Gathering Memory Hierarchy Statistics in QEMU

Clément Deschamps, Mark Burton, Eric Jenn, Frédéric Pétrot
GreenSocs, IRT Saint Exupéry, Univ. Grenoble
Can I optimize my code for the memory hierarchy?

• Just after “Does my code work”, this seems to be the biggest preoccupation for embedded software.

• And we have answered – till now, by counting accesses.

But now...

**CACHE COHERENCY**
Takeaway message:

EMU + PV+T =
Qemu: defacto standard Virtualizer

- GreenSoc is a contributor:
  Xilinx components, power, reset, clocks, Multi-Core TCG Kernel...
  Regular committers from many organizations

# of Architectures: 18
# of CPU's: 1100
# of Commits: 43000
# of Contributors: 1000
# of Lines of Code: 989,863
Qbox, Connecting Qemu and SystemC

- Connect QEMU and SystemC
- Can use any QEMU model within SystemC

- How is synchronization done:
  - WHOLE OTHER DISCUSSION...
  - Come to the SystemC Evolution Day!
“PV+T” timewarp.....

• Sooo long ago, sooo far away, 3 people tried to convince the TLM WG (!) that they should support a modeling style they called “PV+T” – they failed!

• The idea was simple, simulate your device, and then add timing.
• Run a functional model, record all the information you need, and feed that into a model of the architecture to evaluate the time.
“PV+T” timewarp.....


Function separate from time?

NOTICE:
Information flows one way!
(The timing model does not even have to execute at the same time, it can post-process)
The plan

• Capture enough information from QEMU to feed into a cache model
  – (No need to know the ‘actual’ data!)
  – Need to know all D-Side accesses
  – Need to know all I-Side accesses

• Run the cache model in a separate host threads

• Model L1, (and L2 Cache coherency).
QEMU - MTTCG

Multi Thread Tiny Code Generator
GreenSocs contribution to QEMU, started in 2014.
Can potentially generate a lot of **concurrent** memory accesses.
(just like real hardware)

Alex Bennée, Towards multi-threaded TCG, KVM Forum 2015
Memory paths
The value given by e.g. a DMI access
Memory paths

Same for both I side and D side
Memory paths

QEMU's softmmu
QEMU - Memory Access

Basic Block

IN: ldr r1, [fp]

If the entry is in the Softmmu TLB, the path is very quick

Alex Bennée, Towards multi-threaded TCG, KVM Forum 2015
Path specifics

• On the I side, we need both Virtual and Physical addresses (e.g. A53 I-side caches are virtually indexed and physically tagged!).
  – Notice that on the I-Side, the QEMU softMMU does a direct translation from Guest virtual to Host virtual. We have added Guest physical addresses to this structure.
  – But, this can be calculated as ‘JIT Compile time’, rather than for each access – so we generate I side accesses per TB. (Handled by a new field in TLB structure).

• D-side slow : simply modify slow path helper.

• D-side fast path we have to insert code – this is harder . . .
Memory paths

- CPU to Guest Virtual
- Guest Virtual to TLB
- TLB to Guest Physical
- Guest Physical to Memory model
- Memory model to Host Virtual
- Host Virtual to Host Memory

QEMU's softmmu

Add Guest Physical to TLB structure
Path specifics

• On the I side, we need both Virtual and Physical addresses (e.g. A53 I-side caches are virtually indexed and physically tagged!).
  – Notice that on the I-Side, the QEMU softMMU does a direct translation from Guest virtual to Host virtual. We have added Guest physical addresses to this structure.
  – But, this can be calculated as ‘JIT Compile time’, rather than for each access – so we generate I side accesses per TB. (Handled by a new field in TLB structure).

• D-side slow : simply modify slow path helper.

