Homeworks 3-4: Sparse Matrix Vector Multiplication, parallelized

Stella Greenvoss
11/24/2025 — CS431
Computation done on ix-dev and ORCA

Function	Runtime	Speedup from serial	Speedup from max
COO Serial	2.05966	1	1
COO Parallel	0.09873	20.86264	20.86264
CSR Serial	1.58293	1	1.30118
CSR Parallel	0.05230	30.26799	39.38398

Table 1: Runtime and speedup for all functions. Computation done on ix-dev.

This paper details my experience attempting to parallelize sparse matrix vector multiplication using different methodologies and different storage formats. The first section describes the process of parallelizing for the CPU using C and OpenMP, with the storage formats COO and CSR. The second section discusses parallelizing for the GPU using CUDA and C++ with the storage formats CSR and ELLPACK. In this particular paper, my focus will be more towards describing my process towards parallelizing these functions (rather than an attempt at describing solid academic findings).

1 Part 1: CPU Parallelism

This section will describe serial and parallel versions of SPMV using both COO and CSR.

1.1 COO and conversion to CSR

COO, or *coordinate* format, stores sparse matrices using three columns: row, indices, and data. Each column has a total of nnz elements, where nnz is the number of nonzero elements in the sparse matrix.

Computing the matrix-vector product serially using a matrix stored in COO is simple[1] and runs in $O(nnz)$. Special care must be taken to ensure off-by-one safety, since the arrays stored in COO tend to be 1-indexed. As might be expected, this serial code performed the worst of the four functions. This is because much less work is being done per cycle, since a single thread must do every computation itself (assuming no compiler optimization). The data access pattern is also suboptimal, being linear for the row and col array accesses but without a clear pattern for r/w to and from the result and input, which negatively contributes to the temporal locality and thus the runtime.

The parallel version of COO-stored SPMV was also straight-forward, although there may be a more complicated implementation that performs better. Since I achieved a $20x$ speedup from the serial version, I decided that this would be sufficient for now. The algorithm is essentially the same, but with two parallel regions: one to initialize the result array and one to compute and add values to the result. To prevent simultaneous access, writing to a position in the result array in parallel must be done using an atomic pragma. It could also be done using an omp_lock, but my brief experiments with using that seemed to show that it was much slower.

Converting from COO to CSR was difficult since parts of it had to be parallel. I had never done a parallel histogram before, and had to use (and fix) my previous implementation of parallel prefix sum. For simplicity, I chose to use the simpler version of parallel prefix sum, which likely makes it overall less efficient than it could be.

[1]See *spmv_coo_ser* in homework3/main.c