built on 2021/07/02 at 15:45:37 ***due:*** *Wednesday, July 23rd @ 11:59pm*

Hand-in Instructions

To submit this assignment, please follow the steps below:

1. Log-in to Syskill using your Mahidol ID (i.e., uXXXXXX).
2. The files to hand in are

   ```
   mm.c mm.h
   ```

   If you want to add any additional files to support your code, please zip up all the files as `proj3.zip`
3. Submit your code on canvas

1 Optimization Lab (120 points)

In this project, everyone will be implementing a code that perform matrix multiply.

1.1 Handout Instructions

You will need a starter package, which ships in the form of a tar file. On syskill, you can find `mm.tar` at

```
/handout/ComSysArch/mm.tar.bz
```

Once extracted, you will find three files: `mm.h`, `mm.c` and `mm-gen.py`.

1.2 The Matrix Generator

First, `mm-gen.py` is a script that generate two matrices giving its input size and a resulting matrix that multiply the two input (`input1.in` and `input2.in`) matrices together into a file called `reference.in`.

Please use this code to test your implementation.

Then your first step is to figure out the cache size. This can be done in multiply ways. Please Google `getting cache size in linux` and explore.

In this question, you are not allowed to change the name of the input file (please use the same name as what your python script generated because it will work with the C code). You also **cannot** change SIZEX and SIZEY in your submission, but feels free to modify when testing your code.

2 Task 1: Baseline Implementation (20 points)

Your task here is simple. Implement a function called `multiply_base` that multiply the two input matrices. Please make sure that the resulting matrices is similar to the reference output matrix generated by python.

Note that is loading the input matrix into a large array called `huge_matrixA` and `huge_matrixB`.

3 Task 2: Flushing Your Cache (15 points)

Once you have a functional code for matrix multiply. Please implement a method to flush your data cache.

This can be done by generating a fresh series of memory accesses. The key here is to make sure that all your accesses will touch all the cache blocks in your cache (so that you are guarantee to kick out any data previously stored in the cache).

Hint: malloc a contiguous address that would cover an entire cache and touch all locations

4 Task 3: Blocked Matrix Multiply (45 points)

This task consists of two main objectives.

4.1 Cache Optimization

By this point, you might realize that our cache hit rate can be somewhat low. In this task, we are going to try to improve the performance of the cache. One common technique to improve the cache hit rate is to implement Blocked Matrix Multiply. Feels free to check out `http://csapp.cs.cmu.edu/3e/waside/waside-blocking.pdf` for more information on why is this a good idea.

4.2 Register Optimization

Remember that our old friend x86 has a number of registers? If you look at the innermost loops, you will see that there are also frequent data reuse for data across each loop iterations. One common technique you can do to make things faster is to unroll the loop so that reused data are put close to each other. This way, the data would still be in the register files, leading to a very quick access. Please perform loop unrolling as necessary.

4.3 Putting things together

In this task, complete the function `multiply` to take advantage of the concept of blocked matrix multiply.

To measure performance, please take a look at `gprof`, which is a very nice profiling tool that can be used to profile the performance of an application. Feels free to use this to your advantage and see how to 1) profile the region of interest (i.e., your multiply function) and 2) measure its cache statistics.

5 Task 4: Parallelization using pthread and Beyond (40 points)

In this task, we will try to speed things up even further using multi-threading. To do this, please first make a copy of a working solution (so that you have a complete solution to tasks 1–3).

Then, copy `mm.c` and `mm.h` into `mm-mt.c` and `mm-mt.h` (and make sure to modify the header file accordingly.

Now, this new set of files are your canvas. Use the concept of multi-threading to speedup the performance.

Below are examples of things you can do to make your program faster:

- Instead of reading one file at a time, read them in parallel using pthread.
- Instead of doing one block at a time, multiply multiple blocks in parallel using pthread.
- Does it make sense to store both matrices in a row-column order? Is that cache friendly?

Explore. Do as much as you can. Have fun. Please remember that the performance of matrix multiply is very important to machine learning applications. The grade for this task will be based on both your creativity and the actual performance gain.