SPEC HPC Gem5 Benchmarking

`authors:`

- Mitchell Arndt: arndt20@purdue.edu
- Abhinav Goel: geol39@purdue.edu

This project was aimed to bring the SPEC HPC benchmarks into a gem5 simulation testing environment to characterize each of the included workloads.

Project Setup

Gem5

Use this step if you are building on your own machine. Make sure to install all needed dependencies described here

1. clone gem5 repo: `git clone https://gem5.googlesource.com/public/gem5`
2. build gem5:
    1. `cd gem5`
    2. `scons build/X86/gem5.opt -j 16`
        1. this command uses the following format: `scons build/{ISA}/gem5.{variant} -j {cpus}`
        2. On the first time you build, you will be prompted with a message about `missing the gem5 style or commit message` hook. Press enter to agree and continue the build.
3. build m5 utility
    1. `cd gem5/util/m5`
    2. `scons build/x86/out/m5`

> Note: this process will take quite a bit of time

Building Disk Image

> Note: this stage must be completed after the gem5 build above

1. Download Packer Utility
    1. `cd spec-hpc_tests/disk-image/`
    2. `wget https://releases.hashicorp.com/packer/1.4.3/packer_1.4.3_linux_amd64.zip`
    3. `unzip packer_1.4.3_linux_amd64.zip`
    4. you now have access to the `./packer` binary
2. Validate build image config
    1. `./packer validate spec-hpc/spec-hpc.json`
3. Build image

        1. `./packer build spec-hpc/spec-hpc.json`
    4. As a sanity check, verify the disk image size (in kB)
        1. `du -s --apparent-size spec-hpc/spec-hpc-image/spec-hpc`
        2. The `spec-hpc.json` script specifies {"image_size": "30000"} which means you should expect ~30GB for the image size
            1. Should you want to reduce the size of the disk image, you can change this value in the spec-hpc.json file and rebuild the image. It is recommended that you go no lower than 10GB ({"image_size" : "10000"}) to hold the OS and have space for benchmark simulation.

Environment Config

1. ensure that the `/proc/sys/kernel/perf_event_paranoid` variable is set to 1
    1. This is required by the gem5 simulator to run full system simulations

Makefiles

This section will discuss the various Makefiles in this project that are useful for running benchmarks and general debugging

`spec-hpc-tests/Makefile`

- Makefile targets:
    - all: responsible for running the benchmark tests
        - usage: `make benchmark={cloverleaf, tealeaf} cores={number of processors} cache={cache size}`
    - pull_archive: download the results from our testing in a github release
        - usage: `make pull_archive version={GitHub Release Version}`

`spec-hpc-tests/disk-image/spec-hpc/Makefile`

- Makefile targets:
    - all: prints available options (see below)
    - mount: mounts disk image to filesystem
        - usage: `make mount`
        - Note: you will need sudo permissions to do this
    - unmount: unmounts disk image from the filesystem
        - usage: `make unmount`
        - Note: you will need sudo permissions to do this

`spec-hpc-tests/disk-image/spec-hpc/remotes/Makefile` This Makefile is used by the x86-spec-hpc.py script to build and run the benchmarks inside the gem5 simulation.

- Makefile targets:
    - {benchmark}_build: contains commands to build the benchmark
        - usage: `make {benchmark}_build`
    - {benchmark}_run: contains the commands to run the benchmark

        - usage: `make {benchmark}_run CORES={cores}`
    - {benchmark}_cleans: cleans temp/build files of benchmark
        - usage: `make {benchmark}_clean`

> Note: This Makefile allows for abstraction of each individual benchmarks into one run script. To add more of the SPEC hpc benchmarks, add the source code into remotes and fill in the necessary commands in this Makefile for building and running the benchmark from the source.

Run Tests

Tests are conducted by using the `Makefile` in the root directory of this project.

Example:

```
make benchmark=cloverleaf cores=4 cache=512kB
```

> See Makefiles Section for more details

Parsing Results

After a benchmark simulation completes, a set of files will be generated: `archive/{benchmark}-p{cores}-{cache size}-lock`. Note that the `-lock` keyword indicates that the simulation has completed and the lack of this keyword indicates the benchmark is still running.

