Debugging Large Scale Applications with Virtualization

Filippo Gioachin

September 27, 2010
## Committee

<table>
<thead>
<tr>
<th>Name</th>
<th>Affiliation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laxmikant Kalé</td>
<td>Computer Science</td>
</tr>
<tr>
<td>William Gropp</td>
<td>Computer Science</td>
</tr>
<tr>
<td>Ralph Johnson</td>
<td>Computer Science</td>
</tr>
<tr>
<td>Luiz De Rose</td>
<td>Cray Inc. (external)</td>
</tr>
</tbody>
</table>
Outline

● Introduction
  – Motivations
● Debugging on Large Machines
  – Unsupervised Execution
● Virtualized Debugging
  – Separation of Entities
● Processor Extraction
● Provisional Message Delivery
● Conclusions
Outline

- Introduction
  - Motivations
- Debugging on Large Machines
  - Unsupervised Execution
- Virtualized Debugging
  - Separation of Entities
- Processor Extraction
- Provisional Message Delivery
- Conclusions
Motivations

- Debugging is a fundamental part of software development
- Parallel programs have all the sequential bugs:
  - Memory corruption
  - Incorrect results
  - ....
Motivations (2)

- Parallel programs have new types of bugs:
  - Data races / multicore (heavily studied in literature)
  - Communication mistakes
  - Synchronization mistakes / Message races

- To complicate things further:
  - Non-determinism
  - Problems may show up only at large scale
Problems at Large Scale

- Problems may not appear at small scale
  - Races between messages
    - Latencies in the underlying hardware
  - Incorrect messaging
  - Data decomposition
- Important to handle large scale applications
Challenges to Large Scale Debugging

- Infeasible
  - Debugger needs to handle many processors
  - Human can be overwhelmed by information
  - Machine not available
  - Long waiting time in queue
- Expensive
  - Large machine allocations consume a lot of computational resources

Techniques used in this thesis

- Tight RTS integration
- Unsupervised execution
- Virtualized debug
- Processor extraction
- Provisional delivery
Thesis Goals

● New techniques to help debugging large scale parallel programs
  – Tight integration with runtime system
  – Processor virtualization
● Applying these techniques to message driven parallel programs
Outline

- Introduction
  - Motivations
- Debugging on Large Machines
  - Unsupervised Execution
- Virtualized Debugging
  - Separation of Entities
- Processor Extraction
- Provisional Message Delivery
- Conclusions
Traditional Debugging

- TotalView, Allinea, Eclipse
  - The user manages all the processors directly
- STAT (Stack Trace Analysis Tool) using MRNET
- ATP (Abnormal Termination Processing)
- Relative debugging (requires working program)

External processes that supervise the application
- Information access
- Scalability challenge
Tight Integration with Runtime System

- We use the same communication infrastructure that the application uses to scale
  - Scale debugger with the application
Scalability (1)

[Graph showing scalability of Kraken Cray XT5]
Scalability (3)

The graph shows the time (in milliseconds) taken by different operations on the Kraken Cray XT5 supercomputer as the number of processors increases. The operations include Debugger attach, Set Breakpoint, and PEstat (bcast). The graph indicates that the time taken for all operations increases with the number of processors, but the rate of increase varies depending on the operation.
Autoinspection

- The programmer should not manually handle all the processors
  - Unsupervised execution
  - Notification to the user from interesting processors
    - User-defined
    - System-defined
  - Breakpoints
  - Assertion failure
  - Abort / signals
  - Memory corruption

- Discussed in the prelim (Thesis Chapter 3)
Outline

- Introduction
  - Motivations
- Debugging on Large Machines
  - Unsupervised Execution
- Virtualized Debugging
  - Separation of Entities
- Processor Extraction
- Provisional Message Delivery
- Conclusions
Virtualized Emulation

- Use emulation techniques to provide virtual processors to display to the user
  - Use BigSim emulation tool
    - Cannot assume correctness of program
  - Debugger needs to communicate with application
  - Single address space

Communication under Emulated Environment

- **Virtual Processor**
  - Worker Thread
  - Communication Thread

- **Communication Thread**
  - Message Queue
  - Converse Main Thread

- **Real PE 12**
  - VP 87
  - VP 513

- **Comm. Gateway**

---

27 September 2010

Filippo Gioachin
Resource Consumption: Jacobi (on NCSA's BluePrint)

- User thinks for one minute about what to do:
  - 8 processors
    - 86 sec.
    - ~0.2 SU
  - 1024 procs
    - 60.5 sec.
    - ~17 SU

![Graph showing time per iteration for different numbers of processors.](chart.png)
Demo
Usage: Starting
Usage: Debugging
Separation of Virtual Entities

- Single address space shared by different entities
  - Virtual processors for emulation
  - Multiple chares in Charm++
  - Multiple user-level threads

- One entity can overwrite memory of another entity
  - Dangling pointers (memory reallocated)
  - Pointers passed between entities
  - Spurious writes (e.g. buffer overflow)
Memory Corruption Detection

- Protect memory such that spurious writes can be detected
- Exploit the scheduler in message driven systems

User code: process message

Pick message

Has corruption occurred?

