Janus Supercomputer Bootcamp – Updates and Some Parallel Computing

Shelley Knuth and Pete Ruprecht, Research Computing, University of Colorado-Boulder

shelley.knuth@colorado.edu peter.ruprecht@colorado.edu

Link to survey on this topic: http://goo.gl/forms/8VidcwOhRT

Slides: https://github.com/ResearchComputing/Final_Tutorials

Outline

- Update on awarded Janus MRI Proposal
- UCD BiPM HPC Update
- Parallel Programming Examples
- One-On-One Interactions and Tutorials

Next-Generation Supercomputer at CU-Boulder

- Funded via an NSF MRI grant awarded jointly to CU-Boulder and CSU
- $2M to CU and $700K to CSU … with matching funds the hardware budget is about $3.5M
- RFP has been published, vendor award by end of November (hopefully!)
- Installed and running late spring 2016

Next-Generation Supercomputer

- Expected performance about 450 TFLOPS (compared to about 170 for Janus)
- Compute nodes
  - Expect 24 real cores and 128 GB RAM
- 10 GPU/visualization nodes
  - 2x NVIDIA K80 GPUs
- 5 High-memory nodes
- 20 Xeon Phi (“Knight’s Landing”) nodes
- “Omni-Path” high-performance interconnect
- 1 PB of high-performance scratch storage

Next-Generation Supercomputer

- 10% of CPU-hours available to RMACC
- Opportunities to buy additional nodes to improve your job priority and guarantee an allocation
- "Name the supercomputer" contest
  - Best suggestion wins a Kindle Fire tablet
  - Send entries to becky.yeager@colorado.edu by 10/31

UCD BiPM HPC Update

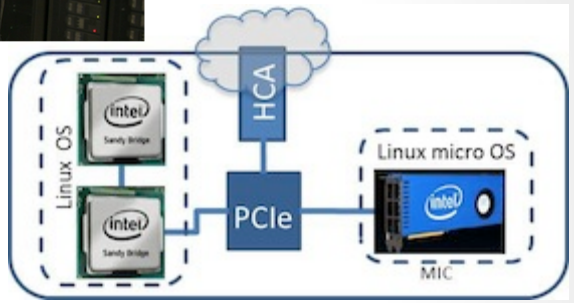
Parallel Computing with Examples (OpenMP)

Outline

- Shared memory
- What is OpenMP?
- How is OpenMP used?
- Parallel region
- Public/Private variables
- Examples

Programming to Use Parallelism

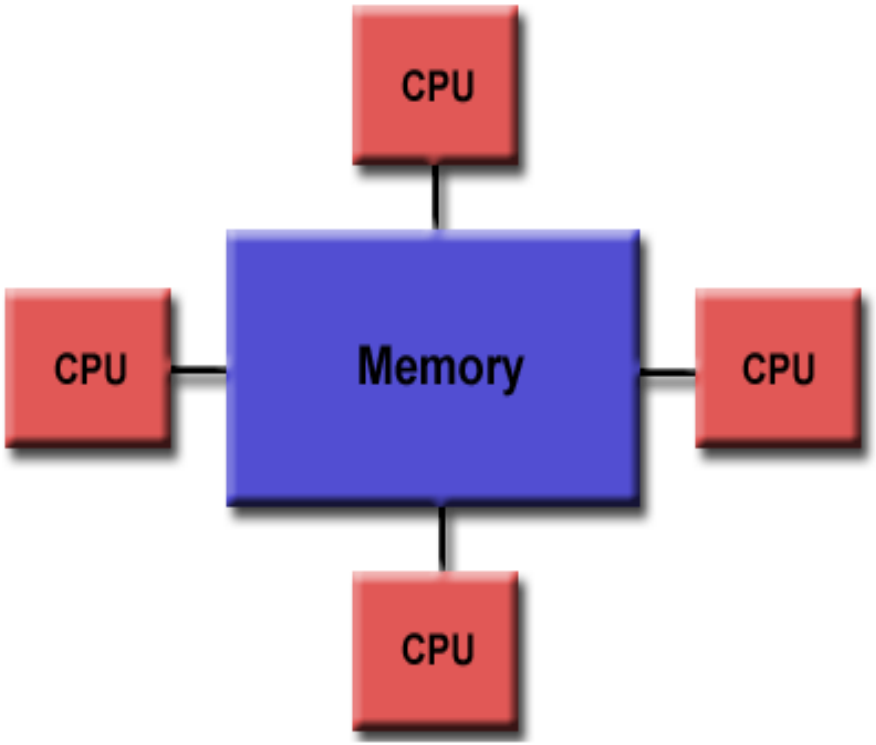
- Parallelism across processors/threads - OpenMP



