COSC 594 homework – dense linear algebra

Due Wed, Mar 11, 2020

1. (10 pts) Given these two random matrices:

```
A1 = [
   0.402616844233997  -0.540310355415576   0.363228620941588
  -0.934132984522105   1.253603286723906  -0.842746441301808
   0.268024766095426  -0.359688420873504   0.241803913458064
]

A2 = [
   0.451322112370230  -0.528303260602689   0.327102613488652
  -0.930885422784320   1.254404142018829  -0.845154875946141
   0.206180105805249  -0.374933846900485   0.287677048422013
]
```

   In Matlab, Python, etc., generate a random 3×1 vector x_0.

   For each of the matrices A1 and A2: Use x_0 to generate a right-hand side $b = Ax_0$. Now x_0 is nominally the exact solution to $Ax = b$. Solve $Ax = b$. Compute the relative forward error,

$$\frac{\|x - x_0\|}{\|x_0\|},$$

   and the relative backward error (residual),

$$\frac{\|b - Ax\|}{\|b\|}.$$

   Compare the entries of the computed x and x_0. Explain the results for each of A1 and A2.

   Turn in your Matlab, Python, etc. script and the output of computed quantities.

2. (90 pts) Implement parallel matrix-matrix multiply using SUMMA and MPI.

   Include a test check against a reference implementation such as ScaLAPACK's `pdgemm` to verify that your implementation is correct. The relative forward error should be small:

$$\frac{\|C - C_{\text{ref}}\|_{\text{fro}}}{\|C_{\text{ref}}\|_{\text{fro}}} < \text{tol} \cdot \epsilon$$

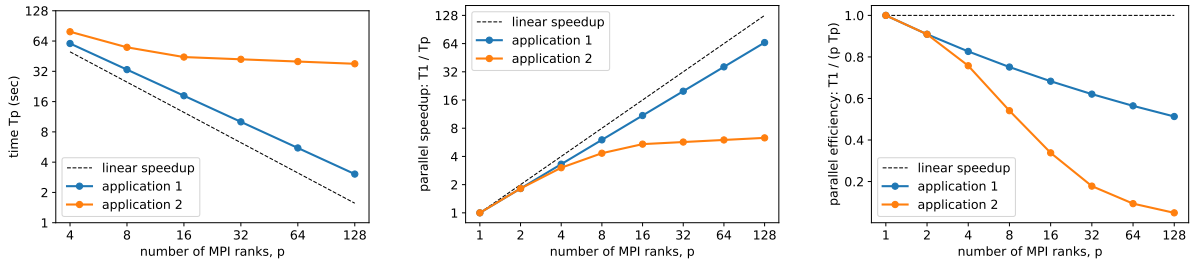
   where the tolerance tol is some small constant like tol=10, and machine epsilon (ϵ) is given in the slides. There's a more rigorous, complicated error formula for gemm, but this simple formula should be sufficient.

   You may make simplifying assumptions, e.g.,

   - $C = AB$ (NoTrans, NoTrans, alpha=1, beta=0 case)
   - Block distribution, that is, one block per MPI rank (as opposed to 2D block-cyclic)
   - Convenient dimensions, such as n divisible by block size nb

   You may use C, C++, or Fortran. Use optimized BLAS `dgemm` (e.g., from OpenBLAS, BLIS, or Intel MKL) for the local matrix multiply.

   - If using Fortran, use the traditional Fortran `dgemm`.
     See https://software.intel.com/en-us/mkl-developer-reference-fortran-gemm



(a) time in log-log scale (b) parallel speedup in log-log scale (c) parallel efficiency in semilogx scale

Figure 1: Strong scaling results for fictitious applications.

- If using C, I recommend using `cblas_dgemm`.
  See https://software.intel.com/en-us/node/520775
- If using C++, use either `cblas_dgemm` or BLAS++ `blas::gemm`.
  See https://bitbucket.org/icl/blaspp/
  and https://icl.bitbucket.io/blaspp/doxygen/html/group__gemm.html

Set the number of BLAS threads to the number of physical CPU cores per MPI rank. Don't use hyperthreads. For instance, if you run one MPI rank per node and a node has 16 cores, set threads=16. If you run one MPI rank per core, set threads=1. Most BLAS libraries use the OpenMP number of threads:

```
bash:  export OMP_NUM_THREADS=1
tcsh:  setenv OMP_NUM_THREADS 1
```

Turn in your source code, Makefile, and a report including performance graphs and discussion of the questions below. Describe your implementation choices and discuss your performance results.

(a) (40 pts) Source code and Makefile.

(b) (10 pts) Test and plot various matrix sizes, e.g., n = 1000 to 20,000. Compare to the theoretical peak performance of the machine.

(c) (10 pts) For some fixed large size, test and plot how well your implementation scales with different numbers of MPI ranks, e.g., 1, 4, 9, 16. SUMMA should work for any number of MPI ranks, whereas Cannon and Fox would require square grids. Plotting either time (fig. 1(a)) or parallel speedup (fig. 1(b)) in log-log scale will show a straight line for perfect linear scaling. Parallel efficiency (fig. 1(c)) is another way to view this data.

(d) (10 pts) Compare results with an optimized implementation such as ScaLAPACK's PBLAS `pdgemm`.
See https://software.intel.com/en-us/mkl-developer-reference-c-p-gemm

(e) (10 pts) In terms of the matrix size n, block size nb, and number of ranks p, how much data is communicated at each step? Consider a broadcast to r ranks as r point-to-point communications.

(f) (10 pts) How much memory is required by each MPI rank?