Cornell University
Center for Advanced Computing

Parallel Debugging with DDT

Nate Woody

Debugging

- **Debugging** is a methodical process of finding and reducing the number of bugs, or defects, in a computer program or a piece of electronic hardware thus making it behave as expected. Debugging tends to be harder when various subsystems are tightly coupled, as changes in one may cause bugs to emerge in another.
- A **debugger** is a computer program that is used to test and debug other programs.
- This can be hard enough with a single local process and but get's many times more complicated with many remote processes executing asynchronously. This is why **Parallel Debuggers** exist.

Debugging Requirements

- In general, while debugging you may need to:
  - Step through code
  - Set/Run to breakpoints
  - Examine variable values at different points during execution
  - Examine the memory profile/usage
  - Provide source-level information after a crash
- For MPI and OpenMP Code we have additional requirements
  - All of the above for remote processes
  - Examine MPI message status
  - Step individual processes independent of the rest

DDT

- **DDT – D**istributed **D**ebugging **T**ool (www.allinea.com)
- A graphical debugger for scalar, multi-threaded and parallel applications for C, C++ and Fortran
- DDT's provides graphical process grouping functionality. DDT makes it really easy to assign arbitrary processes into groups which can be acted on separatly.
- Provides memory debugging features as well, things like checking pointers, array bounds, etc.
- Provides functionality to interact reasonable with STL components (ie you can see what a map actually contains) and create views for your own objects.
- Allows viewing of MPI message queues for running processes

Integrating with SGE

- DDT works by submitting a job to the Ranger development queue and attaches the process once it's started on a compute node. This works by a job template.
- The template then takes arguments provided by the DDT GUI to generate a batch script that executes your job with the appropriate arguments.
- The default configuration of DDT (which you get when you start DDT the first time) provides a default template that prepares an SGE job template.
- The template allows you enough flexibility to provide most arguments that you would need to set (walltime, account number, number of processors, etc).

login4% cat $DDTROOT/templates/sge.qtf

#!/bin/bash

DDT Job Template ($DDTROOT/template/sge.qpt)

```
#!/bin/bash
# QUEUE_TAG: {type=text,label="Queue",default=development}
# WALL_CLOCK_LIMIT_TAG: {type=text,label="Wall Clock Limit",default="0:30:00",mask="9:09:
09"}
# PROJECT_TAG: {type=text,label="Project"}
#$ -N DDTJOB
#$ -q QUEUE_TAG
#$ -o ddt.o$JOB_ID
#$ -cwd
#$ -j y
#$ -V
#$ -l h_rt=WALL_CLOCK_LIMIT_TAG
#$ -pe 16way NUM_PROCS_TAG
#$ -A PROJECT_TAG
# Run the program with the debugger
ibrun DDTPATH_TAG/bin/ddt-debugger DDT_DEBUGGER_ARGUMENTS_TAG
      PROGRAM_ARGUMENTS_TAG
```

Pulls settings from GUI

Parallel Environment setting is fixed!

Executes your job inside the debugger via ibrun

DDT Demo

- By far the best way to show what DDT can do is to start it up and look at it and show some things with it. Once we do this, we'll have everybody log in and make sure they can DDT started.
- We'll talk about:
  - Creating and altering groups
  - Stepping groups and processes
  - Show Cross-group comparison
  - Show Memory Usage/Profiling
  - Show MPI Queues
  - Show multi-dimensional array viewer

Starting DDT

- Login to ranger with an X tunnel

  $ ssh –X ranger.tacc.utexas.edu

- We need a binary compiled with debugging flags. If you don't have a binary already on ranger, you can get one from the train00 directory

  login3% mkdir ~/ddt

  login3$ cp ~train00/ddt_debug/debug_code.f .