- Parallelism across multiple nodes - MPI

www.scan.co.uk

Shared-memory Model



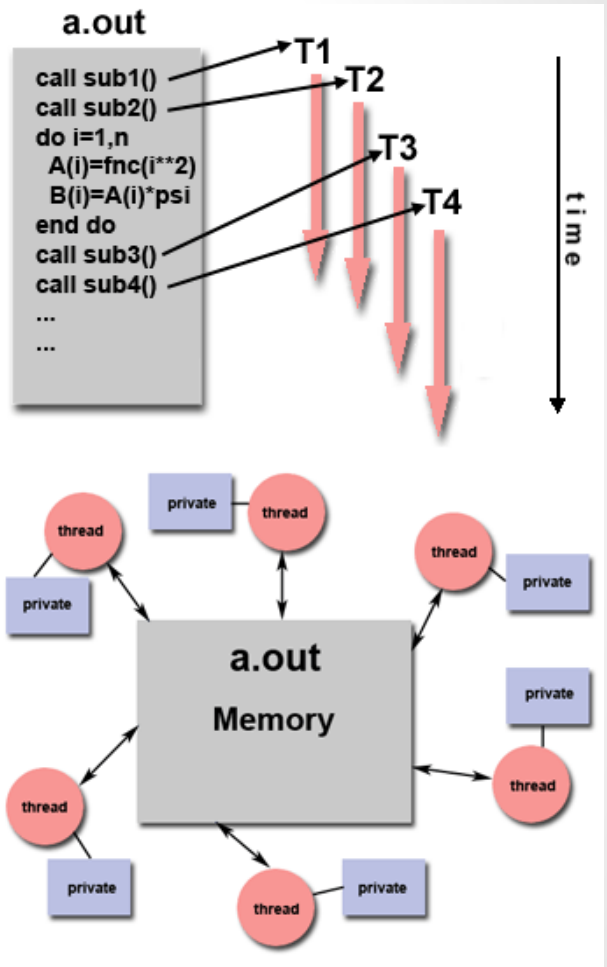
The concept is that all processors can access all memory available

Multiple processors can perform tasks on their own but share the same memory

Source: https://computing.llnl.gov/tutorials/parallel_comp/#ModelsShared

Multi-Threaded, Shared Memory Parallelism

- Main program does many things, including run subroutines
  - Threads that can be run concurrently
  - Share the same resources from the main program, but also has local data
  - Threads communicate through global memory
  - Must ensure multiple threads don't update concurrently
  - Where OpenMP and programmers come in



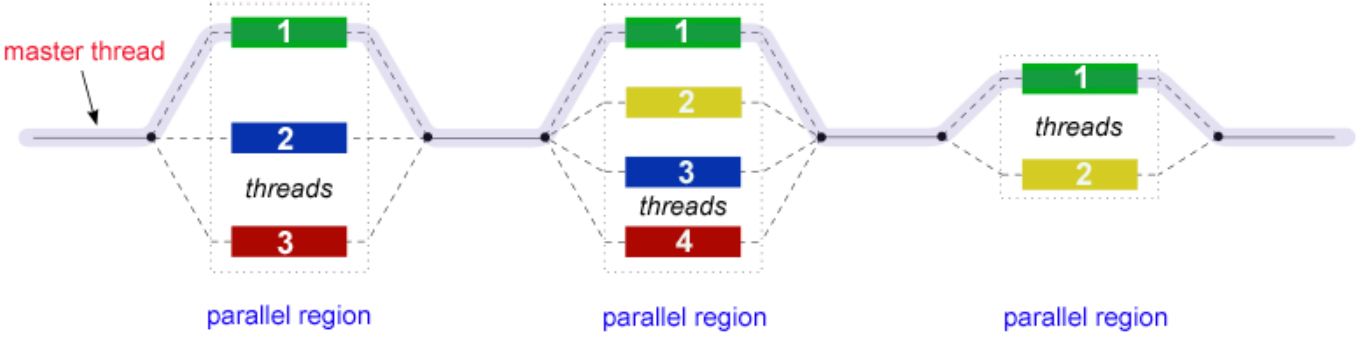
Source: https://computing.llnl.gov/tutorials/parallel_comp/#ModelsShared

OpenMP

- OpenMP: An application programming interface (API) for parallel programming on multiprocessors
- Uses shared memory
- OpenMP is used through compiler directives embedded in Fortran, C, or C++ code
- Directs multi-threaded, shared memory parallelism
- Can do a lot with only a handful of commands
- Intended to be easy to use

OpenMP – Fork/Join

- OpenMP programs start with a single thread (master)
- Then Master creates a team of parallel “worker” threads (FORK)
- Statements in block are executed in parallel by every thread
- At end, all threads synchronize and join master thread



OpenMP Directives

- Comments in source code that specify parallelism for shared memory machines
  - Enclosing parallel directives
- **FORTRAN**: directives begin with **!$OMP**, **C$OMP** or ***$OMP**
- **C/C++**: directives begin with **#pragma omp**

OpenMP Fortran: General Code Structure – Parallel Regions

Parallel regions are blocks of code that will be executed by multiple threads

```
1      !$OMP PARALLEL
2            code block
3            call work(…)
4      !$OMP END PARALLEL
```

```
Line 1        Team of threads formed at parallel region.
Lines 2-3     Each thread executes code block and subroutine
              calls. No branching (in or out) in a parallel
              region.
Line 4        All threads synchronize at end of parallel
              region (implied barrier).
```

