Direct and Iterative Linear Solvers

As part of your BabyBLAS assignment you have to code direct and iterative linear solvers to solve the linear set of equations $\hat{A}\vec{x} = \vec{b}$. Writing fault-tolerant linear solvers has traditionally been a problem for students. So that you can focus on the parallel code development, I have written a set of serial direct and indirect linear solvers for you. The direct solver checks to see if the matrix is strictly diagonally dominant and, if this is the case, does a straight LU decomposition with no row interchanges. If the matrix $\hat{A}$ is not diagonally dominant then Gaussian elimination with partial pivoting is used to solve the system. This process should only fail in cases where a diagonal element becomes zero during the reduction phase. For the iterative case all of the diagonal elements have to be non-zero. The program first checks for this and if if finds a zero along the diagonal it switches to the direct solver. If this test is passed the Jacoby iterative algorithm is used to solve the system. There is no guarantee that the system is solvable with the Jacoby method unless the spectral radius (a property tied to the eigenvalues of $\hat{A}$) meets certain criteria. For that reason the method will terminate either after it achieves convergence or a certain number of iterations are completed. In the case where the maximum number of iterations is used the program will then pass the original system to the direct solvers to complete the job.

Preparation:

As today you will ONLY be examining the serial aspect of the code, you will not need to log into hammer. You can work locally on the linux workstations in the classroom. In other words, you can log onto hammer if you want to, but you won't need it today and the calculations might run faster locally. So…

Clone (or pull) the repository:

```
https://github.com/threadmaster/bblas_starting.git
```

to a directory you own. The is the same repository you cloned the other day so you might can do a "git pull" to just recover the new stuff I added. Once you have cloned the repository, do a

```
cd bblas_starting
```

and then do a

```
git checkout linearsolvers
cd serial
```

A quick look at the dls function:

Open the file dls.c in the serial directory.

1. Skip over the first portions that are needed for inter-language linkage and the function prototype.
2. The dls.c function first checks the matrix for diagonal dominance. You could improve the performance (minimally) by writing the diagonal dominance check in parallel. The code is found at the end of the file.
3. The remainder of the code utilizes numerous looping regions. Take some time and think about the best way to parallelize this code as this will undoubtedly be the most time consuming thing you do. Remember, I am not asking you to dive into the linear algebra to try and come up with more efficient methods linear algebra algorithms. Your goal is to take this code and do what you can to make it more efficient using parallelism.

A quick look at the ils function:

Open the file ils.c in the serial directory.

4. The first thing that you will notice is that there are three prototypes in this function, one to call dls, one to check for zeros along the diagonal of the matrix, and one to check for convergence. The check for zeros and the check for convergence are potential candidates for parallel code development.
5. The workhorse of the iterative solver code is the while loop that iterates until convergence is met. Classically, these kernels are where iterative solvers get their performance gains. See what you can do to make it better -- but stick to the Jacoby Iterative Algorithm for solving the linear system.

Makefile Adjustments:

I have already made the corrections to the makefile in the serial directory.

Driver Program:

I made many changes to the driver.f90 program code to allow for testing of different types of linear systems as well as the different solvers. The code will now allow you to select if you want to test matrix multiplication or the linear solvers. Do a

```
cd ..
vi driver.f90
```

If you look at the code you will notice that there are MANY more #ifdef clauses and I also added significant amounts of code for accessing online matrices or for building specific types of matrices. By default the program attempts to test matrix multiplication

The following options are available at compile time

-DACCURACY_TEST

This runs the accuracy test code and pulls information from the website

-DLS_TEST

This instructs the program to test the linear solvers, by default the direct linear solver is used

-DITERATIVE

This instructs the program to test the iterative linear solver

-DDIAGDOM

This instructs the program to build diagonally dominant matrices for the linear solvers

-DSPARSE

This instructs the program to build sparse matrices for the linear solvers

Let's try some tests. Do a

```
cd ..
vim Makefile.inc
```

And modify the FFLAGS variable to

```
FFLAGS = -cpp -O2 -DACCURACY_TEST
```

And then…

```
make clean; make; driver
```

This should have tested the matrix multiplication code as we did before. Now modify FFLAGS again

```
FFLAGS = -cpp -O2 -DACCURACY_TEST -DLS_TEST
```

And then…

```
make clean; make; driver
```

This will test the direct linear solver. To test the iterative solver…

```
FFLAGS = -cpp -O2 -DACCURACY_TEST -DLS_TEST -DITERATIVE
```

Followed by a…

```
make clean; make; driver
```

The iterative solver will fail on this test and should revert back to the direct solver. To properly test the iterative solver, we need to tell the program to use a diagonally dominant system. Modify the FFLAGS variable in the makefile as such:

FFLAGS = -cpp -O2 -DACCURACY_TEST -DLS_TEST -DITERATIVE -DDIAGDOM

Followed by a

make clean; make; driver

The iterative solver should complete this time. Look carefully at your results and look at the code. Do the iterative and direct solvers give the same values? Any idea why?

Based on the type of test you are doing, the information printed in the second column of the output will vary. If you are doing matrix multiplication, it should print the sum of the diagonal elements of the matrix which, based on the way the matrices were constructed, should equal the dimension of the matrix. If you testing the linear solvers, then the value printed should equal the largest absolute difference between any one of the true solutions to the the system and the result computed by the program. For example, if the following table represented the results of using the one of the linear solvers

Actual Solution	Estimated Solution	Absolute Difference
1.00	1.01	0.01
2.00	2.11	0.11
3.00	3.05	0.05

Then the second column of the output would be 0.11. Remember, the linear solvers return a vector (column 1) as a solution. The program searches the vector and compares it to the actual result.

Performance as a function of Matrix Type and Method

For the linear solvers turn OFF the accuracy testing by not including the directive in the compile flags and then. For each of the following tests, make the noted changes to the FFLAGS variable in Makefile.inc and then build and run the program.

Test 1: -DLS_TEST
make clean; make; driver 1000 3000 200 1

Test 2: -DLS_TEST -DDIAGDOM
make clean; make; driver 1000 3000 200 1

Test 3: -DLS_TEST -DSPARSE
make clean; make; driver 1000 3000 200 1

Test 4: -DLS_TEST -DDIAGDOM -DSPARSE
make clean; make; driver 1000 3000 200 1

Test 5: -DLS_TEST -DITERATIVE
make clean; make; driver 1000 3000 200 1

Test 6: -DLS_TEST -DITERATIVE -DDIAGDOM
make clean; make; driver 1000 3000 200 1

Test 7: -DLS_TEST -DITERATIVE -DSPARSE
make clean; make; driver 1000 3000 200 1

Test 8: -DLS_TEST -DITERATIVE -DDIAGDOM -DSPARSE
make clean; make; driver 1000 3000 200 1

In the Following table, record your results using your largest system (dimension=3000)

TEST #	**ITERATIVE**	**DIAG. DOM.**	**SPARSE (Y/N)**	**SUCCESS (Y/N)**	**CPU TIME (SEC.)**	**MFLOPS**
1	N	N	N			
2	N	Y	N			
3	N	N	Y			
4	N	Y	Y			
5	Y	N	N			*USE PAPI*
6	Y	Y	N			*USE PAPI*
7	Y	N	Y			*USE PAPI*
8	Y	Y	Y			*USE PAPI*

Based on comparing runtimes and results, what can you say about the speed and accuracy of the direct vs. iterative solvers as a function of MATRIX TYPE?