Building and Running gem5 with ALPHA_FS (full system) Mode

(Written by kyungsu.kang@gmail.com)

This tutorial introduces how to building and running Gem5 with ALPHA_FS mode. Gem5 is a full system multiprocessor simulator that is capable of booting a full Linux system.

Section 1. gem5 basic set-up procedure

1. Preparing Gem5 for External Tool Dependency

Please, refer to http://www.m5sim.org/Dependencies and install the tools that you need.

2. Compiling Gem5 for ALPHA_FS Mode

a. Get gem5
   i. % hg clone http://repo.gem5.org/gem5
   ii. % cd gem5
b. Make ALPHA_FS configuration file on the directory "build_opts"
   i. % cd build_opts
   ii. % cp ALPHA ALPHA_FS
   iii. Modify ALPHA_FS by inserting the below line anywhere;
   FULL_SYSTEM = 1
   iv. % cd ..
c. Build gem5 (This will take a few tens minute.)
   i. % scons build/ALPHA_FS/gem5.opt
   ii. More details about build gem5, refer to http://www.m5sim.org/Build_System

2. Installing full system files

a. Download the full system files; most the default full system files don't work well with SPLASH-Download the ones that work for Parsec for M5 and use them instead. Get tsb_osfpal from the obsolete full system files if you want to run more than 8 processors.
   i. Download gem5_fullsystem.tar.gz from http://bigmail.mail.daum.net/Mail-bin/bigfile_down?uid=dYBLARPhzvuEUBVa9e7hszxxaCjSQbUY)
   ii. % tar xvfz gem5_fullsystem.tar.gz
b. Edit environment path; SysPaths.py, Benchmarks.py, FSConfig.py
   i. % cd configs/common
   ii. Modify SysPaths.py by Path = ['/dist/m5/system', '/home/kang/gem5/dist']
   iii. Modify Benchmarks.py by disk('linux-parsec-2-1-m5.img')
   iv. Modify FSConfig.py by binary('tsb_osfpal')
   v. % cd ../..

3. Running Gem5

a. % ./build/ALPHA_FS/gem5.opt ./config/example/fs.py
   If you can see the message below from the system.terminal file, the booting is successful.
   Script from M5 read file is empty, starting bash shell...
   #
   For details, refer to $GEM5/m5out/system.terminal to monitor the process booting the linux kernel.

4. Using M5Term to interact with the Gem5 simulation system

a. % cd $GEM5/util/term
b. % make
c. % make install
d. % m5term localhost 3456
   Now you can successfully communicate with the simulated Gem5 system.

Section 2. Running My Own Application in Gem5 with ALPHA_FS Mode

1. Obtaining cross-compile tools for ALPHA ISA

a. Download cross-compile tools (e.g., gcc-4.3.2, glibc-2.6.1 (NPTL,x86/64)) from http://www.m5sim.org/Download
b. mkdir CROSSCOMPILE
c. % cd CROSSCOMPILE
d. % cp /home/kang/Downloads/alphaev67-unknown-linux-gnu.tar.bz2 ./
e. % tar xfvj alphaev67-unknown-linux-gnu.tar.bz2

2. Compiling each program (e.g., parallel integral image kernel with pthread)

a. Download scan.c from http://blog.daum.net/yjnks/92) to $GEM5/tests/test-progs/scan/
b. % cd ./tests/test-progs
c. % mkdir scan
d. % cd scan
e. % cp /home/kang/Downloads/scan.c ./
f. % $CROSSCOMPILE/alphaev67-unknown-linux-gnu/bin/alphaev67-unknown-linux-gnu-cc –o scan.out scan.c –lpthread –static

3. **Loading the application binaries to the pre-compiled OS image** (You can also refer to http://gem5.org/Frequently_Asked_Questions#How_do_I_add_files_to_a_disk_image.3F)

a. % cd /home/kang/gem5
b. % mkdir MOUNT
c. % /bin/mount -o loop,offset=32256 ./dist/disks/linux-parsec-2-1-m5.img ./MOUNT
d. % cd ./MOUNT/benchmarks
e. % mkdir scan
f. % cd scan
g. % cp /home/kang/gem5/tests/test-progs/scan/scan.out ./
h. % cd /home/kang/gem5
i. % /bin/umount ./MOUNT

4. Running your own application on Gem5 with M5Term

a. % ./build/ALPHA_FS/gem5.opt ./config/example/fs.py
b. % m5term localhost 3456 (executing this on the other terminal)
c. % cd /benchmarks/scan/scan.out
Now you can successfully see the simulation results.

