MAGPIE TUTORIAL

First steps with gem5

Abdoulaye Gamatié, Pierre-Yves Péneau

LIRMM / CNRS-UM, Montpellier

ComPAS Conference, June 2017, Sophia-Antipolis

Other contributors: S. Senni, T. Delobelle, Florent Bruguier, L. Torres, G. Sassatelli

PROJET FINANCÉ PAR L'ANR – PROJECT FUNDED BY THE ANR

Roadmap

- Requirements for this tutorial
- Preparing gem5 to simulate an application
  - Operating system configuration
  - Application compilation
  - Hard drive (disk image) preparation
- Automate the application execution
  - Shell scripts
- Accelerate execution with checkpoints

REQUIREMENTS FOR THIS TUTORIAL

Connection to the remote server

- Open 2 terminals
- Connection to the remote server in each terminal

```
$ ssh etu-f_compas2017-XX@muse-login.hpc-lr.univ-montp2.fr
```

- MAGPIE configuration in one terminal only:

```
$ source $HOME/work/env.sh # Could takes few seconds
$ python --version (returns Python 2.7.12)
```

gem5 pre-setup

- For those who are on their **laptop with administrative rights**, download a script at this URL:

https://frama.link/compas17_nfs

- Install the following package: **sshfs**

- Execute this script on your own machine
  - Requires administrative rights to mount remote filesystem (enter your remote access password 3 times)

GEM5 SETUP

Operating system configuration

- gem5 needs a compiled kernel and a disk image with full operating system stuff

- Default kernels are provided in **`$M5_PATH/binaries`**
  - 32 bits : `vexpress.aarch32.ll_20131205.0-gem5`
  - 64 bits : `vexpress.aarch64.20140821`

- Disk image with complete Ubuntu system are also given in **`$M5_PATH/disks`**
  - Ubuntu 11.04 32 bits : `aarch32-ubuntu-natty-headless.img`
  - Ubuntu 14.04 64 bits : `aarch64-ubuntu-trusty-headless.img`

Operating system configuration

- Linux also needs a DTB file (Device Tree Blob/Binary)

- Contains hardware information required by Linux
  - Number of CPUs, frequency, DVFS information
  - Level of caches, size, line size
  - Bus frequency
  - Address range for devices
  - …

- Located in `$M5_PATH/binaries`
  - Ex: `vexpress.aarch32.ll_20131205.0-gem5.1cpu.dtb`

Application cross-compilation

- MAGPIE is designed to work with ARM Instruction Set Architecture (ISA)
  - Could be extended though
- Our machines are (mostly) x86
- We need a **cross-compilation**: generate ARM assembly from another ISA (i.e., x86)

Application cross-compilation

- Cross-compiler available on remote machine (32-bits)

```
$ arm-unknown-linux-gnueabi-gcc --version
```

  - Note: this compiler is for ARM 32-bits only

- Replace **`gcc`** by this one to produce ARM 32-bits executables

- Always compile with **`-static`** flag
  - This gem5 version doesn't support dynamic linking

Application cross-compilation

- Example : `$MAGPIE/app/hello_magpie.c`

```
$ arm-unknown-linux-gnueabi-gcc –static –o
  $HOME/app/hello_magpie $MAGPIE/app/hello_magpie.c
```

- Verification

```
$ file $HOME/app/hello_magpie
ELF 32-bit LSB executable, ARM, EABI5 version 1
(SYSV), statically linked, for GNU/Linux 3.12.72,
not stripped
```

  - This is also available on your local machine in `/tmp/app`

Disk image modification

- Disk images are just ISO file : we can mount, read and write into them
- The following commands require **administrative rights**
  - That won't work on this server
- We are just showing you how to proceed

Disk image modification

- For this tutorial we use this disk image: `linux-aarch32-ael.img` (32-bits)
  - Located in `$HOME/disks` on the remote server

```
$ ls $HOME/disks
linux-aarch32-ael.img
```

  - Available on your local machine in `/tmp/disks`

```
$ ls /tmp/disks
linux-aarch32-ael.img
```

This is the same file !

Disk image modification

- Mount the disk image:

```
$ sudo mount –o loop,offset=32256                    \
  /path/to/linux-aarch32-ael.img /mnt
```

- Copy the executable inside:

```
$ sudo cp /path/to/hello_magpie /mnt/benchmark
```

- Finally, unmount and synchronize:

```
$ sudo umount /mnt && sudo sync
```

gem5 setup conclusion

- Operating system:
  - in `$M5_PATH/binaries`
  - With DTB files
- Application is statically compiled for ARM-32 bits
- Disk image is in `$M5_PATH/disks`
  - And `$HOME/disks`
  - Application is copied to the disk image in `/benchmark`

