



# Getting Started with Debugging GPUs at ORNL

David Lecomber  
david@allinea.com  
CTO

- A revolution is in progress in computing and at ORNL
  - Multi-core, many-core, GPUs
  - Hybrid code is here! MPI + OpenMP + CUDA
- *“Software has become the #1 roadblock ... Many applications will need a major redesign”*

IDC HPC Update, June 2010

  - Most ISV codes do not scale
  - High programming costs are delaying GPU usage

- Software tools company since 2001
  - Allinea DDT – the scalable parallel debugger
  - Allinea OPT – the optimization tool for MPI and non-MPI
  - Users at all scales – at 1 to 100,000 cores and above
  - World's only Petascale debugger!

*Simplifying the challenge of multi- and many-core development*

- Bugs at **scale** need a debugger at **scale**
  - ... until recently debuggers limited to ~4,000-8,000 cores
- Bugs **on GPUs** need a debugger **for GPUs**
  - ... until recently GPU software couldn't be debugged

## Aviation and Defence

NORTHROP GRUMMAN



## Climate and Weather



## Energy



## Electronic Design Automation



## Academic

Over 200 universities





Partnership to develop Petascale debugger with NVIDIA support

University of California



Partnership to develop Petascale/ Exascale tools and standards



Partnership on Full Scale debugging on IBM Blue Gene /P & /Q



**Allinea DDT** is “*Debugger of Choice*” on NERSC 5 and NERSC 6 and first implementation on CRAY XE6

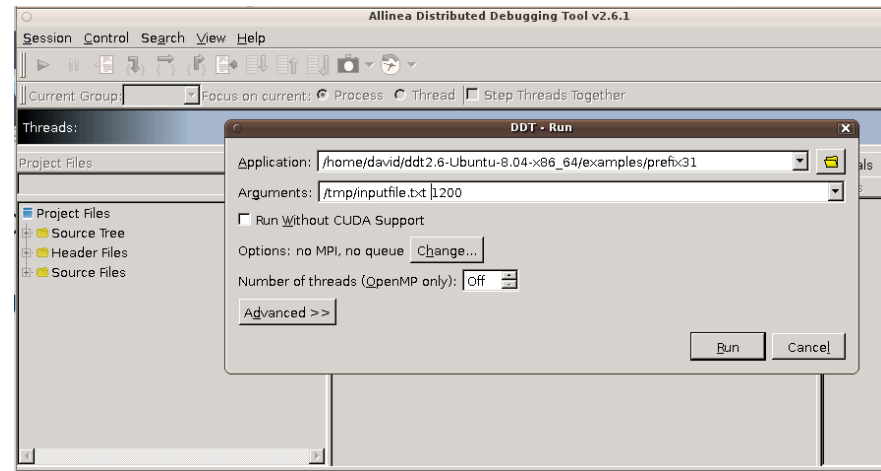


Partnership with CEA French Atomic Energy Authority on scalable programming and CUDA

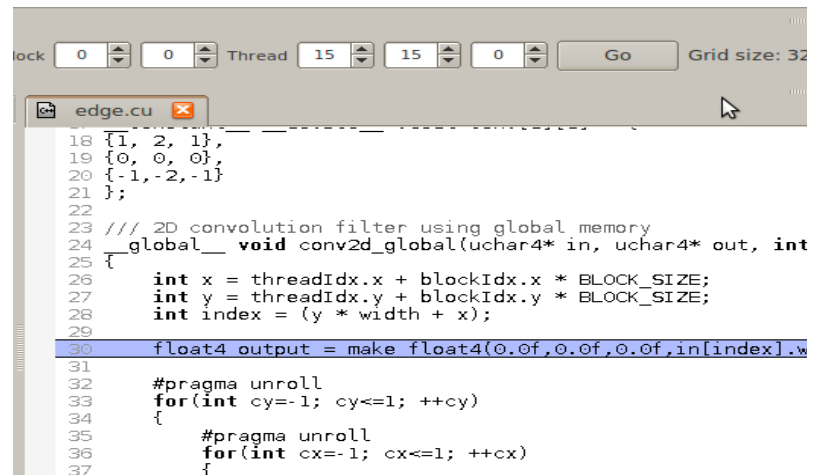


Partnership on Keeneland project to help solving software challenges introduced by mixed architectures

