Monthly Summary: February, 2021

This month, the QRFactor library was tested with data from the *Province* data set in order to test for bugs or performance decreases due to the DLL format. Within the context of a client program, the factoring and per-time step solving times did not vary significantly from the times achieved by the previous versions of `QRFactor` .

Further functionalities were added to the QRFactor DLL, including a GPU check function that aborts the program if no GPU is detected on the system. Most significantly, functions to allow changes to the full system matrix values during the evolution of the system were added and tested in a client program. To achieve this function, a map container was added to the Coefficients class to store size and absolute position information from each subsystem as it was read in. The map key is the subsystem ID number, which is assigned during the first read-in of the subsystem. The map value corresponding to this key is a Subsystem structure that contains the absolute position of the (0,0) entry of the subsystem within the full system. Thus, the absolute position of any value is calculated by matching the subsystem ID and adding the relative position of the entry to the position stored in the map value.

Changing or removing elements from the full system matrix can be performed multiple times. Once the values in question have been changed, the system matrix must be reassembled and refactored before solving can continue. These functions are performed by `QRFactor::refactor`, after which the same `QRFactor::solve` function can be called repeatedly. The calling procedure from the client program is shown in § 1 below. The refactoring also converts the system matrix back into CSR form and frees memory that is not being used by the matrix.

With these additional methods, the QRFactor DLL now possesses all the functionalities that have been discussed up to this point. Unless there are additional functionalities that are required, it would be appropriate to begin integrating the QRFactor DLL into a test version of the PSCAD EMT software.

1 Changing Values Within the Full System Sparse Matrix

Consider a client program that is using the QRFactor DLL. We will assume that all subsystems have already been read in, that the full system sparse matrix has been created and factored (for more details on this procedure, see the December-January Summary document). Typically, some solving steps will have already taken place before any values within the system matrix change, but we will omit these for clarity.

```
#include "qrgpu.h"  // Header file for QRFactor library

int main()
{
    /*
    * Assume an instance of QRFactor has already been created
    * factored, and used to solve Ax=b. We now wish to change
    * some of the values in A.
    * As an example, set the (known to be non-zero) value in
    * position (0,0) of subsystem 2 to zero. Then, add two non-zero
    * values at (0,1) and (1,0) of subsystem 25
    */
    qr = QR_changeValue(qr, 2, 0, 0, 0.0);
    qr = QR_changeValue(qr, 25, 0, 1, 3.14159);
    qr = QR_changeValue(qr, 25, 1, 0, 3.14159);

    /*
    * All changes have been made, so rebuild and refactor A
    */
    qr = QR_refactor(qr);

    /*
    * Solving the linear system can now resume using the normal
    * syntax. This process can be repeated any number of times,
    * and can even change the total number of non-zero values in
    * the system
    */

    return 0;
}
```