Question – 4

Assignment – 3

In this problem, you will utilize the OpenBLAS library available on Explorer. To use Open BLAS, you will need to issue load openblas/0.3.29. You can refer to the blas_simple.c code provided for an example code.

a) Develop a matrix-matrix multiplication, multiplying two single-precision floating point matrices that are 256x256 elements. Compare your implementation to your dense implementation in problem 3. Discuss which result is faster and why.
b) Run your program OpenBLAS accelerated program on two different CPU platforms in Explorer. Discuss the CPUs you are using, and the performance differences you obtain.

Answer:

4 (a):

The below is the output using OpenBLAS library

```
[sridhar.pray@explorer-01 OpenBLAS-0.3.29]$ gcc openblas.c -o openblas -L~/mylibs/openblas/lib -lopenblas -lm
[sridhar.pray@explorer-01 OpenBLAS-0.3.29]$ ./openblas
Naive multiplication time (seconds): 0.059805
OpenBLAS multiplication time (seconds): 0.019467
Maximum difference between naive and BLAS results: 1.52588e-05
```

The below is the output for the 3rd problem.

```
[sridhar.pray@explorer-01 OpenBLAS-0.3.29]$ gcc -o matmul_optimized matmul_optimized.c -fopenmp
[sridhar.pray@explorer-01 OpenBLAS-0.3.29]$ ./matmul_optimized
starting dense matrix multiply...
a result 4.44488e+07
The total time for matrix multiplication with dense matrices = 118.257538 ms
starting sparse matrix multiply...
A result 0
The total time for matrix multiplication with sparse matrices = 102.725800 ms
The sparsity of the a and b matrices = 0.750977
```

After comparing the implementations of 3rd and 4th problem it can be said that OpenBLAS is significantly faster than both the dense and sparse implementations. The optimized BLAS library completes the matrix multiplication in 19.467 ms, which is markedly faster than the 118.257538 ms for the dense matrix implementation and even faster compared to the sparse implementation (102.725800 ms).

Reasons for Performance Difference:

- **Algorithmic Efficiency**: OpenBLAS is a highly optimized library employing sophisticated methodologies, including efficient data structures, kernel routines customized for certain CPU architectures, and fine-grained parallelism. It probably utilizes SIMD (Single Instruction, Multiple Data) instructions and further low-level optimizations that beyond standard manual enhancements.

- **Parallelism and Hardware Utilization**: OpenBLAS effectively leverages many cores and vector instructions of contemporary CPUs, which may not be fully harnessed by manually optimized code utilizing OpenMP without extensive understanding of hardware intricacies.
- **Cache and Memory Management**: OpenBLAS may employ certain algorithms for cache optimization and memory access patterns specifically designed for matrix operations, therefore decreasing cache miss rates and enhancing the throughput of matrix calculations.

4 (b):

CPU – 1: Intel ® Xeon ® Gold 5318Y CPU @ 2.10GHz

```
[sridhar.pray@explorer-01 OpenBLAS-0.3.29]$ gcc openblas.c -o openblas -L~/mylibs/openblas/lib -lopenblas -lm
[sridhar.pray@explorer-01 OpenBLAS-0.3.29]$ ./openblas
Naive multiplication time (seconds): 0.059805
OpenBLAS multiplication time (seconds): 0.019467
Maximum difference between naive and BLAS results: 1.52588e-05
```

CPU – 2: Intel ® Xeon ® CPU E5-2698 v4 @ 2.20Ghz

```
[prayags@rho OpenBLAS-0.3.29]$ gcc blas.c -o blas -I~/mylibs/openblas/include -L~/mylibs/openblas/lib -lopenblas -lm
[prayags@rho OpenBLAS-0.3.29]$ ./blas
Naive multiplication time (seconds): 0.126432
OpenBLAS multiplication time (seconds): 0.001851
Maximum difference between naive and BLAS results: 5.34058e-05
```

From the timings shown OpenBLAS speeds things up dramatically on both—but in slightly different ways. Here are a few points worth noting:

- **Xeon E5-2698 v4** (Broadwell generation) is older. Despite having a nominally higher clock (2.20 GHz), it does not support newer instruction sets like AVX-512 (Broadwell supports AVX2) whereas the **Xeon Gold 5318Y** comes from a more recent Xeon Scalable generation (Ice Lake or Cooper Lake, depending on exact model). This architecture can support AVX-512, has more advanced micro-architectural improvements, and sometimes better memory bandwidth or caches.
- Modern Xeon cores frequently exhibit superior single-thread IPC (instructions per cycle), enhanced cache efficiency, or expedited memory access, enabling them to surpass earlier CPUs at marginally reduced clock frequencies
- On the **Xeon E5-2698 v4**, OpenBLAS reduces the time from approximately 0.126 seconds to approximately 0.00185 seconds, resulting in a significant acceleration. On the **Xeon Gold 5318Y**, OpenBLAS reduces the execution time of the naive code from approximately 0.0598 seconds to approximately 0.0195 seconds. That is a significant acceleration, though not as pronounced as with the **E5-2698 v4**. A possible explanation may be the differing default threading or vectorization paths that OpenBLAS selects during runtime. On certain platforms, OpenBLAS autonomously identifies the accessible vector instructions and the core count, significantly influencing the resultant speedup.