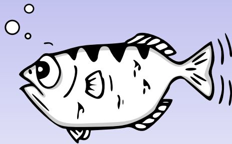


Debugging vs. Hardware Transactional Memory

Andreas Arnez

August 8, 2015, GNU Tools Cauldron, Prague

IBM

1 **Hardware Transactional Memory**

1.1 Overview

1.2 Example

2 Debug Scenarios vs. HTM

3 HTM Debug Support

4 Possible GDB Improvements

5 More Possible GDB Improvements

 © 2015 IBM Corporation

Hardware Transactional Memory (HTM)

What is it?

"Transaction":

Instruction sequence whose memory accesses appear as a single concurrent operation.

- Bracketed by special instructions for *transaction begin* and *-end*.

Compared to other concurrency control mechanisms:

- No lock needed ⇒ may improve locality.
- Can *abort* (with roll-back), e.g.,
  - at data conflicts;
  - upon transaction overflow;
  - at interrupts.

  "best effort HTM"
- Intermediate state not observable ⇒ unique challenges for debugging!

 © 2015 IBM Corporation

Hardware Transactional Memory (HTM)

Implementations

Intel 64 Architecture – Transactional Synchronization Extensions (TSX)

Hardware Lock Elision (HLE)	`XACQUIRE` / `XRELEASE`
Restricted Transactional Memory (RTM)	`XBEGIN` / `XEND`

Power ISA – Transactional Memory Facility

Normal transactions	`tbegin.` / `tend.`
Rollback-only transactions	`tbegin.` / `tend.`

z/Architecture – Transactional Execution Facility

Non-constrained transactions	`TBEGIN` / `TEND`
Constrained transactions	`TBEGINC` / `TEND`

 © 2015 IBM Corporation

Example: z/Architecture Constrained Transaction

Atomically swap two pointers in a structure

Assuming these declarations:

```
struct node { struct node *a, *b; };
void atomic_swap (struct node **ref);
```

atomic_swap could be implemented like this:

```
tbeginc   0, 0xc000      # save %r0-%r3
lg        %r1, 0(%r2)    # p = *ref
lg        %r2, 0(%r1)    # a = p->a
lg        %r3, 8(%r1)    # b = p->b
stg       %r2, 8(%r1)    # p->b = a
stg       %r3, 0(%r1)    # p->a = b
tend
br        %r14           # return
```

© 2015 IBM Corporation

1 Hardware Transactional Memory

2 Debug Scenarios vs. HTM

- **2.1** Crash in Transaction
- **2.2** Single-Stepping
- **2.3** Breakpoints
- **2.4** Watchpoints

3 HTM Debug Support

4 Possible GDB Improvements

5 More Possible GDB Improvements

 © 2015 IBM Corporation

Crash in Transaction

Scenario

```
...        ...             # assume %r2 points to all-zeros
tbeginc    0, 0xc000
lg         %r1, 0(%r2)     # now %r1 == 0
lg         %r2, 0(%r1)     # ← crash here
lg         %r3, 8(%r1)
stg        %r2, 8(%r1)
stg        %r3, 0(%r1)
tend
```

```
(gdb) run
Starting program: /home/arnez/tst/tst

Program received signal SIGSEGV, Segmentation fault.
atomic_swap () at atomic_swap.S:8
8               tbeginc 0, 0xc000
```

 © 2015 IBM Corporation

Crash in Transaction

Analysis

What happened?

Exception $\rightarrow$ transaction abort $\rightarrow$ roll-back. Then:

- PC points at **tbeginc** again.
  - **Note:** other HTM implementations may behave differently, e.g. **tbegin** would stop at the *abort handler*, i.e., *after* **tbegin**.
- Registers are restored to their values before the transaction.
- All the transaction's memory stores are undone.
  - But other threads may have clobbered the same areas by now.

This means:

*The user does not know what **really** caused the fault.*

 © 2015 IBM Corporation

Single-Stepping

```
⇒   tbeginc   0, 0xc000
    lg        %r1, 0(%r2)
    lg        %r2, 0(%r1)
    lg        %r3, 8(%r1)
    stg       %r2, 8(%r1)
    stg       %r3, 0(%r1)
    tend
```

`(gdb) stepi` lands…

- **A** …after **tbeginc**?
- **B** …after **tend**?

