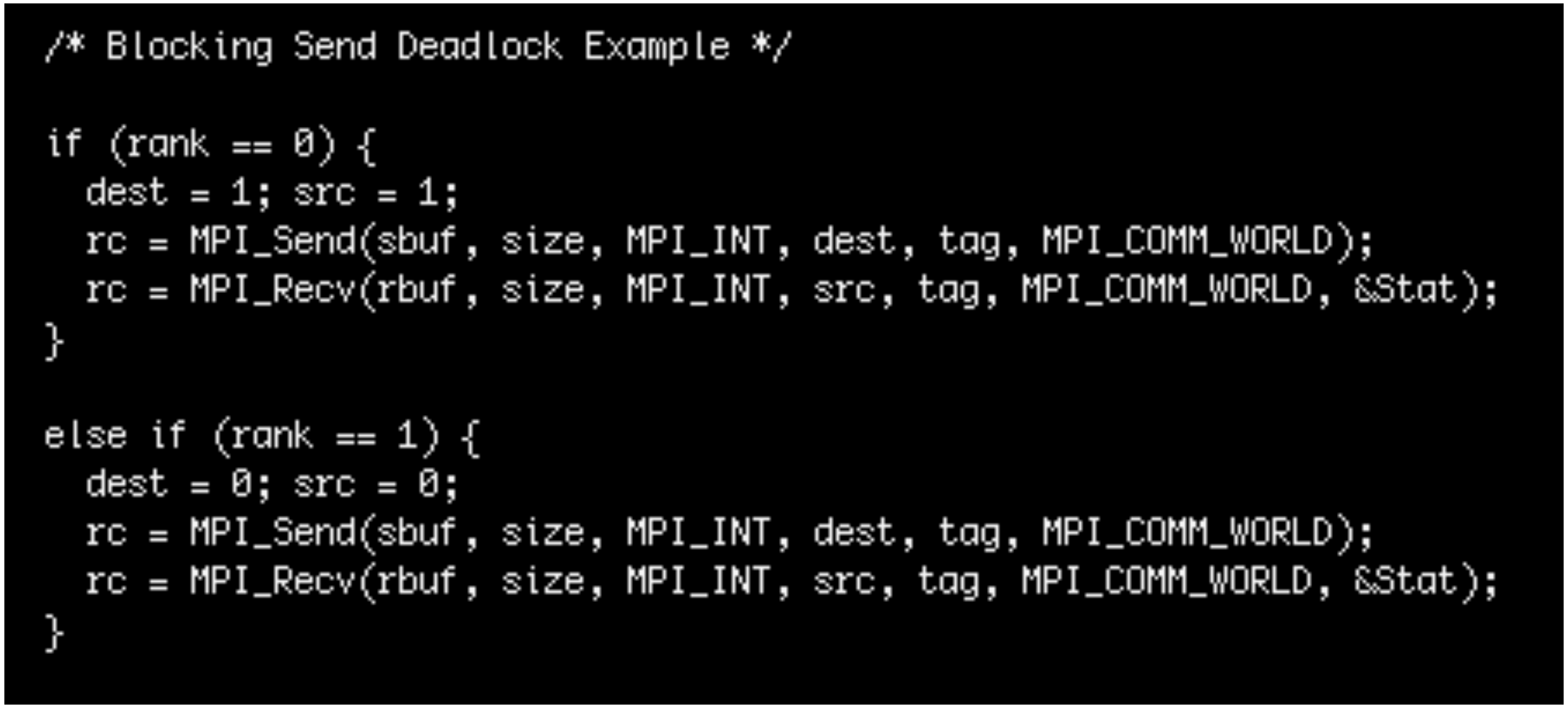
Why So Many Kinds?

- Do ranks synchronize?
- Will message ever be buffered on sending or receiving side?
  - Who manages the buffer?
- How do I know when buffer is safe to be reused?
- How do I know when message has been received?

Standard Mode

- This is probably what you will want to use
  - Hides most details from programmer
  - Does not (necessarily) synchronize ranks
  - MPI determines whether to use system buffer
- Blocking vs non-blocking
  - Buffer safe to use after blocking call returns
  - Must use additional MPI polling/test routines for non-blocking
  - Non-blocking routines allow overlap of computation and communication

Quiz: Will This Deadlock?



```
/* Blocking Send Deadlock Example */

if (rank == 0) {
  dest = 1; src = 1;
  rc = MPI_Send(sbuf, size, MPI_INT, dest, tag, MPI_COMM_WORLD);
  rc = MPI_Recv(rbuf, size, MPI_INT, src, tag, MPI_COMM_WORLD, &Stat);
}

else if (rank == 1) {
  dest = 0; src = 0;
  rc = MPI_Send(sbuf, size, MPI_INT, dest, tag, MPI_COMM_WORLD);
  rc = MPI_Recv(rbuf, size, MPI_INT, src, tag, MPI_COMM_WORLD, &Stat);
}
```

FAS INFORMATION TECHNOLOGY

Answer: It Depends!!

