Parallel Computing with Examples (OpenMP)

Shelley Knuth, Research Computing, University of Colorado-Boulder

shelley.knuth@colorado.edu

Questions? #RC_Meetups

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

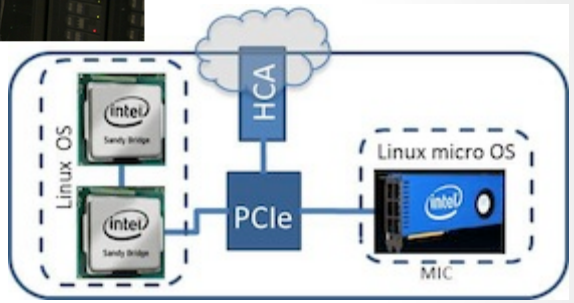
Slides: https://github.com/ResearchComputing/Final_Tutorials

Outline

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

Programming to Use Parallelism

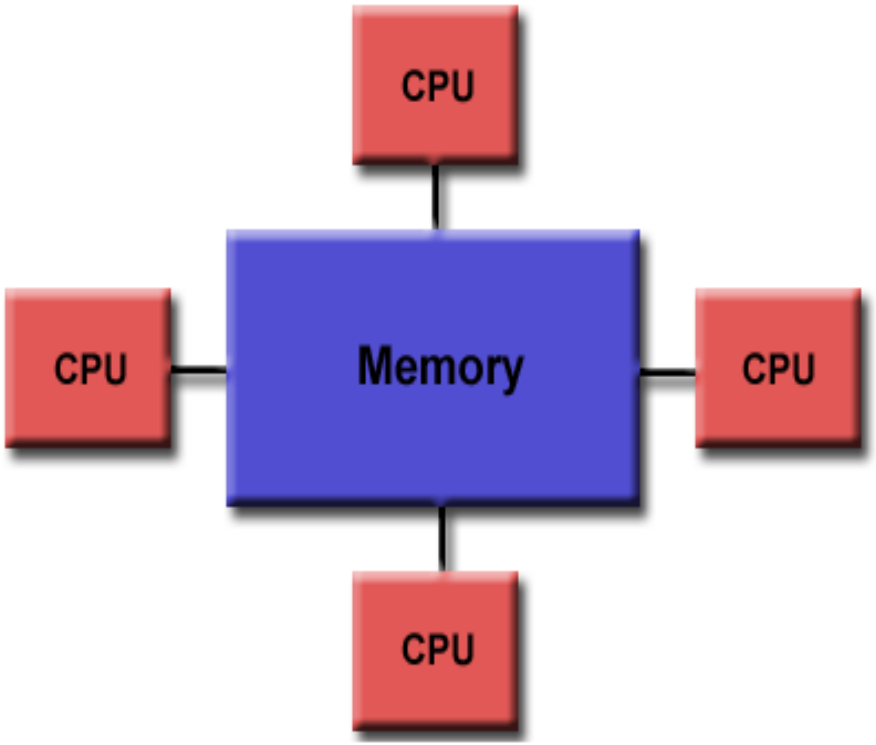
- Parallelism across processors/threads - OpenMP



