LTTng on the Intel Xeon Phi
+
Debugging RPC with GDB

Simon Marchi

Laboratoire DORSAL
Département de génie informatique

POLYTECHNIQUE MONTRÉAL
AFFILIÉE À L'UNIVERSITÉ DE MONTRÉAL

Outline

- Introduction / context of the work
- LTTng on the Intel Xeon Phi
- Debugging RPC with GDB
- Questions

Introduction / context

- Ported LTTng to Tilera Tile-Gx, a many-core processor
- Next step: LTTng on the Intel Xeon Phi, but...
- Let's find something related to many-cores and GDB...

Outline

- ~~Introduction / context of the work~~
- LTTng on the Intel Xeon Phi
- Debugging RPC with GDB
- Questions

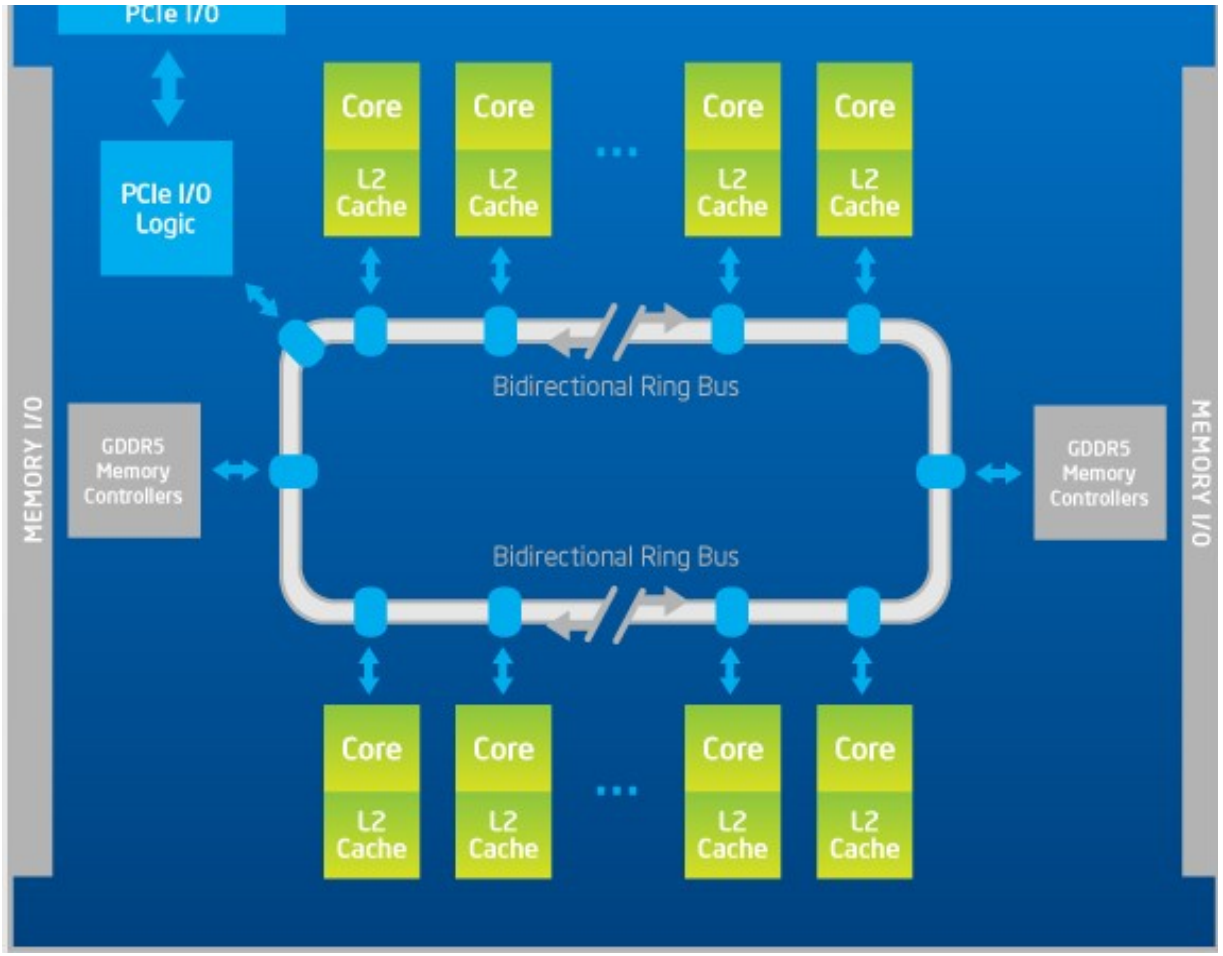
LTTng on the Intel Xeon Phi

- Reminder...
- Coprocessor for offloading heavy tasks
- Linux system in itself
- Network connection with host via PCIe
- Modes of execution:
  - Native
  - Partial offloading

