The ARMv8 Simulator

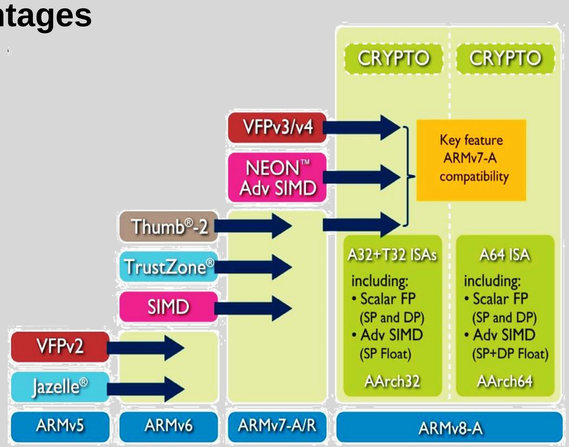
Tao Jiang[1,2], Lele Zhang[1,2], Rui Hou[1], Yi Zhang[1,2], Qianlong Zhang[1], Lin Chai[1,2],
Jing Han[1,2], Wuxiang Zhang[1], Cong Wang[1,2], Lixin Zhang[1]

Institute of Computing Technology, Chinese Academy of Sciences[1]

University of Chinese Academy of Sciences, Beijing, China[2]

ARMv8 architecture

- Next version of the ARM architecture
- First ARM 64-bit instruction set (A64)
  - –Full compatibility with ARMv7
- Focus on power efficient architecture advantages



Why need ARMv8 simulator?

The first ARMv8 CPU are due in 2014 !

- No ARMv8 performance simulator available
- Goal of Our ARMv8 Simulator
  - –Easy to use
  - –Easy to debug
  - –Reliable
  - –Accurate
  - –Multiple CPU models
  - –Power simulation

- It is the first open source ARMv8 performance simulator

ARMv8 simulator features

Features	Support details
CPU Models	Atomic Simple, Timing Simple, In-Order, O3
TLP	Multi-core and SMT
Memory System	Ruby and Classic
System mode	SE mode
Power model	McPAT
ISA	All the instructions other than SIMD

- Based on gem5
  - –a modular simulation platform
  - –support most commercial ISAs
- Implement decoder of A64 instructions
- Support System-call Emulation mode
- Support to interface with other gem5 modules

