

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. <code>sqrt(-0.0)</code> , 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

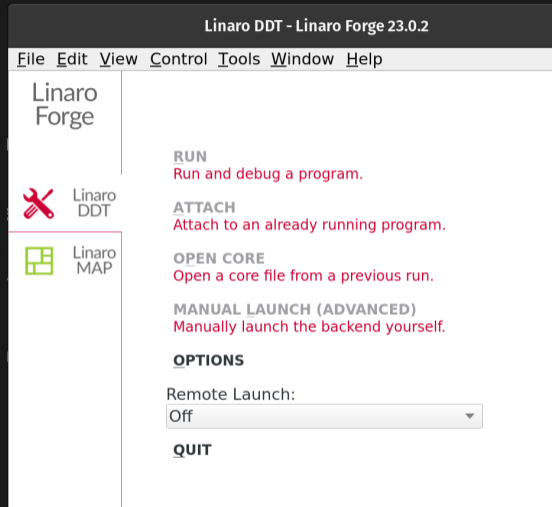
- Command-line based debuggers: GDB, LLDB
- Graphical based debuggers: **DDT**, Visual Studio, Eclipse
 - ▶ Nice, more intuitive graphical user interface
 - ▶ Front-end to command-line based tools: Same concepts
 - ▶ Need graphics support: X11 forwarding (or VNC)
- The rest of the workshop will focus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

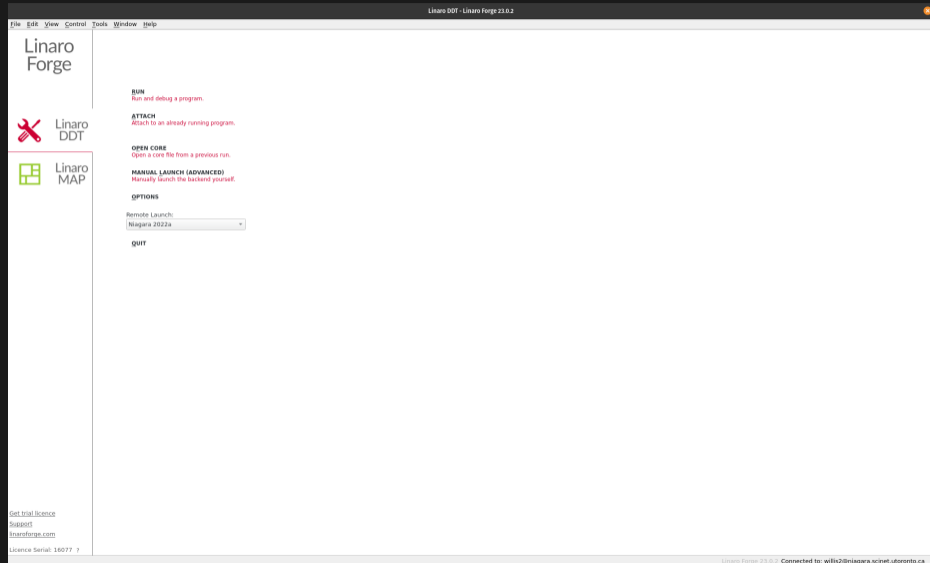
- Load DDT:

```
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

Run [Close]

Application: /gpfs/fs0/scratch/s/scinet/willis2/ddt-mmult/mmult_c [Details]

Application: /gpfs/fs0/scratch/s/scinet/willis2/ddt-mmult/mmult_c [Folder]

Arguments: [Dropdown]

stdjn file: [Dropdown] [Folder]

Working Directory: /gpfs/fs0/scratch/s/scinet/willis2/ddt-mmult [Folder]

MPI: 4 processes, Open MPI (Compatibility) [Details]

Number of Processes: 4 [Dropdown]

Processes per Node: 1 [Dropdown]

Implementation: Open MPI (Compatibility) [Change...]

mpirun arguments: [Dropdown]

OpenMP [Details]

CUDA [Details]

ROCM [Details...]

Memory Debugging: Fast, 1 guard page after, Backtraces, Pr [Details...]

Submit to Queue [Configure...] [Parameters...]

Environment Variables: none [Details]

Plugins: none [Details]

[Help] [Options] [Run] [Cancel]

Memory Debugging Options [Close]

Preload the memory debugging library Language: Recommended [Dropdown]

Note: Preloading only works for programs linked against shared libraries. If your program is statically linked, you must relink it against the dmalloc library manually.

Heap Debugging

Fast Balanced Thorough Custom

Enabled Checks: -fence,free-protect,free-blank,alloc-blank [More Information]

Heap Overflow/Underflow Detection

Add guard pages to detect out of bounds heap access

Guard pages: 1 [Dropdown] Add guard pages: After [Dropdown]

Advanced

Set node memory threshold at 90 [Dropdown] percent

Check heap consistency every 100 [Dropdown] heap operations

Store stack backtraces for memory allocations

Only enable for these processes:

[Green Bar] 100% [Select All] [x2] [x0.5] [1%]

[Help] [OK] [Cancel]

User Interface

Linaro DDT - Linaro Forge 23.0.2

The screenshot displays the Linaro DDT user interface. The main window shows the source code of a C program named `mmult.c`. The code is as follows:

