Computer Architecture Fall 2025

Final Project

- **Deadline**: 23:59, 2026-01-05 (no late submissions allowed)
- **Submission**: via NTU COOL, more info in Submission Requirements section below.
- **Contact**: If you have any questions, please email the TAs with the subject "[CA2025 final]".
  - Circle: r14943176@ntu.edu.tw
  - Alex: r14943141@ntu.edu.tw

Archive Content

Unified CLI for all final-project experiments based on gem5. Every command below assumes:

- `GEM5_HOME` points to a gem5 checkout (source directory).
- You are inside the `formace-lab/` directory.

Prerequisites

1. Install System Dependencies

gem5 requires various system packages. Install them using:

```
# Essential build tools and libraries for gem5
sudo apt install -y build-essential git m4 scons zlib1g zlib1g-dev \
   libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-
       dev \
   python3-dev python3-pip libboost-all-dev pkg-config

# RISC-V toolchain (required for compiling RISC-V binaries)
sudo apt install -y gcc-riscv64-linux-gnu g++-riscv64-linux-gnu
```

2. Untar

```
tar -zxvf CA2025_final.tgz
```

Your directory structure should look like:

```
final/              # modified gem5
└── formace-lab/    # This repository
```

3. Build gem5.fast

```
# Build gem5.fast (takes ~5-10 minutes)
cd formace-lab
python3 -m venv venv
venv/bin/python -m pip install -r requirements.txt
venv/bin/python main.py build --target gem5 --jobs $(nproc)
```

This will build `gem5.fast` at `$GEM5_HOME/build/RISCV/gem5.fast`.

Thus, the default `GEM5_HOME` environment variable should be set to `../`. For example, `export GEM5_HOME=../ && venv/bin/python main.py <command line option for main.py>`

Note: There are three case studies in this final project, which are enumerated below with the corresponding command to execute each case. We recommend you to browse through the case studies first, and then consult Submission Requirement for what you need to submit.

Case 1 – Cache Experiments

1.1 L1 Cache Size Sweep (Teaching Example)

This case displays PMU summary - use it to learn how to extract metrics from stats.txt

```
export GEM5_HOME=/path/to/gem5 && venv/bin/python main.py example --out
    output/<student_id>/case1/1-1
```

This example will:

1. Run the `mm` benchmark (matrix multiplication) with different L1 cache sizes (8kB, 32kB, 64kB)
2. **Display a PMU metrics summary** showing IPC, L1D MPKI, L2 MPKI, and Branch MPKI
3. Generate `all_results.csv` with the extracted metrics

Key metrics to extract from stats.txt:

- `IPC` = Instructions Per Cycle = `numInsts / (simTicks / 1000)`
- `L1D MPKI` = L1 Data Cache Misses Per Kilo Instructions = `(l1d.overallMisses / numInsts)* 1000`
- `L2 MPKI` = L2 Cache Misses Per Kilo Instructions = `(l2.overallMisses / numInsts)* 1000`
- `Branch MPKI` = Branch Mispredictions Per Kilo Instructions = `(branchPred.condIncorrect / numInsts)* 1000`

Note: Case 1.1 is included for teaching purposes and is NOT graded. Use it as a reference for the CSV format.

1.2 L2 Cache Size Sweep

```
export GEM5_HOME=/path/to/gem5 && venv/bin/python main.py suite cache-
    size --benchmarks mm --l1d-size 32kB --l2-sizes 128kB 256kB 512kB
    --out output/<student_id>/case1/1-2
```

1.3 Replacement Policy

```
export GEM5_HOME=/path/to/gem5 && venv/bin/python main.py suite cache-
    policy --benchmarks cache_thrash --cache-policy LRU FIFO TreePLRU
    --l1d-size 32kB --l2-size 256kB --out output/<student_id>/case1/1-3
```

Case 2 – Branch Study

2.1 LocalBP vs BiModeBP

```
export GEM5_HOME=/path/to/gem5 && venv/bin/python main.py suite branch
    --benchmarks towers --predictors LocalBP BiModeBP --out output/<
    student_id>/case2/2-1
```

2.2 Correlated Branches

```
export GEM5_HOME=/path/to/gem5 && venv/bin/python main.py suite branch
    --benchmarks branch_corr --predictors LocalBP BiModeBP --out output
    /<student_id>/case2/2-2
```

2.3 Biased Branches

```
export GEM5_HOME=/path/to/gem5 && venv/bin/python main.py suite branch
    --benchmarks branch_bias --predictors LocalBP BiModeBP --out output
    /<student_id>/case2/2-3
```

All stats (IPC, branch MPKI, etc.) land in `all_results.csv`.

