Homework Report
Assignment #3, CSC 746, Fall 2021

Peter Ijeoma *

SFSU ID: 921373073

ABSTRACT

Investigate the effect of parallelization on the performance of vector-matrix multiply using OpenMP at various concurrency levels for both static and dynamic scheduling. The performance of basic vector-matrix multiply, parallel vector-matrix multiply (best configuration - schedule and concurrency level) are compared to reference CBLAS implementation. These experiments are done for various problem (matrix) sizes for an nXn matric and nX1 vector.

1 INTRODUCTION

The problem:

- The performance of vector-matrix multiplication, Y = Y + A*X where A is a nXn matrix and Y and X are each nX1 vectors is the subject of this exercise. We want to analyze the effects of parallelization on the performance of vector-matrix multiply. This will involve doing vector-matrix multiply for various matrix sizes - 1024, 2048, 4096, 8192, and 16384. The serial VMM performance, and the best parallel configuration (schedule and concurrency level) will be compared to CBLAS implementation of VMM.

Approach:

- Two code versions are created - one for the basic (serial) vector-matrix multiplication and the other for parallel vector-matrix multiplication using OpenMP pragmas and directives. We run each code for each matrix size n (1024, 2048, 4096, 8192, 16384) and record the run times. We also run the parallel code at different level of concurrency - 1, 2, 4, 8, 16, 32, and 64 and also record the run times. This data is used in comparing both versions of code against the benchmark CBLAS run time for vector-matrix multiplication. MFLOPs (computed from FLOPs and run times) per second is the measure of performance used.

Finding and Results:

- It is observed that basic and parallel VMM exhibit the same performance trend.
- Though CBLAS implementation of VMM performs 5 times better than serial VMM, it is only about 2 time better than parallel VMM for the matrix sizes investigated.

2 IMPLEMENTATION

Three separate code files are created; one for basic vector-matrix multiplication, another for parallel vector-matrix multiplication and the third for CBLAS vector-matrix multiplication. The run time results of the code firles are used to compute the MFLOPS for each and plotted on charts for comparison and analysis. The result of the multiplication from the basic and parallel code files is also compared to that obtained with the CBLAS code file for correctness.

*e-mail: pijeoma@mail.sfsu.edu

2.1 Basic Vector-Matrix Multiplication

This is implemented by two nested for loops. The outer loops iterates on the rows of the matrix (A) while the inner loop iterates on the rows of the vectors X and Y.

```
1   for (int i = 0; i < n; i++) // iterate for n
     rows
2   {
3      for (int j = 0; j < n; j++)
4      {
5         y[i] += A[i*n + j] * x[j];    // A is in
     row major
6      }
7   }
```

Listing 1: Basic vector-matrix multiply

2.2 Parallel Vector-Matrix Multiplication

The implementation is similar to that for basic vector-matrix multiplication. The only difference is the OpenMP pragmas that specify that the for loops can be done in parallel. The inner for loop has a reduction arround Y[i] to solve for race condition on Y[i] and thus ensure the result is not unpredictable.

```
1   #pragma omp parallel for
2   for (int i = 0; i < n; i++) // iterate for n
     rows
3   {
4      #pragma omp parallel for reduction (+:y[i])
5      for (int j = 0; j < n; j++)
6      {
7         y[i] += A[i*n + j] * x[j];    // A is in
     row major
8      }
9   }
```

Listing 2: Parallel Vector-Matrix multiplication

3 RESULTS

The two code files in this analysis were compiled with maximum compiler optimization and run on Cori KNL nodes.

3.1 Computational Platform and Software Environment

Cori nodes are configured as follows: [1]

- OS: Ubuntu (64 - bit)
- Each node is Intel Xeon phi 7250 - 68 cores 1.4GHz
- Each core has 2 512 bit vector processing units
- Each core has 4 hardware threads - 272 threads total

- 44.8 GFLOPs/core (theoretical peak)
- 3TFLOPs/node (theoretical peak)
- Each node has 96GB DDR4-2400 memory
- Each node has 460 GB/s peak bandwidth
- Each core has 64KB L1 cache (32KB instruction, 32 KB data)
- Each tile (2 cores) share a 1MB L2 cache

3.2 Test and Methodology

The objective is to measure and compare the performance of Basic, Parallel, and CBLAS versions of vector-matrix multiplication for the five matrix sizes - 1024, 2048, 4096, 8192, and 16384.

