Architetture dei Sistemi di Elaborazione O2GOLOV **Laboratory 5**	Delivery date: Until 2 AM, November 6th 2024 Expected delivery of lab_5.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**	**Configuration 5**
First changed paramenter	CPU_IntALU-> count=6	L1Cache-tag_latency=1	the_cpu.numROB Entries = 128	L1Cache-tag_latency=1	L1Cache-tag_latency=1
Second changed paramenter		L1Cache.response_latency=1	the_cpu.numIQEntries = 6	L1Cache.response_latency=1	L1Cache.response_latency=1
Third changed paramenter		L1Cache.data_latency=1		L1Cache.data_latency=1	L1Cache.data_latency=1
Fourth changed paramenter		L1Cache.assoc=4		L1Cache.assoc=4	L1Cache.assoc=4
Fifth changed paramenter				the_cpu.numROB Entries = 128	the_cpu.numROB Entries = 128
6th changed parameter				the_cpu.numIQEntries = 6	the_cpu.numIQEntries = 12

Original CPI (no hardware optimization): 2.190180

	Configuration 1	**Configuration 2**	**Configuration 3**	**Configuration 4**	**Configuration 5**
CPI	2.190180	2.047219	1.605301	1.447422	1.202202
Speedup (wrt Original CPI)	1	1.069831806	1.364342264	1.513159258	1.821806984

Despite the hardware enhancements for increasing the CPU performance, remember that optimizing compilers for programs in high-level code also 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:

```
ase_riscv_gem5_sim > $ setup_default
6  ##############################################
7  #########   CROSS COMPILER RISCV  #############
8  ##############################################
9  export CC="/mnt/d/gem5_simulator/riscv_toolchain/bin/riscv64-unknown-elf-gcc"
10 export CC_INSTALLATION_PATH="/mnt/d/gem5_simulator/riscv_toolchain/"
11 ## optimization flags for the compiler
12 export OPTIMIZATION_FLAGS="-O0 "
13
```

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	0.456584	0.408966	0.40600	0.407997	0.406004	0.405801
Configuration 1	0.456584	0.408966	0.40600	0.407997	0.406004	0.405801
Configuration 2	0.488468	0.428646	0.436444	0.435337	0.436444	0.436365
Configuration 3	0.622936	0.576217	0.579632	0.571329	0.579632	0.579701
Configuration 4	0.690884	0.633028	0.637306	0.626976	0.637306	0.634392
Configuration 5	***0.831807***	0.793875	0.795641	0.776690	0.795641	0.789945
Program Size [Bytes] *Conf 5*	10700	10548	10524	10502	10524	10524

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 guys:
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:

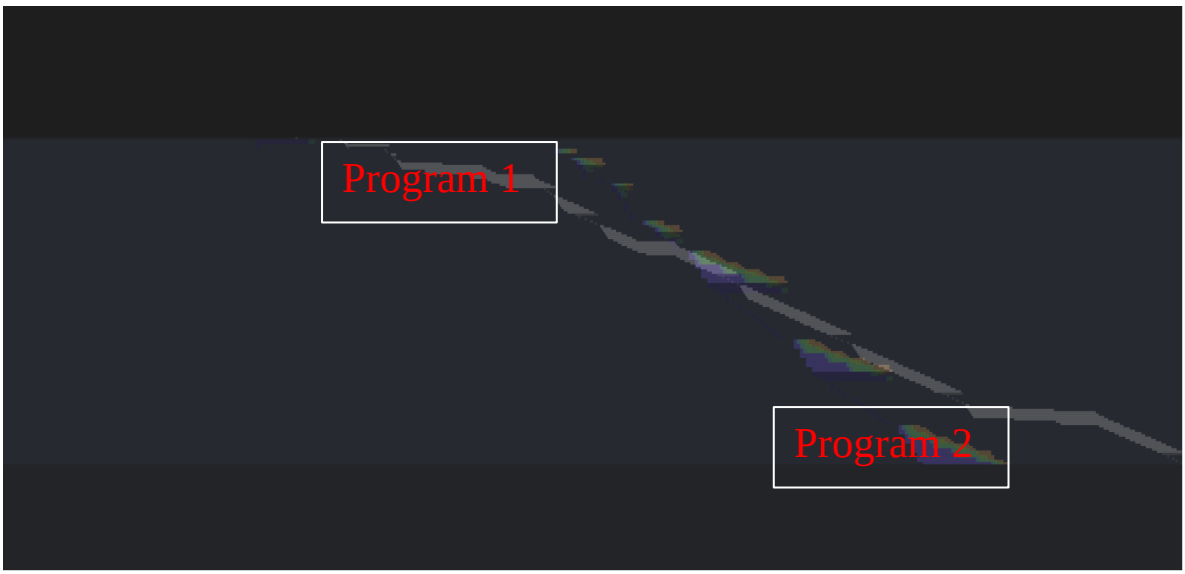
Given your benchmark (*my_c_benchmark.c*), select the best optimization to obtain **your best angle of optimization,** compared to the baseline configuration **(***riscv_o3_custom.py; -O0***).**

