Workflow for generating CitcomS benchmark results

September 5, 2014

Code changes to CitcomS

Calculating the volume average quantities

The following function has been added to `Process_buoyancy.c` to compute $\langle T \rangle$ and $\langle V_{rms} \rangle$

```
/*
 * compute the volume average of temperature and RMS velocity
 */
void compute_volume_avg(struct All_variables *E, double *T_avg, double *Vrms_avg)
{
  int m, n, i;
  double *S1[NCS], *S2[NCS];

  for(m=1; m<=E->sphere.caps_per_proc; m++)
  {
    S1[1] = (double *)malloc((E->lmesh.nno+1)*sizeof(double));
    S2[1] = (double *)malloc((E->lmesh.nno+1)*sizeof(double));
  }

  for(m=1; m<=E->sphere.caps_per_proc; m++)
  {
    for(i=1; i<=E->lmesh.nno+1; i++)
    {
      S1[m][i] = E->T[m][i];
      S2[m][i] =
        E->sphere.cap[m].V[1][i]*E->sphere.cap[m].V[1][i] +
        E->sphere.cap[m].V[2][i]*E->sphere.cap[m].V[2][i] +
        E->sphere.cap[m].V[3][i]*E->sphere.cap[m].V[3][i];
    }
  }

  *T_avg = return_bulk_value_d(E, S1, 1); /* 1 => need volume average */
  *Vrms_avg = sqrt(return_bulk_value_d(E, S2, 1)); /* 1 => need volume average */

  for(m=1;m<=E->sphere.caps_per_proc;m++) {
    free((void *)S1[m]);
    free((void *)S2[m]);
  }
}
```

The following function has been added to `Output.c`

```
void output_volume_avg(struct All_variables *E, int cycles)
{
```

```
  /* volume average output of temperature and rms velocity */
  void compute_volume_avg();

  int j;
  char output_file[255];
  FILE *fp1;
  double T_avg=0.0, V_rms_avg=0.0;

  /* compute horizontal average here.... */
  compute_volume_avg(E, &T_avg, &V_rms_avg);

  if (E->parallel.me == 0)
  {
    sprintf(output_file,"%s.volume_avg", E->control.data_file);
    fp1=fopen(output_file,"a+");
    fprintf(fp1,"%d %.4e %.4e %.4e\n",
            cycles, E->monitor.elapsed_time, T_avg, V_rms_avg);
    fclose(fp1);
  }
}
```

Temperature Initial Conditions for the B Benchmarks

The function construct_tic_from_input has been modified to set up the temperature initial condition for the B benchmarks. This modified code is called by setting tic_method=6 in the parameter file

```
static void construct_tic_from_input(struct All_variables *E)
{
    double mantle_temperature;

    switch (E->convection.tic_method){
    .
    .
    .
    case 6:
      /* a conductive temperature profile + perturbations at all layers */
      conductive_temperature_profile(E);
      add_perturbations_at_all_layers_B(E);
      break;
    .
    .
    .
    }
}
```

The new function add_perturbations_at_all_layers_B is defined as

```
static void add_perturbations_at_all_layers_B(struct All_variables *E)
{
  /*
   * This method generates the initial temperature profile according to
   * equation (47) of the Zhong et. al. 2008 paper. (4,0)+(4,4) I.C. is
   * hard coded
   */
```

```
    int m, i, j, k, node;
    int p;
    int nox, noy, noz, gnoz;
    double r1, t1, f1, tlen, flen, rlen, con;

    nox = E->lmesh.nox;
    noy = E->lmesh.noy;
    noz = E->lmesh.noz;
    gnoz = E->mesh.noz;

    rlen = M_PI / (E->sphere.ro - E->sphere.ri);

    for (p=0; p<E->convection.number_of_perturbations; p++) {
        con = E->convection.perturb_mag[p];

        if (E->parallel.me_loc[1] == 0 && E->parallel.me_loc[2] == 0
            && E->sphere.capid[1] == 1 )
            fprintf(stderr,"Initial temperature perturbation:  mag=%g\n", con);

        if(E->sphere.caps == 1) {
          myerror(E, "add_pertubrbations_at_all_layers_B is not implemented for the Regional model");
        }
        else {
            /* global mode, add spherical harmonics perturbation */

            for(m=1; m<=E->sphere.caps_per_proc; m++)
                for(i=1; i<=noy; i++)
                    for(j=1; j<=nox;j ++)
                        for(k=1; k<=noz; k++) {
                            node = k + (j-1)*noz + (i-1)*nox*noz;
                            t1 = E->sx[m][1][node];
                            f1 = E->sx[m][2][node];
                            r1 = E->sx[m][3][node];

                            E->T[m][node] += con * (modified_plgndr_a(4,0,t1) +
                                (5.0/7.0)*cos(4*f1)*modified_plgndr_a(4,4,t1))
                                * sin((r1-E->sphere.ri) * rlen);
                        }
        } /* end if */
    } /* end for p */

    return;
}
```

