Custom benchmarks

Outline

So far, we have used workloads from gem5-resources (which we'll cover a bit more in the next section).

In this section, we'll cover how to create or modify your own workloads.

Things we'll cover

- gem5-bridge (m5ops)
- Cross compiling

Region of interest

Not all of the code in an application is an important part of the workload. We might only be interested in a specific part of the code.

In gem5, we can mark a region of interest (ROI) using m5ops.

For example, let's look at the matrix multiply code we used in the previous section.

See `gem5-resources/src/matrix-multiply/matrix-multiply.c`.

Notice that most of this code is initialization and printing. The actual matrix multiplication is only a few lines.

```
const int size = 100;
int first[size][size], second[size][size], multiply[size][size];
printf("Populating the first and second matrix...\n");
for(int x=0; x<size; x++) {
    for(int y=0; y<size; y++) {
        first[x][y] = x + y;
        second[x][y] = (4 * x) + (7 * y);
    }
} printf("Done!\n");
printf("Multiplying the matrixes...\n");
for(int c=0; c<size; c++) {
    for(int d=0; d<size; d++) {
        int sum = 0;
        for(int k=0; k<size; k++) {
            sum += first[c][k] * second[k][d];
        }
        multiply[c][d] = sum;
    }
} printf("Done!\n");
printf("Calculating the sum of all elements in the matrix...\n");
long int sum = 0;
for(int x=0; x<size; x++)
    for(int y=0; y<size; y++)
        sum += multiply[x][y];
printf("Done\n");
printf("The sum is %ld\n", sum);
```

What we want to do

We would like to mark the matrix multiplication part as the region of interest.

This way, we can see the statistics of the matrix multiplication part only.

There are also other things we may want to communicate from the *guest* application to the simulator.

- Exit the simulation
- Take a checkpoint
- Where the simulation currently is (e.g., after Linux boot in FS mode)
- Many others

We can do this with gem5-bridge (m5ops)

What is m5ops

- The **m5ops** (short for m5 opcodes) provide different functionalities that can be used to communicate between the simulated workload and the simulator.
- The commonly used functionalities are below. More can be found in the m5ops documentation:
  - exit [delay]: Stop the simulation in delay nanoseconds
  - workbegin: Cause an exit event of type "workbegin" that can be used to mark the beginning of an ROI
  - workend: Cause and exit event of type "workend" that can be used to mark the ending of an ROI
  - resetstats [delay[period]]: Reset simulation statistics in delay nanoseconds; repeat this every period nanoseconds
  - dumpstats [delay[period]]: Save simulation statistics to a file in delay nanoseconds; repeat this every period nanoseconds
  - checkpoint [delay [period]]: Create a checkpoint in delay nanoseconds; repeat this every period nanoseconds
  - switchcpu: Cause an exit event of type, “switch cpu,” allowing the Python to switch to a different CPU model if desired

IMPORTANT

- ***Not all of the ops do what they say automatically***
- Most of these only exit the simulation
- For example:
  - exit: Actually exits
  - workbegin: Only exits if configured in `System`
  - workend: Only exits if configured in `System`
  - resetstats: Resets the stats
  - dumpstats: Dumps the stats
  - checkpoint: Only exits
  - switchcpu: Only exits
- See gem5/src/sim/pseudo_inst.cc for details.
- The gem5 standard library might have default behaviors for some of the m5ops. See src/python/gem5/simulate/simulator.py for the default behaviors.

> **Note**: gem5 version 25.0 will make some major updates to this API.

More about m5ops

There are three versions of m5ops:

1. Instruction mode: it only works with simulated CPU models
2. Address mode: it works with simulated CPU models and the KVM CPU (only supports Arm and X86)
3. Semihosting: it works with simulated CPU models and the Fast Model

Different modes should be used depending on the CPU type and ISA.

The address mode m5ops will be covered in 07-full-system as gem5-bridge and 08-accelerating-simulation after the KVM CPU is introduced.
In this session, we will only cover the instruction mode.

When to use m5ops

There are two main ways of using the m5ops:

1. Annotating workloads
2. Making gem5-bridge calls in disk images

In this session, we will focus on learning how to use the m5ops to annotate workloads.

