
My own question: I think that, AOS should be faster than COO, but it turns out it's only faster when the matrix is small.

When the matrix is small, the memory locality of AOS is greater, because the indices, and the values are packed together inside a tuple.

The best explanation I can come up for a slower performance on AOS matrix vec is because of the extra memory overhead created by the tuple data type. Which needs more scrutinization.

---

1. 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).

The performance is way less compare to results obtained for dense matrix multipliction.

The best performance is `4.44748` GFLOPs/s, coming from CSR and CSC^T matrix vector multiplication.

Sparse matrix multiplication has memory usage of around O(N*l) where N is the size of the row and l is the maximum number of elements in that row. The computations needs to compute on all the double, which suggest lower memory locality compare to the dense matrix multiplication, decreasing the numerical intensity of the sparse matrix vector algorithm.

The sparse Matrix Vec has one int for indexing, 3 doubles for matrix vector multiplication, and 2 floating point operations inside the for loop.

Which is `2/(24+4) flops/byte`, 1/14. For dense matrix vector algorithm, it's `1/4 flops/byte`. I look back to PS4, and it suggests that a perfect use of L1 should get us to `6 flops/s`. And for dense matrix vector it should be capped at `47.2 Gflops/s` if L1 is used perfectly.

I make the statement that dense matrix vec is `1/4 Flops/bytes` because, not counting the running index, there are `(2N^2)` flops in total and there are `8*(N^2 + N)` bytes in total. Dividing then gives something slightly less than `1/4`

---

2. 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?

I took a look at the roofline graph from PS4, if make use of DRAM, then `1/14 flops/byte` should give around `1.1` flops/sec.

The dense matrix vector mulptication is about `(1/4)*14` times more intense compare to sparse matrix vector multiplication. Which suggest 2 times more performance, if no cache reuse.

---

3. 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).

COO and CSR asymptoptically approaches the same Gflops as the size `N` increases.

COO produces `2.24873` flops and CSR produces `4.44748` at `N = 64`.

CSR is 2 times faster because it has higher numerical intensity.

It hoisted the `column_nums_` out of the fastest running for loop, reducing the number of floats double during the computation from 3, to 2.

Therefore the numerical intensity will be given as: `2/16`, `1/8`.

However it turns out to be an under estimate on the speed improvement but close nonetheless.

---

4. How does the performance (in GFLOP/s) for sparse-matrix by dense matrix product (**SPMM**) compare 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.

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.