Debugging Parallel Programs with DDT

Presented by Rebecca Hartman-Baker

STAP Team

Outline

- Introduction to debugging parallel programs
  - Methods of debugging parallel programs
  - Why use a debugger?
  - What can a debugger do for me?
- Introduction to DDT
  - About Allinea DDT
  - DDT capabilities overview
  - Using DDT
- Hands-on session
  - Your code or my test codes

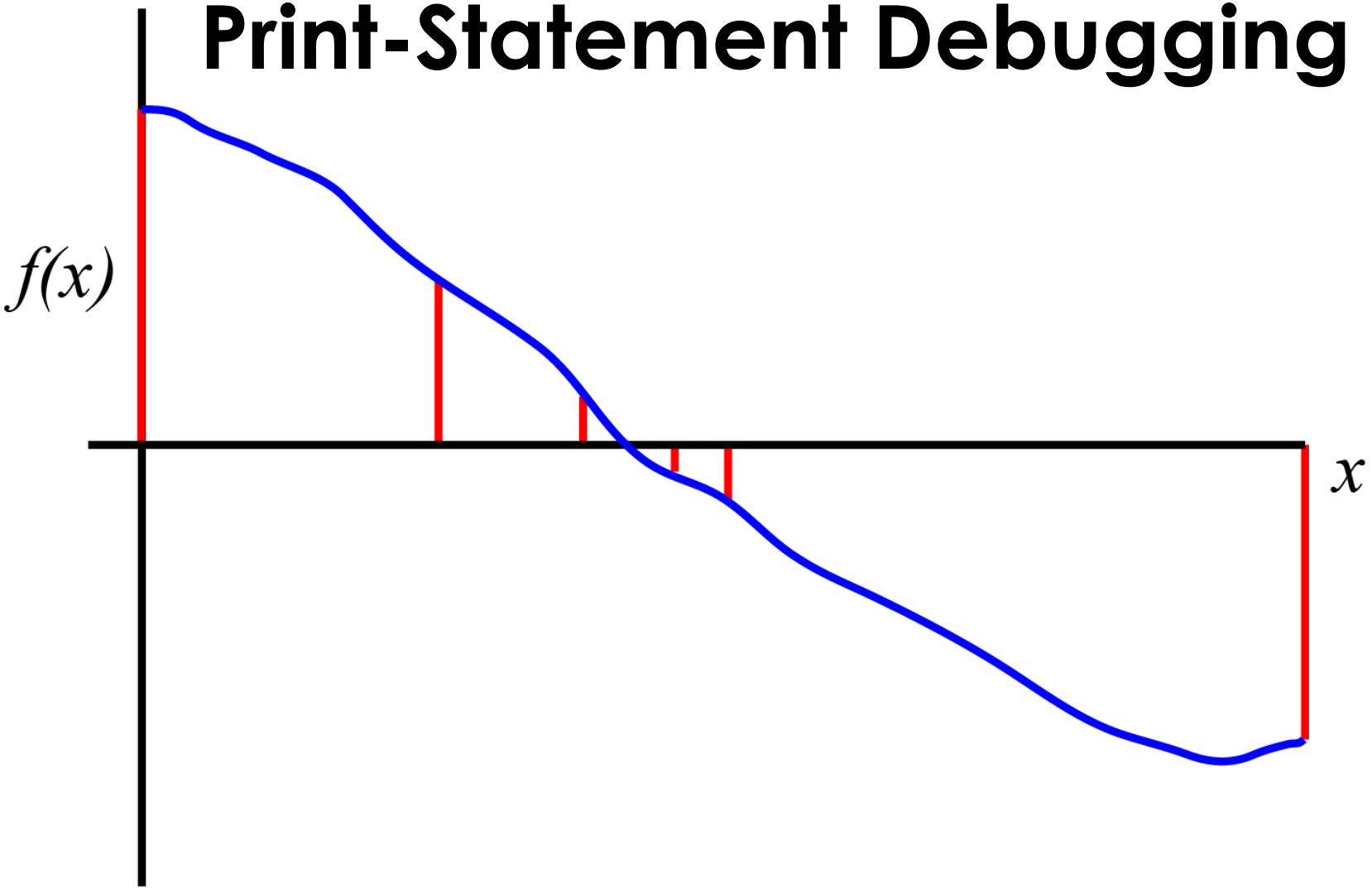
Debugging Parallel Programs

- Parallel programs are hard to debug
  - Serial programs are hard enough!
  - Parallel programming adds complexity
  - Must consider concurrency, synchronization, communication, blocking/non-blocking calls, etc.
- Ways to debug parallel programs
  - Print-statement debugging
  - Code reading and role-playing (I'm P0, you're P1)
  - Arts & Crafts/drawing
  - ***Using a debugger***

