

# ACRC How-To: Recipes for Using Allinea's DDT Parallel Debugger on BlueCrystal Phase 3

## Table of Contents

ACRC How-To: Recipes for Using Allinea's DDT Parallel Debugger on BlueCrystal Phase 3.....	1
Introduction.....	1
Building your Program.....	1
Enabling X11 Forwarding from the Cluster.....	2
Recipe #1: Using DDT to Submit a Job to the Queuing System.....	3
Recipe #2: Attaching DDT to a Job Already Running.....	10
Summary.....	13
Appendix.....	14

## Introduction

This document is *not* intended to be an introduction to using debuggers or, for that matter, an introduction to Allinea's DDT debugger. Rather, it is a practical guide on how you can start to use the DDT parallel debugger on BlueCrystal Phase 3. Allinea produce very good documentation and I would recommend that anyone new to DDT starts by reading their user guide:

- <http://content.allinea.com/downloads/userguide.pdf>

Those new to debugging altogether might like to read a more general introduction to the topic. Two guides that could be considered as primers are:

- <http://www.dirac.org/linux/gdb/01-Introduction.php>
- <http://heather.cs.ucdavis.edu/~matloff/UnixAndC/CLanguage/Debug.html>

There is an inherent tension between an interactive activity like debugging and the way in which we typically run jobs on a compute cluster, such as BlueCrystal. Since there are many users and competition for the compute resource of BlueCrystal, we submit our jobs to a fair-share scheduler which decides *when* those jobs are going to be run (and on which compute nodes). Thus it can be difficult to simply start up DDT and begin your debugging session. In response to this, I will present two different *recipes* for initiating a debugging session—one where we ask DDT to submit the job in question to the queue (and to wait until it is running), and another where we attach DDT to a job that we had previously queued and that is now running. Bear in mind, however, that neither approach is a panacea to the competition for compute resource—you will always have to wait longer if you wish to run a job using many processors.

## Building your Program

OK. Enough of the preamble, let's get going. You can access DDT by loading the module:

```
module add allinea/tools/4.0
```

When compiling your code, be sure to use the `-g` flag, so that your executable is instrumented with extra symbol information. Also be mindful that flags, such as `-O3`, which strongly optimise for speed, can make many changes to your code and hence make debugging harder. For example, `-O3`

may actually delete lines of your code, which will make adding a breakpoint to them rather tricky! For the examples below, I've used MVAPICH2 as my MPI library (set in my .bashrc):

```
module add mvapich2/gcc/64/1.7-qlc
```

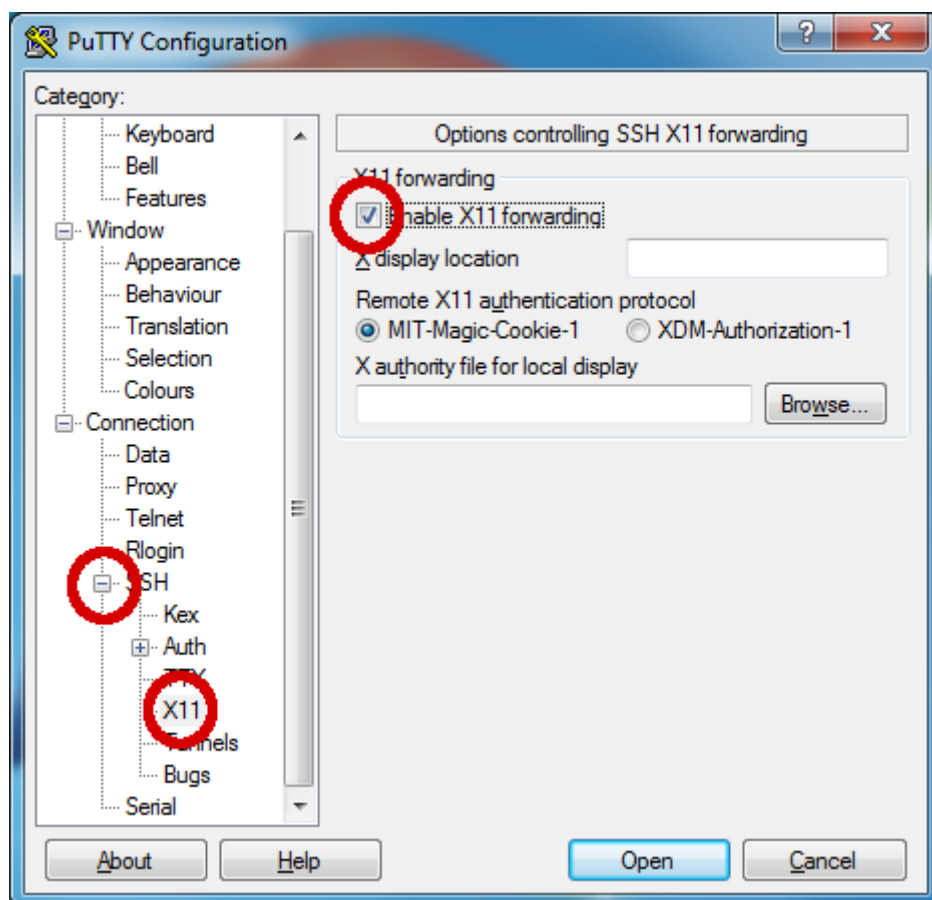
At this point we reach the nub of why I've used the word *recipe* in the title of this document. DDT will be harder or easier to use depending upon which of the MPI libraries available on BlueCrystal phase 3 you've selected. Through a process of trial and error, I've discovered that the default settings of DDT will work well with the above module loaded. This is not to say that DDT will not work with other MPI libraries, but that you'll have to work a bit harder to find the right configuration. One approach may be to follow my recipes to get started and then to branch out when you're comfortable.

