GenSim

Hands-On Session

Harry Wagstaff, Tom Spink, Bruno Bodin, Bjoern Franke

Institute for Computing Systems Architecture
University of Edinburgh

October 2018

Introduction

In this session, you will learn:

- How to build GenSim from source
- How to perform simulations using ArchSim
- How to collect useful information from ArchSim
- How to add to existing GenSim models

Introduction

GenSim tested on the following Linux distributions:

- Ubuntu 16.04
- Ubuntu 18.04
- Fedora 26
- Fedora 27
- ArchLinux
- Debian 9.3

If you have another distribution, or another OS, you may need to use a GenSim VM image to take part in this session.

Materials

To complete this session, you will also need to simulate some binaries. These binaries are pre-built, and can be obtained from the GenSim Tutorial downloads section at:
`http://bitbucket.org/gensim/gensim-tutorial/downloads`
Extract this archive to a directory of your choice. In this slide deck, we'll be referring to this directory as `$(mat)`, e.g.,

`$(mat)/running-archsim/hello`

Building GenSim

Building GenSim takes three steps:

1. Install dependencies
2. Check out GenSim source code
3. Compile

Install dependencies

GenSim has the following dependencies, which are generally available and can be installed with your distro's package manager (they may have different names):

- autoconf
- cmake
- default-jre-headless
- g++
- libantlr3c-dev
- libncurses5-dev
- make
- mercurial
- wget
- zlib1g-dev

Check Out Source Code

GenSim source code is kept in a Mercurial repository on BitBucket. The code can be obtained by checking out that repository, by running the command:

```
hg clone http://bitbucket.org/gensim/gensim
```

After the repository is checked out, you can change directory into the repository:

```
cd gensim
```

Compile

At this point, everything should be ready for you to compile GenSim! Simply run

```
make
```

... and a short while later GenSim should be compiled. If you have a multicore machine and wish to use additional compilation agents, you can run

```
make -j{N}
```

Where {N} is the number of build agents to use.

The Built Tools

You can find the built targets in gensim/build/dist/bin:

archsim	The ArchSim simulator
archsim-armv7a-user	A script to run ARMv7a binaries
gensim	The GenSim ADL Processing tool
TraceCat	A tool to format binary trace files
TraceLess	A pager for binary trace files
TracePCDiff	A diff tool (based on PC) for trace files

Getting a GenSim VM

If you can't get GenSim to build, or don't have access to one of the supported Linux distributions, you can try using a pre-prepared VirtualBox image.

Running ArchSim

Now that everything is built, it's time to run some simulations!

```
gensim/build/dist/bin/archsim-armv7a-user $(mat)/running/hello
```

Running ArchSim

Now that everything is built, it's time to run some simulations!

```
gensim/build/dist/bin/archsim-armv7a-user $(mat)/running/hello
```

Output:

```
Archsim: The Edinburgh High Speed (EHS) Simulator
        University of Edinburgh (c) 2017
                Revision : 5979651a74c9+ tip
                Configuration: "Debug (Opt)"

Hello, world!
```

Advanced Configuration

Although the default configuration is fine for straightforward simulation, if you want to simulate other architectures or systems then you need to call ArchSim directly.

Advanced Configuration

`archsim-armv7a-user` is a script which calls ArchSim. Let's have a closer look at it, and in particular the line which actually calls ArchSim:

```
$ARCHSIM -m arm-user -s armv7a -l contiguous -e $ELF
```

Advanced Configuration

`archsim-armv7a-user` is a script which calls ArchSim. Let's have a closer look at it, and in particular the line which actually calls ArchSim:

```
$ARCHSIM -m arm-user -s armv7a -l contiguous -e $ELF
```

The Emulation Model - How is the binary loaded? What happens when an exception occurs?

Advanced Configuration

`archsim-armv7a-user` is a script which calls ArchSim. Let's have a closer look at it, and in particular the line which actually calls ArchSim:

```
$ARCHSIM -m arm-user -s armv7a -l contiguous -e $ELF
```

The Guest Architecture - What guest architecture should be used?

Advanced Configuration

`archsim-armv7a-user` is a script which calls ArchSim. Let's have a closer look at it, and in particular the line which actually calls ArchSim:

```
$ARCHSIM -m arm-user -s armv7a -l contiguous -e $ELF
```

The Memory Model - How should underlying memory be stored?

Advanced Configuration

`archsim-armv7a-user` is a script which calls ArchSim. Let's have a closer look at it, and in particular the line which actually calls ArchSim:

```
$ARCHSIM -m arm-user -s armv7a -l contiguous -e $ELF
```

The target binary - which binary should be loaded for simulation?

Advanced Configuration

Let's try running with a custom configuration. We'll add the `--verbose` flag, which will cause simulation information to be printed:

```
archsim -m arm-user -s armv7a -l contiguous --verbose -e $(mat)/running/hello
```

This will cause some simulation statistics to be printed, including speed, instruction count, simulation time, etc.

