Table of contents

1 GDB Fast tracepoints
   Multicore debugging
   Fast vs Normal tracepoints
   Limits of GDB Fast tracepoints
   Combining LTtng and GDB

2 AMD GPGPU
   GPUOpen
   CodeXL
   GDB
   Future Work
Multicore debugging

Challenges of parallel debugging

- Scalability to 100-1000s cores
- Ease of use of available commands
- Efficient data collection with dynamic tracepoints
- Conditional and thread-specific breakpoints
- Minimal perturbation of debuggee
Tracing with GDB

Normal tracepoint

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

Fast tracepoint

Implemented in the debuggee memory space using a jump instruction and displaced code.
Insertion of a dynamic jump

Replacement of a 5 bytes instruction with a jump to the tracing function
Implementation limits

Use a single buffer shared by every threads

- No concurrent data collection
- Global spinlock in the jump pad

GDBServer handles data transfer

- Stops the program to empty the buffer
- GDBserver must handle the data as well as normal debugging operations
Flush of the agent’s buffer

GDBServer copies the buffer content into its own memory and restarts the thread.
Dynamic insertion of LTTng UST tracepoints

GDB

Provides a way to interact with the program, insert dynamic jumps patches and inspect the program symbols and state.

LTTng UST

Provides a framework for fast tracepoints that scales using lockless buffers.
Implementation

- Use a library of pre-defined tracepoint functions
- Each function has a specific buffer size
- GDB select the appropriate function and links it
- Reuse most of the fast tracepoints logic
Performance comparison

![Graph showing iteration duration (µs) vs number of threads for three different methods: Proposed GDB, Spinlock, and Spinlock + Flush. The graph indicates that the proposed method has the lowest iteration duration compared to the other two methods.]
The GPUOpen Initiative

**GPUOpen**

An initiative launched in 2015 by AMD to provide an open-source software stack to interact with graphic cards for professional use and personal use.

**Heterogeneous System Architecture (HSA) Foundation**

- Provide a standardized interface for programmer
- Multiple instruction sets
- GPUOpen is an implementation by AMD
HSA Specification

Image from HSA Foundation website
A tool suite developed by AMD, the version 2.0 has just been released and is now open-source. Its capacities include:

- Integrated debugging with AMD GPU
- GPU and CPU profiling
- Power profiling
- Using GPU and CPU performance counters
AMD has started to work on a debugger for their gpu using the GPUOpen software stack based on GDB 7.8.

**Features**

- Integrated CPU and GPU debugging
- Inspect GPU state
- Trace kernel launches
Future work

GPUOpen

The debugging and tracing challenges of heterogeneous and manycore systems will be investigated using the GPUOpen software stack.

Fiji Nano graphic card

The Fiji nano, with 4096 processors in 64 compute groups, is the main platform that will be used for these experimentations.
Any Questions?

Contact
didier.nadeau@polymtl.ca