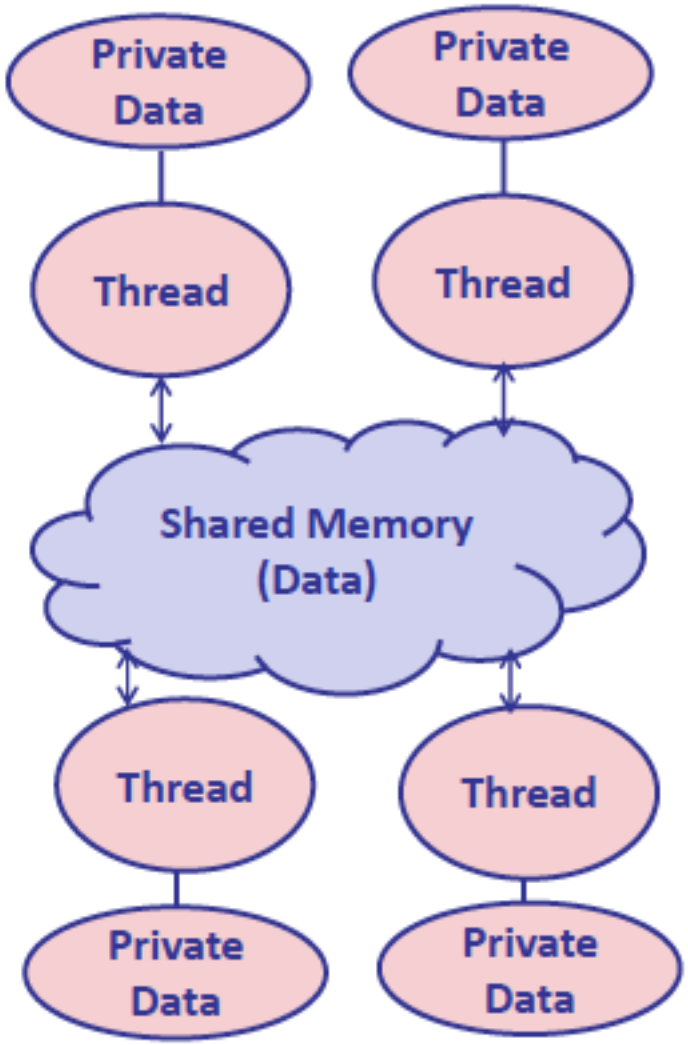
```
Use the thread number to divide work among threads.
```

Parallel Regions

- When thread hits PARALLEL directive, creates team of threads
  - Becomes master
  - Code is duplicated and all threads execute that code
  - Runs the same code on different data
    - Split up loops and operate on different data
  - Only master thread continues after implied barrier
- Can determine number of threads by:
  - Setting the number threads to a default number or within code
  - Allowing number of threads to change from one parallel region to another

Shared and Private Variables

- When specifying the **PRIVATE** clause, that variable is private to each thread
  - Each thread has own unique copy
  - Can only be accessed by the threads that own it
  - Variables declared in private subroutines are default private
  - Index variables are also default private
- When specifying **SHARED** clause, all threads can access that data
  - Global variables are shared by default



Private Variable Example

```
#pragma omp parallel for shared(a,b,c,n) private(temp,i)
      for (i=0; i< n; i++){
            temp = a[i] / b[i];
            c[i] = temp + cos(temp);
      }
```

- All threads can access a, b, c, and n
- Each loop has own private copy of index i