Print-Statement Debugging

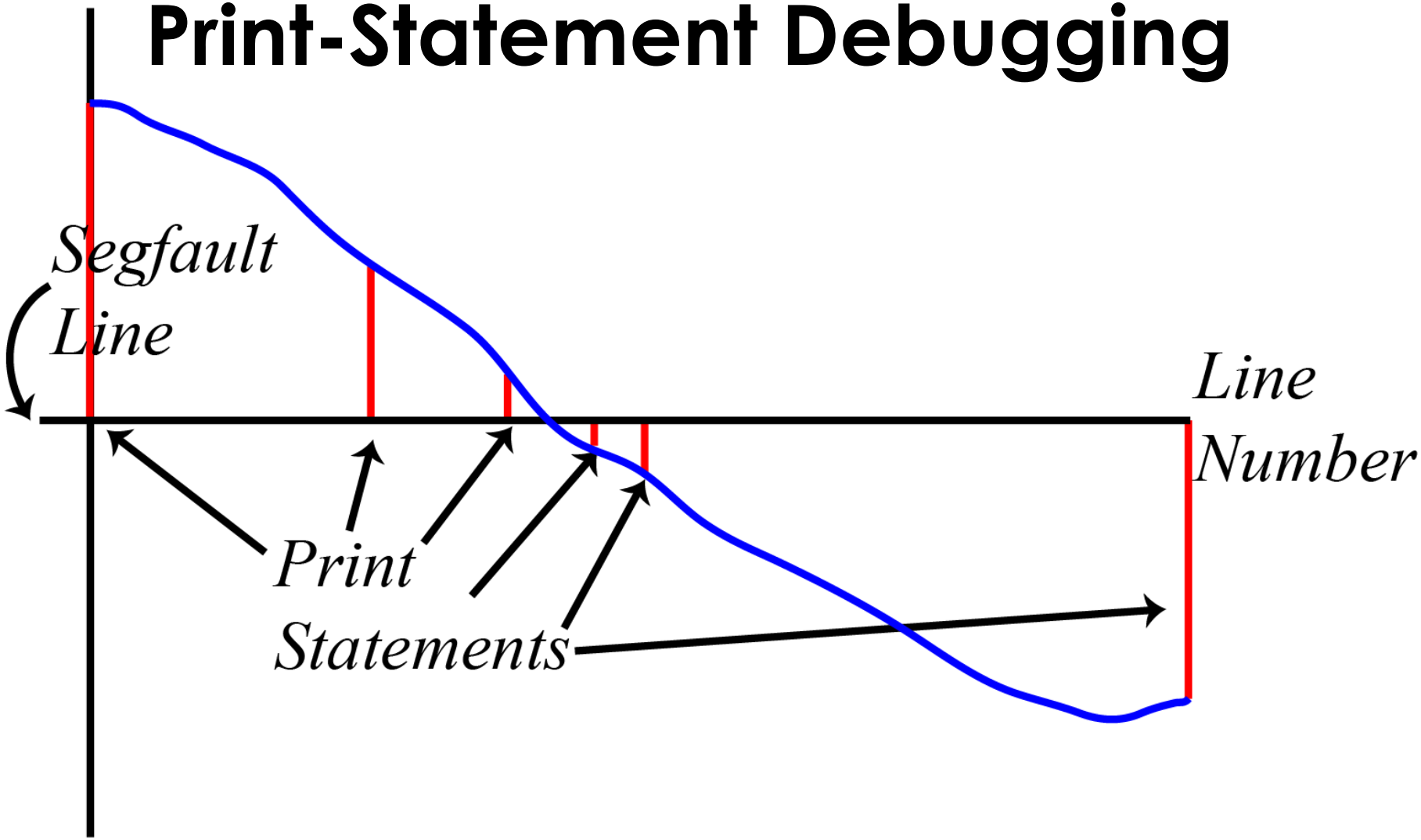
- Each processor dumps print statements to `stdout` or into individual output files, e.g., `log.0001`, `log.0002`, etc.
- *Advantages:* easy to implement, independent of platform or available resources
- *Disadvantages:* time-consuming, extraneous information in log files, tedious, not scalable (imagine 100K "`Hi from processor x`" messages?!?!)

Print-Statement Debugging



- Analogous to bisection method of root finding – very slow!

Print-Statement Debugging



- Analogous to bisection method of root finding – very slow!

Code-Reading and Role-Playing

- Find a group of willing participants (alternatively, do it all yourself)
- Read through code from point of view of each processor at each step

- Create a great big chart that maps behavior of every process
- *Advantages:* helps you to learn code, and to learn who your true friends are ;)
- *Disadvantages:* time-consuming, tedious, not scalable (unless you are very popular!)

Arts & Crafts/Drawing

- Print out P copies of code
- Cut and paste relevant lines of code on individual papers for each processor

- On large paper or poster board, align papers at synchronization points, draw lines representing communication, etc.
- *Advantages:* get to play with scissors and glue, learn how code works
- *Disadvantages:* time-consuming, space-consuming, not scalable!

Using a Debugger

- Invoke executable within debugger
- Typically, must recompile with `-g` flag and optimizations off (for best fidelity)

- Advantages:
  - Debugger will concentrate on the state of the variables in the code, you figure out what it means
  - Time-saving: can often isolate problem in a single trial (especially segfaults)
- Disadvantages:
  - Some debuggers not available on all platforms
  - Sometimes code fails only with optimizations on, can be hard to locate exact place where things go wrong

Why Use Debuggers?

