Parallel Debugging with DDT

James Willis (SciNet)

April 28, 2025

Outline

- Software Bugs
- What is Debugging?
- Symbolic Debuggers
- What is DDT?
- Setting up DDT on Teach
  - Hello-mpi Hands-on
- Client-Server Mode
- Matrix-Matrix Multiply Hands-on Example

Outline

- Other Useful Features of DDT
  - Attach Mode
  - Submitting Jobs to a Scheduler
  - Running Core Files

Software Bugs

Software Bugs

- Writing clean, efficient, error-free code is nearly impossible
- At some point you will run into situations such as:
  - Compile time errors
  - Segmentation faults

```
laptop:~$ gcc app.c -o app
laptop:~$ ./app
Segmentation fault
```

  - Your code doesn't do what you expect
  - Incorrect results
- These are all examples of what is known as a *software bug*

Common Symptoms

- Compile time errors
  - Code syntax errors (easy to fix)
  - Linker errors when linking against libraries
  - Cross-compilation, i.e. compiling for a different computing architecture compared to the host
  - Compilation warnings

  Always turn compiler warnings on and fix or understand them before running your code! It will save you future headaches.

- But just because it compiles does not mean it is correct!

Common Symptoms

- Runtime errors
  - Floating point exceptions
  - Segmentation faults
  - Aborted
  - Incorrect output (e.g. NaNs, Inf)

Error Examples

Type	Reason
Arithmetic	Corner cases e.g. `sqrt(-0.0)`, infinities
Memory Access	Index out of range, uninitialised pointers
Logic	Infinite loop, corner cases
Misuse	Wrong input, ignored error, no initialisation
Syntax	Wrong operators/arguments
Resource Starvation	Memory leak, quota exceeded
Parallel	Race conditions, deadlock

What is going on?

- Almost always, a condition you are sure is satisfied, is not
- But your application likely relies on many such assumptions
- First order of business is finding out what is going wrong and what assumption is not warranted
- Follow the *Fundamental Principle of Confirmation*:
  - Process of confirming, one by one, that many things you **believe** to be true about the code are actually true
- Debugger: program to help detect errors in other programs
- **Debugging: Methodical process of finding and fixing flaws in software**
- **You are the real debugger**

How to Avoid Debugging

- Write better code:
  - Simple, clear, straightforward code
  - Modular (no global variables or 10,000 line functions)
  - Avoid "cute" tricks (no obfuscated C code winners)
- Improve your code/algorithm/language/API understanding
- Don't reinvent the wheel, use existing libraries
- Write (simple) tests for each part of your code
- Use version control (GIT) so you can "roll back" your code if a bug is found

Debugging Workflow

- As soon as you are convinced there is a real problem, create the simplest test case that reproduces the bug
- This is science: model, hypothesis, experiment, conclusion
- Try a smaller problem size, turning off physical effects with options, etc. until you have a simple, fast repeatable example of the bug
- Try to narrow it down to a particular module/function/class. For Fortran, switch on bounds checking (`-fbounds-check`)
- Now you're ready to start debugging

Ways to Debug

- Preemptive:
  - Turn on compiler warnings: fix or understand them!

```
laptop:~$ gcc/gfortran -Wall
```

  - Check your assumptions (e.g. use `assert`)
- Inspect the exit code and read the error messages!
- Use a debugger
- Add print statements
  - **No way to debug!**

What's wrong with using print statements?

Strategy

- Constant cycle:
  1. Strategically add print statements
  2. Compile
  3. Run
  4. Analyse output
- Bug not found? Repeat from 1. again
- Have to remove extra code after the bug is fixed
- Rinse and repeat for each bug

Disadvantages

- Time consuming
- Error prone
- Changes memory, timing...

There's a better way!

Symbolic Debuggers

Symbolic Debuggers

Features

1. Crash inspection
2. Function call stack
3. Step through code
4. Automated interruption
5. Variable checking and setting

Use a graphical debugger or not?

- Local work station: graphical is convenient
- Remotely (Niagara): can be slow, but we will look at ways to improve this later

In any case, graphical and text-based debuggers use the same concepts

Preparing Your Code for the Debugger

- Add debugging flags when compiling your code:

```
laptop:~$ gcc/g++/gfortran -g [-gstabs]
laptop:~$ icc/icpc/ifort -g [-debug parallel]
laptop:~$ nvcc -g -G
```

- Optional flag: switch off optimisation -O0 (sometimes symbol values are hidden to the debugger at higher optimisation levels e.g. -O2, -O3)

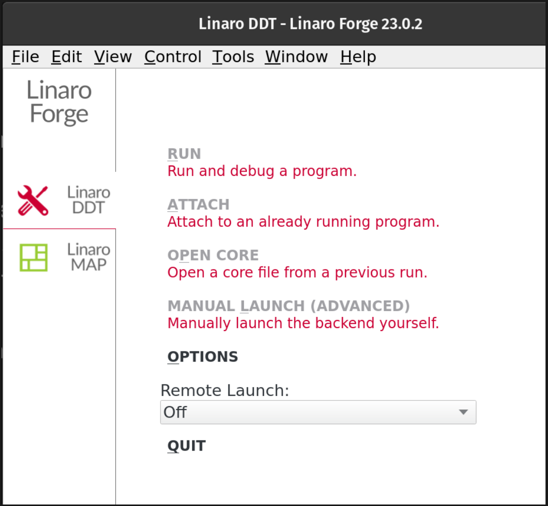
Examples of Symbolic Debuggers

- Command-line based debuggers: GDB, LLDB
- Graphical based debuggers: **DDT**, Visual Studio, Eclipse
  - Nice, more intuitive graphical user interface
  - Front-end to command-line based tools: Same concepts
  - Need graphics support: X11 forwarding (or VNC)
- The rest of the workshop will focus on DDT

What is DDT?

What is DDT?

