CS601:Software Development for Scientific Computing
Assignment 1

Ashwin Waghmare 210010060
Dhrutika Kalidindi 210010021
Tejal Ladage 210010026

September 2023

1. **Matmul Problem**

   (a) The execution time and throughputs for all the possible loop orderings are:

LOOP ORDER	EXECUTION TIME(sec)	THROUGHPUT(FLOPS)
ijk	22.7165	756.27M
ikj	1.28159	13.41G
jik	24.8242	752.704M
jki	25.8764	663.92M
kij	1.46965	11.69G
kji	47.2897	363.29G

   ikj ordering is best as it has good temporal and spacial locality.

   (b) The execution time and throughputs for no optimization and different optimization levels are:

LOOP ORDER	EXECUTION TIME(sec)	THROUGHPUT(FLOPS)
no optimization	22.606	759.97M
-01	22.7178	756.23M
-02	22.5029	763.45M
-03	23.5796	728.59M
-04	22.6274	759.25M

   The -02 optimization level gives the best output.Matrix multiplication is often memory-bound, meaning that the performance is limited by memory access patterns.Aggressive optimization levels like -O3 or -O4 can sometimes introduce optimizations that increase cache misses and negatively affect performance due to cache thrashing.This might have caused -02 to be benchmark.

   (c) The execution time and throughputs for matmul using sdot() and sgemm() are:

LOOP ORDER	EXECUTION TIME(sec)	THROUGHPUT(FLOPS)
sdot	22.6406	758.80M
sgemm	0.021554	797.06G

   We can deduce that sgemm is benchmark for matrix multiplication because of its cache-friendliness,usage of parallelization and also it chooses best algorithm based on the given matrix size and hardware architecture.Whereas using sdot() is similar to ijk loop order and it doesnt have good locality.

2. **Matvec Problem**

- Observations:
  - Matvec using intrinsics completes in 0.10s, which has a speed up of 2.99 as compared to the reference code which completes in 0.29s.
  - The reason of speed up is Unrolling and is discussed in further points.
  - Accuracy seems to be 100%.
- In matvec.cpp we are multiplying an arbitrarily large matrix with an arbitrarily large vector using SSE intrinsics. Following is the step-wise explanation of the code:

  (a) Declared the following data registers: a_row for accessing elements of row-wise elements of mat_a, b_column to access elements from vec_b, coin_bag for storing multiplication of a_row and b_column, coin for adding multiplication values.
  (b) Outer for loop iterates over rows of mat_a.
  (c) coin elements are set to zero.
  (d) Load four elements from mat_a and four from vec_b. Multiply them using _mm_mul_ps() and accumulate it in coin.
  (e) Using _mm_hadd_ps() twice sums up the vector. Store this result in vec_c.
- Here we have used Unrolling, an optimization technique that aims to reduce loop overhead and improve cache performance. Instead of performing a single operation for each element in C within the innermost loop, unrolling combines multiple operations into a single loop iteration. This can reduce the number of loop iterations and make better use of the CPU cache, resulting in faster matrix multiplication.