AUTOMATE APPLICATION EXECUTION

Automate application execution

- gem5 provides an interface to execute commands inside the operating system: `rcS` files
- Shell script automatically executed after the boot phase
- Example in `$GEM5/configs/boot/hello.rcS`

```
#!/bin/sh

echo "Hello World"
/sbin/m5 exit      # Special instruction to exit gem5
```

Automate application execution

- Open a new file in `$HOME/app/hello_magpie.rcS`
  - vi, vim, emacs or nano are installed
- Our `hello_magpie` application is located in `/benchmark`
- What would be the content of this file ?

```
#!/bin/sh

cd /benchmark      # Change directory
./hello_magpie     # Launch application
/sbin/m5 exit      # Special instruction to exit gem5
```

- This will be executed after booting

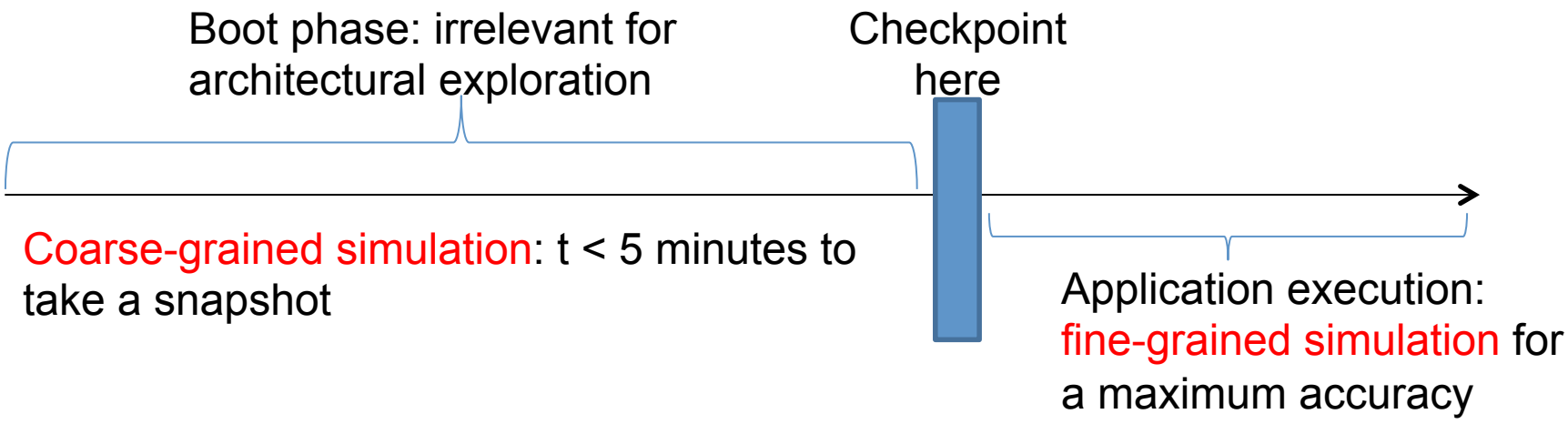
- The simulation will automatically exit when finished

USE CHECKPOINTS

What's a checkpoint ?

- Booting Linux in gem5 could be very long
  - ~30 minutes with 1 core & fine-grained simulation
  - One way to mitigate this: **checkpoints**

- Checkpoints are snapshots of the system state
  - Taken with coarse-grained simulation
  - Restore from checkpoint with high level of details



Taking a checkpoint

- Restrictions when taking checkpoint and restoring
  - Use the same amount of main memory
  - Use the same disk image
  - Use the same number of cores

- How to take a checkpoint for MAGPIE:

Output folder !

```
$ cd $GEM5
$ ./build/ARM/gem5.fast –ed $CHKPT/chkpt-1core-2GB                \
  configs/example/fs.py                                           \
  --script configs/boot/hack_back_ckpt.rcS                        \
  --disk-image $M5_PATH/disks/linux-aarch32-ael-filled.img        \
  --dtb-filename vexpress.aarch32.ll_20131205.0-gem5.1cpu.dtb      \
  --mem-size 2GB ––num-cpus 1
```

- Use your disk image if possible: `$HOME/disks/linux-aarch32-ael.img`

Taking a checkpoint

- gem5's output:

```
gem5 Simulator System.  http://gem5.org
[…]
**** REAL SIMULATION ****
[…]
Writing checkpoint
info: Entering event queue @ 2789663483500.
Starting simulation...
Exiting @ tick 2791525410000 because m5_exit
instruction encountered
```

- Checkpoint is located in `$CHKPT/chkpt-1core-2GB`

```
$ ls $CHKPT/chkpt-1core-2GB
[…]
cpt.2789663483500
```

CONCLUSION

Conclusion

- Linux kernel is already compiled and ready
- Application has been cross-compiled for ARM-32
- Disk image has been modified and now contains our application
- The `rcS` file automates the execution of our application
- A checkpoint has been taken to accelerate the simulation