Case 3 – Prefetchers

3.1 Upstream Prefetcher Sweep

```
export GEM5_HOME=/path/to/gem5 && venv/bin/python main.py suite
    prefetch-upstream --benchmarks stream --prefetchers
    StridePrefetcher AMPMPPrefetcher BOPPrefetcher --stride-degree 1 2
    4 --out output/<student_id>/case3/3-1
```

3.2 GHB-Lite Bring-up

Background: This case implements a simplified version of the Global History Buffer (GHB) prefetcher. For details on the original algorithm, see:

- Paper: Memory Prefetching Using Adaptive Stream Detection (Nesbit & Smith, MICRO 2004)

1. **Build the GHB unit test binary**:

   ```
   export GEM5_HOME=/path/to/gem5 && venv/bin/python main.py build --
       target ghb-test --jobs $(nproc)
   ```

2. **Run the host-side unit tests**:

   ```
   venv/bin/python -m pytest tests/test_ghb.py -v
   ```

   - PASS: `4 passed` (BuildPatternFromPC, PageCorrelationWorksWithoutPC, PatternTablePredictsMostLikelyDelta, FallbackUsesRecentDeltas)
   - FAIL: pytest shows which specific test failed. Fix the bug before continuing.

3. **Performance comparison** (Case 3.2 - compare GHB vs Baseline stride prefetcher):

   ```
   export GEM5_HOME=/path/to/gem5 && venv/bin/python main.py suite
       prefetch-ghb --out output/<student_id>/case3/3-2
   ```

   This runs a fixed configuration comparing your GHB_LiteStudent against BaselineStride (degree=1) across 7 benchmarks:

   - **Benchmarks**: stream, vvadd, qsort, towers, mm, binary_search, pointer_chase
   - **Cache config**: L1I=32kB, L1D=32kB, L2=256kB

   The output displays a performance comparison table showing:

   - ΔIPC: IPC improvement (positive = better)
   - ΔL1D_MPKI: L1D MPKI change (negative = fewer misses, better)
   - ΔL2_MPKI: L2 MPKI change (negative = fewer misses, better)

   `all_results.csv` records detailed metrics for each benchmark+prefetcher combination.

Practical Tips

- **Outputs**

  Every suite command writes:

- `all_results.csv`
- `benchmark/config/(nocache|caches)/stats.txt`

Note: Suite commands (Case 1.2+) do NOT display PMU metric summaries. You must extract metrics from the generated `stats.txt` files yourself (see Case 1.1 for guidance on which stats to extract).

- **Building**

  Before running any experiments, build gem5 using: `bash export GEM5_HOME=/path/to/gem5 && venv/bin/python main.py build --target <target>` Available build targets:

  - `gem5` — build `gem5.fast`
  - `ghb-test` — build only the GHB unit test binary
  - `all` — build all binaries

- **Testing**

  The project includes four test suites in `tests/`:

  1. **test_build.py** – Tests that gem5.fast and ghb_history.test.opt can be built
  2. **test_ghb.py** – GHB prefetcher unit tests (4 tests for GHB logic)
  3. **test_smoke.py** – End-to-end smoke tests for all 6 Cases (1.1, 1.2, 2.1, 2.2, 3.1, 3.2)
  4. **test_main.py** – Unit tests for main.py structure and constants

  You can run tests with: `bash GEM5_HOME=/path/to/gem5 venv/bin/python -m pytest tests/`

Submission Requirements

What to Submit

You must submit **THREE** items in your submission tarball, and you can use the `submit` command to automatically package all items (see below):

1. **summary.csv** - A CSV file summarizing ALL your experiment results.
2. **gem5 source code** - Your complete gem5 source implementation, including all files required to rebuild gem5.
3. **experiment outputs** - Complete `output/<your_student_id>/` directory with all `stats.txt` files from each experiment.

Creating summary.csv

You must manually create `summary.csv` by extracting PMU metrics from each experiment's `stats.txt` file. Reference Case 1.1 for guidance on which statistics to extract. A template `summary.csv.example` is provided in `final/formace-lab/`.