A *"Step through":*

- ✘ Not implementable with real transactions.
- ➔ Maybe emulate?
  - ✘ But may lose transaction semantics.

B *"Step over":*

- ➔ Needs special hardware and/or OS support.
  - ✔ Available on Linux on z Systems.

 © 2015 IBM Corporation

Breakpoint in Transaction

Scenario

```
tbeginc   0, 0xc000
lg        %r1, 0(%r2)
lg        %r2, 0(%r1)
lg        %r3, 8(%r1)      # ← set breakpoint here
stg       %r2, 8(%r1)
stg       %r3, 0(%r1)
tend
```

```
(gdb) run
Starting program:  /home/arnez/tst/tst

Program received signal SIGILL, Illegal instruction.
atomic_swap () at atomic_swap.S:8
8               tbeginc 0, 0xc000
```

 © 2015 IBM Corporation

Breakpoint in Transaction

Analysis

Sequence of Events

GDB

1. place trap at breakpoint
2. resume inferior
5. intercept SIGILL
6. compare PC to breakpoint address $\rightarrow$ *no match!*
7. forward SIGILL to inferior.

Inferior

3. hit trap $\rightarrow$ transaction abort $\rightarrow$ roll-back
4. get SIGILL at **tbeginc**

```
(gdb) c
Continuing.

Program terminated with signal SIGILL, Illegal instruction.
The program no longer exists.
```

 © 2015 IBM Corporation

Watchpoint in Transaction

Set watchpoint for the node's second field, `node.b`:

```
tbeginc   0, 0xc000
lg        %r1, 0(%r2)
lg        %r2, 0(%r1)
lg        %r3, 8(%r1)
stg       %r2, 8(%r1)        # ← watchpoint hit
stg       %r3, 0(%r1)
tend
```

```
(gdb) watch node.b
Hardware watchpoint 2:  node.b
(gdb) c
Continuing.  ...hangs...
```

 © 2015 IBM Corporation

Watchpoint in Transaction

Analysis

Sequence of Events

GDB

1. activate hardware watchpoint
2. resume inferior
5. intercept SIGTRAP
6. check whether `node.b` has changed $\rightarrow$ *no!*
7. ignore watchpoint hit
8. go back to step 2.

Inferior

3. hit watchpoint $\rightarrow$ transaction abort $\rightarrow$ roll-back
4. get SIGTRAP at **tbeginc**

 © 2015 IBM Corporation

1 Hardware Transactional Memory

2 Debug Scenarios vs. HTM

3 HTM Debug Support

- **3.1** Non-Transactional Stores
- **3.2** Transaction Diagnostics
- **3.3** Gaining Control Over Transactions

4 Possible GDB Improvements

5 More Possible GDB Improvements

 © 2015 IBM Corporation

Non-Transactional Stores

I've been hunting this bug all night and day;
now I know: to find it a ***trace*** *I must add!*
But soon it shows – oh, it's just too sad –:
aborted transactions wipe their traces away.

But. . .

. . . what if we can temporarily escape transaction semantics?

Power ISA:

```
…
tsuspend.
stw        r5, 0(r11)
tresume.
…
```

z/Architecture:

```
…
ntstg   %r5, 0(%r11)
…
```

 © 2015 IBM Corporation

z/Architecture Transaction Diagnostic Block

Program-Interruption Transaction Diagnostic Block (TDB):

- Stored by hardware upon interrupted transaction.
- E.g., at breakpoint, watchpoint, crash in transaction.

Field	Offset	Description
tdb0	0	format, flags, transaction nesting depth
tac	8	transaction abort code
tct	16	conflict token
atia	24	aborted-transaction instruction address (ATIA)
tr0	128	aborted-transaction general register 0
…	…	…
tr15	248	aborted-transaction general register 15

✔ Access from GDB:

```
(gdb) p/a $atia
$1 = 0x8000060c <atomic_swap+12>
```

 © 2015 IBM Corporation

Gaining Control Over Transactions

Example: z/Architecture

Feature	**Description**	**Possible GDB exploitation**
Transactional-execution control	Enable/disable HTM instructions.	Intercept transaction begin.
PER event-suppression control	Suppress certain debug events during transactions.	*Step over* transactions.
Transaction diagnostic control	Randomly abort transactions.	Test inferior's abort path.

