How To Parallel Program

Shelley Knuth
shelley.knuth@colorado.edu

www.rc.colorado.edu Questions? #RC_BasicSC

Link to survey on this topic: http://tinyurl.com/rcpresurvey

Slides:
https://github.com/ResearchComputing/Final_Tutorials/tree/master/Basics_Supercomputing

Outline

- Parallel Computing with OpenMP
- Parallel Computing with Matlab

Parallel Computing with Examples (OpenMP)

Outline

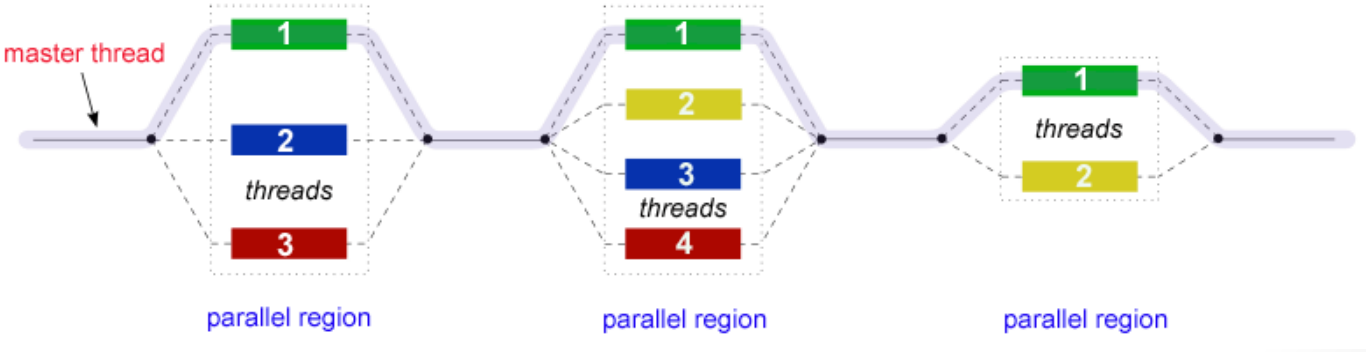
- Shared memory
- What is OpenMP?
- How is OpenMP used?
- Parallel region
- Public/Private variables
- Examples

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 – 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



Source: https://computing.llnl.gov/tutorials/openMP/#Introduction

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

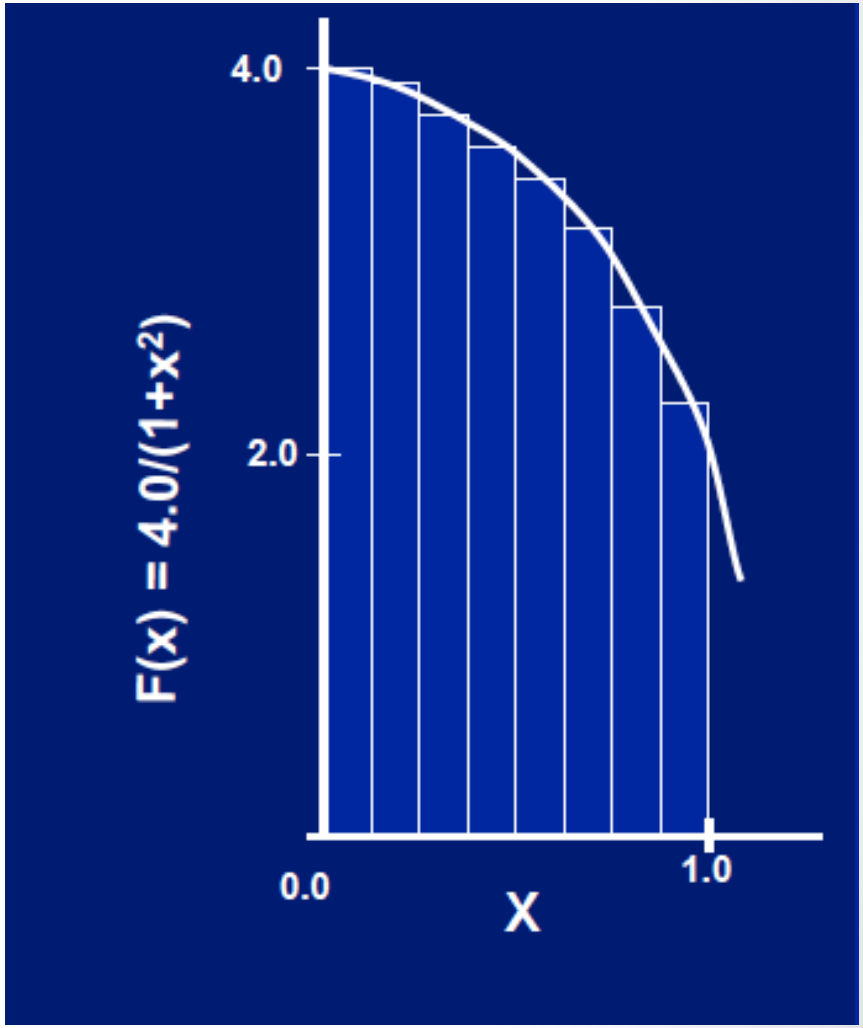
- When thread hits PARALLEL directive, creates team of threads
  - Becomes master
  - Code is duplicated and all threads execute that code
  - Runs the same code on different data
    - Split up loops and operate on different data
  - Only master thread continues after implied barrier
- Can determine 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