Each program iterates over the matrix sizes and for each run the elapsed time is recorded. Using the run (elapsed) time and $2n^2$ FLOPs (number of FLOPS for vector-matrix multiplication), MFLOPs per second is computed and plotted on a graph.

The performance (in MFLOPS) of each program is compared to a benchmark that is based on the CBLAS library.

The parallel code is also run at both static and dynamic concurrency modes, for the following levels of cuncurrency - 1, 2, 4, 8, 16, 32, and 64.

3.3 Basic vs CBLAS Vector-Matrix Multiplication

Tables 1, and 2 show the raw performance data collected for Basic and CBLAS Vector-Matrix Multiplications for the same matrix sizes.

Problem size	FLOPs	Elapsed time	MFLOP per Second
1024	2097152	0.001	2097.15
2048	8388608	0.02	419.43
4096	33554432	0.07	479.35
8192	134217728	0.29	462.82
16384	536870912	1.15	466.84

Table 1: Basic Vector-Matrix Multiplication performance shown in terms Run Times and MFLOP per second for various matrix (problem) sizes on Cori KNL

Problem size	FLOPs	Elapsed time	MFLOP per Second
1024	2097152	0.001	2097.15
2048	8388608	0.001	8388.61
4096	33554432	0.01	3355.44
8192	134217728	0.06	2236.96
16384	536870912	0.24	2236.96

Table 2: CBLAS Vector-Matrix Multiplication on Cori KNL. Performance is show as elapsed time and MFLOP per second

The run time for Basic Vector-Matrix multiplication for matrix size 1024 and CBLAS VMM for matrix sizes 1024 and 2048 were so small that 0.00 seconds was the actual time reported. For the purposes of computing MFLOP per second, I approximated 0.00 to 0.001.

Table 3 and figure 1 show how their performance compare for each matrix size, using MFLOPs as a measure of performance.

Looking at figure 1 and tables 1, 2, and 3 the following are observed:

- Basic Vector-Matrix Multiplication performed best for matrix size 1024. Its performance steeply degraded from matrix size 1024 to 2048. After matrix size 2048, its performance stay almost the same - with very negligible degradation with each larger matrix size. This shows that Basic VMM is good for matrix sizes up to 1024.

Problem Size	Basic MFLOPs	CBLAS MFLOPs
1024	2097.15	2097.15
2048	419.43	8388.61
4096	479.35	3355.44
8192	462.82	2236.96
16384	466.84	2236.96

Table 3: Comparison of Basic and CBLAS Vector-Matrix multiplication using MFLOPs per second as measure of performance on Cori, for various matrix sizes

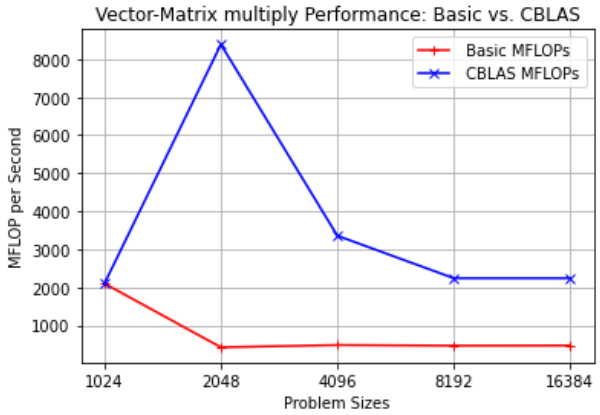

Figure 1: Plot for Basic vs CBLAS Vector-Matrix Multiplication performance. MFLOPs per second at various matrix sizes is used as measure of performance.