Key metrics to extract from stats.txt (see Case 1.1 for formulas):

- `IPC` = Instructions Per Cycle = `system.cpu.numCycles / system.cpu.committedInsts`
- `L1D_MPKI` = L1 Data Cache Misses Per Kilo Instructions = `(system.cpu.dcache.overallMisses / system.cpu.committedInsts)* 1000`
- `L2_MPKI` = L2 Cache Misses Per Kilo Instructions = `(system.l2.overallMisses / system.cpu.committedInsts)* 1000`
- `Branch_MPKI` = Branch Mispredictions Per Kilo Instructions = `(system.cpu.branchPred.condIncorrect / system.cpu.committedInsts)* 1000`

Directory Structure

Your `output/` directory **must** follow this structure:

```
output/
└── <your_student_id>/          # e.g., B12345678/
   ├── summary.csv         # *** REQUIRED: Your manually created summary ***
    ├── case1/
    │   ├── 1-1/                # Case 1.1 (Example)
    │   │   ├── mm/
    │   │   │   └── l1d_*/caches/stats.txt
    │   ├── 1-2/                # Case 1.2 (L2 Sweep)
    │   │   └── mm/
    │   │       └── l2_*/caches/stats.txt
    │   └── 1-3/                # Case 1.3 (Policy)
    │       └── cache_thrash/
    │           └── policy_*/caches/stats.txt
    ├── case2/
    │   ├── 2-1/                # Case 2.1
    │   ├── 2-2/                # Case 2.2
    │   └── 2-3/                # Case 2.3
    └── case3/
        ├── 3-1/                # Case 3.1
        └── 3-2/                # Case 3.2 (GHB)
```

Creating Your Submission Archive

Use the built-in `submit` command to package your submission:

```
# Package your submission (requires GEM5_HOME to be set)
export GEM5_HOME=/path/to/gem5
venv/bin/python main.py submit <your_student_id>

# This creates: <your_student_id>_submission.tar.gz
# The archive includes:
# 1. Complete gem5 source code (with your implementations)
# 2. formace-lab/ directory
```

```
# 3. output/<your_student_id>/ with all experiment results
```

IMPORTANT - Your Responsibility:

The `submit` command is provided as a convenience tool, but **YOU are responsible** for verifying your submission is correct:

1. **Verify the archive extracts correctly**:

   ```
   # Extract to a test directory
   mkdir /tmp/verify-submission
   cd /tmp/verify-submission
   tar -xzf <your_student_id>_submission.tar.gz
   ```

2. **Verify your code compiles**:

   ```
   cd /tmp/verify-submission
   export GEM5_HOME=$(pwd)
   cd formace-lab
   python3 -m venv venv
   venv/bin/pip install -r requirements.txt
   venv/bin/python main.py build --target gem5 --jobs $(nproc)
   ```

   - If build fails, your submission will receive **zero points**

3. **Verify all experiment outputs exist**:

   ```
   # Check that output/<your_student_id>/ has all required cases
   ls -R formace-lab/output/<your_student_id>/

   # Verify summary.csv exists
   ls formace-lab/output/<your_student_id>/summary.csv

   # Verify all stats.txt files exist
   find formace-lab/output/<your_student_id>/ -name "stats.txt" | wc
       -l
   ```

   - Missing `output/<your_student_id>/` → **zero points**
   - Missing `summary.csv` → **significant point deduction**
   - Missing stats.txt files → **partial credit only**

4. **Quick verification commands**:

   ```
   # Check archive contents
   tar -tzf <your_student_id>_submission.tar.gz | less

   # Check your output directory is included
   tar -tzf <your_student_id>_submission.tar.gz | grep "output/<
       your_student_id>" | head -20
   ```

```
# Verify gem5 source is included
tar -tzf <your_student_id>_submission.tar.gz | grep "^src/" | head
     -20
```

Grading Criteria

CSV Summaries — 30%

- Applies to: Case 1.2, 1.3, 2.1, 2.2, 2.3, 3.1
- You must fill out `summary.csv` and include the corresponding `stats.txt` files for each case.
- **5 points per case** (total 30 points)

GHB Implementation — 70%

- Applies to: Case 3.2
- There will be 10 benchmarks (7 public + 3 hidden).
- You must modify the gem5 source code to implement the GHB prefetcher, build gem5, and pass all benchmarks.
- **7 points per benchmark**:
  - 3 points — correctness
  - 4 points — performance compared to TA's baseline