- **DDT** stands for **Distributed Debugging Tool**
- Powerful GUI-based commercial debugger by *Linaro*
- Developed for debugging parallel, multi-threaded, and distributed applications
- Widely used in high-performance computing environments
- Available on Niagara and other Alliance systems (*Note: license only allows debugging up to 256 processes*)



DDT Features

- **Key Features:**
  - Parallel and distributed debugging capabilities
  - Graphical user interface for intuitive navigation
  - Support for multiple programming languages (e.g. C, C++, Fortran, Python)
  - Supports MPI, OpenMP, threads, CUDA, ROCm and more
  - Integrated performance analysis tools - *MAP*
  - Memory debugging functionalities

Launching DDT on Teach

- Load the latest software stack with a compiler and MPI module:

```
teach-login01:~$ module load StdEnv/2023
```

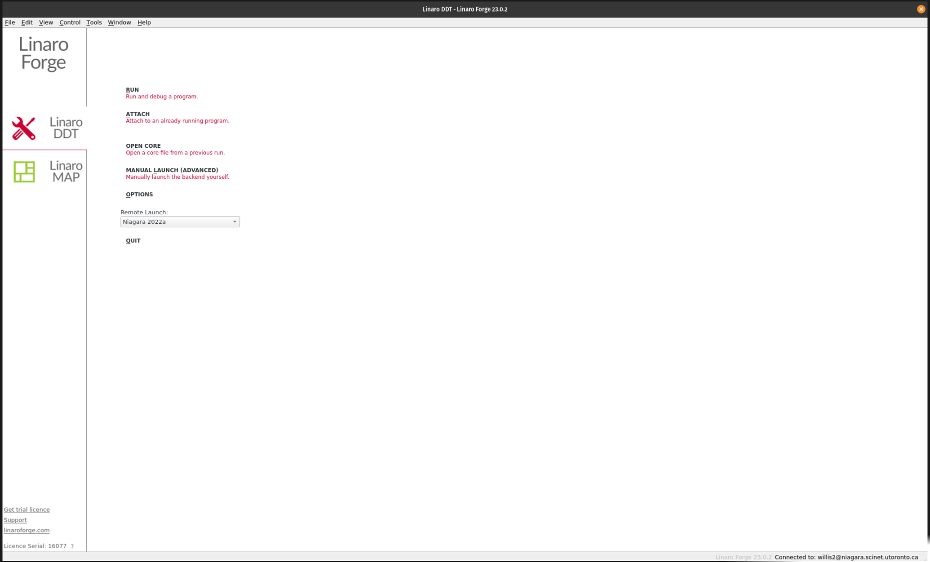
- Load DDT:

```
teach-login01:~$ module load ddt-cpu/23.1.1
```

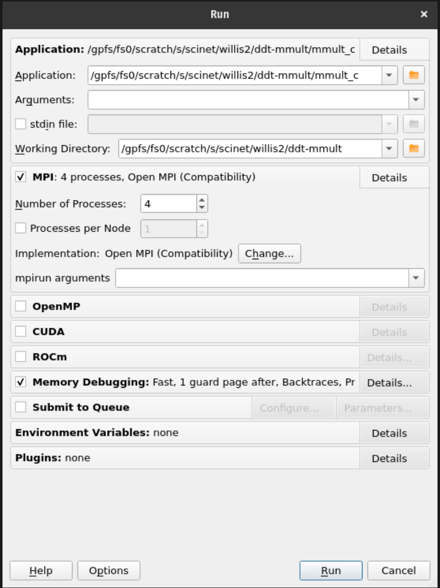
- Start DDT with one of these commands:

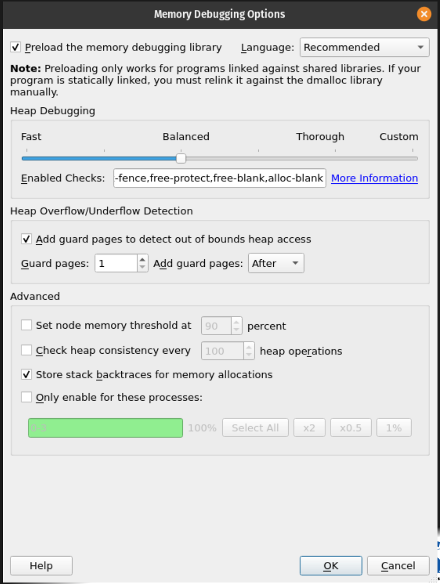
```
teach-login01:~$ ddt
teach-login01:~$ ddt <exe compiled with -g flag>
teach-login01:~$ ddt <exe compiled with -g flag> <arguments>
teach-login01:~$ ddt -n <numprocs> <exe compiled with -g flag> <arguments>
```