Files needed for generating the plots and tables

The following four files are needed for generating the plots and tables. The "A1_10" prefix is the value of the `datafile` parameter from the configuration file. In this particular example, we have `nprocz=2`, hence there are only 2 `*.horiz_avg.*` files. Also, 10000 is the **last time step** for which we have the recorded data.

A1_10.o3993348 The output logfile generated when CitcomS is run on Stampede (or some other cluster). This file has information for the surface and bottom heat flux.

A1_10.horiz_avg.0.10000 The horizontal average file for the 0th z process, at the 10000th time step.

A1_10.horiz_avg.1.10000 The horizontal average file for the 1st z process, at the 10000th time step.

A1_10.volume_avg The volume average file that has the numbers for $\langle T \rangle$ and $\langle V_{rms} \rangle$

Generating the data files

Surface heat flux

```
cat A1_10.o3993348 | sed -n 's/surface heat flux= //p' | awk 'NR%2==0' >> surface-heat-flux-A1_10
```

Extract the horizontally averaged r values

```
awk '{print $1}' A1_10.horiz_avg.0.10000 >> R_vals_0

awk '{print $1}' A1_10.horiz_avg.1.10000 >> R_vals_1

cat R_vals_0 R_vals_1 >> A1_10_r
```

`A1_10_r` has duplicate entries; the last line from `R_vals_0` and the first line from `R_vals_1`. This needs to be fixed manually.

Extract the horizontally averaged T values

```
awk '{print $2}' A1_10.horiz_avg.0.10000 >> T_vals_0

awk '{print $2}' A1_10.horiz_avg.1.10000 >> T_vals_1

cat T_vals_0 T_vals_1 >> A1_10_T
```

`A1_10_T` has duplicate entries; the last line from `T_vals_0` and the first line from `T_vals_1`. This needs to be fixed manually.

