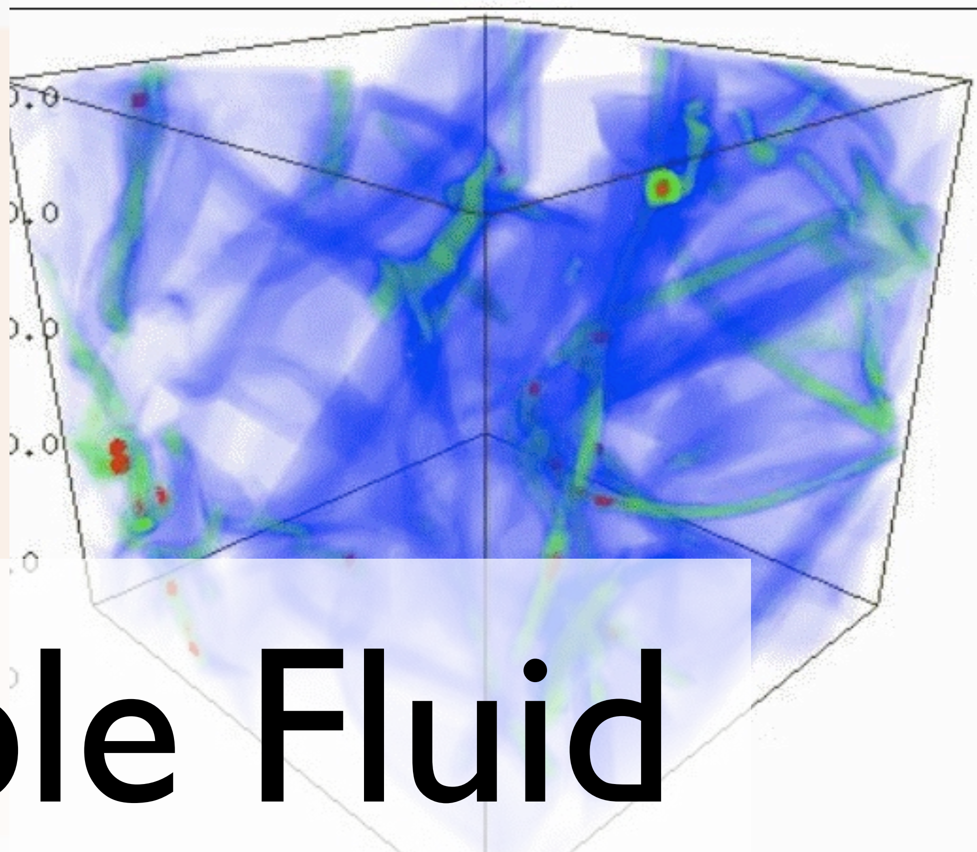
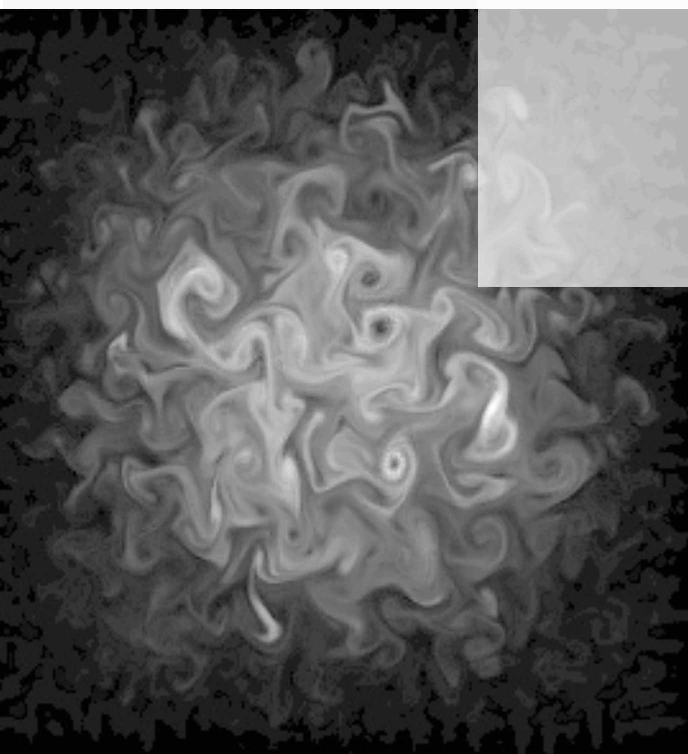
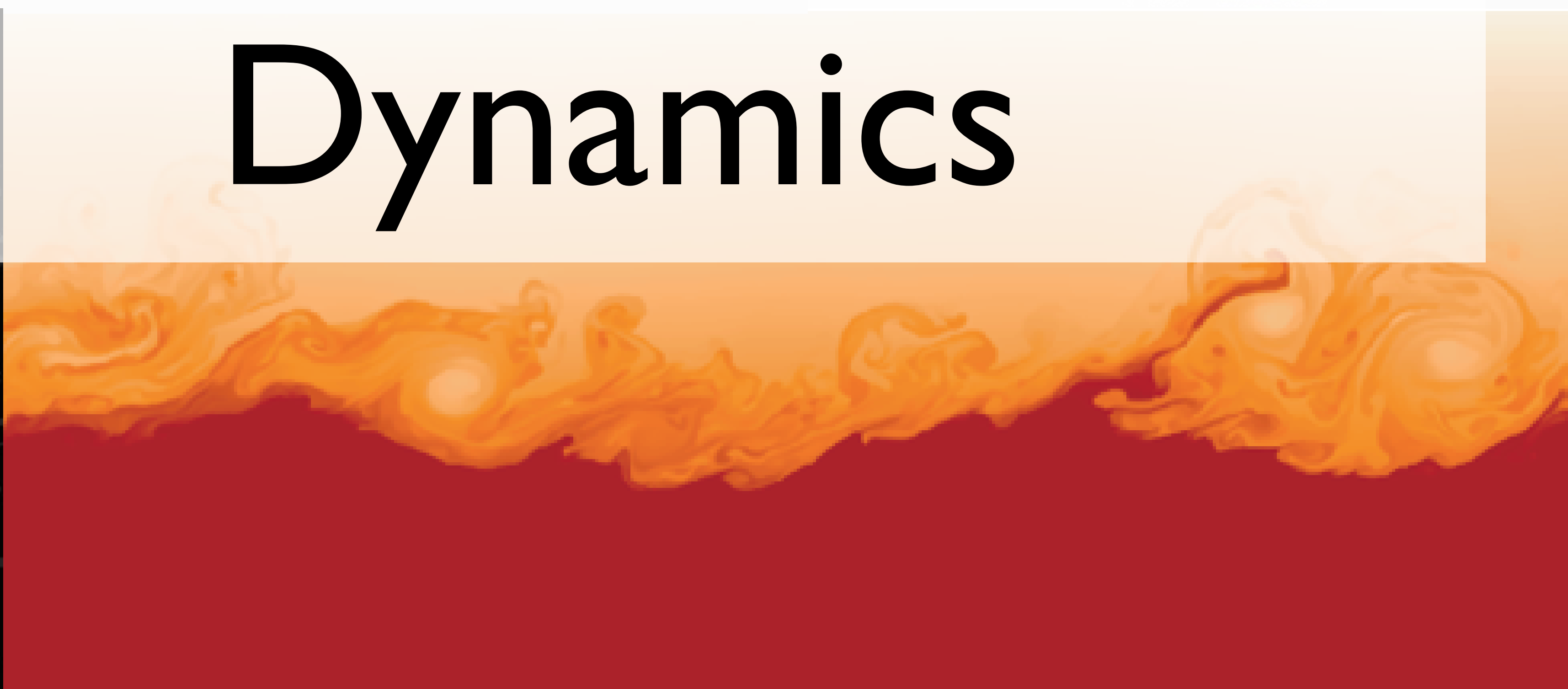
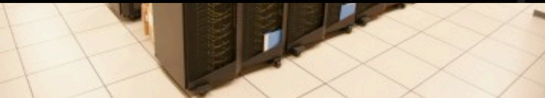








