Canvas Group Name:Group 100 Student 1:Andrew Nguyen Student 2: Allen Yabut

CSE120 HA5: Locality Detective

Due Date: Tuesday 11/26/19

In this assignment, you will optimize a matrix multiplication kernel C = AB where A, B, and C are NxN matrices by optimizing for spatial and temporal locality. This assignment requires you to write C-code that needs to be submitted as part of your submission. Your submission needs to consist of this filled out pdf, the C-code file(s) as well as a README.txt file that explains how to compile and run the code. You will only receive points when we can reproduce your results with the submitted code. Code that does not compile will receive 0 points. Please provide readable code and comments whenever useful. If we have problems understanding or running your code we might not be able to give you full credit so make it easy for us. You can use any machine (for instance, the unix timeshare) to compile and run your code. Run multiple experiments and average, especially if the machine you are executing on is under heavy load.

Use **N = 1024** for all assignments. You can find a skeleton of the matrix multiplication code here, which you need to use as a basis for this assignment:

https://drive.google.com/file/d/1Lid8sZp2_NQieZD6hEaHMFifIfQ9Vhx6/view?usp=sharing

1) Unoptimized Kernel

Compile the kernel with gcc using optimization level -O3 and measure the execution time:

a) Determine the processor frequency of your system for instance by checking "/proc/cpuinfo" on a Linux system.

   Frequency: 1200 MHz (1 Points)

b) Execution Time: 2.845462 (1 Points)

c) What is the combined size of the 3 matrices in Bytes: 25 MB (2 Points)

d) What is the total number of bytes of matrices A,B,C transferred from memory to the processor for executing the unoptimized kernel?

   17.179869184 GB (2 Points)

2) Transpose Optimization

Add a method “transpose” that computes the transpose of a given matrix. Apply the transpose function to improve the execution time of the unoptimized kernel. To measure execution time it is not required to take into account the time of executing transpose (You can do it as part of the initialization). What execution time and speed up can you achieve? Verify that the code using the transposed matrix computes the same result matrix C as the base approach of 1.)

a) Execution Time: 2.106041 Speedup: 1.35109525408 (4 Points)

3. Blocking/Tiling Optimization

The given matrix multiplication kernel exhibits temporal locality that can be exploited via tiling. Tiling partitions the matrices into smaller sub-matrices and applies the matrix multiplication operation to these tiles instead of multiplying entire rows and columns at once.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

=

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

X

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

a) Explain why the tiling optimization can improve performance. Describe the available temporal locality and how it can be exploited via tiling (2 Points)

Tiling works because cache size is limited and if the rows are long, the cache throws away what was loaded in initially. This is a problem because when you reach the end of a row or column and start the next one, you will need data from the cache to come back from memory again after you threw them out.
Tiling allows for values to go into more than result value before they’re removed from the cache.

b) Implement the tiling optimization for the matrix multiplication kernel. Make the number of tiles a configurable parameter (as a power of 2). Run the algorithm with all possible tile sizes (1, 2, 4, 8, 16, .. , 512, 1024 tiles per row/column of the original matrix) and measure the execution times. Verify that the code using the tiling approach computes the same result matrix C as the base approach of 1.) (8 Points)

1 - 3.15
2 - 1.68
4 - 1.22
8 - 1.23
16 - 1.46
32 - 1.53
64 - 1.88
128 - 2.00
256 - 2.05
512 - 2.08
1024 - 2.09

c) Given the results of your experiments can you guess anything about the memory hierarchy (e.g. size of caches, cache line size)? **(2 Extra Points)**

I would guess that the cache line size is about 4 bits long because that's where it is the most optimal.