A short parallel computing class for C/C++

Andrew W. Steiner

UTK/ORNL

July 6, 2023

Admin

- Logins on bridges2.psc.edu and neutrino.desktop.utk.edu
- Slack channel
- Github repository: https://github.com/awsteiner/parallel

Outline

- Makefiles and Environment Variables
- HPC Filesystems
- OpenMP or Shared Memory Parallelism
- Group Exercises

Makefiles and Environment Variables I

- Terminal is a GUI, plus a "shell"
- Interaction between the user and the operating systems
- I will often assume bash, but the default on MacOS is zsh
- Determine your shell using e.g. `ps`
- Setting and displaying environment variables depends on which shell you use
- On bash, show your environment variables using `env` and set using e.g. `export VAR=value`
- Particularly important environment variables: `PATH, LD_LIBRARY_PATH, EDITOR, CC, CXX, OMP_NUM_THREADS, DISPLAY, PS1, USER, SHELL, TERM, LSCOLORS, HOME`
- Common pattern: `env | grep -i path`

Makefiles and Environment Variables II

- Basic structure

```
target: dependencies
    build rules
```

- e.g.

```
code: code.o
    $(CXX) -o code code.o

code.o: code.cpp code.h
    $(CXX) -o code.o -c code.cpp
```

Makefiles and Environment Variables II

- Notice how these targets uses environment variables?
- Make creates a "subshell" and executes commands within that subshell (defaults to `sh`)
- @ prefix doesn't echo commands
- - prefix allows commands to fail
- Variables referred to by `$( )`
- Need two $$'s for one $
- Makefile is picky about tabs

Makefiles and Environment Variables III

- Two tricks:
  - Variable as the result of a shell command
  - Foreach

HPC Filesystems

- System configurations vary widely: always read the HPC documentation
- Can be significantly different compared to logging into a standard Unix computer
- Login nodes and compute nodes
- Networked file systems: user directories and project directories
- Not all file systems are accessible from both the login and the compute nodes
- Frequently, output files must be stored on a scratch filesystem, and copied to a user (or project) directory later

HPC Filesystems II

- Login nodes are shared among many users (and thus slow), so be careful
  - No computation, especially no parallelism
  - No parallel compiling with "make -j"
  - No time-consuming tasks (e.g. no `ls -lR /`)
- File I/O can be the **most time-consuming part of code execution**.
- Often, Globus is preferred over ssh for large file transfers

Modules

- Code often needs to access libraries, but different users need different libraries
- Some of these libraries require different compilers or environment variables
- In most systems, this is handled by the **`module`** command
- **`module load`**, **`module purge`** **`module help`**
- Unfortunately, many of these modules are poorly documented

Parallel Computing

- Often it is better to use previously-developed code
- But even then, it is better to understand the concepts behind the code you are using

OpenMP or Shared Memory parallelism

- Memory space shared between "threads"
- OpenMP, POSIX threads, Boost threads, etc.
- Relatively small code changes
- Can lead to very confusing behavior

- Requires compiling with, e.g. `-fopenmp` and `#include <omp.h>`
- Danger zones:
  - Multiple threads writing and reading the same location at the same time
  - Multiple threads waiting for each other to complete ("deadlock")
  - Algorithms which depend sensitively on the order of operations ("race conditions")

OpenMP or Shared Memory parallelism II

- Simple parallel region:

```
#include <omp.h>
#pragma omp parallel
{
  #pragma omp for
  for(size_t i=0;i<100;i++) {
    // ... do something 100 times
  }
}
```

OpenMP or Shared Memory parallelism III

- Determine the number of threads: `omp_get_num_threads()`
- Determine the current thread: `omp_get_thread_num()`
- These latter two functions are only well-defined inside a parallel region
- By default, all variables in the work sharing region are shared except the loop iteration counter.
- "Some variables need to be shared, some need to be private, and you the programmer have to specify what you want."

OpenMP or Shared Memory parallelism IV

- Accessing shared memory:
    - Reading memory locations at the same time from multiple threads is typically (but not always) safe
    - Writing memory locations at the same time from multiple threads is typically (but not always) unsafe

- A simple paradigm is to perform a difficult parallel calculation and store the results for every OpenMP thread
- and then to perform a simple calculation to sort or process those results afterwards in a non-parallel region

OpenMP or Shared Memory parallelism V

- In C/C++, care must be taken to understand if constructors and destructors are inside or outside parallel regions
- It is dangerous to create an object inside a parallel region, and destroy it outside the parallel region
- Not all objects are "thread-safe"
- Reading is often thread-safe, but writing is rarely thread-safe
- You can use const references to help write thread-safe code
- `omp_test*.cpp` example

OpenMP or Shared Memory parallelism VI

- Special regions inside parallel regions:
  - critical: the enclosed code block will be executed by only one thread at a time, and not simultaneously executed by multiple threads. It is often used to protect shared data from race conditions.
  - atomic: the memory update (write, or read-modify-write) in the next instruction will be performed atomically. It does not make the entire statement atomic; only the memory update is atomic. A compiler might use special hardware instructions for better performance than when using critical.
  - ordered: the structured block is executed in the order in which iterations would be executed in a sequential loop

OpenMP or Shared Memory parallelism VII

- Special regions inside parallel regions:
  - barrier: each thread waits until all of the other threads of a team have reached this point. A work-sharing construct has an implicit barrier synchronization at the end.
  - nowait: specifies that threads completing assigned work can proceed without waiting for all threads in the team to finish. In the absence of this clause, threads encounter a barrier synchronization at the end of the work sharing construct.

OpenMP or Shared Memory parallelism VIII

- The for loop cannot exit early, for example:

```
// BAD - cannot parallelize with OpenMP
for (int i=0;i < 100; i++) {
  if (i > 50)
    break;
}
```

OpenMP or Shared Memory parallelism IX

- Values of the loop control expressions must be the same for all iterations of the loop.

  For example:

```
// BAD - cannot parallelize with OpenMP
for (int i=0; i < 100; i++) {
  if (i == 50)
    i = 0;
}
```

- Threads that exceed their stack allocation may or may not seg fault. An application may continue to run while data is being corrupted.

Group Exercises

- Create a OpenMP parallel program, including makefile targets, to compute Pascal's triangle (using addition) to a user-specified number of rows, and output the result to a file at the end.
- Ensure that, at some point during the execution, the program prints out the current number of OpenMP threads
- Go through Project #1, and write down how OpenMP parallelism is used in anneal_para.h.
- Is there a place where the OpenMP parallelism should be changed? What is the overhead associated with the OpenMP parallelism in this code?
- In Project #1, propose a rewrite of the makefile to use OpenMP parallelism and demonstrate its use
- Expand Project #1 to minimize a more difficult function in 10 dimensions in which all 10 variables are constrained to be between 0 and 1

References

- Some material taken from: bowdoin.edu/~ltoma
- A good reference is at LLNL