Interconnection Networks
ECE 8823 A / CS 8803 – ICN
Spring 2017
Lab 4: Cache Coherence Traffic

In this lab, you will study the impact of the NoC on the full-system. Inside gem5, you will boot up Linux on a multicore system with x86 cores and run applications from the PARSEC2.0 benchmark suite. You will study two cache coherence protocols, and vary the parameters of the NoC and study their impact on full-system runtime.

Step 0:
Clone the following copy of gem5
```
hg clone /nethome/tkrishna3/teaching/simulators/gem5_fs
```

On a fresh login to the machines, don't forget to source the environment file:
```
source <path_to_gem5>/my_scripts/set_env.sh
```

Build the X86 ISA and the MOESI_hammer protocol. This is a broadcast-based protocol, modeled after AMD's HyperTransport, with Private L1 + Private L2 per tile.
```
python `which scons` -j 16 build/X86_MOESI_hammer/gem5.fast PROTOCOL=MOESI_hammer
```

Build the X86 ISA and the MOESI_CMP_directory protocol. This is a directory-based protocol with a Private L1 per tile + Shared L2 slice per tile.
```
python `which scons` -j 16 build/X86_MOESI_CMP_directory/gem5.fast PROTOCOL=MOESI_CMP_directory
```

Step 1:
Run Command:
```
./build/X86_MOESI_hammer/gem5.fast \
--outdir my_STATS/X86_MOESI_hammer__blackscholes__router-pipe-stages-1 \
configs/example/fs.py \
--checkpoint-restore=1 \
--checkpoint-dir=/nethome/tkrishna3/gem5_fs_resources/checkpoints/x86-linux/64c-1GB/parsec_roi/simsmall/x86-linux_64c-1GB_blackscholes \
--work-end-exit-count=100 \
--ruby \
--restore-with-cpu timing \
--num-cpus=64 \
--num-dirs=4 \
--mem-size=1GB \
--num-l2caches=64 \
--l1d_size=32kB \
--l1i_size=32kB \
--l1d_assoc=4 \
--l1i_assoc=4 \
--l2_size=128kB \
--l2_assoc=8 \
--network=garnet2.0 \
--topology=MeshDirCorners \
--num-rows=8 \
--vcs-per-vnet=8 \
--num-pipe-stages=1
```

The parameters in blue are what you will be varying in this lab.

What the command does: You are restoring the program from a checkpoint taken right after the OS was booted on a 64-core system and the application started to run. You are specifying a system configuration (coherence protocol, cache sizes, network topology and configuration). You are then running work-end-exit-count (e.g., 100) number of *work-items* and measuring how long the application took to finish running these work items.

Benchmarks:

You will run the following 4 benchmarks:

`blackscholes`
`bodytrack`
`canneal`
`swaptions`

You can run them by just changing the name in the `checkpoint-dir`

Stats File:

`my_STATS/X86_MOESI_hammer__blackscholes__router-pipe-stages-1/stats.txt`

You can give any name to the output stats directory – just make sure it is unique for all your configurations and you are not overwriting other stats.

Each of your simulations will create its own folder in my_STATS, with the appropriate **`stats.txt`** file.

You need to read out the following 2 stats:

`sim_ticks` → This is the total application runtime (in cycles)

`system.ruby.network.average_packet_network_latency` → This is the average latency of the network packets

What to Run:

1. For all benchmarks, run **`X86_MOESI_CMP_directory`** with `--num-pipe-stages=1`
2. For all benchmarks, run **`X86_MOESI_CMP_directory`** with `--num-pipe-stages=5`
3. For all benchmarks, run **`X86_MOESI_hammer`** with `--num-pipe-stages=1`
4. For all benchmarks, run **`X86_MOESI_hammer`** with `--num-pipe-stages=5`

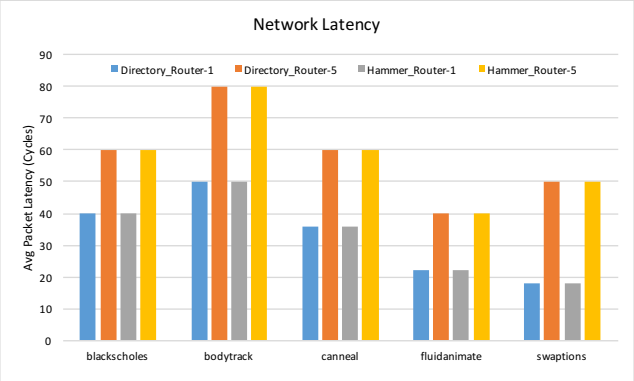
NOTE: FULL-SYSTEM SIMULATIONS CAN TAKE MULTIPLE MINUTES TO HOURS. We would recommend launching them in parallel and waiting till they finish.

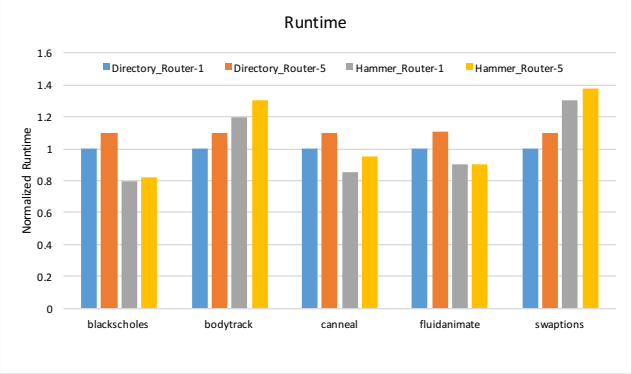
What to Plot:

1. Plot the *absolute network packet latency* (in cycles) for each benchmark, for the 4 configurations in a bar graph.

2. Plot the *normalized application runtime* for each benchmark, for the 4 configurations. Each benchmark bar should be normalized to its runtime with **`X86_MOESI_CMP_directory`** with `--num-pipe-stages=1`. Thus for all benchmarks, the first bar will be 1, and the remaining will be relative to it.

For instance, following are examples of the kind of graphs you should plot.





Add both the plots in a Report.

Answer the following questions in the report:

- What is the average network delay (in cycles) with 1-cycle routers for MOESI_CMP_directory across all benchmarks?
- What is the average network delay (in cycles) with 1-cycle routers for MOESI_hammer across all benchmarks?
- For MOESI_CMP_directory, what is the % reduction in runtime for each benchmark when going from a 5-cycle router per hop to 1-cycle router per hop?
  [blackscholes: XX %, bodytrack: XX%, …]
- For MOESI_hammer, what is the % reduction in runtime for each benchmark when going from a 5-cycle router per hop to 1-cycle router per hop?
  [blackscholes: XX %, bodytrack: XX%, …]
- Which protocol (Hammer or Directory) has lower runtime across most benchmarks? Can you think why based on the description in Step 0?

What to Submit:

Report.pdf