Source: intel.com

LTTng on the Intel Xeon Phi

- 57 physical cores, hyper threaded 4 way
- 6 GB of RAM



LTTng on the Intel Xeon Phi

- Possible problems
  - UST tracepoint registration is long, sometimes hits timeout
  - Buffer memory usage: subbuf size x subbuf num x cores
    - Default kernel: 256 kB x 4 x 228 = 228 MB
  - RAM filesystem: can't save big traces there
  - Network trace streaming: slowdown due to PCIe bus contention?
  - Tracepoints in offloaded code?

LTTng on the Intel Xeon Phi

- Example of offload

```
int main() {
    int a = 5, b = 7, result;

    #pragma offload target(mic) in(a,b) out(result)
    {
        <tracepoint here?>
        result = a + b;
    }
    printf("Result is %d\n", result);

    return 0;
}
```

Outline

- ~~Introduction / context of the work~~
- ~~LTTng on the Intel Xeon Phi~~
- Debugging RPC with GDB
- Questions

Debugging RPC with GDB

- Problem: programs that use remote procedure calls are cumbersome to debug
- Goal: help the user debug the logical flow of the program, from the client to the server
- Each RPC library is different, so we need knowledge about specific libraries
  - SunRPC, XML-RPC, DBus and “home-made” RPC

Debugging RPC with GDB

- Common pattern:
  - Client has a stub function that initiate the RPC
  - Server has a corresponding callback
  - The user doesn't care about what is in between

Debugging RPC with GDB

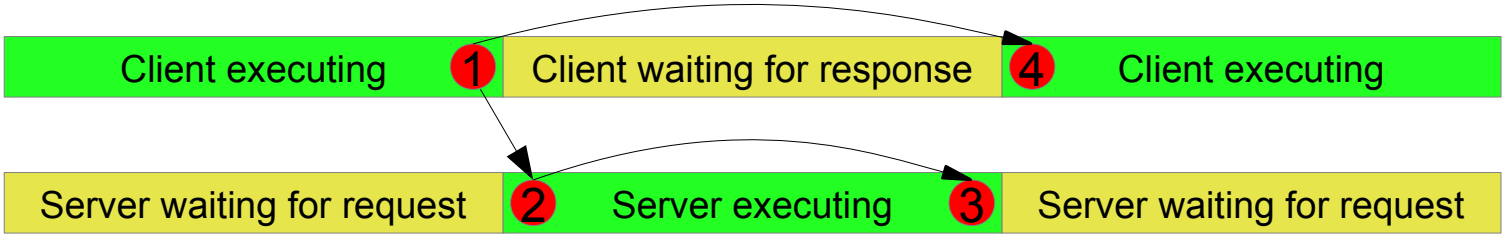
- New commands
  - step-rpc
    - Make a step, try to go “through” the RPC
    - If no RPC call is made, it results in normal step
  - finish-rpc
    - Complete the current RPC call and stop
  - backtrace-rpc
    - Print a combined backtrace between the server and the client, hiding the middleware as much as possible

Debugging RPC with GDB

- GDB provides a (almost) complete Python API
  - Set breakpoints and finish breakpoints programmatically
  - Breakpoint hit callback
  - Inspect threads, inferiors (debugged processes)
  - Inspect stack frames
  - Parse and evaluate language expressions
  - ...
- Missing
  - Inferior control (stop a thread, start a thread)
  - UI Thread switching

Debugging RPC with GDB

- Internals
  1. Client-side “start” breakpoint (installed when step-rpc)
  2. Server-side “start” breakpoint
  3. Server-side “finish” breakpoint
  4. Client-side “finish” breakpoint



Debugging RPC with GDB

- Limitations
  - Both processes need to run under the same GDB
  - Requires debug info for the RPC library
  - Can conflict with network timeouts
- Possible improvements
  - Multiple levels of RPC

Debugging RPC with GDB

- Questions ?