- One factor is that standard blocking send may or may not synchronize ranks
- Synchronization depends on whether "eager" or "rendezvous" protocol is used
  – Rendezvous synchronizes ranks and minimizes use of system buffers
  – This is usually a runtime tunable based on message size; default usually ~16-64KB
  – Exact behavior dependent on implementation type of interconnect (RDMA will usually override)

Make Sure You Code Correctly

- Always match send/recv
  - Avoid "head to head" like my deadlock example
- Use non-blocking if you want to scale to large rank counts
- Underlying protocols/buffering will vary with interconnect type, MPI implementation, message size and rank count, among other things
  - Check MPI standard for correct behavior
  - Just because it worked once doesn't mean bug free

Point to Point Considerations

- Need to avoid deadlock situations
- Ensure rank pairs, tags and request handles scale with # ranks
  - Correct execution at largest anticipated scale
- Cannot access buffer until safe to do so
  - Technically this may include read-only access!!
- Debugging is hard
  - Actually can be really REALLY hard
  - More on this later...

FAS INFORMATION TECHNOLOGY

User Defined Datatypes

- May create new datatypes based on MPI primitive datatypes
- May be non-contiguous
- Several methods available
  - Contiguous
  - Vector
  - Indexed
  - Struct

Misc MPI Functions

- Grid topologies
- Dynamic processes
- MPI I/O
  - More on this later
- There are probably some routines I have never even heard of...
- Lots of references for all of these

Parallelization Strategies

- Code has small memory footprint but long run time
  - Use collective communication routines
- Desire is to run "bigger" simulations than is available with current SMP hardware
  - Use point to point routines
- I/O bound
  - Can read/write from each process and/or MPI I/O
  - I/O is very complex topic though...

Let's Take a Break!

Getting MPI in Your Environment

- FAS RC uses software modules
- Easiest to put module load command in startup files
  - Ensures any process will have correct environment
  - Works for all MPI modules on Odyssey
  - Only works if you are using same MPI for all jobs
- Put module load command in LSF script
  - But make sure you can execute module command!!

Running MPI Jobs

- Make sure your default environment does NOT dump core files!!
  - limit coredumpfile 0
- Recent versions of MPI modules will have all remote processes inherit environment of submission shell
  - This doesn't work for older OpenMPI modules
- If you set manually remember that OpenMPI needs to find orted in your path

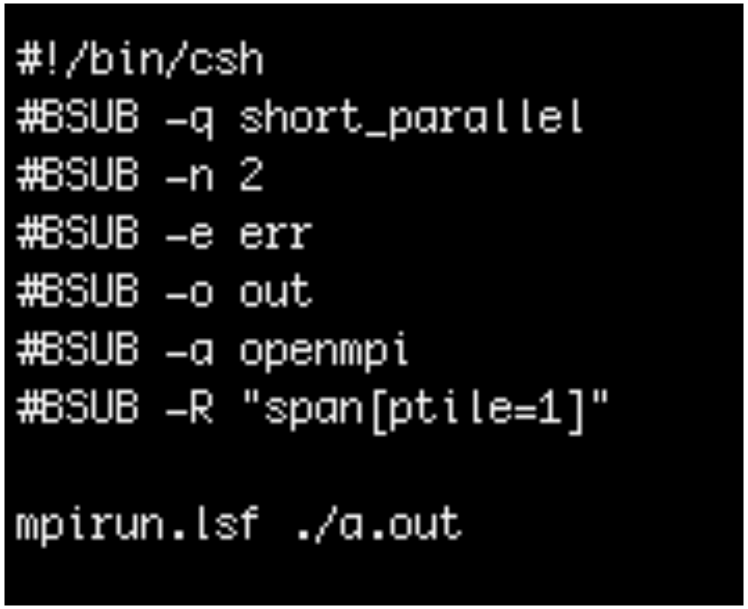
FAS
INFORMATION TECHNOLOGY

MPI Launch Mechanisms

- MPI programs on Odyssey use a unique process for each MPI rank
- Loose integration between LSF and MPI
  - Node reservation and process launch decoupled
  - Should use mpirun.lsf script to ensure processes are scheduled on reserved nodes
  - Do not use machinefiles
  - Can launch script which then calls MPI binary

Sample LSF Script

- Use –a to specify MPI implementation
- Use ptile to control how many MPI processes are launched per node



```
#!/bin/csh
#BSUB -q short_parallel
#BSUB -n 2
#BSUB -e err
#BSUB -o out
#BSUB -a openmpi
#BSUB -R "span[ptile=1]"

mpirun.lsf ./a.out
```