RISC-V

GenSim also has a RISC-V model. We can use this model by changing the command-line flags:

```
archsim -m riscv-user -s riscv -l contiguous -e $(mat)/running/hello-riscv
```

Obtaining Simulation Statistics

ArchSim provides features to collect simulation statistics in a variety of ways. We've already seen how basic statistics can be collected, but they're mainly about the simulator rather than the simulated binary.
Additional profiling features include histograms of:

- Instruction types executed
- PC frequencies
- Instruction code frequencies

Profiling Information

These histograms can be obtained with the following command-line flags:

- Instruction types: `--verbose --profile`
- PC frequencies: `--verbose --profile-pc`
- Instruction codes: `--verbose --profile-ir`

The IR and PC histograms are written out to ir_freq.out and pc_freq.out.

Tracing

If these features do not provide enough detail, then a full trace can be obtained and processed.
The following flags can be added to enable tracing:

```
--trace --trace-file $(output-file)
```

This will write a trace out to a file called $(output-file).

Tracing

The TraceCat and TraceLess tools can be used to view the traces. Alternatively, a custom tool can be used to perform other analysis or trace-based simulation.

```
build/dist/bin/TraceLess $(trace file)
```

This tool behaves like `less` (although not quite as feature rich)

Modifying an Existing Model

Now, let's try modifying the RISC-V model.

The New Instruction

Let's add an integer dot product instruction to the instruction set. The instruction will operate on two pairs of registers:

```
$(rd) = $(rs1) * $(rs1+1) + $(rs2) * $(rs2 + 1)
```

RISC-V has instruction space reserved for customisation, which this instruction will use.

Encoding the Instruction

First, we need to settle on the encoding for the instruction. The 32-bit RISC-V opcode space is divided into 28 subspaces, several of which are reserved for custom extensions.

We will use the custom-0 space, which uses the 0b0001011 opcode.

We'll use the R-Type format, since we're doing an operation on registers.

To keep things simple, we'll fill 0s in to the instruction function fields.

Instruction Syntax

Let's start editing the model! Open up `gensim/models/risc-v/riscv_isa.ac`.
This is the Syntax Description file for the base RISC-V instruction set. To keep things simple, we'll edit this file rather than creating a new file for our extension.

Instruction Syntax - Adding the new Instruction

If you scroll to line 39, you'll find a section of the file which declares which instructions are available in the instruction set. Each instruction is attached for a format, and they are declared in a 'template'-like manner.

Line 52 specifies the instructions for the R-Type format. Add a `dotproduct` entry to the end of this line, like this:

```
ac_instr<Type_R> add,sub,sll,slt,sltu,xor,srl,sra,or,and,dotproduct;
```

Instruction Syntax - Declaring the Instruction Behaviour

From Line 74, the instruction semantic behaviours are declared. We need to add a new line for our dotproduct instruction

```
ac_behaviour dotproduct;
```

Instruction Syntax - Decoding the new Instruction

From Line 124, the decoding for each instruction is described. For our new dotproduct instruction, we'll need to add a couple of new lines. Add the following lines to the section (the position doesn't matter):

```
dotproduct.set_decoder(opcode=0x0b, funct3=0x0, funct7=0x0);
dotproduct.set_behaviour(dotproduct);
```

The Instruction Semantics

That's all we need to decode the instruction. Now, we need to specify the semantics of the instruction. Open `gensim/models/risc-v/execute.riscv`.

In this file, the semantic behaviour of each RISC-V instruction is specified.

The Instruction Semantics

Add the following text to the start of the file:

```
execute(dotproduct) {
        uint32 rs1a = read_register_bank(GPR, inst.rs1);
        uint32 rs1b = read_register_bank(GPR, inst.rs1+1);
        uint32 rs2a = read_register_bank(GPR, inst.rs2);
        uint32 rs2b = read_register_bank(GPR, inst.rs2+1);

        uint32 result = (rs1a * rs1b) + (rs2a * rs2b);

        write_register_bank(GPR, inst.rd, result);
}
```

Building the model

We've now added enough to the model to decode and execute our new instruction, so we need to rebuild the model. Go to the root of the gensim repository and run `make`. If the system builds with no errors, then you can now run simulations using the new instruction!

Using The Instruction

Now that we have built the model containing the new instruction, we can run a program which uses it! The `$(mat)/riscv-dotproduct` directory contains an example program which uses this instruction. We can run this program to test our new instruction:

```
archsim -s riscv -m riscv-user -l contiguous -e $(mat)/riscv-dotproduct/dotproduct
```

```
Operands: (1804289383, 846930886) (1681692777, 1714636915)
C Result: 4088578517
Asm Result: 4088578517
```

Recap

To summarize, we've:

- Checked out and built GenSim
- Run some simple simulations using ArchSim
- Added a new instruction to the RISC-V model
- Run a binary which uses the new instruction

Conclusion

Now that you've had a bit of experience using GenSim and ArchSim, we'll look in more detail at the individual components of a GenSim model.