# Debugging and Profiling Parallel Programs on Titan

Fernanda Foertter

foertterfs@ornl.gov

# Outline

- About Allinea DDT
- DDT capabilities overview
- Using DDT
- GPU profiling


Oak Ridge National Laboratory

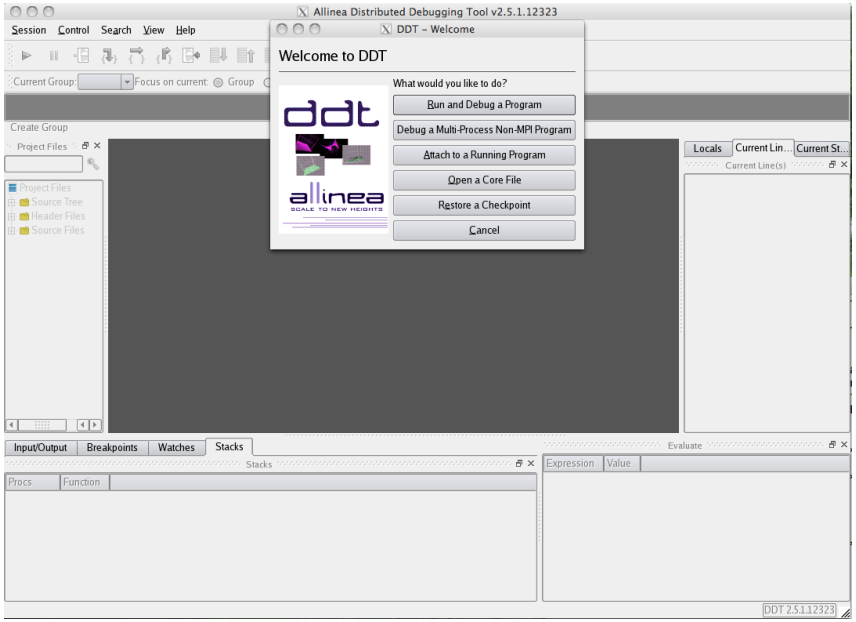
# About Allinea DDT

- Distributed Debugging Tool
- Capable of debugging codes written with MPI, OpenMP, threading, GPGPU
- Allinea collaborating with ORNL to create debugging tool
- Easy to use, intuitive

