Iowa High Performance Computing Summer School 2015

HYDRO Example Parallel Hydrodynamics Program

HYDRO is a parallel code for the simulation of two-dimensional hydrodynamics problems, and serves as a relatively simple but practical example of how parallel programming is used in practice. Here we outline the equations evolved by the code and describe how to use the code.

1 Equations

The equations of hydrodynamics are

$$\frac{\partial \rho}{\partial t} + \mathbf{u} \cdot \nabla \rho = -\rho \nabla \cdot \mathbf{u} \tag{1}$$

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho} \nabla p \tag{2}$$

$$\frac{\partial p}{\partial t} + \mathbf{u} \cdot \nabla p = -\gamma p \nabla \cdot \mathbf{u} \tag{3}$$

where the convective derivative is given by

$$\frac{d}{dt} \equiv \frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla \tag{4}$$

and the adiabatic index is given by γ.

The system for linear sound waves is given by the matrix equation

$$\begin{pmatrix} \omega & -\rho_0 k_x & -\rho_0 k_y & 0 \\ 0 & \omega & 0 & -k_x/\rho_0 \\ 0 & 0 & \omega & -k_y/\rho_0 \\ 0 & -\gamma p_0 k_x & -\gamma p_0 k_y & \omega \end{pmatrix} \begin{pmatrix} \rho_1 \\ u_x \\ u_y \\ p_1 \end{pmatrix} = 0, \tag{5}$$

giving the dispersion relation

$$\omega^2(\omega^2 - k^2 c_s^2) = 0 \tag{6}$$

where the equilibrium sound speed is given by $c_s^2 = \gamma p_0/\rho_0$ and $k^2 = k_x^2 + k_y^2$. The eigenfrequency for the sound wave is therefore

$$\omega = \pm k c_s \tag{7}$$

with an eigenfunction given by

$$u_x = \pm c_s \frac{k_x}{k} \frac{p_1}{\gamma p_0}, \tag{8}$$

$$u_y = \pm c_s \frac{k_y}{k} \frac{p_1}{\gamma p_0}, \tag{9}$$

and

$$\rho_1 = \frac{p_1}{c_s^2} \tag{10}$$

for a pressure perturbation with amplitude δp_0 of the form

$$p_1 = \delta p_0 \cos(k_x u_x + k_y u_y - \omega t). \tag{11}$$

2 Compiling and Running HYDRO

HYDRO comes packed in a tar file with the Makefile necessary to compile it. If you are porting it to a new system, you may have to modify the Makefile to get it to work, in particular specifying the compiler flags to specify automatic promotion of real variables to double precision (typically `-r8`), the compiler optimization level you desire, and the debug flag `-g` if you wish to do profiling or debugging.

To compile HYDRO on a system to which it has already been ported:

1. Create a directory called `hydro` and copy the tar file (for example, `hyd100524.tar`) into that directory.
2. Unpack that tar file using
   `tar -xvf hyd100524.tar`
3. Type `make` to compile. This will produce the executable `hydro.e`
4. To test that everything has worked, you may run the sample1.in file on multiple rocessors (must be an even number of processors), for example
   `mpirun-np 16 ./hydro.e sample1.in`
   The first (and only) argument of the executable is the name of the input file. Note the command to run HYDRO in parallel will generally depend on the system.
5. To delete the current executable so that you can recompile, type `make clean`.

3 HYDRO Input File

A sample input file, `sample1.in`, comes packaged with HYDRO. This example is used to describe the variables in the input file. The sample input file follows on the next page.

3.1 Sample HYDRO Input File

```
!========================================================================
!  sample1.in Sample input file
!  HYDRO SAMPLE INPUT FILE
!
!  Greg Howes
!  2009 JUL 28
!  NOTE: This is a dimensional code, with all variables in SI units
!========================================================================
!  REQUIRED NAMELISTS
!========================================================================
!  SYSTEM, PLASMA, AND MODEL PARAMETERS
&system
!Number of gridpoints-------------------------------------------------
nx = 128 !Number of gridpoints in x
ny = 128 !Number of gridpoints in y
!Length of simulation box (m)-----------------------------------------
lx = 1.0 !Length of simulation domain in x
ly = 1.0 !Length of simulation domain in y
!Timestepping---------------------------------------------------------
dt = 0.00025 !Timestep (s)
nsteps=4000 !Total number of timesteps to run
nsave=400 !Save output at each nsave steps
save0=.true.  !T=Output Initial Conditions
!Initial conditions---------------------------------------------------
init = 0 !  0 = sound wave initial conditions
/
!========================================================================
!========================================================================
!  OPTIONAL NAMELISTS
!========================================================================
!  Sound Wave Initialization
&sound
!Equilibrium conditions-----------------------------------------------------
rho0 = 5.0 !Equilibrium density (kg/m**3)
p0 = 3.0 !Equilibrium pressure (N/m**2 = kg/(s**2 m))
ux0 = 0.0 !Equilibrium velocity in x (m/s)
uy0 = 0.0 !Equilibrium velocity in y (m/s)
gamma = 1.666667 !Adiabatic Index
!Sound wave solutions
nkx0 = 1 !Integral Sound Wavenumber in x (Number of wavelengths per lx)
nky0 = 0 !Integral Sound Wavenumber in y (Number of wavelengths per ly)
p1 = 0.003 !Pressure perturbation (N/m**2)
/
!========================================================================
!========================================================================
```