How to use m5ops

m5ops provides a library of functions for different functionalities. All functions can be found in gem5/include/gem5/m5ops.h.
The commonly used functions (they are matched with the commonly used functionalities above):

- `void m5_exit(uint64_t ns_delay)`
- `void m5_work_begin(uint64_t workid, uint64_t threadid)`
- `void m5_work_end(uint64_t workid, uint64_t threadid)`
- `void m5_reset_stats(uint64_t ns_delay, uint64_t ns_period)`
- `void m5_dump_stats(uint64_t ns_delay, uint64_t ns_period)`
- `void m5_checkpoint(uint64_t ns_delay, uint64_t ns_period)`
- `void m5_switch_cpu(void)`

In order to call these functions in the workload, we will need to link the m5ops library to the workload. So first, we need to build the m5ops library.

Building the m5ops library

The m5 utility is in gem5/util/m5 directory.
In order to build the m5ops library,

1. `cd` into the `gem5/util/m5` directory
2. run `scons [{TARGET_ISA}.CROSS_COMPILE={TARGET_ISA CROSS COMPILER}] build/{TARGET_ISA}/out/m5`
3. the compiled library (`m5` is for command line utility, and `libm5.a` is a C library) will be at `gem5/util/m5/build/{TARGET_ISA}/out`

Notes

- If the host system ISA does not match with the target ISA, then we will need to use the cross-compiler.
- `TARGET_ISA` has to be in lower case.

> You'll need to do this in a minute

Controlling the simulation

Now, we have a way to communicate from the guest application to the simulator.
The next step is to take some action based on the state of the guest application.

Examples of actions we might want to take:

- Dump the statistics
- Reset the statistics
- Exit the simulation
- Take a checkpoint
- Switch the CPU model

The way gem5 allows you to do this is that the m5ops functions *exit* the simulation loop and return control back to the Python script.
The `Simulator` class in gem5 provides a way to override the default bahavior for different exit events.

Controlling behavior on simulation loop exits

The `Simulator` class in gem5 provides a way to override the behavior for different exit events.
It has a parameter, `on_exit_event`, that takes a dictionary of handlers for different exit events.

Each of these handlers can specify a set of actions to take when the simulation loop exits.
In other words, the first time the simulator exits for a reason you can take one action, and the second time another action.
These are specified as Python *generators*.

See `gem5/src/python/gem5/exit_event.py` for the list of exit events.

```
class ExitEvent(Enum):
    EXIT = "exit"  # A standard vanilla exit.
    WORKBEGIN = "workbegin"  # An exit because a ROI has been reached.
    WORKEND = "workend"  # An exit because a ROI has ended.
    ...
```

Types of behaviors

You can take many different actions when the simulator loop exits (all in the Python script). You can add any code here that you want.

There are many examples in `gem5/src/python/gem5/simulate/exit_event_generators.py`.

For example, you can dump the stats and reset the stats.

```
def dump_reset_generator():
    while True:
        m5.stats.dump()
        m5.stats.reset()
        yield False
```

This will dump the stats and reset the stats every time the simulation loop exits.
It returns `False` to indicate that the simulation should continue where it left off.

Exercise: Our own matrix multiply

In this exercise, we will update the matrix multiply code to mark the matrix multiplication part as the region of interest.
Then, we will compare the results with and without the ROI.

Questions

1. What percent of the total instructions are in the ROI?
2. What is the cache hit ratio in the ROI?
3. Can you reduce the total simulation time?

Step 1: build the m5ops library for x86

Go to the `gem5/util/m5` directory and build the m5ops library for x86.

Step 1: Answer

```
cd gem5/util/m5
scons build/x86/out/m5
```

> Note: although we are using Scons to build these, it's a different environment from building gem5 with different targets and options.
> This is a different build process than gem5. This is very confusing. I'm sorry.

Step 2: Modify the workload to have ROI annotations

Copy the matrix multiply code from gem5-resources/src/matrix-multiply/matrix-multiply.c to your own workload.

Use the `m5_work_begin` and `m5_work_end` functions to mark the matrix multiplication part as the region of interest.