- Debuggers can save time
  - With print-statement debugging, must insert print statements into code, sift through print statements, and find error
  - Debugger allows you to find the line where problem occurs in a single trial (no bisection)
- Complexity of bugs grows with complexity of code
  - More lines of code, more potential for errors
  - More complicated algorithm, more potential for errors
  - Parallelism only adds complexity
  - Some bugs occur only at scale

What can Debuggers Do for Me?

- Save time
- Allow user to concentrate on code, not background info
- View only variable values that are needed; view values not previously believed to be needed
- Pinpoint where things go wrong quickly
- Step through code and find cause of bug
- Run code at proper scale to find error

What Can Debuggers Do for Me?

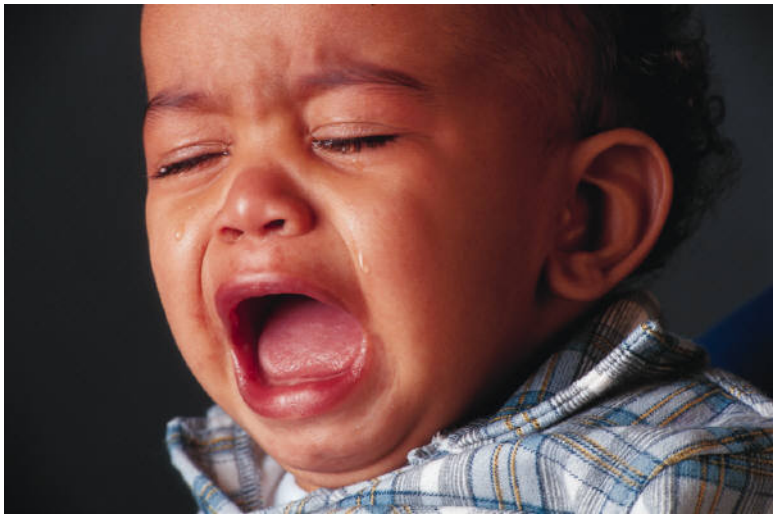
- Types of bugs
  - Segfaults
  - Memory errors
  - Algorithmic errors
  - Typos
  - “Improvements”
  - Things that happen only at scale
  - Etc.

A Come-to-Debuggers Moment

- There was once a grad student who could have been done with his/her dissertation SIX MONTHS EARLIER if he/she had been open to learning to use debuggers.
- “Oh no,” thought the grad student, “It will take me longer to learn to use a debugger than to just find this one last bug in my code.” But that was never the last bug. There was another, and another, and another…
- ***It takes an initial investment to learn to use a debugger, but that investment will more than pay off in no time.***

A Come-to-Debuggers Moment

- That grad student can't have his/her 6 months back, but we can learn from the sad story and invest some time learning to use a debugger!



That unfortunate grad student

You, having learned to use a debugger!

About Allinea DDT

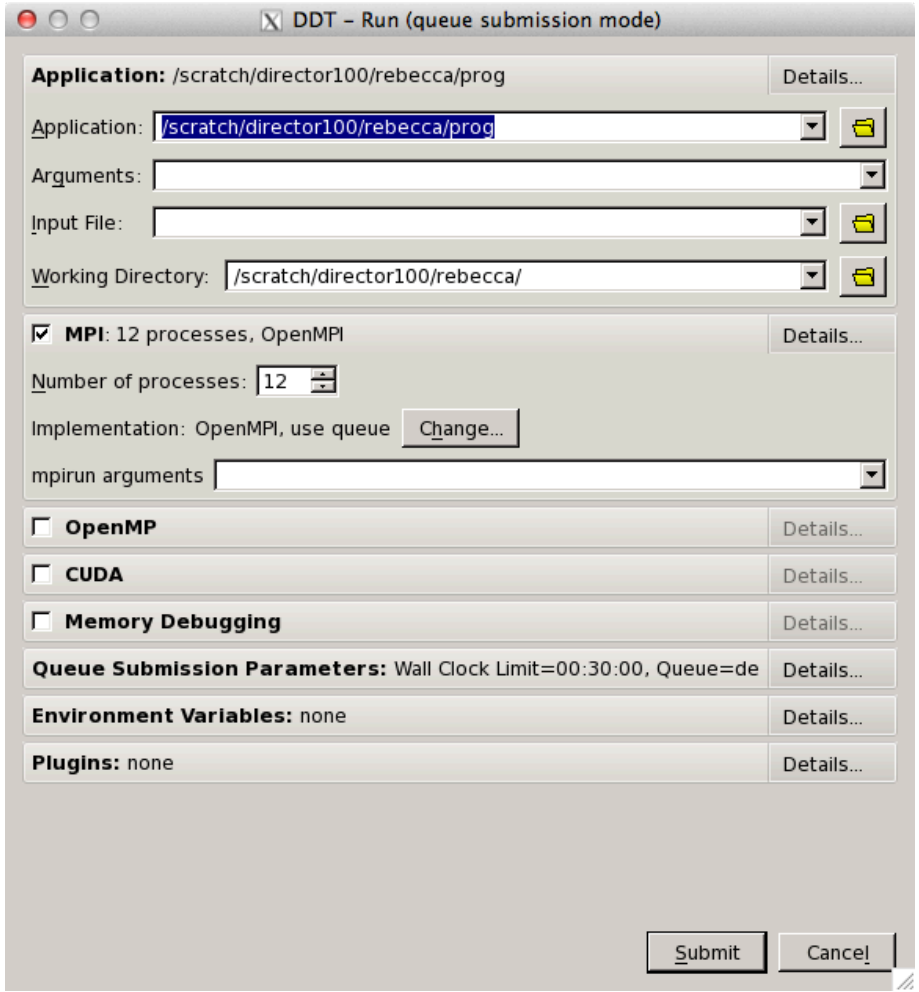
- Distributed Debugging Tool
- Capable of debugging codes written with MPI, OpenMP, threading, GPGPU
- Allinea collaborating with Oak Ridge National Laboratory (home of #1 Cray supercomputer) to create petascale debugging tool
- Available on all iVEC supercomputers
- Easy to use, intuitive