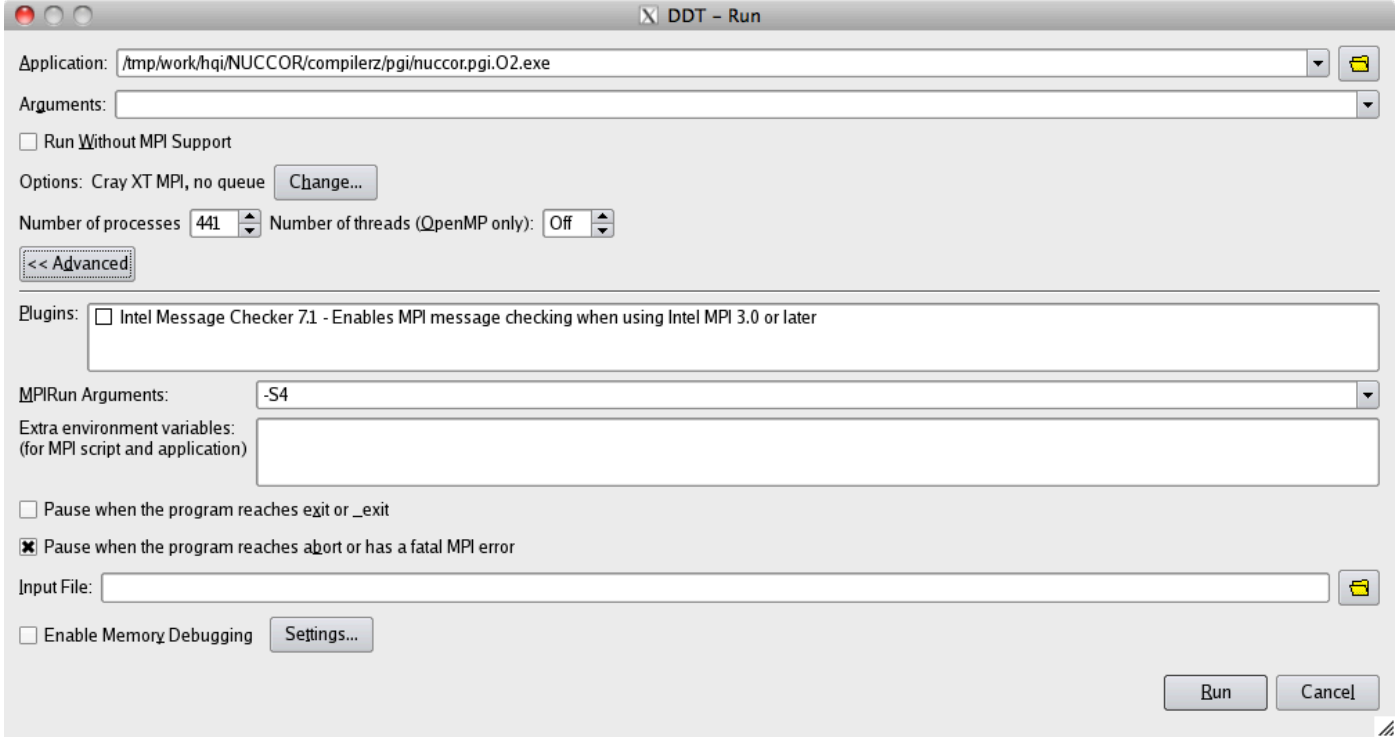

Oak Ridge National Laboratory

# DDT Capabilities Overview

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



# DDT Capabilities Overview

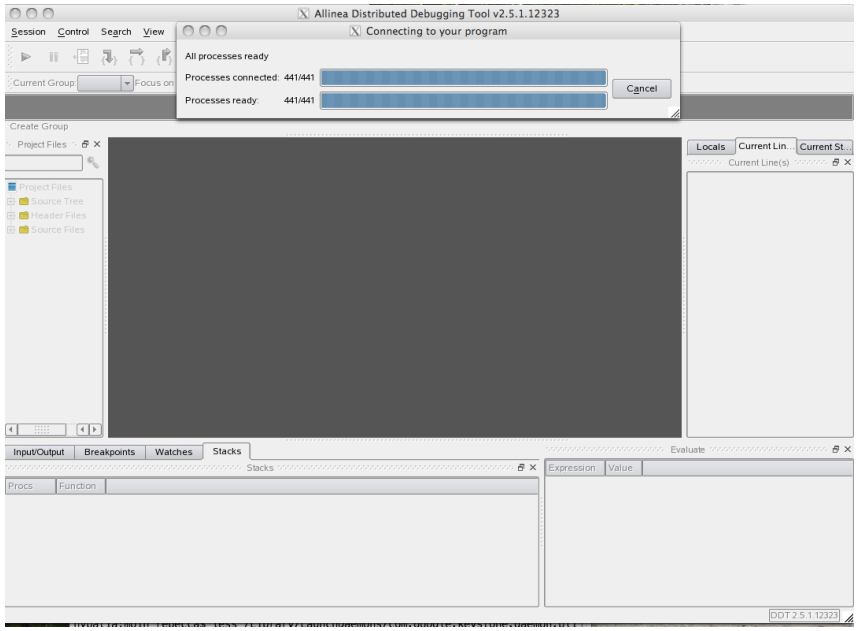



- Running a job
  - Enter application name
  - Can have DDT launch job, or run interactive job
  - Set arguments as necessary

