Sniper hands-on

Install

- Sniper site: http://snipersim.org/
- Sniper support group: https://groups.google.com/forum/#!forum/snipersim
- OS: Ubuntu
- Required root permissions
- Git repository was received by mail but still seems to be good.
- Pin need to be downloaded separately – only pin3.7 is working with sniper right now

Install instructions

```
mkdir downloads
cd downloads/
wget https://software.intel.com/sites/landingpage/pintool/downloads/pin-3.7-97619-g0d0c92f4f-gcc-linux.tar.gz &
tar xvzf pin-3.7-97619-g0d0c92f4f-gcc-linux.tar.gz
cd ..
git clone http://snipersim.org/download/bfd8ed6e5a50b1e2/git/sniper.git
mv downloads/pin-3.7-97619-g0d0c92f4f-gcc-linux sniper/pin_kit
sudo apt-get install libc6-dev-i386
g++ --version   to get your_g++_version
sudo apt-get install g++-your_g++_version-multilib   replace with g++ version
sudo apt-get install lib32z1-dev
sudo apt-get install libbz2-dev
sudo apt-get install libboost-dev
sudo apt-get install libsqlite3-dev
make

./run-sniper -c gainestown -c base -- /bin/ls                test run
```

Files

- default performance model is “rob performance model” – model of one core, with OOO.
  - You can find all files under: common/performance_model/performance_models/rob_performance_model/
- config files: config/
  - basic files: are rob.cfg with gainestown.cfg
- PIN FE – sift/recorder/recorder_base.cc
  - in PIN FE you can extract real data on the instruction pre simulation and save it to be used later on simulation. In example save destination value / number of registers…
- common/performance_model/performance_models/rob_performance_model/rob_timer.cc
  - out of order operations on uops
- common/performance_model/performance_models/micro_op_performance_model.cc
  - micro ops creation and treatment

Files (continue)

- common/trace_frontend/trace_thread.cc : void TraceThread::handleInstructionDetailed(Sift::Instruction &inst, Sift::Instruction &next_inst, PerformanceModel *prfmdl) Decoding instruction

Debug

- FE (PIN) debug could be using --gdb flag to start gdb shell allowing to debug changes made in FE side of sniper.
- BE – debug simulation by printing out relevant information, simply as “std::cout”

config tweaks

- add config options:
  - common/misc/config.cc
  - in example:

    adding m_UOP_debug = Sim()->getCfg()->getBool("uopcache/debug");

    and then bool getUOPdebug() const { return m_UOP_debug; }

    so it could be used in your files to enable/disable DEBUG prints:

    ```
    if ( Sim()->getConfig()->getUOPdebug()) std::cout << “DEBUG …… “
    ```

  - then in config file it should be added:

    [uopcache]

    debug=true
  - it could be also be added in command line: -c uopcache/debug=true

stats

- sim.out file contain simulations details
  - number of instructions
  - cycles
  - cache data
  - IPC and more
- ./tools/dumpstats.py > stats – generating stats collected in simulation:
  - detail stats on caches /performance models/ rob / uops and more.

stats tweaks

- add stats to be collected,
  - include #include "stats.h“ in your file.
  - create variable to collect your stats
  - register your variable: registerStatsMetric("uopcache", 0 , "uopcache_hits", &uopcache_hits);
  - then it will be shown in dumpstats.py results.