Intel® Xeon Phi™ Product Family
The GNU* Project Debugger (GDB)

Software and Services Group
Intel Corporation
Agenda

• Overview
• Intel® Xeon Phi™ Coprocessor Features
• Native / Manycore hosted Debugging
• Remote Debugging and Offloaded Execution
• Debugging Data Races
• Shared Virtual Memory
• Eclipse Integration
• Summary
Overview

Linux*: Intel debug solution based on GNU* GDB 7.5
  • Capabilities are upstreamed to GNU* community
  • C/C++ support, improved Fortran support
  • Parallel Debug Extensions (PDBX)
  • BTrace: crash-resistant back-trace
  • Pointer Checker: assist in finding pointer issues
  • Intel® Transactional Synchronization Extensions

Intel® Debugger (Intel® IDB) is deprecated
  • Last version remains 13.0
Overview: Intel® Xeon Phi™

GDB* supports Intel® Xeon Phi™ Coprocessors

- Available via Intel® MPSS
  - Via product suite e.g., Intel® Composer XE

GDB* on Intel® Xeon Phi™ Coprocessors

- Native and cross/remote debugger versions
- C/C++ support, improved Fortran support
- Parallel Debug Extensions (PDBX)
GDB* in Intel MPSS

Coprocessor

Debug server and host debugger (PDBX, etc.)
/usr/linux-k1om-4.7/linux-k1om/usr/bin/gdbserver
/usr/linux-k1om-4.7/bin/x86_64-k1om-linux-gdb

Native debugger (no PDBX)
/usr/linux-k1om-4.7/linux-k1om/usr/bin/gdb

Host debugger (PDBX, etc.)
/opt/intel/mic/bin/gdb
Agenda

- Overview
- Intel® Xeon Phi™ Coprocessor Features
  - Native / Manycore hosted Debugging
  - Remote Debugging and Offloaded Execution
  - Debugging Data Races
  - Shared Virtual Memory
  - Eclipse Integration
- Summary
List all new vector and mask registers

(gdb) info registers zmm

K0  0x0  0

<table>
<thead>
<tr>
<th>Zmm31</th>
</tr>
</thead>
<tbody>
<tr>
<td>v16_float = {0x0 &lt;repeats 16 times&gt;},</td>
</tr>
<tr>
<td>v8_double = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},</td>
</tr>
<tr>
<td>v64_int8 = {0x0 &lt;repeats 64 times&gt;},</td>
</tr>
<tr>
<td>v32_int16 = {0x0 &lt;repeats 32 times&gt;},</td>
</tr>
<tr>
<td>v16_int32 = {0x0 &lt;repeats 16 times&gt;},</td>
</tr>
<tr>
<td>v8_int64 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},</td>
</tr>
<tr>
<td>v4_uint128 = {0x0, 0x0, 0x0, 0x0}</td>
</tr>
</tbody>
</table>

Disassemble instructions

(gdb) disassemble $pc, +10

Dump of assembler code from 0x11 to 0x24:
0x0000000000000011 <foobar+17>:
  vpackstorelps %zmm0,-0x10(%rbp){%k1}
0x0000000000000018 <foobar+24>:
  vbroadcastss -0x10(%rbp),%zmm0
Agenda

• Overview

• Intel® Xeon Phi™ Coprocessor Features

• Native / Manycore hosted Debugging

• Remote Debugging and Offloaded Execution

• Debugging Data Races

• Shared Virtual Memory

• Eclipse Integration

• Summary
Native / Manycore hosted Debugging

Run GDB* on the Intel® Xeon Phi™ Coprocessor

```bash
ssh -t mic0 /usr/bin/gdb
```

To attach to a running application via the process-id

```bash
(gdb) shell pidof my_application
42
(gdb) attach 42
```

To run an application directly from GDB*

```bash
(gdb) file /target/path/to/application
(gdb) start
```
Agenda

• Overview
• Intel® Xeon Phi™ Coprocessor Features
• Native / Manycore hosted Debugging
• Remote Debugging and Offloaded Execution
• Debugging Data Races
• Shared Virtual Memory
• Eclipse Integration
• Summary
Remote Debugging

Run GDB* on your localhost
/usr/linux-k1om-4.7/bin/x86_64-k1om-linux-gdb

Start gdbserver on the coprocessor
- To remote debug using ssh
  (gdb) target extended-remote | ssh -T mic0 gdbserver -multi IP:port
- To remote debug usingstdio
  (gdb) target extended-remote | ssh -T mic0 gdbserver -multi -

Attach to a running application via process ID (pid)
(gdb) file /local/path/to/application
(gdb) attach <remote-pid>

Run an application directly
(gdb) file /local/path/to/application
(gdb) set remote exec-file /target/path/to/application
Offloaded Execution

Command line debugging into an offloaded code section on the host does not “switch” to a debugger on the target

• No command-line debug synchronization (host / coprocessor)
• GUI integration does provide this “glue” logic; see “Eclipse* Integration”

Debugging offloaded code via command line

1. Wait within the offloaded code section
   ```c
   volatile int loop = 1;
   do {
       volatile int a = 1;
   } while (loop);
   ```

2. Attach to offload process on coprocessor via PID

Note: cross-compiling the entire application and debugging the previously offloaded section natively might be easier.
Agenda

- Overview
- Intel® Xeon Phi™ Coprocessor Features
- Native / Manycore hosted Debugging
- Remote Debugging and Offloaded Execution
- Debugging Data Races
- Shared Virtual Memory
- Eclipse Integration
- Summary
Debugging Data Races: Example