Implementation

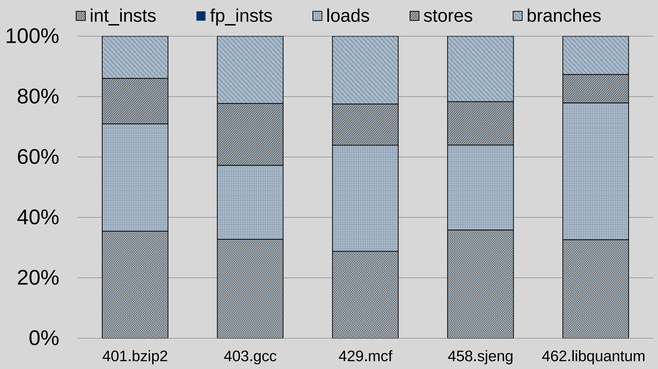
- Implement A64 instructions based on gem5
  - –Analyze “ARMv8 Instruction Set Overview”
  - –Consult ARMv8 gcc compiler
  - –including integer and floating-point instructions
- Support System-call Emulation (SE)
  - –Run binary executable files
  - –Emulate the system calls
- Validation
  - –Compare with ARMv8 Foundation Model
  - –Evaluate some benchmarks

```
    0: system.cpu.[tid:0]: Setting int reg 32 (32) to 0x7fffffff000.
    0: system.cpu.[tid:0]: Setting int reg 0 (0) to 0.
    0: system.cpu.[tid:0]: Setting int reg 8 (8) to 0xffffffffffffffff.
    0: global: Reading From misc reg 51 (51) : 0xc50008
    0: global: Reading From misc reg 55 (55) : 0
    0: global: Reading From misc reg 96 (96) : 0x98aa4
    0: global: Reading From misc reg 97 (97) : 0x40e048e0
    0: global: Reading From misc reg 76 (76) : 0
    0: system.physmem: IFetch of size 4 on address 0xbf0 data 0xd280001d
    0: system.cpu.[tid:0]: Setting int reg 31 (31) to 0.
    0: system.cpu.[tid:0]: Setting int reg 29 (29) to 0.
    0: system.cpu T0 : 0x400bf0    : movzImm_64 r29, 0x0, LSL #0x0 : IntAlu :  D=0x0000000000000000
  500: system.physmem: IFetch of size 4 on address 0xbf4 data 0xd280001e
  500: system.cpu.[tid:0]: Setting int reg 31 (31) to 0.
  500: system.cpu.[tid:0]: Setting int reg 30 (30) to 0.
  500: system.cpu T0 : 0x400bf4    : movzImm_64 r30, 0x0, LSL #0x0 : IntAlu :  D=0x0000000000000000
 1000: system.physmem: IFetch of size 4 on address 0xbf8 data 0x910003fd
 1000: system.cpu.[tid:0]: Setting int reg 31 (31) to 0.
 1000: system.cpu.[tid:0]: Reading int reg 32 (32) as 0x7fffffff000.
 1000: system.cpu.[tid:0]: Setting int reg 29 (29) to 0x7fffffff000.
 1000: system.cpu T0 : 0x400bf8    : addImm64_2  r29, r32, 0x0, LSL #0x0 : IntAlu :  D=0x0000007fffffff000
 1500: system.physmem: IFetch of size 4 on address 0xbfc data 0xaa0003e5
 1500: system.cpu.[tid:0]: Setting int reg 31 (31) to 0.
 1500: system.cpu.[tid:0]: Reading int reg 31 (31) as 0.
 1500: system.cpu.[tid:0]: Reading int reg 0 (0) as 0.
 1500: system.cpu.[tid:0]: Setting int reg 5 (5) to 0.
 1500: system.cpu T0 : 0x400bfc    : orrReg64  r5,r31, r0, LSL 0x0 : IntAlu :  D=0x0000000000000000
 2000: system.physmem: IFetch of size 4 on address 0xc00 data 0xf94003e1
 2000: system.cpu.[tid:0]: Setting int reg 31 (31) to 0.
 2000: system.cpu.[tid:0]: Reading int reg 32 (32) as 0x7fffffff000.
 2000: global: Reading From misc reg 51 (51) : 0xc50008
 2000: global: Reading From misc reg 55 (55) : 0
 2000: global: Reading From misc reg 96 (96) : 0x98aa4
 2000: global: Reading From misc reg 97 (97) : 0x40e048e0
 2000: global: Reading From misc reg 76 (76) : 0
 2000: system.physmem: Read of size 8 on address 0x84000 data 0x1
 2000: system.cpu.[tid:0]: Setting int reg 1 (1) to 0x1.
 2000: system.cpu T0 : 0x400c00    : ldrImm120c                    : MemRead :  D=0x0000000000000001 A=0x7fffffff000
 2500: system.physmem: IFetch of size 4 on address 0xc04 data 0x910023e2
 2500: system.cpu.[tid:0]: Setting int reg 31 (31) to 0.
 2500: system.cpu.[tid:0]: Reading int reg 32 (32) as 0x7fffffff000.
 2500: system.cpu.[tid:0]: Setting int reg 2 (2) to 0x7fffffff008.
 2500: system.cpu T0 : 0x400c04    : addImm64_2  r2, r32, 0x8, LSL #0x0 : IntAlu :  D=0x0000007fffffff008
 3000: system.physmem: IFetch of size 4 on address 0xc08 data 0x910003e6
```

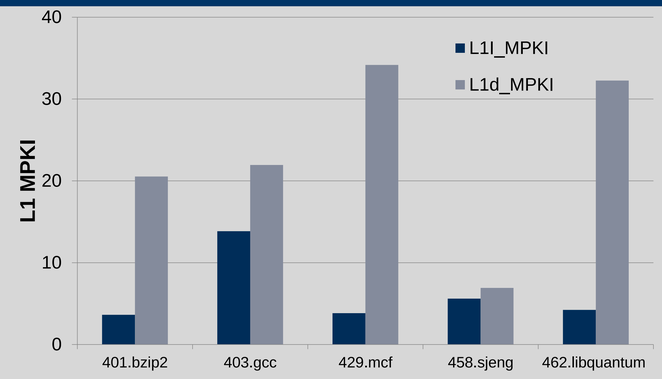
Validation & Results

- "apple-to-apple" comparison with ARMv8 Foundation Model
- Workloads
  - –Micro-benchmarks
    - Assembly instructions
  - –SPEC CPU2006
  - –Fhourstones
  - –Dhrystone
  - –Stream

Test results



Future Work



- Supports a large number of statistics collection
  - –Cache miss
  - –Instruction execution status
  - –Power consumption
  - –…
- ARMv8 simulator is working, but could be better
- More extensions possible
  - –Full-system simulation
  - –More system calls
  - –Trace driven mode