```
82
83
84 int main(int argc, char *argv[])
85 {
86     int mr, nproc, sz, slice;
87     double *mat_a, *mat_b, *mat_c;
88     char filename[32];
89     int remainder;
90     MPI_Status st;
91
92     MPI_Init (&argc, &argv);
93     MPI_Comm_rank(MPI_COMM_WORLD, &mr); // my rank
94     MPI_Comm_size(MPI_COMM_WORLD, &nproc); // number of processors
95
96     if (mr == 0)
97     {
98         printf(WORKED_EXAMPLE_NOTICE "\n\n");
99     }
100
101     if (argc > 3 || (argc > 2 && strcmp(argv[1], "-h") != 0) )
102     {
103         if (mr == 0)
104         {
105             printf("Usage: ./mmult [-h] SIZE FILENAME\n \
106                 \t-h: display this help message\n \
107                 \tSIZE: size of the matrix to compute (default is %d)\n \
108                 \tFILENAME: output matrix file name (default is %s)\n", DEFAULT_SIZE, DEFAULT_FN);
109         }
110
111         return 1;
112     }
113     else
114     {
115         if (argc > 1)
```

The interface includes a menu bar (File, Edit, View, Control, Tools, Window, Help), a toolbar with various icons, and a status bar. On the left, there is a 'Project Files' tree showing the project structure. On the right, there is a 'Locals' panel displaying the current state of variables:

Name	Value
<return value>	0
argc	1
argv	0x7fffffffa008
mr	0
nproc	4199357
sz	0
slice	0
mat_a	0x7fffffffa000
mat_b	0x400af0
mat_c	0x401370
filename	""
remainder	0
st	0

At the bottom, there is an 'Evaluate' panel showing the current values of `slice` (0) and `mr` (0). The 'Stacks (All)' panel at the bottom left shows the current stack frame:

Processes	Threads	Function
4	4	main (mmult.c:93)
4	4	progress_engine (opal_progress_threads.c:105)
4	4	progress_engine (pmix_progress_threads.c:232)
4	4	ucs_async_thread_func (thread.c:130)

User Interface

Linaro DDT - Linaro Forge 23.0.2

File Edit View Control Tools Window Help



Current Group: All Focus on current: Group Process Thread Step Threads Together

All 0 1 2 3

Create Group

Project Files

Search (Ctrl+K)

- Application Code
- Sources
 - mmult.c
 - main(int argc, char * argv[]): int
 - mainf(int sz, double * A): void
 - mmult(int sz, int nslices, double *
 - mwritef(int sz, double * A, char * fr
- External Code