DDT Capabilities Overview

- Compile code with `-g` flag
- On iVEC systems:
  - `module load ddt`
  - `ddt &`
- Launch DDT from scratch directory
- Can run it within interactive job, or have DDT launch job

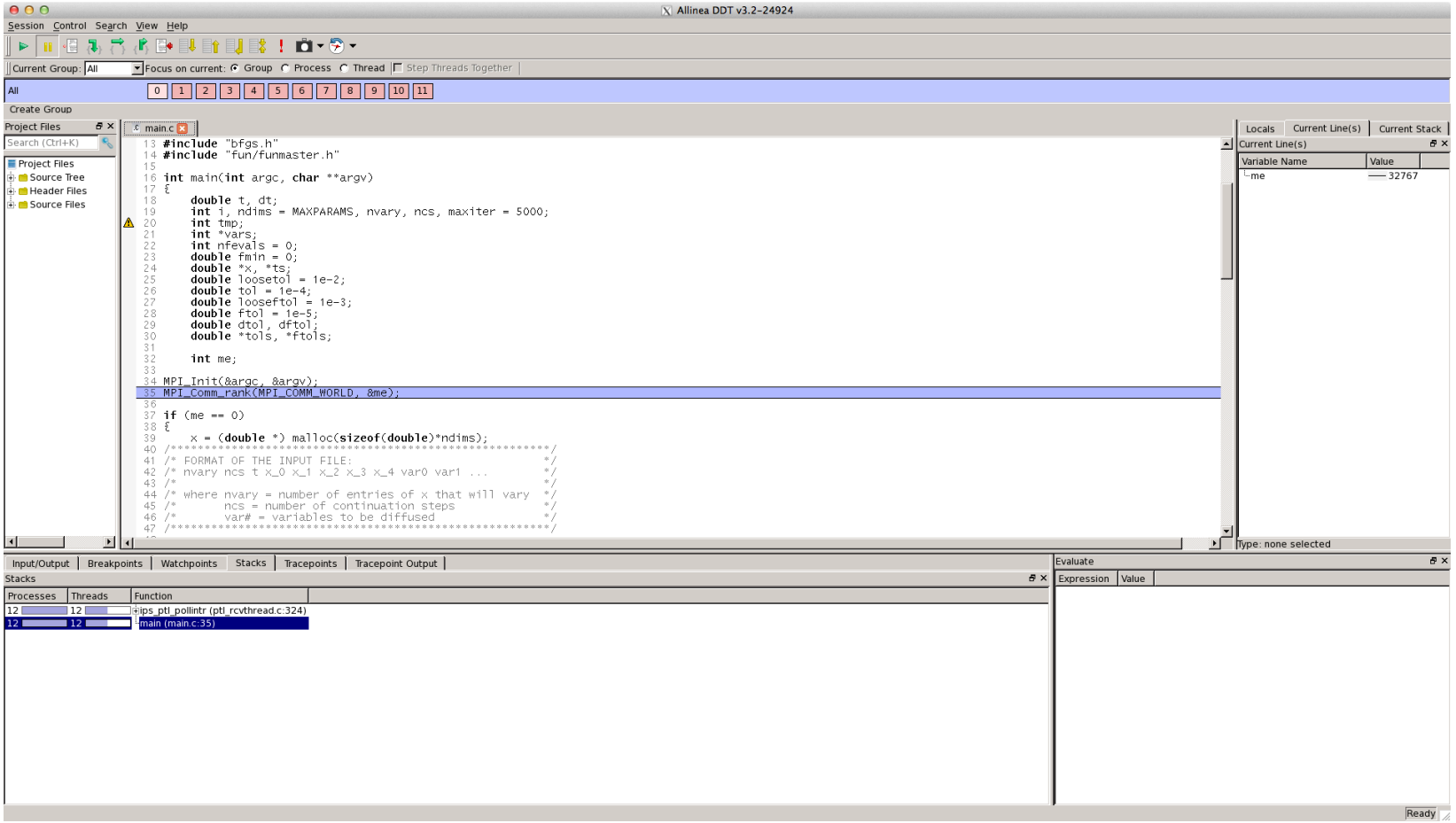
DDT Capabilities Overview

- Running a job
  - Enter application name
  - Can have DDT launch job, or run interactive job
  - Set arguments as necessary



