Assignment 3 - Sparse matrix formats on GPUs

Submission Deadline: Mo 29 November, 10am

Task 1: So far we learned about the CSR format. On CPUs this is a widely used standard format. However, it has some severe disadvantages on GPUs, but also on modern vector extensions (AVX, etc.) of CPUs. The paper Improving the performance of the sparse matrix vector product with GPUs by Vazquez et. al. describes these difficulties and also the alternative Ellpack and Ellpack-R formats that improve on the shortcomings of CSR on GPUs and offer more performance on these devices. Summarize the difficulties of the CSR format and how Ellpack and Ellpack-R solve these difficultues. For a very high mark look for some more modern papers on GPU based matvecs and describe what other modern developments have happened since that paper.

Task 2: Given a sparse matrix in CSR format. Write a CPU based Numba routine that converts this matrix into the Ellpack-R format.

Implement a new class `EllpackMatrix` derived from `scipy.sparse.linalg.LinearOperator`, which in its constructor takes a Scipy sparse matrix in CSR format, converts it to Ellpack-R and provides a routine for matrix-vector product in the Ellpack-R format. At the end the following prototype commands should be possible with your class.

```
my_sparse_mat = EllpackMatrix(csr_mat)
x = numpy.random.randn(my_sparse_mat.shape[1])
y = my_sparse_mat @ x
```

Print to PDF

The sparse-matrix vector product at the end shall be executed in the Ellpack-R format.

For your implementation you can either use CPU based Numba code using `prange` for multithreading or write an implementation in Numba-Cuda. For an overall mark of the assignment beyond 80% we would expect a Numba-Cuda implementation (assuming also all other parts are of high standard).

Demonstrate in your solution that your code provides the correct result by verifying for a 1000×1000 sparse random matrix with the standard CSR matvec of sparse matrices and showing for 3 random vectors that the relative distance of your Ellpack-R matvec to the CSR matvec result is in the order of machine precision.

Use the `discretise_poisson` method from the lecture notes to generate the sparse matrix for the Poisson discretization and plot the times for a single matvec for growing matrix-sizes (go as high as you think is reasonable) using the standard Scipy csr matvec and your own Ellpack-R implemementation.

Your implementation may not be faster since the matrix derives from a very simple 2d problem with few elements per row. This is not the situation where the additional complexitiy is usually needed.

Finally, go shopping in the Matrix Market and try to find two matrices that better show off the Ellpack-R format and do timing comparisions for your chosen matrices.

Previous
Assignment 2 - GPU Accelerated solution of Poisson problems

Next
Assignment 4 - Time-dependent problems