

Exercise 1:

Software Optimizations

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

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

Parameters	Configuration 1	Configuration 2	Configuration 4	Configuration 5
First changed parameter	FetchWidth=8	numIQEntries=4	FetchWidth=8	FetchWidth=8
Second changed parameter	DecodeWidth=8	renameWidth=4	FetchBufferSize=32	FetchBufferSize=32
...		numRobs=4	DecodeWidth=8	DecodeWidth=8
...			numIQEntries=8	NumIQEntries=8
...			renameWidth=8	RenameWidth=8
...			numRobs=8	issueWidth=8
...				DispatchWidth=8
...				WbWidth=4

Original CPI (no hardware optimization): 2.083105

	Configuration 1	Configuration 2	Configuration 4	Configuration 5
CPI	1.983529	1.770152	1.236586	1.086349
Speedup (w.r.t. the Original CPI)	1.05	1.18	1.68	1.92

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
5
6 #####
7 ##### CROSS COMPILER RISC-V #####
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	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)
Configuration						
Original	0.480	0.3964	0.4436	0.415	0.4436	0.45862
Configuration	053	46	26	027	26	2
Configuration 1	0.497	0.4191	0.4472	0.417	0.4472	0.46055
	950	23	54	030	54	3
Configuration 2	0.480	0.4800	0.4436	0.415	0.4436	0.45862
	053	53	26	027	26	2
Configuration 4	0.808	0.7495	0.7718	0.730	0.7718	0.78577
	433	22	57	470	57	7
Configuration 5	0.920	0.8076	0.8408	0.793	0.8408	0.84023
	515	53	91	778	91	2
Program Size [Bytes]	3228	3044	3032	3016	3032	3032

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>

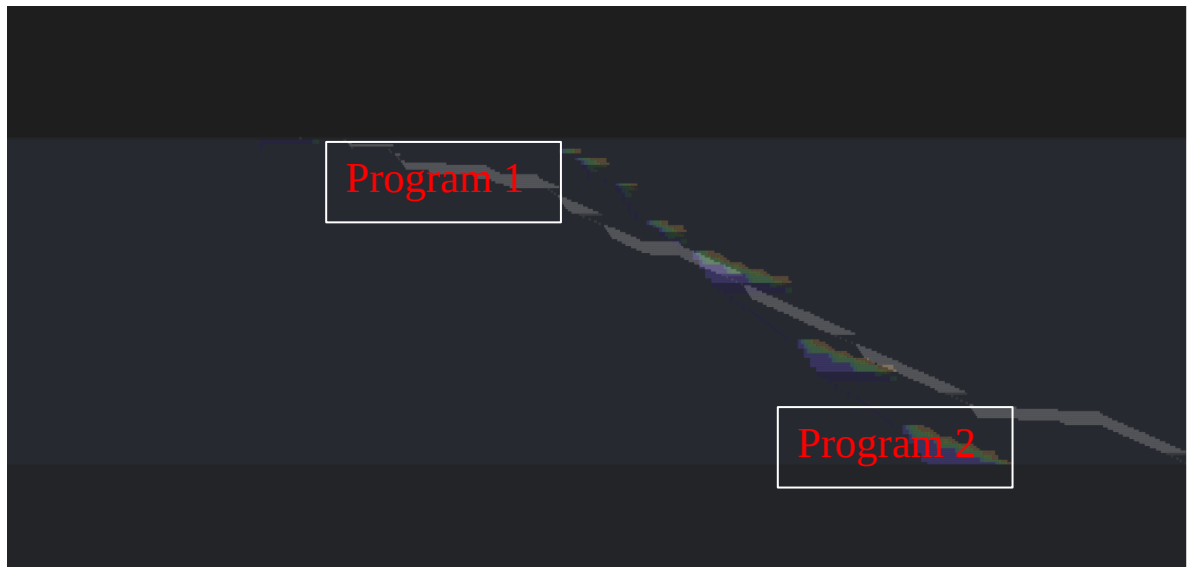
Exercise 2:

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	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)
Configuration						
Original	0.480	0.3964	0.4436	0.415	0.4436	0.45862
Configuration	053	46	26	027	26	2
Configuration 1	0.497	0.4191	0.4472	0.417	0.4472	0.46055
	950	23	54	030	54	3
Configuration 2	0.480	0.4800	0.4436	0.415	0.4436	0.45862
	053	53	26	027	26	2
Configuration 4	0.808	0.7495	0.7718	0.730	0.7718	0.78577
	433	22	57	470	57	7
Configuration 5	0.920	0.8076	0.8408	0.793	0.8408	0.84023
	515	53	91	778	91	2
Program Size [Bytes]	3228	3044	3032	3016	3032	3032

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.

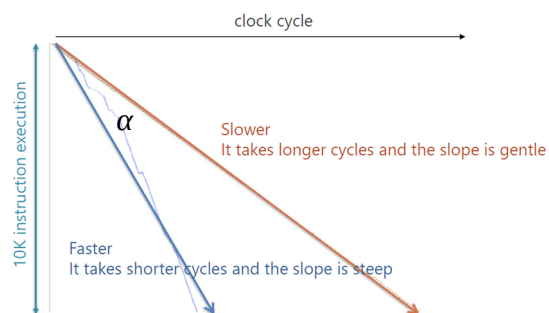


Compute the angle α (named optimization angle) existing between the traces.

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

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

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



The angle of optimization is equal to:

$$\arctan(0,480053) - \arctan(0,920515) = -0,296471384 \text{ rad} = -16,986559 \text{ deg}$$

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?

Comment box

la discontinuità visiva nella pipeline viene generata quando un branch è misstaken e la “grandezza” della discontinuità dipende da due fattori:

- 1) numero di istruzioni flushate (aumentano la discontinuità verticale)
- 2) tempo in attesa di risoluzioni di stalli prima di impiegato prima di riconoscere che il branch sia misstaken (aumenta la discontinuità orizzontale).

Il programma migliorato presenta sicuramente delle riduzioni di stalli per cui la discontinuità orizzontale si riduce, però, avendo aumentato il numero di fetch per singolo colpo di clock, ogni branch misstaken comporta una grande perdita, non solo in efficienza, ma anche in compattezza visiva.



Figure 1: Comparison due programmi

Figure 2: Originale

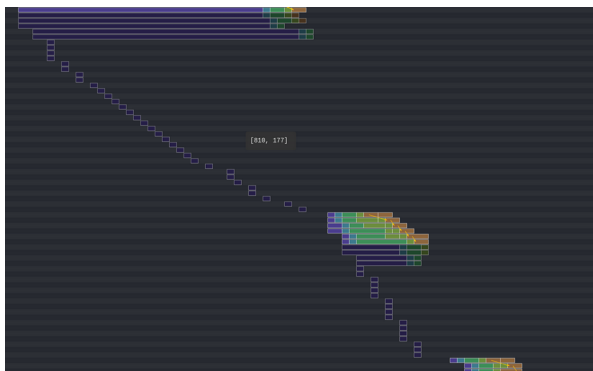


Figure 3: Ottimizzato