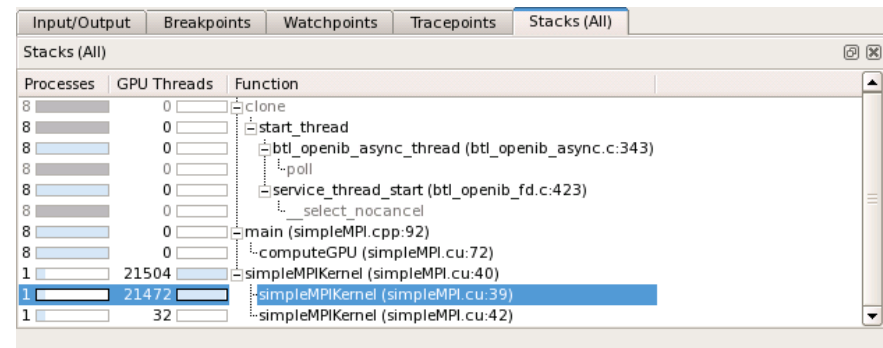
- As easy as debugging ordinary code
  - Compile your application  
`nvcc -g -G prog.cu -o prog`
  - Choose other settings
    - Memory debugging with CUDA memcheck
- Debugs CPU and GPU simultaneously
  - View source, examine variables, control processes and threads

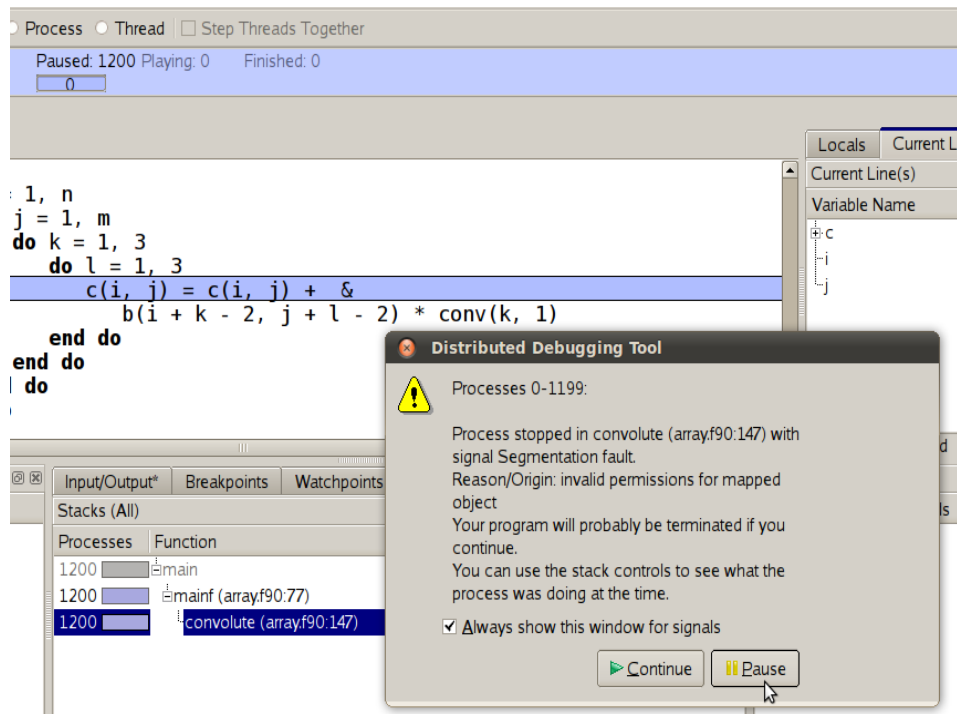


- Set breakpoints
  - Automatically stop on kernel launch
  - Stop at a line of CUDA code
  - Hover the mouse for more information
- Step a warp, block or kernel
  - Follow the logic of individual threads through the kernel
- ... or just run through to a crash