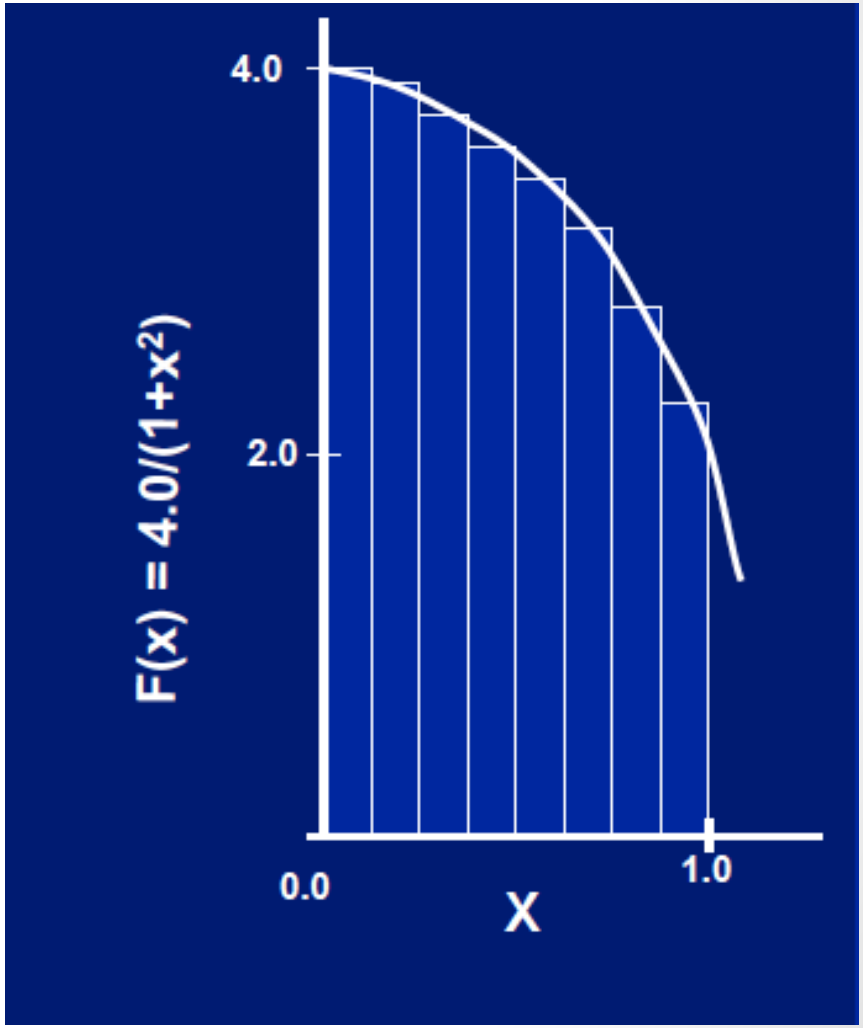
- Variable temp also needs to be private
- Otherwise each thread would be reading/writing to same location

Parallel Region Example

- Finding the integral
  - Area under a curve
  - Sum of the area of all the rectangles underneath the curve (approximate)

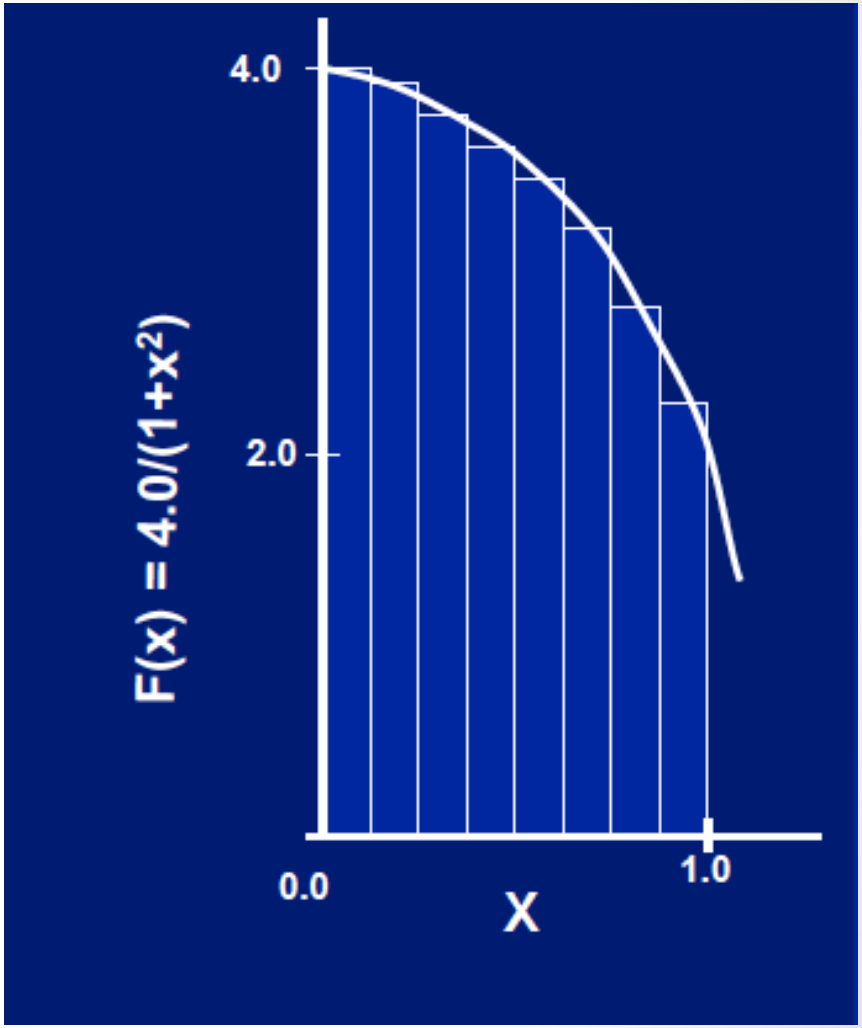
$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

$$\sum_{i=0}^{N} F(x_i)\Delta x \approx \pi$$



Parallel Region Example

- The same code is used to calculate the area of each of the rectangles
- Different threads will calculate different rectangles
- Which rectangles are calculated with each thread is random



OpenMP Compiling

- When compiling must use appropriate compiler flag to turn on OpenMP compilations