You will also need to include **gem5/m5ops.h** in the workload's source file(s) (`<gem5/m5ops.h>`).

Step 2: Answer

Make sure to *not* include the print statements (these are not part of the ROI!).

The `(0,0)` in `m5_work_begin` and `m5_work_end` is the `workid` and `threadid`.
In our case, since we don't care about threads and we have a single work item, we can use `(0,0)`.

```
}
printf("Done!\n");
printf("Multiplying the matrixes...\n");
m5_work_begin(0, 0);
for(int c=0; c<size; c++) {
    for(int d=0; d<size; d++) {
        int sum = 0;
        for(int k=0; k<size; k++) {
            sum += first[c][k] * second[k][d];
        }
        multiply[c][d] = sum;
    }
}
m5_work_end(0, 0);
printf("Done!\n");
printf("Calculating the sum of all elements in the matrix...\n");
```

Step 3: Linking the m5ops library to C/C++ code

After building the m5ops library, we can link them to our workload by:

1. Adding **gem5/include** to the compiler's include search path (`-Igem5/include`)
2. Adding **gem5/util/m5/build/{TARGET_ISA}/out** to the linker search path (`-Lgem5/util/m5/build/{TARGET_ISA}/out`)
3. Linking against **libm5.a** with (`-lm5`)

Write a Makefile to compile the workload with the m5ops library.

Step 3: Answer

```
GXX = g++
GEM5_PATH ?= /workspaces/latin-america-2024/gem5

matrix-multiply:
    $(GXX) -o matrix-multiply matrix-multiply.c \
    -I$(GEM5_PATH)/include \
    -L$(GEM5_PATH)/util/m5/build/x86/out \
    -lm5
```

Note: If you're not running in codespaces, you will need to replace `/workspaces/latin-america-2024/gem5` with the path to your gem5 repository or override the `GEM5_PATH` variable.

Momentary aside

Try to run the `matrix-multiply` binary on the host machine.

What happens?

Answer

The binary will cause an "illegal instruction" error because the host machine does not recognize the m5ops instructions.
These are not "real" x86 instructions, but are instead instructions that are only recognized by gem5.

Step 4: Create a run script to use this workload

Now, we can no longer simply use `set_workload` and `obtain_resource` because we have a custom workload.
We'll see how to use custom workloads with the gem5 standard library in the next section.

For this example, use the `set_se_binary_workload` function in the `board` object to set up the workload.
See src/python/gem5/components/boards/se_binary_workload.py for details.

Use a `BinaryResource` object to pass the binary file to the `set_se_binary_workload` function.
See `src/python/gem5/resources/resource.py` for details.

Use a classic `PrivateL1SharedL2CacheHierarchy`, single channel DDR4 2400, a timing CPU, and a 3 GHz clock.

Step 4: Answer

```
cache_hierarchy = PrivateL1SharedL2CacheHierarchy(
    l1d_size="64kB", l1i_size="64kB", l2_size="1MB",
)
memory = SingleChannelDDR4_2400()
processor = SimpleProcessor(
    cpu_type=CPUTypes.TIMING,
    isa=ISA.X86,
    num_cores=1
)
board = SimpleBoard(
    clk_freq="3GHz",
    processor=processor,
    memory=memory,
    cache_hierarchy=cache_hierarchy,
)
```

```
board.set_se_binary_workload(
    binary = BinaryResource(
        local_path="matrix-multiply"
    )
)
simulator = Simulator(board=board)
simulator.run()
```

Step 5: Controlling the simulation

After running the simulation, you can see the results in the `stats.txt` file.
What do you notice that is different (hint: search for "Begin Simulation")?

See `_default_on_exit_dict` in the documentation for the `Simulator` class. (**Note:** the documentation is wrong.)

You'll see that the default behavior for `workbegin` and `workend` is to reset the stats and dump the stats, respectively.

You can also see this printed in the output of gem5.

Let's override this behavior. See the documentation on the Simulator for details.

For each exit, dump *and* reset the stats.

