Project 5 Overview

KECE456 Code and System Optimization (Fall 2025)

T.A. Inseo Kim

School of Electrical and Computer Engineering

Korea University, Seoul

Contents

Environmental Setup

Recommendations

- **If you are using VMware, apply the settings below:**
  - Virtual Machine Settings → Processors
    - Number of processor cores ≥ 4
      - It is not necessary, but it will reduce your gem5 build time
  - Virtual Machine Settings → Memory
    - Memory for this virtual machine ≥ 8GB
      - If your memory size is small, building gem5 may fail…
      - If your RAM size is small, expand swap memory
  - Virtual Machine Settings → Storage
    - Requires about 20GB free disk space

Prerequisite 1: gcc-10 Install

- **To build PyTorch with source, gcc with version $\geq$ 10 and <13 is required.**
  - If your gcc version is already in that range, pass the commands below.
  - gcc --version
- **command list**
  - sudo apt-get update
  - sudo apt-get install -y software-properties-common
  - sudo add-apt-repository ppa:ubuntu-toolchain-r/test
  - sudo apt-get update
  - sudo apt-get install -y gcc-10 g++-10
  - sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-10 100
  - sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-10 100

Prerequisite 2: gem5 install

- **gem5**
  - Documents: https://www.gem5.org/documentation/
- **command list**
  - sudo apt install build-essential git m4 scons zlib1g zlib1g-dev libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-dev python-dev python
  - sudo apt install scons
  - cd ~/path/to/project5
  - git clone https://github.com/gem5/gem5.git
  - cd gem5
  - git checkout v20.1.0.5

Build gem5 with Source

- **Reference: https://github.com/gem5/gem5.git**
- **Command list**
  - cd ~/path/to/project5/gem5
  - /usr/bin/env python3 $(which scons) -j$(nproc) build/X86/gem5.opt –j{#cores}
    - For the ones who use PC with different architectures, replace “X86” with one of as follows
      - ARM, MIPS, RISCV, NULL
- **When the build procedure is normally completed, the gem5 execution file(gem5.opt) is created.**

```
(base) compiler@ubuntu:~/project5/gem5/build/X86$ ls
arch  config  debug  enums     gpu-compute  learning_gem5  mem     proto   SConscript  systemc
base  cpu     dev    gem5.opt  kern         marshal        params  python  sim         unittest
(base) compiler@ubuntu:~/project5/gem5/build/X86$ ./gem5.opt --version
gem5.opt 2.0
```

<Fig 1. Created directory for profiling after gem5 build >

gem5 Overview

- **Full-system / CPU–Memory simulator**
  - gem5 reproduces the virtual CPU & cache & memory architecture with software, regardless of the actual host CPU (x86/ARM)
  - Thus, the standard virtual CPU spec will be provided in this project(See 11p. for details)
- **Execution Flow**
  1. Define the Simulation Target CPU
     - DerivO3CPU(Out-of-Order), 3GHz, 1~8 cores
  2. Designate the binary to execute
  3. Simulate the entire CPU/memory operation in cycle.
     - Instruction fetch → decode → execute …
     - Memory access: L1 Cache → L2 Cache → DRAM
     - branch prediction, pipeline stall, … etc.
  4. Print the performance metrics
     - cycles, instructions, CPI
     - branches, branch misses, …
     - cache read/write, hit/miss, …
     - DRAM read/write, row buffer hit/miss, …

