DEEC
DEPARTAMENTO DE ENGENHARIA
ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

HIGH-PERFORMANCE COMPUTING ARCHITECTURES

Lab assignment #2

Abstract: The second lab consists in the analysis of in-order and out-of-order processor architectures and their connection with the memory subsystem. For this, students are expected to develop simple programs to stress the processor architectures and to determine the influence of some design parameters.

The laboratory work will be developed using the state-of-the-art gem5 simulator, using realistic processor configurations.

1 INTRODUCTION

For the development of this work, students should use the gem5 processor simulator (http://gem5.org/), which is modular platform for computer-system architecture research and development, being widely adopted both by the academia and by the industry (e.g., ARM).

Students can opt by installing gem5 in their own computers (not recommended), or rely on a previously installed version, available at:

Machine name: `cuda1.scdeec.tecnico.ulisboa.pt` or `cuda2.scdeec.tecnico.ulisboa.pt`
Username: `acedes`*<group number>* (e.g., for group 1, `acedes01`)
Password: `acedes`*<group number>*`_2223` (e.g., for group 1, `acedes01_2223`)
Connection protocol: `SSH`
Gem5 simulator: `/opt/gem5/`
ARM compiler: `/opt/ccross_aarch64/bin/aarch64-none-linux-gnu-gcc`

Under windows, to connect to these machines, you need to either use specific software (e.g., MobaXTerm).

Under a Linux Shell (including for MobaXTerm), to connect to the machines type:

```
ssh acedes<group number>@cuda1.scdeec.tecnico.ulisboa.pt
```

or

```
ssh acedes<group number>@cuda2.scdeec.tecnico.ulisboa.pt
```

To help the development of the work, students should initially follow the gem5 tutorial book available at https://www.lowepower.com/jason/learning_gem5/.

All gem5 files are available at the folder `/opt/gem5/`. The files concerning the tutorial are available under `/opt/gem5/configs/learning_gem5/`

2 DEVELOPMENT AND EXECUTING APPLICATIONS IN GEM5

For this work, students should rely on the ARM v8 instruction set architecture and on two core microarchitecture models: `arm A7` (in-order) and `arm A15` (out-of-order)[1]. The Gem5 simulator supports two execution modes:

- System call emulation is faster to execute but is less precise, as it hides the impact of the operating system in the execution of the programs; yet for most cases it is reliable enough

[1] The internal structure of both in-order and out-of-order processors will be studied in detail in the classes. However, this is not essential for now.

- Full system emulation allows simulating the impact of the operating system but is harder to simulate and leads to longer simulation (i.e., host) execution time. In particularly, it requires the use of an operating system that is properly compiled to the target architecture.

For this work system call emulation will be used, as it is sufficiently accurate and much simpler to use.

2.1 Program compilation

To develop test programs, students should rely on C code and compile using aarch64 `gcc`. Since for the scope of this work Gem5 will be configured in system call emulation, dynamic library linking cannot be used. Hence, the flag `-static` is required to embed all function calls into the binary file. For example, for a source program in file `sample_source_code.c`, compilation into binary file `sample_binary` is performed as follows:

```
/opt/ccross_aarch64/bin/aarch64-none-linux-gnu-gcc -static sample_source_code.c -o
sample_binary
```

There are four useful compilation configurations to keep in mind:

1. No software optimizations (-O0, “oh-zero”):

```
/opt/ccross_aarch64/bin/aarch64-none-linux-gnu-gcc -static -O0
sample_source_code.c -o sample_binary
```

2. Loop level optimizations (-O2, “oh-two”):

```
/opt/ccross_aarch64/bin/aarch64-none-linux-gnu-gcc -static -O2
sample_source_code.c -o sample_binary
```

3. Auto-vectorization with NEON/Advanced SIMD:

```
/opt/ccross_aarch64/bin/aarch64-none-linux-gnu-gcc -static -O3 -march=armv8-
a+simd -fno-tree-vectorize sample_source_code.c -o sample_binary
```

4. Auto-vectorization with Scalable Vector Extension (SVE):

```
/opt/ccross_aarch64/bin/aarch64-none-linux-gnu-gcc -static -O3 -march=armv8-
a+sve -fno-tree-vectorize sample_source_code.c -o sample_binary
```

To aid the development of assembly code, the tool `objdump` is recommended. To use this tool, execute the command:

```
/opt/ccross_aarch64/bin/aarch64-none-linux-gnu-objdump -d sample_binary | less
or
/opt/ccross_aarch64/bin/aarch64-none-linux-gnu-objdump -d sample_binary >
some_file.txt
```

The command portion that leads the | (pipe) character will disassemble the binary `sample_binary`. The final portion will send the disassembly output to the `less` viewer or to `some_file.txt`.

