SDSC Assignment 1

Agrim Jain, Srihari K G, Vivek Pillai

September 2023

1 Introduction

Below we have presented our results and inferences of each question. these question gave a huge insight into the world of code optimization and helped us realize how small changes in code can lead to huge differences.

2 Question 1

In this question we need to make different implementations of matmul and compare their performances.

2.1 Sub-part a

In this sub-part we need to run every possible loop ordering and compare the results. We also needed to make corresponding changes to the make file, such that all 6 loop ordering files are created simultaneously.

As mentioned earlier, we wrote all 6 different loop orderings(6 because there are 3 loops, and using permutations we know that there are 6 different arrangement of all the loops), and below we have tabulated the results of the same:

Table 1: Results of Question 1.a

Loop ordering	Execution times(in sec)	Throughputs(MegaFLOP)
ijk	71.521	240.207
kij	30.67	560.152
ikj	30.8054	557.690
kji	86.3805	198.886
jik	72.1226	238.204
jki	80.3721	213.754

We can see that just by changing the ordering of the loop, the same code can have execution times varying from 30.8 to 86.38. We can infer that this is solely based on the fact that certain loops have better spatial and temporal locality,

due to the method in which we are accessing elements in that loop ordering. This clearly results in huge differences when we operate on matrices of huge size (like 2048).

Another interesting aspect we found is that on applying the same code on matrix of size 2047, or even 2049, the time taken to execute drops by a small amount. This could be attributed to the fact that 2048 are special numbers or due to certain cache effects.

Thus through this question we become aware of the fact that just changing the ordering of the loop, can lead to huge performance boosts.

2.2 Sub-part b

In this sub-part we need to run every possible g++ optimization and compare the results.

As mentioned earlier, we made corresponding changes in the make file to run all 5 different type of g++ optimization on the same code. Below we have a table that summarizes the results obtained:

Table 2: Results of Question 1.a

Optimization	Execution times(in sec)	Throughputs(MegaFLOP)
no-opt	70.7445	242.844
O1	24.069	713.776
O2	26.8225	640.502
O3	6.33196	2713.199
O4	6.14964	2793.638

We can see that just by using different optimization methods, we can vary the execution time from 70.7445 to 6.14964 . We can see that the execution time almost reduces every time as we increase the optimization type from no-opt to O4.

The interesting aspect of this is that we can see a dip between O1 and O2, this can be due to many reason, one of which we think is that like in case of BLAS, we see that for certain time period, BLAS1 outperforms the BLAS2 in a certain range.

Therefore this question enlightens us on the importance and value added by different optimization levels.

2.3 Sub-part-c

We can conclude that BLAS is a powerful library which increases the throughput to a great extent. Using sdot function will require much more time compared to sgemm as it takes 1 row at a time from the matrix. We have used 2D matrix in sdot part to reduce the time from around 25 sec to 1.34056 seconds. sgemm takes very less time and completes the execution in mere 0.023958 seconds.

Table 3: Results of Question 1-c

Implementation	Execution Time(in sec)	Throughput(GigaFLOP)
sdot	1.34056	12.815
sgemm	0.023958	717.083

3 Question 2

Note: To run this question please put the following command :

In the 2nd question of the assignment, we needed to edit the matvec.cpp file which is intended to perform Matrix-Vector multiplication using SSE3 registers to increase the speed of implementation. We basically needed to use multiple intrinsic functions to work with SSE3 registers. Also we can increase the throughput of our program by using it in such a way that we perform loop unrolling.

Once we executed the code, we were able to complete the matrix-vector multiplication, where size of matrix and vector is 2^{14}within a second. Below we have tabulated the results and compared them with standard multiplication of matrix and vector, which is done element-wise. Note: In the standard code we have taken each row of the matrix and taken its inner product with the vector, which was done by multiplying the 2 vectors element-wise and stored accordingly.

Table 4: Results of Question 2

Method	Execution times(in sec)	Throughputs(GigaFLOP)
Matvec using Intrinsics	0.26	33831.127
Matvec using standard code	0.58	15165.678

In order to verify the correctness of the resultant vector, we have compared the values of the 2 vectors we get, that is, one vector from the matvec implementation that used intrinsics, and one that didn't use the intrinsics, The following table summarizes the error and the speedup offered by the intrinsics.

Table 5: Results of Question 2

Speedup	Error
1.89	0e+00

Note: For calculating the error we have added the absolute value of the difference between each element in the 2 vectors, that is vec_ref and vec_c. We then checked that the sum of this error is less than 10^{-6}

As you can see we have increased the speed of program by 1.89, which is a significant amount. In order to increase the speed even further, we can now apply some other tactics as well, like parallel computing. We also tried out increasing the loop unrolling by using more registers, but this is not possible as

there are only a fixed amount of registers, thus if you use too many registers, then you end up accessing memory illegally, and your answer tends to have huge errors.

4 Question 3

In the bonus question we were expected to reciprocate question 2 but for matrix-matrix multiplication, that is we need to execute matrix-matrix multiplication using SSE3 registers to increase the throughput.

Here we changed the size of the matrix to 2^{11}, that is 2048.

Below we have tabulated the results and compared them with standard multiplication of 2 matrix. Note: In the standard code we have taken each row of the matrix and taken its inner product with the column vector of the other matrix, which was done by multiplying the 2 vectors element-wise and stored accordingly.

Table 6: Results of Question 3

Matmul using Intrinsics	Matmul using standard code

In order to verify the correctness of the resultant matrix, we wrote a code to verify that each element that we have got is within some range of the actual value. Basically for each element we compared the difference of the element in Matrix implemented using intrinsics and standard code, and ensure that it is lesser than 10^{-6} .The following table summarizes the error and the speedup offered by the implementation that used intrinsics.

Table 7: Results of Question 2

Speedup	Error
5.67	0e+00

As you can see we have increased the speed of program by 5.67, which is a significant amount. We can also use methods like parellel computing to increase the throughput further. Thus we can clearly see that as the size of operations increase, the efficiency and speed of implementation that use SSE3, and its corresponding intrinsics outperform standard implemenation by a huge margin.

4.1 Inference of questions 2 and 3

This is inference can be clearly made on basis of question 2 and question 3, as question 2 is just matrix-vector multiplication, which is less exhaustive as compared to matrix-matrix multiplication.

5 Contribution

Question 1a: Srihari and Agrim
Question 1b: Vivek
Question 1c: Agrim and Srihari
Question 2: Vivek and Srihari
Question 3: Agrim and Vivek