```
82
83
84 int main(int argc, char *argv[])
85 {
86     int mr, nproc, sz, slice;
87     double *mat_a, *mat_b, *mat_c;
88     char filename[32];
89     int remainder;
90     MPI_Status st;
91
92     MPI_Init (&argc, &argv);
93     MPI_Comm_rank(MPI_COMM_WORLD, &mr); // my rank
94     MPI_Comm_size(MPI_COMM_WORLD, &nproc); // number of processors
95
96     if (mr == 0)
97     {
98         printf(WORKED_EXAMPLE_NOTICE "\n\n");
99     }
100
101     if (argc > 3 || (argc > 2 && strcmp(argv[1], "-h") != 0) )
102     {
103         if (mr == 0)
104         {
105             printf("Usage: ./mmult [-h] SIZE FILENAME\n \
106                 \t-h: display this help message\n \
107                 \tSIZE: size of the matrix to compute (default is %d)\n \
108                 \tFILENAME: output matrix file name (default is %s)\n", DEFAULT_SIZE, DEFAULT_FM);
109         }
110
111         return 1;
112     }
113     else
114     {
115         if (argc > 1)
```

DDT uses a tabbed interface

Locals Current Line(s) Current Stack

Name	Value
<return value>	0
argc	1
argv	0x7ffffffa008
mr	0
nproc	4199357
sz	0
slice	0
mat_a	0x7ffffffa000
mat_b	0x400af0
mat_c	0x401370
filename	**
remainder	0
st	

Input/Output Breakpoints Watchpoints Stacks (All) Tracepoints Tracepoint Output Logbook

Stacks (All)

Processes	Threads	Function
4	4	main (mmult.c:93)
4	4	progress_engine (opal_progress_threads.c:105)
4	4	progress_engine (pmix_progress_threads.c:232)
4	4	ucs_async_thread_func (thread.c:130)

Evaluate

Name	Value
slice	0
mr	0

User Interface

Linaro DDT - Linaro Forge 23.0.2

DDT automatically finds the source code from the executable

The screenshot displays the DDT user interface with the following components:

- Source Code Editor:** Shows the source code for `mmult.c`. The current line is 93, which is `MPI_Comm_rank(MPI_COMM_WORLD, &mr); // my rank`. The code includes MPI initialization, rank determination, and usage instructions.
- Project Files:** A tree view on the left shows the project structure, including `Application Code` and `Sources`.
- Locals Panel:** A panel on the right displays the current stack frame's local variables. The variables and their values are:
 - `<return value>`: 0
 - `argc`: 1
 - `argv`: 0x7fffffffa008
 - `mr`: 0
 - `nproc`: 4199357
 - `sz`: 0
 - `slice`: 0
 - `mat_a`: 0x7fffffffa000
 - `mat_b`: 0x400af0
 - `mat_c`: 0x401370
 - `filename`: ""
 - `remainder`: 0
 - `st`: 0
- Stacks Panel:** A panel at the bottom left shows the current stack frame for `main (mmult.c:93)` and its child threads: `progress_engine (opal_progress_threads.c:105)`, `progress_engine (pmix_progress_threads.c:232)`, and `ucs_async_thread_func (thread.c:130)`.
- Evaluate Panel:** A panel at the bottom right shows the current values of `slice` (0) and `mr` (0).

User Interface

Linaro DDT - Linaro Forge 23.0.2

File Edit View Control Tools Window Help



Current Group: All Focus on current: Group Process Thread Step Threads Together

All 0 1 2 3

Create Group

Project Files

Search (Ctrl+K)

- Application Code
 - /
 - Sources
 - mmult.c
 - main(int argc, char * argv[]): int
 - mainf(int sz, double * A): void
 - mmult(int sz, int nslices, double *
 - mwritef(int sz, double * A, char * fr
- External Code

```
82
83
84
85
86
87 int mr, nproc, sz, slice;
88 double *mat_a, *mat_b, *mat_c;
89 char filename[32];
90 int remainder;
91 MPI_Status st;
92
93 MPI_Init (&argc, &argv);
94 MPI_Comm_rank(MPI_COMM_WORLD, &mr); // my rank
95 MPI_Comm_size(MPI_COMM_WORLD, &nproc); // number of processors
96
97 if (mr == 0)
98 {
99     printf(WORKED_EXAMPLE_NOTICE "\n\n");
100 }
101
102 if (argc > 3 || (argc > 2 && strcmp(argv[1], "-h") != 0) )
103 {
104     if (mr == 0)
105     {
106         printf("Usage: ./mmult [-h] SIZE FILENAME\n \
107             \t-h: display this help message\n \
108             \tSIZE: size of the matrix to compute (default is %d)\n \
109             \tFILENAME: output matrix file name (default is %s)\n", DEFAULT_SIZE, DEFAULT_FM);
110     }
111     return 1;
112 }
113 else
114 {
115     if (argc > 1)
```

Can switch between process views

Locals Current Line(s) Current Stack

Locals

Name	Value
<return value>	0
argc	1
argv	0x7fffffffa008
mr	0
nproc	4199357
sz	0
slice	0
mat_a	0x7fffffffa000
mat_b	0x400af0
mat_c	0x401370
filename	""
remainder	0
st	

Input/Output Breakpoints Watchpoints Stacks (All) Tracepoints Tracepoint Output Logbook Evaluate

Stacks (All)

Processes	Threads	Function
4	4	main (mmult.c:93)
4	4	progress_engine (opal_progress_threads.c:105)
4	4	progress_engine (pmix_progress_threads.c:232)
4	4	ucs_async_thread_func (thread.c:130)

Evaluate

Name	Value
slice	0
mr	0

User Interface

Linaro DDT - Linaro Forge 23.0.2

File Edit View Control Tools Window Help

Current Group: Group 1 Focus on current: Group Process Thread Step Threads Together

All 0 1 2 3

Group 1 0 1

Group 2 2 3

Create Group

Project Files

Application Code

Sources

mmult.c

main(int argc, char * argv[]): int

main(int sz, double * A[]): void

mmult(int sz, int ndices, double *

mark(int sz, double * A, char * f)

External Code

