Beverley,

In http:torroja.dmt.upm.es/ftp/.ctr I have put a tar file with the

code and associated files (ROUGHv7.tar)

and a separate restart file, ctr2p1p.013 (208 MB), that you can use

to start your run.

Please note that this is a hidden link that you have to type

explicitly.

The restart is finally a 2\*pi x pi box, at Retau approx 510. The grid

is enough for Retau=600-630, but my experience is that, once you start

playing with the bc, Retau may easily change by 20-30%, so I left some

leeway. If your Retau falls or climbs too much, the easiest way to

control it is to change the viscosity in the parameter imput file

hre.dat (see below). However, please note that every time you change

the parameters, you should leave about one eddy turnover time (T\_turn

= h/utau) for the flow to settle down. The computational units are

normalized such that (h=1, U\_bulk \approx 0.9), so that utau \approx

0.05 and T\_turn \approx 20 in computational units. I have included in

the tar file the output (\*.cf) from the run I did to shrink the box,

to give you an idea of how things settle down. Most of the columns are

useless, but column 1 is the time and C4 is retau. With the present

smooth wall configuratio, grid and CFL, ideltat \app 0.004, and one

T\_turn \approx 5000 time step. Those were taking about 10.5 s/step in

24 procs. of my old cluster (Xeon 3000 Myrinet), and about half that

in a newer one (Harpertwon Infiniband). One turnover is therefore

about 15 (8) hours of clock time in 24 nodes, or 200-350 CPU

hours. Empirically, you should accumulate about 10 turnovers to get

converged statistics.

Note that these numbers can change a lot once you start playing with

the boundary conditions. In particular, if you introduce substantial v

component (wall-normal) at the wall, where the grid spacing is fine,

your time step will plummet.

The restart file is a fortran direct-access binary file in IEEE little

endian format. It is written by the subroutine escru, and read by

getfil. Both in the source file main.f.

The first thing that you should do is to check whether the code

compiles and runs at the ctr as it is. I have included a sample

makefile (make ROUGHv7). You will probably have to change the compiler

to whatever is installed at the ctr, and the optimization

flags. Please note that this a fairly old fortran file, written

partly in cowboy fortran. Parts of it will not work with bound checkers

(-CB in ifort). They write out of bounds on purpose. Parallelization

is MPI, and most of it is in the global transpose contained in

change.allfac.f.

Please cite for this code

``Effect of wall-boundary disturbances on turbulent channel flows'',

O. Flores \& J. Jim\'enez, {\it J. Fluid Mech.} {\bf 566}, 357--376

(2006)

There are some files that you need to know about.

---------------------------- ctes3D -----------

This include file contains the grid dimensions, NUMBER OF NODES, and

other static parameters. The code has to be recompiled every time you

change something here. Makefile knows that, but, just in case, do a

(rm \*.o) every time you change ctes3D.

Main things you may want to change here are

parameter(numerop=24)

Number of processors to use. This HAS to agree with the number of

procs you ask for in your mpirun (see sample script in dum.sh for

PBS). Otherwise the code will stop on entry.

parameter(blockingik2ki=64) !!! divisor of mgalz !!!

parameter(blockingki2ik=256) !!! divisor of mgalz !!!

These are technical parameters having to do with cache control in

transpose. They should be adjusted for each computer by trial an

error, but they will only result in slower execution if you do it

wrong (probably not by much). HOWEVER, they have to divide the mgals,

as said in the comments. Otherwise, the results could be anything.

parameter(mgalx=512,mgalz=512,my=232)

These are the number of collocation points in x and z (please use

reasonable fft values), and the number of point in y (This is finite

different, so that there are no restriction on this value). As I told

you, the values now are ok for about Retau=600.

parameter(nspec=12)

dimension jspecy(nspec)

c wall units (re630): 3, 6, 15, 32, 52, 76,122,165,251,335,425,512 + 630

data jspecy/ 6, 9, 16, 25, 32, 39, 51, 60, 76, 89,103, 115/

These are the planes at which spectra are compiled and output in the

\*.spe files. The jspecy are plane numbers on the lower half

channel. The code computes the corresponding planes in the upper half,

and accumulates both halves. The actual number of spectral planes is

nspec+1. The central plane is always included. If you give it in

jspecy, it will be output twice.

parameter (iwd=1) !! measured in 4\*byte words

Most fortran compilers now measure the record length of direct acces

files in 4-byte words (iwd=1). If you use a compiler that measures

length in bytes, use (iwd=4).

--------------------------- hre.dat -------------------------

This is the input parameter file. It can be changed without

recompiling the code. Lines starting with CC are ignored. The sample

included is the file used to create the restart ctr2p1p.013.

CC-----------------------------------------------------------------------

CC Re alp bet -u at

CC Reynolds x wave # z wave # the walls

CC 1/viscosity Lx=2\*pi/alp Lz=2\*pi/bet (see readme)

CC

CC \* \* \* \*

10000 0.5 1.0 0.53

The only parameter that is not self explanatory is the last one. The

flow is computed in a frame of reference moving forward with this

speed. The wall is therefore moving backwards. This reduces the

absolute value of the velocity wrt the grid, and increases your time

step (for the same accuracy) by about a factor of two over smooth

walls. If your transpiration velocities are high at the wall, this

parameter will do little (although it should still improve the

accuracy of the streamwise terms). In those cases, your time step will

be limited by the transpiration velocity.