Parallel Region Example

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

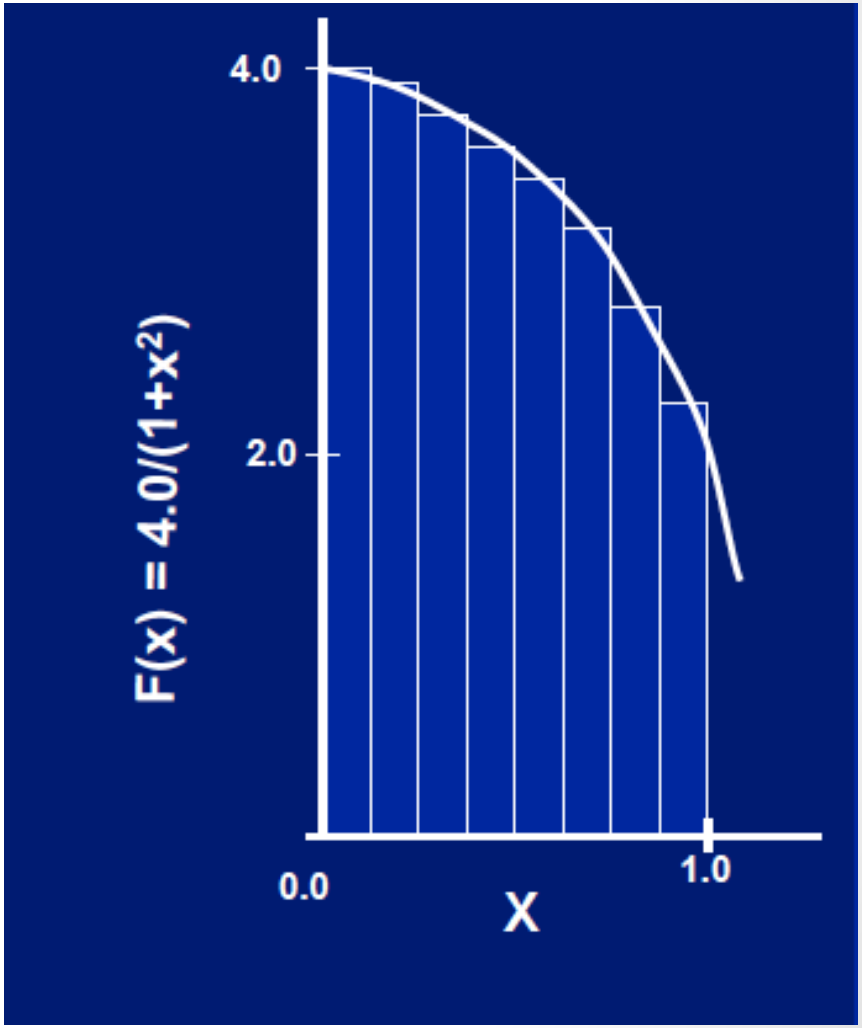
$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