```
int main(int argc, char *argv[])
{
    int nr, nproc, sz, slice;
    double *mat_a, *mat_b, *mat_c;
    char filename[32];
    int remainder;
    MPI_Status st;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &nr); // my rank
    MPI_Comm_size(MPI_COMM_WORLD, &nproc); // number of processors

    if (nr == 0)
    {
        printf(WORKED_EXAMPLE_NOTICE "\n\n");
    }

    if (argc > 3 || (argc > 2 && strcmp(argv[1], "-h") != 0))
    {
        if (nr == 0)
        {
            printf("Usage: ./mmult [-h] SIZE FILENAME\n\n");
            printf("  -h: display this help message\n");
            printf("  -SIZE: size of the matrix to compute (default is %d)\n");
            printf("  -FILENAME: output matrix file name (default is %s)\n", DEFAULT_SIZE, DEFAULT_FN);
        }
    }
}
```

Locals

Name	Value
argc	1
argv	0x7f77777777
nr	0
nproc	4
sz	0
slice	0
mat_a	0x7f77777777
mat_b	0x400a00
mat_c	0x401370
filename	""
remainder	0
st	0

Input/Output Breakpoints Watchpoints Stacks (Group 1)

Processes	Threads	Function
1	1	main (mmult.c:98)
1	1	main (mmult.c:101)
2	2	progress_engine (opal_progress.D
2	2	opal_fibervent2022_event_base_loo
2	2	epoll_dispatch (epoll.c:407)
2	2	progress_engine (pmix_progress.H
2	2	opal_fibervent2022_event_base_loo
2	2	epoll_dispatch (epoll.c:407)
2	2	ucs_async_thread_func (libthread.c:130)
2	2	ucs_event_set_wait (event_set.c:

slice 0

nr 0

- 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

```
82  
83  
84  
85  
86 int mr, nproc, sz, slice;  
87 double *mat_a, *mat_b, *mat_c;  
88  
89  
90 MPI_Status st;  
91  
92  
93  
94  
95  
96 if (mr == 0)  
97 {  
98     printf(WORKED_EXAMPLE_NOTICE "\n\n");  
99 }  
100  
101  
102 if (argc > 3 || (argc > 2 && strcmp(argv[1], "-h") != 0))  
103 {  
104     if (mr == 0)  
105     {  
106         printf("Usage: ./mmult [-h] SIZE FILENAME\n \  
107             \t-h: display this help message\n \  
108             \tSIZE: size of the matrix to compute (default is %d)\n \  
109             \tFILENAME: output matrix file name (default is %s)\n", DEFAULT_SIZE, DEFAULT_FM);  
110     }  
111     return 1;  
112 }  
113 else  
114 {  
115     if (argc > 1)
```

Name	Value
<return value>	0
argc	1
argv	0x7fffffffa008
mr	0
nproc	4199357
sz	0
slice	0
mat_a	0x7fffffffa000
mat_b	0x400af0
mat_c	0x401370
filename	**
remainder	0
st	

Processes	Threads	Function
4	4	main (mmult.c:95)
4	4	progress_engine (opal_progress_threads.c:105)
4	4	progress_engine (pmix_progress_threads.c:232)
4	4	ucs_async_thread_func (thread.c:130)

User Interface

The screenshot displays the Linaro DDT (Data Display Tool) user interface. At the top, the title bar reads "Linaro DDT - Linaro Forge 23.0.2". The interface includes a menu bar (File, Edit, View, Control, Tools, Window, Help), a toolbar, and a "Current Group" dropdown set to "All". Below this, there are visual representations of process and thread groups. The central pane shows the source code for "mmult.c", with a white tooltip box highlighting a line of code: `printf(WORKED_EXAMPLE_NOTICE "\n\n");`. The right-hand side of the interface contains tabs for "Locals", "Current Line(s)", and "Current Stack", with the "Current Line(s)" tab active, showing a table with "Name" and "Value" columns. At the bottom, the "Breakpoints" tab is selected, displaying a table of breakpoints. A text overlay on the right side of the image reads: "Breakpoints Tab Can suspend, jump to, delete, load, save".

Processes	Threads	File	Line	Function	Condition	Start After	Trigger Every	Stop After	Full path	
<input checked="" type="checkbox"/>	All	all	mmult.c	103	main		0	1	Forever	/scratch/iscinet/willis2@dt-mmult/mmult.c
<input checked="" type="checkbox"/>	All	all	mmult.c	118	main		0	1	Forever	/scratch/iscinet/willis2@dt-mmult/mmult.c
<input checked="" type="checkbox"/>	All	all	mmult.c	98	main		0	1	Forever	/scratch/iscinet/willis2@dt-mmult/mmult.c

User Interface

Stacks: Current and Parallel

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

Stacks (All)

Processes	Threads	Function
4	4	main (mmult.c:177)
4	4	mmult (mmult.c:232)
4	4	progress_engine (opal_progress_threads.c:105)
4	4	progress_engine (pmix_progress_threads.c:232)
4	4	ucs_async_thread_func (thread.c:130)

Current Stack

