Architetture dei Sistemi di Elaborazione **O2GOLOV** **Laboratory 5**	Delivery date: 5 November 2024 Expected delivery of lab_05.zip must include: This file in pdf format.

Exercise 1:

Software Optimizations

Starting from Exercise 2 of Lab 4, you are required to further speedup the benchmark (*my_c_benchmark_2*) 😈.

For readability, provide the previously used configurations (Cut & Paste).

Parameters	**Configuration 1**	**Configuration 2**	**Configuration 3**	**Configuration 4**
1st changed parameter	The_cpu.numRobs = 6	the_cpu.fetchWidth = 5	the_cpu.fetchWidth = 5	the_cpu.fetchWidth = 5
2ndchanged parameter	None	the_cpu.decodeWidth = 4	the_cpu.decodeWidth = 4	the_cpu.decodeWidth = 4
3rdchanged parameter	None	the_cpu.numIQEntries = 6	the_cpu.numIQEntries = 6	the_cpu.numIQEntries = 6
4thchanged parameter	None	None	OpDesc(opClass= "IntAlu", opLat=5, pipelined=False)	OpDesc(opClass= "IntAlu", opLat=5, pipelined=False)
5th changed parameter	None	None	None	the_cpu.branchPred = predictor.create_BiModeBP()

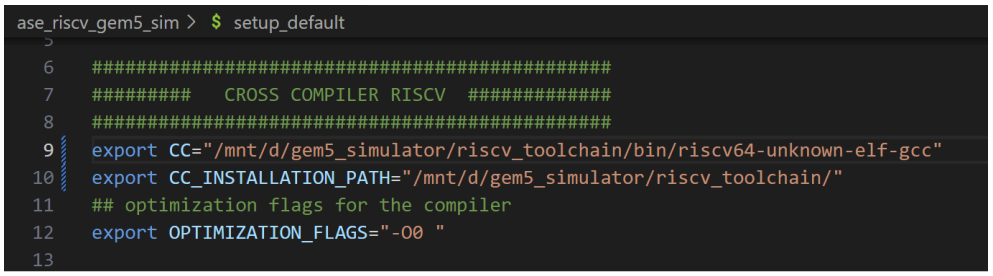
Original CPI (no hardware optimization): 2,408

	Configuration 1	**Configuration 2**	**Configuration 3**	**Configuration 4**
CPI	2,408	1,608	2,154	2,101
Speedup (wrt Original CPI)	1	1,497	1,117	1,147

Despite the hardware enhancements for increasing the CPU performance, remember that optimizing compilers for programs in high-level code exist. The aim of optimizing compilers is to minimize or maximize some attributes of an executable computer program (code size, performance, etc.). They are also aware of hardware enhancements to perform very accurate optimizations.

Compilers can be your best friend (or worst enemy!). The more information you provide in your program, the better the optimized program will be.

You can compile your programs with different SW optimization strategies and/or additional features.
In the *setup_default* file:



You can change the line 12.

Simulate the program for different optimization levels and collect statistics. You are required to change the OPTIMIZATION_FLAGS variable in the *setup_default*. O0 is the default value, you need to change the optimization value accordingly to the values in parenthesis in the following Table.

DO NOT CONFUSE -O3 WITH O3 PROCESSOR.

TABLE1: IPC for different compiler optimization levels and configurations

Optimization / Configuration	Opt lvl 0 (-O0)	Opt lvl 1 (-O1)	Opt lvl 2 (-O2)	Opt size (-Os)	Opt lvl 3 (-O3)	Opt lvl 2 (-O2 --fast-math)
Original Configuration	0,4152	0,3722	0,3928	0,3827	0,3928	0,3923
Configuration 1	0,4565	0,4089	0,4060	0,4079	0,4060	0.4058
Configuration 2	0,6245	0,5858	0,5802	0,5713	0,5802	0,5799
Configuration 3	0,4552	0,4592	0,4757	0,4483	0,4757	0,4688
Configuration 4	0,4613	0,4623	0,4768	0,4493	0,4768	0,4699
Program Size [Bytes]	10084	9932	9908	9886	9908	9908

Regarding the Program Size (Code and Data!!), you can retrieve the size from:

```
~/ase_riscv_gem5_sim$/opt/riscv-2023.10.18/bin/riscv64-unknown-elf-size --format=gnu
--radix=10 ./programs/my_c_benchmark/my_c_benchmark.elf
```

For brave and curious students:
For visualize the enabled optimizations from the compiler perspective, you can run:

```
~/my_gem5Dir$ /opt/riscv-2023.10.18/bin/riscv64-unknown-elf-gcc -Q -O2 --help=optimizers
```

By changing the "-O2" parameter with the desired one, you will find the enabled/disabled optimizations.
Here are some possible types of optimizations:

- https://en.wikipedia.org/wiki/Optimizing_compiler
- https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