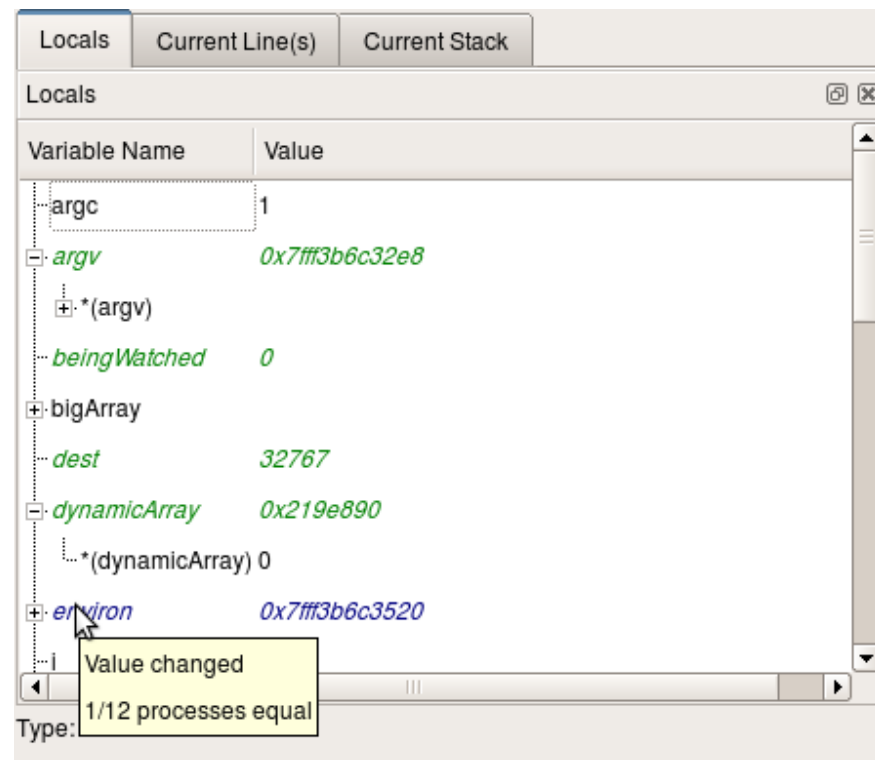
- View all threads in parallel stack view
  - At one glance, see all GPU and CPU threads together
  - Links with thread selection
  - Pick a tree node to select one of the CUDA threads at that location
- Memory debugging
  - CUDA Memcheck detects read/write errors
- Full MPI support
  - See GPU and CPU threads from multiple nodes



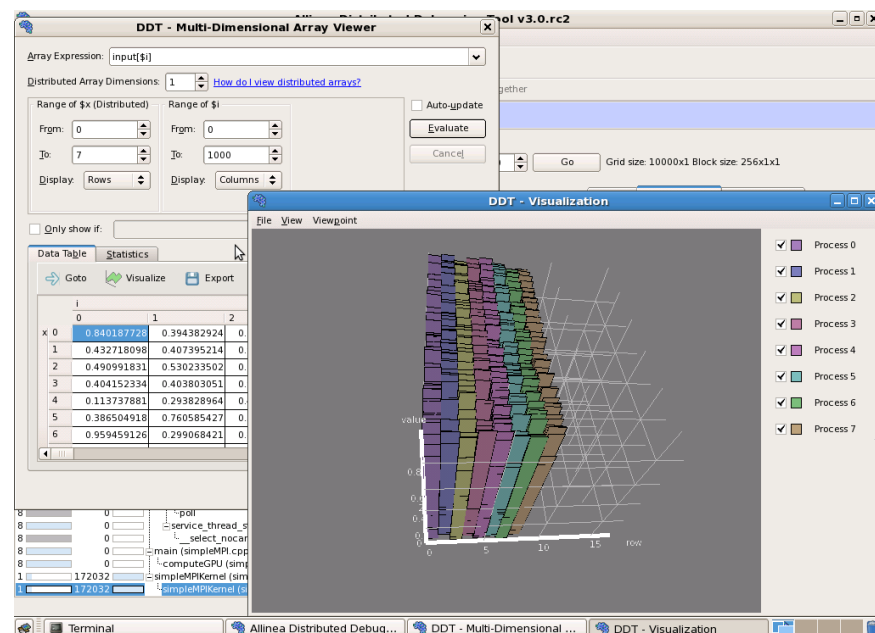


- Immediate stop on crash
  - Segmentation fault, or other memory problems
  - Abort, exit, error handlers
  - CUDA errors
- Scalable handling of error messages
- Leaps to the problem
  - Source code highlighted
  - Affected processes shown
  - Process stacks displayed clearly in parallel