INFORMATION TECHNOLOGY

Parallel Queues on Odyssey

- *_parallel queues + some others
  - These run on 32 GB hero[01-32]* nodes on single IB fabric
  - Scheduled along with serial and other general purpose jobs
- Special project queues
  - These run on 32 GB hero[40-52]* nodes
  - Different IB fabric than *parallel queues
  - May pre-empt serial jobs that land on these nodes (pre-emption means process is sent SIGSTOP)

Cleaning Up Remote Processes

- It is responsibility of MPI implementation to tear down and clean up remote processes
  - Use timeout mechnism
  - Some implementations do a better job than others...
- Programmer should help by checking error conditions and calling Finalize/Abort on all ranks
- Odyssey users may log into remote nodes themselves and kill rogue processes they own

FAS INFORMATION TECHNOLOGY

MPI Tunables

- All MPI implementations have many runtime tunables
- Most of them are for performance
  - The defaults are usually good
  - Can waste a lot of time fiddling with these
- A few might be critical for code to function properly
  - Disable mpi_leave_pinned for OpenMPI
  - This will be covered later in the talk

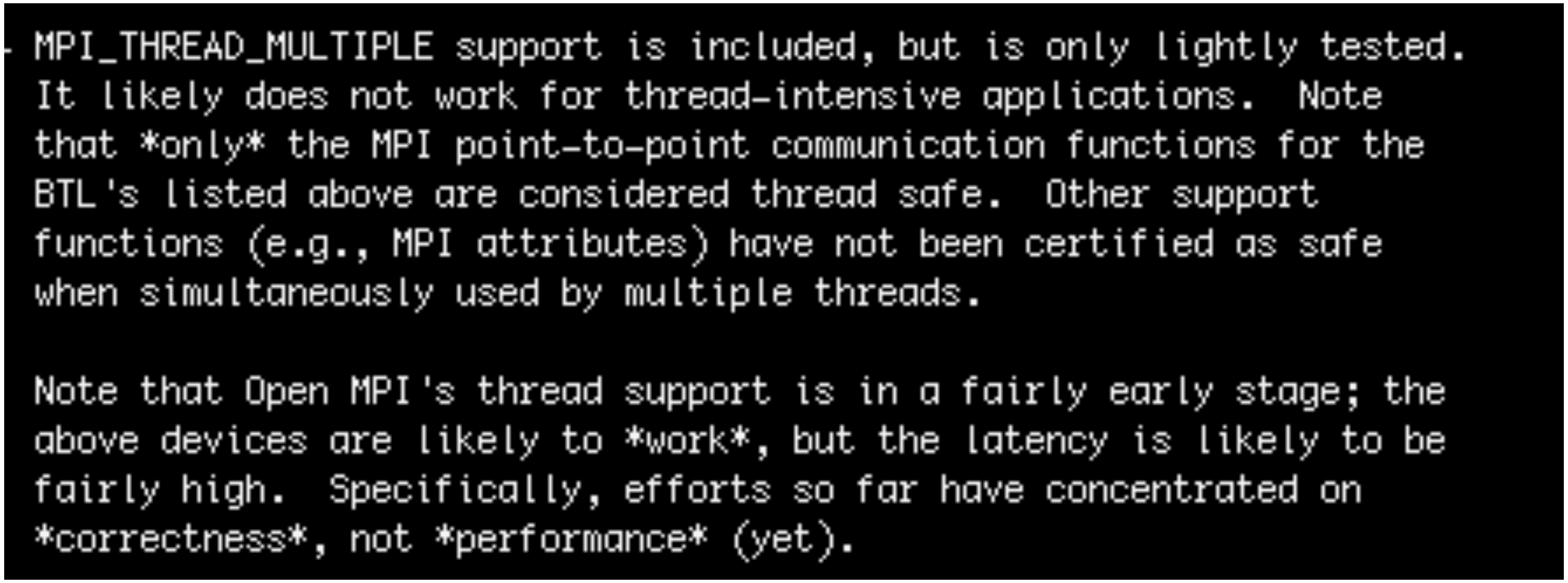
MPI and Threads

- MPI standard defines "thread compliant"
  - All MPI calls must be "thread safe"
  - Blocking routines block calling thread only
- Specific MPI routines to handle threads
  - MPI_Init_thread
  - Specifies level of threading support needed
  - What if you don't alert MPI to thread use?
  - How does message progress happen?