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



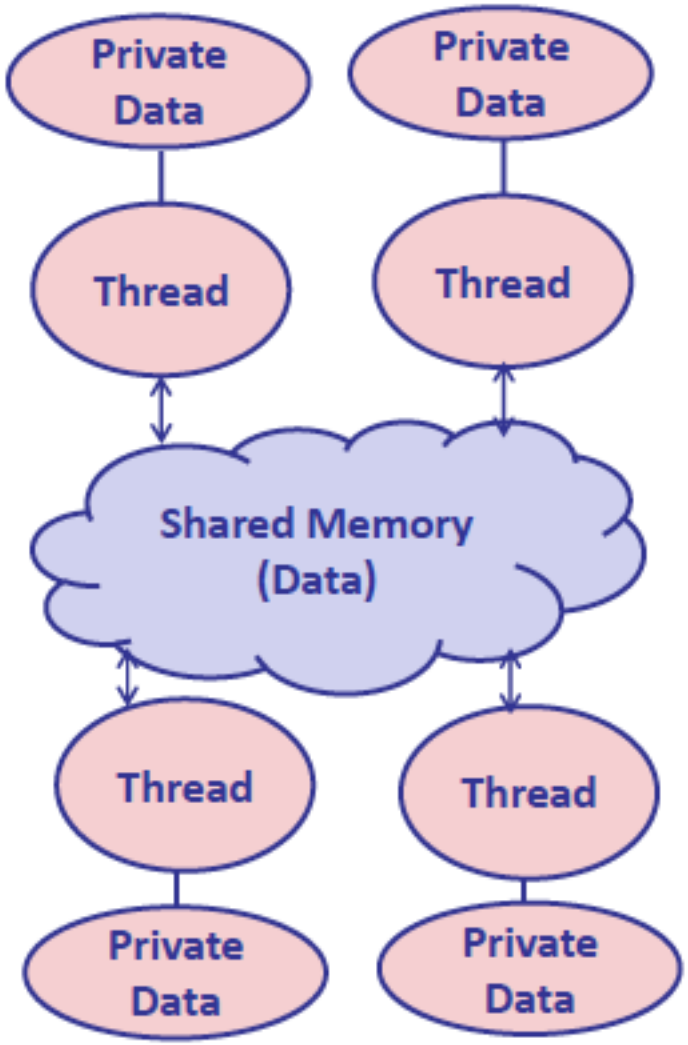
Parallel Region Example

- 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



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

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)**

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	-qopenmp
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

OMP Code Practice – Exercise 1

- Code:

  `omp_hello.f`

  `omp_hello.sh`

- Instructions for running:

```
ssh tutorial-login.rc.colorado.edu -l user00XX
ml slurm
sbatch omp_hello.sh
```

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`

  `for.sh`

- Instructions for running:

```
ssh tutorial-login.rc.colorado.edu -l user00XX
ml slurm
sbatch for.sh
```

Example Code – Exercise 2

- We need to consider whether our code really does experience a speed up
  - Array size 10,000,000
  - Drops by ~30-50%

- Let's see what happens when we change our array size to 10
  - Takes longer for parallel code to run
  - Overhead is more of a factor