Given: global variables
   a=1
   b=2

Given: two threads
   T1: x = a + b
   T2: b = 42

Value of x depends on execution order:
   If T1 runs before T2 → x = 3
   If T2 runs before T1 → x = 43

Data race e.g., “read-write”:
   T2’s update was not visible to T1’s calculation
Debugging Data Races: Symptoms

Debug data race symptoms

• Corrupted results (lost updates, large run-to-run variations in results, etc.)
• Corrupted data structures (crash)

Note: different levels of synchronization possible

• Thread-level ordering (global synchronization)
• Instruction level ordering / visibility (e.g., atomics)
  - Race-free but not necessarily run-to-run reproducible results
• No synchronization (data races might be acceptable)
Debugging Data Races: Detection

How to detect data races?
- Compile with "-debug parallel" (icc/icpc/ifort only)
- Coprocessor: debug server with corresponding host GDB*
- Host: enhanced GDB* (/opt/intel/mic/bin/gdb)

Debugger breaks when race has been detected:
(gdb) pdbx enable
(gdb) c
data race detected
1: write shared, 4 bytes from foo.c:36
3: read shared, 4 bytes from foo.c:40

Breakpoint -11, 0x401515 in L_test___21 () at foo.c:36
*var = 42; /* bp.write */

Stop in the context of the access that triggers a race condition
Debugging Data Races: Filter Sets

Fine-tune detection and analysis via filter sets

- Add filter to selected filter set
  (gdb) pdbx filter line foo.c:36
  (gdb) pdbx filter code 0x40518..0x40524
  (gdb) pdbx filter var shared
  (gdb) pdbx filter data 0x60f48..0x60f50
  (gdb) pdbx filter reads # read accesses

- Ignore events specified by filters (default behavior)
  (gdb) pdbx fset suppress

- Ignore events not specified by filters
  (gdb) pdbx fset focus

- Get debug command help (pdbx)
  (gdb) help pdbx

Use cases
- Focused debugging e.g., debug a single symptom
- Limit overhead and control false positives
Agenda

• Overview

• Intel® Xeon Phi™ Coprocessor Features

• Native / Manycore hosted Debugging

• Remote Debugging and Offloaded Execution

• Debugging Data Races

• Shared Virtual Memory

• Eclipse Integration

• Summary
Shared Virtual Memory

Programming model

1. Expose shared resources
   - Allocate and access shared virtual memory (SVM)
   - Address / pointer is valid on host and coprocessor
2. Offload computation
   - Implicit / sparse data transfers (byte or page granularity)
   - Seamlessly share complex data structures

How it works

• MMU page protection faults used to synchronize memory

Use it via

• MYO C API (“Mine Yours Ours”)
• Intel® Cilk Plus™ (C and C++)
Shared Virtual Memory and GDB*

Enhanced GDB* distinguishes between invalid page faults and page faults used for Shared Virtual Memory.

Example: GDB* without SVM-awareness
• Use of SVM triggers segmentation violation
  (gdb) run
  Program received signal SIGSEGV,
  Segmentation fault.
  0x000000000401c90 in foobar()

Example: enhanced GDB* allows transparent debugging
• Awareness for segmentation violations used by SVM
  (gdb) run
  Program exited normally
• Display shared variables similar to regular variables
Agenda

• Overview
• Intel® Xeon Phi™ Coprocessor Features
• Native / Manycore hosted Debugging
• Remote Debugging and Offloaded Execution
• Debugging Data Races
• Shared Virtual Memory
• Eclipse Integration
• Summary
Eclipse Integration

Eclipse* IDE integration

- Seamless debugging of host and coprocessor
- Simultaneous view of host and coprocessor threads
- Supports offload language extensions (auto-attach to offload process)
- Supports multiple coprocessor cards
- Supports C/C++ and Fortran

Simultaneous and seamless thread debugging.
Agenda

• Overview
• Intel® Xeon Phi™ Coprocessor Features
• Native / Manycore hosted Debugging
• Remote Debugging and Offloaded Execution
• Debugging Data Races
• Shared Virtual Memory
• Eclipse Integration
• Summary
Summary

Support for multiple debug models
• Native / remote target debugging
• Eclipse GUI integration

Aware of extended programming models
• Explicit offload model and SVM model

GDB* for Intel® Xeon Phi™ Coprocessor

Debug your offloaded / native code with GDB*
Questions?
Thank You
INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY
ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS
DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR
IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES
RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY
PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on
Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using
specific computer systems, components, software, operations and functions. Any change to any of
those factors may cause the results to vary. You should consult other information and performance
tests to assist you in fully evaluating your contemplated purchases, including the performance of that
product when combined with other products.

Copyright © 2013, Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Xeon Phi, Core,
VTune, and Cilk are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that
are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and
other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on
microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended
for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for
Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information
regarding the specific instruction sets covered by this notice.

Notice revision #20110804
Backup
Shared Virtual Memory: Intel® Cilk Plus™

Annotate resources to be shared between host and target(s)

- Static and/or dynamic memory
  ```cilk_shared int foo = 40;
  ```

- Functions/methods
  ```cilk_shared int inc_foo() {
  return foo++;
  }
  ```

- Offload computation via _Cilk_offload
  ```int main() {
  print_int(_Cilk_offload inc_foo());
  print_int(inc_foo());
  }
  ```