Compiler / Platform	Compiler	Flag
Intel Linux Opteron/Xeon	icc icpc ifort	-openmp
PGI Linux Opteron/Xeon	pgcc pgCC pgf77 pgf90	-mp
GNU Linux Opteron/Xeon IBM Blue Gene	gcc g++ g77 gfortran	-fopenmp
IBM Blue Gene	bgxlc_r, bgcc_r bgxlC_r, bgxlc++_r bgxlc89_r bgxlc99_r bgxlf_r bgxlf90_r bgxlf95_r bgxlf2003_r *Be sure to use a thread-safe compiler - its name ends with **_r**	-qsmp=omp

Runtime Library Routines

Routine	Purpose
OMP_SET_NUM_THREADS	Sets the number of threads that will be used in the next parallel region
OMP_GET_NUM_THREADS	Returns the number of threads that are currently in the team executing the parallel region from which it is called
OMP_GET_THREAD_NUM	Returns the thread number of the thread, within the team, making this call.
OMP_GET_THREAD_LIMIT	Returns the maximum number of OpenMP threads available to a program

- In C/C++, must include the omp.h header file

Fortran	**INTEGER FUNCTION OMP_GET_NUM_THREADS()**
C/C++	**#include <omp.h>** **int omp_get_num_threads(void)**

OMP Code Practice – Exercise 1

- Code:

  `omp_hello.f`

- Instructions for running:

```
ssh tutorial-login.rc.colorado.edu –l user00XX
ml intel
ml slurm
sinteractive --reservation=anschutz
ifort -qopenmp omp_hello.f -o hello
./hello
```

How Do I Prepare My Code for OpenMP?

- I have code! I want it to be parallel too!

- Steps to go through
  1. Verify that code is parallelizable
     - Make sure you don't have any loop dependencies
  2. Analyze your code
     - Where does the program spend most of its time?
     - Look for loops
       - Typically easy to parallelize
       - Outside of nested loops

How Do I Prepare My Code for OpenMP?

- Steps to go through

3. Restructure code
   - Put `parallel do` constructs around parallelizable loops
   - List variables with appropriate `shared`, `private`, etc. clauses
   - Many other things you can do that we don't cover here

4. Overhead
   - How much time was spent preparing your code for parallelization?
   - Is this more than the time spent running your code serially?

Example Code – Exercise 2

- Code:
  `for.c`

- Instructions for running:

```
ssh tutorial-login.rc.colorado.edu –l user00XX
ml intel
ml slurm
sinteractive –reservation=anschutz

icc for.c -o for_noflag
time ./for_noflag 10000000

icc -qopenmp for.c -o for
time ./for 10000000
```

Example Code – Exercise 2

- We need to consider whether our code really does experience a speed up
  - Array size 10,000,000
  - Drops by ~30%

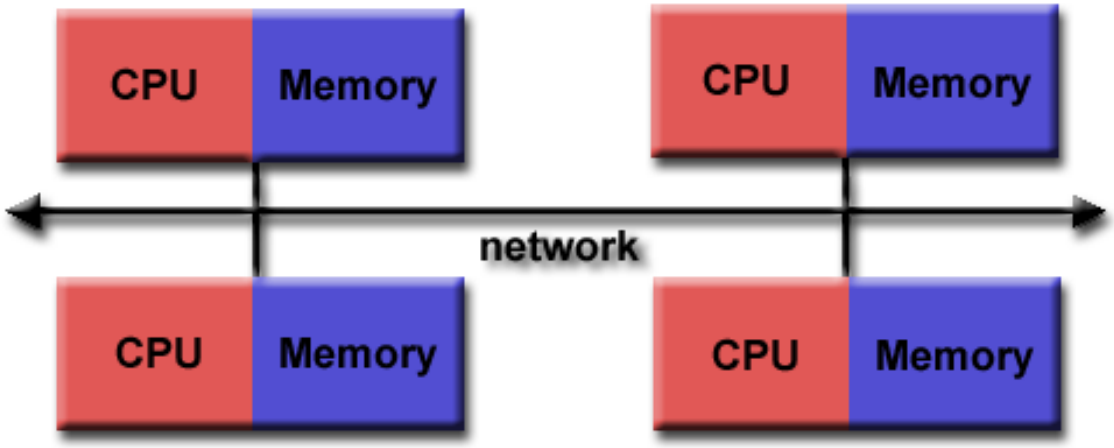
- Also, what are we looking at for overhead times?
  - Array size 10
  - Takes longer for parallel code to run

Parallel Computing with Examples (MPI)

Outline

- Distributed memory
- What is MPI?
- How is MPI used?
- Communicating
- Examples

Distributed-memory Model



Infiniband

Distributed memory requires a communication network to connect memory

Programmers explicitly define how processors access other processor's memory

Source: https://computing.llnl.gov/tutorials/parallel_comp/#ModelsShared

MPI

- MPI is a library specification for message passing
- Widely used standard
- Can run on shared, distributed, or hybrid memory models
- Exchange data between processes through communication between tasks – send and receive data
- MPI can get complicated
- Programmers must explicitly implement parallelism using MPI constructs
- Portable