"Thread Safe" MPI

- Most MPI implementations claim to be "thread safe"
  - But what does this really mean?
- Sometimes requires linking a special version of the MPI library
- Usually ok to have only one thread calling all MPI routines (MPI_THREAD_FUNNELED)
- Would recommend against "threaded" MPI programming model in general

OpenMPI 1.3.2 Release Notes



```
- MPI_THREAD_MULTIPLE support is included, but is only lightly tested.
  It likely does not work for thread-intensive applications.  Note
  that *only* the MPI point-to-point communication functions for the
  BTL's listed above are considered thread safe.  Other support
  functions (e.g., MPI attributes) have not been certified as safe
  when simultaneously used by multiple threads.

  Note that Open MPI's thread support is in a fairly early stage; the
  above devices are likely to *work*, but the latency is likely to be
  fairly high.  Specifically, efforts so far have concentrated on
  *correctness*, not *performance* (yet).
```

FAS INFORMATION TECHNOLOGY

Parallel I/O

- Large complex but very important topic...
  - Needs its own presentation
- Often local scratch is good option
  - Don't use /tmp, use /scratch
  - Clean up after yourself!
- Odyssey has many parallel filesystem options
  - http://hptc.fas.harvard.edu/odyssey/faq/filesystems
- Should parallelism be at application or filesystem level? Or both?

MPI I/O

- MPI I/O was added to MPI2 standard
- Solves many issues of *programming* parallel I/O but mostly silent on *performance*
  - Shared file pointers
  - Collective read/write
  - Portable data representation
- Performance requires a smart programmer, good MPI implementation, and great filesystem!
- Check Web for references/tutorials

MPI Over Infiniband

- Infiniband (IB) is an open standard for a high speed low latency switched network
  - http://www.infinibandta.org/
- Requires special hardware and software
- Capable of remote DMA (RDMA) zero-copy transfers
- Performance results measured on Odyssey
  - Latency: 1.2 usec
  - Bandwidth: 1.2 GB/s

IB "verbs"

- IB does not have an API
- Actions are known as "verbs"
  - Services provided to ULP
  - Send verb, receive verb, etc
- On Odyssey we use the open source OFED distribution from Openfabrics to provide verbs to our MPI implementations

Remote DMA (RDMA)

- HCA directly sends/receives from RAM
- No involvement from main CPU
- But...
  - OS can change virtual <-> physical address mapping at any time!!
  - This creates a race condition between HCA sending data and VM mapping changes
- RDMA not just used in IB

Solution: Register Memory

- Tell OS not to change mapping
  - This is called "pinning" memory
  - Guarantees buffer will stay in same physical location until HCA is done
- In IB verbs this is called "registering" memory
  - Static virtual <-> physical mapping
  - Notifies HCA of mapping
  - Pinned pages cannot be swapped
  - This is very expensive operation

MPI Memory Agnostic

- MPI standard allows ANY buffer to be used in send/recv
- MPI standard tried to address RDMA memory issues in MPI2
  - One-sided operations (MPI_Put/MPI_Get)
  - MPI_Alloc_mem
  - But no one uses these!!
- Users want RDMA performance for any MPI routine

Why Do We Care??

- This may create problems
  - fork()/exec()
  - http://www.open-mpi.org/faq/?category=openfabrics#ofa-fork
- By default most MPI implementations intercept malloc library call
  - This can usually be disabled but you risk MPI being unaware of memory being returned to OS
  - This can happen anyway!!
  - Can control cache via mpi_leave_pinned tunable (and related parameters)

FAS INFORMATION TECHNOLOGY

General Performance Issues

- Good rule of thumb for estimating performance
  - Memory accesses (load/store) ~nanoseconds
  - Interconnect (MPI send/recv) ~microseconds
  - I/O (read/write) ~milliseconds
- Important to understand where the "bottleneck" occurs
- More abstraction/virtualization usually means worse performance

Always be Aware of Resource Usage!

- Are you going to run out of memory?
  - Most nodes on Odyssey have 32 GB
  - Paging/swapping will kill performance and probably the node...
- Do you need to do lots of I/O?
  - Match filesystem with I/O workload
  - Large streaming writes should use lustre (/n/data)
  - Small file I/O... Best to talk with us first
  - Don't hammer our home filesystem please :o)

"How Well Does My Code Scale?"

- How long is a piece of string?
- Many parameters must be fixed before this question becomes meaningful
  - System hardware, software, and environment
  - Path through code
  - Data set
- Scaling is also a function of rank layout
  - # ranks/node
  - Core/memory affinity