• D-side fast path we have to insert code – this is harder . . .
A **Basic Block** corresponds to a list of instructions terminated by a branch instruction. A Translation Block (**TB**) is the translation of a basic block into host instructions.
QEMU - Fast Path / Slow Path

Input Code (ARM)  Output Code (x86-64)

**Basic Block**

**IN:** ldr r1, [fp]
### QEMU - Fast Path / Slow Path

#### Input Code (ARM)

**IN:** `ldr r1, [fp]

#### Output Code (x86-64)

**OUT:** `movl -0x24(%r14), %ebp

**Prologue:**
- `movl -0x24(%r14), %ebp`
- `testl %ebp, %ebp`
- `jl 0x7f70c001234d`
- `movl 0x2c(%r14), %ebp`

**TLB Load:**
- `movl %ebp, %edi`
- `leal 3(%rbp), %esi`
- `c1 ef 05 shrl $5, %edi`
- `andl $0xffff00, %esi`
- `andl $0x1fe0, %edi`
- `leaq 0x6e00(%r14, %rdi), %rdi`
- `cmpl 0(%rdi), %esi`
- `movl %ebp, %esi`
- `jne 0x7f70c0012359`

**Fast Path:**
- `addq 0x10(%rdi), %rsi`
- `movl 0(%rsi), %ebp`

**Epilogue:**
- `movl %ebp, 0(%rsi), %ebp`

**TLB Hit**

**Slow Path:**
- `movq %r14, %r8d`
- `movl $0x2e3, %edx`
- `leaq *0xa(%rip)`
- `movl *0xa(%rip), %eax`
- `movl %ebp, %eax`
- `movl 0x7f70c001233c`

**Go back to execution**

**TLB Miss**

**TLB Load:**
- `movl %ebp, %edi`
- `leal 3(%rbp), %esi`
- `c1 ef 05 shrl $5, %edi`
- `andl $0xffff00, %esi`
- `andl $0x1fe0, %edi`
- `leaq 0x6e00(%r14, %rdi), %rdi`
- `cmpl 0(%rdi), %esi`
- `movl %ebp, %esi`
- `jne 0x7f70c0012359`

**Fast Path:**
- `addq 0x10(%rdi), %rsi`
- `movl 0(%rsi), %ebp`

**Epilogue:**
- `movl %ebp, 0(%rsi), %ebp`

**TLB Miss**

**TLB Load:**
- `movl %ebp, %edi`
- `leal 3(%rbp), %esi`
- `c1 ef 05 shrl $5, %edi`
- `andl $0xffff00, %esi`
- `andl $0x1fe0, %edi`
- `leaq 0x6e00(%r14, %rdi), %rdi`
- `cmpl 0(%rdi), %esi`
- `movl %ebp, %esi`
- `jne 0x7f70c0012359`

**Fast Path:**
- `addq 0x10(%rdi), %rsi`
- `movl 0(%rsi), %ebp`

**Epilogue:**
- `movl %ebp, 0(%rsi), %ebp`

**TLB Miss**
Implementation: Record Load/Store

Extra code for each TLB lookup (even on the fast path!)
Results

Benchmark:
- Modeling an ARM VExpress Board, with a single core cortex-a9
- Running dhrystone for ARM (5,000,000 runs)
- Simulation takes around 11 seconds without instrumentation
- 1.2 Billion guest memory accesses (800M reads, 400M writes)

Comparison of implementations:

- **Version 1**: Call a C helper from TB
- **Version 2**: Generate assembly code in TB to fill a shared buffer.
- **Version 3**: Code in TB, to fill a buffer (lock free).
The Cache model

Model gets addresses that are accessed (for read or write).
   It can model anything.

Model follows real H/W,
   Complete with Linefill buffers, etc.
Time is not modeled
   (e.g. sometime it’s easier to count the statistics at different ‘times’ than the real h/w)

Model ONLY holds addresses, not data
Some results – looks good!

![Bar chart](chart.png)

- L1 Hits
- 1.E+05
- 1.E+06
- 1.E+07
- 1.E+08

**Polybench test benches**

- 2mm
- 3mm
- atax
- bicg
- cholesky
- doitgen
- gemm
- gemver
- gesummv
- mvt
- symm
- syr2k
- syrk
- trisolv
- trmm

**Legend**

- HW
- QEMU
Some results – looks good!

We use the Zynq hardware
And the **arm PMU** to
evaluate real H/W numbers.
And **modeled** the PMU in
**QEMU**!
• QEMU runs at about real-time for a 600 mHz Zynq board.

• Adding a multi-threaded cache model adds ONLY about 10% overhead

This is for an R5 single core target, across all polybench benchmarks.
Questions

Mark @ GreenSocs.com