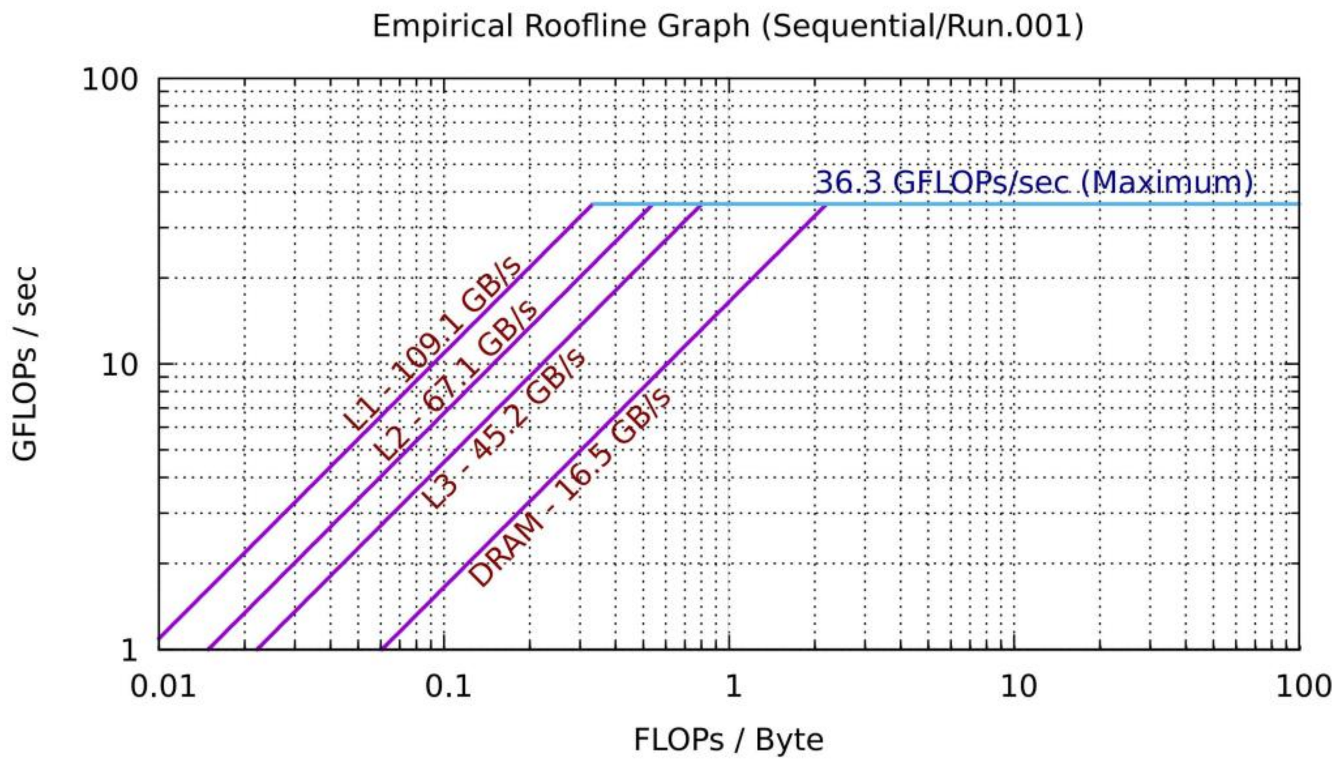
Empirical Roofline Graph (Sequential/Run.001)
GFLOPs / sec
FLOPs / Byte
100
10
1
0.01
0.1
1
10
100
36.3 GFLOPs/sec (Maximum)
L1 - 109.1 GB/s
L2 - 67.1 GB/s
L3 - 45.2 GB/s
DRAM - 16.5 GB/s

Sparse-Matrix Vector Product Performance

How does the performance (in GFLOP/s) for sparse-matrix by vector product compare to what you previously achieved for dense-matrix by dense-matrix product? Explain, and quantify if you can, (e.g., using the roofline model).