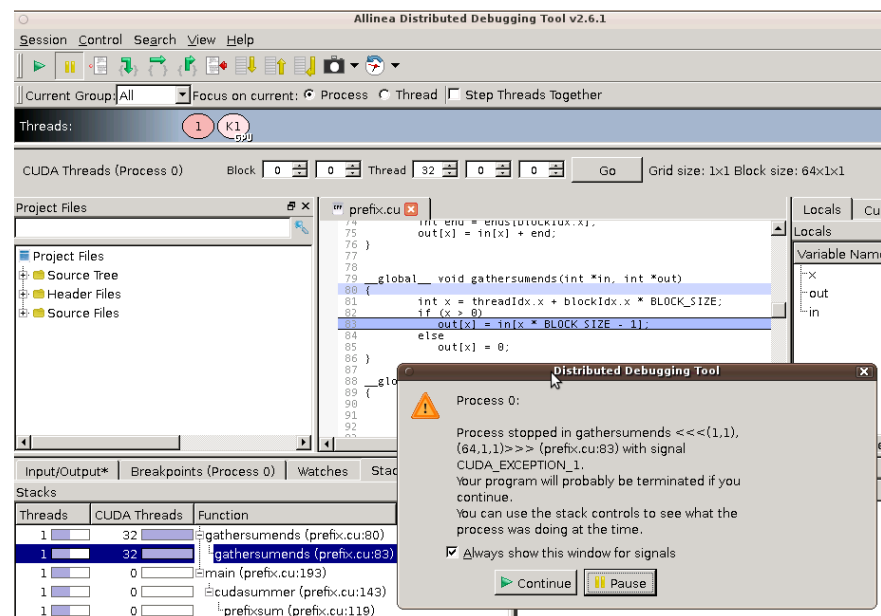
- Full data browsing
  - Local and current line(s) variables
    - Show variables relevant to current position
    - Drag in the source code for more
    - C, C++, F90
    - Shows memory type for CUDA
      - register, shared, ...
- Smart Highlighting
  - Scalable and fast automatic comparison and change detection



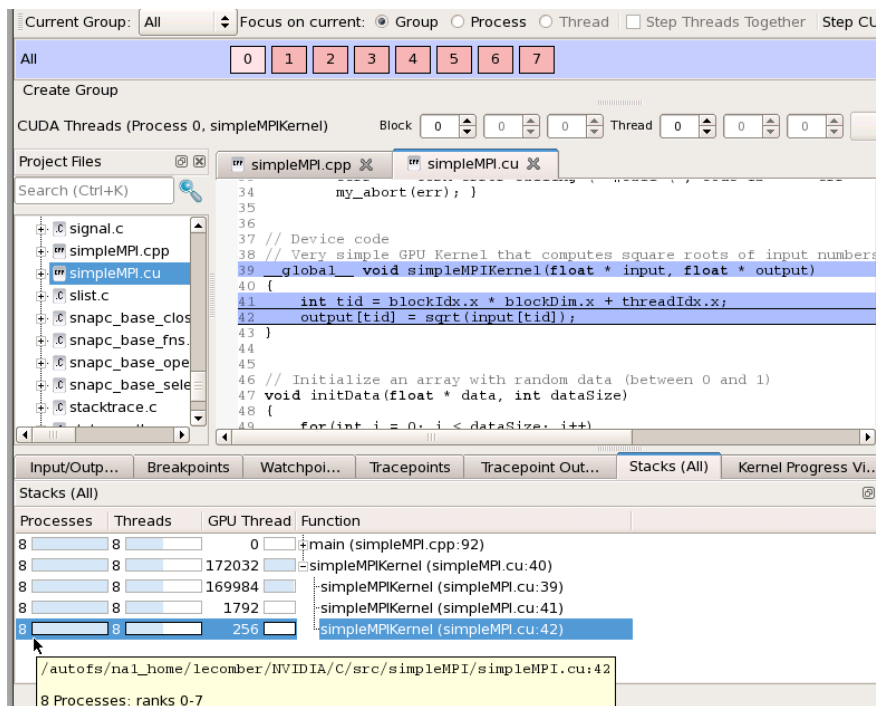
- Grid/block size problems
  - Incorrect dimensions lead to incomplete results
- Easy to see with DDT
  - Use the multi-dimensional array viewer to look at data and find the rough edges
  - 3D display and filtering support
  - **New:** displays data from multiple processes



- Grid/block size problems
  - Sometimes lead to worse consequences: over-writing good values, or crashing
  - Bugs will often trigger “CUDA memcheck” errors
  - A mode of execution similar to DDT's beyond bound checks for CPU programming