Generating the Plots

The Python script `citcoms-benchmark.py` can be used for generating figures 5(a), 5(b), 5(c) and figure 7(a) of the Zhong et. al. 2008 paper. You need to generate the files for the A3 and A8 benchmarks, using the procedure for A1 described above, and correctly specify the path to the data files.

```python
'''
citcoms-benchmark.py
'''
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

d=0.45
kappa=1.0
time_rescale = (d*d)/kappa # t_citcom / time_rescale = t_nondim (or t_paper)
vel_rescale = kappa / d    # v_citcom / vel_rescale = v_nondim (i.e. v_paper)
r_t = 1.0
r_b = 0.55
top_prefac = r_t*(r_t-r_b)/r_b;

dataA1 = np.loadtxt("/home/rkk/A1_10/A1_10.volume_avg")
dataA3 = np.loadtxt("/home/rkk/A3_08/A3_08.volume_avg")
dataA8 = np.loadtxt("/home/rkk/A8_05/A8_05.volume_avg")
surfA1 = top_prefac*np.loadtxt("/home/rkk/A1_10/surface-heat-flux-A1_10")
```

```
surfA3 = top_prefac*np.loadtxt("/home/rkk/A3_08/surface-heat-flux-A3_08")
surfA8 = top_prefac*np.loadtxt("/home/rkk/A8_05/surface-heat-flux-A8_05")
timeA1 = (1.0/time_rescale)*dataA1[:,1]
timeA3 = (1.0/time_rescale)*dataA3[:,1]
timeA8 = (1.0/time_rescale)*dataA8[:,1]
T_avgA1 = dataA1[:,2]
T_avgA3 = dataA3[:,2]
T_avgA8 = dataA8[:,2]
Vrms_avgA1 = (1.0/vel_rescale)*dataA1[:,3]
Vrms_avgA3 = (1.0/vel_rescale)*dataA3[:,3]
Vrms_avgA8 = (1.0/vel_rescale)*dataA8[:,3]

plt.figure(1)
# plt.subplot(3,1,1)
plt.plot(timeA1,T_avgA1, 'g-', label='A1')
plt.plot(timeA3,T_avgA3, 'r-', label='A3')
plt.plot(timeA8,T_avgA8, 'b-', label='A8')
plt.legend(title='Benchmark', loc='best')
plt.xlabel('time')
plt.ylabel('<T>')
plt.title('Time dependence of Volume-averaged temperature')
plt.ylim((0,0.8))
plt.xlim((0,1.0))
plt.yticks([0.0,0.2,0.4,0.6,0.8])

plt.figure(2)
# plt.subplot(3,1,2)
plt.plot(timeA1,Vrms_avgA1, 'g-', label='A1')
plt.plot(timeA3,Vrms_avgA3, 'r-', label='A3')
plt.plot(timeA8,Vrms_avgA8, 'b-', label='A8')
plt.legend(title='Benchmark', loc='best')
plt.xlabel('time')
plt.ylabel('<V_rms>')
plt.title('Time dependence of Root-mean squared velocity')
plt.ylim((0,120.0))
plt.xlim((0,1.0))
plt.yticks(np.arange(0.0,121.0,20.0))

plt.figure(3)
# plt.subplot(3,1,3)
plt.plot(timeA1,surfA1, 'g-', label='A1')
plt.plot(timeA3,surfA3, 'r-', label='A3')
plt.plot(timeA8,surfA8, 'b-', label='A8')
plt.legend(title='Benchmark', loc='best')
plt.xlabel('time')
plt.ylabel('Nu_t')
plt.title('Time dependence of surface Nusselt number')
plt.ylim((0,6.0))
plt.xlim((0,1.0))

plt.show()
```

Calculating mean and standard deviations

The Python script `table6.py` can be used for computing the mean and standard deviations of the values reported in table 6 and 7 of the paper. Again, the path to the data files needs to be set appropriately

```python
#------------------------------------------------------------------------------------------------------
# table6.py : Generate the entries for Table 6 of the Zhong et. al. paper
#------------------------------------------------------------------------------------------------------
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

d=0.45
kappa=1.0
time_rescale = (d*d)/kappa # t_citcom / time_rescale = t_nondim (or t_paper)
vel_rescale = kappa / d    # v_citcom / vel_rescale = v_nondim (i.e. v_paper)
r_t = 1.0
r_b = 0.55
top_prefac = r_t*(r_t-r_b)/r_b;

def get_array_slice(arr, low, high):
  '''
  returns a boolean array with True for arr in[low,high]
  '''

  lower_limit = np.greater_equal(arr, low)
  upper_limit = np.less_equal(arr, high)
  combined = np.logical_and(lower_limit, upper_limit)
  return combined

def compute_and_print_mean_and_std(benchmark, volume_avg_file, heat_flux_file, tlow, thigh):
  data = np.loadtxt(volume_avg_file)
  surf = top_prefac * np.loadtxt(heat_flux_file)
  time = (1.0/time_rescale) * data[:,1]
  T_avg = data[:,2]
  Vrms_avg = (1.0/vel_rescale)*data[:,3]
  combined = get_array_slice(time, tlow, thigh)
  sub_Tavg = T_avg[combined]
  sub_Vrms = Vrms_avg[combined]
  sub_Nu_t = surf[combined]

  print("----------------- %s Benchmark Results -----------------------" % benchmark)
  print("<T> = %8.4f std = %e" % (np.mean(sub_Tavg), np.std(sub_Tavg)))
  print("<V_rms> = %8.4f std = %e" % (np.mean(sub_Vrms), np.std(sub_Vrms)))
  print("<Nu_t> = %8.4f std = %e" % (np.mean(sub_Nu_t), np.std(sub_Nu_t)))
  print("")

compute_and_print_mean_and_std("A1",
  "/home/rkk/A1_10/A1_10.volume_avg",
  "/home/rkk/A1_10/surface-heat-flux-A1_10",
  0.7, 1.0)
```