- The above observation is an indication that at a certain matrix size, (in this case 2048) Basic VMM has maxed out in efficiently using the resources of the machine - cache.
- the CBLAS performs approximately 5 times better than Basic VMM. From matrix size 8192, CBLAS performance remains the same, just like Basic performance, but at the same 5 times improvement compared to Basic VMM performance. At lower (1024 and below) matrix sizes, CBLAS and Basic VMM perform at about the rate rate. CBLAS attains maximum performance at 2048 while Basic performance peaked at 1024 matrix size.

3.4 Evaluation of OpenMP-parallel VMM

Table 4 and figure 2 show that data and the chart for running the parralele code in statitic schedule and the speedup realized at various lelvels of concurrency.

Prob Size	P=1	P=2	P=4	P=8	P=16	P=32	P=64
1024	0.10	0.10	1.00	1.00	1.00	0.10	0.03
2048	0.67	2.00	2.00	20.00	20.00	20.00	2.00
4096	0.78	1.40	3.50	7.00	7.00	70.00	7.00
8192	0.88	1.81	3.63	7.25	14.50	29.00	14.50
16384	0.93	1.85	3.71	7.19	14.38	28.75	28.75

Table 4: Parallel Vector-Matrix Multiply. Table shows the speedup realized at various concurrency levels running under static scheduling for various matrix sizes

Referring to table 4 and figure 2 the following observations can be made:

- Running under static scheduling is most efficient with 4 threads. At this concurrency level, the speedup is linear and slightly increases with increasing matrix size.

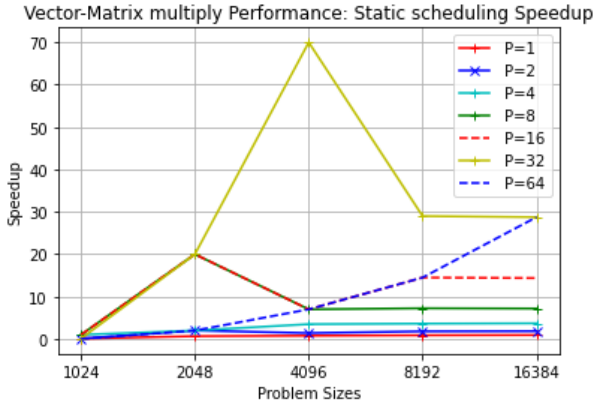

Figure 2: Realized speedup for Vector-Matrix multiplication- static scheduling at various level of concurrency and matrix sizes

- for some matrix sizes and concurrency level, the run time is so small (reported as 0.00) that I approximated them to 0.001.
- The maximum speedup (less than concurrency level) was 29 - at concurrency level 32 and matrix size 8192.
- speedup values greater than concurrency levels where obtained for situations were the run time was less that 0.01 (reported 0.00) and I approximated them to 0.001.

Below, table 5 and figure 3 show the realized performance running Vector-Matrix multiplication under dynamic schedule at various concurrency levels for various matrix sizes.

Prob Size	P=1	P=2	P=4	P=8	P=16	P=32	P=64
1024	0.10	0.10	1.00	1.00	1.00	0.10	0.05
2048	0.67	2.00	2.00	20.00	20.00	20.00	2.00
4096	0.78	1.40	3.50	7.00	7.00	70.00	7.00
8192	0.88	1.81	3.63	7.25	14.50	29.00	14.50
16384	0.93	1.85	3.71	7.67	14.38	28.75	28.75

Table 5: Parallel Vector-Matrix Multiply - Speedup under dynamic schedule at various concurrency levels and for various matrix sizes

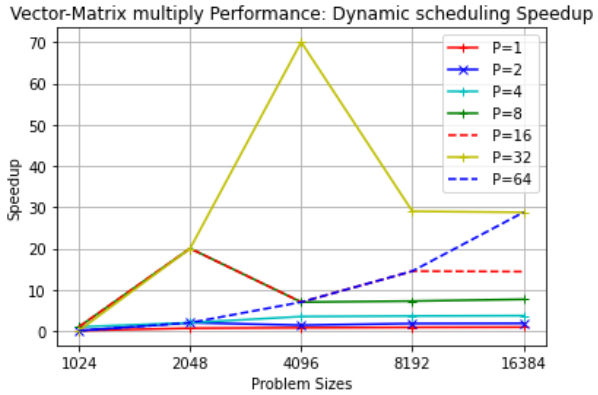

Figure 3: Realized speedup for Vector-Matrix multiplication- run under dynamic scheduling at various level of concurrency and matrix sizes