- Parallelism across multiple nodes - MPI

www.scan.co.uk

Shared-memory Model

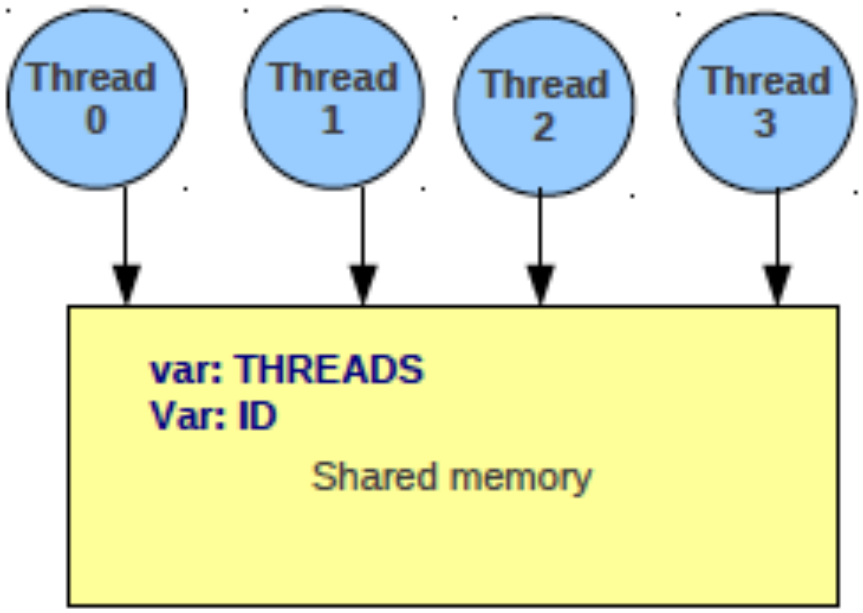


The concept is that all processors can access all memory available

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

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

Shared-memory Model



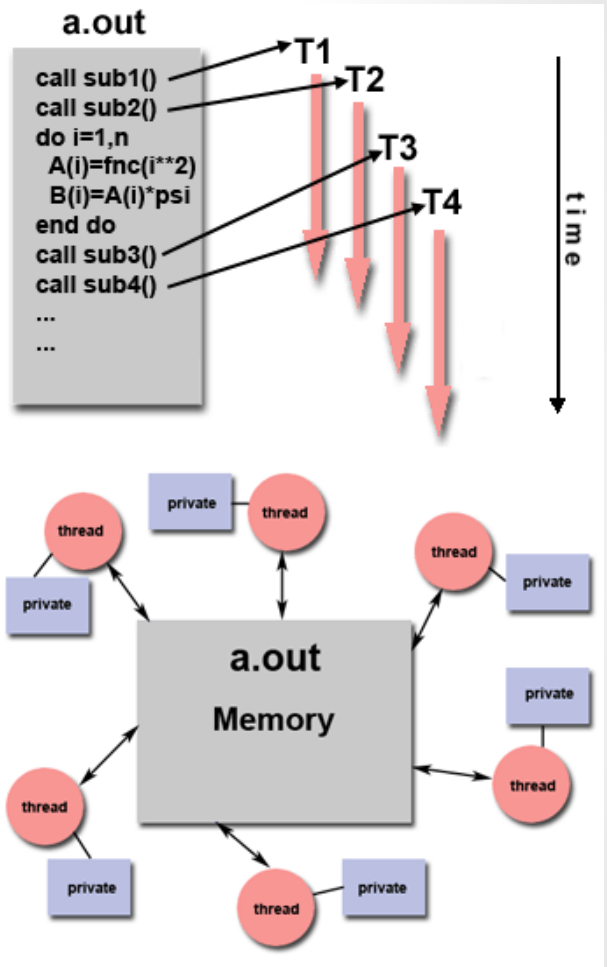
A thread is a block of code with one entry and one exit that is abstract and is mapped onto a physical core. Multiple threads can be mapped onto one core.

Threads communicate by depositing contents in shared memory area

Source: http://people.math.umass.edu/~johnston/PHI_WG_2014/OpenMPSlides_tamu_sc.pdf

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



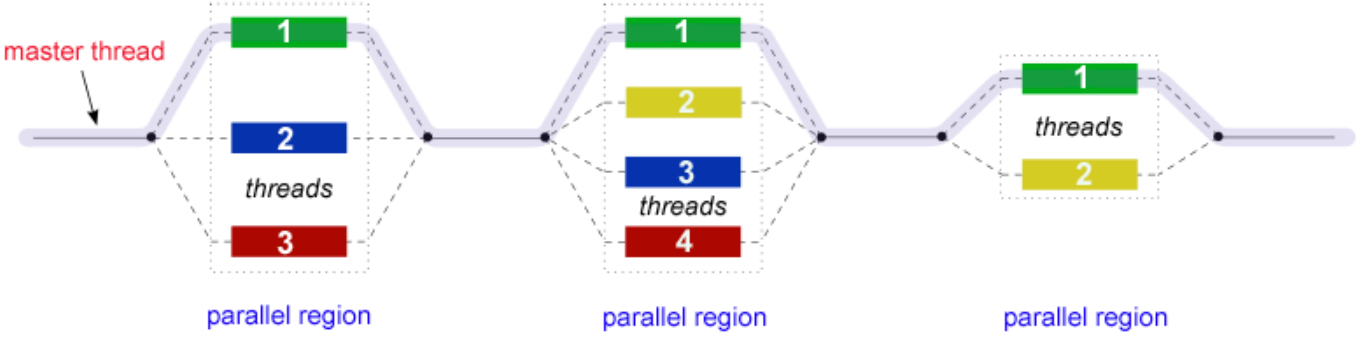
Source: https://computing.llnl.gov/tutorials/parallel_comp/#ModelsShared