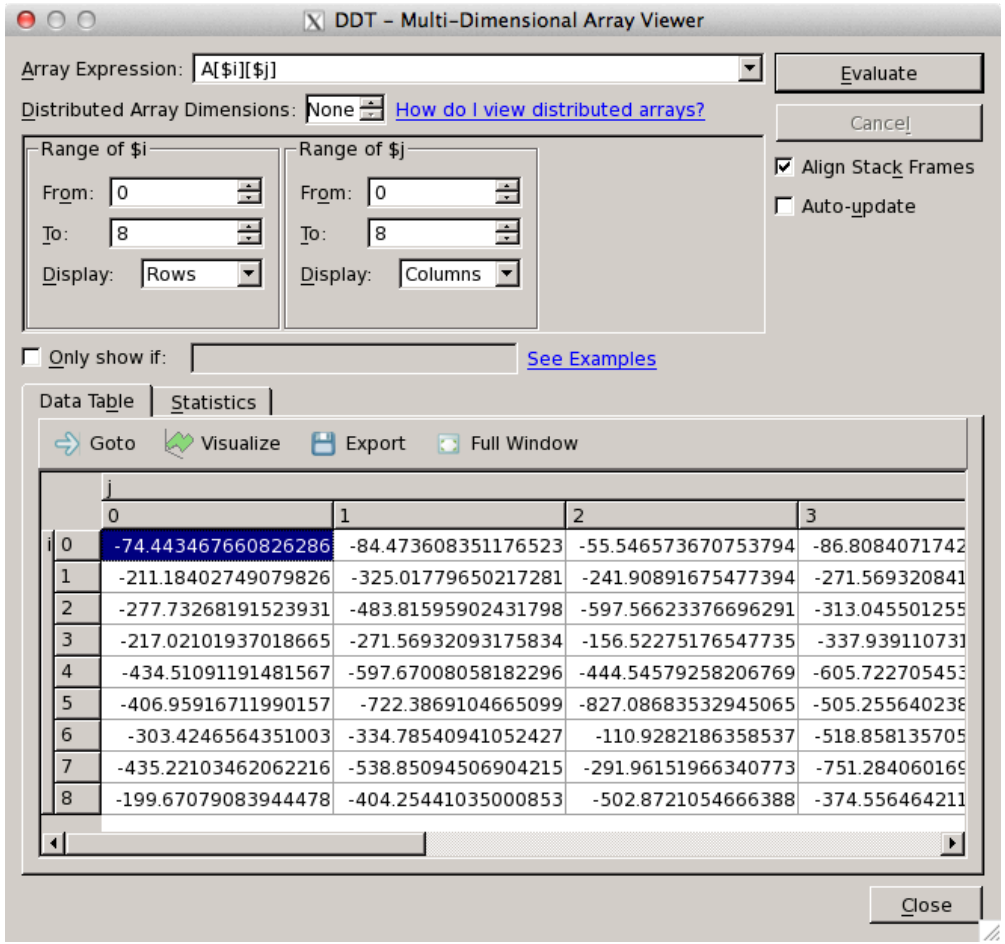
DDT Capabilities Overview

Opening Screen



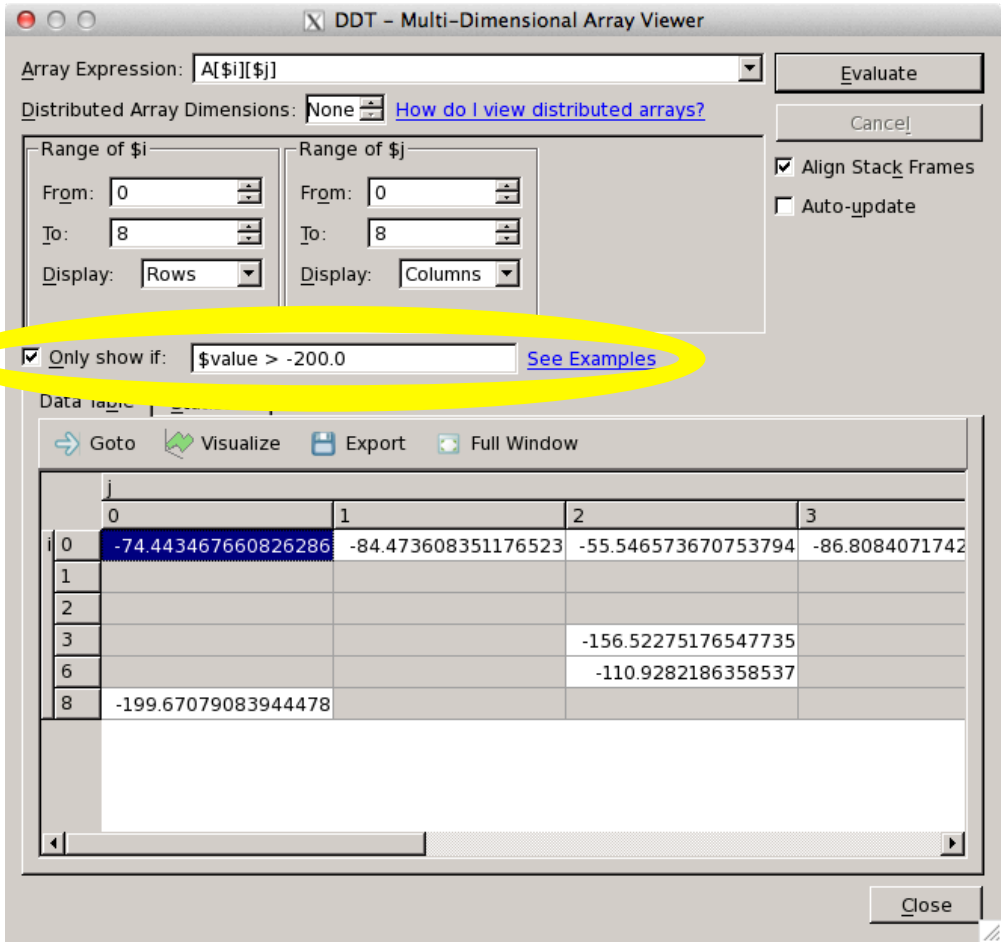
DDT Capabilities Overview

Array Viewer



DDT Capabilities Overview

Array Viewer – see all `A[i][j] > -200.0`



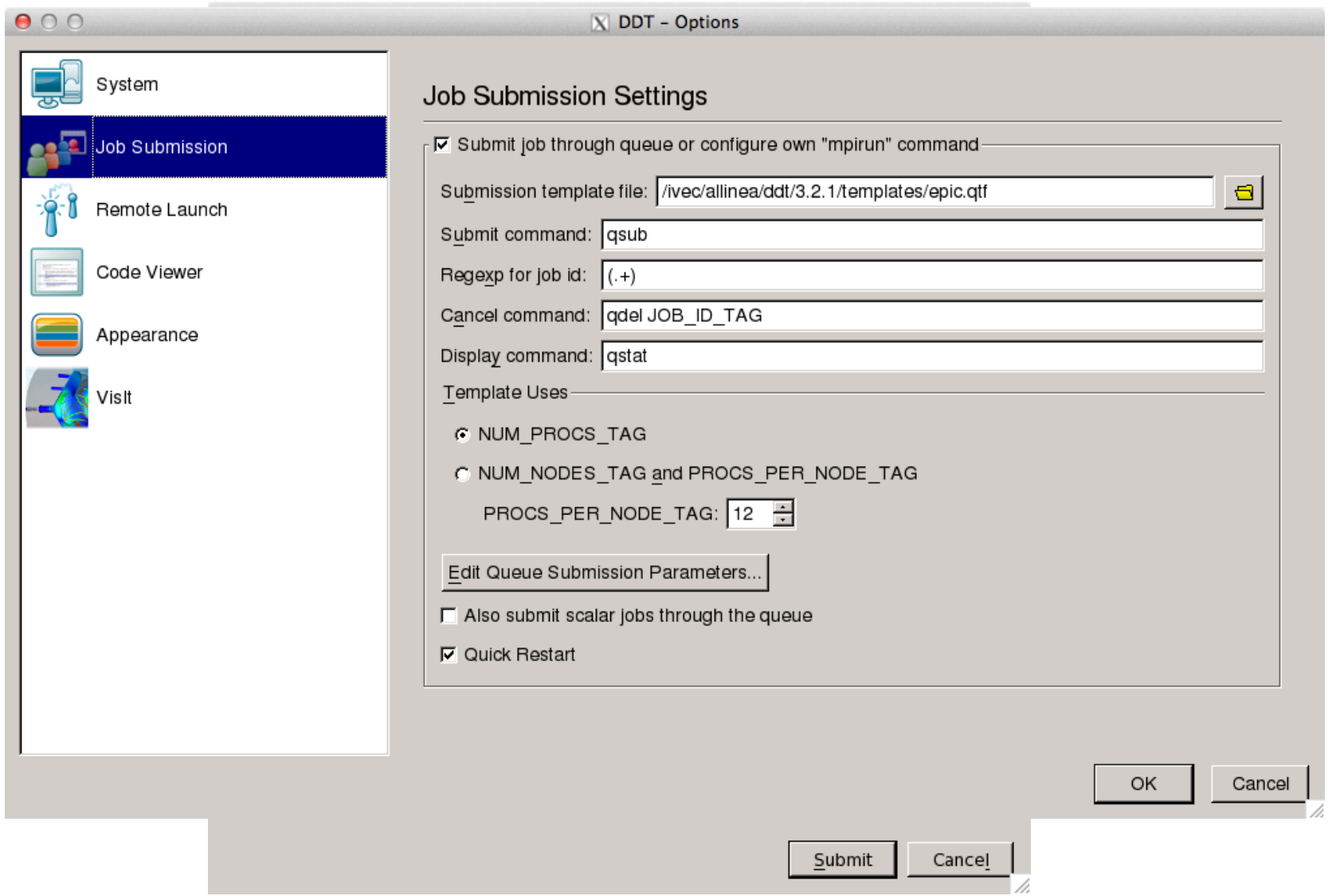
Using DDT: Step 1 -- Compiling

- Compile your code with the usual compiler and `-g` flag