Looking to table 5 and figure 3 the following observations are made:

- Again concurrency level 4 is the most efficient just as was obtained with static scheduling. The speedup is linear and slightly increases with increasing matrix size.
- for some matrix sizes and concurrency level, the run time is so small (0.00) that I approximated them to 0.001.
- The best (under the concurrency level) speedup was 29 - at concurrency level 32 and matrix size 8192 just like was observed with static scheduling.
- speedup values greater than concurrency levels where obtained for situations were the run time was less that 0.01 (reported 0.00) and I approximated them to 0.001.
- the parallelization overhead for concurrency level 64 is quite high such that serial performance is (as expected) better for matrix size 1024.
- Concurrency levels 4 and 64 show the same linear trend. For small matrix sizes though, the overhead at concurrency level 64 is high so observed maximum speedup was 28.75 at matrix size 16384 while for concurrency level 4 it was 3.71 (almost at max 4).

3.5 Parallel P=4 vs CBLAS Vector-Matrix Multiplication

Below is a comparison of my best performing configuration and CBLAS. It is my observation that Static and Dynamic scheduling performed best in concurrency level 4. The table 6 and figure 4 show this comparison for various matrix sizes.

problem size	P=4 MFLOPs	CBLAS MFLOPs
1024	2097.15	2097.15
2048	838.86	8388.61
4096	1677.72	3355.44
8192	1677.72	2236.96
16384	1731.84	2236.96

Table 6: Shows comparison of Parallel (concurrency level - P=4) and CBLAS Vector-Matrix multiplication using MFLOPs as measure of performance at various matrix sizes

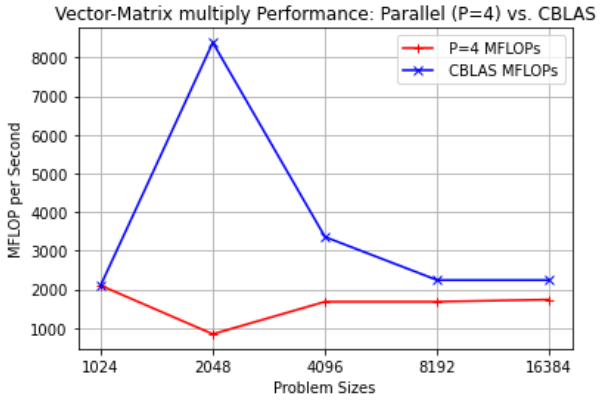

Figure 4: Parallel vs CBLAS Vector-Matrix Multiplication. Concurrency level is 4 at various matrix sizes. The same result is observed for both static and dynamic scheduling

- Though Parallel VMM performed better than serial VMM, the two exhibit the same trend. Better MFLOPs at low matrix size, asteep drop followed by improved performance from 2048 to 4096 and almost plateau (improving ever more slightly with increasing matrix (problem) size.

- This may be as a result of each thread data almost fitting the cache.

3.6 Findings and Discussion

- Comparing CBLAS and Parallel VMM is fair; CBLAS is may be only 2 times the performance of parallel VMM at matrix sizes greater than 4096. This performance gap continues to narrow as the matrix size increases and I think at some matrix size greater than 16384, the two will perform the same or the parallel VMM will perform better than CBLAS.

- Parallel VMM speedup is linear for concurrency levels 4 and 64. For matrix sizes less or equal to 16384 concurrency level 4 is better while concurrency level 64 will be better for higher matrix sizes.

REFERENCES

[1] NERSC. *Cori: KNL Compute Nodes*. URL: `https://docs.nersc.gov/systems/cori/#knl-compute-nodes`.