Reset memory protection

Check memory corruption

* F. Gioachin, L.V. Kalé: "Memory Tagging in Charm++" in Proceedings of the 6th Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging (PADTAD ’08)
Protection Mechanisms

- **Checksum: Cyclic Redundancy Check (CRC)**
  - Compute CRC-32 for all the memory in the system. Recompute upon entry method return

- **Memory copy**
  - Copy all the memory in a system in a separate area. Compare upon entry method return

- **mprotect**
  - Allocate memory with mmap and mark read-only with mprotect. Receive signal upon corruption
Performance Aspect

4,000 allocated blocks

32 MB of allocated memory

Overhead (ms)

Total Allocated Memory (MB)

Number of Allocated Memory Blocks (x1000)
Related Work

- Memory protection
  - Studied for concurrent threads (Data Races)
    - Intel Thread Checker
    - RecPlay
      - Not applicable with only one execution thread
  - TotalView
    - User can mimic the protection manually
Outline

- Introduction
  - Motivations
- Debugging on Large Machines
  - Unsupervised Execution
- Virtualized Debugging
  - Separation of Entities
- Processor Extraction
- Provisional Message Delivery
- Conclusions
Do we need all the processors?

- The problem manifests itself on a single processor
- The cause can span multiple processors (causally related)
  - The subset is generally much smaller than the whole system
- Select the interesting processors and ignore the others
Extracting Processors: Challenges

- Record all data processed by each processor
  - Huge volume of data stored
  - High interference with application (probe effect)
    - The bug may not appear
  - Existing: RecPlay, TotalView
  - Work on reduction in space requirements
    - Online analysis of necessary data
    - Processors grouping
    - Record only some processors?
Fighting non-determinism

- Record all data processed by each processor
  - Huge volume of data stored
  - High interference with application (probe effect)
    - The bug may not appear
- Record only message ordering
  - Must re-execute using the whole machine
  - Based on piecewise deterministic assumption
Three-step Procedure for Processor Extraction

1. Execute program recording message ordering

2. Minimize perturbation (few bytes per message)

3. Iterate for incremental extraction

- Has bug appeared?
  - Yes: Select processors to record
  - No: Replay application with detailed recording enabled

- Is problem solved?
  - Yes: Done
  - No: Replay selected processors as stand-alone

- Use message ordering to guarantee determinism
- Can execute in the virtualized environment

What if the piecewise deterministic assumption is not met?

- Make sure to detect it, and notify the user

  *If all messages during replay are identical to those during record, we can assume the application is deterministic*

- Methods to detect failure:
  - Message size and destination
  - Checksum of the whole message (XOR, CRC32)
ChaNGa
(dwf1.2048 on NCSA's BluePrint)
Debugging Case Study

- Message race during particle exchange
  - Was fixed with tedious print statements
    - Printf often made the bug disappear

```bash
../charmrun +p16 ../ChaNGa cube300.param +record +recplay-crc
../charmrun +p16 ../ChaNGa cube300.param +replay +recplay-crc +record-detail 7

gdb ../ChaNGa
>> run cube300.param +replay-detail 7/16
```
Demo
Record-replay with CharmDebug
Outline

● Introduction
  – Motivations
● Debugging on Large Machines
  – Unsupervised Execution
● Virtualized Debugging
  – Separation of Entities
● Processor Extraction
● Provisional Message Delivery
● Conclusions
Provisional Message Delivery

• Instead of replaying the same message ordering, replay a different one
  – Can force the bug to appear on a smaller scale
  – Automatic or manual

• Manual: programmers may have an idea where the problem lies
  – A specific message may confirm or refute the idea
  – Need a way to test without restarting the application
    • Important for bugs that appear after long time
Testing Execution Paths

- Save state of the running application
  - Deliver the message
  - Rollback to try another path (live)

Message received from the network

Process msg B
Process msg A

FORK
PARENT

Save state
Demo
Provisional Delivery: Example
Outline

- Introduction
  - Motivations
- Debugging on Large Machines
  - Unsupervised Execution
- Virtualized Debugging
  - Separation of Entities
- Processor Extraction
- Provisional Message Delivery
- Conclusions
Thesis Contributions

- Techniques to handle thousands of processors efficiently
  - Unsupervised execution with notification upon event

- Reducing the resources required for debugging of large scale applications
  - Virtualized debugging
  - Processor extraction
  - Provisional message delivery

- Techniques to debug message driven parallel applications
Future Extensions

- Shared memory compliance
- Race detector
  - Automated testing of message delivery to discover message races
- Replay in isolation of single virtual entities
  - Conditions of validity
Peer Reviewed Papers

- F. Gioachin, C.W. Lee, L.V. Kale: "Scalable Interaction with Parallel Applications"; in Proceedings of TeraGrid'09
- F. Gioachin, L.V. Kale: "Dynamic High-Level Scripting in Parallel Applications"; in Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS 2009)
- F. Gioachin, L.V. Kale: "Memory Tagging in Charm++"; in Proceedings of the 6th Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging (PADTAD '08)
- C. Mei, G. Zheng, F. Gioachin, L.V. Kale: "Optimizing a Parallel Runtime System for Multicore Clusters: A Case Study"; in Proceedings of TeraGrid'10