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).
```

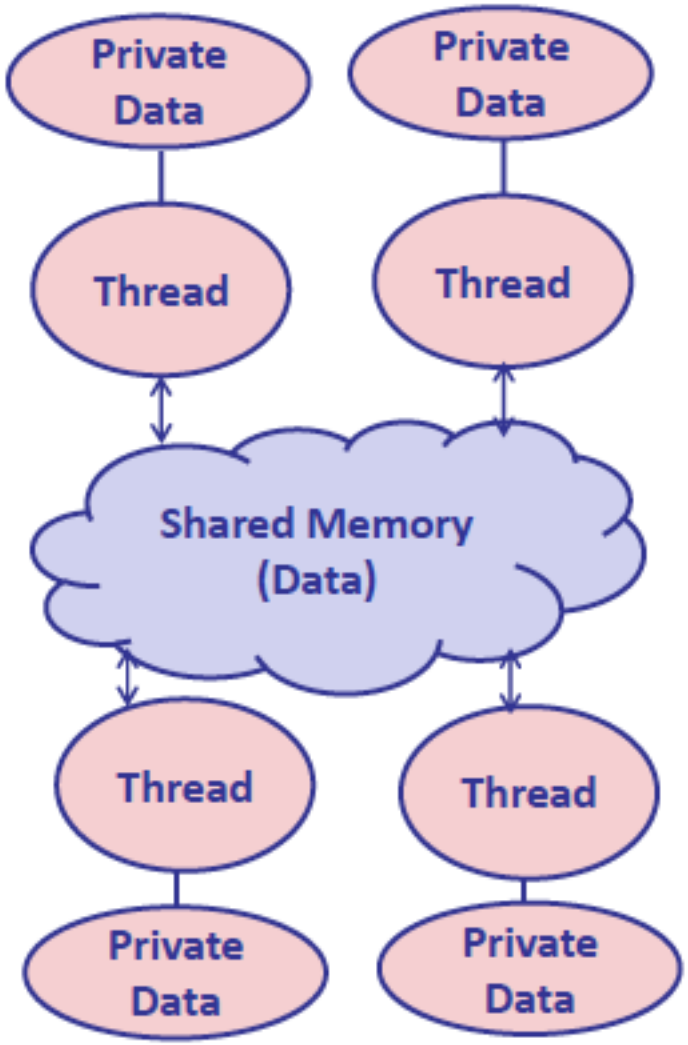
```
Use the thread number to divide work among threads.
```

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

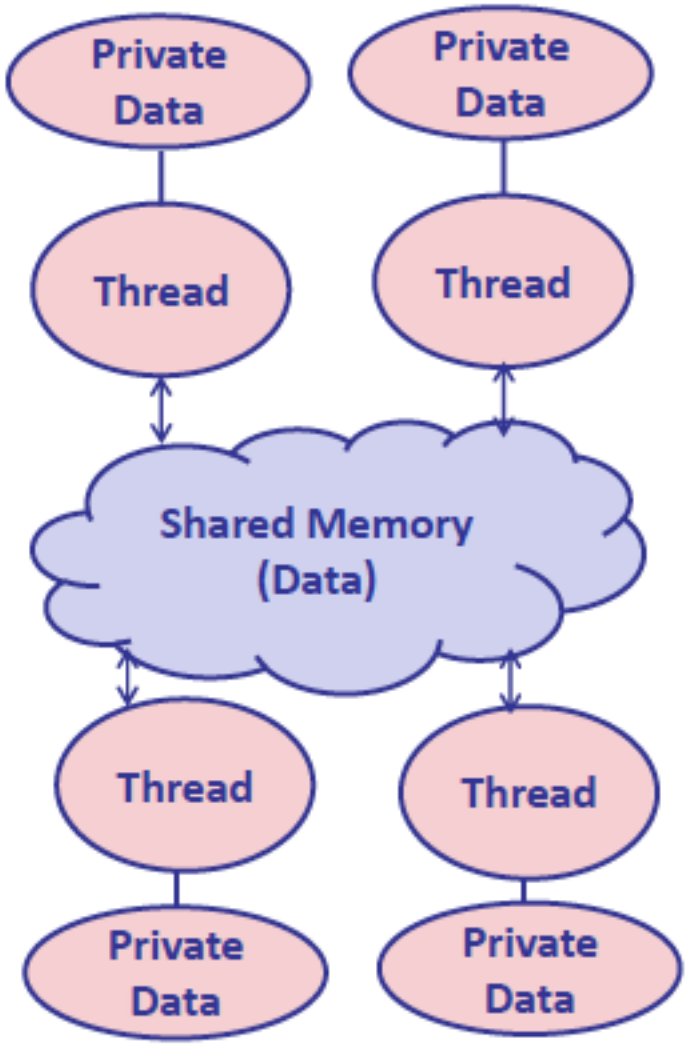
Shared and Private Variables

- When specifying the **PRIVATE** clause, that variable is private to each thread
  - Each thread will have its own unique copy
  - Can only be accessed by the threads that own it
- If there are x team members, there are x+1 copies of the variables in the private clause
  - One global copy and a private copy for each team member
- When specifying **SHARED** clause, all threads can access that data



Shared and Private Variables

- Global variables are shared by default
- Variables declared within subroutines called within private region are by default private
  - Index variables are also default private
- Can change specifications with **DEFAULT** clause



Shared and Private Variables

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

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

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);
      }
```

