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

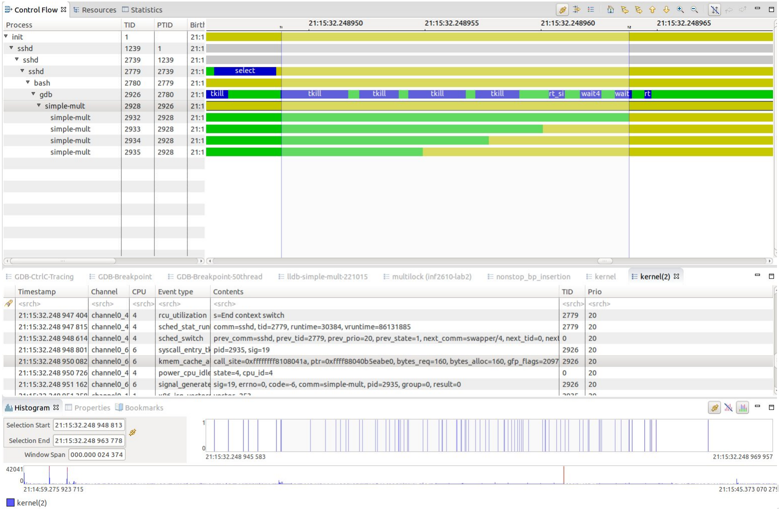
Table of contents

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



Process	TID	PTID	Birth
▾ init	1		21:1
▾ sshd	1239	1	21:1
▾ sshd	2739	1239	21:1
▾ sshd	2779	2739	21:1
▾ bash	2780	2779	21:1
▾ gdb	2926	2780	21:1
▾ simple-mult	2928	2926	21:1
simple-mult	2932	2928	21:1
simple-mult	2933	2928	21:1
simple-mult	2934	2928	21:1
simple-mult	2935	2928	21:1

Timestamp	Channel	CPU	Event type	Contents	TID	Prio
<srch>	<srch>	<srch>	<srch>	<srch>	<srch>	<srch>
21:15:32.248 947 404	channel0_4	4	rcu_utilization	s=End context switch	2779	20
21:15:32.248 947 815	channel0_4	4	sched_stat_run	comm=sshd, tid=2779, runtime=30384, vruntime=86131885	2779	20
21:15:32.248 948 614	channel0_4	4	sched_switch	prev_comm=sshd, prev_tid=2779, prev_prio=20, prev_state=1, next_comm=swapper/4, next_tid=0, next	0	20
21:15:32.248 948 801	channel0_6	6	syscall_entry_tk	pid=2935, sig=19	2926	20
21:15:32.248 950 082	channel0_6	6	kmem_cache_a	call_site=0xffffffff8108041a, ptr=0xffff88040b5eabe0, bytes_req=160, bytes_alloc=160, gfp_flags=2097	2926	20
21:15:32.248 950 726	channel0_4	4	power_cpu_idle	state=4, cpu_id=4	0	20
21:15:32.248 951 162	channel0_6	6	signal_generate	sig=19, errno=0, code=-6, comm=simple-mult, pid=2935, group=0, result=0	2926	20

Selection Start 21:15:32.248 948 813

Selection End 21:15:32.248 963 778

Window Span 000.000 024 374

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 evalution of non-stop breakpoint

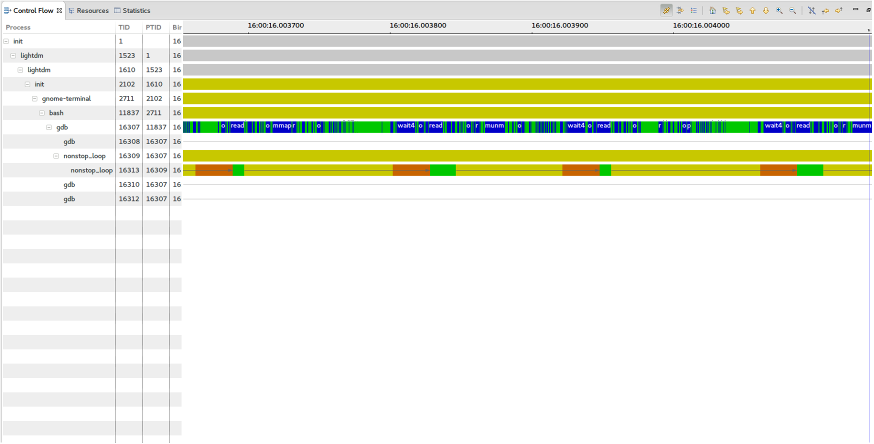
A breakpoint for thread 0 was inserted inside a loop executed by thread 1. The average time per iteration was measured.

- 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



Process	TID	PTID	Bir
init	1		16
lightdm	1523	1	16
lightdm	1610	1523	16
init	2102	1610	16
gnome-terminal	2711	2102	16
bash	11837	2711	16
gdb	16307	11837	16
gdb	16308	16307	16
nonstop_loop	16309	16307	16
nonstop_loop	16313	16309	16
gdb	16310	16307	16
gdb	16312	16307	16

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 μs 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 developpers 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 and 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