Although most of the input file parameters are self-explanatory, we discuss a couple of the parameters here.

The input `save0` is a logical variable: a value of `.true.` will write to disk the initial conditions at time zero. Since saving to disk is a very expensive process, setting this to `.false.` prevents this step.

The timestep `dt` must be chosen to be small enough to satisfy the Courant-Friedrichs-Lewis (CFL) timestep criterion for stability,

$$f_{CFL} = \frac{\Delta t v_{max}}{\Delta x} < 1. \tag{12}$$

For the case of a simple sound wave, the maximum wave speed is the sound speed,

$$c_s = \sqrt{\frac{\gamma p_0}{\rho_0}}, \tag{13}$$

or in the case of the input variables, `cs0`$= \sqrt{\frac{\texttt{gamma p0}}{\texttt{rho0}}}$. For the minimum grid spacing `dr` $=$ `min(`Δ`x,`Δ `y)` and timestep `dt`, the CFL condition becomes

$$f_{CFL} = \frac{\texttt{dt cs0}}{\texttt{dr}} < 1, \tag{14}$$

or simply

$$\texttt{dt} < \frac{\texttt{dr}}{\texttt{cs0}}. \tag{15}$$

4 Discretization

The code `HYDR0` is based on a finite difference scheme for the spatial derivatives and a third-order Adams-Bashforth scheme for advancing the equations in time. A two-dimensional grid of `nx` by `ny` gridpoints is set up to discretization a two-dimensional simulation domain of size `lx` by `ly` with periodic boundary conditions. The 2-D hydrodynamic system involves the independent spatial variables (x, y) and the dependent fluid variables (ρ, u_x, u_y, p). The spatial positions are defined in 1-D array variables
`rx(-1:nx)`
and
`ry(-1:ny)`,
where the simulation domain runs over the `nx` positions from `(0:nx-1)` and the boundary conditions are handled using "ghost zones" at `-1` and `nx`.

The dependent fluid variables are contained in a three dimensional array
`qq(i,ix,iy)`
where the first index `i` determines the fluid variable using indexing integers,

```
irho = 1   !Density
iux = 2    !x-velocity
iuy = 3    !y-velocity
ip = 4     !Pressure
```

Spatial derivatives are discretized using centered finite differences

$$\frac{\partial q_{i,j}}{\partial x} = \frac{q_{i+1,j} - q_{i-1,j}}{x_{i+1} - x_{i-1}} \tag{16}$$

where the fluid variable at $x = x_i$ and $y = y_j$ is given by $q_{i,j}$. Time derivatives are discretized using a third-order Adams-Bashforth scheme,

$$\frac{\partial q}{\partial t} = \frac{q^{(n+1)} - q^{(n)}}{t^{(n+1)} - t^{(n)}} = \frac{23}{12}\left.\frac{\partial q}{\partial t}\right|^{(n)} - \frac{4}{3}\left.\frac{\partial q}{\partial t}\right|^{(n-1)} + \frac{5}{12}\left.\frac{\partial q}{\partial t}\right|^{(n-2)} \tag{17}$$

5 Initialization

The initial conditions for HYDRO are set by choosing the appropriate parameter for the input variable `init`. At present, the only initial conditions is that for a sound wave, `init=0`. This choice sets up a sound wave according the the linear eigenfunction described in Section (1).

Since the third-order Adams-Bashforth scheme requires a knowledge of the time derivatives at the previous two timesteps, the timesteps need to be initialized. This is accomplished by taking two first-order Eulerian timesteps with timesteps $\Delta t =$ `dt/16`. Then, another 30 steps are taken using the third-order Adams-Bashforth scheme with timestep $\Delta t =$ `dt/16`. The values of the time derivatives at $t = 0, \texttt{dt}, \texttt{2dt}$ are all kept, enabling the commencement of the third-order Adams-Bashforth scheme at $t =$ `2dt` using timestep $\Delta t =$ `dt`.