For the matrix dimension 4096: The performance of COO & COO^T is 0.8701 GFlop/s & 0.8739 GFlop/s which is slightly lower than the mult_0 version (without any loop optimizations) of dense matrix multiplication- 0.8873 GFlop/s (for 1024 matrix dimension). The other sparse matrix algorithms CSR, CSC and AOS achieve even better GFlop/s with 1.28293, 0.9485 & 0.9575 GFlop/s. Their transverse versions also achieve better performance with CSR^T = 0.9530, CSC^T = 1.2278 & AOS^T = 1.05892 GFlop/s.
For matrix dimensions higher than 1024, we don't have the performance values for the dense matrix version (Takes Prohibitively long time to run) but the numbers are not expected to increase. So, the matvec version was edited to have a minimum problem size of N(Grid) = 32 i.e N(matrix) = 1024. The GFlop/s for 1024 dimensions is comparable to the 4096 dimensions case reported above.
The ratios are as follows: COO/mult_0 = 0.98; COO^T/mult_0 = 0.98; CSR/mult_0 = 1.44; CSR^T/mult_0 = 1.07; CSC/mult_0 = 1.07; CSC^T/mult_0 = 1.38; AOS/mult_0 = 1.08; AOS^T/mult_0 = 1.19.
Based on the numeric intensity of the algorithms (computed in Q2 below) and the observed GFlops/s, it seems (based on the roofline model) that cache reuse is not available for matrix of this size. Observed GFlops/s numbers are closer to those perdicted by the DRAM line.
The predicted ratios based on the roofline model (analysis shown in Q2 below) are:
COO/mult_0 = COO^T/mult_0 = 0.85; CSR/mult_0 = CSR^T/mult_0 = 1.23 ; CSC/mult_0 = CSC^T/mult_0 = 1.23; AOS/mult_0 = AOS^T/mult_0 = 0.85
The ratios are as expected with CSR & CSC performing better than mult_0 while COO being worse than mult_0. However, AOS performs better than COO as well as mult_0.
The optimized versions of dense matrix multiplication however achieve much better performance in terms of GFlop/s with mult_2(hoisting+tiling) & mult_3 (hoisting+tiling+blocking) achieving 3-4 times as many GFlop compared to sparse multvec and especially their transverse versions mult_t_2 & multi_t_3 achieving 6-8 times as many GFlop/s as the sparse multvec.

Referring to the roofline model and the sparse matrix-vector and dense matrix-matrix algorithms, what performance ratio would you theoretically expect to see if neither algorithm was able to obtain any reuse from cache?

Without any reuse the numeric intensity of dense matrix algorithm mult_0 = (2*N^3)/(3*N^3*8) = 1/12 Flops/Byte, which translates to about 1.3 GFlop/s using DRAM based on the roofline models.
For COO the numeric intensity = (2 NNZ)Flops/(NNZ doubles for storage_ elements + 2 NNZ size_t corresponding to row & col indices + 2N doubles corresponding to x & y arrays).
NNZ = 5N calculated based on the table generated by the matvec.exe. Also, indices are size_t = 8 bytes for a 64-bit system.
Thus numeric intensity of COO = (10*N Flops)/(5N * 8bytes + 10N * 8bytes + 2N * 8 bytes) = 10N/136N = 1/14 Flops/Byte, which translates to about 1.1 Flops/Byte using DRAM.
For CSR the numeric intensity = (2 NNZ)/(NNZ storage_ elements + NNZ column indices + N row indices + 2N elements from x & y).
This is equal to = 10N Flops/(5N * 8 bytes + 5N * 8 bytes + N * 8 bytes + 2N * 8 bytes) = 10/104 = 1/10 Flops/Bytes, which translates to about 1.6 GFlop/s using DRAM.
For CSC the numeric intensity & hence the GFlops/s should be same as CSR based on the roofline model.
For AOS the numeric intensity & hence GFlops/s should be same as COO based on the roofline model.
Based on roofline model, the transpose versions of sparse matvec algorithms should have same performance as the non-transposed versions.

How does the performance (in GFLOP/s) for sparse-matrix by vector product for COO compare to CSR? Explain, and quantify if you can, (e.g., using the roofline model).

For matrix dimension4096: The performance of COO is worse than CSR as expected from the roofline analysis. The predicted ratio is: COO/CSR = 1.1/1.6 (analysis in Q2) = 0.6875.
The observed ratio of COO/CSR = 0.6782; COO/CSR^T = 0.9121; COO^T/CSR = 0.6812; COO^T/CSR^T = 0.9161.
The transpose version of CSR involves writing into 'y' in the innermost loop at each iteration which slows it down compared to the original CSR where a local variable is used for storage in the innermost loop. However, the performance of both the CSR versions is better than COO.

Sparse-Matrix Dense-Matrix Product

How does the performance (in GFLOP/s) for sparse matrix by dense matrix product (**SPMM**) compared to sparse-matrix by vector product (**SPMV**)? The performance for SPMM should be about the same as for SPMV in the case of a 1 column dense matrix. What is the trend with increasing numbers of columns? Explain, and quantify if you can, using the roofline model.

The performance for SPMV, and SPMM in the case of 1 column dense matrix show similar trends with CSR being the best algorithm and COO performing the worst for different values of N(Matrix).

However, the performance of all algorithms (except CSR) in the case of SSPM is worse than its SSPV version, for 1 column dense matrix. CSR for SSPM (1 column) performs better than SSPV.

As we increase the number of columns in the dense matrix B the performance of SSPM in terms of GFlops/s drops.