## Enabling X11 Forwarding from the Cluster

In both the following recipes, we will start up the graphical user interface to DDT. So that the DDT windows can be seen on your screen, you will need to enable X11 forwarding when you connect to the cluster. From a Linux machine, or a Mac, you can do this by passing the `-X` flag to SSH:

```
ssh -X user@bluecrystalp3.acrc.bris.ac.uk
```

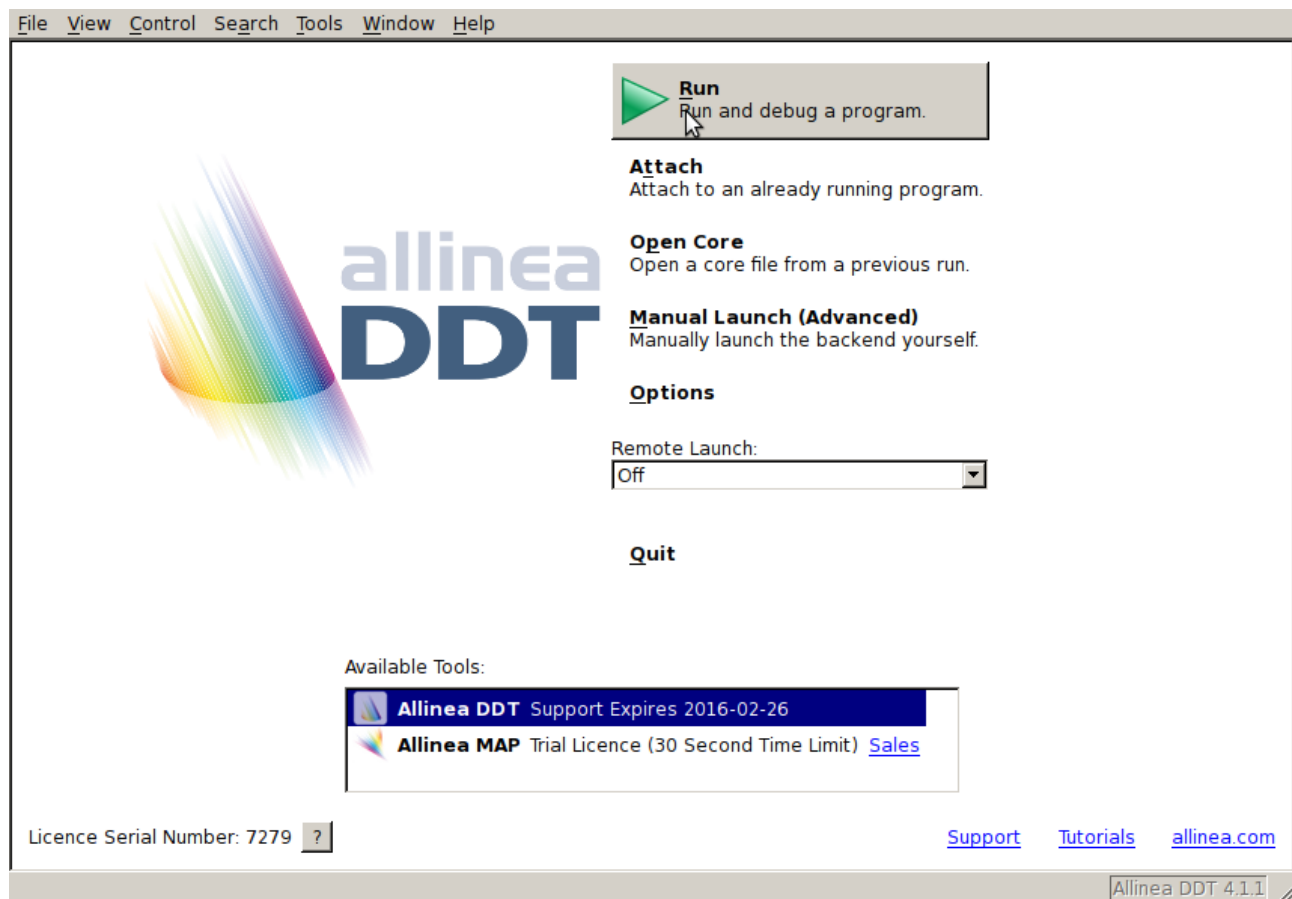
To connect from a computer running Windows, you'll need to start `xming`, which is the window manager and just runs in the background, and also **putty**, to make the SSH connection. When you start up putty, you will need to enter **bluecrystalp3.acrc.bris.ac.uk** as the hostame. In addition, you will need to expand the **SSH** menu entry (near the bottom in the left-hand pane), select the **X11** sub-menu and click the **Enable X11 forwarding** tickbox:



## Recipe #1: Using DDT to Submit a Job to the Queuing System

In this first recipe, we will ask DDT to submit our job to the queuing system. First we start up the DDT GUI by simply typing **ddt**.

After the splash screen, you will see the following interface. For this recipe, we're going to click 'Run':



Clicking Run will bring up the following configuration window. Set the path to your executable in the **Application** field of the Application (top) pane. Next click the Change button in the MPI pane.

**Application:** /panfs/panasas01/isys/ggdagw/hpc-course/mpi/example Details

**Application:** /s/ggdagw/hpc-course/mpi/examples/example1/hello\_world\_c Folder icon

**Arguments:** Dropdown arrow

☐ **stdin file:** Folder icon

**Working Directory:** Folder icon

☒ **MPI:** 4 processes, MVAPICH 2 Details

**Number of processes:** 4 Dropdown arrow

**Implementation:** MVAPICH 2, no queue Change...

**mpirun arguments:** Dropdown arrow

☐ **OpenMP** Details

☐ **CUDA** Details

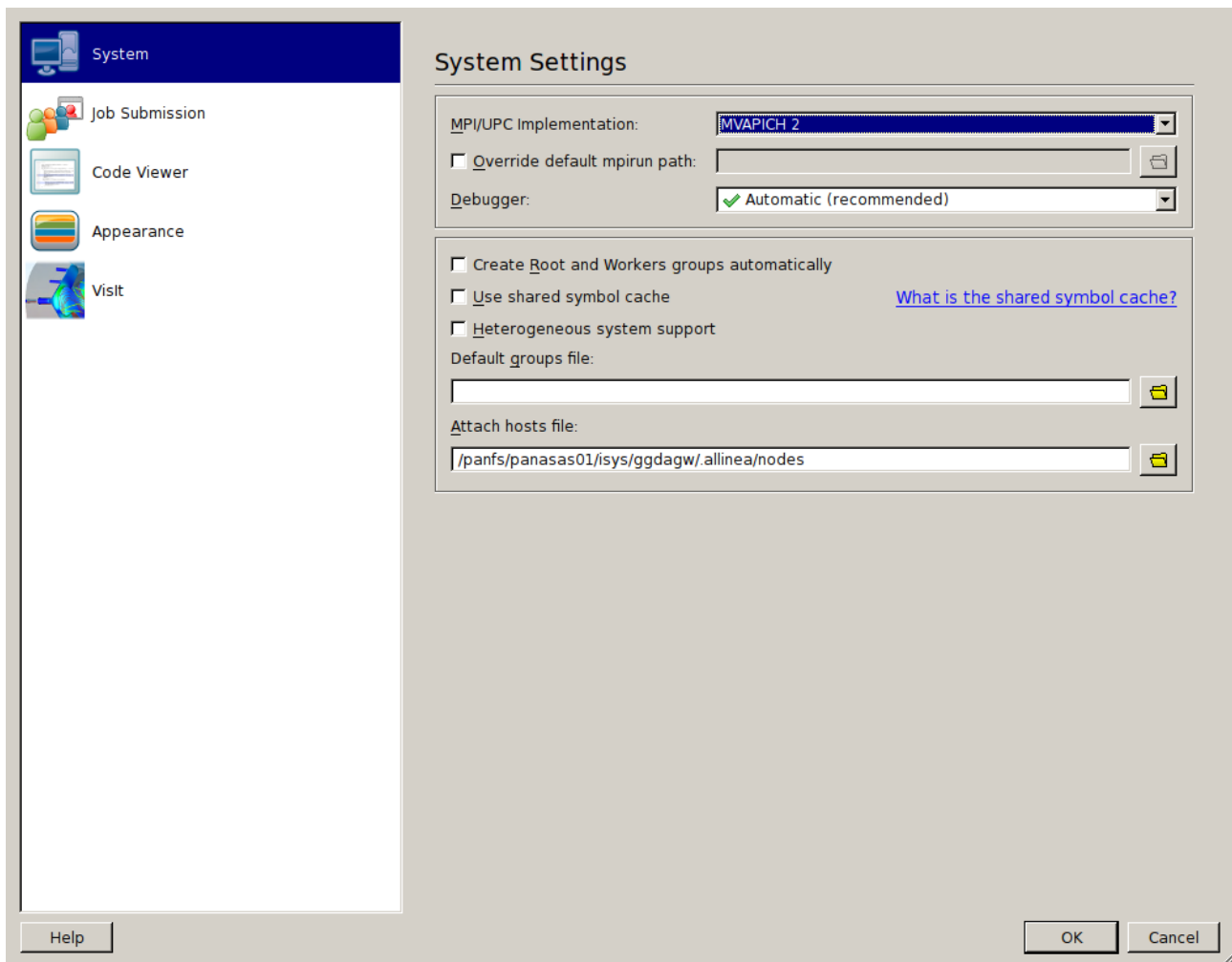
☐ **Memory Debugging** Details...

