Debugging with TotalView on the Blue Gene Q
Early Blue Gene Days
TotalView Blue Gene Support

- TotalView involvement started in 2003 on BG/L

Gotta love that yellow duct tape!
TotalView Blue Gene/L Support

- Support for Blue Gene/L since 2005
- Debugging interfaces developed via close collaboration with IBM (CIOD)
- Used on DOE/NNSA/LLNL's Blue Gene/L system containing 212 K cores
  - Heap memory debugging support added
  - Blue Gene/L scaling and performance tuning project

Blue Gene/L work facilitated Blue Gene/P support
TotalView Blue Gene/P Support

- Continued close collaboration with IBM
- Support for shared libraries, threads, and OpenMP
- TotalView on BG/P has debugged jobs as large as 32,768 cores
- Active workshop participation through the development
  - ANL’s ALCF INCITE Performance Workshop
  - Jülich’s Blue Gene/P Porting, Tuning, and Scaling Workshops
TotalView Blue Gene/Q Support

- Porting TotalView began in June 2011
- Access to Q32 at IBM began in August
- Basic debugging in October 2011
- Used in Synthetic Workload Testing in December (LLNL)
- Fully functional in March 2012
TotalView Blue Gene/Q Support (cont)

- Thanks to the ongoing collaboration with IBM and the BG Kernel Team, early access versions of TotalView were made available for BG/Q.
- At Lawrence Livermore National Laboratory, TotalView has now debugged jobs as large as 768,432 cores.
TotalView

• **TotalView on Blue Gene/Q Today**
  - Lawrence Livermore Labs (LLNL) - USA
  - IDRIS - France
  - CINECA - Italy
  - JULICH - Germany
  - IBM uses TotalView internally for debugging and testing.
  - TotalView is installed on IBM's Blue Gene On Demand Center Q32 (if anyone has access to that system).
Solution: TotalView/MRNet Trees on the IO Nodes

Instead of one daemon managing all 128 CNs

The MRNet Commnode Process connects the daemons to the rest of the tree
TotalView Infrastructure Scalability Strategy

- Implement an additional tree-based infrastructure using MRNet
- Parallelize debugger operations to leverage the tree
  - Convert iteration into Multicast down, and
  - Reduction up, the tree
- Push debugger smarts, not the whole debugger, into the back-end
  - Operations previously handled by the debugger front-end must be pushed down into the debugger back-end or target application
  - Operations requiring symbol table information must send it with the request, or handle it otherwise
- The back-ends must get much smarter, but not much fatter
- Apply "classic optimization" techniques too
  - Caching, hoisting loop invariants, change algorithms/data structures, avoid bottlenecks, e.g., back-end file IO

The "MRNet tree" of servers infrastructure has been added
The "flat vector" of servers infrastructure is still supported
What is TotalView?

A comprehensive debugging solution for demanding parallel and multi-core applications

- Wide compiler & platform support
  - C, C++, Fortran 77 & 90, UPC
  - Unix, Linux, OS X
- Handles Concurrency
  - Multi-threaded Debugging
  - Multi-process Debugging
- Integrated Memory Debugging
- Supports Multiple Usage Models
  - Powerful and Easy GUI – Highly Graphical
  - CLI for Scripting
  - Remote Display Debugging
  - Unattended Batch Debugging
TotalView on BG/Q

- **BG/Q TotalView** is as functional as **BG/P TotalView**
  - MPI, OpenMP, pthreads, hybrid MPI+threads
  - C, C++, Fortran, assembler; IBM and GNU compilers
  - Basics: source code, variables, breakpoints, watchpoints, stacks, single stepping, read/write memory/registers, conditional breakpoints, etc.
  - Memory debugging, message queues, binary core files, etc.