Launching DDT on Teach

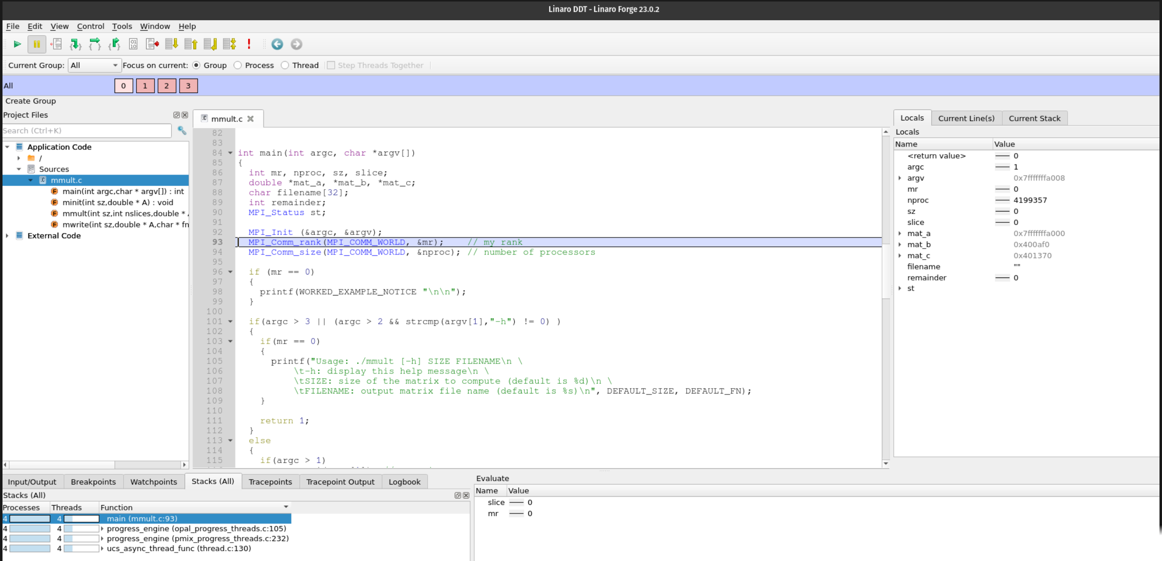


Creating a Debug Session

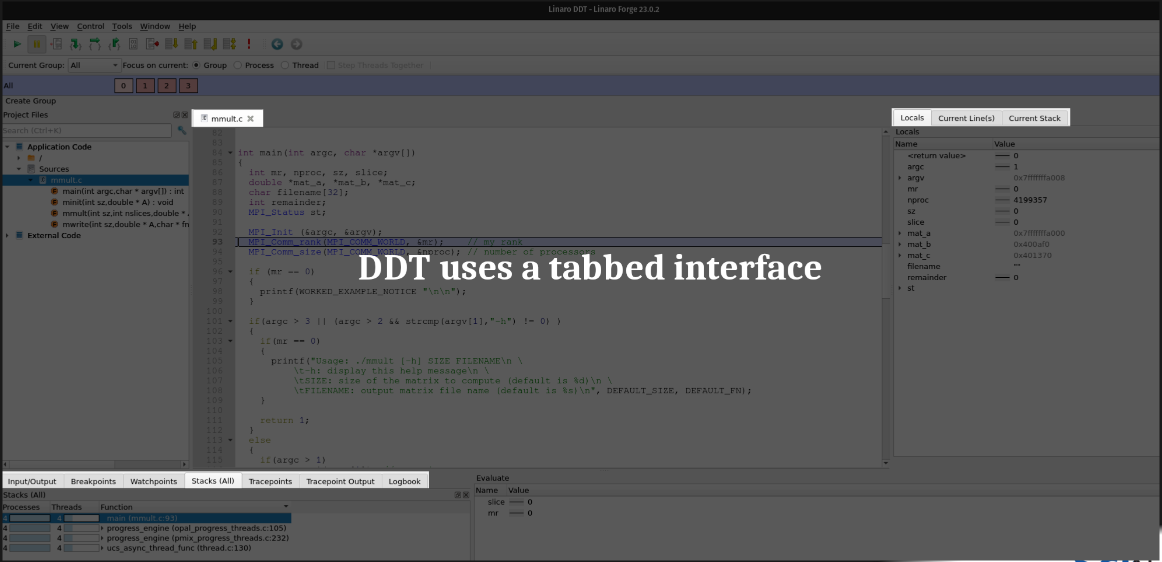




User Interface



User Interface

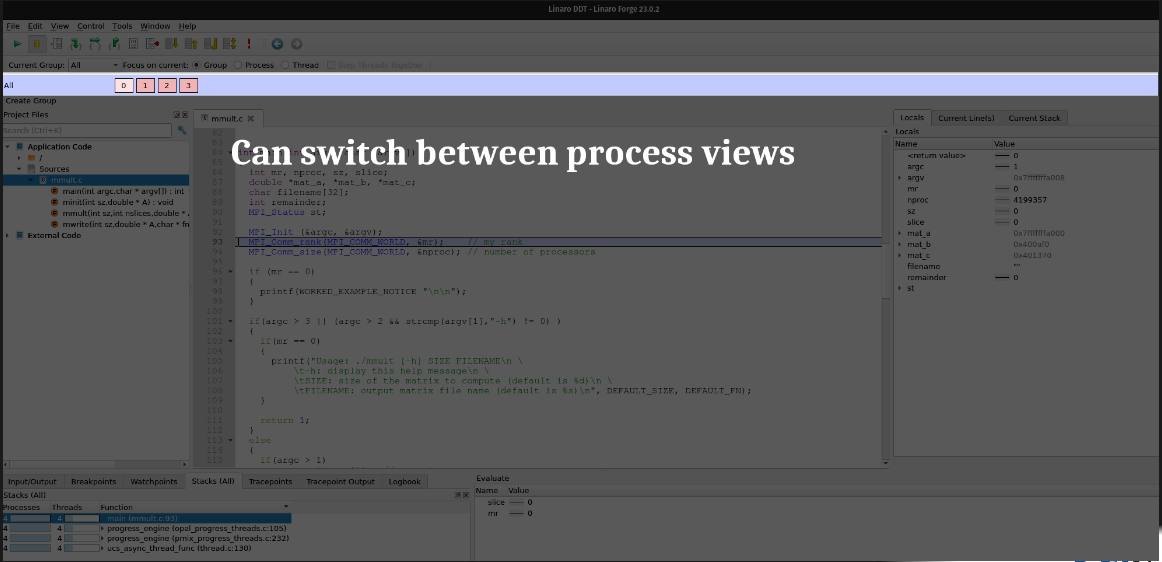


DDT uses a tabbed interface

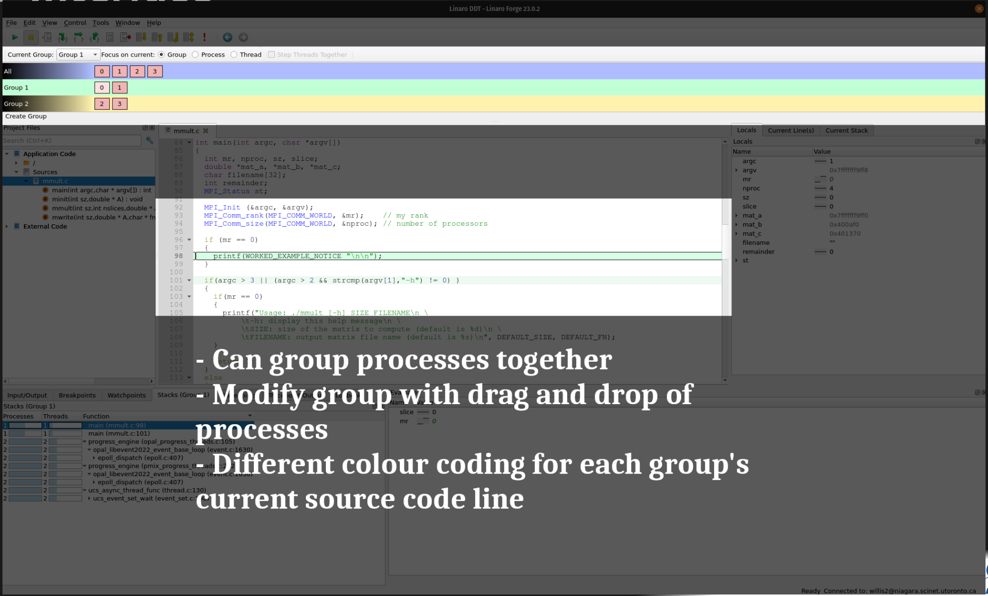
User Interface

DDT automatically finds the source code from the executable