General MPI Code Structure

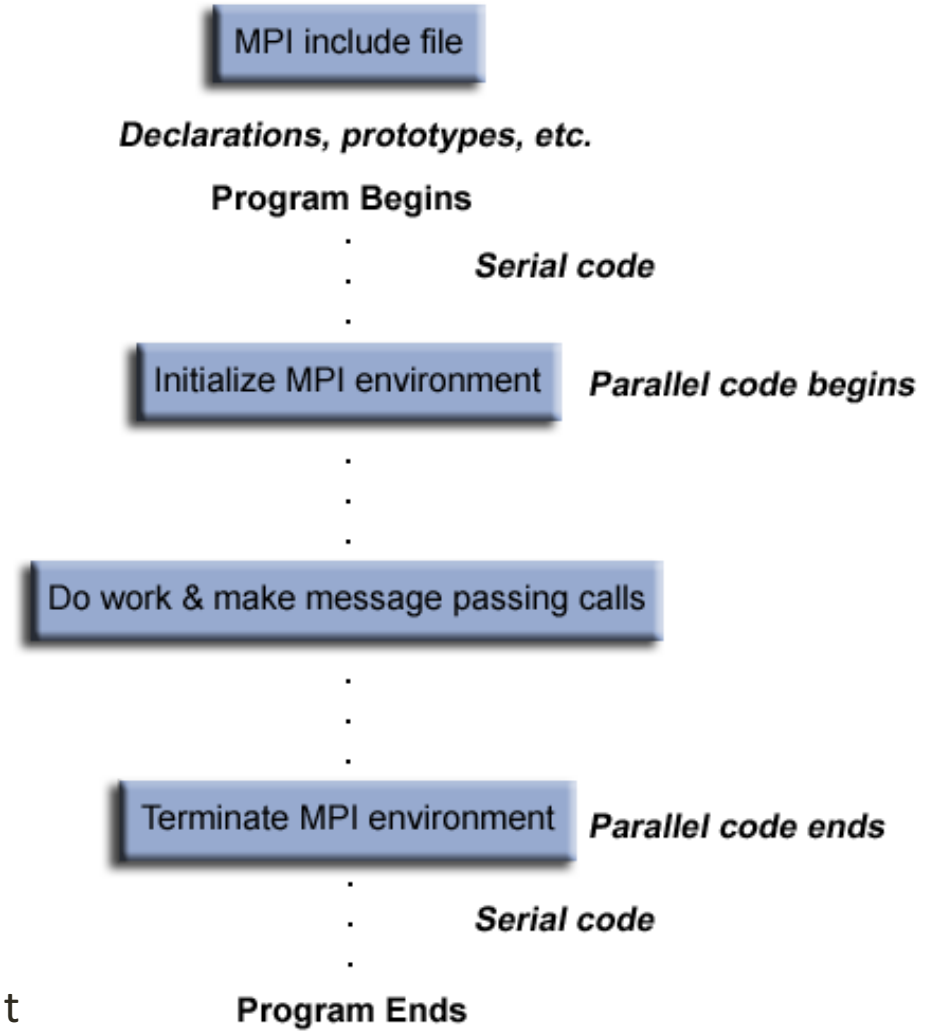
- You must have your header file at the top of any script you develop that uses MPI
- For C:

```
#include mpi.h
```