  - Works better if all optimizations turned off
  - For some compilers, debug flag automatically turns off optimizations
  - If optimizations are on, line numbers may be misaligned or inexact

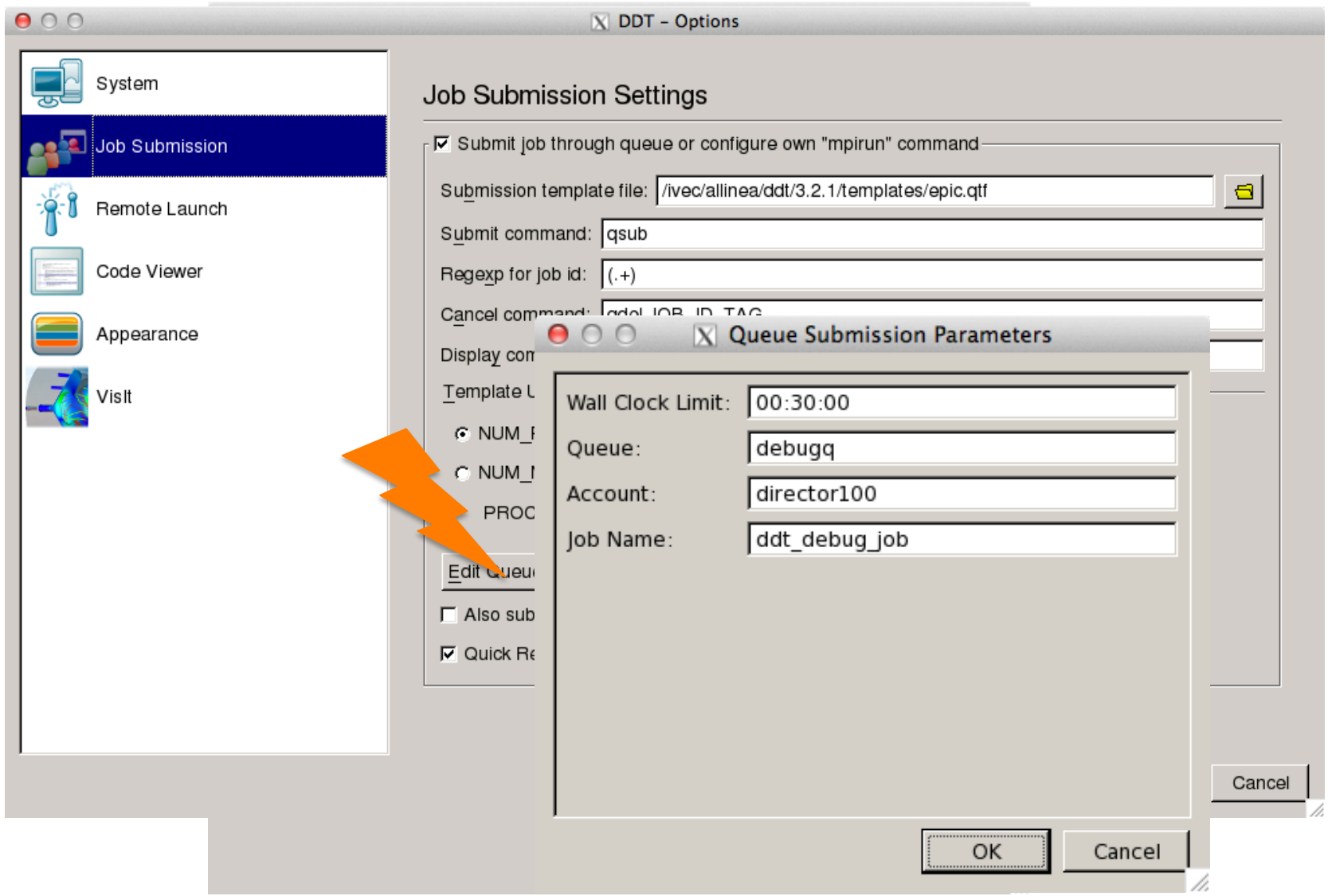
Using DDT: Step 2 -- Running

- You must have logged in with flags to allow X-forwarding
  - `ssh -X user@epic.ivec.org` (linux)
  - `ssh -Y user@epic.ivec.org` (mac)
  - Verify X-forwarding by invoking `xterm &` – if a terminal window appears, X-forwarding works (close it and proceed)
- `module load ddt`
- DDT can launch parallel interactive jobs for you
- Or, you can launch the interactive job and run DDT inside (I prefer this)

