EXERCISE 1
GETTING STARTED

Objectives

The purpose of this exercise is to :

- Learn how to get started with a simple code using Allinea DDT
- Gain knowledge of Allinea DDT GUI to debug simple issues.

Description

The code "cstartmpi" is a simple MPI code written in C. It is meant to be messy and to do basic independant tasks:

- initialize a dynamic array to certain values
- executes a calculation loop on a static array
- prints out the arguments entered as parameters
- sends out messages from all processes to process 0 in order to print a message.

At this stage, it is not necessary to understand what the code is doing. The focus will be the GUI of Allinea DDT and understand how we can easily retrieve information from the debugger in order to find out answers about bugs: where, who, how, and why?

Walkthrough

First, we will need to use the workload scheduler in order to start the code. Please do not run any "make" or "mpirun" command on the front-end node!

To submit a job to the compute nodes, please use the following command:

```
$ oarsub -I
```

This will give you direct access to a compute node. From here, you can anything !

Let's compile the application cstartmpi. There is a makefile for this in the cstartmpi directory. Please not that the "-g" option is necessary to generate debugging informations.

```
$ cd exercise1
```

```
make
```

If you run this application with 4 processes, everything is fine:

```
mpirun -n 4 ./cstartmpi.exe
```

Now try again with some arguments:

```
mpirun -n 4 ./cstartmpi.exe some input arguments
```

The program will abort as there has been a problem:

```
rank 0 in job 52 tenku_60773 caused collective abort of all
ranks
```

The next step is to bring this up in Allinea DDT and find out what happened. The quickest way to start is to run Allinea DDT almost identically to the way you launched MPI. The following command needs to be executed on the front-end node :

```
$ ddt -n 4 ./cstartmpi.exe some input arguments
```

The Allinea DDT GUI will appear - and it will have started your program. You can see the source code, and there is a colour highlighted line. This is the current location that processes are at. Initially, all processes are paused after MPI_Init.

At the top of Allinea DDT you will see a number of control buttons, a bit like a VCR (or PVR for the modern reader). If you hover the mouse over the control buttons, a tooltip will appear that gives the name of the button.

- **Play** – make the processes in the current group run until they are stopped.
- **Pause** – cause the processes in the current group to pause, allowing you to examine them.
- **Add Breakpoint** – adds a breakpoint at a line of code, or a function, that will cause processes to pause as soon as they reach that location.
- **Step Into** – will either step the current process group by a single line, or if the line involves a function call, it will step into the function instead.
- **Step Over** – will step the current process group by a single line.
- **Step Out** – will run the current process group to the end of their current function, and return to the calling location.

Press play to run the program.

Allinea DDT stops with an error message – indicating a segmentation fault.

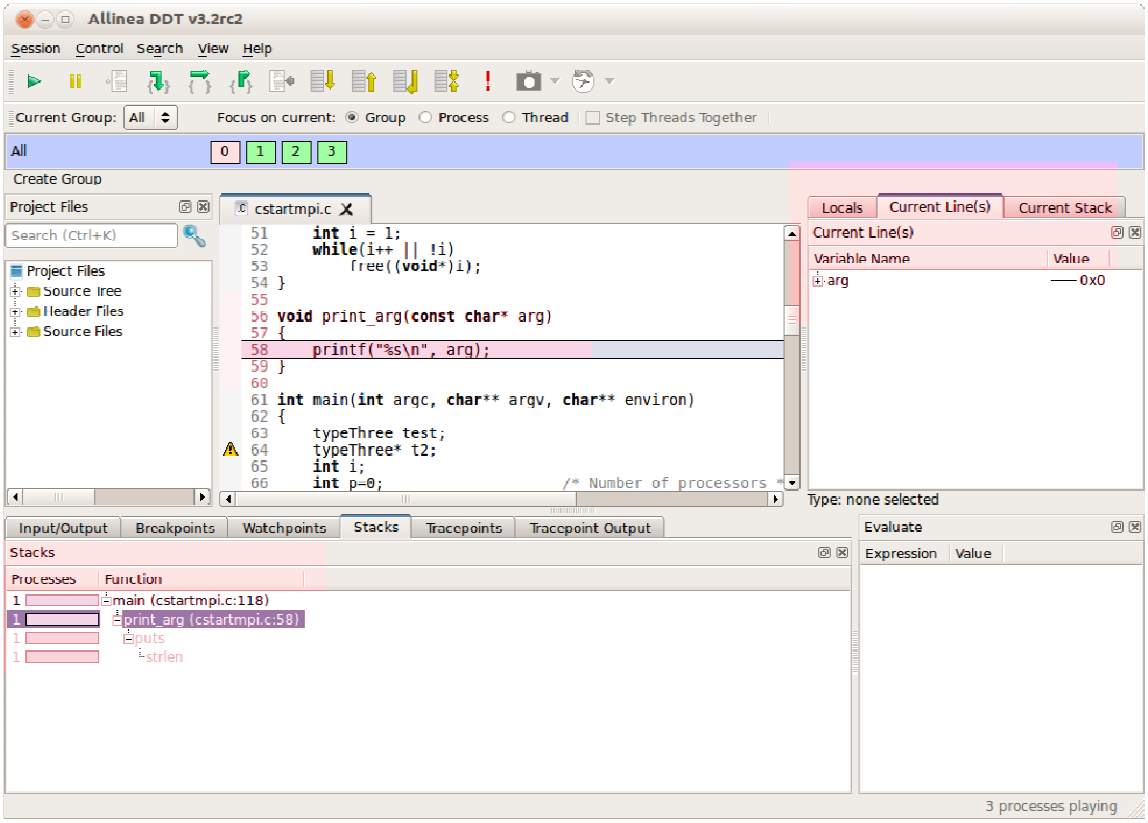

Illustration 1: Allinea DDT display after dismissing the dialog

The screenshot shows Allinea DDT after the dialog has been dismissed – we've colour highlighted the most important parts.

At the bottom of the GUI you can see the Stacks view (you may need to raise the tab by clicking on it to see it). This is tightly connected to the source code, also highlights in the screenshot. and shows where all the paused processes are: all the current function calls – higher points of the tree call the lower branches.

Often just looking at the variables at different points in the stack is enough to tell you why the program crashed.

In this case – you can see arg is the null pointer (0x0) which is invalid for its usage in the printf statement in the code. Hence, the program crashed because print_arg was called with the wrong thing.

Click on the "main" directly above the print_arg function in the Stack View.

This takes you to main which lets you see where that arg value comes from.

Now click on the "Locals" tab (on the right-hand side of the GUI) – you are seeing all the local variables.

Click on the "Current Line" tab to simplify and show only the variables on that line.

Click and drag between lines 113 and 118 in the source code to show all the variables in that region.

You can now see y is clearly incorrect - there aren't that many arguments (argc).

To find why is it wrong, examine the line 117: x is being checked against argc but y is being incremented.

> ***Fix the for loop condition in your favourite editor to read "y < argc" then recompile and re-run - now it works!***

Exercise

Our cstartmpi program has another bug: it runs fine for 4 processes, as we've just seen, but at larger numbers it segfaults again.

```
mpirun -n 5 ./cstartmpi.exe

rank 4 in job 60  tenku_60773   caused collective abort of
all ranks
```

Now it's up to you to find out why – you can join with your neighbour at one computer to run the program with Allinea DDT and work out what's going wrong and whether you can fix it!