**Environment Variables:** none Details

**Plugins:** none Details

Help Run Cancel

On the **System** tab of the spawned window, set the **MPI implementation** to **MVAPICH 2**:



Next move to the **Job Submission** tab. Here you will need to:

- Tick the **Submit job through queue or configure own “mpirun” command box.**
- Change the **Regexp for job id** field from, “**our job (\d+)**” to just, “**(\d+)**”.
- Click the **Specify in Run window** radio buttons in both the **Number of nodes** and **Processes per node** panes.

**Job Submission Settings**

☒ Submit job through queue or configure own "mpirun" command

Submission template file:

Submit command:

Regexp for job id:

Cancel command:

Display command:

Number of processes (NUM\_PROCS\_TAG)

☒ Specify in Run window

☐ Calculate from number of nodes and processes per node

Number of nodes (NUM\_NODES\_TAG)

☒ Specify in Run window

☐ Calculate from number of processes and processes per node

Processes per node (PROCS\_PER\_NODE\_TAG)

☒ Specify in Run window

☐ Fixed:

[Edit Queue Submission Parameters...](#)

☐ Also submit scalar jobs through the queue

☒ Quick Restart

[What is Quick Restart?](#)

Help OK Cancel

Below is the resulting customised job configuration window. I've deliberately chosen to run the job with a modest number of processes so that I stand a chance to progressing quickly though the queue.

The screenshot shows a job configuration window with the following sections:

- Application:** /panfs/panasas01/isys/ggdagw/hpc-course/mpi/example (Details button)
- Application:** s/ggdagw/hpc-course/mpi/examples/example1/hello\_world\_c (Folder icon)
- Arguments:** (Empty text field)
- stdin file:** (Empty text field, Folder icon)
- Working Directory:** (Empty text field, Folder icon)
- MPI:** 4 processes, 2 nodes, 2 ppn, MVAPICH 2 (Details button)
  - Number of processes:** 4 (Spinner)
  - Number of Nodes:** 2 (Spinner)
  - Processes per Node:** 2 (Spinner)
  - Implementation:** MVAPICH 2, use queue (Change... button)
  - mpirun arguments:** (Empty text field)
- OpenMP:** (Details button)
- CUDA:** (Details button)
- Memory Debugging:** (Details... button)
- Queue Submission Parameters:** Wall Clock Limit=00:30:00, Queue: (Details... button)
- Environment Variables:** none (Details button)
- Plugins:** none (Details button)

Buttons at the bottom: Help, Submit, Cancel.

Clicking the Submit button will bring up a window showing the status of your job in the queue:

Your job has been submitted to the queue. Allinea DDT will continue automatically once the job has been started.

