Experiences with the Distributed Debugging Tool

Frank Willmore

willmore@tacc.utexas.edu

Teragrid 2010
Introduction

• The need for a parallel debugging tool
• Configuration
• Process control
• Viewing and editing data
• CUDA support
• Statistics on Usage and Training on DDT at TACC
Debugging Parallel Programs

• Usual problems
  – Memory access issues
  – Special cases not accounted for in the code
  – Wrong arguments to functions
  – ....

• New set of problems
  – Work distribution and partitioning
  – Arguments to parallel function calls
  – Race conditions (shared memory / OMP)
  – Deadlocks (MPI)
  – Indeterminate order of execution
  – ...
About DDT

Allinea Distributed Debugger Tool

• Multiplatform

• Supports all MPI distributions

• Capable of debugging large scale OMP/MPI

• Comprehensive
  – Memory checking
  – MPI message tracking

• Useful Graphical User Interface
setting up a debug session

CONFIGURATION
Using DDT

• Compile your code using the standard debug flags:
  
  % mpicc  -g -O0 ./srcFile.c  
  % mpif90 -g -O0 ./srcFile.f90

• Load the DDT module:
  
  % module load ddt  
  % module list

• Start up DDT:
  
  % ddt ./a.out
Configuration: Welcome Screen

• Three ways of starting a debug session
  – Run and debug
  – Attach to a running program
  – Open core dump file
Configuration: Job Submission

![DDT - Run (queue submission mode)](image)

- Application: /share/home/01157/carlos/debug_examples/parallel/a.out
- Arguments: 
- Run Without MPI Support
- Options: mumpich 1 MPI, use queue
- Queue Submission Parameters: Queue=development, Wall Clock Limit=0:30:00, Project=
- Number of processes: 16
- Number of threads (OpenMP only): 1

Submit | Cancel
Configuration: Options

- Choose the correct version of MPI
  - mvapich 1
  - mvapich 2
  - openMPI

- Leave Debugger on the Automatic setting
Configuration: Queue Parameters

- Choose the queue
- Set the Wall Clock Limit (H:MM:SS)
- Set your project code

![Queue Submission Parameters](image)
Configuration: Memory Checks

- Open the Advanced tab.
- Enable Memory Debugging (bottom left check box)
- Open the Memory Debug Settings
Configuration: Memory Options

- Change the Heap Debugging option from the default **Runtime** to **Low**
- Even the option None provides some memory checking
- Leave Heap and Advanced unchecked unless you really know that you need them...
Memory Checks and Running Time

- The higher the level of memory checking the longer the execution time
- Runtime overhead is NOT LINEAR with the level of memory checking
- Often a low level of memory checking is sufficient to debug non-pathological issues
Job Queuing

Add any necessary arguments to the program. Click the Submit button. A new window will open:

The job is submitted to the specified queue.

An automatically refreshing job status window appears.

The debug session will begin when the job starts.
DDT: The debug session

Right click on a line to set a breakpoint.

Most operations available as buttons in the process controls bar.
focus, breakpoints and watches

PROCESS CONTROL
Stack Window

Local variables for the currently selected line and stack information is visible here.
Parallel Stack View

- Shows position of each thread/process in the code.
- Hover over any function to see a list of processes at that position.
Creating Process Groups

• Ctr-Click to select
• Control->Create Group

• Group is created from selection with default name
• Right-Click the group to edit properties
Changing Focus

- Control->Focus
- Select
  - Group
  - Process
  - Thread
- Or direct selection button

- Process control window changes to reflect selection
Inserting Breakpoints

- Double-click on any line
- Right-click on any line
- Control -> Add Breakpoint
- Breakpoint icon
Conditional Breakpoints

- Right-Click on any breakpoint in the breakpoint window
- Edit Breakpoint
- Write condition in appropriate language
Conditional Breakpoints

• Change will be reflected in breakpoint window
Inserting Watches

- Right-Click in Watch Window
- Right-Click on Local/Evaluation Windows
Watch Point Activation

- When a watch point is triggered DDT will pause execution and issue a warning

- This gives you the opportunity to analyze the current execution state and the values of your variables
multidimensional array viewer, memory stats, parallel message queue

VIEWING AND EDITING DATA
Viewing and Editing Variables

- Local variables appear in the variable window
- Drag & drop variables from Variable window to Evaluation window
- Right-click the variable name in the Evaluation window and select “Edit Value” or “Edit Type”
- View as vector to see dynamically allocated values
Data Values Across Processes

- View->Cross-Process Comparison
- View variable values across all processes in a group

- View->Cross-Thread Comparison
- View variable values across all threads in a process
Data Values Across Processes (2)

Value comparison

Statistical Info

Comparison of a single value

Visual comparison
Array data access
Multi-Dimensional Array Viewer

One of these for each dimension
Data Visualization

Change by selecting task in process control window
Analyzing Memory Usage

Current Memory Usage

Overall Memory Stats
Current memory Usage

Go to View -> Current Memory Usage

Process 0 is using much more memory than the others.

This looks like a memory leak.
Overall Memory Stats (Size)

Go to View -> Overall Memory Stats
Overall Memory Stats (Calls)
Segmentation Faults

- DDT tells you the exact line in the code where the problem occurs.
- DDT tells you the precise reason that caused the problem.
Message Queues

Go to View -> Message Queues

Click on bottom left for additional information (message sizes, etc...)

Pending messages in the “Unexpected” queue are indicative of MPI problems
DDT 2.6: CUDA-enabled
Known issues

• **14.6 Known Issues**
• Only one GPU per physical host (O/S) can be debugged at any one time. This is a limitation of NVIDIA SDK 3.0. Be sure to only debug one GPU per host, by, for example, judicious use of mpirun process-per-node arguments.
• Attaching to an already running CUDA job is not currently supported.
• You must compile with "-g -G" to enable GPU debugging - if you do not your program will run through the contents of kernels without stopping.
• X11 cannot be running on the GPU that is used for debugging.
• The debugger enforces blocking kernel launches.
• It is not yet possible to spot unsuccessful kernel launches or failures. An error code is provided by getCudaLastError() in the SDK which you can call in your code to detect this. Currently the debugger cannot check this without resetting it, which is not desirable behaviour.
• Device memory allocated via cudaMalloc() is not visible outside of the kernel function.
• Host memory allocated with cudaMallocHost() is not visible to the debugger.
• Not all illegal program behaviour can be caught in the debugger - e.g. divide-by-zero.
• It is not possible to step over a subroutine in the device code.
• Multi-threaded applications may not work.
• Device allocations larger than 100 MB on Tesla GPUs, and larger than 32 MB on Fermi GPUs, may not be accessible.
• Breakpoints in divergent code may not behave as expected.
DDT Usage and Training at TACC

• 494 TACC users trained in DDT since 2009
  – 189 users between January and July 2010 (15 remote)
  – 305 users between January and December 2009

• DDT availability in all TACC systems
  – reduces training time
  – Module systems helps with environmental settings
  – One session covers all.
DDT Usage and Training at TACC

- Large number of licenses allows interactive labs
- DDT perceived as easy to use once a configuration example is given
  - Intuitive Graphical User Interface
  - Few questions regarding use during labs
  - Users feel confident debugging their own codes immediately after the training
- Users run successful debugging sessions with their own codes during lab time!