Compressible Fluid Dynamics

SciNet Parallel Scientific Computing Course

Aug 31 - Sept 4, 2009

SciNet High Performance Computing

Equations of Hydrodynamics

$$\frac{\partial}{\partial t}\rho + \nabla \cdot (\rho \mathbf{v}) = 0$$

$$\frac{\partial}{\partial t}(\rho \mathbf{v}) + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) = -\nabla p$$

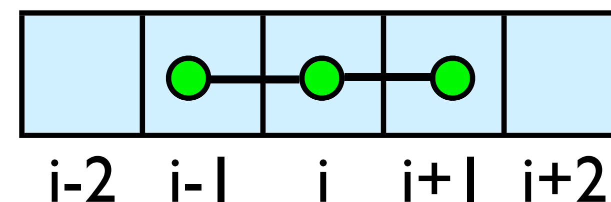
$$\frac{\partial}{\partial t}(\rho E) + \nabla \cdot ((\rho E + p)\,\mathbf{v}) = 0$$

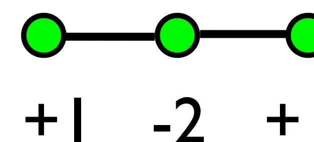
- Density, momentum, and energy equations
- Supplemented by an equation of state - pressure as a function of dens, energy

Discretizing Derivatives

$$\left.\frac{dQ}{dx}\right|_i \approx \frac{Q_{i+1} - 2Q_i + Q_{i-1}}{\Delta x}$$

- Done by finite differencing the discretized values
- Implicitly or explicitly involves interpolating data and taking derivative of the interpolant
- More accuracy - larger 'stencils'





Discretizing Derivatives

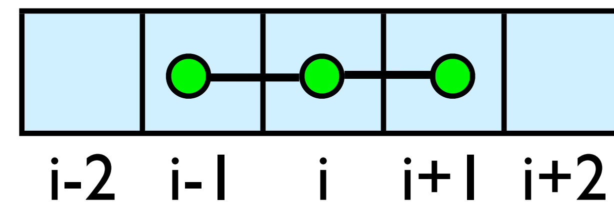
- Explicit hydrodynamics: only need information from as far away as the stencil reaches
- Nearest few neighbors
- Locality galore!

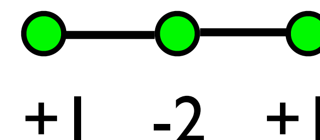
$$\frac{\partial Q}{\partial t} = f\left(\frac{\partial Q}{\partial x}\right)$$

$$\left.\frac{\partial Q^{(n)}}{\partial t}\right|_i \approx \frac{Q_i^{(n+1)} - Q_i^{(n)}}{\Delta t}$$

$$\left.\frac{dQ^{(n)}}{dx}\right|_i \approx \frac{Q_{i+1}^{(n)} - Q_{i-1}^{(n)}}{\Delta x}$$

$$Q_i^{(n+1)} = Q_i^{(n)} + \Delta t f\left(\frac{Q_i^{(n+1)} - Q_i^{(n)}}{\Delta t}\right)$$

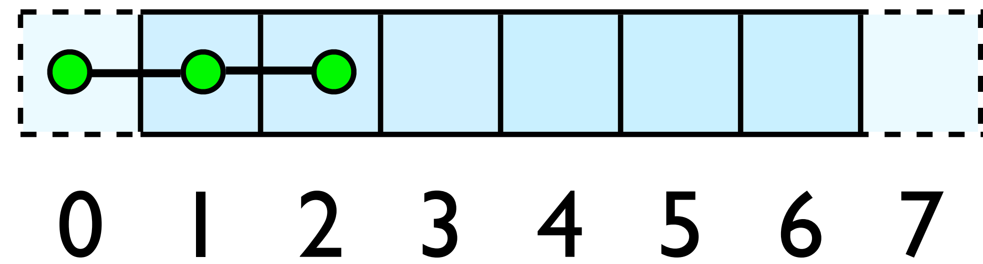




Guardcells

- How to deal with boundaries?
- Because stencil juts out, need information on cells beyond those you are updating
- Pad domain with 'guard cells' so that stencil works even for the 0th point in domain
- Fill guard cells with values such that the required boundary conditions are met

Global Domain

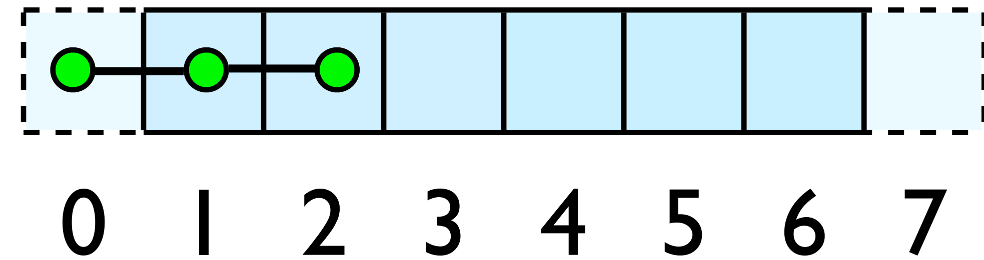


ng = 1
loop from ng, N - 2 ng

Guardcells

- Impose BCs before each timestep
- Our hydro code - 3 common boundary conditions
- ‘outflow’, reflect, and periodic
- Outflow (-1)- cell 0 just gets value from 1
- Reflect (-2); mirror the values
- Periodic(-3); copy values from other side (cell 0 gets values from cell 6)

Global Domain



ng = 1
loop from ng, N - 2 ng

Equations of Hydrodynamics

- Density, momentum, and energy equations
- Supplemented by an equation of state - pressure & temperature as a function of dens, energy