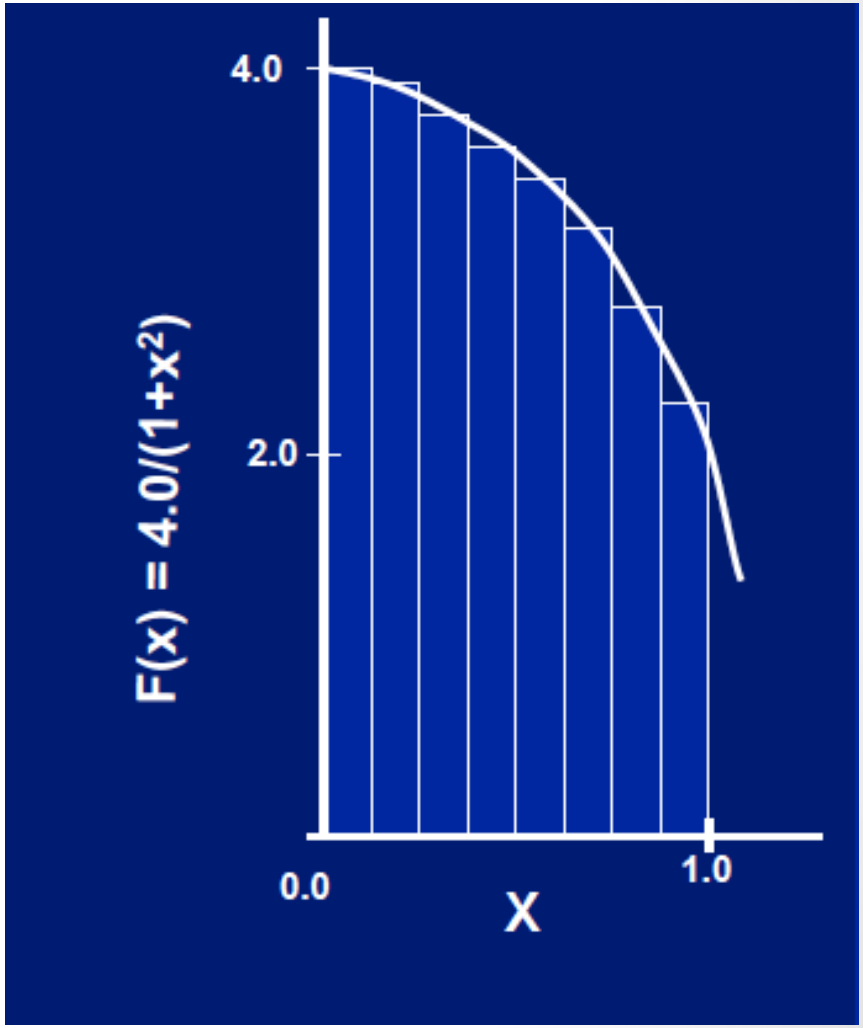
- Variable temp also needs to be private
- Otherwise each thread would be reading/writing to same location

