Full system simulation in gem5

What we will cover

- What is full system simulation?
- Basics of booting up a real system in gem5
- Creating disk images using packer and qemu
- Extending/modifying a gem5 disk image
- m5term to interact with a running system

What is Full System Simulation?

Full-system simulation is a type of simulation that emulates a complete computer system, including the CPU, memory, I/O devices, and system software like operating systems.

It allows for detailed analysis and debugging of hardware and software interactions.

Components Simulated:

- CPUs (multiple types and configurations)
- Memory hierarchy (caches, main memory)
- I/O devices (disk, network interfaces)
- Entire software stack (OS, drivers, applications)

Basics of Booting Up a Real System in gem5

Overview: gem5 can simulate the process of booting up a real system, providing insights into the behavior of the hardware and software during startup.

Steps Involved

1. **Setting Up the Simulation Environment**:
   - Choose the ISA (e.g., x86, ARM).
   - Configure the system components (CPU, memory, caches).
2. **Getting the correct resources such as kernel, bootloader, diskimages, etc.**
3. **Configuring the Boot Parameters**:
   - Set kernel command line parameters, if necessary.
4. **Running the Simulation**:
   - Start the simulation and monitor the boot process.

Let's run a full system simulation in gem5

The incomplete code already has a board built.

Let's run a full-system workload in gem5.

This workload is an Ubuntu 24.04 boot, it will throw three m5 exits at:

- Kernel Booted
- When `after_boot.sh` runs
- After run script runs

Obtain the workload and set exit event

To set the workload, we add the following to materials/02-Using-gem5/07-full-system/x86-fs-kvm-run.py:

```
workload = obtain_resource("x86-ubuntu-24.04-boot-with-systemd", resource_version="1.0.0")
board.set_workload(workload)
```

Obtain the workload and set exit event (conti.)

Let's make the exit event handler and set it in our simulator's object

```
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 True

simulator = Simulator(
    board=board,
    on_exit_event={
        ExitEvent.EXIT: exit_event_handler(),
    },
)
simulator.run()
```

Viewing the terminal/serial output with m5term

Before booting this workload, let's build the `m5term` application so we can connect to the running system.

```
cd /workspaces/2024/gem5/util/term
make
```

Now you have a binary `m5term`.

Watch gem5's output

Now, let's run the workload and connect to the terminal using `m5term`.

```
gem5 x86-fs-kvm-run.py
```

In another terminal, run the following command to connect to the terminal:

```
m5term 3456
```

3456 is the port number on which the terminal is running.
You will see this printed in the gem5 output.

If you run multiple gem5 instances, they will have sequential port numbers.
If you're running in a non-interactive environment, there will be no ports to connect to.

Creating your own disk images

Creating disk images using packer and qemu

To create a generic ubuntu diskimage that we can use in gem5, we will use:

- packer: This will automate the diskimage creation process.
- qemu: We will use qemu plugin in packer to actually create the diskimage.
- ubuntu autoinstall: We will use autoinstall to autmate the ubuntu install process.

gem5 resources already have code that can create the a generic ubuntu image using the above mentioned method.

- Path on codespaces: `/workspaces/2024/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 (switch link to 24.04)

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 then get the above mentioned file.

- First, we need to create an empty diskimage 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 autoinstall command.

- **http_directory**: This directory points to directory that has 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 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 diskimage with the gapbs benchmark

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 `source "qemu" "initialize"` block.
  - `diskimage = true` : This will let packer know that we are using a base diskimage 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 gen5 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 postinstallation 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 diskimage in gem5!

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

Let's use our built diskimage in gem5

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

Let's run the gem5 gapbs config

This script should run the bfs benchmark.

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

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 doesnt 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
```