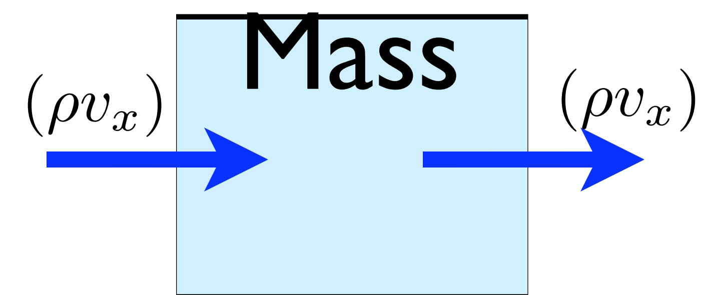
$$\begin{aligned}
\frac{\partial}{\partial t}\rho + \nabla \cdot (\rho \mathbf{v}) &= 0 \\
\frac{\partial}{\partial t}(\rho \mathbf{v}) + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) &= -\nabla p \\
\frac{\partial}{\partial t}(\rho E) + \nabla \cdot ((\rho E + p)\,\mathbf{v}) &= 0
\end{aligned}$$

Conservation Law form

- Conservation of mass, momentum, energy
- These are important properties, want numerical solver to maintain them

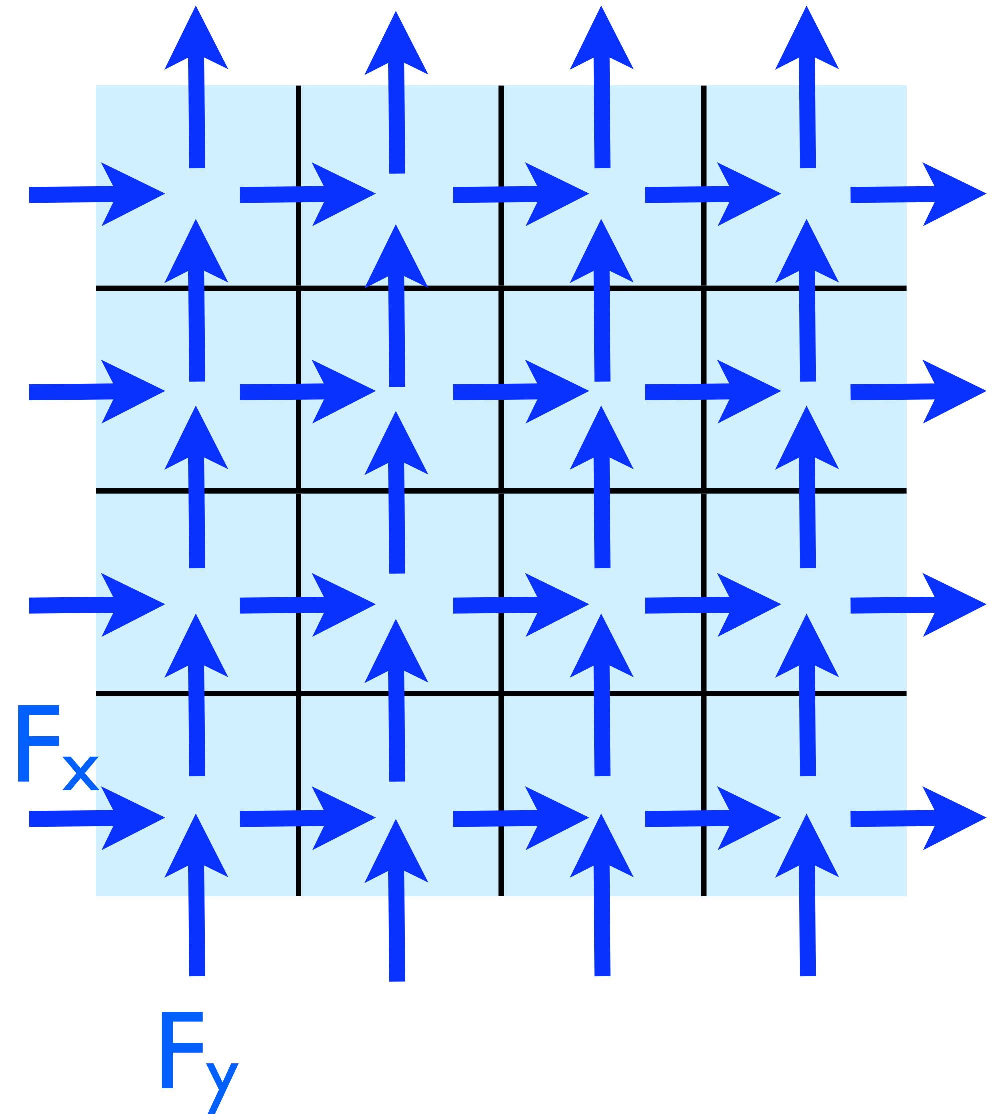
$$
\begin{aligned}
\frac{\partial}{\partial t}\rho + \nabla \cdot (\rho \mathbf{v}) &= 0 \\
\frac{\partial}{\partial t}\rho + \frac{\partial}{\partial x}(\rho v_x) &= 0 \\
\int_{x_L}^{x_R} \frac{\partial}{\partial t}\rho dx &= -\int_{x_L}^{x_R} \frac{\partial}{\partial x}(\rho v_x) \\
\frac{\partial}{\partial t}\mathrm{Mass} &= -(\rho v_x)_R + (\rho v_x)_L
\end{aligned}
$$