```
#1 main (argc=1, argv=0x7fffffa008) at /scratch/ls/scinet/willis2/ddt-mmult/mm  
#0 mmult (t2=64, nlices=4, A=0x7ffff32b6000, B=0x7ffff32a5000, C=0x7ffff
```

User Interface

The screenshot displays the DDT (Data Display Tool) user interface. The main window is titled "Linux DDT - Linux Forge 23.0.2". The interface is divided into several panes:

- Top Panel:** Contains menu options (File, Edit, View, Control, Tools, Window, Help) and a toolbar with various icons for running, stepping, and debugging.
- Current Group:** Shows "All" as the current group, with sub-groups "Group 1" and "Group 2" visible below it.
- Project Files:** A tree view on the left showing the project structure, including "Application Code" and "Sources".
- Code Editor:** The central pane shows the source code for "mmult.c". The current line is highlighted, and the text "Current line variables" is overlaid on the editor.
- Current Line(s) Window:** A window on the right showing the current line variables. It contains a table with the following data:

Name	Value
mr	0
- Stacks (All) Window:** A window at the bottom left showing the stack of processes and threads. The top entry is "main (mmult.c:31)".
- Evaluate Window:** A window at the bottom right showing the current line variables, identical to the "Current Line(s)" window.

User Interface

The screenshot displays the Linaro DDT - Linaro Forge 22.0.2 user interface. The main window shows the source code for `mmult.c`. The `locals` window is open, showing the current state of local variables. The `stacks` window is also open, showing the current stack frame.

Code Editor:

```
C[1*sz+] += zwa;
}
}
int main(int argc, char *argv[])
{
    int nr, nproc, sz, slice;
    double *mat_a, *mat_b, *mat_c;
    char filename[32];
    int remainder;
    MPI_Status st;

    MPI_Init (&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &nr); // my rank
    MPI_Comm_size(MPI_COMM_WORLD, &nproc); // number of processors