- **PLUS**, features unique to **BG/Q TotalView**
  - QPX (floating point) instruction set and register model
  - Fast compiled conditional breakpoints and watchpoints
  - Asynchronous thread control
TotalView at Argonne

• Licensing
  BG/P: 2048 processes (Latest version available 8.9.0.0)
  BG/Q: 8192+ processes (Research license)

• Startup overview
  Compile–g–O0
  OMP code compile -qsmp=omp:noauto:noopt
  BG/Q: /soft/debuggers/totalview
  Need X11 server and ssh –X forwarding
  [BG/Q] Copy job scripts from /soft/debuggers/scripts/totalview
    runtv.sh
    submit.sh
TotalView Scripts

/s/debuggers/scripts/totalview-examples/

- **To submit:**
  ```bash
  #!/bin/bash
  qsub -t 60 -n 128 --mode script -O LOG --env DISPLAY=$DISPLAY ./runtv.sh
  echo "After your job starts, do a 'tail -f LOG' to see output"
  ```

- **The job script runtv.sh :**
  ```bash
  #!/bin/sh
  # Modify the totalview arguments for your situation
  echo "Starting Cobalt job script"
  echo "DISPLAY is $DISPLAY"
  /soft/debuggers/totalview -args runjob -p 1 -n 128 --block $COBALT_PARTNAME --verbose 2 --envs PAMID_VERBOSE=1 :yourprogram.exe
  ```
Key BG/Q TotalView Features

- Asynchronous thread control
  - Allows you to individually control the execution of threads
  - Run and halt individual threads
  - Single-step a group of threads in lockstep
  - Hold and release the execution of individual threads
  - Create stop-thread and thread barrier breakpoints
A Sampling of Key BG/Q TotalView Features

- Asynchronous Thread and Process Control
- Subset Debugging
- Visualization
- Memory Debugging
- Remote Display Debugging
Scope: Thread/Process Control

Pre-defined Groups

- **Control Group**
  - All the processes created or attached together

- **Share Group**
  - All the processes that share the same image

- **Workers Group**
  - All the processes or threads that are not recognized as manager or service processes or threads

- **Lockstep Group**
  - All threads at the same PC

- **Call Graph Group**
  - All processes going through the same node in the call graph

- **User Defined Group**
  - Process group defined in Custom Groups dialog
Thread/Process Control
Setting Breakpoints

![Image of Action Point Properties window showing options for Breakpoint, Barrier, Evaluate, Group, Process, and Thread, with Location: /home/ehinkel/Source/combined.cxx#505]
Advanced BG/Q TotalView Features

• **Fast compiled conditional breakpoints and watchpoints**
  - Conditional breakpoints and watchpoints execute in as little as 7 \( \mu \)secs
  - Conditional expressions are compiled and dynamically patched into the process
  - Evaluation is performed in parallel by the triggering thread
Evaluation Breakpoints

Test Code Changes on the Fly!

- Test small source code patches
- Call functions
- Set variables
- Test conditions
- C/C++ or Fortran
- Use program variables
- Can’t use C++ constructors
Subset Debugging
With TotalView
Subset Attach

You need not be attached to the entire job
Subset Attach

You need not be attached to the entire job

- You can be attached to different subsets at different times through the run
- You can attach to a subset, run till you see trouble and then 'fan out' to look at more processes if necessary.
- This greatly reduces overhead
- It also reduces license size requirements
Message Queue Graph

- Hangs & Deadlocks
- Pending Messages
  - Receives
  - Sends
  - Unexpected
- Inspect
  - Individual entries
- Patterns
Message Queue Debugging

- Filtering
  - Tags
  - MPI Communicators
- Cycle detection
  - Find deadlocks
Visualization

Get the big picture – Observe anomalies – Utilize Pattern recognition – Save time!
... And Don’t Forget the Memory!
Memory bugs often go undetected until the worst possible time

- Symptoms often surface long after the actual damage is done
- Some only surface after hours or even days of operation
- In many cases, the programs affected are “innocent bystanders”