Change in mass = -outflux + influx



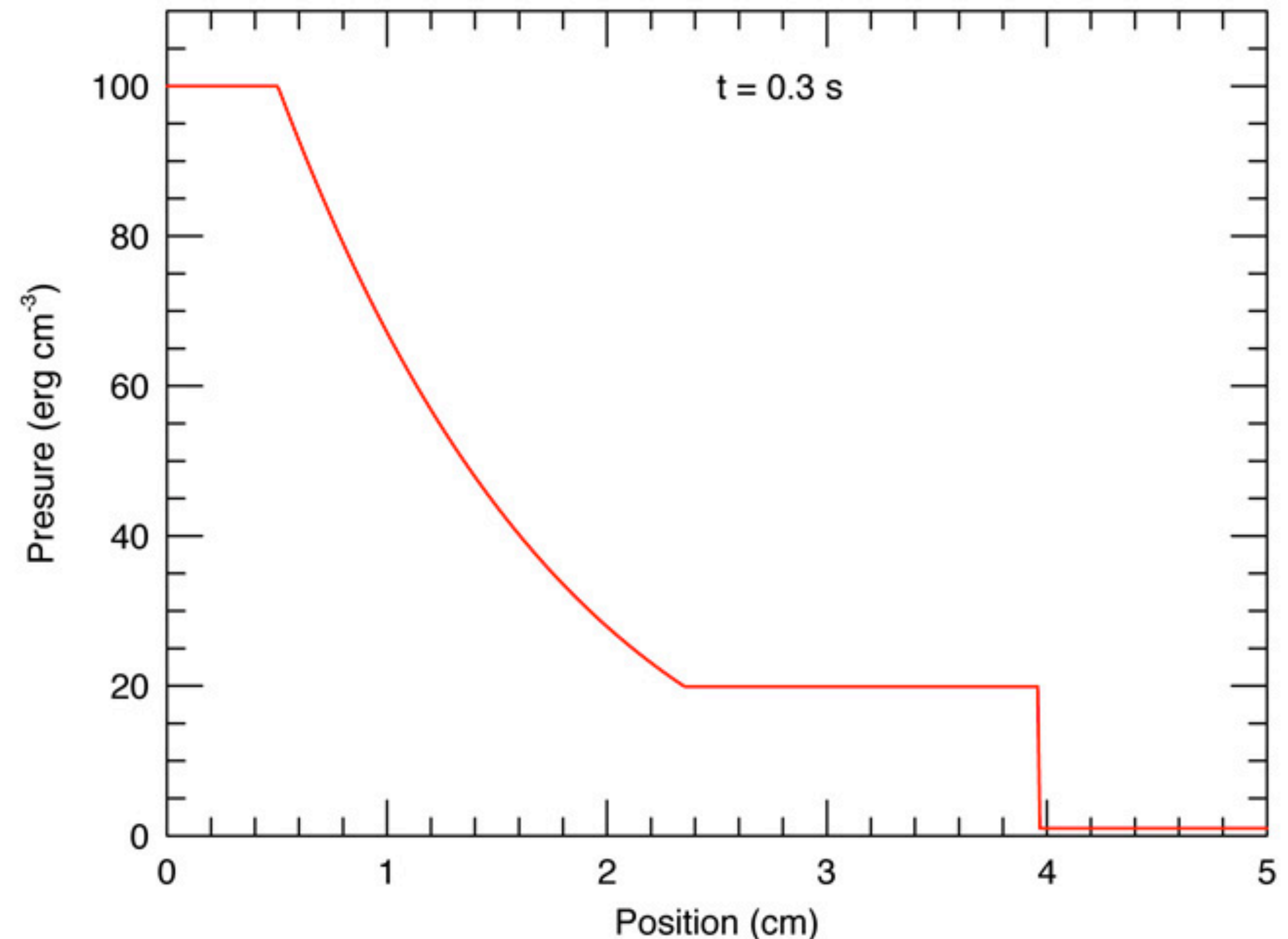
Finite Volume Method

- Conservative; very well suited to high-speed flows with shocks
- At each timestep, calculate fluxes using interpolation/finite differences, and update cell quantities.
- Use conserved variables -- eg, momentum, not velocity.



Flux Calculations

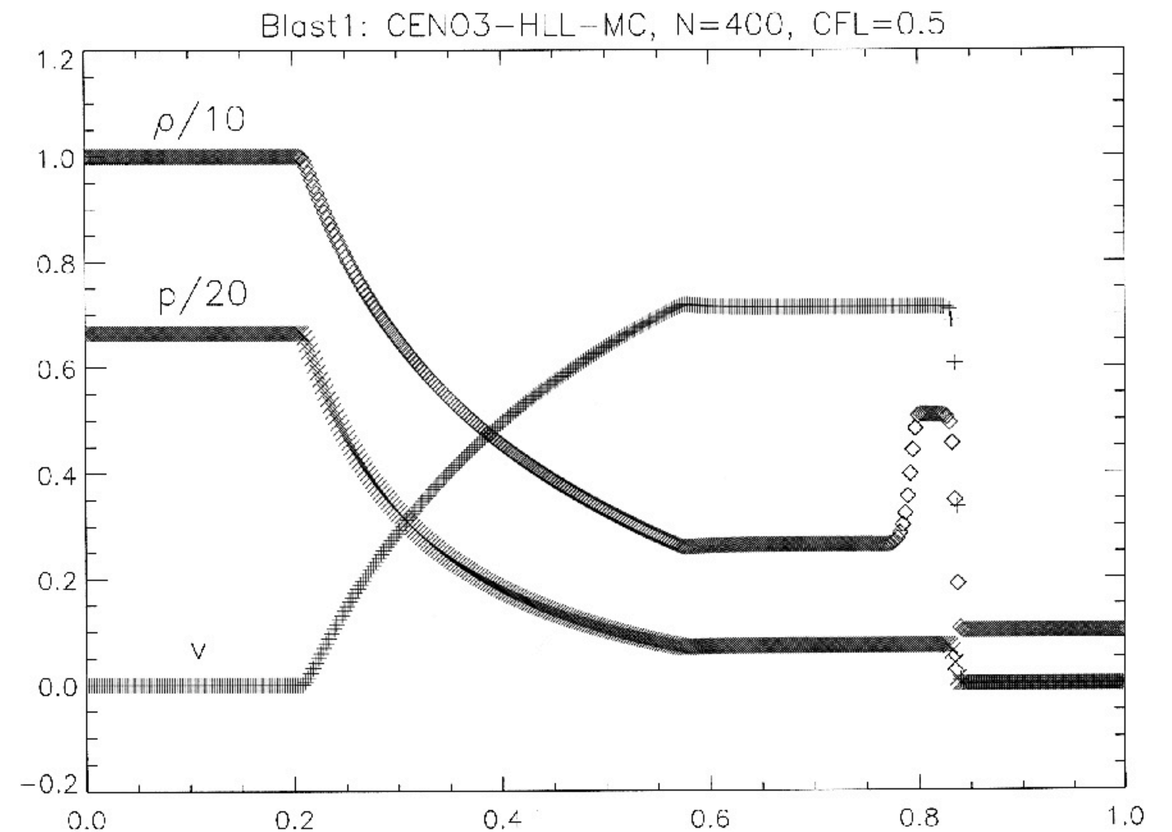
- Compressible flows: common to use Godunov-based schemes
- At cell interfaces, a Riemann problem is solved -- exact solution to a fluid jump
- Expensive, but does a great job of dealing with shocks



Frank Timmes, http://cococubed.asu.edu/code_pages/exact_riemann.shtml

Flux Calculations

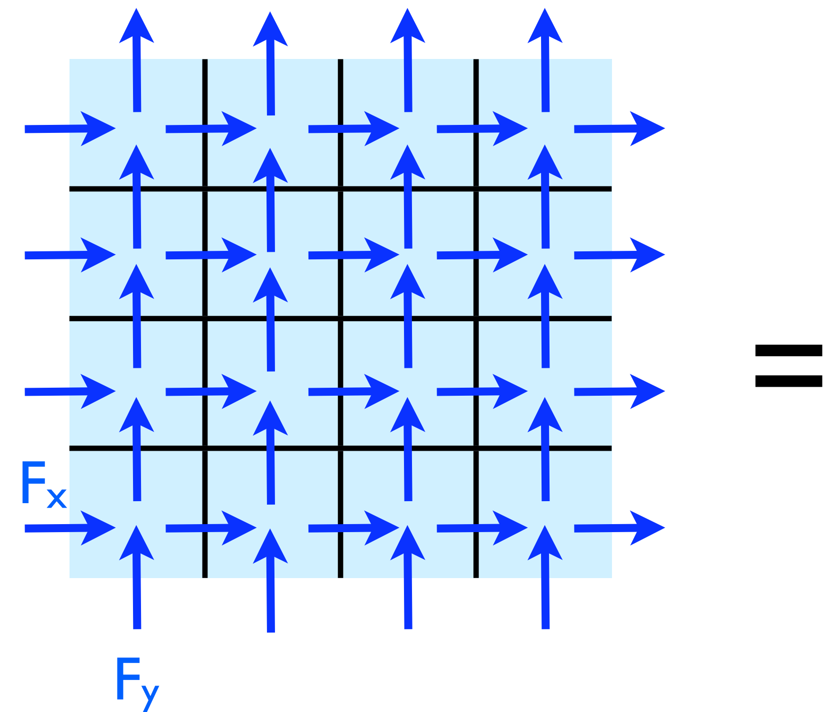
- We're using a 'central scheme' or 'Kurganov scheme'
- No Riemann solve; average over possible waves
- Averaging means shocks are smeared out compared to Riemann solvers; but much faster, simpler to code (particularly for RHD, MHD)