234804.master	6108	tf6460	0 Q gpu
234805.master	6110	tf6460	0 Q gpu
234806.master	6111	tf6460	0 Q gpu
234807.master	6112	tf6460	0 Q gpu
234808.master	6114	tf6460	0 Q gpu
234809.master	6115	tf6460	0 Q gpu
234810.master	6116	tf6460	0 Q gpu
234811.master	6117	tf6460	0 Q gpu
234812.master	6119	tf6460	0 Q gpu
234813.master	6121	tf6460	0 Q gpu
234814.master	6123	tf6460	0 Q gpu
234815.master	6124	tf6460	0 Q gpu
234816.master	6125	tf6460	0 Q gpu
234817.master	6126	tf6460	0 Q gpu
234828.master	run_rend.pbs	chgjb	0 Q veryshort
234831.master	submit_gpu.sh	jb1805	0 Q gpu
234834.master	incline.4lev.sh	ggs1c	04:08:40 C glaciol
234838.master	allingsK63b	ggdagw	00:00:00 C veryshort
234839.master	allinSUVF5X	ggdagw	0 Q veryshort

Waiting for job '234839' to start...

Help Cancel

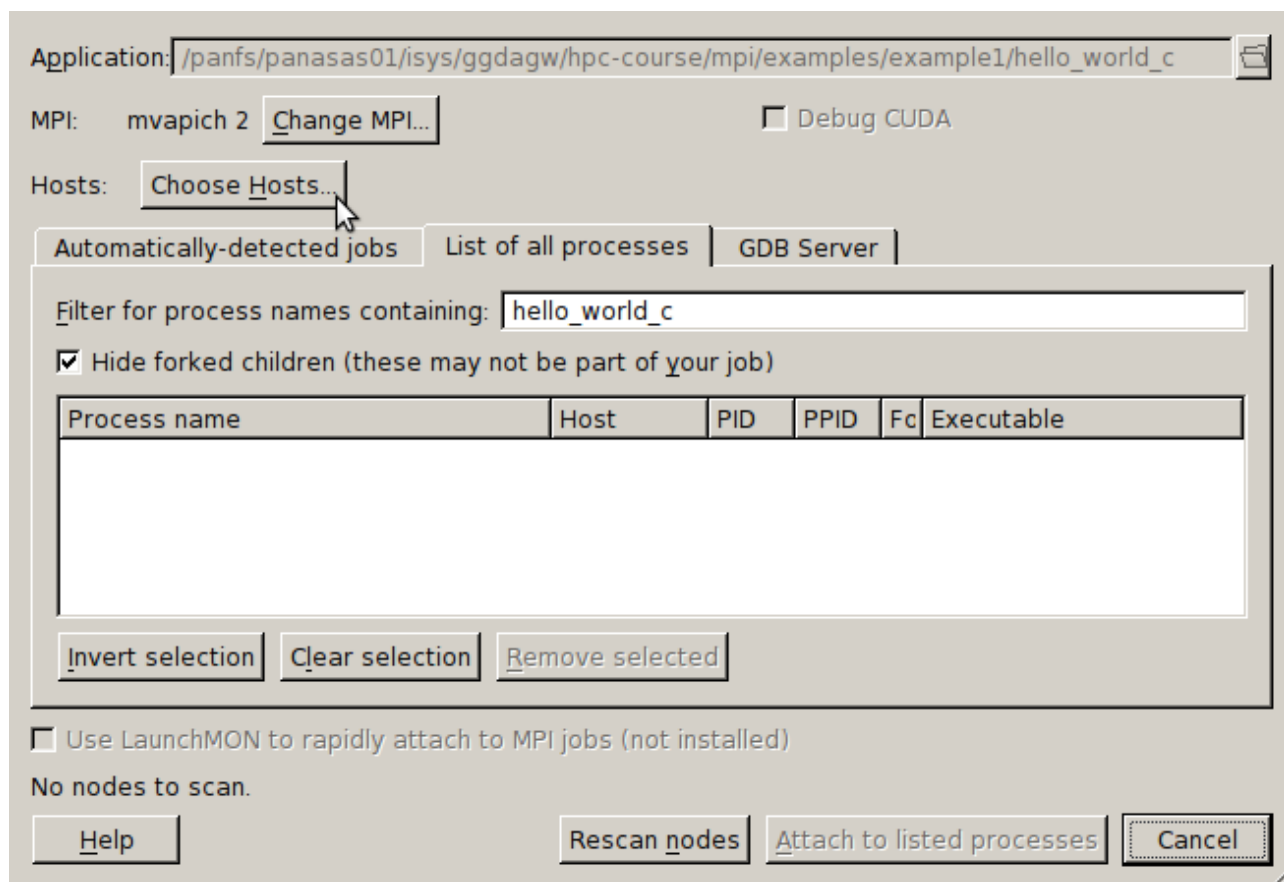


Once your job has started, you will be presented with the debugging interface below and you will be able to set breakpoints, step your code and inspect the values of variables, as per normal. Notice the clickable pink/red boxes at the top of the window. These indicate the rank of the process in the MPI cohort which you are currently debugging.