- For Fortran:

```
  use mpi
```

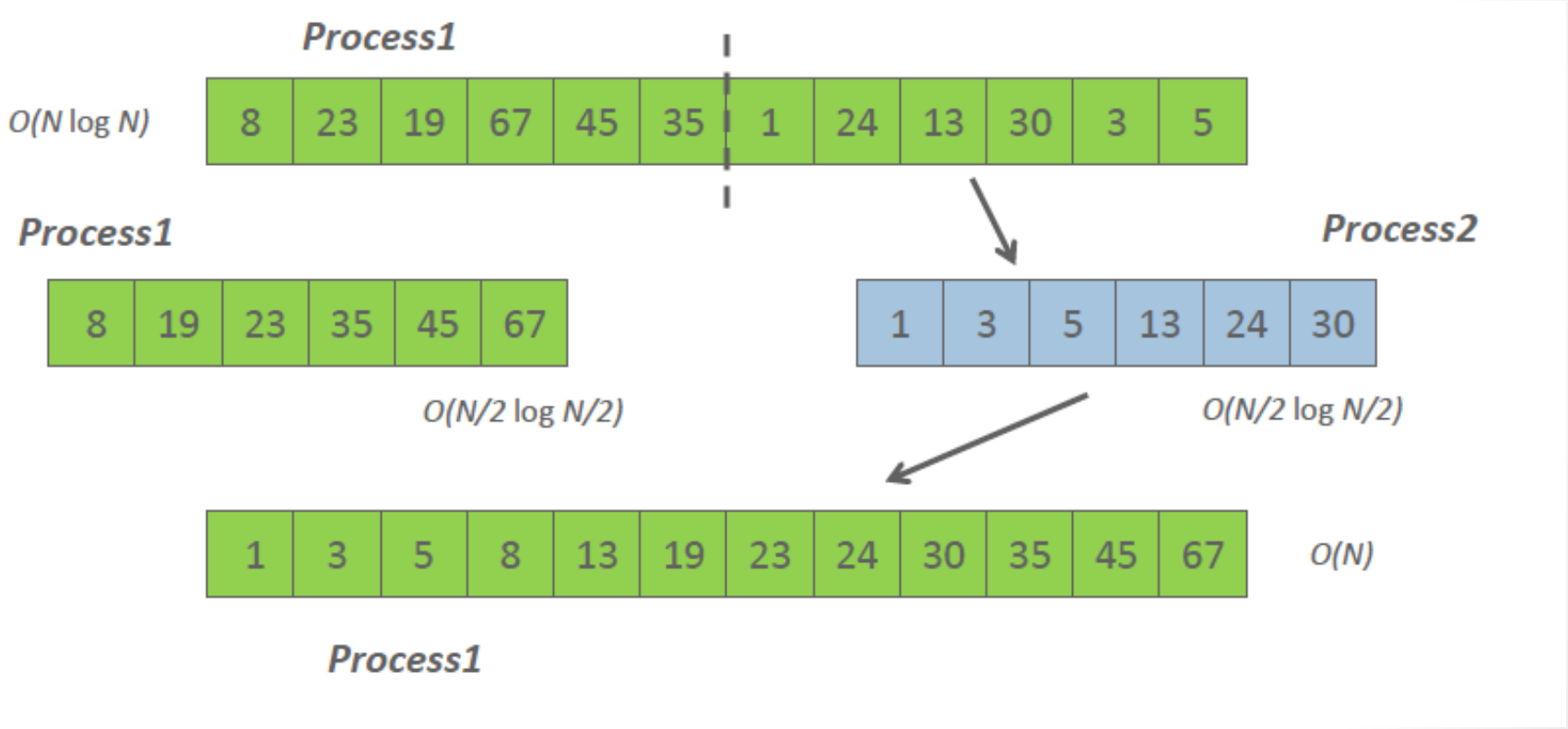
https://computing.llnl.gov/tutorials/mpi/#What



Message Passing

- A program that runs on a node is called a **process**
- When a program is run a process is run on each processor in the cluster
- These processes communicate with each other using message passing
- Message passing allows us to copy data from the memory of one process into another
- Message passing systems must at a minimum support system calls for sending and receiving messages

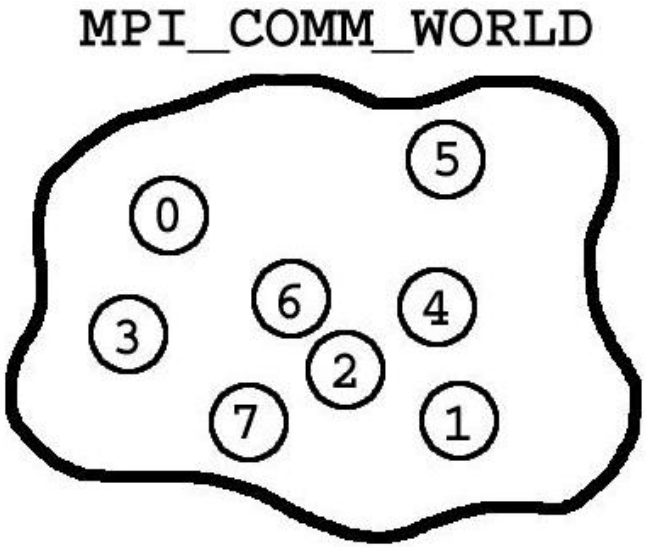
Example – Sorting Integers



http://htor.inf.ethz.ch/teaching/mpi_tutorials/ppopp13/2013-02-24-ppopp-mpi-basic.pdf

MPI Communicators

- Communicators used to group collections of processes allowed to communicate with each other
- Assigns integers to each process at initialization
  - Called “rank”
- Programmer uses rank to specify destination or source for sending/receiving
- Initially all processes grouped into MPI_COMM_WORLD



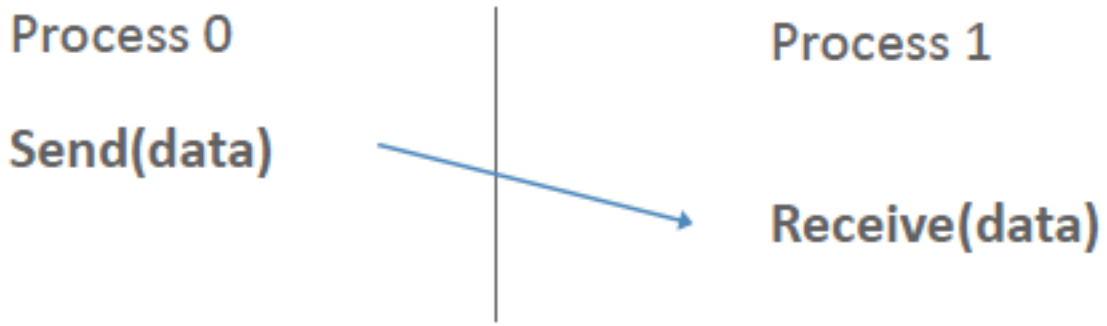
https://www.rc.usf.edu/tutorials/classes/tutorial/mpi/chapter2.html