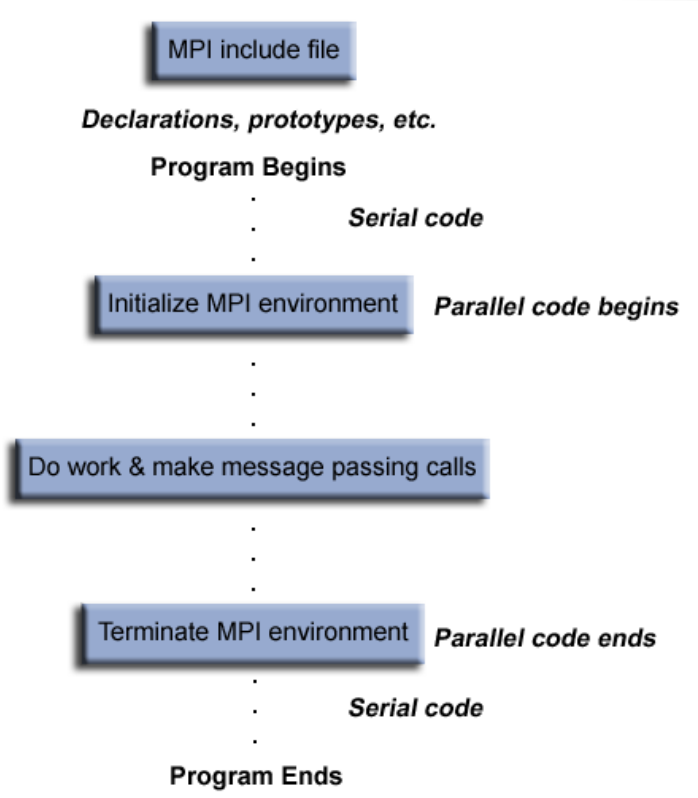
Parallel Computing with Examples (MPI)