To generate plots for the simulation data run the following:

`./parse_stats.py`

> IMPORTANT: This script expects that you have run to completion a subset of tests specified in Required Runs
>
> ```
> for b in benchmark:
>   for p in [1, 2, 4, 8]: #processors
>     # run with cache size = 32kB
>   for c in ["8kB", "32kB", "128kB", "512kB", "2048kB"]:
>     # run with cores = 4
> ```

This script will generate the following plots in the `parse_out/` directory:

- `speedup.png`
    - Shows the parallel speedup achieved over the sweep of 1, 2, 4, 8 processors keeping the cache size constant
    - This is measured from taking execution time of the simulated processor with multi-threads vs. a single thread execution
- `cl_working_set.png`

    - Shows the working set of the cloverleaf benchmark over a sweep of cache sizes holding the number of processors constant
    - This is measured by dividing the cache misses / cache accesses
- `tl_working_set.png`
    - Shows the working set of the tealeaf benchmark over a sweep of cache sizes holding the number of processors constant
    - This is measured by dividing the cache misses / cache accesses
- `miss_rate.png`
    - Shows the miss rates of the L1 cache over a sweep of cache sizes while holding the number of processors constant
    - This is measured by dividing the cache misses / cache accesses
- `off-chip-traffic.png`
    - Shows the load/store requests from each processor over a sweep of cache sizes holding the processor constant
    - This is measured by dividing the loads/stores of the `L2 cache` by the total number of instructions executed
- `on-chip-traffic.png`
    - Shows the reads/writes of a processor over a sweep of cache sizes holding the processor constant
    - This is measured by dividing the loads/stores of the `L1 cache` by the total number of instructions executed

Runs Required

```
for b in benchmark:
  for p in [1, 2, 4, 8]: #processors
    # run with cache size = 32kB

    # These runs provide:
    #   parallel speedup data
    #   on-chip traffic data (bytes/instr)
    #   off_chip traffic data (bytes/instr)
  for c in ["8kB", "32kB", "128kB", "512kB", "2048kB"]:
    # run with cores = 4

    # These runs provide:
    #   benchmark working set data
    #   aggregate miss rate data
```

Debugging Tips

Over the course of this project, there were may issue that arose that weren't trivially solvable. The following is a list of some helpful information/tips that were useful at getting the simulations to run.

Mounting Disk Image

> Note: We have created a Makefile command that allows you to mount directly to the disk image that is created from this exact configuration. Should you need to change it, this information may be useful.

To mount a partition inside the disk image you need to calculate the offset of where the partition starts.

- `cd spec-hpc-tests/disk-image/spec-hpc/spec-hpc-image`
- `parted spec-hpc`
    - `unit`
    - `B`
    - `print`
    - q to exit parted menu

The output should look something like this:

```
bash$ parted spec-hpc
GNU Parted 2.3
Using picked.img
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) unit
Unit? [compact]? B
(parted) print
Model:  (file)
Disk /home/spec-hpc-tests/disk-image/spec-hpc/spec-hpc-image/spec-hpc: 31457280000B
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start     End           Size          Type     File system  Flags
1      1048576B  31456231423B  31455182848B  primary  ext4         boot
```

- Next create a folder where you can mount the drive
    - `mkdir spec-mnt`
- mount drive to the partition given from parted
    - `sudo mount -o loop,offset=1048576 spec-hpc spec-mnt`

References for disk mounting:

- check disk size
- find partition

Tmux

Simulations on gem5 take long. Really long actually. This is a useful tool for creating a shell that will persist when a ssh session is closed over the course of your multi-day simulation run.

- create new tmux window session

    - `tmux`
- detach window:
    - `ctrl` + `b` then `d`
- kill window:
    - `ctrl` + `b` then `&` then confirm with `y`
- name window:
    - `ctrl` + `b` then `$` then type `custom_name`
- list sessions
    - `tmux ls`
- attach to tmux terminal
    - `tmux attach`
    - `tmux attach -t <session name>`

References

- this disk image installation was based on the spec hpc quick start tutorial
- m5 simulation commands