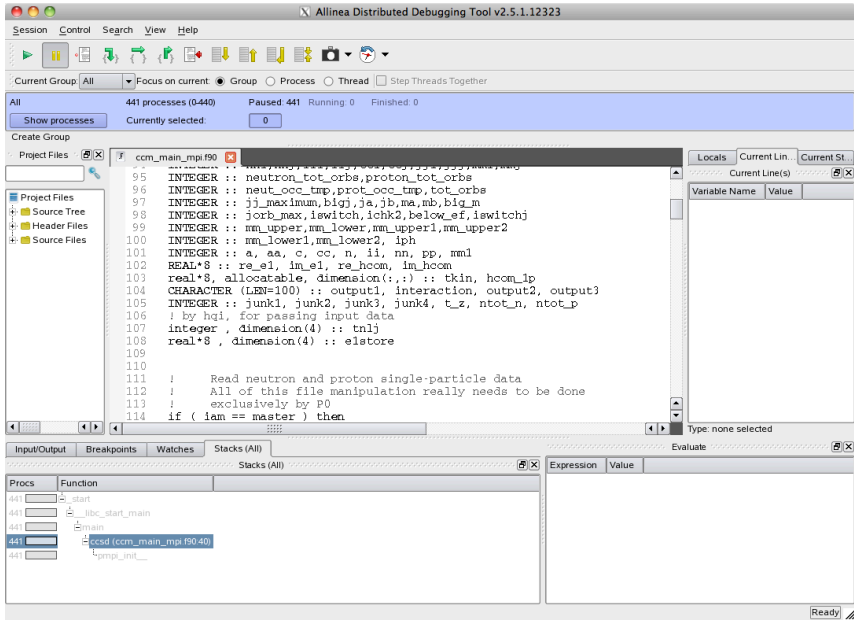

Oak Ridge National Laboratory

# DDT Capabilities Overview

## DDT Starting up

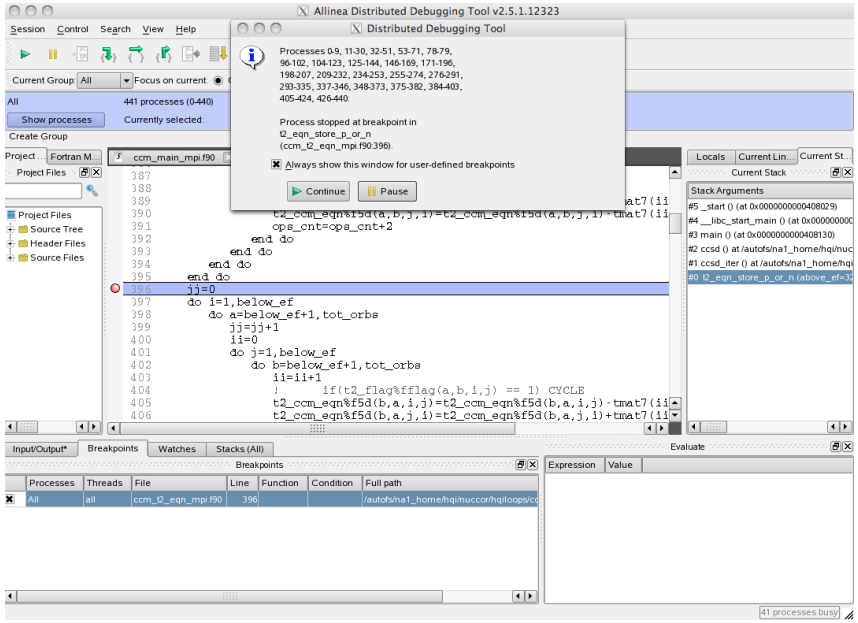