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

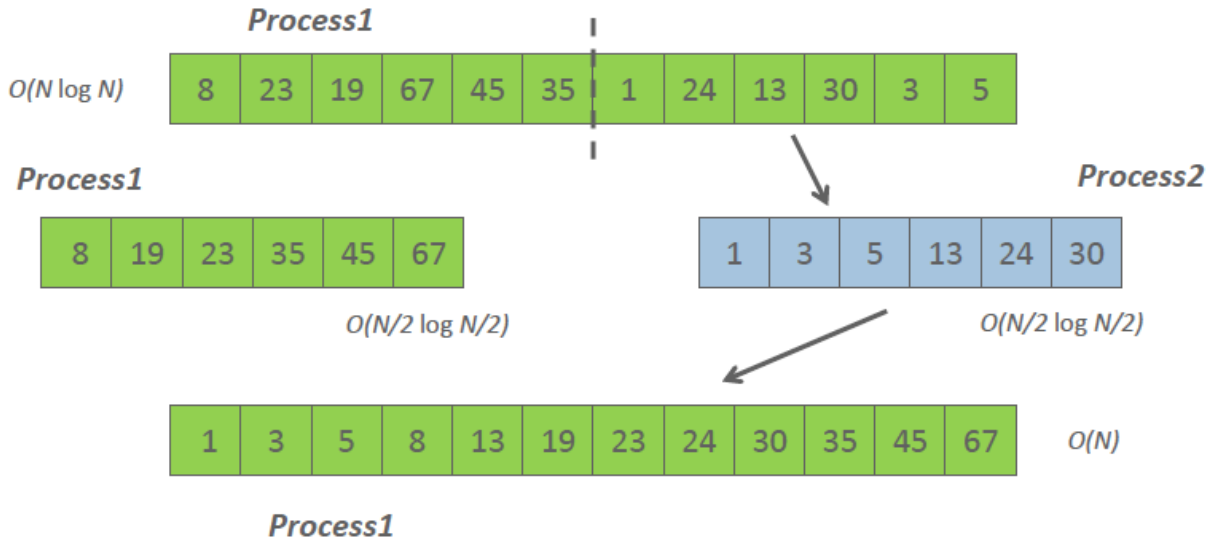


https://computing.llnl.gov/tutorials/mpi/#What

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

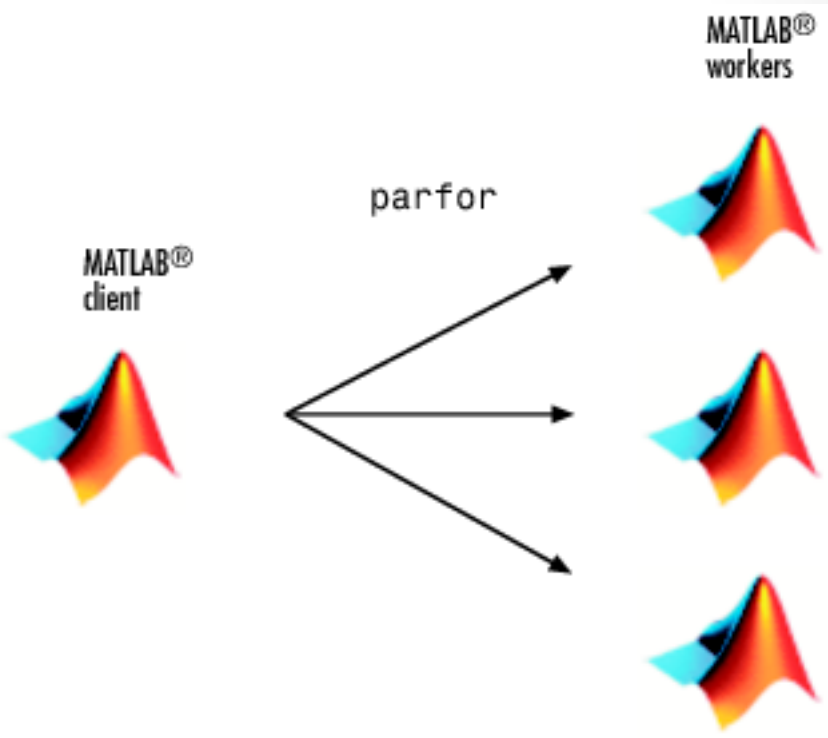


http://htor.inf.ethz.ch/teaching/mpi_tutorials/ppopp13/2013-02-24-ppopp-mpi-basic.pdf

Parallel Computing with Examples (Matlab)

Running Matlab in Parallel

- **Workers:** copies of the original client created to assist in computation



Parallel Computing Toolbox (PCT)

- Additional toolbox as part of Matlab
- Perform parallel computations on multicore computers, GPUs, and computer clusters
- Many Matlab functions work in concert with the PCT
- Simple to utilize with just the use of certain commands

Parallel and Not Parallel

Not Parallel:

```
for i=1:10

   x=x(i)+1;

end
```

Parallel:

```
matlabpool open 4
   parfor i=1:10
      x=x(i)+1;
   end
matlabpool close
```

parfor

- Easy to use
- Allows parallelism in terms of loops
- When client reaches a parfor loop iterations of loop are automatically divided up among workers
- Parfor requires results be completely independent
- Cannot determine how loops are divided

Running Matlab in Parallel On Lots of Cores

- Typically see a significant speed up when using parfor vs. when not
  - If code is parallelizable
- However, this might not always be the case
- Might spend more time in overhead
  - If code isn't parallelizable
  - If code isn't that complicated

Running Matlab in Parallel

- Let's take ordinary code that is already running and convert it to run in parallel

- matlab_parallel_serial.m
- matlab_parallel_tutorial.m

Spmd Command

- Single process, multiple data
- The spmd command ensures more control
- Can parallelize much more than just loops
- Like a very simplified version of MPI
- More flexibility than parfor
- However, need to know what you're doing

Distributed Computing Toolbox

- PCT allows you to run programs in parallel across many processors
- DCT allows you to run across nodes
  - Allows you to run easily on clusters
  - Supports resource managers
  - Not an option for Janus

Other Options

- Parallel R: https://earthlab.github.io/r/R-parallel_r/
- Parallel Python: http://materials.jeremybejarano.com/MPIwithPython/

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
- http://heather.cs.ucdavis.edu/ParallelR.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/

Now what?

- Get an account on Janus!
- Email rc-help@colorado.edu for any help!
- Join our meetup group! https://www.meetup.com/University-of-Colorado-Computational-Science-and-Engineering/
- Email shelley.knuth@colorado.edu if you want to be added to a new email list about upcoming workshops!
- Fill this out! http://tinyurl.com/curc-survey16

Questions?

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

- Link to survey on this topic:

http://tinyurl.com/curc-survey16

- Slides: https://github.com/ResearchComputing/Final_Tutorials/tree/master/Basics_Supercomputing