Exercise 8: Caches-Aware Programming of Matrix Multiplication

Matrix multiplication is one of the most important operations in modern computing applications, for instance neural neural networks used in autonomous cars. In this exercise you will optimize matrix multiplication code to improve performance and evaluate the results for the different cache sizes and structures.

Task 1. Run Matrix Multiplication

The file `tiled_matmul.c` contains the C code for the basic matrix multiplication and tiled matrix multiplication known from the lecture.

1. Use the Compiler Explorer to generate RISC-V assembly code that you can execute in Ripes. Start with the file `tiled_matmul.c`.
   Compiler:rv32-gc clang trunk
   Optimization: -O3

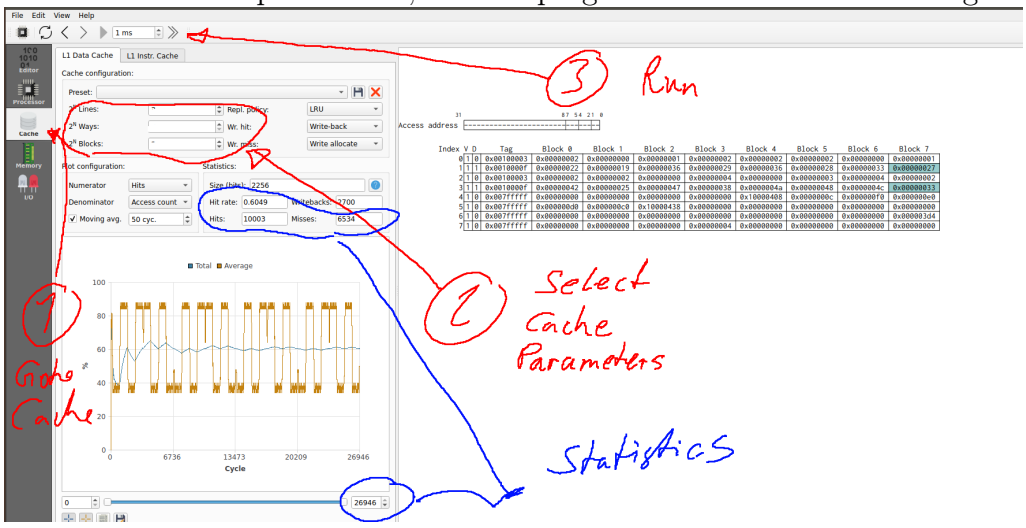
2. Copy the resulting assembly code into a file `rv_assembly.asm` in the `exercises/08_cache directory`.

3. To make you life easier, the script clang_replace.py makes some changes required to run a clang output in RIPES. Usage:
   `python clang_replace.py rv_assembly.asm`
   This creates a new file `ripes_rv_assembly.asm`

4. Open `ripes_rv_assembly.asm` in RIPES and make changes in the assembly code if necessary.

5. Select the "Single Cycle Processor" in RIPES

6. Select some cache parameters, run the program and check the resulting statistics



Task 2. Optimize Basic and Tiled Matrix Multiplication

In Task 1 you got one matrix multiplication running. In this part of the exercise your task is to optimize and improve the performance of the program which depends here on a number of software and hardware parameters. We define performance as execution time of the program under the following assumptions:

- CPI of non-memory instruction: 1
- CPI of memory instruction with cache hit: 2
- CPI of memory instruction with cache miss: 100
- f_{max} of the processor

Here are the steps for you to follow:

1. Change one or more of the following parameters:
   - Tile size: parameter t_size in `tiled_matmul.c`, e.g. 1, 2, 4, 8
   - Cache parameters in RIPES:
     - Lines: 2^0, 2^1, 2^2, 2^3, 2^4, 2^5
     - Ways: 2^0, 2^1, 2^2, 2^3, 2^4, 2^5
     - Blocks: 2^0, 2^1, 2^2, 2^3, 2^4, 2^5
2. Run the program as described in Task 1
3. Add your parameters and the resulting statistics to our **High-Score List**:
   - Total Instructions
   - Hits: Instructions with a Cache Hit
   - Misses: Instructions with a Cache Miss

   `https://dhbwstg-my.sharepoint.com/:x:/g/personal/michael_klaiber_lehre_dhbw-stuttgart_de/ETTWVcleq-hLpEAXGB_vmckB_8BlatYwn9gwpEZjb2t3yw?e=8B0B4u`
4. Try to think of parameters that will result in a better performance and goto step 1.