Benchmark & Modification

Optimizing A Matrix Multiplication Library

- **Matrix Multiplication (baseline)**
  - Execute the provided code(matmul_base.cpp)
    - g++ -O0 -std=c++17 matmul_base.cpp -o matmul_base
- **Matrix Multiplication Optimization (TODO)**
  - Apply code optimization on **matmul_proj5() function** in matmul_opt.cpp
    - g++ -O0 -std=c++17 matmul_opt.cpp -o matmul_opt
- **Check Correctness (TODO)**
  - Check that both matrix multiplication results are the same

```
// COMMIT start
void matmul_proj5(char transa, char transb,
                  int64_t m, int64_t n, int64_t k,
                  float alpha, const float* a, int64_t lda,
                  const float* b, int64_t ldb,
                  float beta, float* c, int64_t ldc) {
  for (int64_t j = 0; j < n; ++j) {
    for (int64_t i = 0; i < m; ++i) {
      float sum = 0.0f;
      for (int64_t l = 0; l < k; ++l) {
        float elemA = (transa == 'T')
                        ? a[l + i * lda] // lda = M
                        : a[i + l * lda];
        float elemB = (transb == 'T')
                        ? b[j + l * ldb] // ldb = K
                        : b[l + j * ldb];
        sum += elemA * elemB;
      }
      // C(i,j) = alpha * sum + beta * C(i,j)
      c[i + j * ldc] = alpha * sum + beta * c[i + j * ldc]; // ldc = M
    }
  }
}
// COMMIT end
```

<Fig 2. Matrix multiplication code section to optimize>

Profiling Guide (1/3)

- **Reference Virtual CPU architecture**
  - Use following spec list to profile your baseline/optimized code.
    - CPU: DerivO3CPU
    - Clock: 3GHz
    - # Core: 1
    - L1I cache: 32kB, 8-way
    - L1D cache: 32kB, 8-way
    - L2 cache: 256kB, 8-way
    - Mem: DDR3_1600_8x8, 8GB
- **NOTICE!! Using other virtual CPU specs are not allowed.**

Profiling Guide (2/3)

- **Profile CPU performance using gem5 by executing matrix multiplication for the following three matrix size combinations.**
  - (16x512) x (512x512), (16x1024) x (512x1024), (16x2048) x (2048x2048)
- **Command**
  - cd ~/path/to/project5/gem5

  ./build/X86/gem5.opt   -d ~/path/to/project5/matmul_out_{base, opt}
  configs/example/se.py   --cmd= ~/path/to/project5/matmul_{base,opt}

  --options=“M K N {iteration number}"   --cpu-type=DerivO3CPU   --cpu-clock=3GHz

  --sys-clock=3GHz   --num-cpus=1   --caches --l2cache   --l1i_size=32kB

  --l1i_assoc=8   --l1d_size=32kB --l1d_assoc=8   --l2_size=256kB

  --l2_assoc=8   --cacheline_size=64   --mem-type=DDR3_1600_8x8

  --mem-size=8GB

```
(base) compiler@ubuntu:~/project5$ ./build/X86/gem5.opt   -d /home/compiler/project5/matmul_base   confi
gs/example/se.py   --cmd=/home/compiler/project5/matmul_base   --options="16 2048 2048 3"   --cpu-type=D
erivO3CPU   --cpu-clock=3GHz   --sys-clock=3GHz   --num-cpus=1   --caches --l2cache   --l1i_size=32kB --
l1i_assoc=8   --l1d_size=32kB --l1d_assoc=8   --l2_size=256kB --l2_assoc=8   --cacheline_size=64   --mem
-type=DDR3_1600_8x8   --mem-size=8GB
```

<Fig 3. gem5 profiling using given reference virtual CPU spec>

고려대학교 KOREA UNIVERSITY

KOREA UNIV. COMMIT

Profiling Guide (3/3)

- **Use stats.txt to gather the baseline/optimize profiling data as follows.**
  - **Execution time** – sim_seconds
  - **# cycles** – system.cpu.numCycles
  - **# instructions** – system.cpu.committedInsts
  - **CPI** – system.cpu.cpi
  - **# Dcache hit/miss** – system.cpu.dcache.overall_hits/misses
  - **# Icache hit/miss** – system.cpu.icache.overall_hits/misses
  - **# L2 cache hit/miss** – system.l2.overall_hits/misses
  - **# branch** – system.cpu.commit.branches
  - **# branch miss** – system.cpu.commit.branchMispredicts
  - **# DRAM read/write request** – system.mem_ctrls.dram.num_reads/writes
  - **Row buffer read/write hit** – ystem.mem_ctrls.dram.read/writeRowHits
- **Analyze the reason for speed-up**
  - Can use other profiling data in stats.txt for analysis.
    - Ex) Cache hit/miss ratio, DRAM row buffer hit/miss ratio … etc.