CC

CC total steps #steps #step

CC

CC (nstep=k\*nimag+1) (write a restart (compute and

CC file every nimag) output cf every nhist)

CC nstep nimag nhist

CC

CC \* \* \*

5001 1000 10

The only thing to note is the extra 1 in the time steps. The code

writes the output file at the beginning of the next step. So if you do

1000 steps, and request a new restart file every 500 steps, the last

file will not be written (and your last 500 steps will be wasted).

IMPORTANT: The \*.cf file is written every nhist steps, but those are

also the times at which the time step is recomputed to keep the cfl

under control. It is important that nhist should not be too large!!!

CC

CC mesh type: mesh (don't touch this too much!!)

CC 0:uniform parameter:

CC 1:tanh

CC 2:sin gamma

2 0.93

CC

CC CFL

CC \*

1.5

This is probably the maximum that you can afford before the time accuracy

is shot.

CC

CC first 0/1 # steps

CC output 0 do nothing between

CC file 1 compile statistics

CC number stat file ntimes

CC id22

CC \* \* \*

14 1 50

The code creates restart and statistics files with serial numbers

starting with id22. In this example it will create

ctr2p1p.014 ctr2p1p.014.sta etc (and then)

ctr2p1p.015 ctr2p1p.015.sta etc ...

Be careful not to overwrite accidentally your input file !!!

The second parameter controls whether the code will create statitics

file (umean, urms, etc, see escru) and spectra, or not. It should

probably always be 1 except if are testing for bugs or measuring

times. Statistics and spectra are accumulated every ntimes steps, but

written only at the same times as the restart files. A new \*.cf \*.sta

and \*.spe file is written for every restart. It is up to you to

average them together when you are through running. Probably after

throwing away the first turnover.

CC

CC

Boundary parameters

CC

CC

mx mz uampl vampl wampl phase speed

CC

72 72 0. 0. 0. 0.

CC

This should be irrelevant if you change the bcs to something else. As

of now, it is saying that you want to put something the 72th

wavenumber in both x and z, with zero amplitude (so, a smooth wall),

moving with phase velocity with respect to the wall (phase

speed). This speed knows about the numerical speed offset in the first

line (you don't have to take it into account). See cross.f for

details of the bc (see below)

CC-----------------------------------------------------------------------

CC 3)FILE NAMES used in the code

CC-----------------------------------------------------------------------

CC file output max 100 char.

CC (the code will add serial number, but give the full path)

ctr2p1p

CC

CC

CC input file max 100 char. (this is the restart file, give full path)

CC

ctr2p1p.009

CC

CC statistics file name max 100 char.

CC (the code will add file types .sta, .cf and .spe +serial)

ctr2p1p

CC

CC

-------------------- BOUNDARY CONDITIONS -----

Since I don't know what you want to do here, I have left what Flores

used in his last run, which is to set one wavevector to a given fixed

amplitude for the three components (in the restart file, these

amplitudes are zero). The way that things are implemented now also

allows for an advection velocity of the forcing. The code creates

three full Fourier planes (uwall, vwall, wwall), that you can set to

anything. They correspond to the lower wall. The boundary conditions

at the upper wall are currently set symmetrically

(vtop=-vbot, omegaytop=omegaybot, dvdytop=dvdybot).

You can also change that to anything, but if you do something fully

nonsymmetric, you would have to change the way that the output

statistics and spectra are added for symmetrical planes. This is done

in cross.f, look for things with a comment BCS-HERE.

NOTES:

1) Setting each wavenumber implies setting two wavevectors, (kx,kz)

and (kx,-kz). In principle they could be set to very different things,

although probably you want to set them to something similar. THE

EXCEPTION is kx=0, where (0,-kz) should be the complex conjugate of

(0,kz) for the velocity to be real. If you don't do it, the code will

do something by itself, but I am not sure what.

2) The same thing has to be done twice. The first step after reading

the restart is done differently from the others, and the computation

of v is repeated. This means that vwall has to be created in this

first step consistent with what you are doing elsewhere. This is at

the top of cross.f, also with a BCS-HERE comment.

3) The planes uwall, vwall, wwwall, are stored in the restart file,

but they are not read in the present version. You may want to read

them to make sure that you are continuing the same forcing strategy as

in the previous run (but then, you may not). If you want to do it, add

the appropriate lines in getfil.

4) Remember that this is a parallel code, and that things done in one

processor have to be communicated explicitly to other processors for

them to know. However, because the boundary planes are small, they are

not sent. All the nodes compute them independently, and they are all

assumed to be doing the same thing. They all know the correct time,

so that, if you do uwall=f(t), they would all create consistent

numbers. However, if you do something else (say, something random) you

have to make sure that they all know what the others are doing

(probably by adding a MPI\_BCAST or an MPI\_ALLREDUCE).

----------------- THINGS FAR IN THE FUTURE ------

Other source files are utilities that you should not need to worry

about. The only possible exceptions are the ffts (cftsingle and

rftsingle). They are single-precision versions of the ncar

package. Ancient, but reliable. I have tested them against fftw and,

unless the latter are really well optimized, these are as good as

anything, and fully portable. However, they are static things. They

have a parameter (different for each of them) which controls the

largest fft that they will do (mmaxt). It is set now at 4096, which

is far higher that you should have to use now. However, if in the

future you decide to run something really big, please remember to

change it. Otherwise you get really nice crashes.

Also, as they are implemented now, mgal can only have factors 2,3 and 5.