User's Manual for Diablo

Jason Hicken

September 18, 2009

Contents

1 Getting the Code **1**
1.1 Cloning the Code using Git 1
1.2 Compiling the Code 2

2 Creating and Using Grids **2**
2.1 Grid Utilities 3

3 Input Parameters **3**
3.1 logical-type paramters 3
3.2 character-type paramters 3
3.3 integer-type paramters 4
3.4 real-type paramters 5

4 Output Files **6**

1 Getting the Code

1.1 Cloning the Code using Git

We use git for version control, so, to get started, you need to tell git about yourself:

```
git config --global user.name "Your Name Comes Here"
git config --global user.email you@yourdomain.example.com
```

To get a copy of jetstream, begin by creating a bare repository based on another lab member's public repository. For example, to clone a copy from Jason Hicken, `cd` into your home directory on oddjob, and use the command

```
git clone --bare ~jehicken/jetstream.git jetstream.git
```

This bare public repository will be the place that you "publish" your code changes.

Next, login into your supercomputer of choice (e.g. hpacf, scinet). From there, you will clone your bare repository off of oddjob:

```
git clone ssh://oddjob.utias.utoronto.ca/~yourusername/jetstream.git
<directory name of your choice>
```

This time, because the `--bare` option was not used, you will get a working directory of the code in the directory you created. Before you proceed with code development, you should probably create a new working branch and check it out (see the git manual).

1.2 Compiling the Code

Before you compile diablo, it is necessary to compile the external libraries. First, change directories into the METIS-4.0 directory and type make. Do the same for the SPARSKIT2 directory. Next, enter the SNOPT directory and type `./configure`, which will automatically generate a GNUmakefile. Some of lines in the GNUmakefile must be changed before compiling SNOPT:

1. change the line starting with FC from g77 to the desired Fortran compiler on your system (e.g., ifort).

2. on the line starting with modules, insert a `#` symbol just before snadiopt, which ensures that `src` and `examples` are the only two modules compiled.

After these changes, you can type make to compile SNOPT. Finally, enter the LAPACK directory and type make; this can take a while; however, you can kill (`ctrl-c`) the compilation once it reaches the subroutines dealing with testing.

Compiling diablo should be straightforward. Just `cd` into the main code directory, and type `make`. This should produce a binary execuable file named `jetstream_<?>` where `<?>` is an extension refering to the architecture. If you want to change the name of the executable, change the `TARGET` variable in `Makefile`.

2 Creating and Using Grids

We use plot3D format grids. These are binary files stored using Fortran records (you cannot easily read/write these files in another language). In the long term, we may switch to another grid format (HDF, CGNS).

To create a grid from scrath, we use ICEM. Read the tutorial on making HEXA meshes. Once you have created a multi-block grid that you want to use, simply export it using plot3D as the solver.

Connectivity files are a bit more tricky. The best option is to define various surfaces in the ICEM geometry file, and then assign boundary conditions to those surfaces. Probably best to talk to one of the more experienced lab members about how to do this. Alternatively,

I have created a grid utility called create_connect that automatically creates the connectivity file; however, it does a greedy search, so it can be slow. In addition, it uses some heuristics to decide if a boundary subface is a solid surface, symmetry plane, or far-field: these heuristics are not guaranteed.

2.1 Grid Utilities

Often, it is necessary to manipulate the grids. Examples include, removing every other grid line (coarsening), splitting blocks, rotating blocks, quality checks, grid information, appending grids, rearranging blocks, etc.

Previous students have written a number of grid utilities, so before you code up your own, you may want to ask one of the vetrans if there is an existing utility. To get started, copy my `grid_utils` directory into your home directory. Eventually, some of these should be distributed with the git repostitory.

3 Input Parameters

The behaviour of the solver is controlled by the options in the input file (the input file is called input.param by default). Sorry for the terse explanation...more to come.

3.1 logical-type paramters

restart: is this a restart?

viscous: is the flow viscous?

p_switch: is the pressure switch on?

wakedrag: is the drag calculated using the wake plane?

wakelift: is the lift calculated using the wake plane?

frz_coef24: are the dissipation coefficients frozen?

calc_onera_cp: calculate the onera cp distribution

3.2 character-type paramters

strtup_prec: start-up phase preconditioner (schur, lusol)

strtup_matv: start-up phase matrix-vector product (apprx, jacbn, frcht, cmplx)

newton_prec: inexact-Newton phase preconditioner (schur, lusol)

newton_matv: inexact-Newton phase matrix-vector product (apprx, jacbn, frcht, cmplx)

3.3 integer-type paramters

pcoef_order: pressure switch for different orders

fourth_deriv: (1=Dieners, 0=Svards 4th derivative)

jac_order: not used presently

int_order: not used presently

verify: type of order-of-accuracy verification

MMS: method of manufactures solutions

MMS_DIM: 2 or 3 dimensions

calc_lift_drag: calculate lift and drag on a restart

order: derivative order of accuracy (2nd, 3rd, 4th)

jac_meth: method used to calculate metric Jacobian (1,2,3)

fjac_order: order of flux jacobian (1st,3rd,4th)

diss_order: dissipation order (2nd, 3rd, 4th)

idmodel: type of dissipation model (1,2,3,4)

istream: coordinate direction of streamwise flow (xyz)

iground: coordinate direction of vertical direction

reorder: type of reordering to use

updat_frq: number of iterations between approx. Jac. updates

nk_global: globalization method

nk_strtup: type of start-up for pseudo-transient

nk_time: sets how the reference time step is calculated

nk_pfrz: iteration at which to freeze the preconditioner

nk_its: max number of Newton-Krylov iterations

pc_its: max number of homotopy predictor-corrector iters

lev_fil: fill-level used for the preconditioner

app_lev_fil: fill-level for approx. newton phase

gmres_its: GMRES iterations used during the linear solve

schur_its: maximum number of approximate Schur iterations

3.4 real-type paramters

r_trans_per: Dissipation r transition percent

newton_relax: Relaxation parameter - newton phase

strtup_relax: Relaxation parameter - start up

dt_min: minimum time step used during start-up

dt_max: maximum time step to use during start-up

A, B: terms in the geometric time step for start-up

beta: the exponent used for the reference time step

alpha: the factor used for the reference time step

lambda: parameter used for SAT-based continuation

rel_tol: relative tolerance with which to converge solver

abs_tol: absolute tolerance with which to converge solver

prec_dlf: prec. 4th-difference dissipation lumping factor

drop_tol: drop tolerance at which to switch to full Newton

pert_tol: tolerance for sub-problems when using homotopy

strtup_tol: tolerance before switching from quasi-Newton

newton_tol: linear solver tolerance once switched to Newton

schur_tol: approximate schur preconditioner tolerance

dis2(3): 2nd-difference dissipation coefficient

dis4(3): 4th-difference dissipation coefficient

Vl: spectral radius scaling for linear waves

Vn: spectral radius scaling for nonlinear waves

wkpt(3): point in the wake plane

4 Output Files

<prefix>.q Flow variable solution file. Visualize the solution in Tecplot using this file together with the input grid file.

<prefix>.solv A verbose output file describing the convergence history of the solver.

<prefix>.his A condensed version of <prefix>.solv. Each line is a major NK iteration, and this file can be used in Tecplot to plot convergence history.

<prefix>.cp Coefficient of pressure distribution. Not yet available for general geometries.