Assignment 4: Optimizing Matrix Operations for Better Cache Locality (100 points)

This assignment has two parts.

Shared Structure of Both Parts

Both parts re-use much of the same code, so the general instructions for how the code is laid out and how to edit/grade your submission is listed here.

General Information

- All matrices in the code are implemented as **1D arrays** that are indexed by multiplying the row offset by the number of values in a column. Can you think of why this is better for cache locality?
- The function `int *access_matrix(matrix_t *mat, int row, int col)` returns a pointer to the integer in a `matrix_t` at row `row` and column `col`. You can read/write this value by dereferencing.

Editing

Only edit the code in the part of the file labelled:

```
// ONLY EDIT THIS LOOP
```

- Importantly, do **not** touch the `MARKER_START` and `MARKER_END` variables in the code. **You may only edit code between the two MARKER variables, as this is where the implemented algorithms are.**
- Do **not** touch anything in the `res` folder. These files are required for the autograder to function.

Grading

- There are 5 test cases. A program that yields a valid matrix gives you 30% of the points for the test case. A program that has less cache misses and evictions than the original gives you 70% of the points for the test case.
- To test your program, simply run `./autograder`.

Testing

You cannot test the cache yourself, as this involves filtering the file generated by our cache simulator, `csim-ref`, but you may compile your program yourself to make sure your matrix operations are working properly. To do this:

```
gcc -Wall -O0 -g matadd.c/matmul.c -o matadd/matmul
```

Choose the appropriate (matadd/matmul) depending on the part you are working on.

Running `matadd` Choose your matrix dimension (suppose it is `n x n`) and choose your values for the matrix. Suppose we choose a `2 x 2` matrix with values

```
1 2
31 3
```

Then we would invoke `./matadd n n entries of n`, which is `./matadd 2 2 1 2 31 3` in this case.

Running `matmul` Choose your matrix dimension (suppose it is `n x n`) and choose your values for the matrix. Suppose we choose a `2 x 2` matrix with values

```
1 2
31 3
```

And a right matrix with values

```
38 1
1 2
```

Then we would invoke `./matmul n n n entries of left entries of right`, which is `./matmul 2 2 2 1 2 31 3 38 1 1 2` in this case. Note how we just put the right matrix's values immediately after the left matrix.

In both cases, the output result is stored in the `.mat` file. You can view this file however you want. In the terminal, for example, you may invoke `cat .mat` after running `matmul` or `matadd` to see the resulting matrix.

Assumptions

- All input matrices are square.
- The cache in both parts is a 1-way (direct mapped) cache with 4 sets, and the number of addresses in each block is 4. An LRU eviction policy is used.

Part 1: Matrix Addition (`matadd`) (40 points)

In this part, you will edit `matadd.c` to optimize adding 1 to every entry in a matrix. The current code written is inefficient from a spatial locality standpoint.

Currently, the code adds 1 to every entry matrix by looping through every **row** entry per column. Why does this program yield bad spatial locality as a result?

Optimize the code by editing the indicated sections as discussed in the previous section, and then run the autograder to make sure your code works.

Part 2: Matrix Multiplication (`matmul`) (60 points)

In this part, you will edit `matmul.c` to improve cache locality in matrix multiplication. Note that the normal pattern of accessing the matrix is not the most cache-optimal way to multiply a matrix, as the right matrix in the matrix multiplication is accessed in a way that is not conducive to spatial locality. Can you change the loop in the `matmul` function to be more optimal to a cache?

Optimize the code by editing the indicated sections as discussed in the previous section, and then run the autograder to make sure your code works.

Hints

You may consult this slide deck (`https://rutgers.instructure.com/files/40122456/`) to help with your understanding.

The file is also on Canvas: `Files -> A4 -> a4_slides.pdf`.

Submission

To submit your code, simply invoke `tar cvf a4.tar a4/` and submit the `a4.tar` file.

To verify your submission, you can check the contents of the tar file without extracting it using `tar tvf a4.tar`. You should see something similar (but it's okay if it's not exactly the same as)

```
drwxr-xr-x NETID/NETID   0 2024-04-22 23:34 a4/
drwxr-xr-x NETID/NETID   0 2024-04-22 23:44 a4/matmul/
-rwxr-xr-x NETID/NETID 4871768 2024-04-22 23:44 a4/matmul/autograder
-rw-r--r-- NETID/NETID     699 2024-04-22 23:44 a4/matmul/.mat
-rw-r--r-- NETID/NETID      18 2024-04-22 23:44 a4/matmul/.csim_results
-rw-r--r-- NETID/NETID      13 2024-04-22 23:44 a4/matmul/.marker
-rw-r--r-- NETID/NETID  887730 2024-04-22 23:44 a4/matmul/.trace
drwxr-xr-x NETID/NETID       0 2024-04-22 23:44 a4/matmul/res/
-rw-r--r-- NETID/NETID    2361 2024-04-22 23:39 a4/matmul/res/matmul.c
-rwxr-xr-x NETID/NETID   25592 2024-04-22 23:37 a4/matmul/res/csim-ref
-rw-r--r-- NETID/NETID    2361 2024-04-22 23:40 a4/matmul/matmul.c
drwxr-xr-x NETID/NETID       0 2024-04-22 23:45 a4/matadd/
-rw-r--r-- NETID/NETID    1828 2024-04-22 23:40 a4/matadd/matadd.c
-rwxr-xr-x NETID/NETID 4866848 2024-04-22 23:36 a4/matadd/autograder
-rw-r--r-- NETID/NETID     375 2024-04-22 23:42 a4/matadd/.mat
-rw-r--r-- NETID/NETID      14 2024-04-22 23:42 a4/matadd/.csim_results
-rw-r--r-- NETID/NETID      13 2024-04-22 23:42 a4/matadd/.marker
-rw-r--r-- NETID/NETID   35530 2024-04-22 23:42 a4/matadd/.trace
drwxr-xr-x NETID/NETID       0 2024-04-22 23:45 a4/matadd/res/
-rw-r--r-- NETID/NETID    1828 2024-04-22 23:40 a4/matadd/res/matadd.c
-rwxr-xr-x NETID/NETID   25592 2024-04-22 23:37 a4/matadd/res/csim-ref
```