6 Parallelization

The code HYDRO is parallelized using the MPI library and employs a simple domain decomposition in the x direction. For `nproc` MPI tasks, the simulation domain is split up into `nproc` subdomains each of size `nx/nproc`. Note that if `nx` is not divisible by `nproc`, then not all of the processors will have the same number of gridpoints in x; the code does handle this by allowing the last processor to have fewer than `nx/nproc` gridpoints in x. Although this parallelization scheme is simple, under certain conditions it can lead to poor scaling (see In-Class Exercises #2).

The code handles passing information between processors by setting up the local dependent fluid variable array to have size `nx/nproc`+2, so that there are "ghost zones" on both x boundaries.

At the beginning of each timestep, HYDRO passes the necessary boundary information needed by each subdomain using send and receive calls to the MPI tasks handling the neighboring subdomains. Care is taken to avoid a deadlock by ensuring that the sends and receives are performing in a unique order. The rule for determining which processors sends first is that the processor with the even number sends first. This effectively prevents the code from allowing odd numbers of processors. All of this code is contained in the module `hydro_bcs.f90`.

7 Output from HYDRO

Presently HYDRO collects dependent variable information from all processors onto the master processor (processor 0), and then writes the information into a single file on the disk. This is NOT a scalable approach, but is taken here for simplicity. For scaling runs of the algorithm (not including saving the output), one can merely set `save0=.false.` and set `nsave > nsteps` and the code will never attempt to save the data, enabling one to simulate large simulation sizes on large numbers of processors.

The output each time the code saves the output is put into a file *runname*`.out.####`. This is an ASCII text file with data in the following columns:

Column	Value
1	time
2	ix
3	iy
4	x
5	y
6	ρ
7	u_x
8	u_y
9	p
10	iproc

Note that, for the current scheme collecting all data on proc0 for output, the last column iproc=0 always.

The output from HYDRO can easily be viewed using `gnuplot`. The following steps will explain how to plot the output and make postscript figures from this output.

1. Run `HYDRO` with the input file `sample1.in`.

2. Navigate into the `data` directory of `HYDRO`. A listing of the directory will show
   ```
   sample1.out.0000 sample1.out.0003 sample1.out.0006 sample1.out.0009
   sample1.out.0001 sample1.out.0004 sample1.out.0007 sample1.out.0010
   sample1.out.0002 sample1.out.0005 sample1.out.0008
   ```

3. Start `gnuplot` with the command
   `gnuplot`
   In `gnuplot`, you can get help for any *command* by using
   `help` *command*

4. Set the data style to plot lines rather than points
   `set style data lines`

5. Plot out the final values of u_x vs. x using the command
   `plot 'sample1.out.0010' u 4:7`
   If Xforwarding has been enabled when you issued your `ssh` command to log onto the remote machine, an Xwindow will pop up with your plot (this may take a little time for the secure connection to be established).

6. To add another curve, such as the initial condition to the same plot, use
   `replot 'sample1.out.0000' u 4:7`
   These two curves should be very close to each other, as the last plot is exactly one period of the wave after the first plot. You can plot two or more curves together with a single command
   `plot 'sample1.out.0000' u 4:7, 'sample1.out.0010' u 4:7`

7. To see how the wave evolves in time over the course of the simulations, plot every other snapshot in time as follows:
   ```
   plot 'sample1.out.0000' u 4:7
   replot 'sample1.out.0002' u 4:7
   replot 'sample1.out.0004' u 4:7
   replot 'sample1.out.0006' u 4:7
   replot 'sample1.out.0008' u 4:7
   replot 'sample1.out.0010' u 4:7
   ```
   You can also label the axes using
   ```
   set xlabel "x"
   set ylabel "u_x"
   ```

8. To output a plot to a Postscript file:
   `set terminal postscript enhanced linewidth 2.0  set output "sample1.ps"` Then issue all of your plotting commands again. Note that the plot on screen will not update with these commands.

   Finally, return the output to the screen by using
   ```
   set output
   set terminal x11
   ```

9. Because `ghostview` is not installed on Moffett, you will need to copy the resulting postscript file back to your local machine to look at it using `gv`. Thus, use the commands
   `scp` *username*`@moffett.rcac.purdue.edu: /hydro/data/sample1.ps ./`
   `gv sample1.ps &`
   The resulting file is plotted in Figure 1.