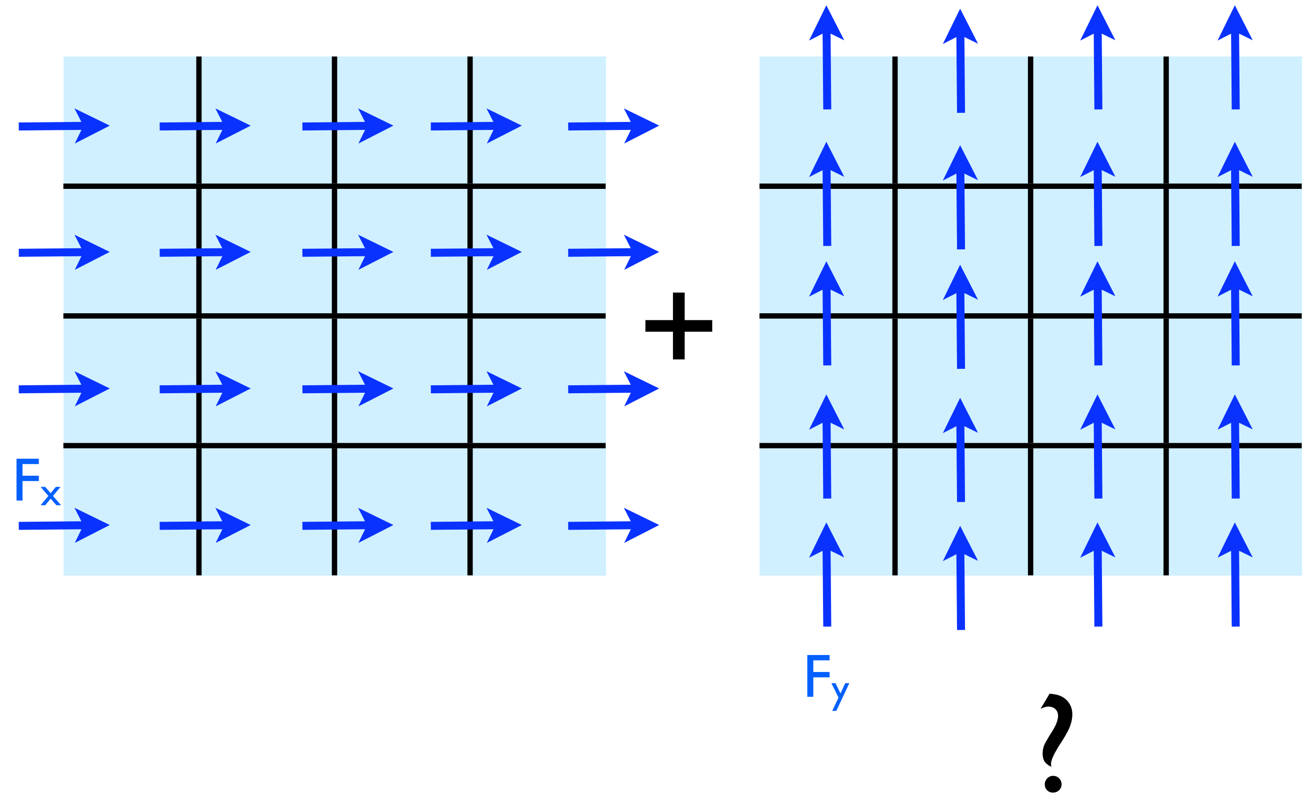


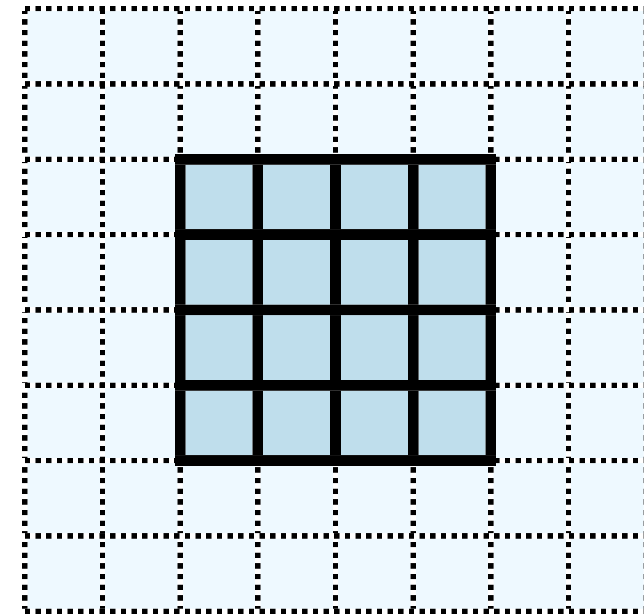
Del Zanna, Bucciantini (2002) A&A **390**:1177