    if (nr == 0)
    {
        printf(WORKED_EXAMPLE_NOTICE "\n\n");
    }

    if (argc > 3 || (argc > 2 && strcmp(argv[1], "-h") != 0))
    {
        if (nr == 0)
        {
            printf("Usage: ./mmult [-h] SIZE FILENAME\n \
                \t-h: display this help message\n \
                \n");
        }
    }
}
```

Locals Window:

Name	Value
<return value>	0
argc	1
argv	0x7ffffff0
nr	0
nproc	4199357
sz	0
slice	0
mat_a	0x7ffffff0
mat_b	0x400a0
mat_c	0x401370
filename	""
remainder	0
st	0

Stacks (All) Window:

Processes	Threads	Function
4	4	main (mmult.c:1)
4	4	progress_engine (opal_progress_threads.c:105)
4	4	opal_sbevent2022_event_base_loop (event.c:1630)
4	4	epoll_dispatch (epoll.c:407)
4	4	progress_engine (pmix_progress_threads.c:232)
4	4	opal_sbevent2022_event_base_loop (event.c:1630)
4	4	epoll_dispatch (epoll.c:407)
4	4	ucs_async_thread_func (thread.c:130)
4	4	ucs_event_set_wait (event_set.c:198)

Local variables for process

User Interface

The screenshot displays the Linaro DDT - Linaro Forge 22.0.2 user interface. The main window is divided into several panes:

- Code Editor:** Shows the source code for `mmult.c`. The current line is `MPI_Comm_rank(MPI_COMM_WORLD, &mr); // my rank`. A large white text overlay "Evaluation window" is positioned over the code editor.
- Project Files:** A tree view on the left showing the project structure, including `Application Code`, `Sources`, and `External Code`.
- Stacks (AE):** A window at the bottom left showing the current stack of threads. The top thread is `main [mmult.c:1]`, followed by `progress_engine (opal_progress_threads.c:105)`, `opal_sbevent2022_event_base_loop (event.c:1630)`, `epoll_dispatch (epoll.c:407)`, `progress_engine (pmix_progress_threads.c:232)`, `opal_sbevent2022_event_base_loop (event.c:1630)`, `epoll_dispatch (epoll.c:407)`, `ucs_async_thread_func (tthread.c:130)`, and `ucs_event_set_wait (event_set.c:198)`.