- Ensure you have your preferred compiler loaded

  login3% module list

  login3% module unload mvapich

  login3% module swap pgi intel

  login3% module load mvapich

Starting DDT

- Compile with debugging flags

  login3% cd ~/ddt

  login3% mpif90 –g –O0 debug_code.f –o ddt_app

- Load the DDT module

  login3% module list

  login3% module load ddt

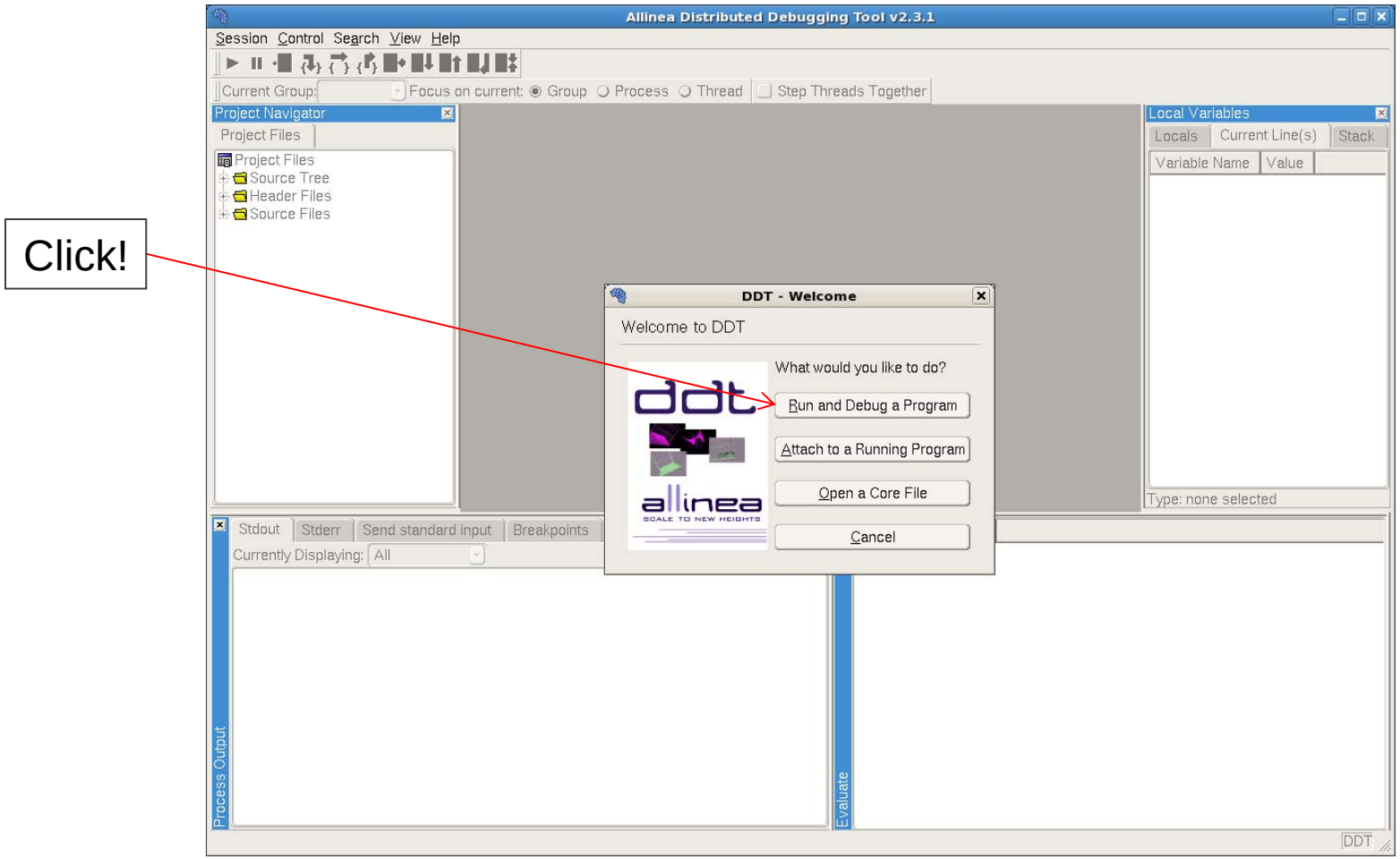
  login3% module list

  login3% echo $DDTROOT

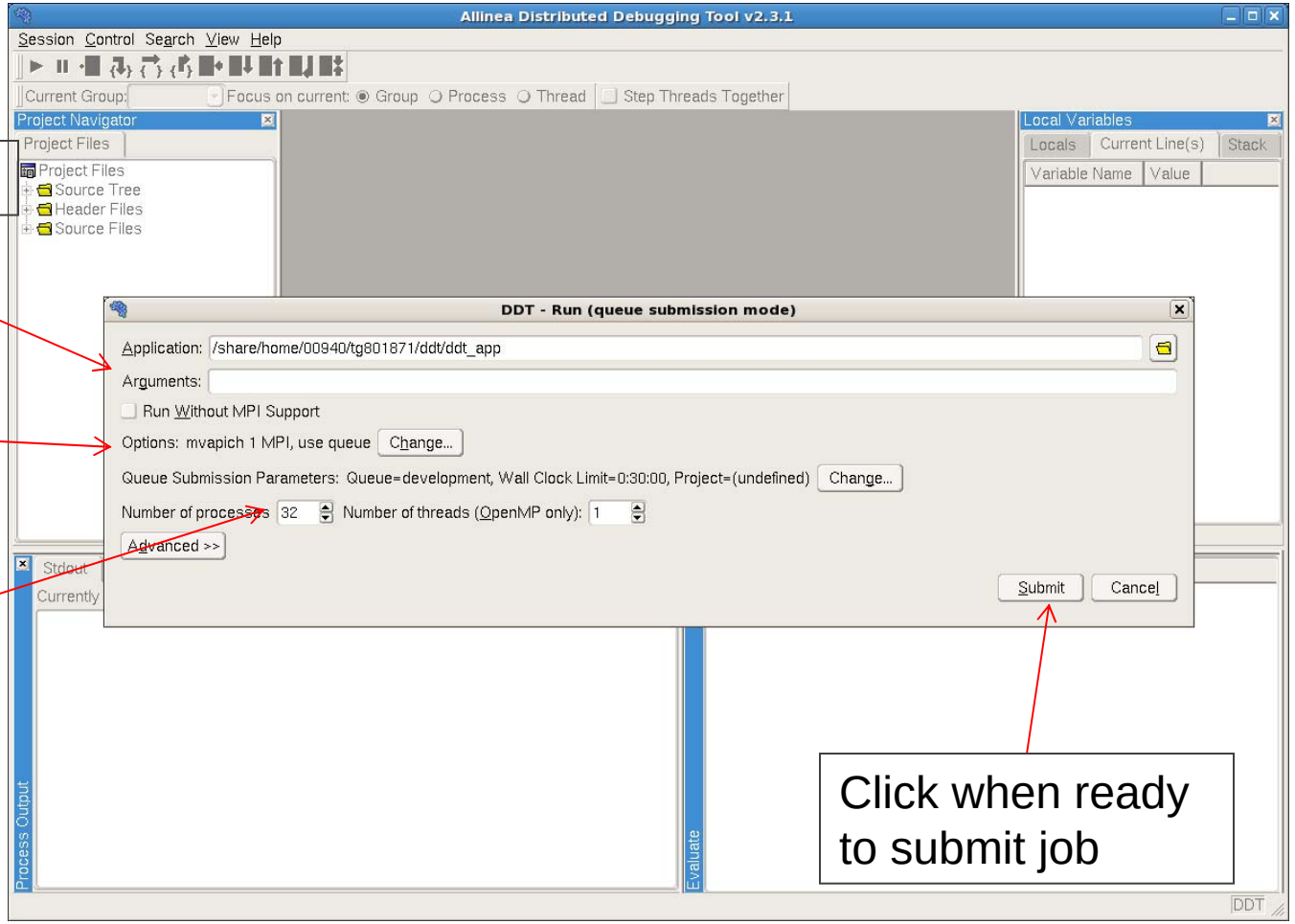
- Start DDT

