

Phys 7411: Debugging Parallel Programs

Karen Tomko and Juana Moreno

ktomko@osc.edu or moreno@phys.lsu.edu

Spring 2009

Today's topics

- Debugging basics
- Complications due to concurrency
- Parallel debugging tools
- References:
 - Chap. 5, Kernighan and Pike, *The Practice of Programming*, Addison Wesley 1999.
 - Chap. 15, Dongarra et. all, *Sourcebook of Parallel Computing*, Morgan Kaufmann/Elsevier, 2003.



Bug

bug *noun*

2. a fault or defect in a computer program, system, or machine

WordNet® 3.0, © 2006 by Princeton University.

Bug prevention

- Program defensively
 - Check return codes/error codes
 - Handle conditions that shouldn't happen, such as invalid data values
- Write and test code incrementally
- Implement and test simple cases first

Tracking Down Bugs

- Debugging is fun
 - Like solving a puzzle or a mystery
- Don't
 - Blame the compiler? computer? library, etc..
the bug is likely yours
 - Panic
- Do
 - Use what you know about the erroneous result to try to infer possible problems
 - Be methodical/systematic in your investigation

Compiler flags that help

- Bounds checking
 - Mbounds[†]
- Generate symbolic information for debugger tools
 - g or -gopt[†]
- Verbose output, helps track down link problems
 - v[†]
- Lint (a C static analysis tool) like flags
 - Wall[‡]

[†] PGI compiler flag

[‡] gcc compiler flag

Guidelines from Kernighan and Pike

- Look for familiar patterns (have I seen this kind of problem before)
- Examine the most recent change
- Don't make the same mistake twice (cut and paste)
- Debug it now not later
- Look at the stack trace
- Read/think before typing
- Explain your code to someone else
- Make the bug reproducible
- Divide and conquer
- Study the numerology of failures (is there a numerical pattern)
- Display output to localize your search
- Write self-checking code
- Write a log file
- Draw a picture (plot of data)
- Use tools (diff, revision control, ...)
- Keep records

Parallel Computing Complications

- Race conditions and deadlock
- Large data sizes
- Long running times
- Lack of interactivity
- Repeatability challenges

Parallel debugging tool chest

- All of the prior guidelines still apply
- Print statements
- Data visualization / Data checks
- Parallel debuggers
- MPI tracing tools

Print statements

- Good for quickly narrowing down the location, how far does my code get before it crashes/hangs (or look at the stack trace)
- Print processor rank or thread id at the beginning of each printed line
- Warning – be careful what you infer from ordering of writes from different threads/processes

Data checks / visualization

- Add tests in your code for anomalous values or zeros
- Compare results or intermediate arrays from sequential code with the same arrays from the parallel code (diff family of tools are useful here, see: ndiff, spiff, sdiff or Kompare)
- Look specifically at data which is transferred or shared between processors or threads
- Trace back from bad result

Parallel debugging tools on Glenn

- pgdbg from the Portland Group
 - OpenMP threads
 - Login nodes or interactive batch session
- totalview
 - MPI or OpenMP
 - Works within batch system, interactive batch session
 - OSC “Using Glenn” training handouts
- http://www.osc.edu/supercomputing/training/opt/opteron_mod2_0801.pdf (slides 28-32)

MPI Message tracing

- Jumpshot

- Log of each MPI call made in your program
- OSC “Using Glenn” training handouts

http://www.osc.edu/supercomputing/training/opt/opteron_mod2_0801.pdf (slides 44-46)