FAS INFORMATION TECHNOLOGY

Performance Tuning on Odyssey

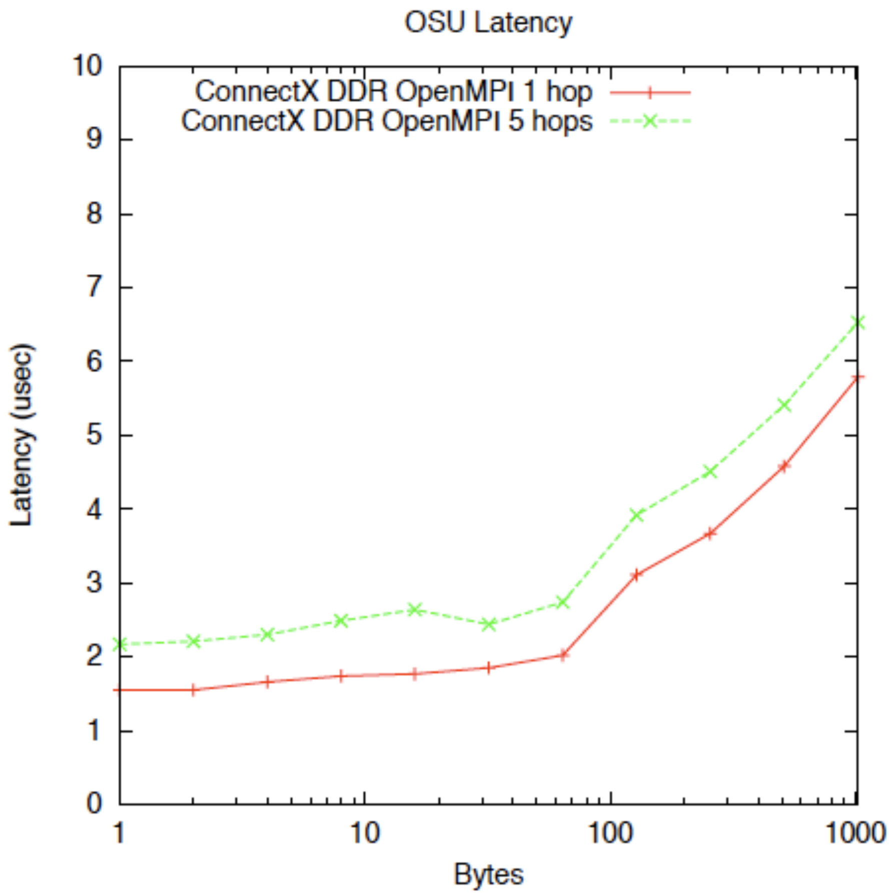
- Very hard to quantify performance improvements on system like Odyssey
- All resources are shared
  - Compute nodes
  - Infiniband interconnect
  - Filesystems (many of which have different performance characteristics!)
- Odyssey is architected to achieve maximum amount of science not best individual job runtime

Variability on Odyssey

- Should expect some amount of variability run to run in most cases
- It is not possible to achieve "benchmark conditions" on this system; it is production environment
- Other sources of variability
  - IB statically routed
  - VM history of node

