PARADE is a cycle-accurate full-system simulation platform for accelerator-rich architectural design and exploration. It extends the widely used gem5 simulator with high-level synthesis (HLS) support.

If you use PARADE in your research, please cite our ICCAD 15 paper:
Jason Cong, Zhenman Fang, Michael Gill, Glenn Reinman. PARADE: A Cycle-Accurate Full-System Simulation Platform for Accelerator-Rich Architectural Design and Exploration. IEEE/ACM International Conference on Computer-Aided Design (ICCAD 2015). Austin TX, USA, November 2015.

A pdf version of this manual is also available in doc/parade_manual.pdf.

Setup Requirements:

1. Currently PARADE has been tested on two 64-bit platforms:
   - CentOS release 6.6 (Final) and CentOS release 6.5 (Final)

2. Tools to build PARADE (mostly required by gem5):
   - GCC-4.8.0 to build PARADE
   - Python 2.6.6
   - Scons-1.3.1
   - Swig-2.0.9
   - Protobuf-2.5.0

3. GCC-4.1.2 to build benchmarks:
   - Note the Linux OS that PARADE simulates needs an older GCC version

4. Mono-2.10.9 to execute C# programs:
   - Our automatic accelerator and application generators are written in C#
   - Some post-processing tools are also written in C#

Build PARADE:

Let’s assume you are in the home directory ($PARADE_HOME) of the checked out PARADE code.

1. Build CPU version:
   - Disable the SIM_DEDICATED_ARA macro in all three files, src/mem/ruby/profiler/Profiler.cc,

src/mem/ruby/system/System.cc, and src/sim/simulate.cc

- ./build.gem5.sh --- note that current PARADE release disabled visualization support though the cache protocol name is "MESI_Two_Level_Trace"
- cp buid/X86/gem5.opt parade-test/ --- backup the built CPU version

2. Build ARA version:
   - Enable the SIM_DEDICATED_ARA macro in all three files, src/mem/ruby/profiler/Profiler.cc, src/mem/ruby/system/System.cc, and src/sim/simulate.cc
   - In current PARADE release, users have to manually configure which accelerators to enable in the source code: RubySystem::startup() function in src/mem/ruby/system/System.cc. For example, if you want to enable only the BlackScholes accelerator, just uncomment the AddOperatingMode lines for BlackScholes and comment all other benchmarks.
   - ./build.gem5.sh
   - cp buid/X86/gem5.opt parade-test/TDLCA_BlackScholes

Build Benchmarks:

Before you start, please download the disk image and Linux binaries to $PARADE_HOME from: http://vast.cs.ucla.edu/software/parade-ara-simulator, and export M5_PATH=$PARADE_HOME. We have prebuilt all the benchmarks used in the ICCAD 15 paper.

If you want to build the benchmarks by your own:

- cd benchmarks; make --- note it needs an older GCC version, e.g., GCC-4.1.2
- mount the disk image (needs root), just check mount.sh and adapt to your own path
- copy the compiled binary to the corresponding benchmark directory of the mounted image
- umount the disk image

Run Benchmarks on PARADE:

We use the well-known three-stage simulation methodology to reduce the simulation time:

- Stage 1, checkpoint the OS booting using atomic mode.
- Stage 2, warmup the cache memory system for application initialization using timing mode.
- Stage 3, simulate the region of interests (ROI, i.e., accelerator part) using detailed mode.

Let's still use the BlackScholes example for demonstration.

1. Checkpoint:
   - ./run.checkpoint.sh
   - cp -r x86-out/cpt.5176168078500 parade-test/ckpt-1core/ --- backup the checkpoint

2. Warmup:
   - Edit run.dedicated.ara.begin.sh, comment all benchmarks except BlackScholes
   - ./run.dedicated.ara.begin.sh
   - ./backup.sh TDLCA begin --- backup the simulation results

3. ROI simulation:
   - Edit run.dedicated.ara.ooo.sh, comment all benchmarks except BlackScholes, note that there are three places to comment: BENCHS, NUM_ACCELERATORS, and WARMUP
   - vi parade-TDLCA-begin/TDLCA_BlackScholes/stats.txt, get the number of "sim_insts" and update it to the WARMUP in run.dedicated.ara.ooo.sh
   - ./run.dedicated.ara.ooo.sh
   - ./backup.sh TDLCA ooo

4. Change benchmark input size:
   - edit configs/boot/BlackScholes.td.rcS
   - Currently we wrap up the benchmarks running using a multi-program wrapper (bench.out in the simulated disk image), so that we can easily support multi-program simulation. Users only need to type the command (in the .rcS file) to the wrapper.
   - For example, "TDLCA_BlackScholes 0 262144" means it runs TDLCA_BlackScholes on core 0 with input size as 262144. You can type multiple commands (separate each command use \n) to run multi-program. For example "TDLCA_BlackScholes 0 262144\n TDLCA_BlackScholes 1 262144"

5. Change simulation configuration:
   - For ARA, we add –lcacc, and users can specify how many accelerators are used with the flag --accelerators=1
   - All other configurations are the same as the original gem5, please check our ISCA 15 tutorial and the gem5 website

Interpret Simulation Results:

1. Overall performance:
    - ./get.stats.sh TDLCA ooo (for CPU, use SW ooo)

2. Breakdown performance:
    - ./get.acc.stats.sh TDLCA ooo for ARA
    - mono ARAResultParser.exe parade-TDLCA-ooo/TDLCA_BlackScholes/result.txt for ARA computation/read/write breakdown, it will generate a .csv file
    - ./get.sw.breakdown.sh SW ooo for SW

3. Power breakdown:
    - For CPU core and cache, ./generate.mcpat.xml.sh TDLCA ooo and then ./generate.mcpat.energy.sh TDLCA ooo
    - For NoC, ./generate.dsent.sh TDLCA ooo
    - For DRAM, it's integrated with gem5, in stats.txt (you don't have to do anything)
    - Finally, get all results together: ./get.energy.stats.sh TDLCA ooo

Further Reading:

For high-level overview of PARADE, please read our ICCAD 15 paper and ISCA 15 tutorial slides (http://accelerator.eecs.harvard.edu/isca15tutorial/slides/isca2015-tutorial-parade.pdf).

Our ISCA 15 tutorial also introduces how to add users' own accelerator and applications that use those accelerators. (Check the benchmarks/ProgramGeneration directory, it has the .type file needed for accelerator generation, and .txt file for application generation, and buildProgs.sh)

Now you are ready to go. Hack the code and have fun!

Some Claims:

We would like to thank the open source gem5 community and all Center for Domain-Specific Computing (CDSC) members at UCLA who contribute to PARADE.

Current PARADE release mainly supports the simulation of dedicated ARA as presented in the ICCAD 15 paper. We didn't release the composable ARA version or the visualization tool since they are still under testing. In addition, we cannot open source the AutoPilot high-level synthesis (HLS) tool due to the license issue. But we release all the accelerators used in the ICCAD 15 paper. Users can use their own tools to get the detailed timing info such as pipeline II, clock, area, and power. Once these numbers are given, users can integrate their own accelerators into PARADE for system-level design and evaluation.

Zhenman Fang, Postdoctoral Researcher
Center for Domain-Specific Computing, UCLA
Email: zhenman@cs.ucla.edu
Website: https://sites.google.com/site/fangzhenman/