Monthly Summary: March, 2020

During the month of March, 2020, the GPU solution for PSCAD problems changed significantly. Instead of a solution that relied on dense matrix formats, the entire solution was rewritten in terms of sparse matrices and using the lowest level functions possible. As this summary will show, we have demonstrated a significant preliminary speedup that we will continue to develop over the coming months.

Low Level Functions

The previous approach to solving the sparse system $A\mathbf{x} = \mathbf{b}$ relied on CUDA Toolkit functions from the cuSolver and cuBLAS packages. While these functions were able to offer greater control over the factoring of the matrix A, they were not able to take advantage of the sparse nature of A. Therefore, a new type of solution was required.

The new solution was founded on functions described in the cusolverSp branch of cuSolver that is included in the standard CUDA Toolkit. An important advancement was the communication and factoring of the system matrix A using compressed formats. To do so, matrix A was described in a compressed format by the pointers `csrValA`, `csrRowPtr`, `csrColPtr`, and the integers `rowsA`, `colsA`, and `nnzA` and were copied to the GPU. Once there, the QR factorization took place on the GPU itself and entirely in sparse format. The factored matrix was kept on the device for the next stage of the solve. Next, a data file for $\mathbf{b}$ at a specific time step was read in and sent to the device. Once there, the function `cusolverSpDcsrqrSolve` solved for the vector $\mathbf{x}$ on the GPU. Finally, the result was then sent back to the host and the process repeated for the next time step.

Preliminary Timings

Table 1 is a summary of preliminary results from a subset of the *Province* data using a Debug build on laptop with an NVIDIA Quadro RTX 3000 GPU. That is to say, there are many caveats that should be kept in mind at this stage. With these in mind, we can see that using low level functions compatible with sparse methods has resulted in a 35x speedup in matrix factoring, and an 18x speedup in per-time step solving. We expect that these speedups will be further improved by running a Release build on the University of Winnipeg's servers.

CUDA API	**Process**	**Time (ms)**
cuSolver	QR Factor	~ 390
cuSolver	$A\mathbf{x}(t_0) = \mathbf{b}(t_0)$	~ 44
cuSolverSp	QR Factor	~ 11
cuSolverSp	$A\mathbf{x}(t_0) = \mathbf{b}(t_0)$	~ 2.4

Table 1: *Comparing preliminary timings for the main stages of the linear solver with either cuSolver or cuSolverSp APIs. Solving times are* **per time step**.