Full system simulation in gem5

What is Full System Simulation?

Basics of Booting Up a Real System in gem5

Unlike in SE mode where we can just provide a binary, in FS mode we need to provide much more information to boot up a real system. In particular we need:

1. A disk image containing the operating system and any necessary software or data. This disk image serves as the virtual hard drive for the simulated system.

2. A kernel binary compatible with the simulated architecture is needed to boot the operating system.

Beyond these essentials, you might need to provide other files like a bootloader, depending on the complexity of the simulation.

Interacting with gem5 via m5term

In FS mode, gem5 provides a terminal interface to interact with the simulated system. This is useful for debugging and monitoring the system as it runs. It is just like connection to a real-world computer via a terminal.

Once the simulation is running gem5 will open a port on your host machine that you can connect to using a terminal emulator.

We provide `m5term` to connect to this port and interact with the simulated system. To build it:

```
cd gem5/util/term
make
```

Now you have a binary `m5term`. This accepts one argument: the port number to connect to.

Getting `m5term`'s port number

When you run a gem5 simulation in FS mode, gem5 will print the port number that `m5term` should connect to. This is usually the last line of the output.
Near the start of an FS mode simulation you'll see a line printed in the terminal output that specifies the port number. E.g.:

```
board.pc.com_1.device: Listening for connections on port 3456
```

Please note, there is also a similar statement for the GDB port. Make sure you are connecting to the correct port. (the GDB port statement is usually 7000 and started with `board.remote_gdb: Listening for...`)

An example of using

Creating disk images using Packer and QEMU

To create a generic Ubuntu disk image that we can use in gem5, we will use:

- Packer: This will automate the disk image creation process.
- QEMU: We will use a QEMU plugin in Packer to actually create the disk image.
- Ubuntu autoinstall: We will use autoinstall to automate the Ubuntu install process.

gem5 resources already has code that can create a generic Ubuntu image using the aforementioned method.

- Path to code: `gem5-resources/src/x86-ubuntu`

Let's go through the important parts of the creation process.

Getting the ISO and the user-data file

As we are using Ubuntu autoinstall, we need a live server install ISO.

- This can be found online from the Ubuntu website: iso

We also need the user-data file that will tell Ubuntu autoinstall how to install Ubuntu.

- The user-data file on gem5-resources specifies all default options with a minimal server installation.

How to get our own user-data file

To get a user-data file from scratch, you need to install Ubuntu on a machine.

- Post-installation, we can retrieve the `autoinstall-user-data` from `/var/log/installer/autoinstall-user-data` after the system's first reboot.

You can install Ubuntu on your own VM and get the user-data file.

Using QEMU to get the user-data file

We can also use QEMU to install Ubuntu and get the aforementioned file.

- First, we need to create an empty disk image in QEMU with the command: `qemu-img create -f raw ubuntu-22.04.2.raw 5G`
- Then we use QEMU to boot the diskimage:

```
qemu-system-x86_64 -m 2G \
      -cdrom ubuntu-22.04.2-live-server-amd64.iso \
      -boot d -drive file=ubuntu-22.04.2.raw,format=raw \
      -enable-kvm -cpu host -smp 2 -net nic \
      -net user,hostfwd=tcp::2222-:22
```

After installing Ubuntu, we can use ssh to get the user-data file.

Important parts of the Packer script

Let's go over the Packer file.

- **bootcommand**:

```
"e<wait>",
"<down><down><down>",
"<end><bs><bs><bs><bs><wait>",
"autoinstall  ds=nocloud-net\\;s=http://{{ .HTTPIP }}:{{ .HTTPPort }}/ ---<wait>",
"<f10><wait>"
```

  This boot command opens the GRUB menu to edit the boot command, then removes the `---` and adds the autoinstall command.

- **http_directory**: This directory points to the directory with the user-data file and an empty file named meta-data. These files are used to install Ubuntu.

Important parts of the Packer script (Conti.)

- **qemu_args**: We need to provide Packer with the QEMU arguments we will be using to boot the image.
  - For example, the QEMU command that the Packer script will use will be:

```
qemu-system-x86_64 -vnc 127.0.0.1:32 -m 8192M \
-device virtio-net,netdev=user.0 -cpu host \
-display none -boot c -smp 4 \
-drive file=<Path/to/image>,cache=writeback,discard=ignore,format=raw \
-machine type=pc,accel=kvm -netdev user,id=user.0,hostfwd=tcp::3873-:22
```

- **File provisioners**: These commands allow us to move files from the host machine to the QEMU image.
- **Shell provisioner**: This allows us to run bash scripts that can run the post installation commands.

Let's use the base Ubuntu image to create a disk image with the GAPBS benchmarks

Update the x86-ubuntu.pkr.hcl file.

The general structure of the Packer file would be the same but with a few key changes:

- We will now add an argument in the `source "qemu" "initialize"` block.
  - `diskimage = true` : This will let Packer know that we are using a base disk image and not an iso from which we will install Ubuntu.
- Remove the `http_directory = "http"` directory as we no longer need to use autoinstall.
- Change the `iso_checksum` and `iso_urls` to that of our base image.

  Let's get the base Ubuntu 24.04 image from gem5 resources and unzip it.

```
wget https://storage.googleapis.com/dist.gem5.org/dist/develop/images/x86/ubuntu-24-04/x86-ubuntu-24-04.gz
gzip -d x86-ubuntu-24-04.gz
```

`iso_checksum` is the `sha256sum` of the iso file that we are using. To get the `sha256sum` run the following in the linux terminal.

```
sha256sum ./x86-ubuntu-24-04.gz
```

- **Update the file and shell provisioners:** Let's remove the file provisioners as we dont need to transfer the files again.
- **Boot command:** As we are not installing Ubuntu, we can write the commands to login along with any other commands we need (e.g. setting up network or ssh). Let's update the boot command to login and enable network:

```
"<wait30>",
"gem5<enter><wait>",
"12345<enter><wait>",
"sudo mv /etc/netplan/50-cloud-init.yaml.bak /etc/netplan/50-cloud-init.yaml<enter><wait>",
"12345<enter><wait>",
"sudo netplan apply<enter><wait>",
"<wait>"
```

Changes to the post installation script

For this post installation script we need to get the dependencies and build the GAPBS benchmarks.

Add this to the post-installation.sh script

```
git clone https://github.com/sbeamer/gapbs
cd gapbs
make
```

Let's run the Packer script and use this disk image in gem5!

```
cd /workspaces/2024/materials/02-Using-gem5/07-full-system
x86-ubuntu-gapbs/build.sh
```

Let's use our built disk image in gem5

Let's add the md5sum and the path to our local JSON .

Let's run the gem5 GAPBS config.

```
GEM5_RESOURCE_JSON_APPEND=./completed/local-gapbs-resource.json gem5 x86-fs-gapbs-kvm-run.py
```

This script should run the bfs benchmark.

Let's see how we can access the terminal using m5term

- We are going to run the same gem5 GAPBS config but with a small change.

Let's change the last `yield True` to `yield False` so that the simulation doesn't exit and we can access the simulation.

```
def exit_event_handler():
    print("first exit event: Kernel booted")
    yield False
    print("second exit event: In after boot")
    yield False
    print("third exit event: After run script")
    yield False
```

Again, let's use m5term

Now let's connect to our simulation by using the `m5term` binary

```
m5term 3456
```