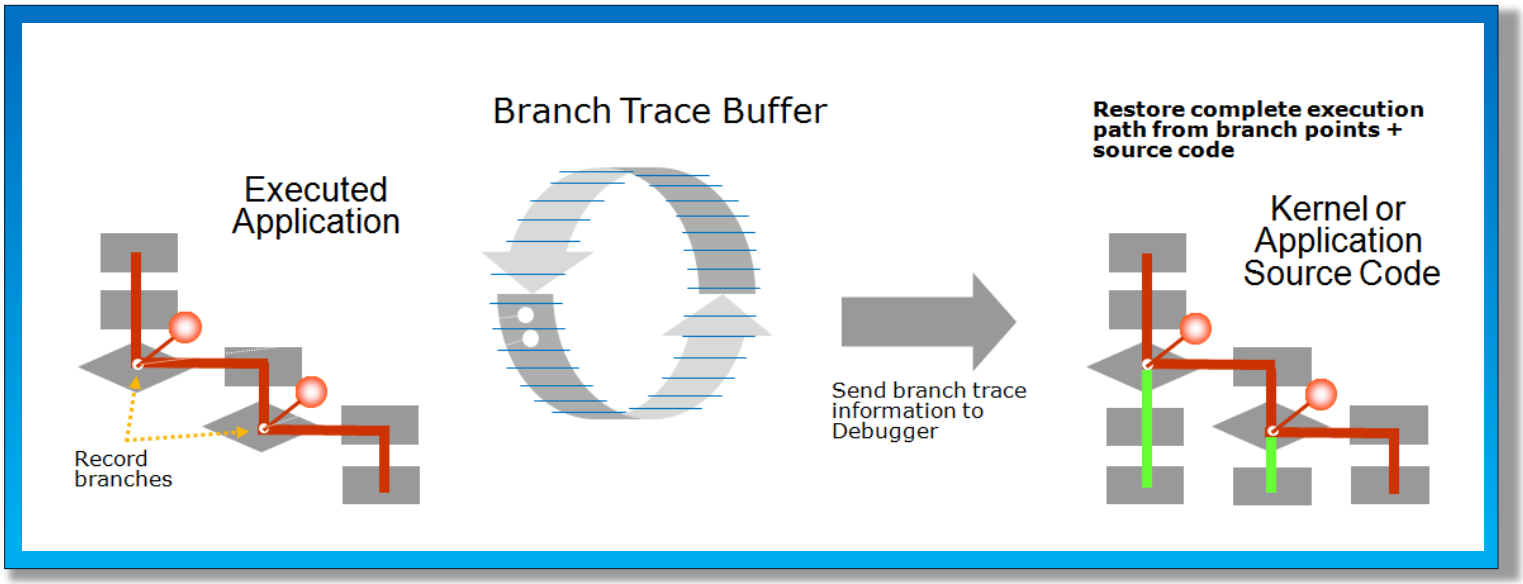
The GNU* Project Debugger - GDB

BTS Instruction Trace



Intel® Atom™ Processor supports *Branch Trace Store* (BTS) using cache-as-RAM or system DRAM

Set breakpoint in OS signal event handler

Unroll execution flow leading up to stack overflow or segmentation fault

Follow execution backwards to where it deviated from expectation

Rerun to that point and analyze memory accesses

Unroll past execution flow:
Where did things start to go wrong?

btrace: h/w supported branch tracing

```
(gdb) btrace list 1-3
1 in foo () at foo.c:23-28
2 in foo () at foo.c:21-22
3 in bar () at foo.c:18
(gdb) btrace /m 3
foo.c:18:       foo (42)
  0x400515 <bar+15>: mov $0x2a, %rax
  0x400519 <bar+19>: call 0x400400 <foo>
(gdb)
```

- H/W records per-thread branch trace.
- Show compact control flow overview.
- Show detailed execution trace disassembly.

Answers "how did I get here?"

btrace: command line syntax

1. btrace enable/disable [all] [auto] [<num>[-<num>]]:

   start/stop recording the branch trace for the current [all] [new] [listed] thread[s]

1. btrace list [/a] [/f] [/l] [/t] [<num>[-<num>]]:

   display a list of blocks

1. btrace [/n] [/m] [+, -, <num>[-<num>]]:

   display the disassembly of one block

"help branchtracing " for syntax details

btrace: use cases

Debug internal state corruption.

Debug stack corruption (broken backtrace).

Quick control flow overview.

btrace: Example

(gdb) btrace enable auto

program crashed and back trace is not much help:

(gdb) run
Starting program: ../gdb/trace/examples/function_pointer/stack64 Program received signal SIGSEGV, Segmentation fault. 0x000000000000002a in ?? ()
(gdb) bt
#0 0x000000000000002a in ?? ()
#1 0x0000000000000017 in ?? ()
#2 0x000000000040050e in fun_B (arg=0x4005be) at src/stack.c:32
#3 0x0000000000000000 in ?? ()

Look at the branch trace.
List of blocks starts from the most recent block (ending at the current pc) and continues towards older blocks such that control flows from block n+1 to block n.

