Large Scale Debugging
Project Meeting Report - December 2015

Didier Nadeau
Under the supervision of Michel Dagenais

Distributed Open Reliable Systems Analysis Lab
École Polytechnique de Montréal
Table of contents

1 Debugging
   Scalability
   Non-Stop Debugging

2 Tracepoint
   Normal tracing
   Fast tracing

3 OpenMP
   The OpenMP Standard
   Debugging OpenMP
   OpenMP Accelerators
Challenges of parallel debugging

- Scalability to hundreds and thousands of cores
- Ease of use of available commands
- Efficiently collect data from dynamic tracepoints
- Conditional and thread-specific breakpoints
- Minimal perturbation of debuggee
Stopping and continuing threads
Non-Stop debugging

Impact of non-stop debugging

GDB supports non-stop mode: a breakpoint can affect specified threads only.

- Thread identification is handled by GDB
- Context switching is costly
- GDB is single-threaded, possible bottleneck
Non-Stop debugging

Impact evaluation of non-stop breakpoint

A breakpoint for thread 0 was inserted inside a loop executed by thread 1. The average time per iteration was measured.

- Without breakpoint: $10.6 \pm 0.3 \mu s$
- With breakpoint: $133 \pm 4 \mu s$

Using multiple threads

With 7 threads executing the loop, the iteration took up to $727 \pm 14 \mu s$. This is a worst case scenario where GDB is the bottleneck.
Non-Stop Breakpoint
Tracing with GDB

Normal tracepoint

The standard tracing mode uses breakpoints. The debugger collects information and resumes execution. The overhead is very large, possibly more than 100 $\mu s$ per breakpoint.

Fast tracepoint

A fast tracepoint is implemented in the debuggee memory space using a jump and displaced code.
Fast tracepoint

Features

• The main GDB thread is responsible for data collection
• Limited to 5 bytes instructions
• Available as a library to use with GDB

GDB Tracepoint on manycore

It would be interesting to verify if the current implementation scales well.
OpenMP

What is OpenMP

OpenMP is a programming standard that allows developers to easily create multiple threads. It defines an API that is implemented on multiple platforms by various companies.

Features

- Code portability
- Synchronization directives
- Data scope directives
- Support of code offloading to accelerators
Debugging OpenMP programs

OpenMP

OpenMP allows easy parallelization by providing a high-level API. However, this abstraction could be a limitation for debugging.

Possible ideas

- Backtrace for each thread based on the master thread
- Compare private copies of a shared value
- List OpenMP task waiting to be processed
- OpenMP dynamic instrumentation with tracepoints
- Heterogeneous tracing and debugging with OpenMP Target
Accelerators

OpenMP Target

OpenMP 4.0 includes supports for accelerators on which code would be offloaded. It aims to provide a simpler programming interface to use accelerators such as GPUs.

Xeon Phi support

The Xeon Phi supports OpenMP target using Intel Parallel Studio 2015. Intel’s OpenMP implementation is open-source.
Accelerator Debugging

Interesting features

Several features could be interesting to debug offloaded OpenMP code, such as:

- Possibility to step into the target
- Possibility to trace the target
- Synchronization of trace between target and host
Future Work

Manycore debugging

Studying the scalability of debugging and dynamic instrumentation of OpenMP programs with GDB on the Xeon Phi

OpenMP

Useful debugging information for OpenMP and mapping lower level information to the OpenMP constructs.

OpenMP Target

How to integrate debugging and tracing of heterogeneous architectures?
Any Questions?

Contact
didier.nadeau@polymtl.ca