The screenshot displays the Visual Studio Code debugging interface for an MPI program. The top status bar indicates the current group is 'All' and the focus is on the current process. Below this, a row of pink/red boxes shows the ranks of the processes in the MPI cohort, with rank 2 selected. The main editor shows the source code of 'hello\_world.c' with a breakpoint set at line 51. The 'Locals' pane on the right shows the current line(s) and the values of local variables: 'hostname' is 'node31-006', 'rank' is 2, and 'size' is 4. The 'Stacks' pane at the bottom shows the call stack, with 'main (hello\_world.c:51)' at the top. The bottom status bar shows 'Ready'.

## Recipe #2: Attaching DDT to a Job Already Running

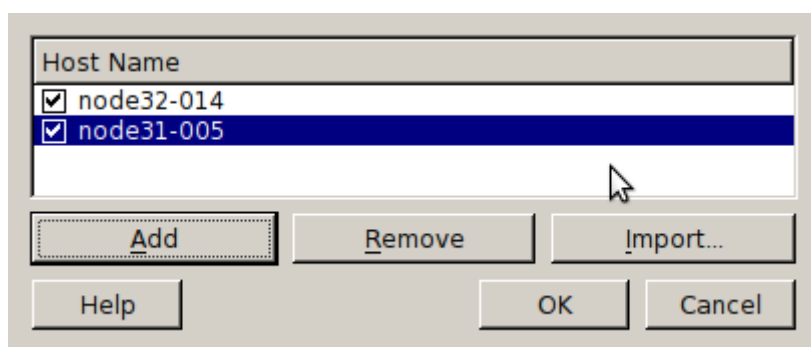
This recipe will be useful when you anticipate that your job will not start running until you are elsewhere (perhaps asleep at home!), or when you have a long running job that you might like to periodically attach to, inspect its progress, and then detach from again. Clicking **Attach** from the DDT start screen will bring up the following configuration window:



The screenshot shows the DDT configuration window. At the top, the 'Application' field is set to `/panfs/panasas01/isys/ggdagw/hpc-course/mpi/examples/example1/hello_world_c`. Below this, the 'MPI' field is set to 'mvapich 2' with a 'Change MPI...' button. A 'Debug CUDA' checkbox is present and unchecked. The 'Hosts' field has a 'Choose Hosts...' button. Below these are three tabs: 'Automatically-detected jobs' (selected), 'List of all processes', and 'GDB Server'. A filter box contains 'hello\_world\_c'. A checkbox 'Hide forked children (these may not be part of your job)' is checked. Below this is a table with columns: Process name, Host, PID, PPID, Fc, and Executable. The table is currently empty. Below the table are buttons for 'Invert selection', 'Clear selection', and 'Remove selected'. At the bottom, there is a checkbox 'Use LaunchMON to rapidly attach to MPI jobs (not installed)' which is unchecked. Below that, it says 'No nodes to scan.' and there are buttons for 'Help', 'Rescan nodes', 'Attach to listed processes', and 'Cancel'.

Again we choose MVAPICH2 as our MPI implementation. The code I used for this example is given in the Appendix. It is a simple 'hello, world' MPI program written in C. Since it is a very short program I have added an (empty) infinite loop to the code. If I had not done this, the job would have completed before I had the chance to connect DDT up to it.

After compiling the code, I submitted it to the queue and waited for it to start running (monitoring the queue using `qstat` or `showq`). Once running, I could determine which nodes it was running on using `qstat -n -u <username>`. Armed with this knowledge, I could click the **Choose Hosts** button and add the appropriate node names to the list:



The screenshot shows the 'Choose Hosts' dialog box. It has a list box titled 'Host Name' containing two entries: 'node32-014' and 'node31-005'. Both entries have a checked checkbox to their left. Below the list box are buttons for 'Add', 'Remove', and 'Import...'. At the bottom are buttons for 'Help', 'OK', and 'Cancel'.

Once the hosts are set, and the **Filter** field filled out with the name of your executable, the relevant processes will automatically appear in the window and you can click the **Attach to listed processes** button.