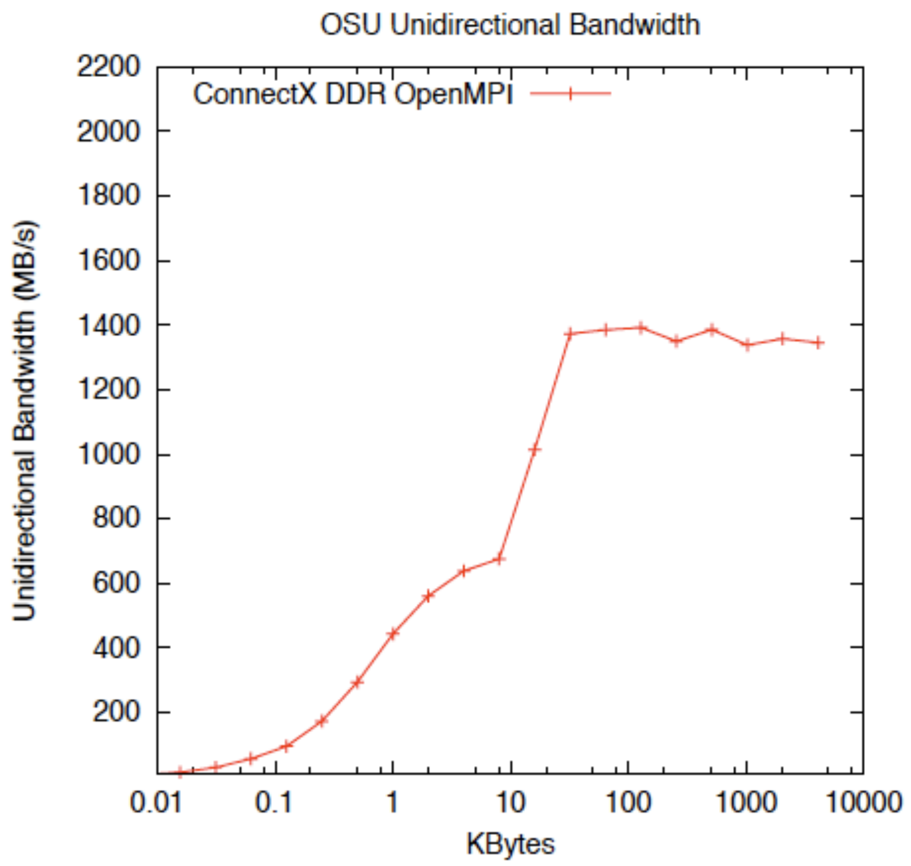
FAS INFORMATION TECHNOLOGY

MPI Latency on Odyssey



FAS INFORMATION TECHNOLOGY

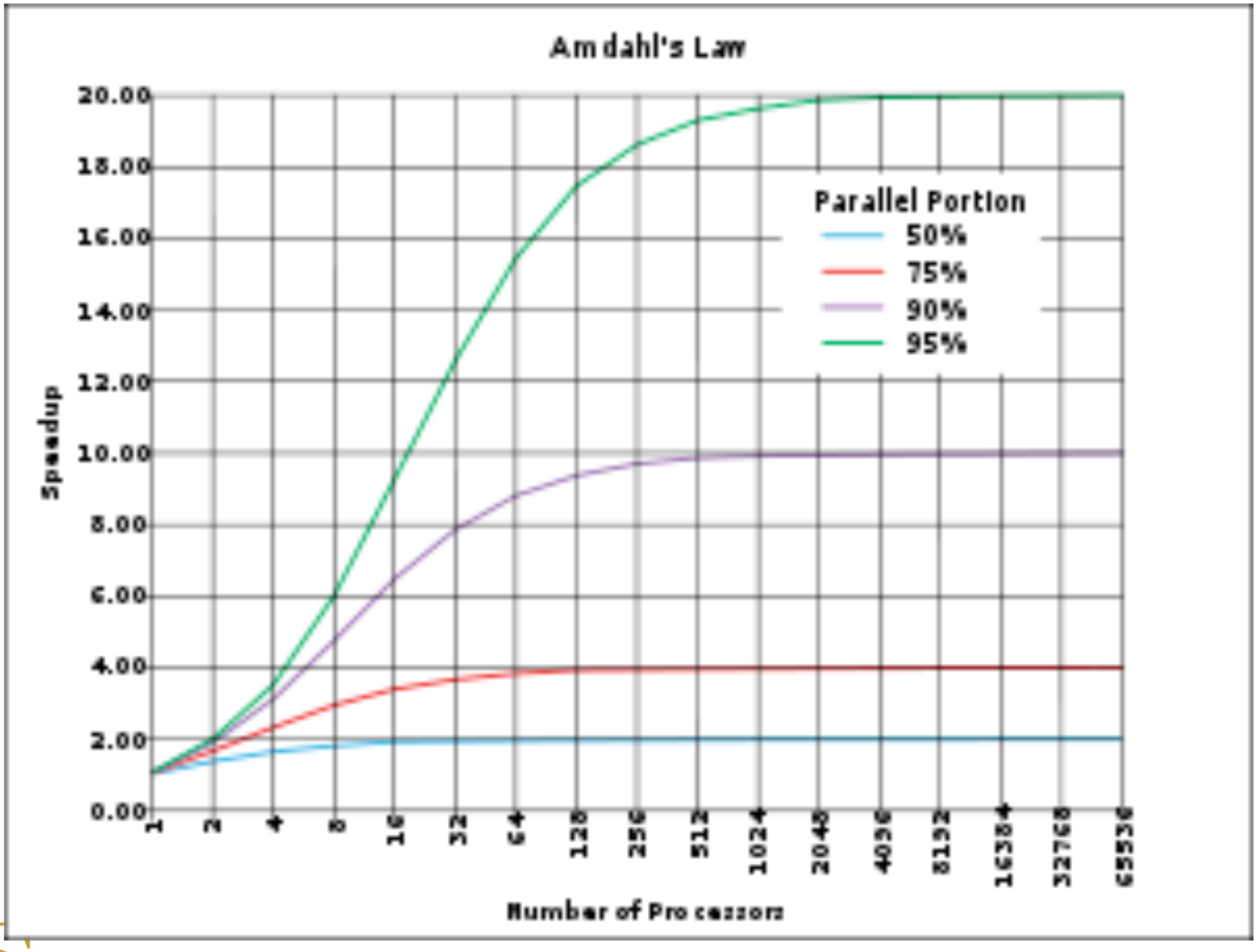
Unidirectional Bandwidth on Odyssey



Application Scaling

- Programmers often use Amdahl's law to estimate scaling
- Speedup is limited by the time needed for the serial portion of program
- Sources of serial execution
  - I/O
  - Startup/shutdown
  - Synchronization points

Amdahl's Law



Amdahl's Law
Speedup
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Number of Processors
Parallel Portion
50%
75%
90%
95%