Step 5: Answer

```
def workbegin_handler():
    print("Workbegin handler")
    m5.stats.dump()
    m5.stats.reset()
    yield False

def workend_handler():
    print("Workend handler")
    m5.stats.dump()
    m5.stats.reset()
    yield False

simulator = Simulator(board=board,
    on_exit_event={
        ExitEvent.WORKBEGIN: workbegin_handler(),
        ExitEvent.WORKEND: workend_handler()
    }
)
```

The handlers are "generators" not functions (hence the `yield`).
We're yielding `False` to indicate that we don't want to exit the simulation.

When we constructor the simulator, we pass the `on_exit_event` parameter with a dictionary of the handlers.
By using generators, we can handle cases where multiple events happen.

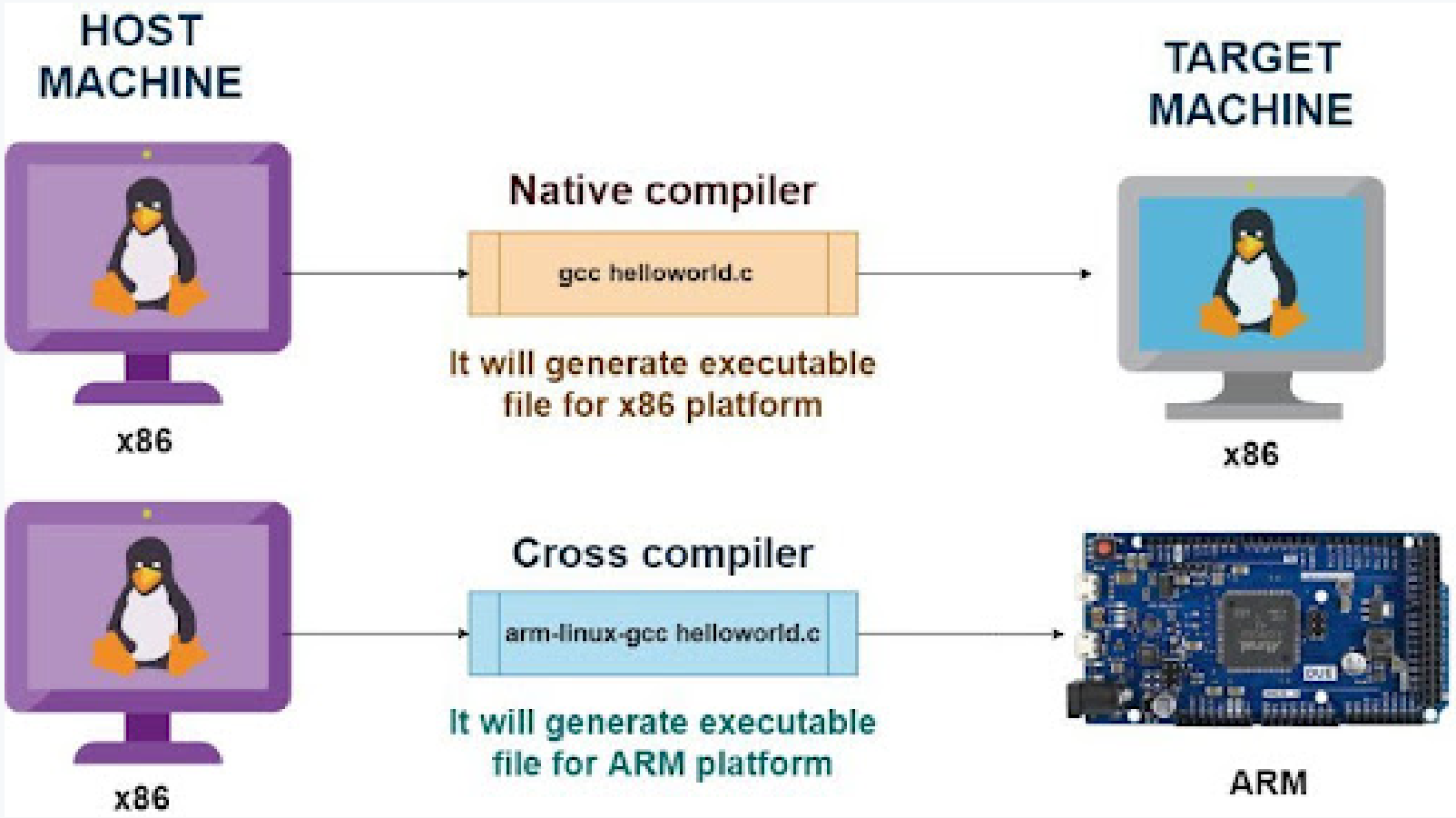
Exercise questions

1. What percent of the total instructions are in the ROI?
2. What is the cache hit ratio in the ROI?
3. Can you reduce the total simulation time?