(gdb) btrace list 1-7
1 in ?? ()
2 in fun_B () at src/stack.c:36-37
3 in fun_B () at src/stack.c:32-34
4 in main () at src/stack.c:57
5 in fun_A () at src/stack.c:22-25
6 in fun_A () at src/stack.c:18-20
7 in main () at src/stack.c:51-56

from main(), we called first fun_A() and then fun_B(). The call to fun_A() returned, and we crashed somewhere in fun_B().

btrace: Example

Look at the disassembly of the last 3 blocks in original control flow (i.e. reverse trace) order, starting from the call to fun_B() from main().

/m interleaves source info

(gdb) btrace /m 1-3

```
src/stack.c:32  static long fun_B(void* arg) {
 0x000000000040050e <fun_B+1>:        mov    %rsp,%rbp
 0x0000000000400511 <fun_B+4>:        mov    %rdi,-0x18(%rbp)
src/stack.c:33      struct B_arg* myarg = arg;
 0x0000000000400515 <fun_B+8>:        mov    -0x18(%rbp),%rax
 0x0000000000400519 <fun_B+12>:       mov    %rax,-0x8(%rbp)
src/stack.c:34      if (!myarg) return -1;
 0x000000000040051d <fun_B+16>:       cmpq   $0x0,-0x8(%rbp)
 0x0000000000400522 <fun_B+21>:       jne    0x40052d <fun_B+32>
src/stack.c:36      return myarg->arg1 + myarg->arg2;
 0x000000000040052d <fun_B+32>:       mov    -0x8(%rbp),%rax
 0x0000000000400531 <fun_B+36>:       mov    (%rax),%rdx
 0x0000000000400534 <fun_B+39>:       mov    -0x8(%rbp),%rax
 0x0000000000400538 <fun_B+43>:       mov    0x8(%rax),%rax
 0x000000000040053c <fun_B+47>:       lea    (%rdx,%rax,1),%rax
src/stack.c:37  }
 0x0000000000400540 <fun_B+51>:       leaveq
 0x0000000000400541 <fun_B+52>:       retq
⇒0x000000000000002a:  Cannot access memory at address 0x2a
```

fun_B() is executed and returns to an invalid address suggesting a corrupted stack.
fun_B() leaves but that there was no corresponding push on entry to fun_B()
=> the function pointer comp that was called in main() had been corrupted.

Data Race Detection

Compile with –debug parallel (icc/icpc/ifort only)
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
36*var = 42; /* bp.write */
```

Stop in the context of the racing access

Data Race Detection: Filters

Filters and filter sets to fine-tune the analysis

- Suppress: ignore specified regions
- Focus: ignore non-specified regions

```
(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 fset list
focus filter set bug0815, 1 filters
focus filter set bug42, 2 filters
suppress filter set default, 0 filters
```

Filter all read accesses

```
(gdb) pdbx filter reads
```

Lots of useful filter operations

Date Race Detection: Manage Overhead

Race detection is expensive – very expensive

- Both in performance and memory consumption

Manage overhead effectively through

- Selective enabling
- Focus filter sets
- Global filter on reads

It's an interactive debug tool – use it that way!

Effective filter management is key to fast data race debugging.

Intel® Inspector and GDB* work hand-in-hand

- Two-step approach for debugging data races
  - Step 1: whole-program analysis using Intel® Inspector XE
    - Gives a list of all data races in the program
  - Step 2: reproduce individual races in GDB*
    - Select race to debug
    - Start a new focus filter set
    - For each source line in the race report:
      - Add a filter for that line
- GDB* will now break at each instance of the selected data race with very low overhead

Pointer Checker in Compiler

Pointer checking

- Checks before any memory access through a pointer that the pointer address is inside the object pointed to.

Dangling pointer checking

- Checks for accesses through pointers that have been freed.

Enabled via compile time switches.

Implemented mostly in runtime library code (library is automatically linked in via icl / icc command line)

A user API allows control over what happens when a violation is detected.

Intrinsics allow user to create or destroy bounds for a pointer.

Example

```
void main()
{
        char a[5];
        foo(a);
}
void foo(char *a)
{
        int i;
        for (i=0; i<100; i++) {
            a[i] = 0;
        }
}
```

Compiling and Running

```
icc main.c foo.c –check-pointers=write -g
main
A
B
C
D
E
CHKP: Bounds check error
Traceback:
foo [0x4010E0]
main [0x40104D]
__tmainCRTStartup [0x4014A6]
BaseThreadInitThunk [0x760E3677]
RtlInitializeExceptionChain [0x77869F02]
RtlInitializeExceptionChain [0x77869ED5]
```

Legal Disclaimer & Optimization Notice

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.

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