Dimensional Splitting

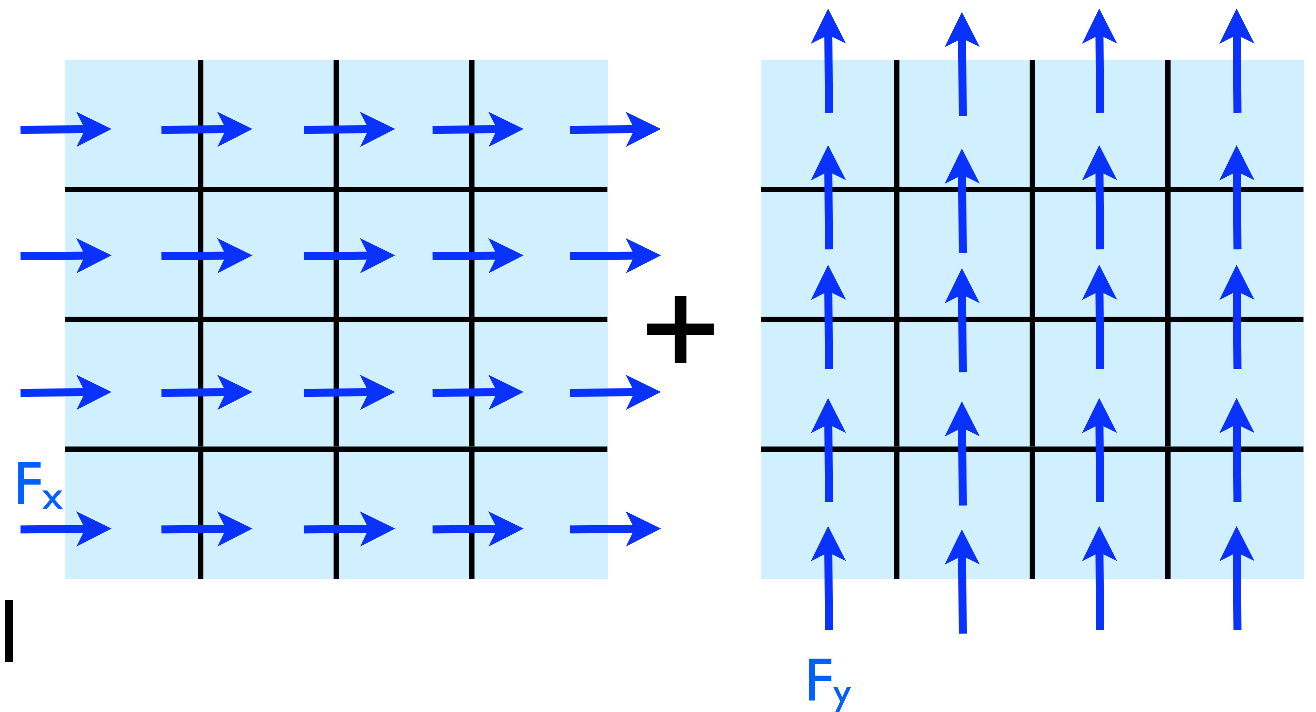
- Strang Splitting: Operators (including X and Y hydro operators) can be done separately, at cost of limiting time accuracy to Δt^2.
- Not at all obvious that should work as well as it does.
- Makes code much easier - get a 1d solver working, build 3d solver trivially







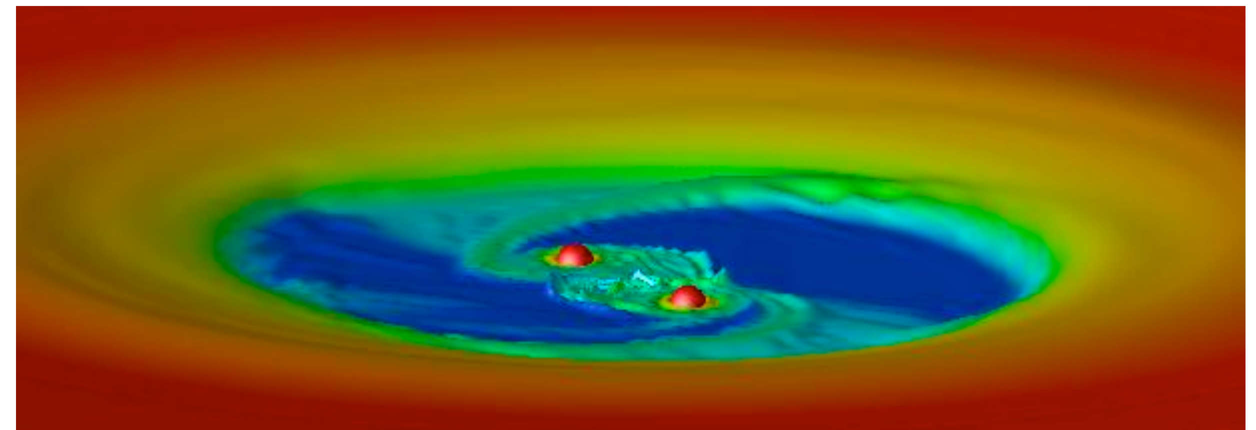
Hydrodynamics



- Finite volume dimensionally split central scheme
- Need only local info (+/- 2 zones in each dimension)
- Implemented with dimensional splitting; sweep in x, then y (then y, then x)

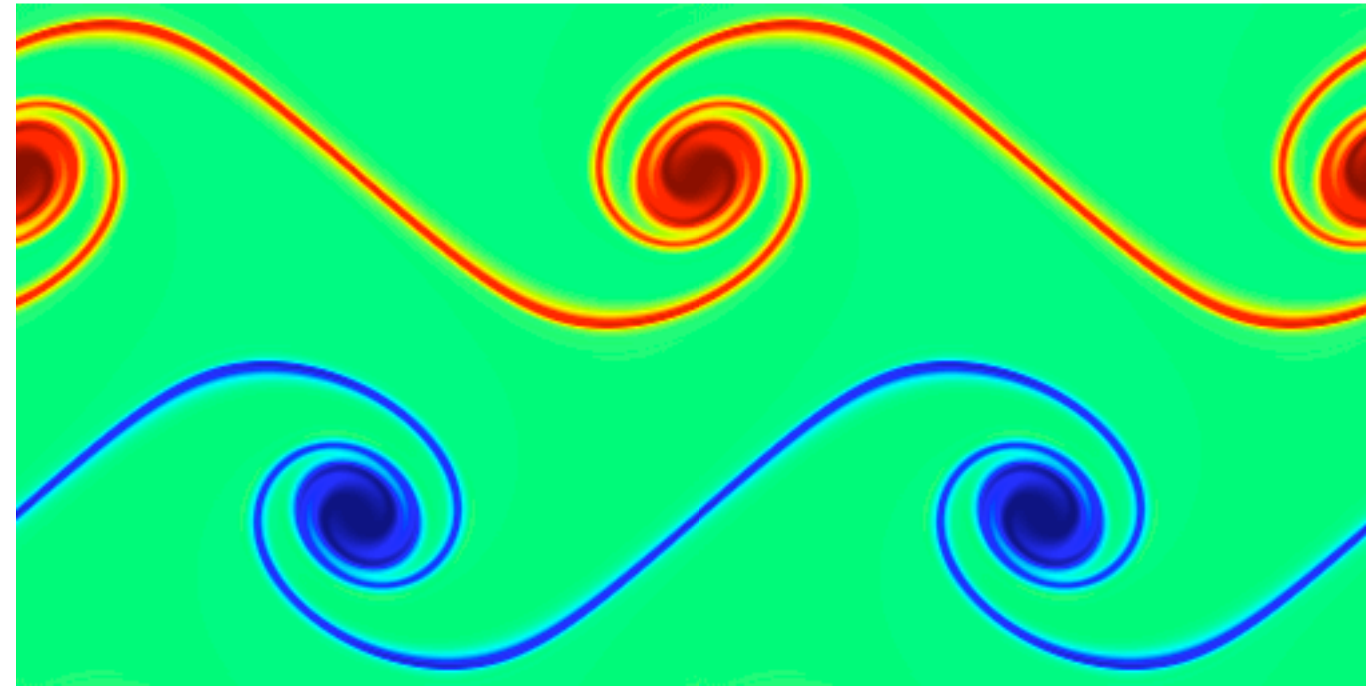
Other Hydrodynamic approaches

- Finite difference approaches; don't work in fluxes. Easier to incorporate some types of physics with high time accuracy.
- Parallelization issues same as finite volume codes.