Application:

MPI: mvapich 2  ☐ Debug CUDA

Hosts: node31-005, node32-014

Filter for process names containing:

☒ Hide forked children (these may not be part of your job)

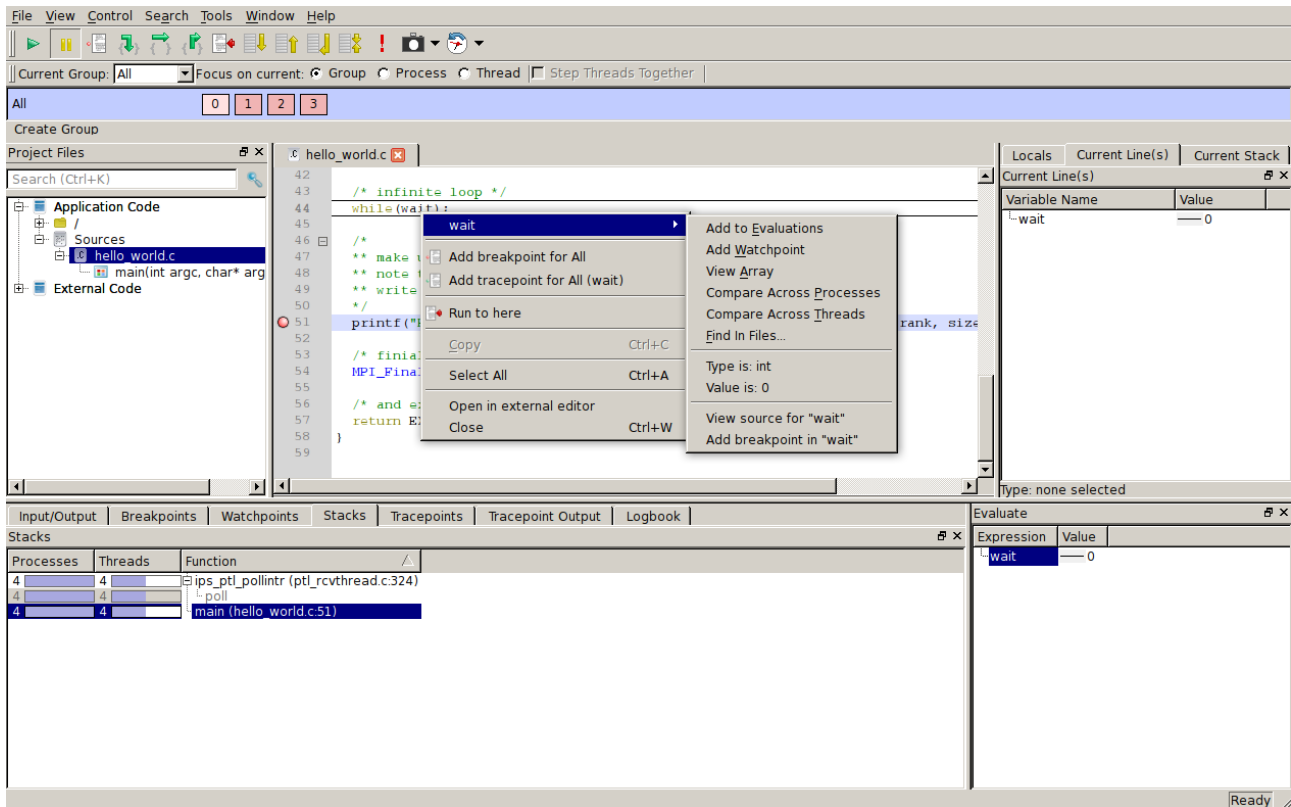
Process name	Host	PID	PPID	Fc	Executable
hello_world_c	node31-005	963	938	no	/panfs/panasas01/isys/...
hello_world_c	node31-005	964	938	no	/panfs/panasas01/isys/...
hello_world_c	node32-014	99073	99071	no	/panfs/panasas01/isys/...
hello_world_c	node32-014	99074	99071	no	/panfs/panasas01/isys/...

☐ Use LaunchMON to rapidly attach to MPI jobs (not installed)