User Interface

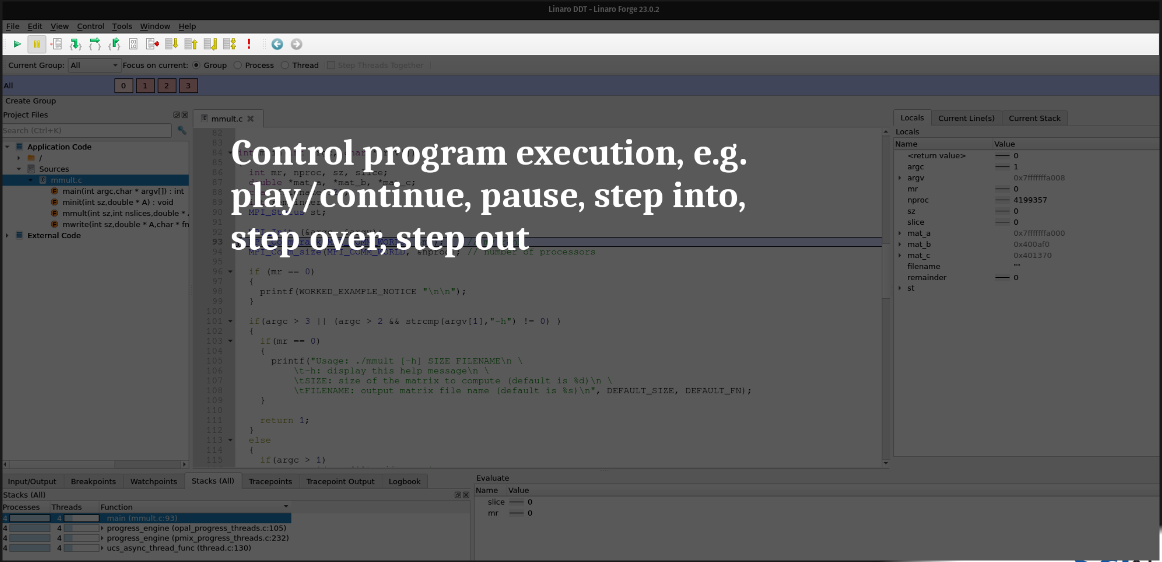


User Interface



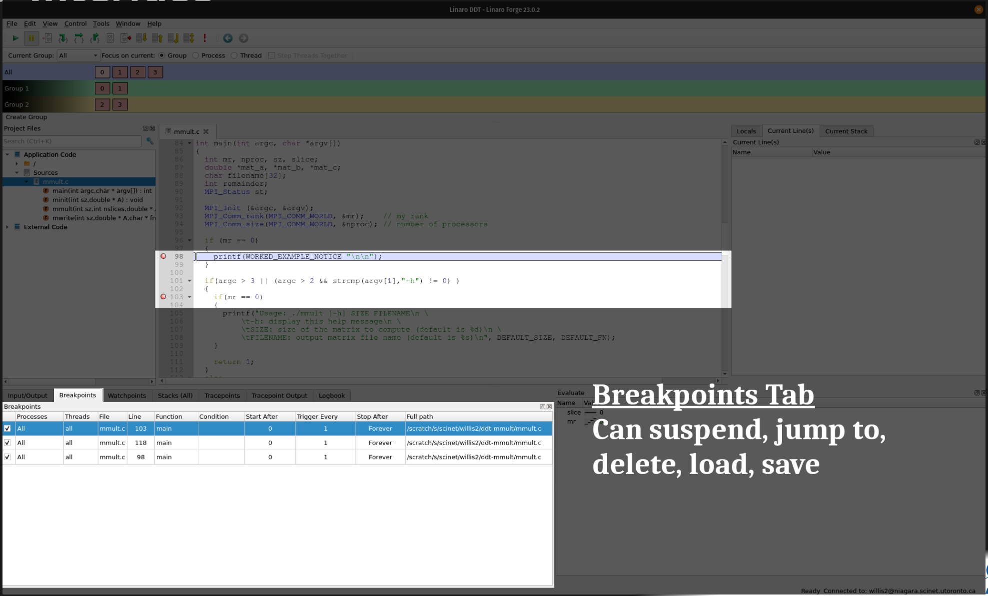
- Can group processes together
- Modify group with drag and drop of processes
- Different colour coding for each group's current source code line

User Interface



Control program execution, e.g. play/continue, pause, step into, step over, step out