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						
Configuration 1			Best IPC			
Configuration 2						
Configuration 3						
Configuration 4						
Configuration 5						Worst IPC
Program Size [Bytes]						

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	0.456584	0.408966	0.40600	0.407997	0.406004	0.405801
Configuration 1	0.456584	0.408966	0.40600	0.407997	0.406004	0.405801
Configuration 2	0.488468	0.428646	0.436444	0.435337	0.436444	0.436365
Configuration 3	0.622936	0.576217	0.579632	0.571329	0.579632	0.579701
Configuration 4	0.690884	0.633028	0.637306	0.626976	0.637306	0.634392
Configuration 5	***0.831807***	0.793875	0.795641	0.776690	0.795641	0.789945
Program Size [Bytes] *Conf 5*	10700	10548	10524	10502	10524	10524

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.

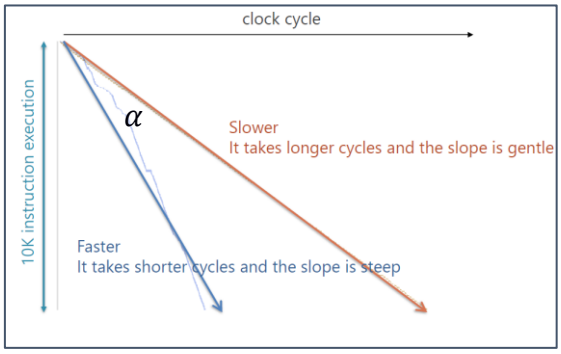


Compute the angle $\boldsymbol{\alpha}$ (named optimization angle) existing between the traces.

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 $\boldsymbol{\alpha}$**:

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



The angle of optimization is equal to:

$$\tan^{-1}\left(\frac{15121}{6945}\right) - \tan^{-1}\left(\frac{8300}{6945}\right)$$

$$= 15.2517231927391°$$

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

La pipeline ottimizzata presenta una significativa riduzione degli stalli rispetto a quella di base. Questo miglioramento è dovuto a diverse ottimizzazioni applicate, che consentono una gestione più efficiente delle dipendenze tra istruzioni e una migliore capacità di riempimento dei buffer delle diverse fasi, evitando la saturazione prematura della cache. Di conseguenza, il processore può lavorare in modo più continuo, riducendo i tempi di inattività.
A livello visivo, la pipeline ottimizzata appare più "ripida" rispetto alla versione base, con un angolo di ottimizzazione di circa 15°. Questa maggiore pendenza riflette un flusso di esecuzione più regolare e meno frammentato, grazie alla riduzione degli stalli.
Per migliorare ulteriormente le prestazioni della pipeline, potrebbe essere utile aumentare la larghezza dei buffer nelle fasi di fetch, decode e, soprattutto, nella fase di commit, dove ancora si osservano alcuni rallentamenti anche nella versione ottimizzata. Questo permetterebbe di gestire un numero maggiore di istruzioni simultaneamente, massimizzando il parallelismo e riducendo ulteriormente il numero di stalli e cache miss.