**MemoryScape: Fully Integrated in TotalView**

- Programs run nearly full speed
- Low performance overhead
- Low memory overhead • Efficient memory usage
The Agent and Interposition
The Agent and Interposition

Process

User Code and Libraries

Heap Interposition Agent (HIA)

Malloc API

Allocation Table

Deallocation Table

TotalView
### Linking Your Application with the HIA Agent

As with Blue Gene/P, if your program is dynamically linked you can avoid having to link the heap agent with your program if you set the LD_PRELOAD variable in your program's environment. For full details, see related documentation at:


<table>
<thead>
<tr>
<th>Platform</th>
<th>Compiler</th>
<th>Binary Interface</th>
<th>Additional linker options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cray XT, XE, XK CLE</td>
<td>-</td>
<td></td>
<td><code>-L&lt;path&gt; -ltvheap_cnl_static</code></td>
</tr>
<tr>
<td>Cray XT3 Catamount</td>
<td>-</td>
<td>64</td>
<td><code>-L&lt;path&gt; -lgmalloc -ltvheap_xt3</code></td>
</tr>
<tr>
<td>IBM Blue Gene/L (static)</td>
<td>-</td>
<td>32</td>
<td><code>-L&lt;path&gt; -ltvheap_bluegene</code></td>
</tr>
<tr>
<td>IBM Blue Gene/P (dynamic)</td>
<td>-</td>
<td>32</td>
<td><code>-L&lt;path&gt; -ltvheap -Wl,-rpath,&lt;path&gt;</code></td>
</tr>
<tr>
<td>IBM Blue Gene/P (static)</td>
<td>-</td>
<td>32</td>
<td><code>-L&lt;path&gt; -ltvheap_bluegene_p</code></td>
</tr>
<tr>
<td>IBM Blue Gene/Q (dynamic)</td>
<td>IBM/GCC</td>
<td>64</td>
<td><code>-L&lt;path&gt; -ltvheap_64 -Wl,-rpath,&lt;path&gt;</code></td>
</tr>
<tr>
<td>IBM Blue Gene/Q (static)</td>
<td>IBM/GCC</td>
<td>64</td>
<td><code>-L&lt;path&gt; -Wl,@&lt;path&gt;/tvheap_bgqs.ld</code></td>
</tr>
</tbody>
</table>
MemoryScape Feature Highlights

- Automatic allocation problem detection
- Heap Graphical View
- Leak detection
- Block painting
- Dangling pointer detection
- Deallocation/reallocation notification
- Memory Corruption Detection - Guard Blocks
- Memory Hoarding
- Memory Comparisons between processes
- Collaboration features
MemoryScape

Visualize Your Program’s Environment…
Memory Event Details Window

Event: Guard corruption error - Bounds error: The guard area around a block has been overwritten

Status: Allocated

Flags: Operation in Progress

Hexadecimal | Count: 80

Bytes

Generate
Memory File

[Image of the Memory Event Details Window]

View in Block Properties window

Help
Memory Corruption Report

Corrupted Memory Report

Options
- Enable Filtering

Table:

<table>
<thead>
<tr>
<th>Preceding Block</th>
<th>Corrupted Block</th>
<th>Following Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Backtrace/Source | Memory Content

Backtrace

Process | Function | Line # | Source Information
------- | -------- | ------ |-------------------
59      | malloc   | 166    | malloc_wrappers_dlope
| corrupt_data | 108 | main.cxx |
| main   | 295     | main.cxx |
| __libc_start_main | | libc.so.6 |
| start  |         | filterapp-mni |

Source

/home/demouser/src/main.cxx

107    po = (int *) malloc( size * sizeof(int));
108    p1 = (int *) malloc( size * sizeof(int));
109    p2 = (int *) malloc( size * sizeof(int));
110
111    __ Common corruption cases. Oop
112    for( i=0; i<size; i++ )
113    

