Canvas Group Name: Student 1: Student 2:

CMPE110 HA5: Locality Detective

Solution

Due Date: Tuesday 03/05/19

In this assignment, you will optimize a matrix multiplication kernel C = AB where A, B, and C are NxN matrices, by optimizing for the 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 the code when we can reproduce your results with the submitted code. Code that does not compile will receive 0 points. Please use the 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.

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/1l6xBbRO0aHcNq4LSiZcU2pcITdR8lY4C/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:___2GHz_____ (1 Point)

b) Execution Time:___7.414689s_____ (1 Point)

c) What is the combined size of the 3 matrices in Bytes: _24 MB_ (1 Point)
1K * 1K * 8Byte (uint64) = 8MB * 3 matrices = 24MB

d) What is the total number of bytes transferred from memory to the processor for executing the unoptimized kernel?
A+B has 3 loops: 1K*1K*1K*8Byte*2 Matrices = 16GB
C has 2 loops: 1K*1K*8Byte = 8MB

_____16GB+8MB __ (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: ____1.576589__ Speedup: ____4.7x_____ (4 Points)

b) Given the achieved speedup, can you make any assumptions about the cache architecture of your system? Explain:

Transpose improves spatial locality for the B matrix by maximizing cache hits for B. An upper bound is given by Cache size/element size. 8Bytes * 4.7 = 37. The next greater power of 2 is 64, so we deduce that the cache line size of our system is 64 Bytes.
The code for the transpose/tiling version of the algorithm can be found here:
https://canvas.ucsc.edu/courses/12652/files?preview=659094

(1 Point)

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

Assume a matrix multiplication C = A*B, where B is **not** transposed. As we multiply each row of A with each column of B, in the example above each column of B would be accessed 4 times. We can make use of this temporal locality by guaranteeing that every time a column is accessed, it is in the cache. As all columns are too large to fit in the cache we tile the large matrix into smaller ones. By performing a matrix multiplication on smaller blocks we can ensure that partial columns can be accessed multiple times, exploiting temporal locality

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)

Blocked Non-transposed Matmul:

blocks	blocksize	time	speedup
1	8388608	6.686708	1.003660247
2	2097152	3.613004	1.857507769
4	524288	2.378056	2.822129925
8	131072	2.394268	2.803020798
16	32768	2.312061	2.902684228
32	8192	2.1652	3.099567246
64	2048	2.009575	3.33960315
128	512	1.704785	3.936674126
256	128	2.296789	2.921984997
512	32	5.01977	1.336950299

Blocked transposed Matmul:

blocks	blocksize	time	speedup
1	8388608	1.567445	4.281606691
2	2097152	1.539768	4.358567654
4	524288	1.529577	4.387607162
8	131072	1.499786	4.474760399
16	32768	1.396644	4.805220944

32	8192	1.177936	5.697408858
64	2048	1.086694	6.175779934
128	512	1.012745	6.626725385
256	128	0.992709	6.760473613
512	32	1.440802	4.657949531