Joanna Klukowska
joannakl@cs.nyu.edu

Project 3: Writing Cache-Aware Code.

Due date: Apr. 17, 11:55PM EST.

You may discuss any of the assignments with your classmates and tutors (or anyone else) but **all work for all assignments must be entirely your own**. Any sharing or copying of assignments will be considered cheating.

Project Description

In this project you will practice anlyzing cache use and how to tweak C code to make it (more) cache friendly.

The repository contains five files: `level_0`, ..., `level_4` with different C functions that perform operations that are fairly memory intensive. Your task is to implement equivalent functions (i.e., your functions should perform exactly the same tasks) but in a cache-aware manner. Your functions should be called `level_0_opt`, ..., `level_4_opt`.

Notes:

- Do not modify the code where the comments state so.
- You can add any variables or data structure you want, but you are not allowed to remove/modify available variables or data structures.
- Feel free to add additional helper functions. If you do, name them starting with the name of the function in which they are used. For example, if you want a helper function for `level_1_opt` function, it should be called `level_1_opt_helper` (or some other suffix). The cache misses that occur within the helper function will count towards the function that calls it.
- Do not use recursion.
- Your changes should not change the underlying algorithm. They should make the existing implementation more efficient. For example, if one of the functions implements a bubble sort, do not change it to quick sort.
- Always remember that cache friendly means locality of data access, both temporal and spatial.
- You should first understand what the function does, then attempt to optimize it.

Working on this Project

Your repository contains the following files

- `proj3.c` - the actual program that calls all the functions and performs preliminary correctness checks (should not be modified);
- `proj3.h` - the header file containing structure definitions, and function declarations (;if you need to define additional structures or declare additional functions, you should add them to this file)
- `level_0.c`, ..., `level_4.c` - functions that you need to optimize;
- `Makefile` - the makefile for this project

Compiling and Building the Code

You can combile and build the code by running

```
$ make clean
$ make
```

Joanna Klukowska
joannakl@cs.nyu.edu

(`make clean` is optional, but this makes sure that you are recompiling all the files that are required for the project).

This produces `proj3` program that shows which of the functions completed and if the results produced by the optimized functions match the results produced by the unoptimized functions (note, that this program does not perform a complete check, but should be a good indicator if the functions are setting the right values). The sample output is shown below:

```
$ ./proj3
level_0 completed
level_0_opt completed
        level_0_opt PASSED
level_1 completed
level_1_opt completed
        level_1_opt FAILED
level_2 completed
level_2_opt completed
        level_2_opt FAILED
level_3 completed
level_3 completed
        level_3_opt FAILED
level_4 completed
level_4 completed
        level_4_opt FAILED
$
```

Testing Cache Performance of the Code

When you complete each level, you should run the program in the cache simulator to see its cache performance. To do so, execute `make run` followed by `make results`. The sample outputs are shown below:

```
$ make run
valgrind --tool=cachegrind --D1=1024,16,32 --cachegrind-out-file=cache.trace ./proj3 > valgrind.log 2>&1

$ make results
cg_annotate --threshold=0.05 --show=D1mr,D1mw cache.trace
--------------------------------------------------------------------------------
I1 cache:         32768 B, 64 B, 8-way associative
D1 cache:         1024 B, 32 B, 2-way associative
LL cache:         6291456 B, 64 B, 12-way associative
Command:          ./proj3
Data file:        cache.trace
Events recorded:  Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Events shown:     D1mr D1mw
Event sort order: Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Thresholds:       0.05 100 100 100 100 100 100 100 100
Include dirs:
User annotated:
Auto-annotation:  off

--------------------------------------------------------------------------------
     D1mr      D1mw
--------------------------------------------------------------------------------
5,925,632 3,677,960  PROGRAM TOTALS

--------------------------------------------------------------------------------
     D1mr      D1mw  file:function
--------------------------------------------------------------------------------
  560,109         0  /build/buildd/eglibc-2.19/stdlib/random_r.c:random_r
  486,932   315,536  /build/buildd/eglibc-2.19/stdlib/random.c:random
        4 2,293,043  /home/asia/Data/NYU_Teaching/csci201/projects/proj3/proj3.c:main
   50,001    90,000  /build/buildd/eglibc-2.19/stdlib/erand48_r.c:erand48_r
1,310,723   262,145  /home/asia/Data/NYU_Teaching/csci201/projects/proj3/level_3.c:level_3
```

Joanna Klukowska
joannakl@cs.nyu.edu

```
  409,379   482,926  /build/buildd/eglibc-2.19/malloc/malloc.c:_int_malloc
        0         0  /build/buildd/eglibc-2.19/stdlib/rand.c:rand
   77,739         0  /build/buildd/eglibc-2.19/stdlib/drand48-iter.c:__drand48_iterate
1,048,578         0  /home/asia/Data/NYU_Teaching/csci201/projects/proj3/level_0.c:level_0
  131,075         0  /home/asia/Data/NYU_Teaching/csci201/projects/proj3/level_0.c:level_0_opt
  642,503         0  /home/asia/Data/NYU_Teaching/csci201/projects/proj3/level_1.c:level_1
        0         0  /build/buildd/eglibc-2.19/stdlib/drand48.c:drand48
  651,548         3  ???:???
  266,100    56,253  /build/buildd/eglibc-2.19/malloc/malloc.c:malloc
   10,204     1,577  /build/buildd/eglibc-2.19/malloc/malloc.c:_int_free
   20,983     9,960  /home/asia/Data/NYU_Teaching/csci201/projects/proj3/level_4.c:addFront
        0    68,266  /home/asia/Data/NYU_Teaching/csci201/projects/proj3/proj3.c:flood_cache
  120,779    80,007  /home/asia/Data/NYU_Teaching/csci201/projects/proj3/level_4.c:level_4
   65,652         0  /build/buildd/eglibc-2.19/malloc/malloc.c:free
   62,501     3,126  /home/asia/Data/NYU_Teaching/csci201/projects/proj3/level_2.c:level_2
```

This table contains numbers of cache read misses (D1mr) and cache write misses (D1mw). You should be looking at the lines that show the results of all of the `level_N` and `level_N_opt` functions. In the above example you can see that the `level_0` function had 1,048,578 cahe read misses and the `level_1_opt` function had 131,075 cache read misses (this is almost the order of magnitude difference!!!). The reduction in the number of cache misses will not be always so large. Your objective is to both numbers (for cache read and write misses) to be as low as possible. (Note, that sometime reducing one of those implies the other one goes up. This is fine, as long as there is overall benefit.)

The other optimized versions of functions will appear on the list when you write their implementation.

How will we grade this?

We will compare the number of cache misses in the code for each `level_N` function and its corresponding `level_N_opt` function. Your grade for each level will be determined based on the correctness of the code (i.e., does the optimized function perform the same operations as its unoptimized equivalent?) and based on the decrease in the number of cache read/write misses.

Questions

Post any questions you have regarding this assignment to Piazza under the ”homeworks” topic.

Committing and Pushing Your Work

You should commit your code regularly to the repository on GitHub (this is your backup and a proof to us that you are working on it). At the least, you should commit your work after every single problem. You are **required** to make at least 5 commits (including the push to the remote repository) before the due date for the project.

Please, run `make clean` before committing the changes so that the object files are not uploaded together with your sources.