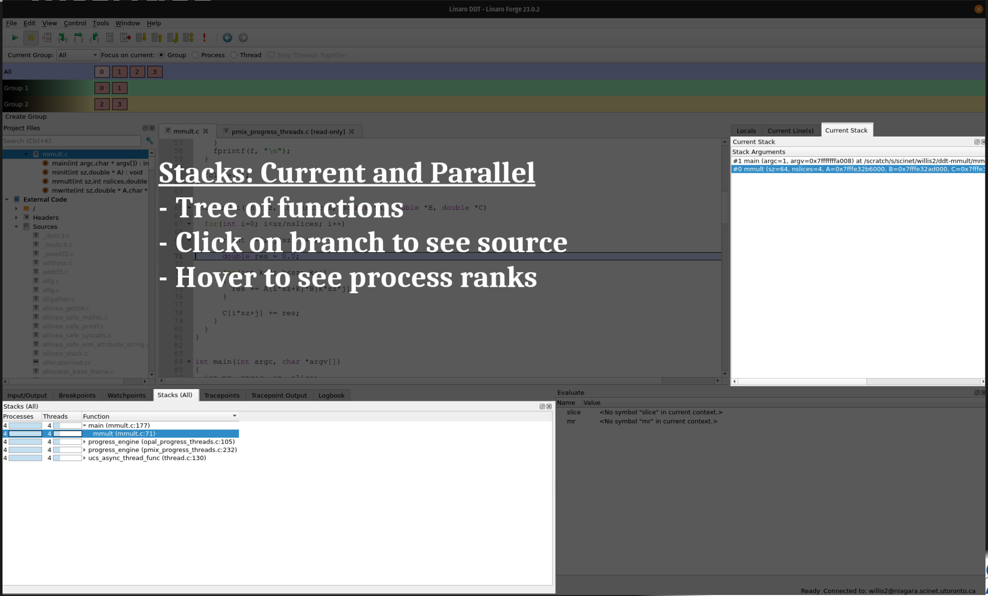
User Interface



Breakpoints Tab

Can suspend, jump to, delete, load, save

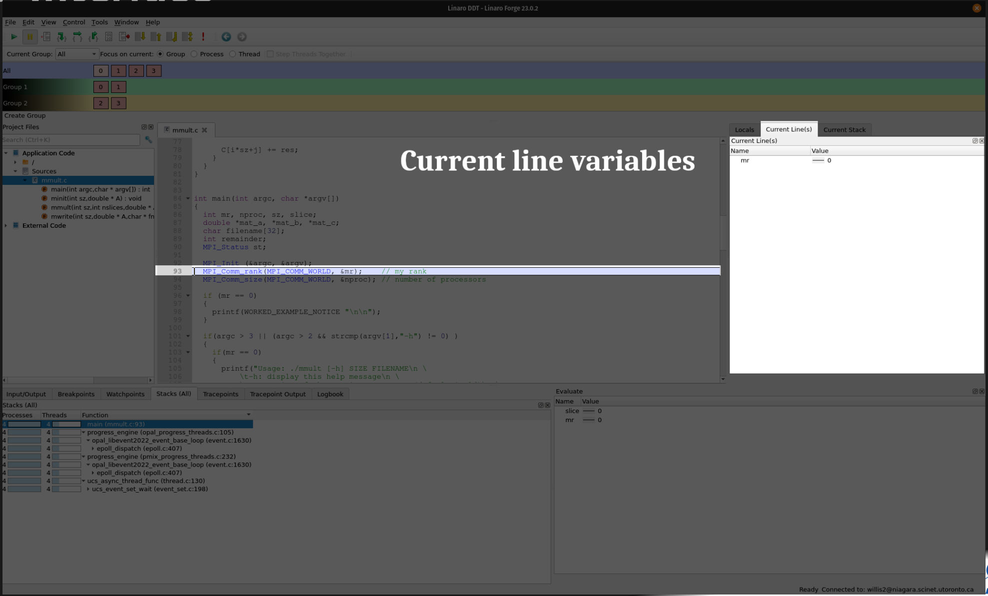
User Interface



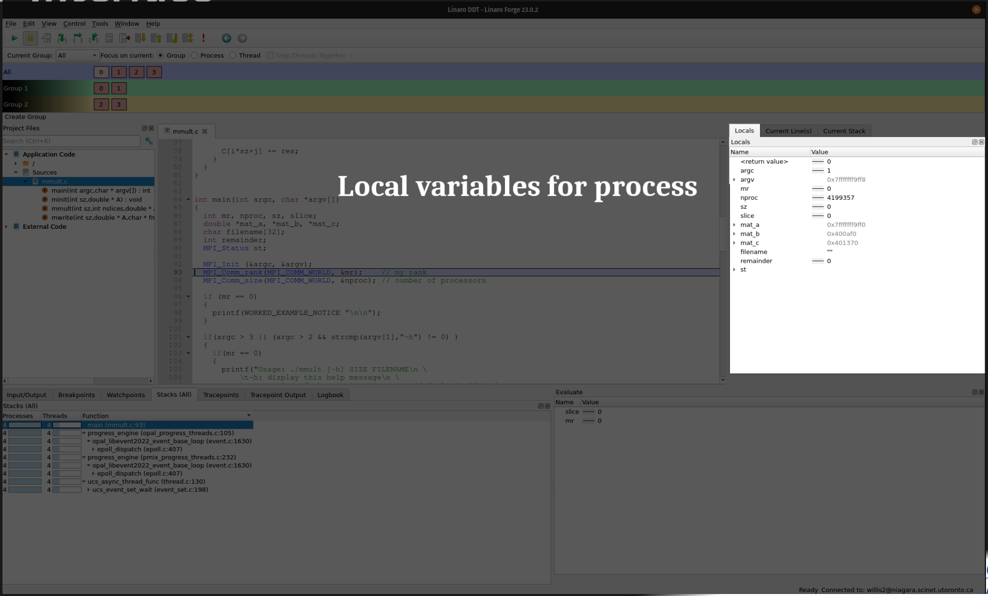
Stacks: Current and Parallel

- Tree of functions
- Click on branch to see source
- Hover to see process ranks

User Interface

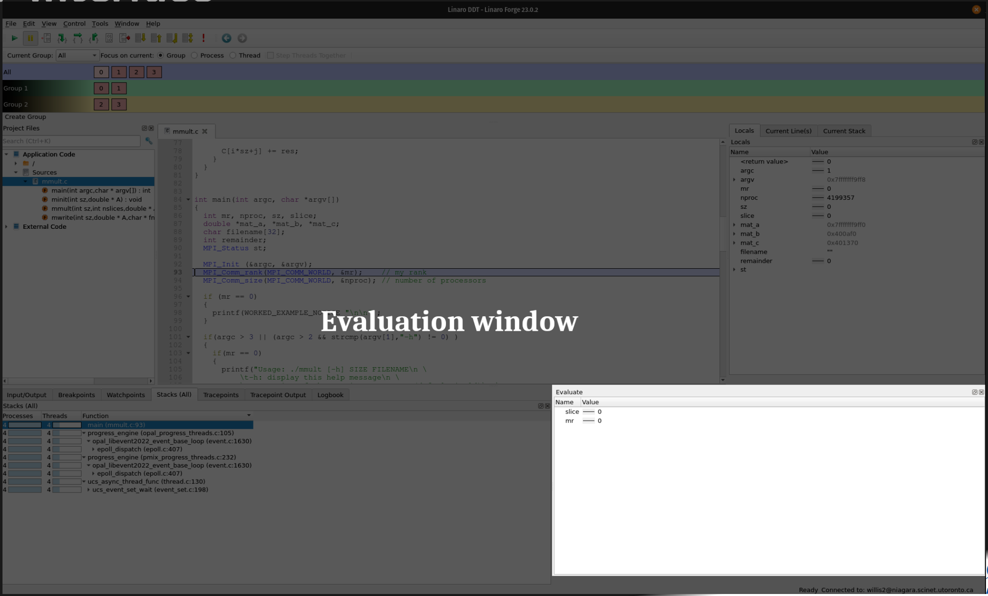


User Interface



Local variables for process

User Interface



Evaluation window

DDT Setup Demonstration

Hands-on `hello-mpi` Example

- Login to Teach:

```
laptop:~$ ssh -X USERNAME@teach.scinet.utoronto.ca
```

- Load compilers, MPI library and `ddt`:

```
teach-login01:~$ module load StdEnv/2023 ddt-cpu/23.1.1
```

- Copy examples from the course directory:

```
teach-login01:~$ cp -r /home/l/lcl_uothpc245/hpc245starter/ddt-examples .
```

- Compile MPI Hello World example, `hello-mpi.c`:

```
teach-login01:~/ddt-examples/ddt-hello-mpi$ mpicc -g hello-mpi.c -o hello-mpi
```

- Run ddt:

```
teach-login01:~/ddt-examples/ddt-hello-mpi$ ddt -n <numprocs> hello-mpi
```

- Experiment with different features of DDT

Client-server Mode