## Opening Screen

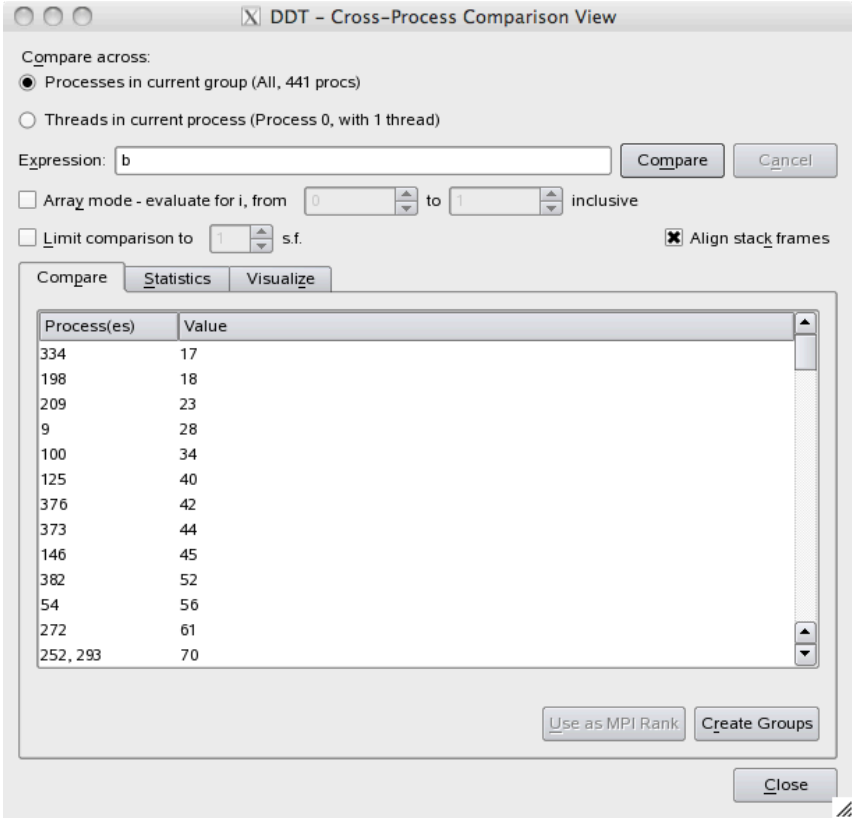


# DDT Capabilities Overview

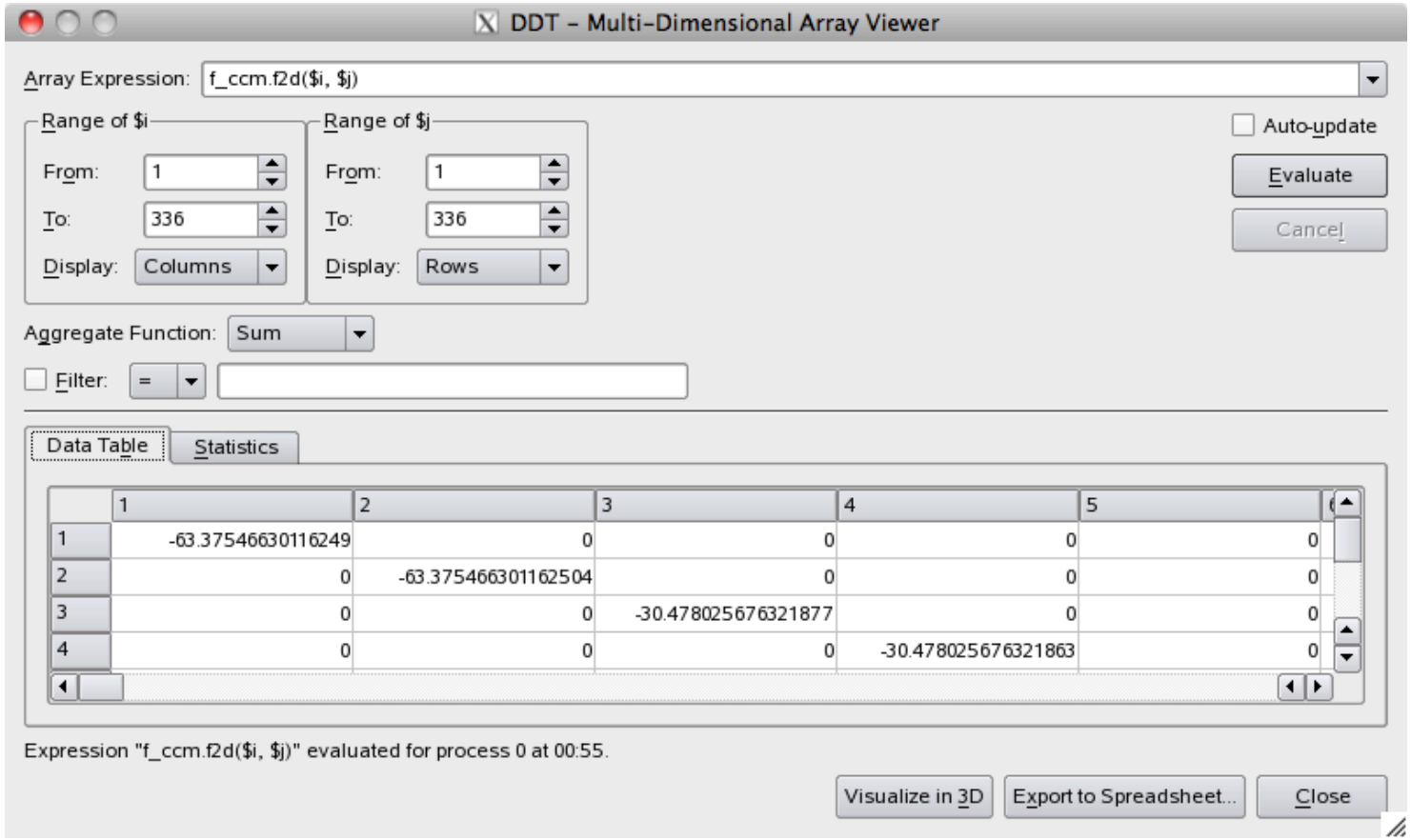
## Insert Breakpoints, and Pause at that Point