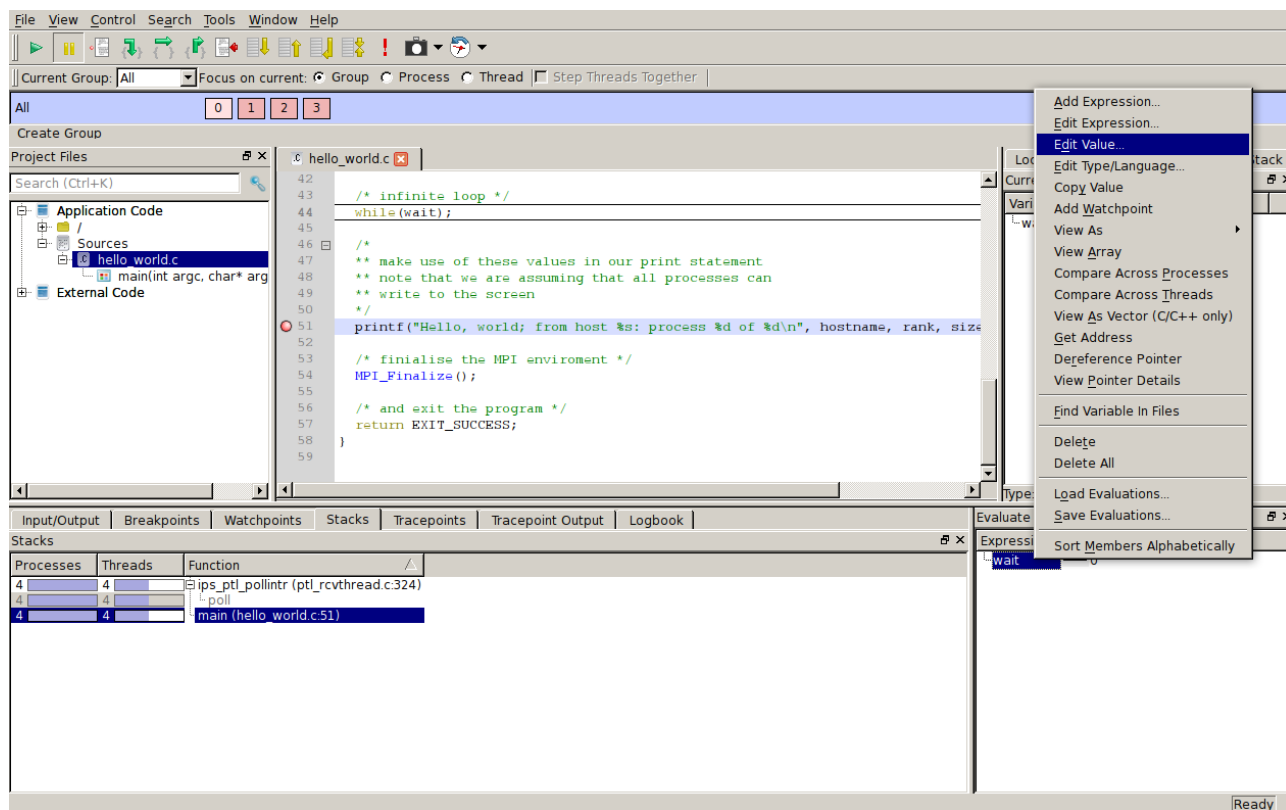
2 nodes scanned.

Once DDT has successfully attached itself to the list of processes, you will be presented with the debugging window.

Before we can start any meaningful debugging, however, we must first release processes from the infinite loop. To do this, right-click on the '**wait**' in '**while(wait)**' and select **Add to Evaluations**:



Then right-click on 'wait' from the Evaluate pane and select **Edit value**. Setting the wait variable to a value of zero will release processes from the infinite loop and allow you to debug your code in the normal fashion:



## Summary

We are fortunate to have access to a powerful parallel debugger such as Allinea's DDT on BlueCrystal phase 3. In this document, I have aimed to provide two concrete recipes for getting started with DDT, as well as pointers to further documentation for the tool. There is a learning curve associated with using any new tool and DDT is no different (although the GUI nature of the tool significant aids rapid familiarisation). However, I believe that time invested in learning how to use a debugger will be repaid many times over and I hope these recipes help you along that road.

## Appendix

Below is the code for a simple 'hello, world' MPI program that I used when writing this document:

```
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"

int main(int argc, char* argv[])
{
    int rank;                /* 'rank' of process among it's cohort */
    int size;                /* size of cohort, i.e. num processes started */
    int flag;                /* for checking whether MPI_Init() has been called */
    int strlen;              /* length of a character array */
    int wait = 1;            /* flag for infinite loop */
    enum bool {FALSE,TRUE}; /* enumerated type: false = 0, true = 1 */
    char hostname[MPI_MAX_PROCESSOR_NAME]; /* character array to hold hostname */

    MPI_Init( &argc, &argv );

    MPI_Initialized(&flag);
    if ( flag != TRUE ) {
        MPI_Abort(MPI_COMM_WORLD,EXIT_FAILURE);
    }

    MPI_Get_processor_name(hostname,&strlen);
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );

    /* infinite loop */
    /*while(wait);*/

    printf("Hello, world; from host %s: process %d of %d\n", hostname, rank, size);

    MPI_Finalize();

    return EXIT_SUCCESS;
}
```