Richard Günther, University of Tübingen.
http://www.tat.physik.uni-tuebingen.de/~rguenth/

Other Hydrodynamic approaches

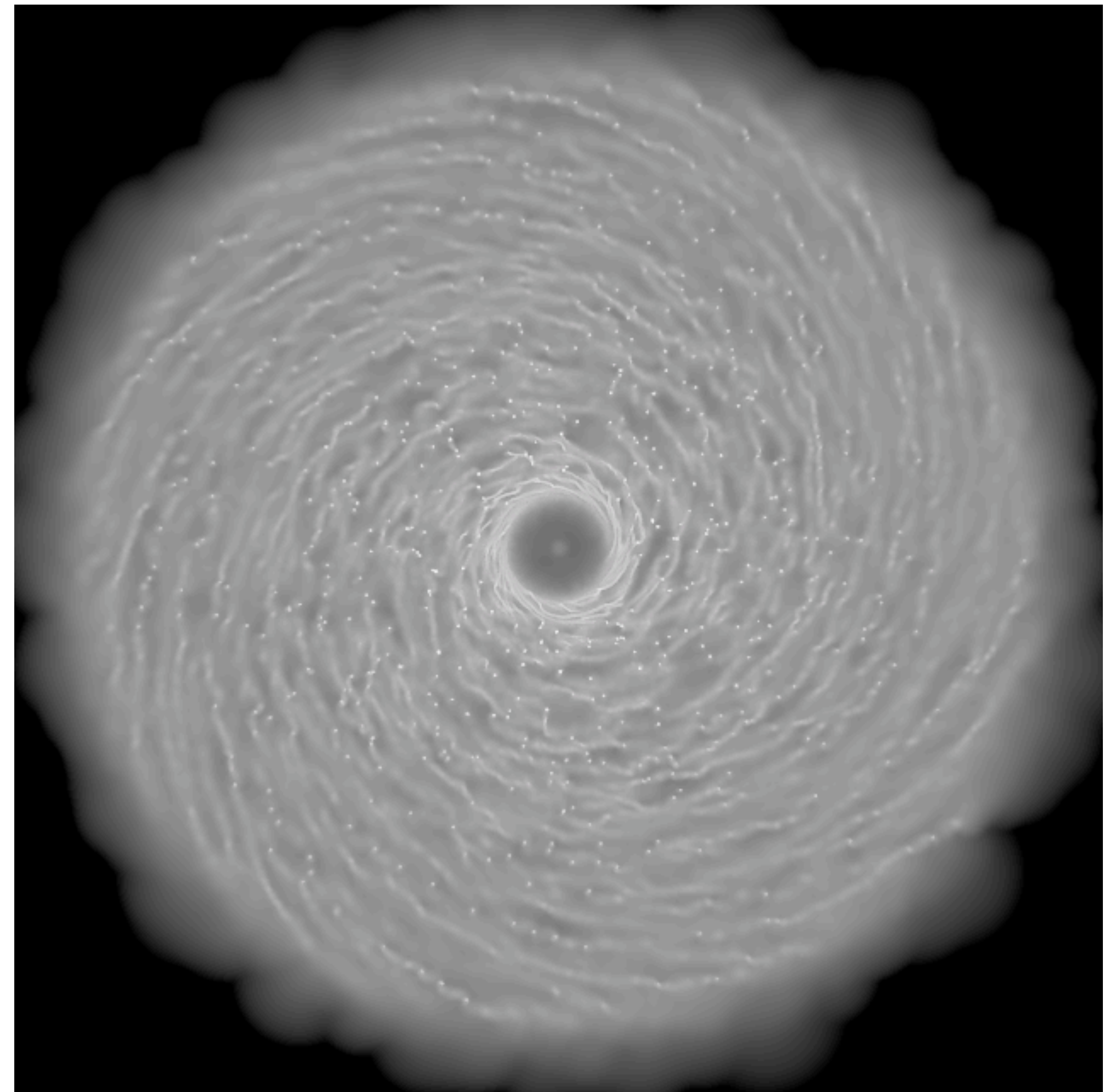


- Incompressible flows
- Additional complexity: elliptical solver (implicit scheme)
- What we have here + CG
- Or Multigrid: also mostly guardcell filling