Other Performance Issues

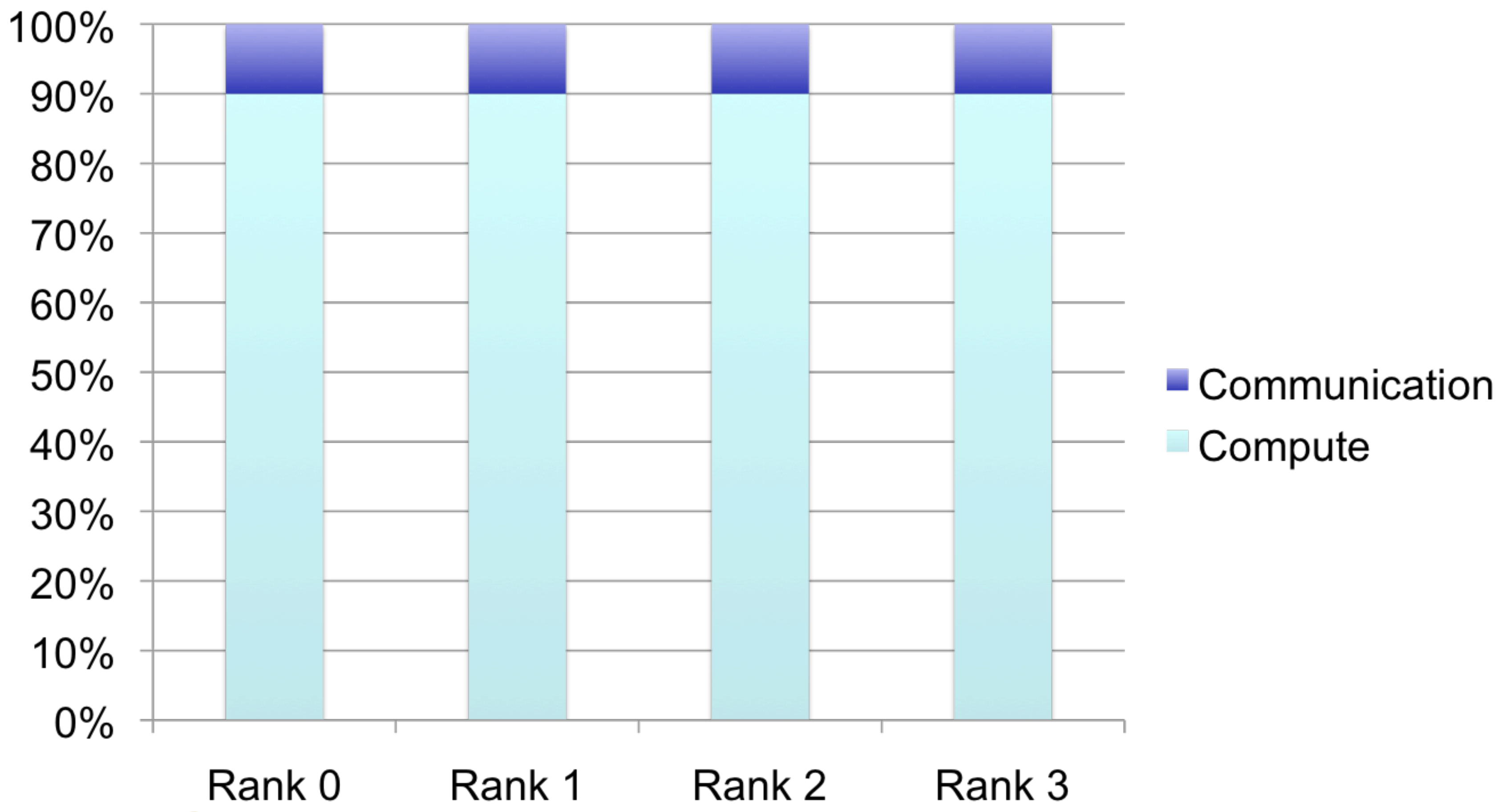
- Amdahl's Law is theoretical limit of speedup
  - Not that useful IMHO...
- Other important factors are *communication overhead* and *load balancing*
- Communication overhead
  - You can never achieve perfect speedup even if all execution can be parallelized
  - Communication overhead will often dominate for many codes at large PE count