Section 3. Running SPLASH-2 in Gem5 with ALPHA_FS Mode

1. **Obtaining SPLASH-2 benchmark**
   a. Get SPLASH-2; the original website has been down since late 2009, so try downloading from the wayback machine. When you untar the file it may say "unexpected end of file", but that is OK.
      i. % mkdir SPLASH-2
      ii. % cd SPLASH-2
      iii. wget http://web.archive.org/web/20080528165352/http://www-flash.stanford.edu/apps/SPLASH/splash2.tar.gz
      iv. % tar xvfz splash2.tar.gz
   b. Patch SPLASH-2
      i. % cd splash2
      ii. wget http://www.capsl.udel.edu/splash/splash2-modified.patch.gz
      iii. gzip –d splash2-modified.patch.gz
      iv. patch –p1 < splash2-modified.patch
   c. Test SPLASH-2
      i. % cd codes
      ii. In codes/Makefile.config change the BASEDIR to where you downloaded it, and on line 9 change the macros to "c.m4.null.POSIX" to support parallelism.
      iii. % cd kernels/fft
      iv. % make
      v. % ./FFT -t

2. **Compiling SPLASH-2 benchmark**
   a. Modifying CC, CFLAGS, and LDFLAGS in codes/Makefile.config
      i. Change CC as
      CC:= /home/kang/gem5/CROSSCOMPILE/alphaev67-unknown-linux-gnu/bin/alphaev67-unknown-linux-gnu-gcc
      ii. Insert two CFLAGs next to the last CFLAG as
      CFLAGS := $(CFLAGS) –I/home/kang/gem5/CROSSCOMPILE/alphaev67-unknown-linux-gnu/alphaev67-unknown-linux-gnu/sys-root/usr/include
      CFLAGS := $(CFLAGS) –static –static-libgcc
      iii. Insert LDFLAG next to the last LDFLAG as
      LDFLAGS = $(LDFLAGS) –L/home/kang/gem5/CROSSCOMPILE/alphaev67-unknown-linux-gnu/alphaev67-unknown-linux-gnu/lib
      iv. % cd ./kernels/fft
      v. % make

3. **Loading the application binaries to the pre-compiled OS image** (You can also refer to http://gem5.org/Frequently_Asked_Questions#How_do_I_add_files_to_a_disk_image.3F)
   a. Copy FFT onto the dist image gem5 will boot
      i. % cd /home/kang/gem5
      ii. % /bin/mount -o loop,offset=32256 ./dist/disks/linux-parsec-2-1-m5.img ./MOUNT
      iii. % mkdir ./MOUNT/benchmarks/splash2
      iv. % cp ./SPLASH-2/splash2/codes/kernels/fft/FFT ./MOUNT/benchmarks/splash2
      v. % /bin/umount ./MOUNT
   b. Make rcS script file at ./configs/boot/fft.rcS
      i. Make a file, fft.rcS
      #!/bin/sh
      cd benchmarks/splash2
      echo "Running FFT now…"
      ./FFT –t –p1
      #Gracefully exit gem5
      /sbin/m5 exit
      ii. Add fft benchmark to configs/common/Benchmarks.py as
      'fft': [SysConfig('fft.rcS', '512MB')],

c. Run gem5
   i. % cd /home/kang/gem5
   ii. % ./build/ALPHA_FS/gem5.opt ./configs/example/fs.py –n 1 –b fft
   iii. % m5term localhost 3456 (executing this on the other terminal)

ETC.

Run (almost) all kernels and applications in M5

All the kernels run in M5 without any modification and report that they passed a self test (run the command with the -t parameter):
fft, lu, radix, cholesky

raytrace: Runs, but the program to check the output does not work.
ocean: Runs and the output matches correct.out within rounding tolerances.
radiosity: Significant differences between correct.out and M5's output, need to get XGL or YGL to verify the output.
water-*: Runs, but no way to verify results.
fmm: If it doesn't compile try renaming the `_Complex` type and using `__DBL_DIG__` and `__DBL_MAX__`. The output doesn't match correct.out.
barnes: Runs, but no option to produce output to verify results.
volrend: This application depends on libtiff, which is a challenge to compile. To compile libtiff first remove the `-ansi` flag in the `Makefile`. In `tiffcompat.h` comment out lines 173-187 (the malloc precompiler directives) and replace with just `#include <malloc.h>`. Remember we need to build for alpha, so add `include ../../../Makefile.config` to get all our alpha cross compiler settings. However, during the build libtiff actually compiles a program, runs it, and compiles the output. Obviously this won't work if we compile it for alpha, so we need to compile just that program for x86. Under the target `g3states.h:` change the `${CC}` to just `gcc`, or whatever x86 compiler you use. Now type `make all` to compile libtiff. Go up a directory and volrend will make without problems.
raytrace: Use libtiff from volrend to compile rltotiff. If it doesn't work try commenting out the `BYTESWAP` in `tiff_rgba_io.c`.
Other Notes:
If you get the error "`panic: Need to implement cache resending nacked packets!`" for anything over 16 cores see this email. However, if you change the buffer in `src/mem/Bridge.py` it won't have any affect. Instead add these lines to your `fs.py` (or wherever you are configuring the system you simulate).

```
self.bridge.req_size_a = 128
self.bridge.req_size_b = 128
self.bridge.resp_size_b = 128
self.bridge.resp_size_b = 128
```

Contact

Philip Jagielski at gmail
Note: do not ask me to share the document with you. I have been getting a large amount of spam using the “Request to share this document” vector so I will not respond. Please email me directly.