Valgrind on RISC-V

1. Introduction to Valgrind's internals and RISC-V
2. Enablement of Valgrind on RISC-V
3. Current state and future work

Petr Pavlů `<petr.pavlu@dagobah.cz>`

Valgrind structure

- Coregrind – core facilities, glues everything together.
- VEX – library for dynamic binary instrumentation and translation.
- Tool plugins – Memcheck, Nulgrind, Cachegrind, Callgrind, DHAT, DRD, Helgrind, Massif.

Valgrind artwork, retrieved from `https://valgrind.org/`.

VEX overview

- The VEX library powers Valgrind's dynamic re-compilation process.
- 8-phase translation:

Phase	Translation
⋆ 1. Disassembly	machine code $\rightarrow$ tree IR
2. Pre-instr. optimization	tree IR $\rightarrow$ flat IR
3. Instrumentation	flat IR $\rightarrow$ flat IR
4. Post-instr. optimization	flat IR $\rightarrow$ flat IR
5. Tree building	flat IR $\rightarrow$ tree IR
⋆ 6. Instruction selection	tree IR $\rightarrow$ instruction list
7. Register allocation	instruction list $\rightarrow$ instruction list
⋆ 8. Assembly	instruction list $\rightarrow$ machine code

Asterisk indicates architecture-specific phases.

RISC-V overview

- An open-standard instruction set architecture (ISA).
- Started in 2010 at UC Berkeley as a part of research project.
- 32-bit and 64-bit ISA, the 128-bit variant in draft state.

RISC-V logo, retrieved from `https://riscv.org/`.

64-bit RISC-V ISA

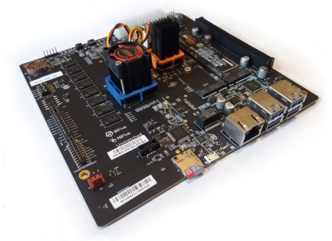
- 64-bit base ISA and main extensions:

Name	Description	#Instrs
RV64I	Base instruction set	52
RV64M	Integer multiplication & division	13
RV64A	Atomic	22
RV64F	Single-precision floating-point	30
RV64D	Double-precision floating-point	32
RV64Zicsr	Control & status register	6
RV64Zifencei	Instruction-fetch fence	1
RV64C	Compressed	37

- RV64G is a shorthand for RV64IMAFDZicsrZifencei.
- Linux distributions target RV64GC.

Valgrind on RISC-V

- Target is the 64-bit RISC-V architecture and the Linux operating system (RISCV64/Linux).
- An out-of-tree port, started a year ago.
- Available at `https://github.com/petrpavlu/valgrind-riscv64/`.
- Development done on QEMU and HiFive Unmatched, running the openSUSE Tumbleweed distribution.



Photography of HiFive Unmatched, own work.

Example translation

- Translated instruction is ADDI (Add Immediate). Mnemonic: `addi rd, rs1, imm12`.
- Input trace is for simplicity limited to only one instruction (`--vex-guest-max-insns=1`).
- Selected tool is Nulgrind (`--tool=none`) $\rightarrow$ no instrumentation is inserted.

ADDI example – 1. Disassembly

- Input:

```
0x10534:   13 85 15 00   # addi a0, a1, 1
```

- Output IR:

```
------ IMark(0x10534, 4, 0) ------
PUT(96) = Add64(GET:I64(104),0x1:I64)
PUT(272) = 0x10538:I64
PUT(272) = GET:I64(272); exit-Boring
```

ADDI example – 6. Instruction selection

- Input IR:

```
------ IMark(0x10534, 4, 0) ------
PUT(96) = Add64(GET:I64(104),0x1:I64)
PUT(272) = 0x10538:I64; exit-Boring
```

- Output instruction list:

```
0   (evCheck) lw t0, -2040(s0); c.addiw t0, -1;
       sw t0, -2040(s0); bge t0, zero, 1f;
       ld t0, -2048(s0); c.jr 0(t0); 1:
1   ld       %vR4, -1944(s0)
2   li       %vR5, 0x1
3   add      %vR3, %vR4, %vR5
4   sd       %vR3, -1952(s0)
5   (xDirect) li t0, 0x10538; sd t0, -1776(s0);
       li t0, <disp_cp_chain_me_to_fastEP>;
       c.jalr 0(t0)
```

ADDI example – 7. Register allocation

- Input instruction list:

```
0   (evCheck) [...]
1   ld      %vR4, -1944(s0)
2   li      %vR5, 0x1
3   add     %vR3, %vR4, %vR5
4   sd      %vR3, -1952(s0)
5   (xDirect) [...]
```

- Output instruction list:

```
0   (evCheck) [...]
1   ld      a7, -1944(s0)
2   li      a6, 0x1
3   add     a5, a7, a6
4   sd      a5, -1952(s0)
5   (xDirect) [...]
```

ADDI example – 8. Assembly

- Input instruction list:

```
0   (evCheck) [...]
1   ld      a7, -1944(s0)
2   li      a6, 0x1
3   add     a5, a7, a6
4   sd      a5, -1952(s0)
5   (xDirect) [...]
```

- Output machine code:

```
0   83 22 84 80 FD 32 23 24 54 80 63 D5 ...
1   83 38 84 86
2   05 48
3   B3 87 08 01
4   23 30 F4 86
5   B7 02 01 00 9B 82 82 53 23 38 54 90 ...
```

Test results

- Initial focus on Memcheck and Nulgrind tests.
- Current state:

Test set	#Failed/#Total
Memcheck	26/219
Nulgrind	5/140
Cachegrind	7/7
Callgrind	12/15
DHAT	0/8
DRD	18/130
Helgrind	45/55
Massif	2/37
GDBserver	24/25

Current state and future work

- Current focus is on functionality and correctness.
- Enabled RV64GC instructions:

Name	Description	#Enabled/ #Instrs
RV64I	Base instruction set	51/52
RV64M	Integer multiplication & division	12/13
RV64A	Atomic	22/22
RV64F	Single-precision floating-point	2/30
RV64D	Double-precision floating-point	2/32
RV64Zicsr	Control & status register	0/6
RV64Zifencei	Instruction-fetch fence	0/1
RV64C	Compressed	36/37

- Future plans: codegen optimizations, support for other extensions (BitManip, Vector, . . .)?