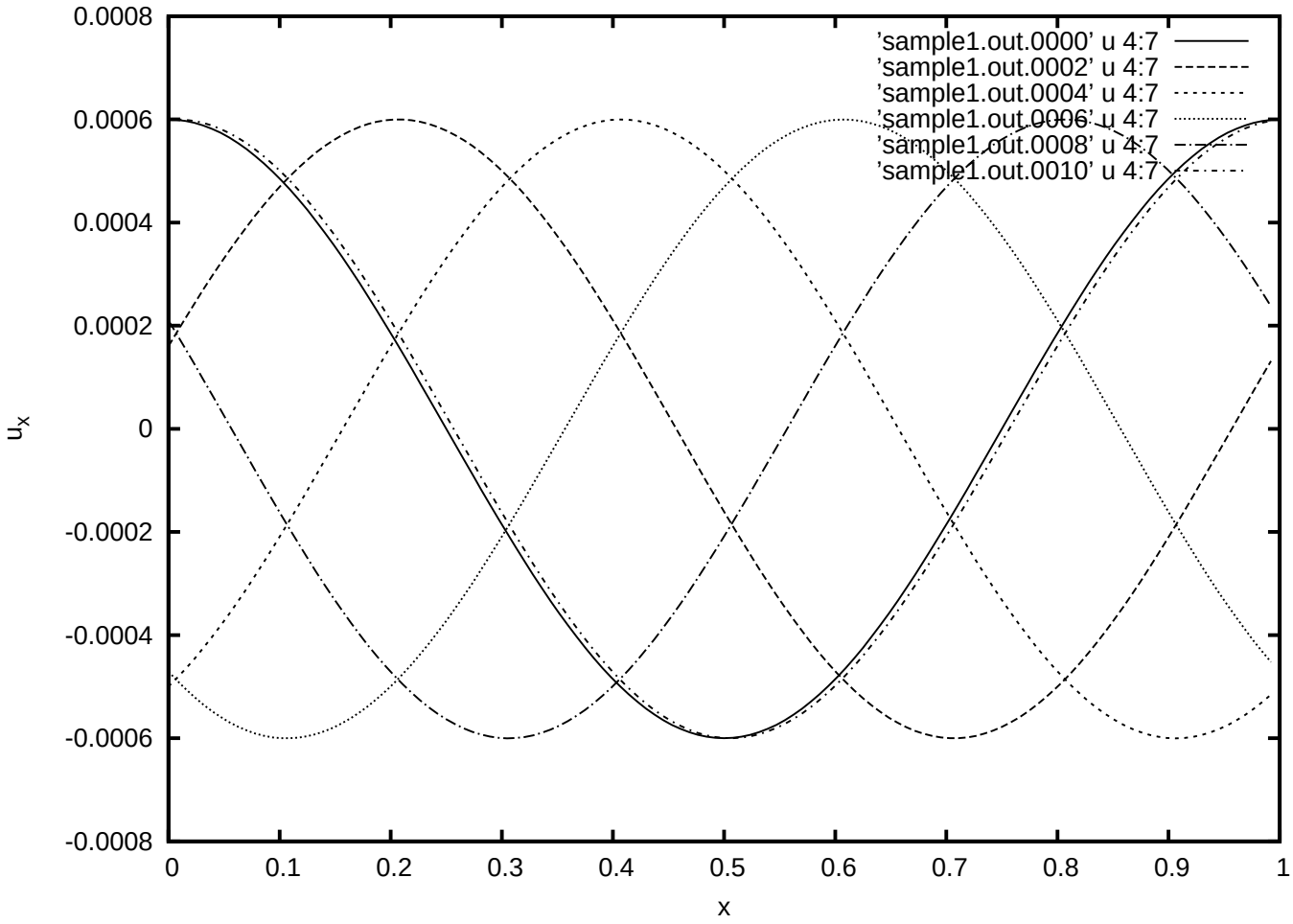

Figure 1: Postscript file output of `HYDRO` data for run `sample1.in` generated using `gnuplot`.