- This mode can be very beneficial if you have a slow internet connection
- Keeps the bulk of the computation on Teach (*server*)
- Only sends minimal amounts of information (network traffic) to your locally running version of DDT (*client*)
- Results in a much smoother experience, avoids slow/laggy interface

Client-server Mode Setup

Setting up the server side

- Connect to Teach and create a startup script which will be run by the server and load the modules that your code needs:

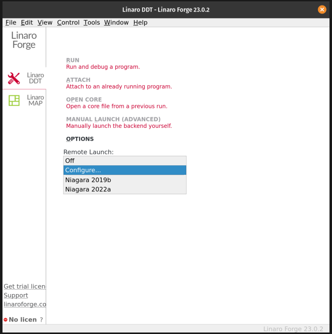
```
#!/bin/bash
module purge
module load StdEnv/2023
module load ddt-cpu/23.1.1
module load python/3.11.5
module load scipy-stack/2025a mpi4py/4.0.3
export OMPI_MCA_pml=ob1
export ARM_TOOLS_CONFIG_DIR=${HOME}/.arm
mkdir -p ${ARM_TOOLS_CONFIG_DIR}
```

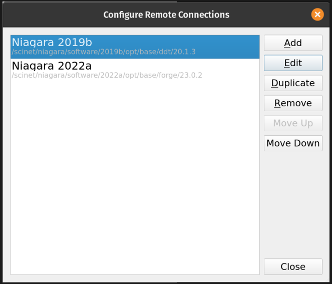
- Name it `ddt_remote_setup.sh` and place it in `$HOME`

Client-server Mode Setup

Setting up the client side

1. Download DDT on your local machine from Linaro and make sure the version matches the one on Teach (23.1.1): https://www.linaroforge.com/downloadForge/
2. Launch `ddt` and select *Configure* from *Remote Connections*

