Advanced Computer Architecture #HW1

Due: 11/14 (Mon.). No late delivery is allowed

In computer system design, there are several interacting parameters or dimensions affecting the system performance and cost. A designer is dedicated to finding the optimal choice, which is usually a compromise, depending on system usage, technology, cost, etc. For designers to explore the design space, there is a need to have an effective and efficient computer system simulator.

gem5 is a computer system simulator which is widely used in academia and industry for computer system architecture research. It is a modular platform encompassing system-level architecture as well as processor microarchitecture. Additionally, gem5 is a community led project with an open governance model.

RISC-V is an open standard instruction set architecture (ISA) based on established reduced instruction set computer (RISC) principles. Unlike most other ISA designs, the RISC-V ISA is provided under open source licenses that do not require fees to use. A number of companies are offering or have announced RISC-V hardware, open source operating systems with RISC-V support are available and the instruction set is supported in several popular software toolchains.

In this project, you are going to simulate the compiler-based optimizations for reducing cache miss rate via gem5 and RISC-V architecture. To answer the following questions, you need to report the simulation results (involving at least the **number of ticks** and **miss rates**), and compare and analyze them.

Question 1: Implement the loop interchange method (Page 40 of Slide 2) in C language and simulate the row major-based and the column major-based codes.

Question 2: Implement the blocking method (Pages 41-42 of Slide 2) in C language.

2.1) Simulate the non-blocking and the blocking codes. You need to specify the array size and the blocking factor by yourself. One requirement is that your blocking code should be able to demonstrate the effectiveness of reducing cache miss rate.
Note: You can use the default cache setting for cache size and associativity. A large array size would result in huge simulation time.
(NOTE: You may not be able to demonstrate the difference with a large cache size in the following sub-experiments.)

2.2) Simulate the blocking code in 2.1) with at least **FIVE** different blocking factors to show how the blocking factor affects the cache miss rate. Do not change the array size, and the cache size and associativity.

2.3) Simulate the blocking code with the best blocking factor (leading to the lowest cache miss rate) in 2.2) and at least **THREE** different cache associativity to show how cache associativity affects the cache miss rate. Do not change the array size and the cache size.

2.4) Again, simulate the blocking code with the best parameter setting in 2.3) with at least **THREE** different cache size to show how cache size affects the cache miss rate. Do not change the array size.

Examples of compiling your code and using gem5:

RISC-V compiling command:

```
$ /opt/riscv/bin/riscv64-unknown-linux-gnu-gcc blocking.c -o0 -o blocking.o -static
```

gem5 simulation command:

```
$ ./build/RISCV/gem5.opt ./configs/example/se.py -c blocking.o --caches
```

(Note: gem5/config/common/Options.py can give you an overview of command parameters.)

Deliveries

1) Report in PDF format of answering the above questions.
   Ex. m12345678_hw1.pdf

2) All the c codes you develop.

3) Upload your source codes and PDF tarball (*.tgz) to Moodle.
   Ex. m12345678_hw1.tgz

References:

https://www.gem5.org/
https://en.wikipedia.org/wiki/Gem5
https://en.wikipedia.org/wiki/RISC-V
https://www.gem5.org/documentation/general_docs/building
https://github.com/riscv-collab/riscv-gnu-toolchain