# 1 Background

Parafos was begun as a side project by me (Jeff Heys) at the University of Colorado at Boulder when I was in the department of applied mathematics. I had the following objectives (priorities):

1. first-order systems least-squares (FOSLS) formulations,
2. fully parallel on distributed memory machines – this was accomplished using MPI and *hypre* such that the machine can run on any number of machines (in theory from 1 to 1000's),
3. 3-dimensions,
4. higher-order finite elements – the code can use any tensor product basis (i.e., hexahedral elements) of order $2^n$ (i.e., trilinears, triquadratics, triquartics, $8^{th}$-order, etc...) with full isoparametric mapping, and
5. simple, text-file, based input and output interface that can currently read/write OpenDX files (also means are provided for converting cubit meshes to OpenDX files).

The first objective, the use of FOSLS or least-squares finite element methods LSFEM is probably the most unique aspect of the code. These methods hold the promise of solving Navier-Stokes and elasticity equations with *optimal computational scalability* independent of time step size, but they have other difficulties, such as conservation and accuracy is some norms. This code was developed to test least-squares on big, real world problems. It is NOT user friendly, and new capabilities are being added every day. If you would like to try the code, you should email me (jheys@asu.edu) to discuss the physical problem you are trying to solve.

# 2 Compiling

Parafos requires *hypre* from CASC at LLNL and Metis for mesh partitioning. The location of *hypre* and Metis should be set in the makefile. To compile, first type 'make clean' to remove all the old .o files. Next, type 'make' to build parafos, 'make partition' to build the partitioner code, 'make collect' to build the code to collect output files into a single file, and 'make cubit2dx' if you wish to convert Exodus mesh files (in ASCII) from the Cubit mesh generator to parafos input files.

# 3 Partitioner

The partition program is primarily designed to partition a finite element mesh into a set of subdomains, each stored in a separate file. The program is run by typing 'partition #proc source(=1 or 2) ....'. The first number following partition is the number of processors to partition the mesh between. The second number is set to 1 to use the built in mesh generator, which is simple generator that subdivides a cube into hexahedral elements. If the built in mesh generator is used, the '1' should be followed by the number of elements in each direction and the prefix name of the output files. For example, 'partition 8 1 16 32 64 mesh' will partition a 16 by 32 by 64 mesh among 8 processors, and the 8 output files will be named mesh#.dx. These mesh files can be visualized using OpenDX. The second number is set to 2 to input a mesh from cubit and is followed by the names of the input file and output file prefix. For example, 'partition 8 2 cubmesh mesh' will partition the mesh cubmesh.dx among 8 processors, and the output files will be named mesh#.dx. One of the limits of the existing code is that the mesh must be partitioned on a single processor, which limits the size of the global mesh. However, mesh storage is not expensive so this limitation has never been realized in practice.

# 4 Collector

The collector collects a series of output files into a single file. The command format is 'collect #proc prefix'. For example, 'collect 12 out' will collect the files out0.dx through out11.dx into a single file named out.dx.

# 5 Parafos

This program must be run using mpirun (or equivalent) command, and it begins by reading in the 'input.dat' file. My hope is that the only changes the user will ever need to make can be made in input.dat and recompilation can be avoided. Of course, this will never be the case for complex problems. Let's look at each line of input.dat:

1. The problem can be set to 'laplace', 'ns', 'vort', 'newvort', or 'verde'. Laplace is the FOSLS form of Laplace's equation, which has 4 unknowns, 7 equations in the functional, and 4 weak boundary conditions. "ns" stands for the Navier-Stokes equations. This is the original FOSLS formulation using velocity gradients, and it has 13 unknowns, 25 functional equations, and 20 or so weak boundary conditions. This formulation does not work very well. The "vort" problem is the widely used velocity-vorticity formulation of the Navier-Stokes equations. This formulation has 7 unknowns, 8 equations in the functional, and about 5 weak boundary conditions. The final possibility is "newvort", and it is a new vorticity formulation developed by Tom Mantueffel. It has 9 unknowns, 12 equations in the functional, and about 10 weak boundary conditions.

2. The second line is the prefix of the mesh file, i.e., if the prefix is 'mesh' there should be a file named 'mesh#.dx' for each processor #. This string should be the same as the final string passed into the partitioner.

3. The number of quadrature points to use for integration of the elements. Set to 2 for full integration or 1 for reduced integration on trilinears. Solver performance is significantly affected by reduced integration so always set this to 2.

4. The location of the quadrature points. Can be 'gauss' for Gauss quadrature, or lobatto for the chebyshev-lobatto points.

5. The number of weak boundary conditions in the functional. This line should also be set based on the problem chosen in line 1. Setting it too high will not cause problems, but too low will cause problems.

6. The maximum number of iterations. Set to 1 for a linear problem, or more than one for nonlinear problems.

7. Iteration tolerance is the relative change in the functional before stopping iterations in nonlinear problems.

8. The code is currently capable of global $p$-refinement, i.e., refining trilinears to triquadratics to triquartics etc... Typically, this should be set to zero to prevent refinement, but if higher accuracy is desired, set it to 1 or 2. Warning – large amounts of memory are needed if this is set to 2 or higher.

9. The maximum number of AMG iterations.

10. The tolerance to stop the iterative solver (i.e., AMG/CG).

11. The print level for the solver.

12. The type of wrapper to use on AMG – 0 is no wrapper, 1 is CG, and 2 is GMRES.

13. The strong threshold used by AMG.

14. The rest of the values are unused except the final line, the Reynold's number.

Parafos has a few output files. First, fun.txt contains information about the functional. Next, each processor outputs solution information based on the settings in the 'dx_out' function and 'dx_vort' function. All problems are currently set up to solve on a $4 \times 1 \times 1$ domain. Any other domain will require changing the set_bc.c functions and the solve() function in solve.c. You may want to consult with me before attempting these changes.