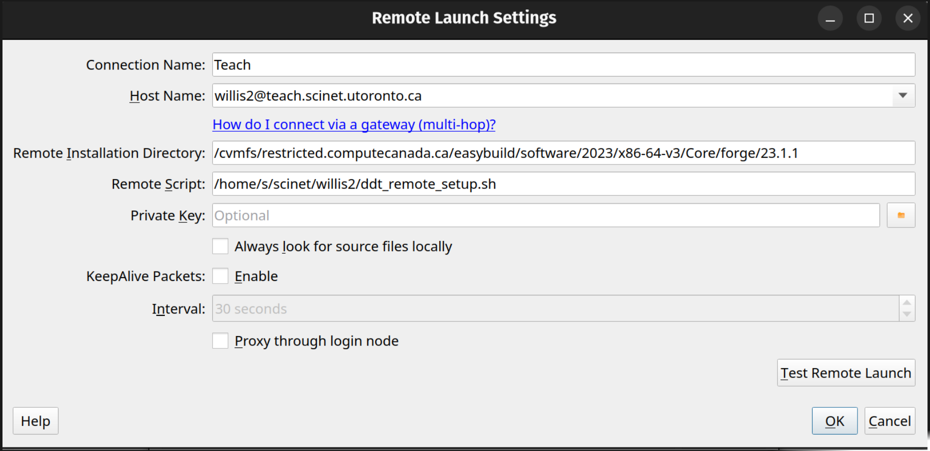




Client-server Mode Setup

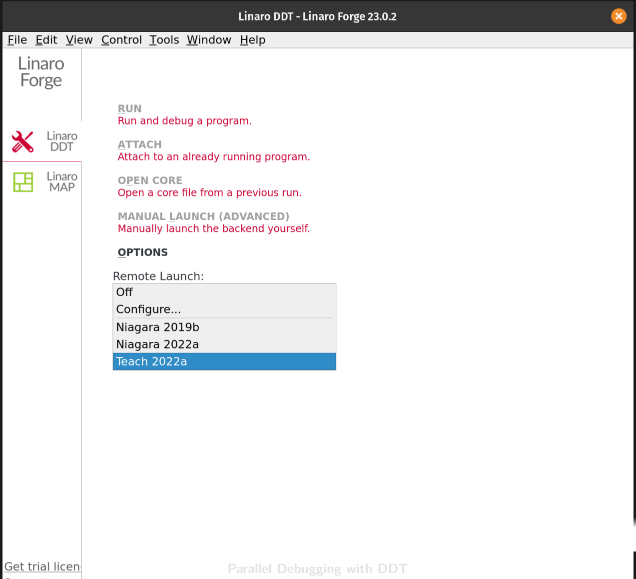
3. Click *Add* and fill out the fields as shown below

Note: *Remote Installation Directory* can be found by running `echo $EBROOTALLINEA` on Teach



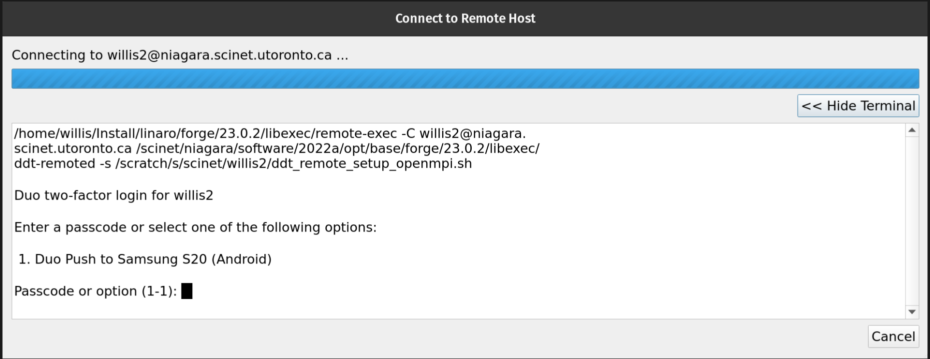
Client-server Mode Setup

4. Click *OK* and now the DDT starting screen should look like this:



Client-server Mode Setup

5. If you have MFA enabled follow the instructions outlined in the text box:



- More detailed instructions can be found here: https://docs.linaroforge.com/23.0.2/html/forge/forge/connecting_to_a_remote_system/connecting_remotely.html
- Please try settting up DDT with client-server mode

Memory Debugging in DDT

- Memory debugging can be turned on in the *Run* window
- Causes the code to stop on an error i.e. memory corruption/leak
- Allows you to check the pointer where the memory corruption has occurred
- Can give an overall view of the memory stats/usage
- Lets look at a real example

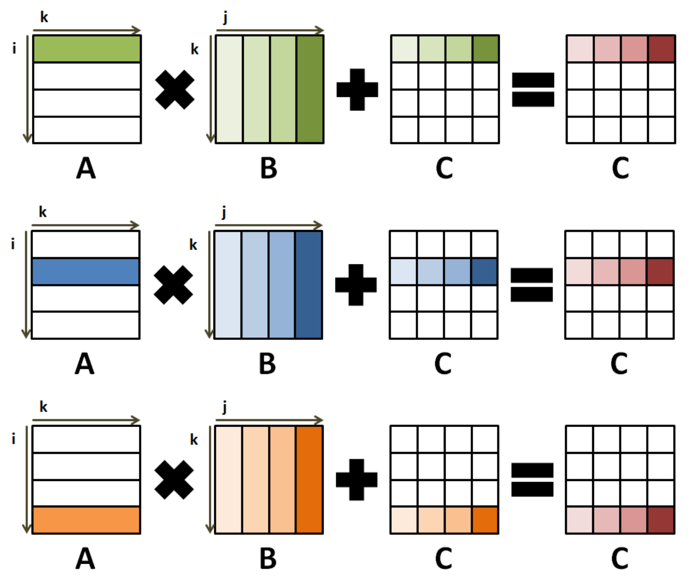
Matrix-Matrix Multiply Example

- Imagine we want to compute the result of this Matrix equation in parallel with MPI:

$$C = A * B + C$$

- The algorithm works as follows:
  1. Rank 0 initialises A, B and C
  2. Rank 0 sends the entire matrix B, with slices of A and C to all other ranks
  3. Each rank performs matrix multiplication on their domain and computes a slice of C
  4. Rank 0 collects the slices of C from each rank and forms the final matrix C
  5. Rank 0 writes C to a file

Matrix-Matrix Multiply Example



Hands-on Matrix-Matrix Multiply

- Change to the `ddt-mmult` directory from the course examples and compile the code:

```
teach-login01:~/ddt-mmult$ make
```

- This will build C and Fortran executables with `-g -O0` named `mmult_c` and `mmult_f`
- The example can then be run with:

```
teach-login01:~/ddt-mmult$ mpirun -np 4 ./mmult_c
```

- For python, load a python, scipy and mpi4py module before compiling the C and Fortran libraries with:

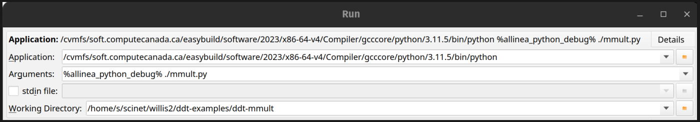
```
teach-login01:~/ddt-mmult$ ml python/3.11.5 scipy-stack/2025a mpi4py/4.0.3
teach-login01:~/ddt-mmult$ make -f mmult_py.makefile
```

