Hardware-Assisted Online Monitoring

Volker Stolz

Invited talk

VORTEX 2022

6 June 2022

Supported by the Norwegian Research Council through grant “COEMS Training Network” (309527)
Hardware-assisted Online Monitoring

- **Overview**
  - Why? How?
  - Some examples from the RV community and beyond
- **An example in depth:** the COEMS hardware platform
  - Data race detection
Why Hardware-assisted Online Monitoring?

- Advantage of FPGAs [Baumeister et al. 2019]:
  - faster processing speed due to the inherent parallelism
  - energy consumption
  - ease of integration within a CPS

- With respect to taxonomy of Runtime Verification:
  - may be used with or without(!) instrumentation
  - requires centralized architecture
  - easiest to use for asynchronous online monitoring

ZedBoard with Zynq-7000

Taxonomy [Falcone et al. 2018]
Some Assembly Required

▶ General workflow:
  - Translate your favourite specification language into VHDL or Verilog...
  - ...either directly or through some middleware (RustHDL, Kôika, Cava, ...).
  - Provide adapters for I/O
  - Synthesize & deploy!

▶ Challenges:
  - How to develop adapters?!
  - Synthesis sloooow
  - Limited space on FPGA
  - How to reconfigure running system with monitor?

- Stream-based specification language
- Sensors = input, output = *periodic* or *event-based streams*

compiled into VHDL components:
High-/Low-level controller + queue

Event sources:
- **Program Flow Trace** ("jumps" between basic blocks)
- Instrumentation for data accesses

Processing:
- DIFT coprocessor on FPGA
- Executes abstract program on IF data (separate RAM)
- Precomputed effect of each BB through static analysis
Design Space & Alternatives

- Tracing abstract events?
  Send data from system/application to FPGA
- Tracing actual binaries?
  Use hw-tracing facilities!
  - reconstruct program flow
  - combine with static analysis
  - intersperse with necessary data through instrumentation

Alternative to SoC:
PCI Express-based cards

- “Easy” to combine with ARM CoreSight or Intel PT
- You still need the trace decoder wrt. the binary

Issue: sourcing
(thank you, Bitcoin)
Hardware-based Approach to Monitoring Races

Idea of the https://www.coems.eu project:

- Software-/instrumentation-based approaches have high overhead
- Offloading runtime checks to separate hardware
- Out-of-band monitoring, no runtime reflection possible
- Avionics & railway domain:
  - predictable performance independent from whether or not race checking is enabled;
  - can’t swap between normal and debugging version in the field.

- Variable/memory accesses
- Compute sets of locks held by threads which are accessing variables/memory
- Initialize the set of locks guarding variable/memory with the set of all locks
- On each access, \( \text{guarding} := \text{guarding} \cap \text{holding} \)
- Error when \( \text{guarding} == \emptyset \)
COEMS Trace Box containing FPGA (left) and SoC (right)
Lock instrumentation & race monitoring with COEMS