The numeric intensity of matmat for COO is given by: 2*NNZ*(M)/(2*N*M doubles corresponding to B & C matrices + NNZ doubles corresponding to the storage_ elements + 2*NNZ size_t corresponding to the row and column indices), where B and C are matrices of size NxM and NNZ = 5N.

Thus numeric intensity of COO = 10*N*M flops/(2*N*M*8 bytes + 5N*8 bytes + 2*5*N*8 bytes) = 10NM/(16NM + 40N + 80N).

Similarily for CSR = 10NM/(16NM + 40N + 48N). These function increases with the number of columns M in B, for a given N size. So, Flops/Byte should gradually increase leading to better GFlops/sec. But, observed trends show decreasing GFlops/sec for all the methods at all values of N. This might be due to the inavailability of cache and usage of slower memory as the number of columns are increased.

The GFlops/sec numbers for already quite low (about 1 GFlop/sec) for the matvec product, which from the roofline plot should correspond to the DRAM plot based on the numeric intensity of the algorithms. So, it's not explainable based on the roofline plots.

How does the performance of sparse matrix by dense matrix product (in GFLOP/s) compare to the results you got dense matrix-matrix product in previous assignments? Explain, and quantify if you can, using the roofline model.

The performance of SPMM is worse compared to the dense matrix product mult_0. Since running mult_0 for matrix sizes above 1024 takes too long, we tested the SPMM algorithms for N(Matrix) = 16, 64, 256 and 1024, with B matrix having the same number of columns as row i.e. N. We have the mult_0 performance values for these problems sizes from previous assignment.
N(Matrix) = 16 : COO/mult_0 = 0.6408/2.1082 ; CSR/mult_0 = 2.2001/2.1082 ; CSC/mult_0 = 0.6535/2.1082 ; AOS = 0.6535/2.1082 ;
N(Matrix) = 64 : COO/mult_0 = 0.6145/1.8435 ; CSR/mult_0 = 2.2393/1.8435 ; CSC/mult_0 = 0.6775/1.8435 ; AOS = 0.6573/1.8435 ;
N(Matrix) = 256 : COO/mult_0 = 0.5898/1.0637 ; CSR/mult_0 = 0.9816/1.0637 ; CSC/mult_0 = 0.6030/1.0637 ; AOS = 0.6434/0.8873 ;
N(Matrix) = 1024: COO/mult_0 = 0.5328/0.8873 ; CSR/mult_0 = 0.6732/0.8873 ; CSC/mult_0 = 0.4569/0.8873 ; AOS = 0.5381/0.8873 ;
We see that SPMM algorithms (except CSR) perform worse than mult_0 for all the problem sizes. CSR performs better for smaller problem size but the performance drops with increasing problem size.
The numeric intensity for COO can be found by substituing M=N in the formula above: 10N^2/(16N^2+120N). Similarily, for CSR numeric intensity = 10N^2/(16N^2 + 88N). For large enough N, Flops/Byte = 1=/16 = 0.625 (approx), which translates to about 10 GFLops/sec based on the Roofline model using the DRAM.
So, the performance is not explainable using the roofline model.

Array of Structs vs Struct of Arrays (AMATH583)

How does the performance
(in GFLOP/s) for sparse-matrix by vector product of struct of arrays (COOMatrix) compare to that of array of structs (AOSMatrix)? Explain what about the data layout and access pattern would result in better performance.

The AOS matvec performance is comparable with SOA matvec performance and slightly better at times, with the ratio SOA/AOS = 0.98 (mean); SOA/AOS = 0.07(standard deviation). Since, the innermost loop requires access to the row index, col index and value of given element at the same time, AOS might offer slight advantage because in the case of AOS both the indices and the value of a given element are contiguously stored, while SOA store these 3 elements separately.

How does the performance
(in GFLOP/s) for sparse-matrix by dense matrix product of struct of arrays (COOMatrix) compare to that of array of structs (AOSMatrix)? Explain what about the data layout and access pattern would result in better performance.

The AOS matmat performance is better than SOA matvec performance with the ratio COO/AOS less than 1 (about 0.95) for various size of B (32, 64, 128, 256). Since the innermost loop requires access to the row index, col index and value of given element at the same time, AOS is expected to offer slight advantage because in the case of AOS both the indices and the value of a given element are contiguously stored, while SOA store these 3 elements separately.

About Midterm

The most important thing I learned from this assignment was ...

Sparse matrix algorithms do not necessarily provide better cpu usage in terms of number of computations per unit of time (GFlop/s) but can speed up performance by doing lesser number of computations.

One thing I am still not clear on is ...

How to explain the low SPMM performance numbers?

Why is SOA considered 'faster' than AOS? It should depend on how an algorithm iterates through the data. In our results we found AOS & SOA to be comparable in performance, with AOS achieving better GFlops/s than SOA in some instances.

Why SSPM (1 column) has worse performance as compared to the SSPV for different algorithms?