- Then run the example:

```
mpirun -np 4 python ./mmult.py
```

Hands-on Matrix-Matrix Multiply

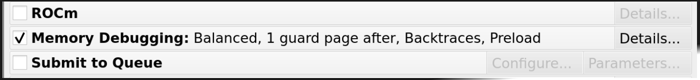
- Try running the code. What output do you get?
- Run the code in DDT to find out what the error is
- Note: if running with python you will need the setup shown below



or from the command line:

```
teach-login01:~/ddt-mmult$ ddt -n 4 python %allinea_python_debug% ./mmult.py
```

- Can you locate the error?
- Can you fix it?
- Hints: Try running with *memory debugging* enabled and make sure **Add guard pages** is enabled



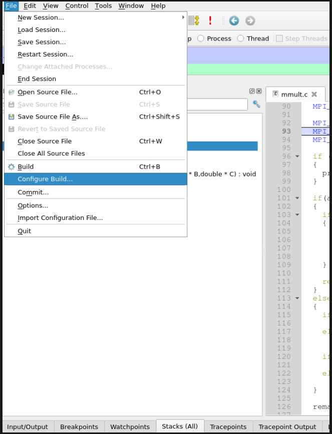
Matrix-Matrix Multiply Demonstration

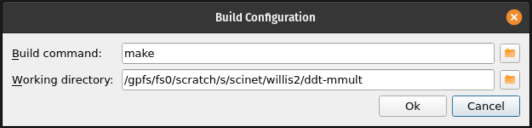
Other Useful Features of DDT

- Editing and recompiling code from within DDT GUI
- Attaching to a running job
- Submit SLURM jobs with DDT
- Running with core files

Editing and Compiling

- DDT also has the ability to edit and recompile source code on-the-fly
- Making it much easier to try potential bug fixes

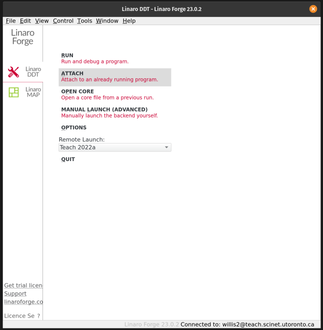


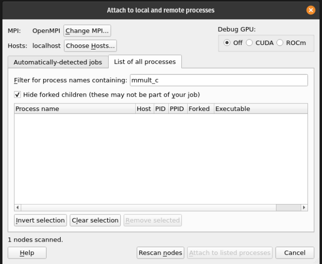


- Build demonstration

Attach to a Running Job

- DDT allows you to attach to an already running job
- For example, say you have submitted a job to the scheduler on Teach and want to monitor it
- You can use the *Attach* button
- More detailed instructions can be found here: https://docs.linaroforge.com/23.0.2/html/forge/ddt/get_started_ddt/attaching_to_running_programs.html#index-9





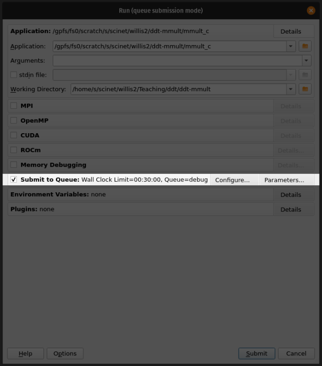
- Attach demonstration

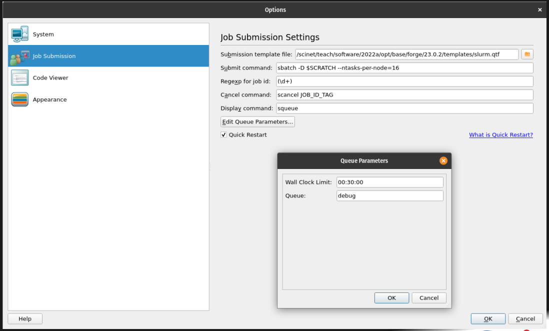
Submit SLURM Jobs

- DDT allows you to submit jobs directly to the SLURM scheduler on Teach
- The job will be monitored in the queue and as soon as it runs DDT will attach to the job

Setup

- Click *Run -> Submit to Queue -> Configure*





Running `.core` Files

- When your code terminates unexpectedly it will generate what is known as a *core dump*
- A core dump is a set of files ending in `.core` per process running
- Each `.core` file contains the process's address space (memory) at the time of the crash
- DDT allows you to run with the core files showing the state of the code at the time of the crash
- Can be useful if your job fails after running for a long time
- If no core dump is generated check that `ulimit -c` is set to `unlimited`, this sets the maximum size a `.core` file can be
- Demonstration

Running DDT inside Open OnDemand

- DDT can also be run inside the new Open OnDemand portal via the Remote Desktop application
- This allows you to run DDT from your browser
- Once you are in the remote desktop, you can launch DDT as you would on Niagara or Teach
- This is a great way to run DDT if you are having issues with X11 forwarding or can't install DDT on your local machine
- More information on Open OnDemand can be found here: https://docs.scinet.utoronto.ca/index.php/Open_OnDemand_Quickstart
- Demonstration

Summary

- DDT is a powerful graphical debugger
- Supports parallel debugging in multiple languages (e.g. C, C++, Fortran, Python)
- Supports MPI, OpenMP, threads, CUDA and more
- DDT documentation: https://docs.linaroforge.com/23.0.2/html/forge/ddt/index.html

Support

Questions? Need help?

Don't be afraid to contact us! We are here to help.

- Email to **support@scinet.utoronto.ca** or to **niagara@computecanada.ca**

References

- Slide 19: Linaro DDT
- Slide 38: Matrix-Matrix Multiply Worked Example
- Slide 44: Client-Server Mode
- Slide 51: Attach Mode
- Slide 52: Submitting to a Queue