L. Convent, S. Hungerecker, T. Scheffel, M. Schmitz, D. Thoma, A. Weiss: Hardware-Based Runtime Verification with Embedded Tracing Units and Stream Processing, RV’18, LNCS.
Example of incorrect locking

```c
/* get an exclusive lock on both balances before updating (there's a problem with this, see below) */
pthread_mutex_lock(transaction_mtx);

if( !DATA_RACE || (DATA_RACE & (id != 0)) ){
    /* In case of DATA_RACE flag is 'on', the thread_id 0 forgets to lock the accts[from].mtx mutex */
    pthread_mutex_lock(&accts[from].mtx);
}

pthread_mutex_lock(&accts[to].mtx);

pthread_mutex_unlock(transaction_mtx);

/* Do the actual transfer. */
if (accts[from].balance > 0) {
    payment = 1 + rand_range(accts[from].balance);
    accts[from].balance -= payment;
}

pthread_mutex_unlock(&accts[to].mtx);

if( !DATA_RACE || (DATA_RACE & (id != 0)) ){
    /* For symmetry -- don't unlock if racy: */
    pthread_mutex_unlock(&accts[from].mtx);
}
```
Header of the TeSSLa spec, including all the incoming events from the instrumented code
Details on Leucker's
https://www.tessla.io.

```haskell
in produce: Events[Unit]
in consume: Events[Unit]

def numProduce := count(produce)
def numConsume := count(consume)
def safe := numProduce - numConsume <= 2
```
```python
def lock_0 := filter(mutexlockaddr == 1, mutexlockaddr == 1)
def unlock_0 := filter(mutexunlockaddr == 1, mutexunlockaddr == 1)
def lock_1 := filter(mutexlockaddr == 24808, mutexlockaddr == 24808)
def unlock_1 := filter(mutexunlockaddr == 24808, mutexunlockaddr == 24808)
def dyn_temp := 4 * (((dyn_lock >> 1) >> 1))
def slot := dyn_lock - dyn_temp
def dyn_lock_0 = filter(dyn_temp, slot == 0)
def lock_4 := filter(mutexlockaddr == dyn_lock_0, mutexlockaddr == dyn_lock_0)
def unlock_4 := filter(mutexunlockaddr == dyn_lock_0, mutexunlockaddr == dyn_lock_0)
def read_0 := filter(readaddr == 24532, readaddr == 24532)
def write_0 := filter(writeaddr == 24532, writeaddr == 24532)
def access_0 := merge(read_0, write_0)
def access_after_pc_0 := on(last(pcreateid, access_0), line)
def thread_accessing_0 := last(threadid*32768 + line, access_after_pc_0)
def holding_0 := default(merge(last(threadid, lock_0), last(-1, unlock_0)), -1)
def protecting_0_with_0 := detect_change(default( thread_accessing_0/32768 == last(holding_0, thread_accessing_0), true))
def error_0 := on(thread_accessing_0, !(protecting_0_with_0 || protecting_0_with_1 || protecting_0_with_2 || protecting_0_with_3 || protecting_0_with_4))
```
Lock instrumentation & race monitoring with COEMS

```c
void f() {
    ... lock(&m);
    x++;
    ...
}
```

```
main.c
... call @lock(@m)
load @x
add ...
store @x 
... 

main.bc
... call @lock(@m)
ITM(... line ...) 
ITM(lock, @m)
ITM(... line ...)
ITM(read, @x)
load @x
add ...
ITM (...line...)
ITM(write, @x)
store @x 
... 
```

```
epu.cfg.txt
... 
```

```
enclustra
 ... 
```

```
CoreSight
 ... 
```

```
main.bc
... call @lock(@m)
load @x
add ...
store @x 
... 
```

```
epu-output
... 
```

```
epu-cfg.txt
... 
```

```
FPGA
 ... IR
```

```
TeSSLa-
interpreter
... 
```

```
w.tessla
... 
```

```
mkDR
... 
```

```
sw.tessla
... 
```

```
trace.txt
... 
```

Volker Stolz
Hardware-Assisted Online Monitoring
Recompilation of Specifications

Advantage of COEMS:

▶ Can reconfigure monitoring on the fly...
Advantage of COEMS:

- Can reconfigure monitoring on the fly...
- \ldots for some value of “on the fly”.
Recompilation of Specifications

Advantage of COEMS:

- Can reconfigure monitoring on the fly...
- ... for some value of “on the fly”.
- Size of specification scales with number of locks & memory locations (not: threads!)

EPU compilation time for hw.tessla
EPU compilation time for hw.tessla

TeSSLa interpreter startup time for sw.tessla
Events emitted by the COEMS trace box after processing the hardware half of the TeSSLa spec.

Race report obtained by processing the above events including debug information
Overhead introduced by the instrumentation

- COEMS platform in general does not need instrumentation for control-flow events...
- ...but we need instrumentation if data is involved.
- Ideally: Arm CoreSight/Intel PT can do this via simple `mov`-instruction
We got it to work :-)  
(Hardware, specification language with compiler, toolchain)  
Acceptable overhead to industry partners  
Can’t rule out some (inherited) false-positives

```c
for (i = 0; i < N_THREADS; i++)
    pthread_join(ts[i], NULL);
for (total = 0, i = 0; i < N_ACCOUNTS; i++)
    total += accts[i].balance;
printf("Total money in system: %ld\n", total);
```
We got it to work :-)  
(Hardware, specification language with compiler, toolchain)

Acceptable overhead to industry partners

Can’t rule out some (inherited) false-positives

```
for (i = 0; i < N_THREADS; i++)
    pthread_join(ts[i], NULL);
for (total = 0, i = 0; i < N_ACCOUNTS; i++)
    total += accts[i].balance;
printf("Total money in system: %ld\n", total);
```

Visit us on
https://www.coems.eu
or
https://tessla.io