## Can view the value of scalar variable across all processes



# DDT Capabilities Overview




- View array on single process
- Can also view statistics, visualize, evaluate subsets

# Using DDT: Step 1 -- Compiling

**For regular debugging:**

- 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

**For memory debugging:**

- Compile with ddtmem* wrappers
  - `ddtmem-ftn`
  - `ddtmem-cc`
  - `ddtmem-CC`
- Must `module load ddt` before compiling, and rename compilers in makefiles, configure, etc.

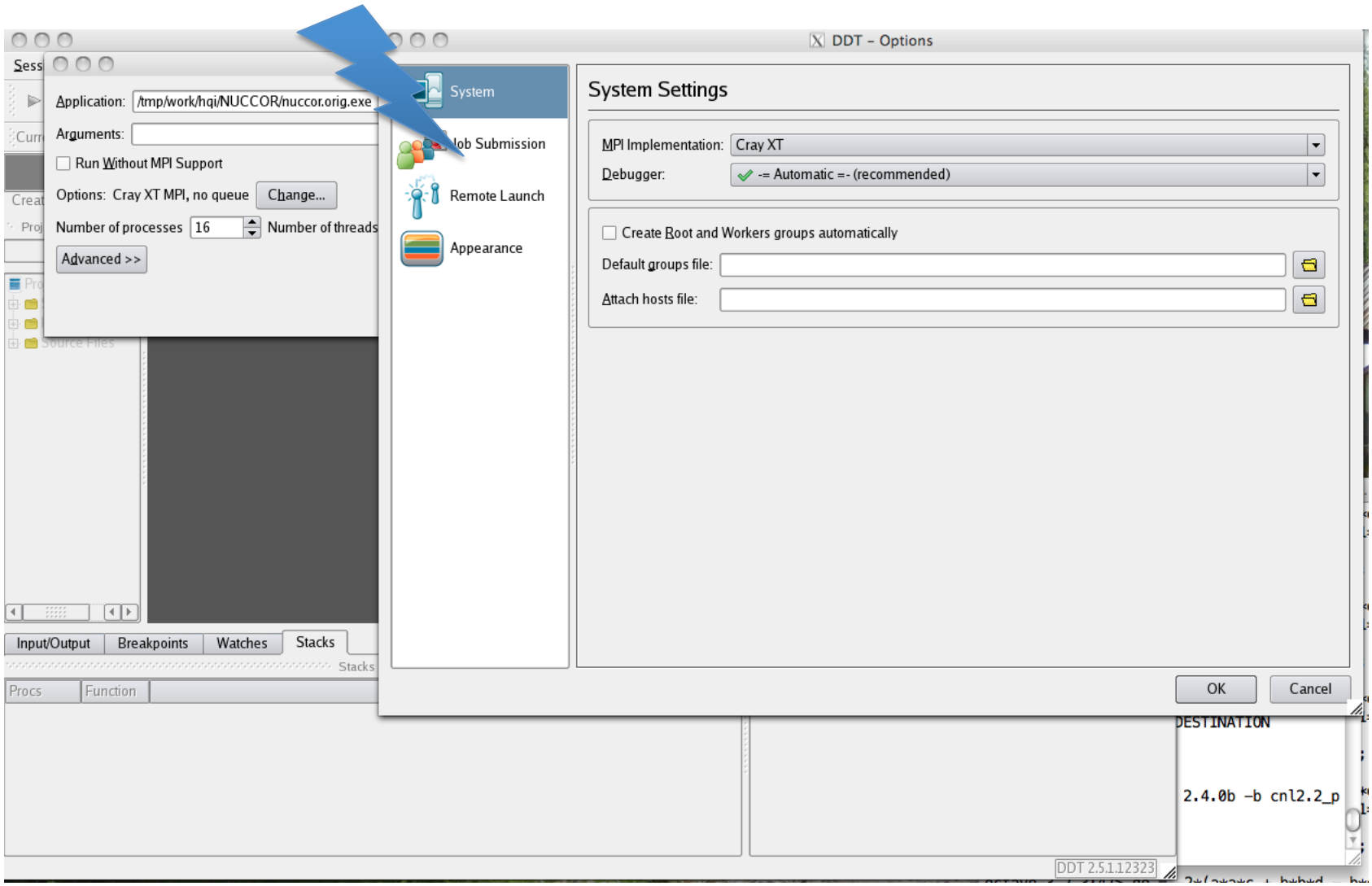

Oak Ridge National Laboratory

# Using DDT: Step 2 -- Running

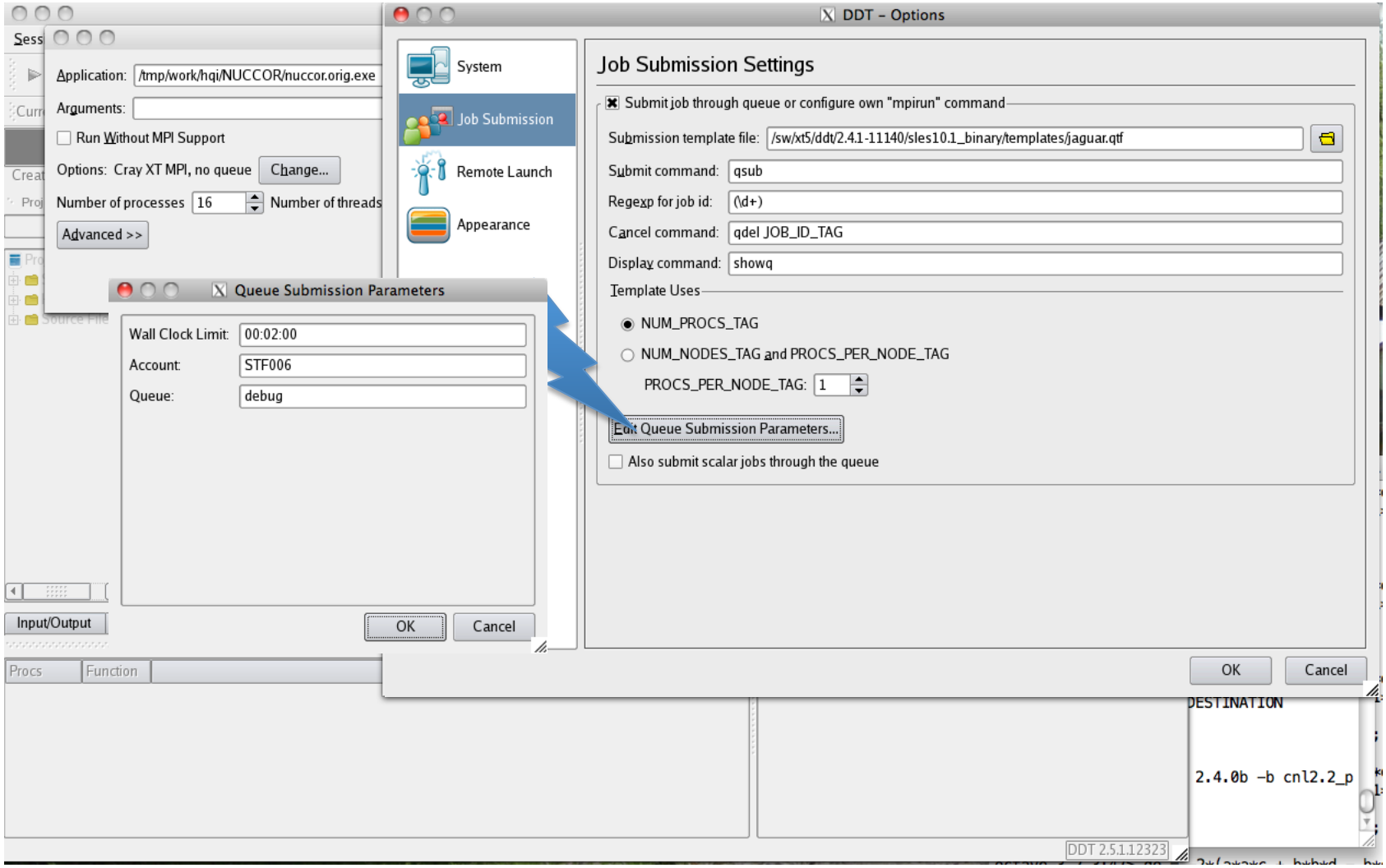
- You must have logged in with flags to allow X-forwarding:
  - `ssh -X` *`user`*`@jaguar{pf}.ccs.ornl.gov` (linux)
  - `ssh -Y` *`user`*`@jaguar{pf}.ccs.ornl.gov` (mac)
- `module load ddt`
- DDT can launch parallel interactive jobs for you
- Or, you can launch the interactive job and run DDT inside (I prefer this)