To understand the binary files, students the following documentation is recommended:

- **Overview of the ARM aarch64 ISA:** https://armkeil.blob.core.windows.net/developer/Files/pdf/graphics-and-multimedia/ARMv8_InstructionSetOverview.pdf

- **Arm Architecture Reference Manual:** https://developer.arm.com/docs/ddi0487/latest/arm-architecture-reference-manual-armv8-for-armv8-a-architecture-profile

- **A Sneak peak into SVE and VLA programming:** https://developer.arm.com/-/media/Arm Developer Community/Images/White Paper and Webinar Images/HPC_White_Papers/a-sneak-peek-into-sve-and-vla-programming.pdf?revision=5abd0d7b-e853-4e96-931b-4d18b2273813

- **Arm A64 Instruction Set Architecture:** https://developer.arm.com/docs/ddi0596/d/a64-sve-instructions-alphabetic-order

- **ARM Architecture Reference Manual Supplement, The Scalable Vector Extension (SVE)** https://static.docs.arm.com/ddi0584/a/DDI0584A_a_SVE_supp_armv8A.pdf
- **GCC Extended ASM:** https://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html

2.2 Executing the applications with the Gem5 simulator

To run the compiled applications in the Gem5 simulator, a command like the following must be run:

```
<GEM5_DIR>/build/ARM/gem5.opt <PROCESSOR CONFIGURATION> <binary> <binary_args>
```

In this work the `<PROCESS CONFIGURATION>` is specified by the two_level.py Python file specified in `<GEM5_DIR>/configs/learning_gem5/ (<GEM5_DIR>=/opt/gem5)`.

3 CLASS EXERCISE:

Start by experimenting with the Gem5 online tutorial (http://learning.gem5.org/book/part1).

The objective of this work is to create specific micro-benchmarks that stress specific components of an out-of-order architecture, namely:

a. BASELINE: Maximum performance, measured in IPC (mandatory).
b. ISSUE STALLS: Micro-benchmark that generates stalls during issue (e.g., due to renaming, ROB).
c. BRANCH MISS-PREDICTION: Micro-benchmark that generates branch miss-predictions.
d. CHACHE MISSES: L1 and L2 micro-benchmarks that generates cache misses at L1 and L2, respectively.

For the completion of this assignment only 3 benchmarks need to be created, out of which case a) is mandatory and serves as reference. Students are required to create microbenchmarks for 2 of the three remaining cases, with one final case remaining as optional.

All the development should be performed by considering the execution of an out-of-order core (processor model `DerivO3CPU`). However, the final results and analysis should also consider the in-order core (processor model `MinorCPU`)

To accomplish this objective, use the following steps:

1. Create a folder for your group project under `/home/accedes`*<group number>*`/lab2`. For example, group 3, should work under directory `/home/acedes03/lab2/`.

2. Create a small basic benchmark `ubench.c` (e.g., a hello world program) and compile it:

   ```
   /opt/ccross_aarch64/bin/aarch64-none-linux-gnu-gcc -O0 -static ubench.c -o ubench
   ```

   To simplify you can first add the ARM compiler to your path. First type

   ```
   echo $PATH
   ```

   and check whether the directory `/opt/ccross_aarch64/bin/` is already listed. If not, type:

   ```
   export PATH=/opt/ccross_aarch64/bin/:$PATH
   ```

   then simply use:

   ```
   aarch64-none-linux-gnu-gcc -static ubench.c -o ubench
   ```

3. Copy the base configuration files from `/opt/gem5/configs/learning_gem5/part1/*` to your `lab2/` folder and update the path on each copied file

   Note: copying only the `two_level.py` file will not suffice as there is as a dependency with the `caches.py`.