Environment Management Routines

- These routines set the MPI execution environment, and cover many purposes
- Some common routines:
  - MPI_INIT
  - MPI_COMM_SIZE
  - MPI_COMM_RANK
  - MPI_FINALIZE

How Do I Write A Program in MPI?

- Application needs to specify:
  - How do you compile and run the MPI application?
  - How will the processes be identified?
  - How will the data be described?



http://htor.inf.ethz.ch/teaching/mpi_tutorials/ppopp13/2013-02-24-ppopp-mpi-basic.pdf

Compiling and Running an MPI Application

- MPI applications can be written in C, C++, or Fortran and appropriate calls to MPI can be added where required
- Compiling code:
  - Regular code:

```
gcc test.c –o test        ifort test.f -o test
```

  - MPI applications:

```
mpicc test.c –o test      mpifc test.f –o test
```

- Running code:
  - Regular code:

```
./test
```

  - MPI applications (running with 16 processes):

```
mpiexec –np 16 ./test
```

MPI Library on Janus

- Unlike OpenMP, with MPI you need to have the appropriate library loaded in your environment
- Research Computing recommends impi
- To load these, just type:

```
ml gcc
```

then

```
ml impi
```

At the command line

Compiling An Application

- Before compiling an application, you MUST:

- Include the MPI header file
  - Needed to use all the MPI Library calls
- Initialize the MPI environment
  - MPI_INIT()
- Specify an end to the MPI environment at end of program
  - MPI_Finalize()

Example Fortran Code

Fortran code: simple.f90

To run:

```
ml slurm
ml gcc
ml impi
sinteractive --reservation=anschutz
mpif90 simple.f90 -o simple
mpiexec -np 8 ./simple
```

OpenMP vs. MPI

Fortran code: hello.f90

The same code we ran as OpenMP modified for MPI

To run:

```
mpif90 hello.f90 -o hello
mpiexec -np 8 ./hello
```

Time?

Communication

- One process sends a copy of data to another process and that process receives it
- Requires the following information
  - Sender needs to know
    - Who to send the data to
    - What kind of data to send
    - A tag (like an email subject) so the receiver understands what's being sent
  - Receiver maybe needs to know
    - Who is sending the data
    - What kind of data is sending
    - The tag

MPI_SEND (Fortran)

- MPI_SEND(buf, count, datatype, dest, tag, comm, ierr)
- Basic sending operation
- Routine returns only after the application buffer in the sending task is free for reuse
  - In some sense, a send cannot complete without acknowledgment from the receiving process
  - Can be changed
  - Out of scope here

What does this mean?

- **Buffer**: Usually variable name that is to be sent/received
- **Count**: number of data elements of a particular type to be sent
- **Datatype:** pre-defined data type of data (MPI_CHARACTER, MPI_INTEGER, etc)
- **Dest:** destination – indicates the process where the message should be delivered. Sent as the rank of the receiving process

- MPI_SEND(buf, count, datatype, dest, tag, comm, ierr)

What does this mean?

- **Tag**: Arbitrary number assigned by the programmer to identify a message.
- **Comm:** communicator. Usually MPI_COMM_WORLD
- **Ierr:** error message

- MPI_SEND(buf, count, datatype, dest, tag, comm, ierr)

MPI_RECV (Fortran)

- MPI_REV(buf, count, datatype, source, tag, comm, status, ierr)

- **Status:** implies the source of the message
  - Integer array the size of MPI_STATUS_SIZE
- **Tag:** Can use MPI_ANY_TAG to receive any message regardless of tag

MPI Communication

Fortran code: ping.f90

To run:

```
mpif90 ping.f90 –o ping
mpiexec -np 8 ./ping
```

References

- https://portal.tacc.utexas.edu/c/document_library/get_file?uuid=c3c38847-ca7e-41bf-aefa-fb232a777699&groupId=13601
- https://computing.llnl.gov/tutorials/openMP/
- http://openmp.org/mp-documents/omp-hands-on-SC08.pdf
- https://computing.llnl.gov/tutorials/mpi/
- http://htor.inf.ethz.ch/teaching/mpi_tutorials/ppopp13/2013-02-24-ppopp-mpi-basic.pdf
- https://www.rc.usf.edu/tutorials/classes/tutorial/mpi/

Questions?

- Email rc-help@colorado.edu
- Twitter: @CUBoulderRC

- Link to survey on this topic: http://goo.gl/forms/8VidcwOhRT

- Slides: https://github.com/ResearchComputing/Final_Tutorials