FAS INFORMATION TECHNOLOGY

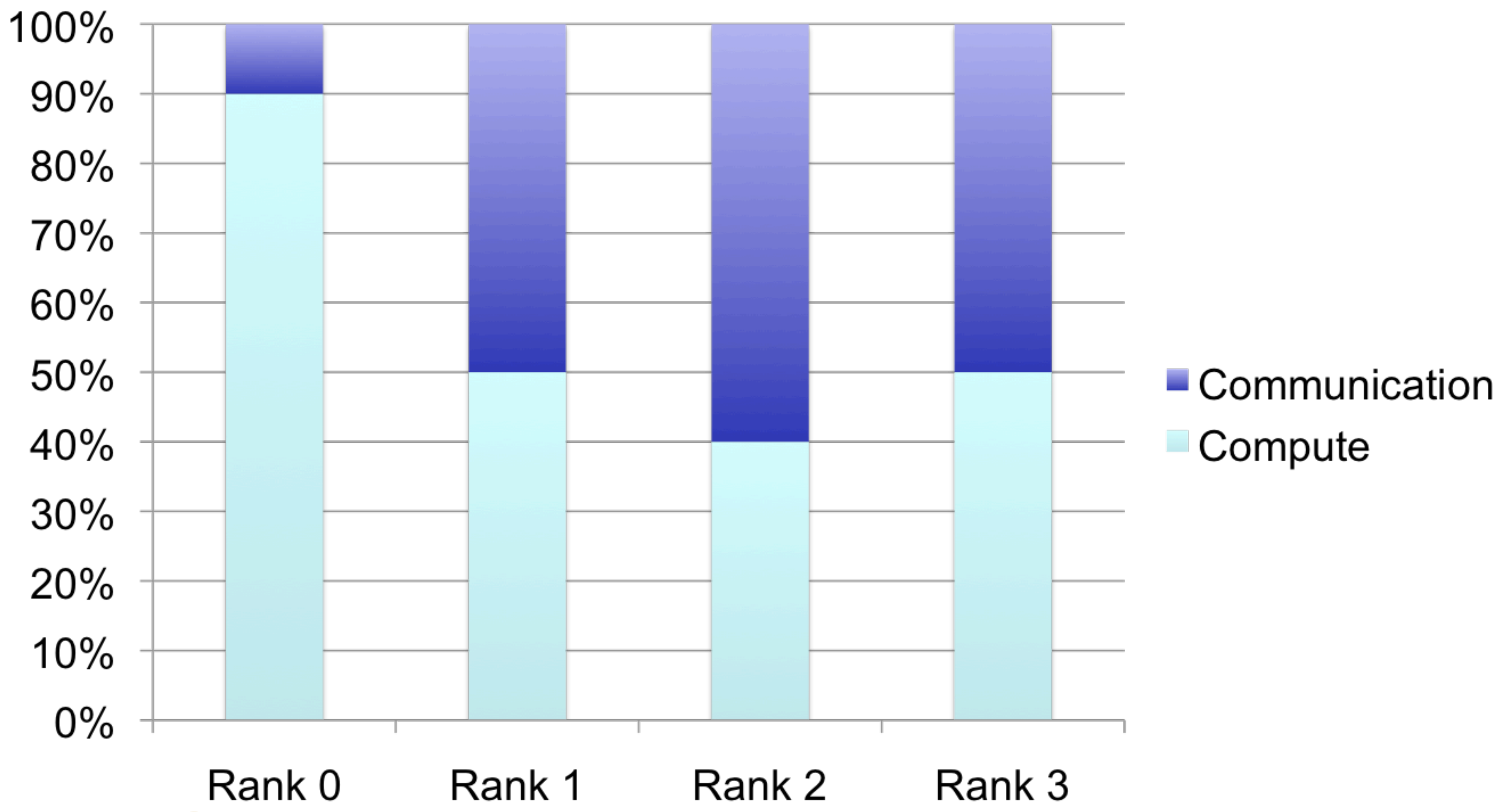
Load Balancing

- A parallel program will only run as fast as the slowest rank
- Should distribute workload evenly across ranks
- Often most difficult part of parallel programming!
  - Often good serial algorithms work poorly in parallel
  - Sometimes better to change algorithms entirely
  - May want to duplicate computational work rather than add communication overhead
  - Check literature

Good Load Balancing



Poor Load Balancing



Scaling Application Workload

- As you add more ranks you should increase problem size
- Increasing rank count with fixed problem size leads to communication overhead dominating run time
  - Amount of work per rank decreases
  - Can occasionally see "superscaling" where speedup is better than linear at large rank counts
  - This is due to resident set fitting entirely in cache

FAS INFORMATION TECHNOLOGY

MPI Debugging

- Totalview is good for deterministic problems on small scale
  - See FAS RC web site for instructions
  - Has a lot of useful functionality
  - Hard to use in production batch environment like Odyssey
- Remember printf changes timing!!
- Can use wrappers
  - Write MPI_ routine that calls PMPI_

FAS INFORMATION TECHNOLOGY

Debugging Large Parallel Jobs

- Bugs often only happen at scale and with optimization turned on
  - Most non-trivial bugs are due to race conditions
- Debugging hangs
  - Attach gdb to rank 0 and some other random rank
  - Can usually figure out location of the problem
- Debugging non-deterministic crashes
  - Observe behavior as you vary rank count
  - Selectively dump core files

Conclusions

- MPI library is standard for writing parallel scientific applications
- Will be supported for longer than science will be interesting!
- Learning curve is shallow for API but steep for scaling to large # PE's
- Odyssey is a fantastic resource for academic researchers wanting to develop parallel codes

Any Questions?

- Harvard Research Computing Web Site
  - http://hptc.fas.harvard.edu/
- Email
  - rchelp@fas.harvard.edu
  - kaltz@fas.harvard.edu

FAS INFORMATION TECHNOLOGY