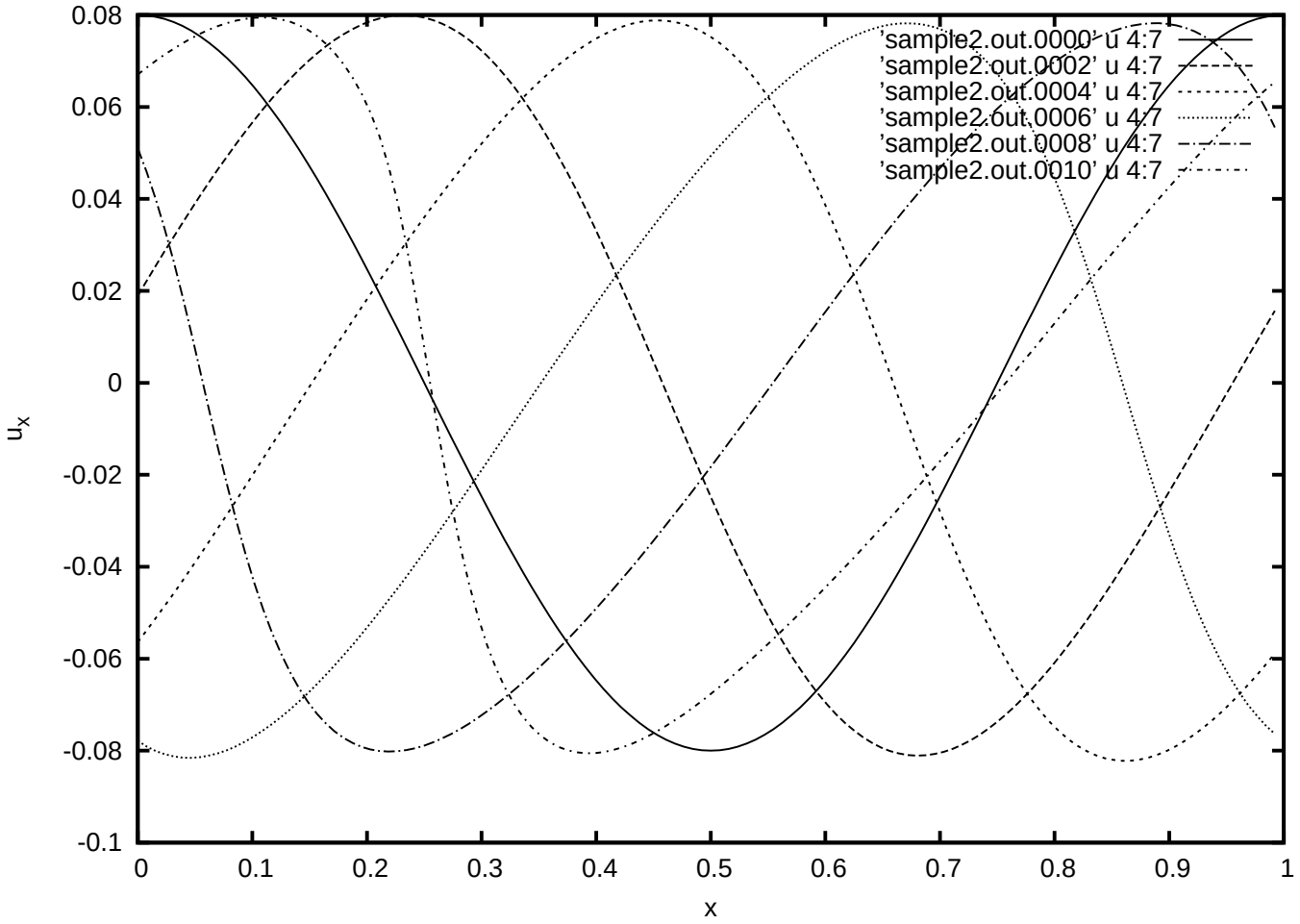

Figure 2: Postscript file output of `HYDRO` data for the nonlinear run `sample2.in` generated using `gnuplot`.

10. Nonlinear evolution of a finite amplitude sound wave: The output in Figure 1 is relatively boring because it corresponds to strictly linear waves, with negligible change of the waveform in time. However, if we change the amplitude of the sound wave in the input file from `p1 = 0.003` to `p1 = 0.4` and run the simulation again, we find the results in Figure 2. Here we can see that the leading edge of the sound wave steepens while the trailing edge flattens. This is due to the change of the sound speed with position due to changes in the density and pressure.

8 Timing Statistics from HYDRO

Note that, at present, the timing statistics do not work on Neon, requiring some debugging. When the timing statistics work, the output appears as follows:

HYDRO also outputs timing information in the file `sample1.time` as follows:

```
  Output from run sample1.in using 16 processing cores
Initialization     7.971010E-01 sec  ( 0.0133 min)   4.725 %
Save               6.122648E+00 sec  ( 0.1020 min)   36.293 %
Advance            9.946471E+00 sec  ( 0.1658 min)   58.960 %
    Local          8.304620E+00 sec  ( 0.1384 min)   49.227 %
    Communication  1.268621E+00 sec  ( 0.0211 min)   7.520 %
Total              1.686998E+01 sec  ( 0.2812 min)
```