Debugging on Graham with DDT

Sergey Mashchenko
(SHARCNET / Compute Ontario / Compute Canada)

July 4, 2018
Outline

- Overview of DDT
- Using DDT on Graham
- Advanced features of DDT
- Questions?
Overview of DDT
DDT (former Allinea; now ARM) is a powerful commercial debugger specifically designed for HPC.

It can debug serial, multi-threaded, MPI, CUDA codes, and any combinations of the above.

It has all the features a debugger needs, including

- Play / pause / step through commands
- Breakpoints / watchpoints / tracepoints
- Display / edit values of variables
- Memory debugging
A lot of the DDT's functionality is for dealing with parallel codes, e.g.

- Easy access to any MPI process or thread (on CPU or GPU)
- Control the execution of processes or threads either in groups or individually
- Visualization of ongoing MPI communications
- Displaying the values of a variable across MPI ranks or threads
For more information, check the DDT wiki page on Compute Canada web portal,

https://docs.computecanada.ca/wiki/ARM_software

and the DDT User guide,

https://developer.arm.com/docs/101136/latest/ddt
Using DDT on Graham
• DDT is a GUI application, so one has to ensure that X11 forwarding is enabled (ssh -Y), and that an X Window server is running on your terminal.
  – On Windows, use a free application MobaXterm (ssh client and X Window server)
  – On Mac, use an XQuartz app for the X Window server functionality

• Graham doesn't have dedicated development nodes (like orca), so one has to reserve node(s) using salloc or sbatch commands.
Basic usage

$ ssh -Y user@graham.computecanada.ca

- **Serial / MPI:**
  $ salloc --x11 --time=0-3:00 --mem-per-cpu=4G --ntasks=4 -A def-user
  $ mpicc -g -O0 code.c -o code
  $ module load ddt-cpu
  $ ddt ./code

- **OpenMP:**
  $ salloc --x11 --time=0-3:00 --mem=16G --cpus-per-task=4 -A def-user
  $ icc -g -O0 -qopenmp code.c -o code
  $ module load ddt-cpu
  $ ddt ./code
• **CUDA:**
  
  $ salloc --x11 --time=0-3:00 --mem-per-cpu=4G --ntasks=1 --gres=gpu:1 -A def-user  
  $ module load cuda  
  $ nvcc -G -g -O0 -arch=sm_60 code.cu -o code  
  $ module load ddt-gpu  
  $ ddt ./code
Advanced features of DDT
Watchpoints

- Unlike breakpoints (which are associated with a specific line in code, with an optional condition), watchpoints are used to pause at any line where the watched variable (or expression) changes its value.
- Changing the default “write” mode to “read” mode will force DDT to pause the next time the variable is accessed in the code.
Tracepoints

- Tracepoints allow you to print certain variables' values at certain lines of the code without pausing the code.
- Can be set from the source code window (right-click), or by right-clicking in the Tracepoints view and selecting Add Tracepoint.
- This option is particularly useful in the offline (non-interactive) mode of using DDT (we'll talk about it later), where it is set via DDT command line option “--trace-at=...”. 
Hybrid codes

- Debugging “vanilla” parallel programs (e.g. MPI) is already a difficult task.
- Debugging hybrid codes (MPI+OpenMP, MPI+CUDA etc) is even harder, and a tool like DDT becomes invaluable.
- DDT provides an easy way to switch “focus” from MPI ranks to CPU threads to GPU threads.
- Breakpoints and watchpoints can be process and thread specific.
- Variable values across all ranks or threads can be displayed.
Large jobs

- `salloc` has a runtime limit of 3 hours. Also, the wait time can become very long if asking for more than one node.
- If a bug is encountered at a predictable point, one can write a checkpointing file right before it happens, and do interactive debugging from that point on.
- How to debug codes which are large or where a bug is encountered at a random point, likely beyond the 3 hour limit of `salloc`?
Attaching to a running job

- One possibility is to use the DDT's advanced feature “Attach to an already running program”.
  - Submit your job via sbatch
  - Launch ddt without arguments from a login node
  - Use squeue command to find out which node(s) are used by the job
  - Choose the “Attach to an already running program” option.
  - Click on Choose Hosts button, and add the job nodes there.
  - In most cases DDT will automatically detect all the processes from your code.
Core files analysis

- If your code's bug results in a crash producing core* files, one can use another advanced DDT functionality, Open Core, to gain insight on the reasons for crashing.

- Compile your code with "-g", submit it via sbatch. Make sure you run it from Project or Scratch file system (on Home file system no core files are created.)

- After the code crashes, launch ddt without arguments, and choose the Open Core option. Add your core files and the path to your code there.

- You can now see the state of the code at the time of crashing.
Offline debugging

- Finally, one could also try the Offline debugging option.
- Submit “ddt --offline ... ./code” to the scheduler via sbatch command.
- There are many ddt switches which can be used in the offline mode. E.g. the following command will do an offline debugging of a 4-ranks MPI job which will save snapshots of the stack/variables every 10 minutes to a log file:
  
  ```
  $ ddt --offline -n 4 --snapshot-interval=10 ./code
  ```

- There is a limited support for breakpoints and tracepoints.
Questions?

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