Rogue Wave Software
Block Summary Data

Cursor rollover provides rapid access to block summary data.
Expanding the Debugging Experience with TotalView

Remote Display Debugging
Remote Display Debugging

**Easy, Secure and Fast Remote Debugging Access**

- TotalView Remote Display has two components:
  - a Client, which runs on the user’s remote system
  - a Server, which runs on any TotalView system, and manages the connections between the host and client
- RDC offers the ability to easily set up a remote debug session.
- The Remote Display Client, a free download, is available for:
  - Linux x86
  - Linux x86-64
  - Windows XP
  - Windows Vista
  - Mac OSX
- RDC presents a window on your machine that displays TotalView executing and debugging on a remote system.
## Remote Display Debugging

1. **Enter the Remote Host to run your debug session:**
   - Remote Host: [Input Field]
   - User Name: [Input Field]

2. **As needed, enter hosts in access order to reach the Remote Host:**
   - | Host | Access By | Access Value | Commands |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>User Name</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>User Name</td>
<td></td>
</tr>
</tbody>
</table>

3. **Enter settings for the debug session on the Remote Host:**
   - Path to TotalView on Remote Host: [Input Field]
   - Arguments for TotalView: [Input Field]
   - Your Executable (path & name): [Input Field]
   - Arguments for Your Executable: [Input Field]
   - Submit Job to Batch Queueing System: [Dropdown]
   - [TotalView, MemoryScape]

4. **Enter batch submission settings for the Remote Host:**
   - Submit Command: [Input Field]
   - Script to execute via Submit Command: [Input Field]
   - Additional Submit Command Options: [Input Field]
Remote Display Debugging

Remote Display Security

- Remote Display uses SSH
- Allows only RFB (Remote Frame Buffer) connections to and from the host
- No other incoming access to the Server is allowed
- The Server only connects back to the Viewer via SSH
- Only one Viewer connection is allowed to the Server
- The user is prompted for password information as required
- Also required:
  - ssh must be available on all non-Windows systems being accessed.
  - X Windows must be available on the remote system.
Remote Display Debugging

Session Profile Management

- Profiles are saved with profile names
- Multiple profiles can be generated for various environments
- Profiles can be exported and shared
- Shared Profile files can be imported and shared by other users
Remote Display Debugging

Remote Display Client (RDC)

The scientific community's demand for extreme computational power is not always met because of the difficulty of connecting to host machines, which are sometimes in another time zone, or on another continent. TotalView provides integrated remote display capability to help you fill that void.

SSH encryption gives you secure, rapid access to TotalView and MemoryScape without a local copy license. Connect from a Windows or Linux desktop or laptop to any server or supercomputer architecture that's running TotalView and MemoryScape.

- Easy to set up
- Save, load, and share connection profiles
- End-user installation

Download these free plugins to use TotalView on a remote server:

- Remote Display Client, Windows Installer, version 1.3.1 (Win-32)
- Remote Display Client, version 1.3.1 (Linux-x86-64, Linux-x86)
- Remote Display Client, Mac Installer, version 1.3.1 (Darwin-x86)

Developing for Parallel Architectures

- Code debugging
  - Highly scalable interactive GUI debugger
    - Easy to use -- without sacrificing detail that users need to debug
    - Used from workstations to the largest supercomputers
  - Powerful features for debugging multi-threaded, multi-process, and MPI parallel programs
  - Compatible with wide variety of compilers across several platforms and operating systems
- Memory Debugging
  - Parallel memory analysis and error detection
  - Easily integrated into the validation process
- Reverse Debugging
  - Parallel record and deterministic replay within TotalView
  - Run programs “backwards” to find bugs
  - Now with Record On-Demand
- GPU CUDA Debugging
  - Full Hybrid Architecture Support
  - Asynchronous Warp Control
  - Multi-Device and MPI Support
- Intel PHI (MIC) Support
Developing parallel, data-intensive applications is hard.
We make it easier.

www.roguewave.com