  login3% ddt ddt_app

Starting DDT



Running a job



Add any arguments

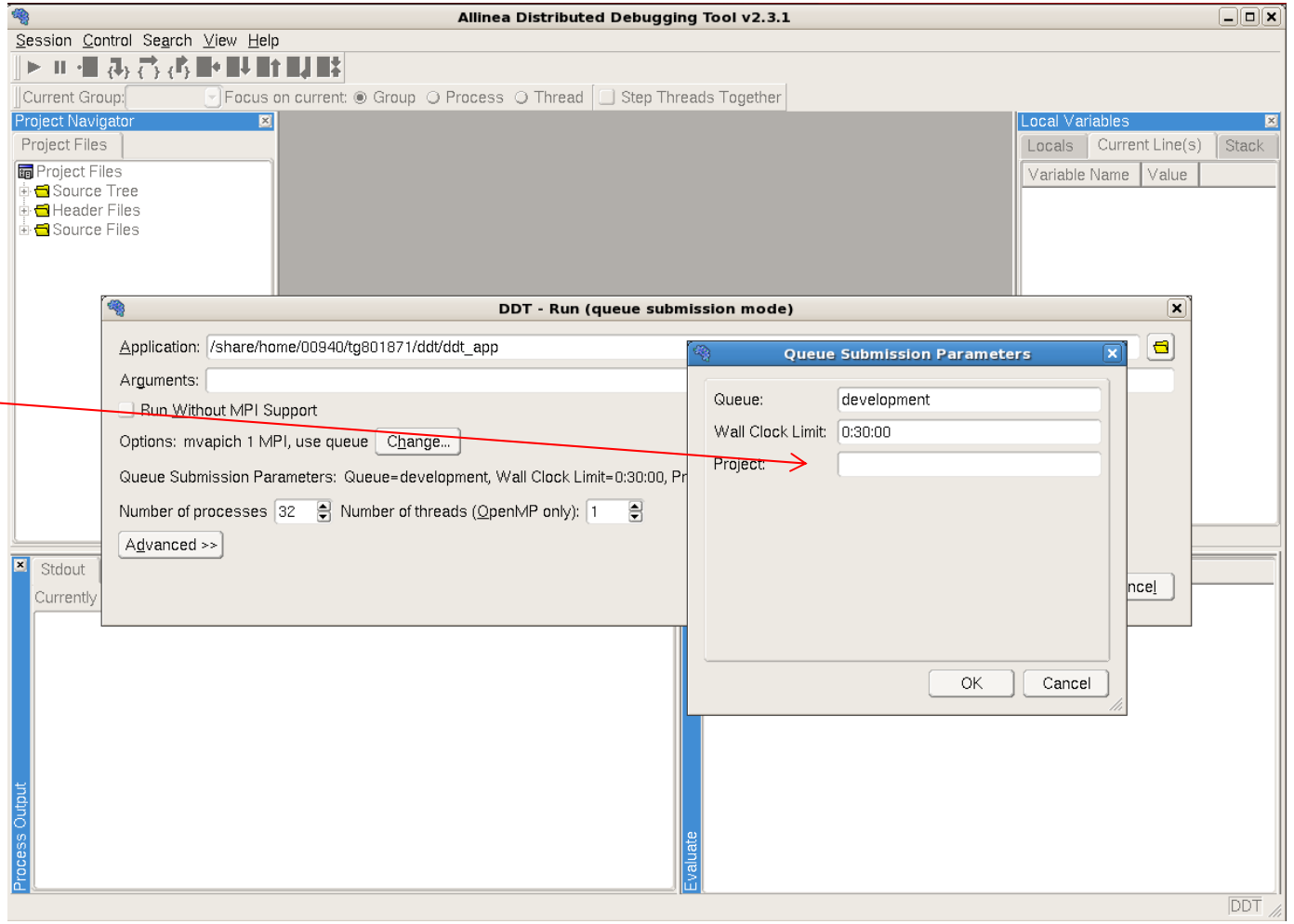
Ranger default

Sets number of nodes