Parallel Region Example

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

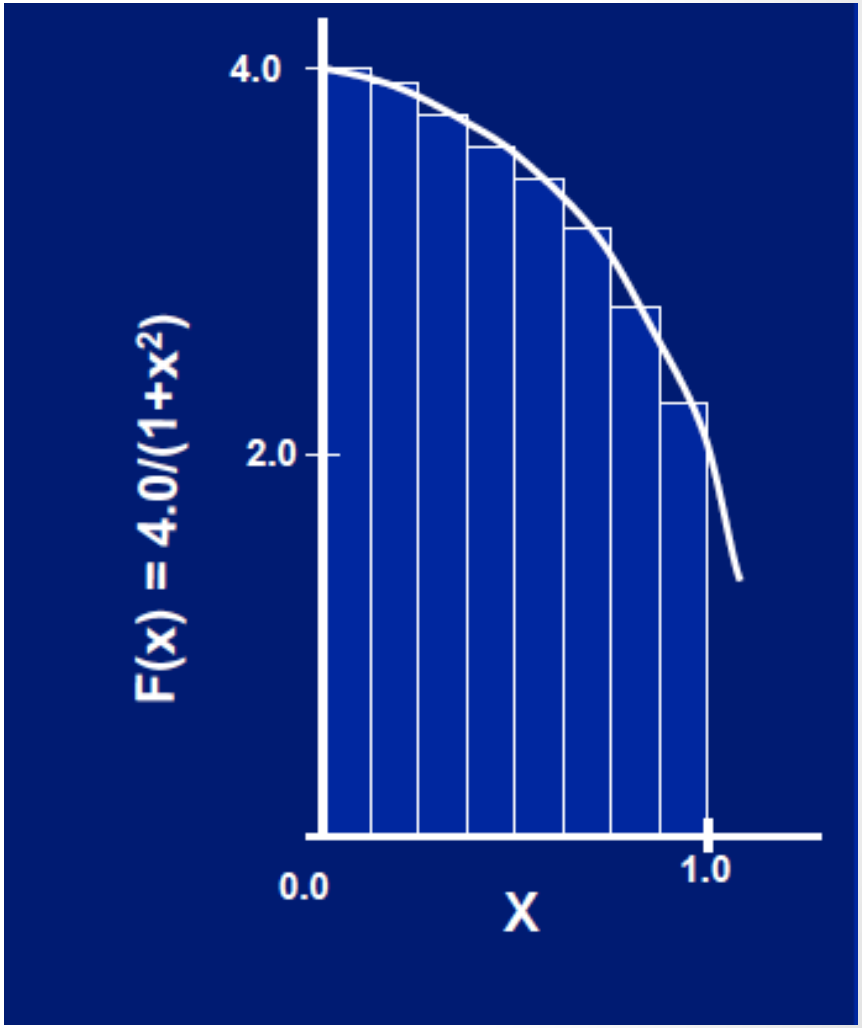
$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

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



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 Fortran some routines are functions; some are subroutines
- In C/C++, all are subroutines. 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 gcc/5.1.0
ml slurm
sinteractive --reservation=meetup
gfortran -fopenmp 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 gcc/5.1.0
ml slurm
sinteractive –reservation=meetup

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

gcc -fopenmp 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

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/#Introduction
- http://openmp.org/mp-documents/omp-hands-on-SC08.pdf
- https://computing.llnl.gov/tutorials/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

- Questions? #RC_Meetup