Debugging OpenMP programs

Sergey Mashchenko
(SHARCNET / Compute Ontario / Compute Canada)

Outline

- Introduction
- Overview of DDT
- OpenMP debugging demo
- Questions?

Introduction

Parallel vs. serial

- Parallel programming is more difficult than serial programming each step of the way:
  - Designing stage
  - Coding
  - Debugging
  - Profiling
  - Maintenance

Parallel bugs

- In addition to usual, “serial” bugs, parallel programs can have “parallel-only” bugs, such as
  - Race conditions
    - When results depend on specific ordering of commands, which is not enforced
  - Deadlocks
    - When task(s) wait perpetually for a message/signal which never come

Race condition

- In OpenMP, race conditions result from misuse of shared variables, when
  - a variable is mistakenly labeled as shared (where in fact it needs to be private), or
  - a variable is correctly labeled as shared, but the access to the variable wasn't properly protected (serialized)
- The risk of having an issue of the first kind can be greatly reduced if one resorts to always use "default(none)" clause in OpenMP pragmas.
- Another risk factor is the "nowait" clause; if in doubt, test your code with all "nowait" clauses removed, to see it fixes the issue.

Race condition (cont.)

- Race condition manifests itself as wrong and variable code results. (You get different results every time you run the code, or only for some runs, and when you change the number of threads.)
- As only shared variables are at risk of creating race conditions, use them sparingly (only when truly necessary), and pay a lot of attention to them during debugging.

Example of race condition

```
#pragma omp parallel sections shared(a,b,c)
#section
a = b + c;
#section
b = a + c;
#section
c = b + a;
```

Deadlocks

- It happens when thread(s) lock up while waiting on a locked resource that will never become available.
- The sign of a deadlock: the program hangs (always or sometimes) when reaching a certain point in the code.

Deadlocks (cont.)

- Prevention strategies:
  - Be very careful with conditional clauses using threadID as an argument, as common OpenMP constructs (for/do, single) require all the threads in the team reaching them.
  - Communications between threads (using a shared variable) have to use “flush” pragma, on both writing and reading sides.
  - Don't forget to unset locks after setting them.

Example of a deadlock

```
#pragma omp parallel sections
#section
{omp_set_lock(&locka);
omp_set_lock(&lockb);
omp_unset_lock(&lockb);
omp_unset_lock(&locka);}
#section
{omp_set_lock(&lockb);
omp_set_lock(&locka);
omp_unset_lock(&locka);
omp_unset_lock(&lockb);}
```

Tools

- Debugging of codes (including parallel ones) could be as primitive as inserting multiple printf statements.
- But given the extra difficulty of dealing with parallel code issues, debugging and profiling of parallel codes better be done using proper tools.

Tools (cont.)

- SHARCNET has two tools suitable for OpenMP debugging installed on multiple systems:
  - DDT: commercial serial/parallel (MPI, OpenMP, CUDA) debugger which includes a memory debugger. Suitable for all programmer levels (from a beginner to an expert).
  - VALGRIND: powerful open source memory debugger. Mostly for advanced/expert programmer levels.

Allinea software

- For the rest of this webinar, I will focus on advanced parallel debugging tool developed by Allinea and installed on multiple SHARCNET clusters (orca, monk, kraken etc.), DDT.
- For detailed information on how to use DDT on our clusters, check this wiki page:

  https://www.sharcnet.ca/help/index.php/DDT

Overview of DDT

Intro

- The Distributed Debugging Tool (DDT) is a powerful commercial debugger with a graphical user interface (GUI).
- It is designed for debugging parallel programs (MPI, OpenMP, CUDA), though it can also be used with serial codes.
- The product was developed by Allinea (U.K.). It is installed on many SHARCNET clusters (orca, monk, kraken, requin).

Intro (cont.)

- DDT supports C, C++, and Fortran 77 / 90 / 95 / 2003.
- Detailed documentation (the User Guide) is available as a PDF file on clusters where DDT is installed, in

  /opt/sharcnet/ddt/*/doc

- We also have an online tutorial:

  https://www.sharcnet.ca/help/index.php/Parallel_Debugging_with_DDT

Preparing your program

The code has to be compiled with the switch **`-g`**, which tells the compiler to generate symbolic information required by any debugger. Normally, all optimizations have to be turned off. For example,

```
f90 -g -O0 -openmp -o code code.f

cc -g -O0 -openmp -o code code.c
```

Launching DDT

- All debugging should be normally done on cluster development nodes (orc-dev1...orc-dev4 on orca, kraken-devel1...kraken-devel8).
- As DDT uses GUI, your computer has to have an X window client. It comes bundled with Linux; for Windows use free program MobaXterm; for Macs use free program XQuarz. See for more details:

  https://www.sharcnet.ca/help/index.php/Remote_Graphical_Connections

Launching DDT (cont.)