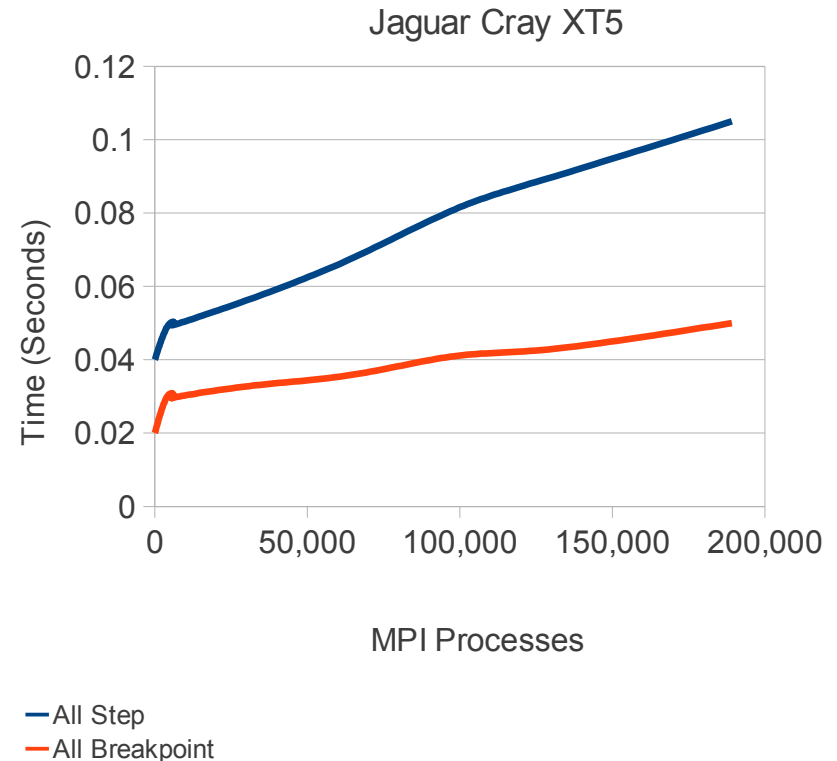


- DDT handles mixed MPI/CUDA code easily
  - Available on ORNL's Yona testbed system
  - Some early limitations due to be fixed by NVIDIA soon –
    - eg. multiple processes per node each with GPU...



- Petascale CPU debugging
  - Tree architecture - logarithmic performance
- Ongoing collaboration
  - With ORNL currently extending DDT to Petascale GPU system - Titan
    - New languages
    - New features
    - Scalability

DDT 3.0 Performance Figures



Proven debugging all the way to Petascale

XT/XE/XK fit Allinea DDT's scalable architecture, perfectly  
220,000 cores at ORNL

Many thousand cores elsewhere on Cray systems - NERSC, HLRS,  
NOAA, ...

*Scalable software needs scalable debugging*

Debugging support for Cray compiler environment

- Fast track debugging
  - Runtime switch to debug non-optimized code with Cray compiler
- PGAS languages joint project completed and in the field
  - Co-Array Fortran and UPC
- Cray GPU compiler - support in progress 2011

Integrates with Cray ATP (Abnormal Termination Process)

- Attach to failing XT/XE applications before they are terminated

GPU systems are delivering beyond Petascale

Cray XK6 – at Petascale for ORNL – and other machines

In progress at Allinea

Petascale debugging to support GPU hybrid systems

- Process control and GUI already in place
- New features to make finding kernel progress easier

More optimization

- Support for “Ugly Sets” - optimizing some of the pathological cases that can degrade performance at scale

Scaling tracepoints to Petascale and off-line debugging

GPU development requires different compilers!

A number of options – but all have a way of debugging

## NVIDIA CUDA C/C++

- Low level – but very well supported for debugging

... and other options we explore now

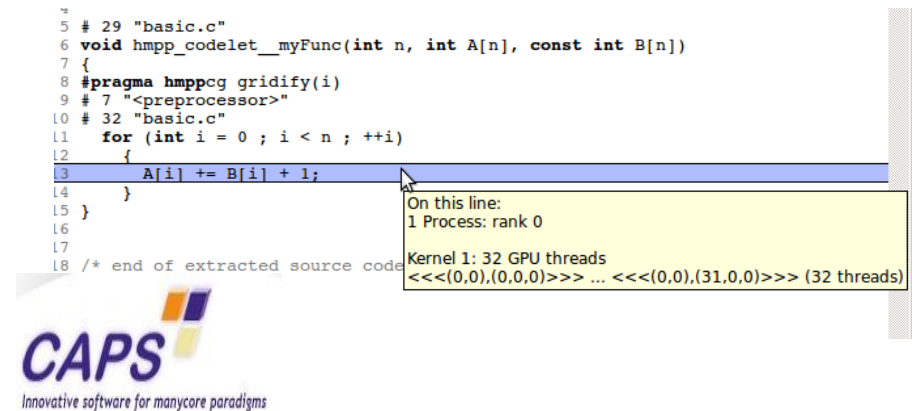
- CAPS HMPP – C/C++/Fortran directives
- Cray OpenMP Accelerator Directives
- Portland Group – accelerators and CUDA Fortran