Setting Queuing Parameters



Setting Queuing Parameters

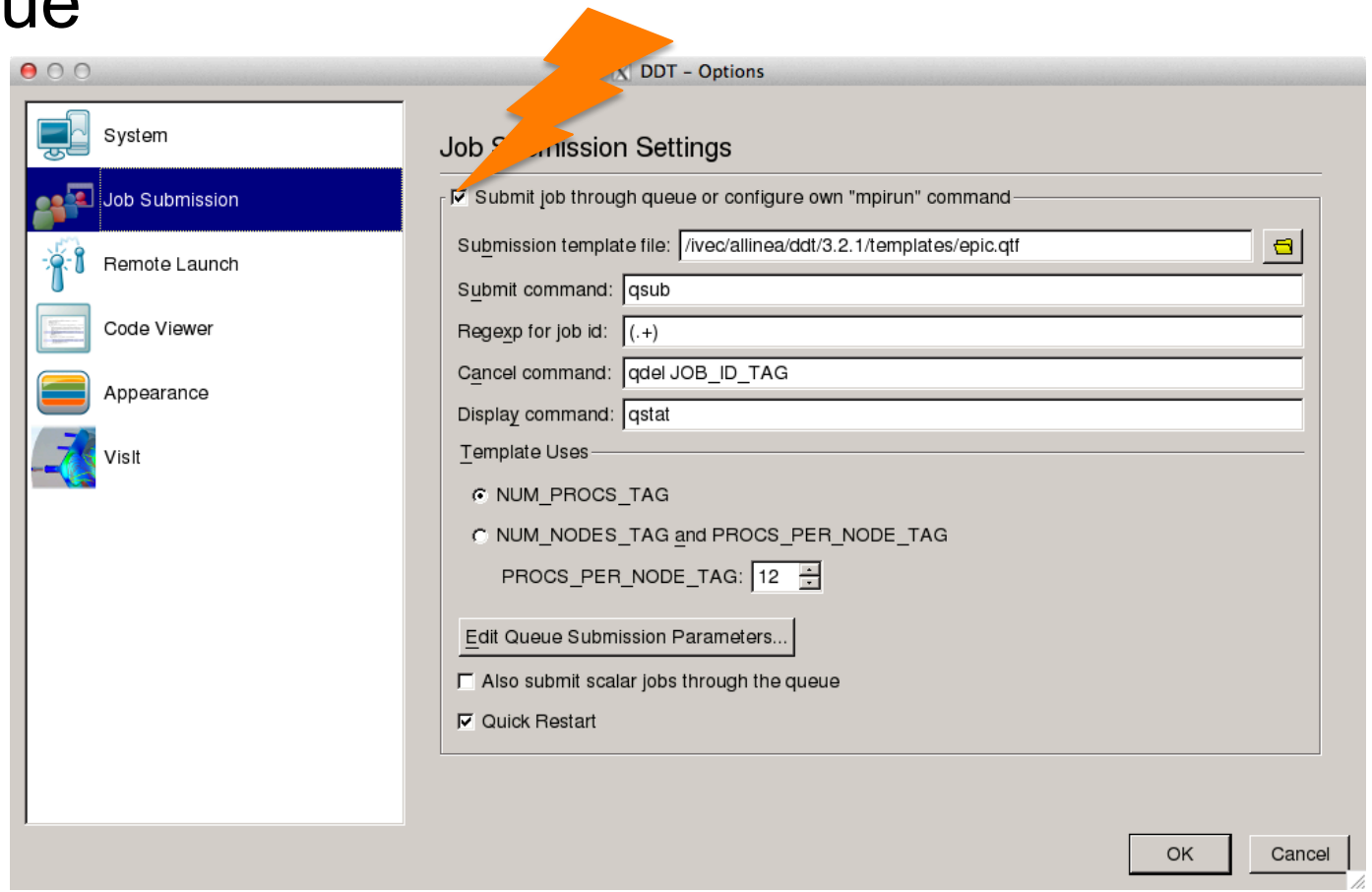


Running from Interactive Job

- `qsub -I -V -X -lwalltime=00:30:00 -W group_list=yourgroup -q debugq`
  - `-I` = interactive
  - `-V` = keep environment variables (useful if ddt module already loaded)
  - `-X` = allow X-forwarding
- Once job is running, invoke ddt: `ddt &`

Running from Interactive Job

- Make sure to untick “Submit job through queue”



Using DDT: Step 3 -- Debugging

- Set breakpoints
- Start/Pause/restart
- Look at variables
- Look at state of program on each processor
- Run program until condition occurs (i.e., stop when `x=6`)

Hands-On Demo

- Mystery!
  - Three darts codes fail – one segfaults, one hangs, and one gets the wrong answer
  - Can you figure out why?
- Instructions
  - `cp /group/courses01/debugging/mystery.tar.gz /scratch/courses01/`*`username`*
  - `tar -xvzf mystery.tar.gz`
  - `cd mystery`
  - `make`
- (Alternatively, work on your own code)