(Note: Use the commit stats (not simulation stats) to calculate the results because there's a bug in resetting the sim stats.)

Cross-compiling

Cross-compiling from one ISA to another.



Goal: Run the same workload with the RISC-V ISA

We will cross-compile the workload to the RISC-V ISA and run it in gem5.

In the codespaces environment, we have installed a RISC-V and an Arm cross-compiler.

You can use `riscv64-linux-gnu-g++` for RISC-V and `aarch64-linux-gnu-g++` for Arm.

Exercise: Comparing RISC-V and X86

In this exercise, we will cross-compile the workload to the RISC-V ISA and run it in gem5.

Questions

1. What is the difference in the number of instructions executed between the RISC-V and x86 ISAs?
2. What is the difference in the cache hit ratio between the RISC-V and x86 ISAs?

Step 1: Cross-compile m5ops for RISC-V

Go to the `gem5/util/m5` directory and build the m5ops library for RISC-V.

You can use `riscv.CROSS_COMPILE=riscv64-linux-gnu-` with scons to cross compile the m5ops library for RISC-V.

Step 1: Answer

```
cd gem5/util/m5
scons riscv.CROSS_COMPILE=riscv64-linux-gnu- build/riscv/out/m5
```

Step 2: Build the workload for RISC-V

Modify the Makefile to build the workload for RISC-V.

Make sure to compile it statically (we'll see what happens with dynamic linking later).

Step 2: Answer

```
RISCV_GXX = riscv64-linux-gnu-g++

matrix-multiply-riscv:
    $(RISCV_GXX) -o matrix-multiply-riscv matrix-multiply.c \
    -I$(GEM5_PATH)/include \
    -L$(GEM5_PATH)/util/m5/build/riscv/out \
    -lm5 -static
```

Step 3: Run the workload in gem5

Update your script to take a parameter to either run the x86 or RISC-V workload.

Run the workload in gem5 with the RISC-V ISA and compare the results!

Questions

1. What is the difference in the number of instructions executed between the RISC-V and x86 ISAs?
2. What is the difference in the cache hit ratio between the RISC-V and x86 ISAs?

Example running with dynamic linking

Let's use the Arm ISA this time (mostly because there is a bug in RISC-V...)

```
matrix-multiply-arm: matrix-multiply.c
    $(ARM_GXX) -o matrix-multiply-arm matrix-multiply.c \
        -I$(GEM5_PATH)/include \
        -L$(GEM5_PATH)/util/m5/build/arm/out \
        -lm5
```

Build the m5 library:

```
scons build/arm64/out/m5
```

Fixing the problem with dynamic linking for different ISA

If you try to run your binary in gem5, you will see the following error:

```
src/base/loader/image_file_data.cc:105: fatal: fatal condition fd < 0 occurred: Failed to open file /lib/ld-linux-aarch64.so.1.
```

This is because gem5 is looking for that file on the host, but it's not in that location.

To fix this, you can redirect the library path in the configuration script.
Make sure to add this *before* calling `set_se_binary_workload`.

```
from m5.objects import RedirectPath
m5.core.setInterpDir("/usr/aarch64-linux-gnu/")
board.redirect_paths = [RedirectPath(app_path=f"/lib",
                         host_paths=[f"/usr/aarch64-linux-gnu/lib"])]
```

This will redirect the `/lib` path in gem5 to `/usr/aarch64-linux-gnu/lib` on the host.

Summary

SE mode does NOT implement many things!

- Filesystem
- Most system calls
- I/O devices
- Interrupts
- TLB misses
- Page table walks
- Context switches
- multiple threads
  - You may have a multithreaded execution, but there's no context switches & no spin locks