 © 2015 IBM Corporation

1 Hardware Transactional Memory

2 Debug Scenarios vs. HTM

3 HTM Debug Support

4 Possible GDB Improvements

5 More Possible GDB Improvements

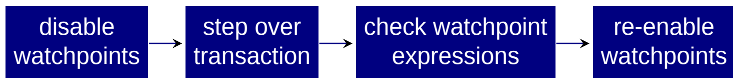
 © 2015 IBM Corporation

Single-Step at Transaction

Step over transaction:

✔ Implemented for Linux on z Systems.

But what about watchpoints? ➔ Special handling required, e.g.:



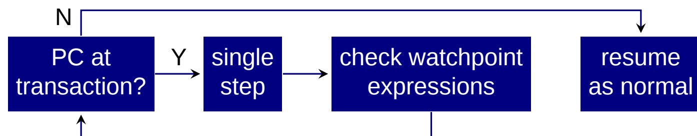

```
(gdb) x/i $pc
=> 0x800005f8 <atomic_swap>:    tbeginc 0,49152
(gdb) stepi
Hardware watchpoint 2:node.b

Old value = (void *) 0x80002080
New value = (void *) 0x80002070
atomic_swap () at atomic_swap.S:15
15                br      %r14
```

 © 2015 IBM Corporation August 8, 2015

Resume at Transaction

OK, assume stepping over transaction works correctly.
But if we do **`continue`** instead? ⇒ Issues with watchpoints again!

➔ Special handling required, e.g.:




```
(gdb) continue
Continuing.
Hardware watchpoint 2:node.b
Old value = (void *) 0x80002080
New value = (void *) 0x80002070
atomic_swap () at atomic_swap.S:15
15                br       %r14
```

 © 2015 IBM Corporation

Watchpoint Hit Within Transaction

Scenario: inferior running $\rightarrow$ watchpoint hit in transaction $\rightarrow$ transaction abort.

➔ Special handling required, e.g.:

"transaction intercept mode"

- If inferior retries transaction:
  - Intercept $\rightarrow$ single-step $\rightarrow$ watchpoint expression changed (again) – *or not*.
- Otherwise:
  - May intercept unrelated transaction.

 © 2015 IBM Corporation

Breakpoint Hit Within Transaction

Scenario: inferior running → breakpoint hit in transaction → transaction abort.

➔ Special handling for breakpoint trap required, e.g.:

```
(gdb) run
Starting program: /home/arnez/tst/tst
Breakpoint 1, atomic_swap () at atomic_swap.S:8
8               tbeginc 0, 0xc000
```

(Please disable breakpoint before continuing.)

 © 2015 IBM Corporation

1 Hardware Transactional Memory

2 Debug Scenarios vs. HTM

3 HTM Debug Support

4 Possible GDB Improvements

5 More Possible GDB Improvements

 © 2015 IBM Corporation

State at Transaction Abort

Question: Where did a breakpoint/watchpoint/crash *actually* occur?

Idea: Treat transaction abort state as an implicit GDB trace snapshot.

- E.g., accessed with a new **`tfind`** sub-command **`xab`**:

```
(gdb) tfind xab
Selected last transaction abort state.
... (do some debugging)
(gdb) tfind end
... (continue live debugging)
```

- E.g., in case of z/Architecture use data from the *transaction diagnostic block*.

 © 2015 IBM Corporation

Step Into Transaction

Possible recipe for (“emulated”) stepping into a transaction:

- Stop other threads until end of transaction.
  - (Non-stop mode off, scheduler-locking mode on, schedule-multiple mode off.)
- Use reversible debugging: Record all instruction steps until end of transaction.
- But don’t step transaction-related instructions; emulate them.
  - E.g., “transaction abort” instruction: roll back transaction and perform appropriate abort handling.

Could combine this with *transaction intercept mode*, to offer a full *transaction emulation mode*.

- Slow, but convenient.
  - No more problems with breakpoints, watchpoints, etc.

 © 2015 IBM Corporation