Oak Ridge National Laboratory

# Setting up Queue Parameters



# Setting up Queue Parameters

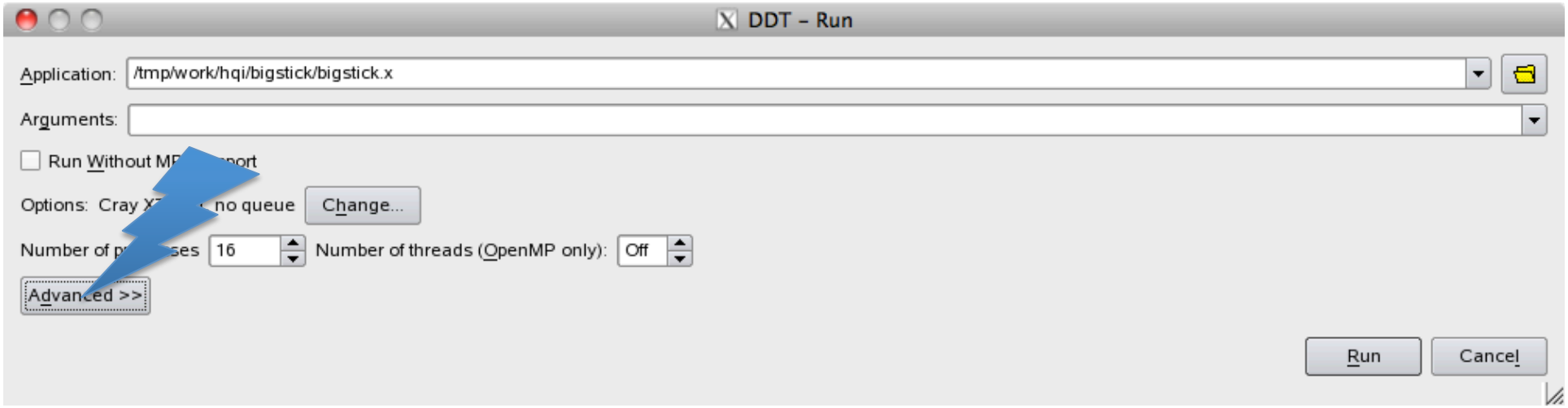





Oak Ridge National Laboratory

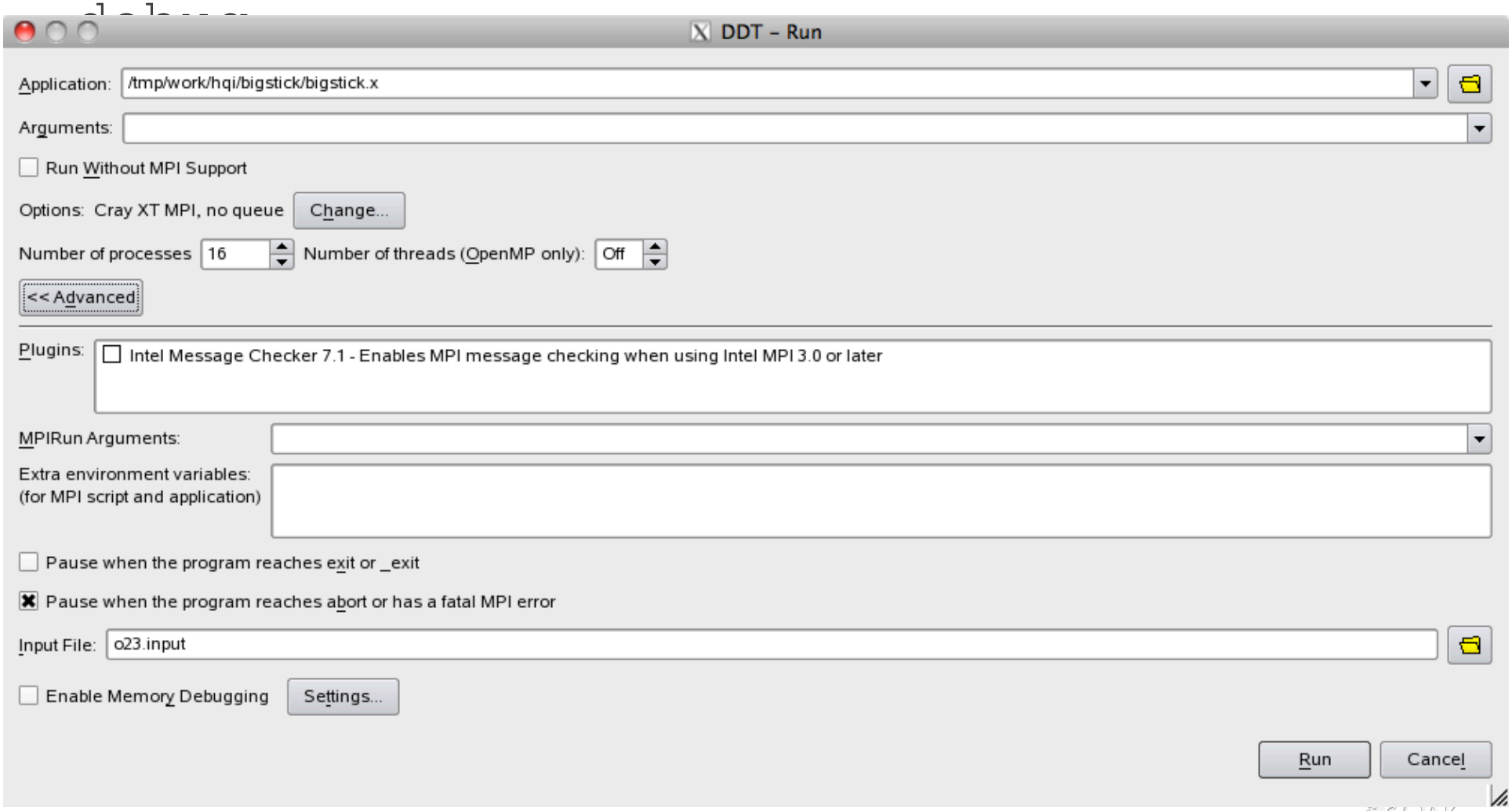
# Running from Interactive Job

- `qsub -I -A XYZ123 -V -lsize=12 -q debug`
- Once job is running, then do `ddt &`