- Locals:** A window on the right showing the current local variables. The variables are: `<return value>` (0), `argc` (1), `argv` (0x77777777), `mr` (0), `rproc` (4199357), `sz` (0), `slice` (0), `mat_a` (0x77777777), `mat_b` (0x400a0b), `mat_c` (0x4013f0), `filename` (""), and `remainder` (0).
- Evaluate:** A small window at the bottom right showing the current evaluation results. The variables are: `slice` (0) and `mr` (0).

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:

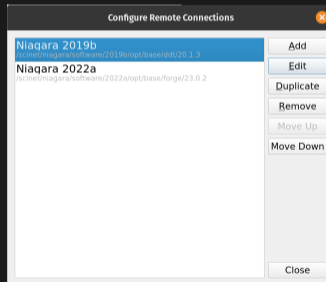
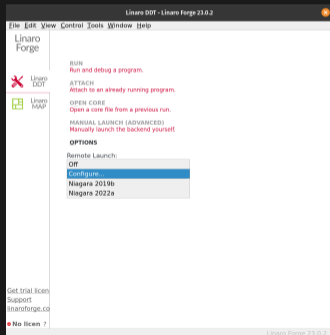
```
#!/bin/bash
module purge
module load StdEnv/2023
module load ddt-cpu/23.1.1
module load python/3.11.5
module load scipy-stack/2025a mpi4py/4.0.3
export OMPI_MCA_pml=ob1
export ARM_TOOLS_CONFIG_DIR=${HOME}/.arm
mkdir -p ${ARM_TOOLS_CONFIG_DIR}
```

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

Client-server Mode Setup

Setting up the client side

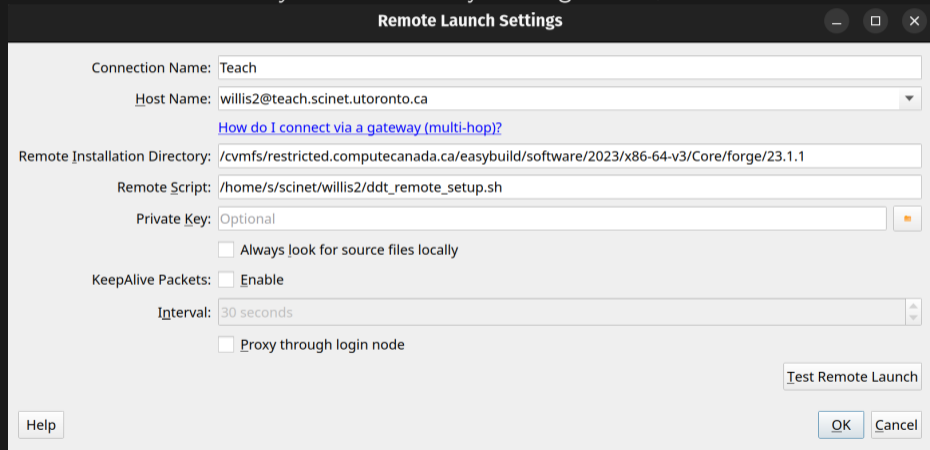
- 1 Download DDT on your local machine from Linaro and make sure the version matches the one on Teach (23.1.1): <https://www.linaroforge.com/downloadForge/>
- 2 Launch `ddt` and select *Configure* from *Remote Connections*



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

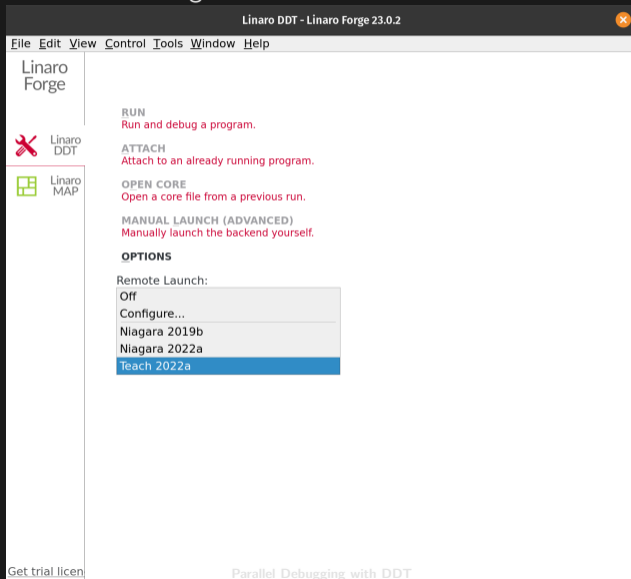


The screenshot shows a dialog box titled "Remote Launch Settings" with the following fields and options:

- Connection Name: Teach
- Host Name: willis2@teach.scinet.utoronto.ca
- Remote Installation Directory: /cvmfs/restricted.computeCanada.ca/easybuild/software/2023/x86-64-v3/Core/forge/23.1.1
- Remote Script: /home/s/scinet/willis2/ddt_remote_setup.sh
- Private Key: Optional
- Always look for source files locally
- KeepAlive Packets: Enable
- Interval: 30 seconds
- Proxy through login node
- Buttons: Help, Test Remote Launch, OK, Cancel

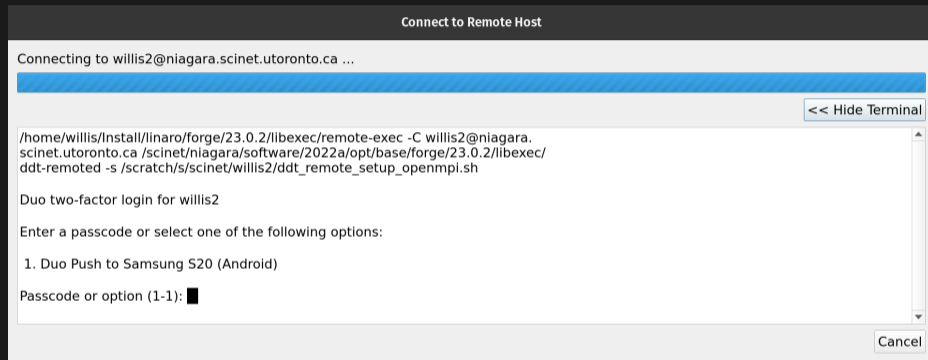
Client-server Mode Setup

- 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:



```
Connect to Remote Host

Connecting to willis2@niagara.scinet.utoronto.ca ...

/home/willis/Install/linaro/forge/23.0.2/libexec/remote-exec -C willis2@niagara.
scinet.utoronto.ca /scinet/niagara/software/2022a/opt/base/forge/23.0.2/libexec/
ddt-remoted -s /scratch/s/scinet/willis2/ddt_remote_setup_openmpi.sh

Duo two-factor login for willis2

Enter a passcode or select one of the following options:

1. Duo Push to Samsung S20 (Android)

Passcode or option (1-1): █

Cancel
```

- 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 setting 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

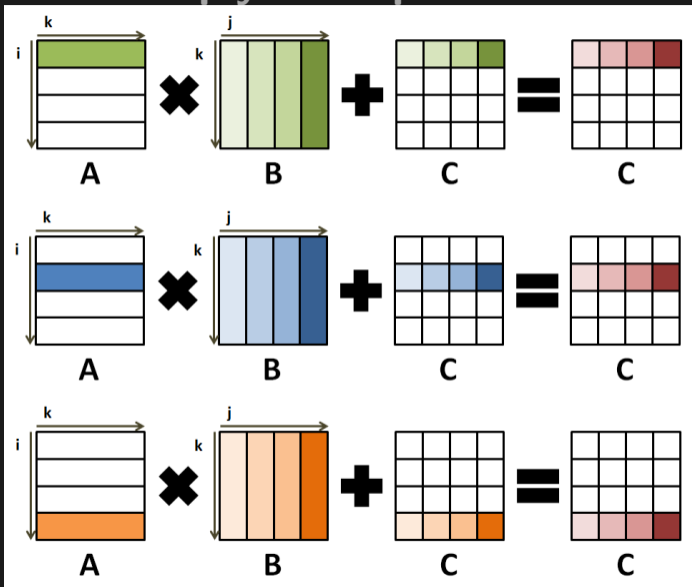
Matrix-Matrix Multiply Example

- 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:

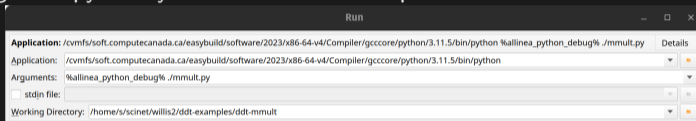
```
teach-login01:~/ddt-mmult$ ml python/3.11.5 scipy-stack/2025a mpi4py/4.0.3  
teach-login01:~/ddt-mmult$ make -f mmult_py.makefile
```

- Then run the example:

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

Hands-on Matrix-Matrix Multiply

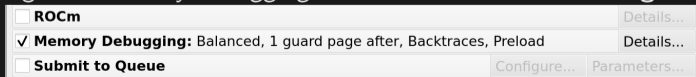
- Try running the code. What output do you get?
- Run the code in DDT to find out what the error is
- Note: if running with python you will need the setup shown below



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



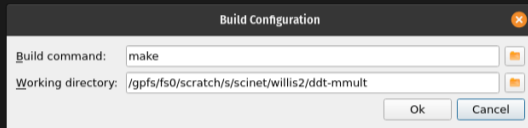
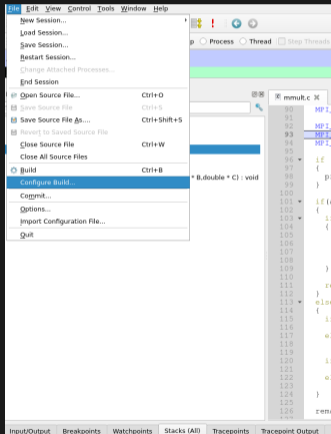
Matrix-Matrix Multiply Demonstration

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

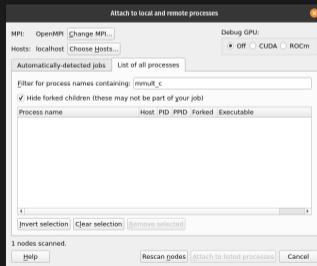
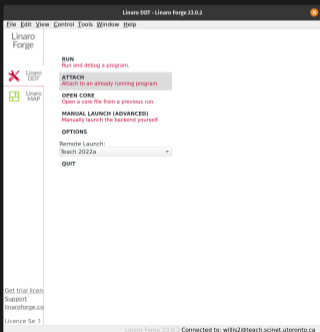
- DDT also has the ability to edit and recompile source code on-the-fly
- Making it much easier to try potential bug fixes



- 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



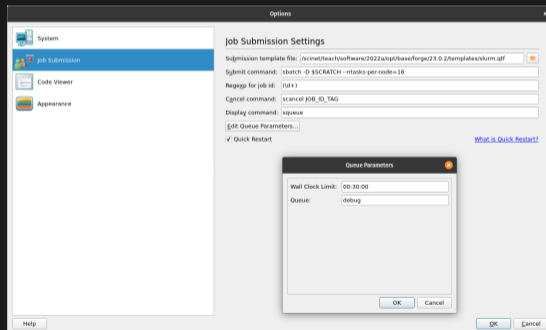
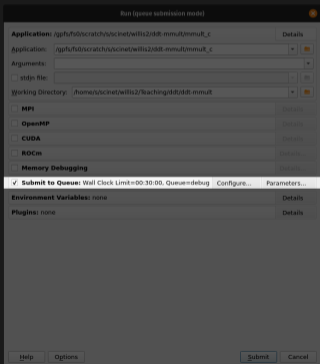
• Attach demonstration

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