Debuggable inside C kernels  
(codelets) on the GPU

F90 multi-dimensional arrays  
support in progress for the  
GPU

Auto-reporting of CAPS  
runtime errors to the DDT GUI

Also able to debug codelets  
running on the CPU





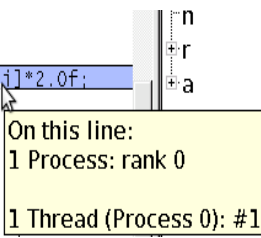
```
1 program main
2 integer, parameter :: n = 1000
3 real, dimension(n) :: input
4 real, dimension(n) :: result
5 integer :: i
6 do i = 1,n
7 input(i) = i*4.0
8 enddo
9 !$omp acc_region
10 !$omp acc_loop
11 do i = 1,n
12 result(i) = input(i) * 4.0
13 enddo
14 !$omp
15 !$omp Kernel 1: 32 GPU threads
16 print<<<(0,0),(0,0,0)>>> ... <<<(0,0),(0,0,0)>>> (32 threads)
17 end program
```

- OpenMP accelerator directives
- Debuggable – run accelerated code on the CPU by setting “-O0”
- Debugging of the GPU itself: work in progress
  - Some F90 examples debuggable
  - Use “-g -Gomp” compiler flags to debug on the GPU

# allinea DDT for Portland compilers

PGI\*

```
#pragma acc region
{
  for( i = 0; i < n; ++i ) r[i] = a[i]*2.0f;
}
/* compute on the host to compare */
for( i = 0; i < n; ++i ) e[i] = a[i]
/* check the results */
for( i = 0; i < n; ++i )
  assert( r[i] == e[i] );
```