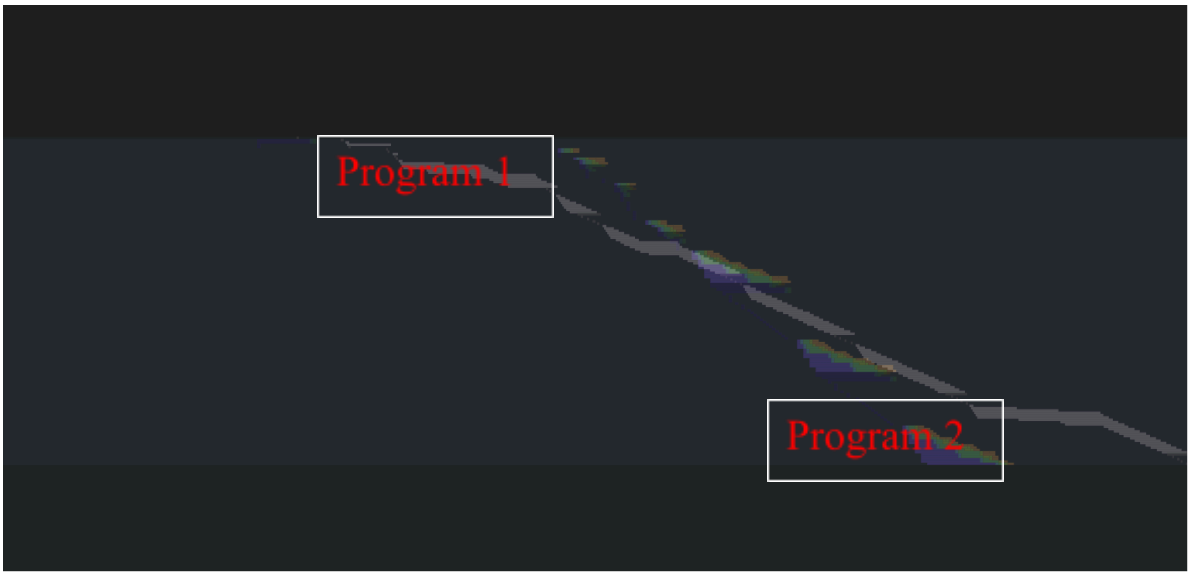
Exercise 2:

Given your benchmark (*my_c_benchmark_2.c*), select the best optimization to obtain **your best angle of optimization,** compared to the baseline configuration **(***riscv_o3_custom.py; -O0***).**

1. Based on Table 1 (from Exercise 1), select the best optimization (for example, the green box corresponding to Configuration 1 with -O2).

Optimization / Configuration	Opt lvl 0 (-O0)	Opt lvl 1 (-O1)	Opt lvl 2 (-O2)	Opt size (-Os)	Opt lvl 3 (-O3)	Opt lvl 2 (-O2 --fast-math)
Original Configuration	0,4152	0,3722	0,3928	0,3827	0,3928	0,3923
Configuration 1	0,4565	0,4089	0,4060	0,4079	0,4060	0.4058
Configuration 2	0,6245	0,5858	0,5802	0,5713	0,5802	0,5799
Configuration 3	0,4552	0,4592	0,4757	0,4483	0,4757	0,4688
Configuration 4	0,4613	0,4623	0,4768	0,4493	0,4768	0,4699
Program Size [Bytes]	10084	9932	9908	9886	9908	9908

2. By using **Konata**, overlap the two pipelines (the original obtained with *riscv_o3_custom.py* and the optimized corresponding to the best SW-HW combination) to compute your angle of optimization.

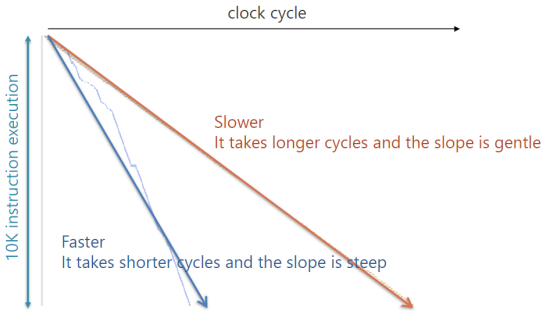


Compute the angle α (named optimization angle) existing between the traces.

Hint: To load different traces in **Konata, load them separately.** Afterward, **righ-click in the pipeline visualizer and select "transparent mode". You need to adjust the scale!**

3. To compute the **angle of optimization** α:

$$\alpha = \arctan\left(\frac{ClockCycles_{baseline}}{Instructions_{baseline}}\right) - \arctan\left(\frac{ClockCycles_{optimized}}{Instructions_{optimized}}\right)$$



The angle of optimization is equal to:
$Clock\ baseline = 15121\ Inst\ baseline = 6945\ Clock\ opt = 11055\ Inst\ opt = 6945$

$$\alpha = 7.468880366$$

4. Do you see any visual improvements (for example, a less discontinued pipeline)? Yes, why? No, why? What is happening? How could they be improved?

The optimized pipeline has fewer discontinuities, suggesting reduced stalls and better handling of branches, and the slope is steeper showing a faster execution of the program.