# Running from Interactive Job

- `qsub -I -A XYZ123 -V -lsize=12 -q debug`



# 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)


Oak Ridge National Laboratory

# NVIDIA Compute Visual Profiler

shared - Compute Visual Profiler - [Session1 - Device_0 - Context_0 [CUDA]]

File Session View Options Window Help

Profiler Output | Summary Table

| | GPU Timestamp (us) | Method | GPU Time (us) | CPU Time (us) | grid size | thread block size | static shared memory per block (bytes) | registers per thread | Occu |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 1432 | GOL | 281.632 | 285.632 | [35 74 1] | [32 16 1] | 2048 | 14 | 1 |
| 8 | 1714 | ghostRows | 2.848 | 4 | [64 1 1] | [16 1 1] | 0 | 12 | 0.167 |
| 9 | 1720 | ghostCols | 4.288 | 8.288 | [65 1 1] | [16 1 1] | 0 | 8 | 0.167 |
| 10 | 1728 | GOL | 282.176 | 286.176 | [35 74 1] | [32 16 1] | 2048 | 14 | 1 |
| 11 | 2010 | ghostRows | 2.784 | 3 | [64 1 1] | [16 1 1] | 0 | 12 | 0.167 |
| 12 | 2016 | ghostCols | 4.32 | 7.32 | [65 1 1] | [16 1 1] | 0 | 8 | 0.167 |
| 13 | 2020 | GOL | 280 | 283 | [35 74 1] | [32 16 1] | 2048 | 14 | 1 |

Analysis

**Session1::Device_0::Context_0**

- Kernel time = 96.90 % of total GPU time
- Memory copy time = 0.8 % of total GPU time
- Kernel taking maximum time = **GOL** (94.5% of total GPU time)
- Memory copy taking maximum time = **memcpyDtoH** (0.5% of total GPU time)
- Total overlap time in GPU = 48.8 micro sec. (0.0% of total GPU time)

**Hint(s)**

- **Double click on the kernel name in the Summary Table to analyze the kernel**
- Analyze kernel **GOL**
- Consider using page-locked memory to attain higher bandwidth between host and device memory. Overuse of pinned memory should be avoided as it may reduce overall system performance.
  Refer to the "Page-Locked Host Memory" section in the "CUDA C Runtime" chapter of the CUDA C Programming Guide for more details.



Oak Ridge National Laboratory

# NVIDIA Compute Visual Profiler

- module load cudatoolkit
- cd /tmp/work/$USER $ export COMPUTE_PROFILE=1 COMPUTE_PROFILE_CSV=1
- aprun a.out
- nvvp &
- Visual IDE version: nvprof &


Oak Ridge National Laboratory

# NVIDIA Nsight

Profile - Timeline for session diverge - Nsight

File Edit Navigate Search Project Run Window Help

*diverge

s 0.25 s 0.5 s

Process: 21240
Thread: 950560576
Runtime API — cuda... cudaMemcpyAsync
Driver API
[0] GeForce GTX 280
Context 1 (CUDA)
MemCpy (HtoD)
MemCpy (DtoH)
Compute — Vec1of32x(int*, int*, int*, int)
63.5% [4] Vec1of... — Vec1of32x(int*, int*, int*, int)
10.6% [4] Vec50(i...
10.6% [4] Vec1of...
9.6% [4] VecThen(...
5.8% [4] Vec32of...
0.0% [4] VecEmpt...
Streams
Stream 1 — Vec1of32x(int*, int*, int*, int)

Properties | Detail Graphs

[0] GeForce GTX 280

| Name | Value |
|---|---|
| Duration | |
| Session | 705.345 ms |
| Compute Utilization | 89.6% |
| Compute/Memcpy | 4,337.261 |
| Overlap | |
| Kernel/Memcpy | 0% |
| Attributes | |
| Compute Capability | 1.3 |
| Maximums | |
| Multiprocessor | |
| Multiprocessors | 30 |
| Clock Rate | 1.296 GHz |
| Concurrent Kernel | false |
| Max IPC | 0 |
| Threads per Warp | 32 |
| Memory | |