4. Edit the copied files and change lines 51-52:

   ```
   # Add the common scripts to our path
   m5.util.addToPath('../../')
   ```

   to

   ```
   # Add the common scripts to our path
   m5.util.addToPath('/opt/gem5/')
   m5.util.addToPath('/opt/gem5/configs/')
   ```

5. Create two copies of the `two_level.py` configuration file (available in the gem5 folder `configs/learning_gem5/part1/`), one for the in-order processor (e.g., `two_level_inorder.py`) and another out-of-order (e.g.., `two_level_ooo.py`). Adapt each configuration file so that the first uses an in-order processor (`MinorCPU`), the other the out-of-order (`DerivO3CPU`). As an example, to generate the in-order core replace lines 96-97:

   ```
   # Create a simple CPU
   system.cpu = TimingSimpleCPU()
   ```

   With

   ```
   # Create a simple CPU
   system.cpu = MinorCPU()
   ```

6. Run the benchmark on both scripts, e.g.,

   ```
   cd /home/acedes<group number>/lab2/

   /opt/gem5/build/ARM/gem5.opt two_level_inorder.py ./ubench

   cp m5out/stats.txt m5out_inorder.txt

   /opt/gem5/build/ARM/gem5.opt two_level_ooo.py ./ubench

   cp m5out/stats.txt m5out_outoforder.txt
   ```

7. Evaluate the difference in performance by comparing the file statistics.

8. Change the benchmark to solve the handout.

4 PROJECT DUE DATE AND REPORT FORMAT

The project should be completed until March 26, with the 4-page report being submitted online (via Fenix) and the corresponding code. Do not forget to clean the submitted files before submitting your work and to clearly point out in the report each script. The report should follow the template for the IEEE Computer society journals:

https://www.ieee.org/publications_standards/publications/authors/author_templates.html

Use the Template selector ➔Transactions, Journals and Letters ➔ IEEE Transactions on Computers

The report should include:

- an abstract (single paragraph, 100-200 words) explaining the objective of the work;
- an introduction section explaining what is the characteristic of both cores that you are trying to exploit in order to solve the problem;
- a solution section describing the structure of your programs – you may use a figure, diagram or pseudo-code to help explaining the idea, but not the original C code;
- an experimental results section showing the obtained results (support your text with tables or figures); by analysing the two models, try to explain the obtained speed-ups;
- a conclusions section.

5 USEFUL STATISTICS (FILE M5OUT/STATS.TXT)

`system.cpu.ipc`	Instructions per cycle
`system.cpu.rename.BlockCycles`	Number of cycles rename is blocking (no physical register available)
`system.cpu.rename.ROBFullEvents`	Number of times rename has blocked due to ROB full
`system.cpu.branchPred.lookups`	Total number of branch predictions (conditional + unconditional). Includes fetched branches that are a result of a miss-speculation
`system.cpu.branchPred.condPredicted`	Total number of conditional branch predictions. Includes fetched branches that are a result of a miss-speculation. Thus, `lookups-condPredicted` is the total number of unconditional branches.
`system.cpu.branchPred.BTBxxx`	Statistics concerning direct or indirect branches that used the Branch Target Buffer. Thus, it excludes branches predicted as not taken, but includes fetched branches that are a result of a miss-speculation
`system.cpu.branchPred.indirectxxx`	Statistics concerning branches whose target is given by a register (indirect branches). Thus, it excludes branches predicted as not taken, but includes fetched branches that are a result of a miss-speculation
`system.cpu.branchPred.usedRAS`	Total number of function returns (uses return address stack)
`system.cpu.commit.branches`	Total number of branches commited. Same as `branchPred.lookups`, but excludes branches that are a result of miss-speculation
`system.cpu.commit.branchMispredicts`	Total number of branches committed that were miss-predicted
`system.cpu.rename.BlockCycles`	Number of cycles rename is blocking (no physical register available)
`system.cpu.rename.ROBFullEvents`	Number of times rename has blocked due to ROB full
`system.cpu.dcache.overall_miss_rate::total`	Miss rate for the L1 data cache
`system.cpu.icache.overall_miss_rate::total`	Miss rate for the L1 instruction cache
`system.l2cache.overall_misses::total`	Miss rate for the L2 cache (instructions and data)

Observation: there is no easy way of counting the branch miss-prediction considering all branches (i.e., including branches that were only fetched as a result of a miss-speculation). To account for branch miss-prediction the metrics for the actual (committed) branches should be used instead. Hence, the branch miss-prediction is given by: `commit.branchMispredicts/commit.branches.`