

**CS 594-004 – Homework # 9**  
**(due April 16<sup>th</sup>)**

There are two parts of this homework.

**PART I:**

The purpose here is for you to get some starting experience in using the PETSc library. As discussed in the “[Discretization of PDEs and tools for the parallel solution of the resulting systems](#)” lecture, PETSc can be used in solving large sparse linear systems that result from PDE discretizations. For example, we would like to use PETSc on the system described on slide # 19.

1. **You have to locate the PETSc library on your system (or install it if it is not there) and compile file ex2.c .**

Files ex2.c and an example Makefile are given. Look at the Makefile; it requires you to have variable PETSC\_DIR pointing to the root where PETSc is installed.

On my system for example I have

```
> setenv PETSC_DIR /home/tomov/petsc-2.3.1-p16/
```

The other variables used in the Makefile are defined in (which gets included)

```
 ${PETSC_DIR}/bmake/common/base
```

Having a good installation of PETSc and variable PETSC\_DIR should allow you to compile with just

```
> make ex2
```

2. **Study file ex2.c. It gives you an example on how to set our matrix in parallel and call PETSc's solvers and preconditioners. File test.c is a similar example but the matrix is read from a file.**

Note that you can set specific solvers and preconditioners by modifying ex2 (through correspondingly the ksp and pc objects) but you can also set them with options at run time. Running for example

```
> mpirun -np 1 ex2 -help
```

will give you all the runtime options.

3. **Report runtime, number of iterations, and norm of the errors for**
  - \* **at least 3 solvers (cg, gmres, tfqmr)**
  - \* **on problem sizes 200x200, 400x400, and 800x800**
  - \* **on 1, 2, and 4 processors.**

To report the errors we use a random vector  $y$ , compute  $b$  as  $A y$ , solve  $A x = b$  and report the 2 norm of  $x - y$ . This is all set in ex2 so a run on a problem of size

100x100 with a gmres solver would look for example something like  
> mpirun -np 4 ex2 -ksp\_type gmres -m 100 -n 100 -random\_exact\_sol

## **PART II.**

In Homework 6 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 myqr\_it are (notations as in myqr\_it.m; this is in case Q has #rows >> #columns)

$Q = Q * \text{inv}(r)$  , 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  $Q = Q * \text{inv}(r)$  and the computed on the GPU  $G = Q' * Q$  will be sent back to the CPU. The algorithm is given in myqr\_it\_GPU (file myqr\_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 myqr\_it which is the same computation but performed entirely on the CPU (this is just an implementation of myqr\_it.m from Homework 6).

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, myqr\_it on the CPU, and the hybrid CPU-GPU myqr\_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 myqr\_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).