COCS 594 – Scientific Computing for Engineers

Homework #11

(due April 22, 2020)

In the April 1 Homework you took a matlab prototype for orthogonalizing a set of vectors and coded it in C using calls to BLAS and LAPACK routines. Here we will use the NVIDIA GPUs to accelerate the computation. The most time consuming operations from algorithm chol_qr_it are (notations as in chol_qr_it.m; this is in case Q has #rows >> #columns)

Q = Q * inv(r) , and

G = Q' * Q.

We will accelerate the computation by performing these 2 operations on the GPU and the rest on the CPU. Namely, we will have Q residing all the time on the GPU's memory, 'r' will be sent to the GPU for performing the update Q = Q*inv(r) and the computed on the GPU G = Q' * Q will be sent back to the CPU. The algorithm is given in chol_qr_it_GPU (file chol_qr_it.cu) where the arguments of the main calls to the BLAS and LAPACK routines are erased. You can see how similar the implementation is to function chol_qr_it which is the same computation but performed entirely on the CPU (this is just an implementation of chol_qr_it.m from Homework 9).

The assignment is to fill up the missing arguments and produce a performance comparison graph (in Gflop/s) for the 3 algorithms (QR using LAPACK, chol_qr_it on the CPU, and the hybrid CPU-GPU chol_qr_it_GPU) on problems of sizes 2048x128, 4096x128, 8192x128, 16384x128, 32768x128, 65536x128, and 131072x128. Note that the code provided would compute the Frobenius and maximum norms of I-Q'Q and A-QR for the two chol_qr_it algorithms and that your implementation will be correct when the corresponding norms are comparable (report on the norms for the problem sizes listed above).

```
cublasSgemm( ... );                  // G = A' *A (GPU computation)
                                     //  both G and A are on the GPU
cublasGetVector( ... );              // G -> work (send the result G to the CPU)
sgesvd_( ... );                      //  [U,S,VT] = svd ( work );
mins = 100.f, maxs = 0.f;            // compute          S = sqrt( S )
for(k=0; k<n; k++){                  //                cond(S) = maxs / mins
      S[k] = sqrt(S[k]);
      if (S[k] < mins)  mins = S[k];
      if (S[k] > maxs)  maxs = S[k];
}
for(k=0; k<n;k++){                   // compute VT = S * VT
     vt = VT + k*n;
     for(j=0; j<n; j++)
        vt[j]*=S[j];
}
sgeqrf_( ... );                      // [q, r] = qr( VT )
if (i==1)
   scopy_( ... );                    // R = VT
else
   strmm_( ... );                    // R = r * R;
cublasSetVector( ... );              //  r –> G (send r in G on the GPU)
cublasStrsm( ... );                  // A = A * inv(G)
```