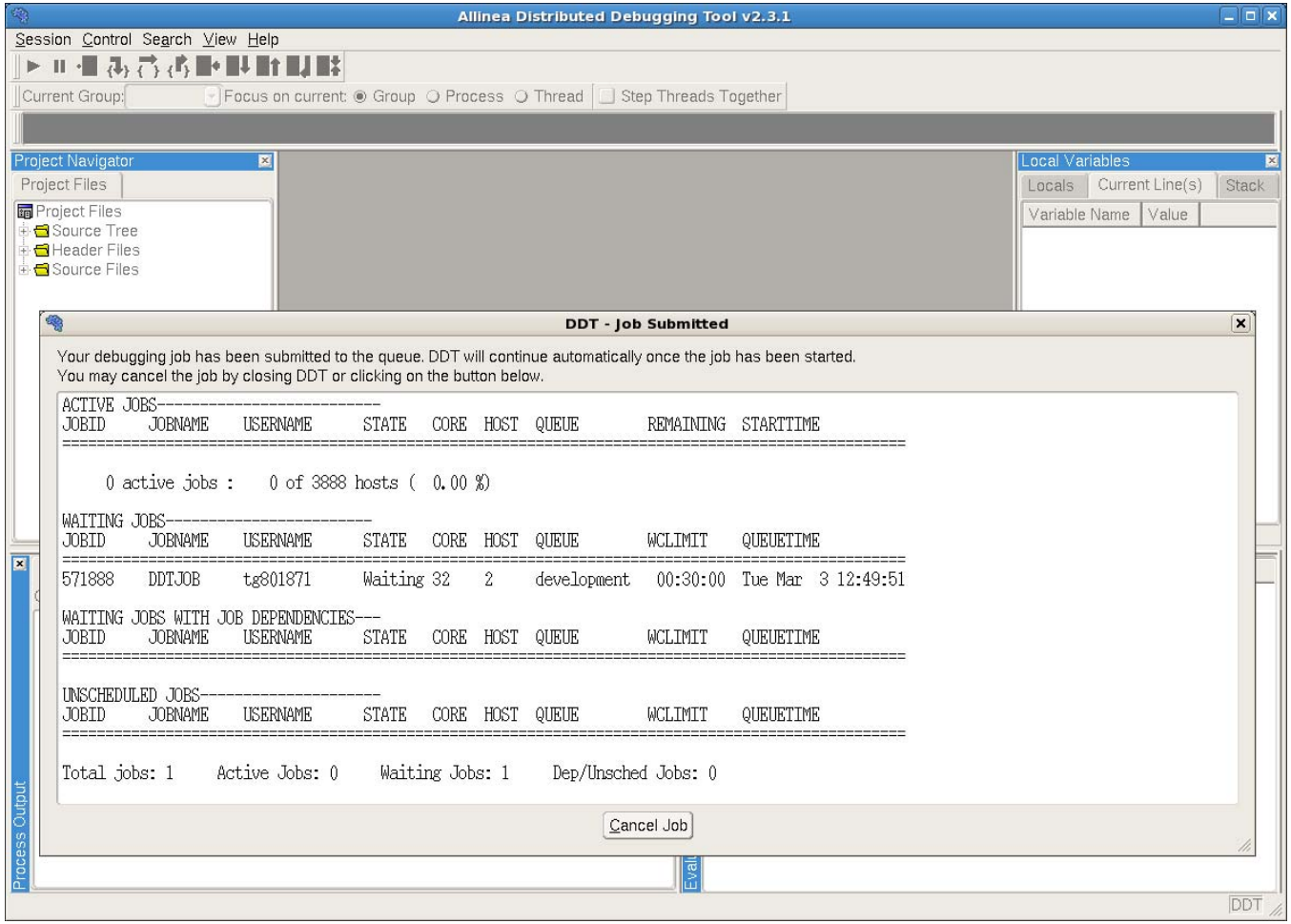
Click when ready to submit job

Account Name

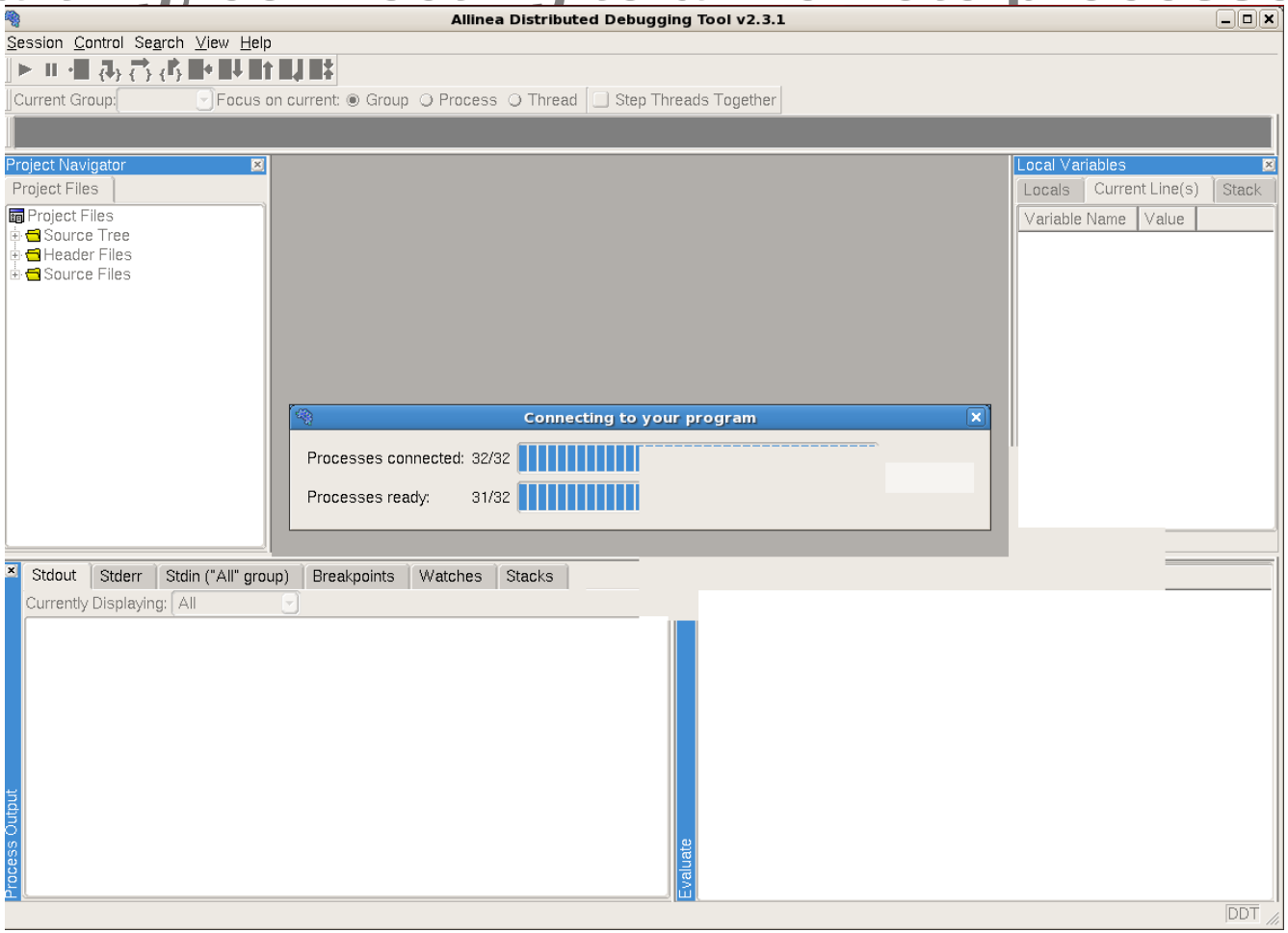
Provide allocation id (qsub -A value) then click OK



