MultiSim

Multiprocessing support for gem5 simulations.

The problem

The gem5 simulator is single-threaded.

This is baked into the core design and is unlikely to change due to the high cost of converting the entire codebase.

Therefore, we cannot "speed up" your work with more cores and threads.

The insight

The gem5 simulator is used for experimentation.

Experimentation involves exploring how variables of interest change the behavior of a system when all other variables are held constant. As such, experimentation using the gem5 simulator requires multiple runs of the simulator.
Multiple instances of gem5 can be run in parallel.

If not a singular gem5 process utilizing multiple threads, why not multiple gem5 processes, each single threaded?

((This is really handy for us as we don't need to worry about the complexities of multi-threading: memory consistency, etc.))

People already do this... kind of...

Go to the 02-Using-gem5/11-multisim/01-multiprocessing-via-script directory to see a completed example of how **NOT** to run multiple gem5 processes.

This is typical but not recommended

Writing a script to run multiple gem5 processes:

1. Requires the user write the script.
    1. Increases the barrier to entry.
    2. Increases the likelihood of errors.
    3. Requires user to manage output files.
2. Non-standard (everyone does it differently).
    1. Hard to share with others.
    2. Hard to reproduce.
    3. No built-in support now or in the future.

A better way

MultiSim is a gem5 feature that allows users to run multiple gem5 processes from a single gem5 configuration script.

This script outlines the simulations to run.
The parent gem5 process (the process directly spawned by the user) spawns gem5 child processes, each capable of running these simulations.

Via the Python `multiprocessing` module, the parent gem5 process queues up simulations ("jobs"), for child gem5 processes ("workers") to execute.

Multisim has several advantages over simply writing a script to run multiple gem5 processes:

1. We (the gem5 devs) handle this for you.
    1. Less barrier to entry.
    2. Lower likelihood of errors.
    3. Multisim will handle the output files automatically.
2. Standardized.
    1. Easy to share with others (just send the script).
    2. Easy to reproduce (just run the script).
    3. Allows for future support (orchestration, etc).

Some caveats (it's new: be patient)

This features is new as of version 24.0.

It is not fully mature and still lacks tooling and library support which will allow for greater flexibility and ease of use.
However, this short tutorial should give you a good idea of how to use it going forward.

Let's go through an example

Start by opening materials/02-Using-gem5/11-multisim/02-multiprocessing-via-multisim/multisim-experiment.py.

This configuration script is almost identical to the script in the previous example but with the argparse code removed and the multisim import added:

To start: Declare the maximum number of processors

```
# Sets the maximum number of concurrent processes to be 2.
multisim.set_num_processes(2)
```

If this is not set gem5 will default to consume all available threads.
We **strongly** recommend setting this value to avoid overconsuming your system's resources.
Put this line near the top of your configuration script.

Use simple Python constructs to define multiple simulations

```
for data_cache_size in ["8kB","16kB"]:
    for instruction_cache_size in ["8kB","16kB"]:
        cache_hierarchy = PrivateL1CacheHierarchy(
            l1d_size=data_cache_size,
            l1i_size=instruction_cache_size,
        )
```

Replace the cache hierarchy in multisim-experiment.py with this and indent the code after the cache hierarchy so all of it is within the inner for loop (`for instruction_cache_size ...`).

Create and add the simulation to the MultiSim object

The key difference: The simulator object is passed to the MultiSim module via the `add_simulator` function.

The `run` function is not called here. Instead it is involved in the MultiSim module's execution.

```
multisim.add_simulator(
    Simulator(
        board=board,
        id=f"process_{data_cache_size}_{instruction_cache_size}"
    )
)
```

The `id` parameter is used to identify the simulation. Setting this is strongly encouraged. Each output directory will be named after the `id` parameter.

Execute multiple simulations

A completed example can be found at /materials/02-Using-gem5/11-multisim/completed/02-multiprocessing-via-multisim/multisim-experiment.py.

```
cd /workspaces/2024/materials/02-Using-gem5/11-multisim/completed/02-multiprocessing-via-multisim
gem5 -m gem5.utils.multisim multisim-experiment.py
```

Check the "m5out" directory to see the segregated output files for each simulation.

Execute single simulations from a MultiSim config

You can also execute a single simulation from a MultiSim configuration script.
To do so just pass the configuration script directly to gem5 (i.e., do not use `-m gem5.multisim multisim-experiment.py`).

To list the IDs of the simulations in a MultiSim configuration script:

```
gem5 {config} --list
```

To execute a single simulation, pass the ID:

```
gem5 {config} {id}
```