On this line:  
1 Process: rank 0  
1 Thread (Process 0): #1

## PGI Accelerator Model

Debug on CPU only – use “-ta=host” to set this

Runs as a single thread on the CPU

Debugging possible – but race conditions wouldn't be seen

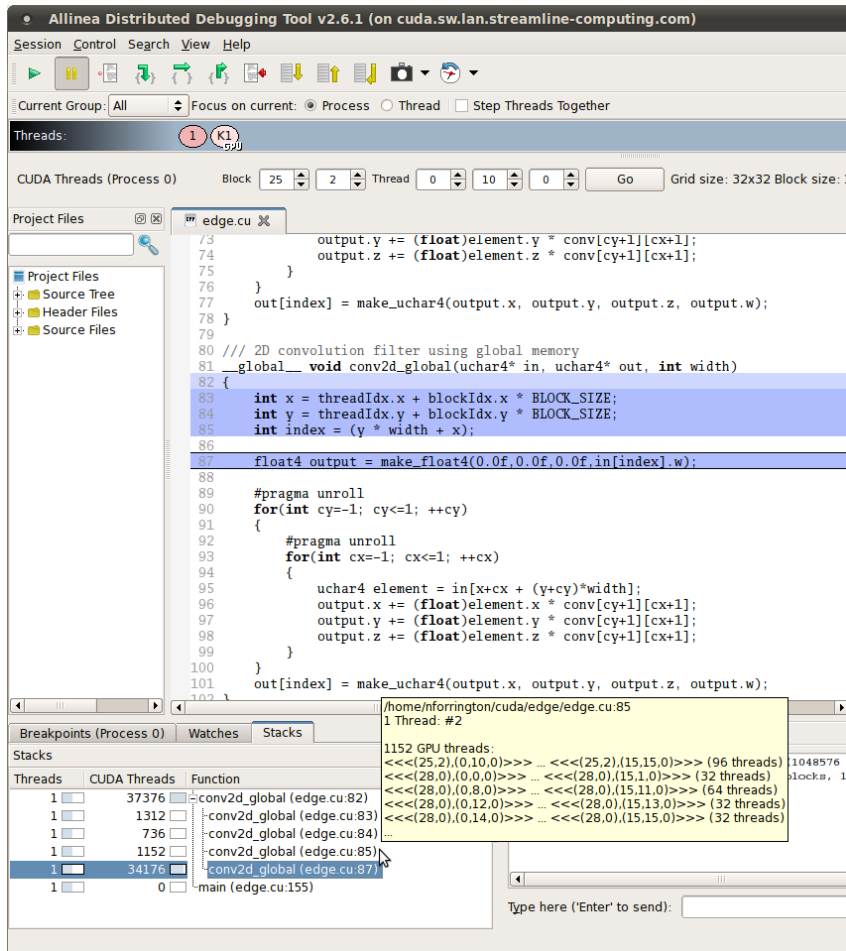
## PGI CUDA Fortran

Debug on CPU only – use “-Mcuda=emu” flag

Runs multithreaded on the CPU if GPU disabled

Works well with DDT

Easy to see missing syncthreads, for example



- Toolkit 3.1 and 3.2
  - Available now in DDT
    - Multi-device support
    - Fermi and Tesla support
    - CUDA Memcheck support for memory errors
    - MPI and CUDA support for GPU clusters
    - Breakpoints, thread control, and data evaluation
    - Stop on kernel launch
- Toolkit 4.0 in Beta
  - C++ kernel debugging

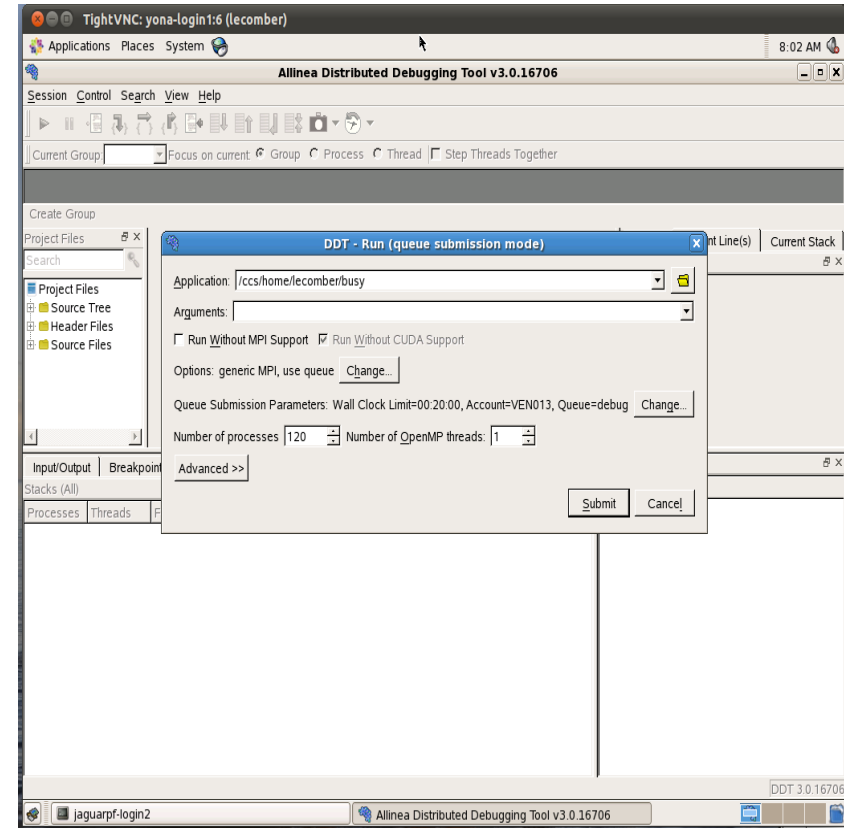
# allinea Using DDT on Jaguar/Titan

Titan will be identical to Jaguar

- module load ddt
- ddt *program-name*

Configured ready to launch a job in the queue

- Remember to compile with “-g” options
- Choose number of processes in DDT and click submit!



- Resources
  - NVIDIA CUDA Zone
    - [http://www.nvidia.com/object/cuda\\_home\\_new.html](http://www.nvidia.com/object/cuda_home_new.html)
  - Evaluation copies of DDT for CUDA and demo video
    - <http://www.allinea.com/cuda>
  - Debugging for CUDA white paper – includes a worked example
    - <http://www.allinea.com>