Analysis | Details | Console

| Name | Start Time | Duration | Grid Size | Block Size | Regs | Static SMem | Dynamic SMem | Size | Throughput | gst 64B | gst 32B | gld 128B | g |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Memcpy HtoD [async] | 64.463 ms | 48.287 µs | n/a | n/a | n/a | n/a | n/a | 256 KB | 5.06 GB/s | n/a | n/a | n/a | |
| Memcpy HtoD [async] | 64.599 ms | 47.68 µs | n/a | n/a | n/a | n/a | n/a | 256 KB | 5.12 GB/s | n/a | n/a | n/a | |
| VecEmpty(void) | 64.677 ms | 1.728 µs | [1,1,1] | [1,1,1] | 0 | 0 | 0 | n/a | n/a | 0 | 0 | 0 | |
| VecThen(int*, int*, int*, int) | 64.721 ms | 6.015 ms | [1,1,1] | [1,1,1] | 11 | 44 | 0 | n/a | n/a | 0 | 2048 | 0 | |
| Vec50(int*, int*, int*, int) | 70.74 ms | 3.01 ms | [1,1,1] | [1,1,1] | 11 | 44 | 0 | n/a | n/a | 0 | 1025 | 0 | |
| Vec1of32(int*, int*, int*, int) | 73.754 ms | 3.01 ms | [1,1,1] | [1,1,1] | 11 | 44 | 0 | n/a | n/a | 0 | 1025 | 0 | |



Oak Ridge National Laboratory

# NVIDIA Nsight integration

C/C++ - bitreverse/src/bitreverse.cu - Nsight

File Edit Source Refactor Navigate Search Project Run Window Help

Project Exp

- bitreverse
  - Includes
  - src
    - bitreverse.cu
  - Debug
- diverge
- findmax

bitreverse.cu

```
 */
__global__ void bitreverse(void *data) {
    unsigned int *idata = (unsigned int*) data;
    idata[threadIdx.x] = bitreverse(idata[threadIdx.x]);
}

/**
 * Host function that prepares data array and passes it to the CUDA kernel.
 */
int main(void) {
    void *d = NULL;
    int i;
    unsigned int idata[WORK_SIZE], odata[WORK_SIZE];

    for (i = 0; i < WORK_SIZE; i++)
        idata[i] = (unsigned int) i;

    CUDA_CHECK_RETURN(cudaMalloc((void**) &d, sizeof(int) * WORK_SIZE));
    CUDA_CHECK_RETURN(
            cudaMemcpy(d, idata, sizeof(int) * WORK_SIZE, cudaMemcpyHostToDevic

    bitreverse<<<1, WORK_SIZE, WORK_SIZE * sizeof(int)>>>(d);
```

Outl | Mak

- stdio.h
- stdlib.h
- WORK_SIZE : const int
- CUDA_CHECK_RETURN()
- bitreverse(unsigned int) :
- bitreverse(void*) : void
- main(void) : int

Problems | Tasks | Console

CDT Build Console [bitreverse]

```
make all
Building file: ../src/bitreverse.cu
Invoking: NVCC Compiler
nvcc -G -g -O0 -gencode arch=compute_13,code=sm_13 -gencode
arch=compute_20,code=sm_20 -gencode arch=compute_20,code=sm_21 -odir "src" -M -o
"src/bitreverse.d" "../src/bitreverse.cu"
nvcc --compile -G -O0 -g -gencode arch=compute_13,code=compute_13 -gencode
arch=compute_13,code=sm_13 -gencode arch=compute_20,code=compute_20 -gencode
arch=compute_20,code=sm_21 -o  "src/bitreverse.o" "../src/bitreverse.cu"
Finished building: ../src/bitreverse.cu
```

Writable | Smart Insert | 2 : 1



Oak Ridge National Laboratory

# CrayPat

- Supports: MPI, CUDA, Cray SHMEMTM, UPC, Co-Array Fortran, OpenMP, OpenACC, C, C++ and Fortran
- module load perftools
- pat_build -O apa my_program
- Produces **my_program+pat** executable
- Submit your job with new exe
- Generate report with:

  -pat_report -T -o report1.txt my_program+pat+PID-nodesdt.xf


Oak Ridge National Laboratory

# Questions?

- Support Overview
- Getting Started
- System User Guides
- KnowledgeBase
- Tutorials
- Training Events
- My OLCF
- **Software**
- Documents & Webforms
- Known Issues
- OLCF Policies

Home › User Support › Software › Debugging and Profiling

## Debugging and Profiling

Programs to aid in the correction and optimization of source code.

### craypat

*Cray Performance Analysis Tools (CrayPAT) can be used to evaluate program execution performance on Cray systems.*

### ddt

*Allinea DDT is an advanced debugging tool used for scalar, multi-threaded, and large-scale parallel applications.*

### dyninst

*Dynamic Instrumentation (Dyninst) is an API for Runtime Binary Code Manipulation.*

### fpmpi

*FPMPI and FPMPI_papi are light-weight profiling libraries that use the pmpi hooks, as specified in the MPI standard.*

### gdb

*The GNU Project Debugger (GDB) lets you debug programs written in Ada, C, C++, Objective-C, Pascal (and many other languages).*

### gptl

*The General Purpose Timing Library (GPTL) is a library to instrument C, C++, and Fortran codes for performance analysis and profiling.*

### hpctoolkit



Oak Ridge National Laboratory

# Hands-On Demo

- Mystery!
  - A code segfaults under certain conditions, but I don't know why
  - Can you solve the mystery?
- Instructions
  - `cp /tmp/work/hqi/mystery.tar.gz /tmp/work/`*`username`*
  - `tar -xvzf mystery.tar.gz`
  - `cd mystery; less README`
- (Alternatively, work on your own code)


Oak Ridge National Laboratory