- First ssh to a cluster, then ssh to a development node. Use “-Y” argument with all your ssh commands, to properly tunnel X11 traffic:

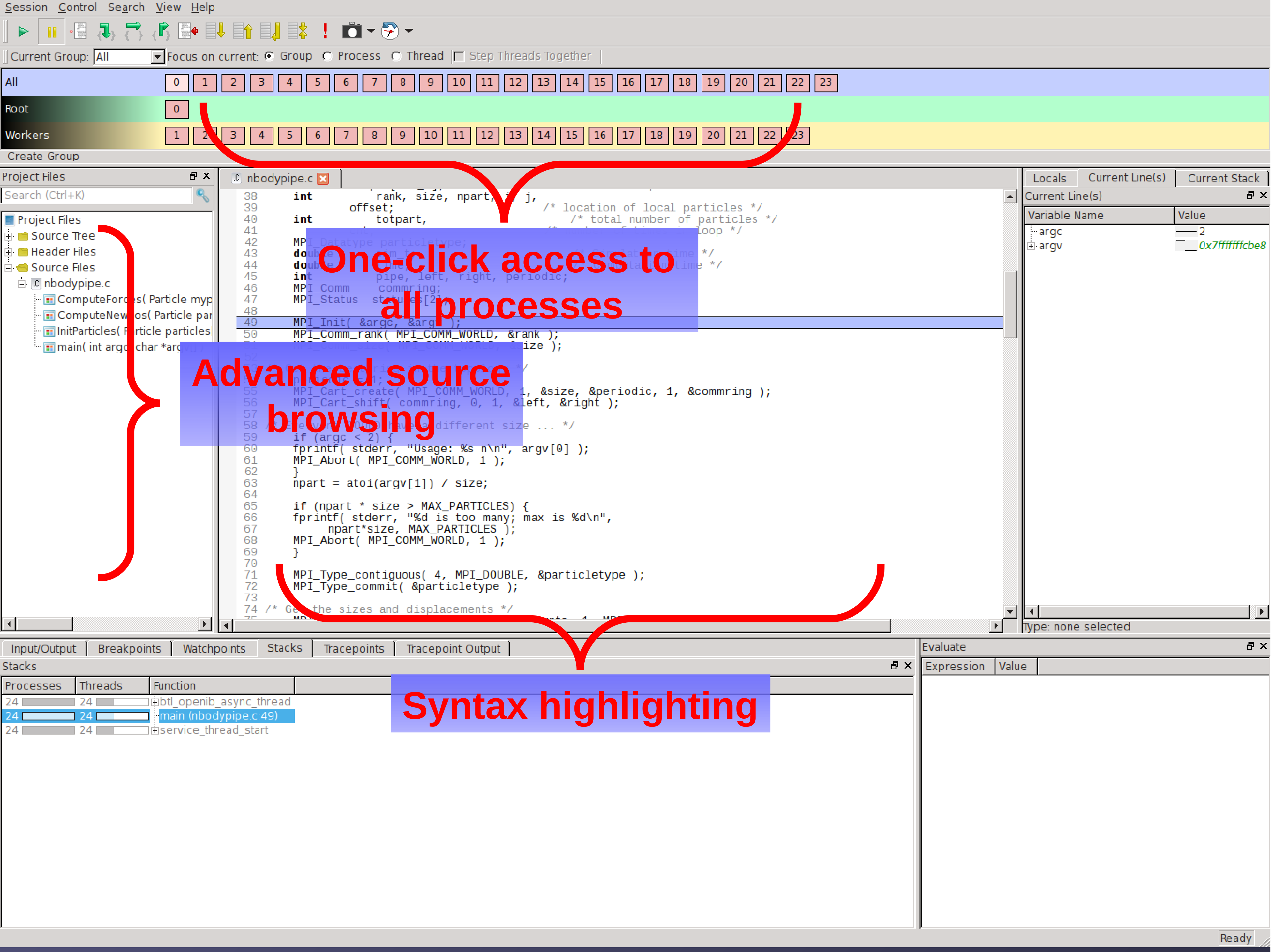
```
$  ssh -Y user@orca.sharcnet.ca
$  ssh -Y orc-dev3
```

- Load the DDT module:

```
$  module load ddt
```

- Then simply type

```
$  ddt  code  [optional code arguments]
```


One-click access to all processes
Advanced source browsing
Syntax highlighting

OpenMP

debugging

demo

Instructions

ssh -Y user@orca.sharcnet.ca

ssh -Y orc-dev{1,2,3,4}

cp -r /home/syam/OpenMP* .

cd OpenMP*

Questions?

- You can always contact me directly (syam@sharcnet.ca) or send an email to help@sharcnet.ca .