Waiting for job to start



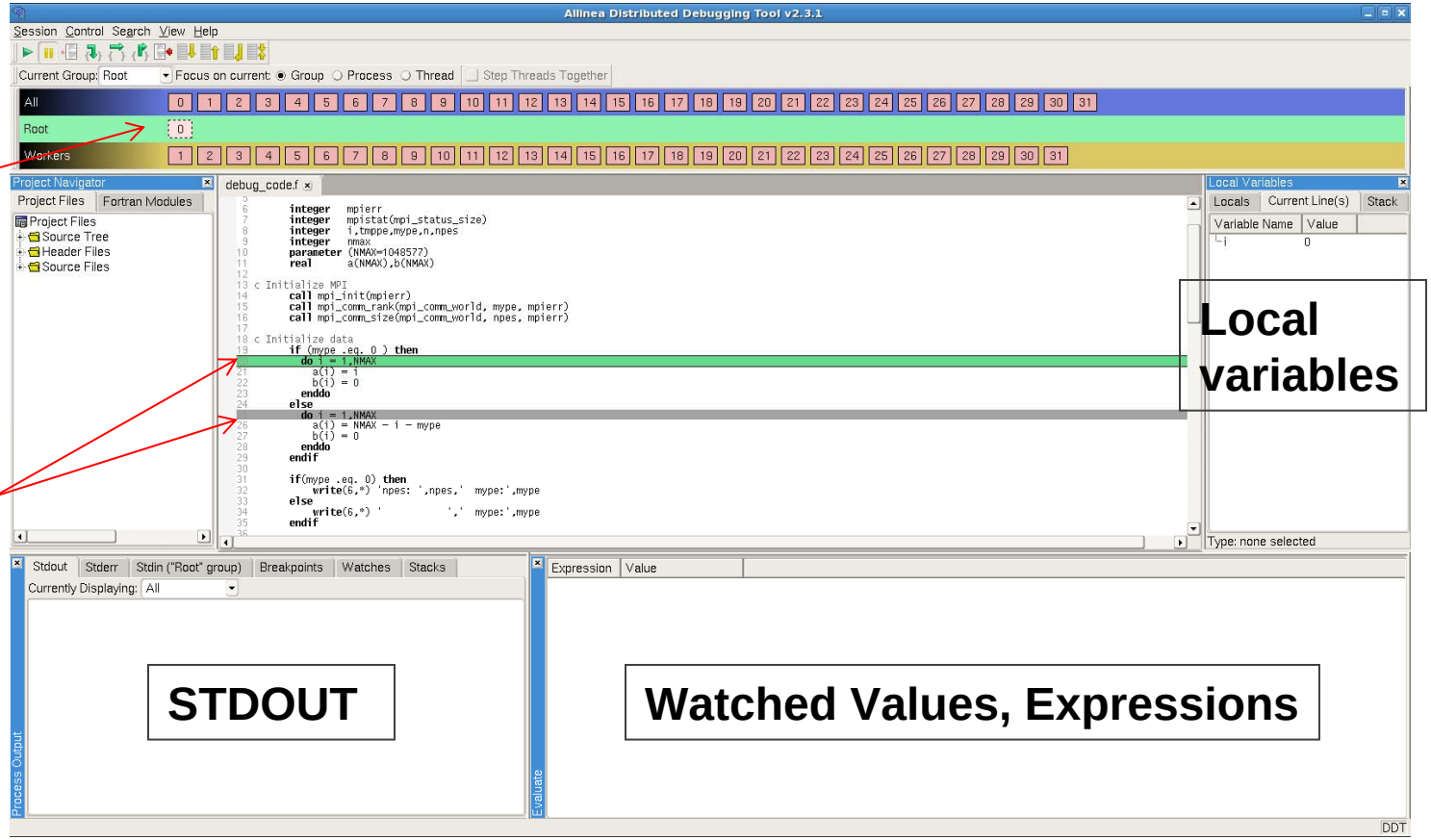
Job starting, connecting to all remote processes



Session started!

Root process is selected

Source locations of processes



DDT

- At this point, DDT should be up and running for you and you only need to load the DDT module and any configuration changes you made (ie Account name) will be saved for the next time you use it.
- It should feel very much like an IDE debugger, just with the added capabilities of viewing remote processes and MPI information.
- It wasn't shown, but this can be used just as well to debug OpenMP programs, though you may need to be careful when stepping through non-threaded sections. Check out the User Guide for any questions you have or request help through the TeraGrid help desk.
- UserGuide: http://www.allinea.com/downloads/userguide.pdf

  Or press F1 while running DDT to call up the help.