Mike Zingale, SUNY Stony Brook
http://www.astro.sunysb.edu/mzingale/pyro/

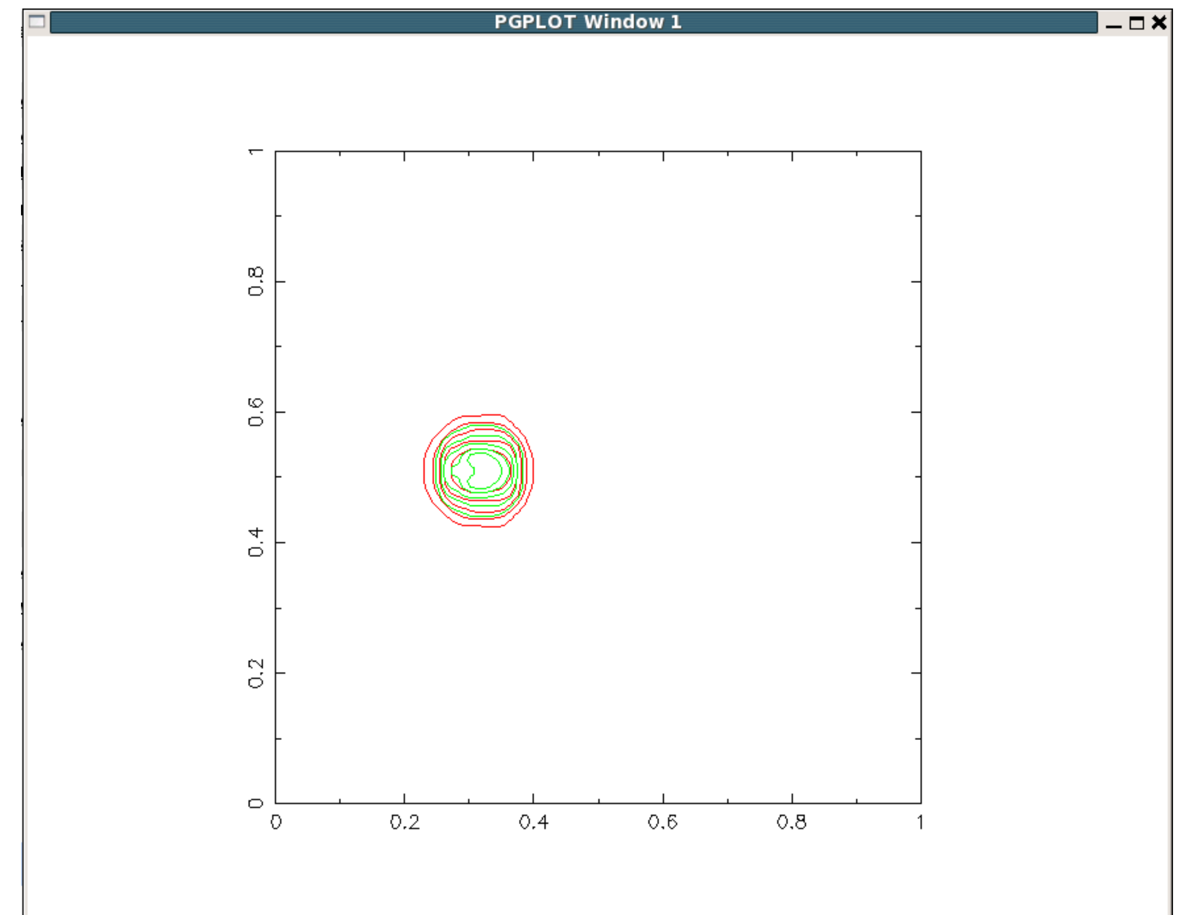
Other Hydrodynamic approaches

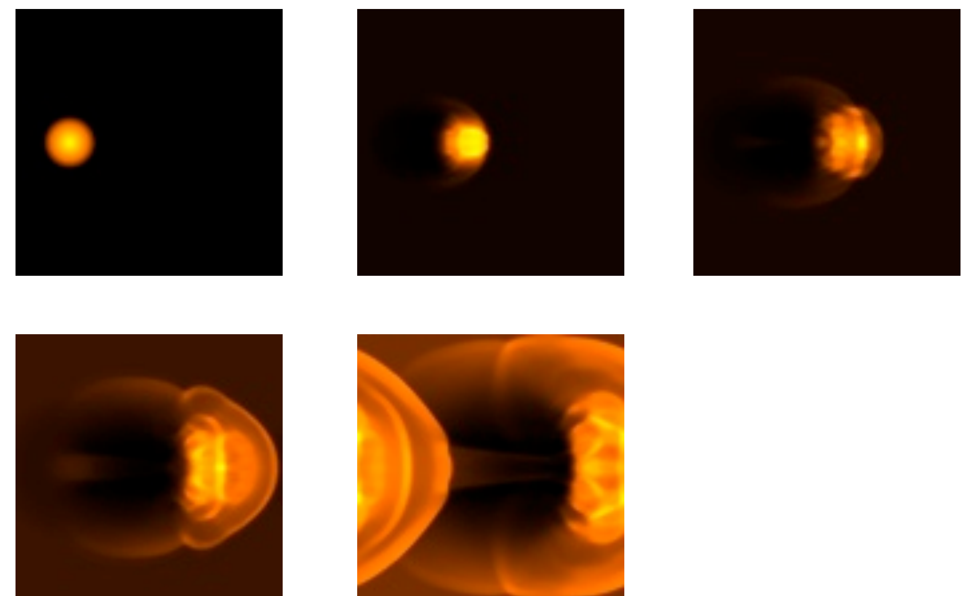
- SPH: no grid at all. Fluid parcels.
- Hard to do highly accurate schemes, but arguably better suited for some problems.
- Gadget-2
- Some of the same parallelization issues as N-body gravity (Thurs)



Single-Processor hydro code

- cd ~/pca/src/hydro
- make
- ./fixed_dt_singleproc
- Takes options:
  - --help
  - --nooutput
  - --simulation={0,1}
  - --npts=N
  - --{x,y}boundary={1,2,3}
  - --dt=X





Single-Processor hydro code

- Set initial conditions
- Apply boundary conditions, EOS
- Loop, calling *kl_timestep()* and maybe some output routines (*domain_plot()* - contours; *domain_output_ppm()* - images)
- Domain and all variables stored in a structure *d*.

```
/* apply boundary conditions and make thermodynamically consistant */
bcs[0] = xbc; bcs[1] = xbc;
bcs[2] = ybc; bcs[3] = ybc;
apply_all_bcs(&d,bcs);
domain_backward_dp_eos(&d);
domain_ener_internal_to_tot(&d);


/* main loop */

tick(&tt);
if (output) domain_plot(&d);
printf("Step\tdt\ttime\n");
for (time=0.,step=0; step < nsteps; step++, time+=2.*dt) {
        printf("%d\t%g\t%g\n", step, dt, time);
        if (output && ((step % outevery) == 0) ) {
                sprintf(filename,"dens_test_%d.ppm", outnum);
                domain_output_ppm(&d, outputvar, filename);
                domain_plot(&d);
                outnum++;
        }
        kl_timestep(&d, bcs, dt);
}
tock(&tt);

/* final output */
if (output) {
        sprintf(filename,"dens_test_%d.ppm", outnum);
        domain_output_ppm(&d, outputvar,filename);
        domain_closeplot();
}
if (simulation == 0) domain_output_dens_r(&d,"sedov-test.dat");

free_domain(&d);
return 0;
```

fixed_dt_singleproc.c line 94

Domain structure

- *d* contains size, coordinate information, and variables *dens*, *ener*, *momx*, *momy* (conserved vars) and *pres*, *temp* (related to others by EOS)

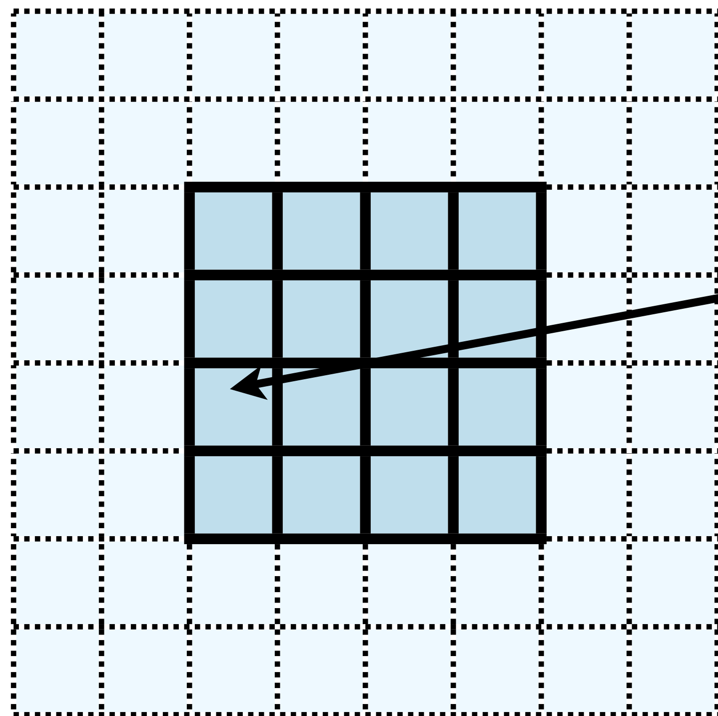
```
typedef struct {
        int Nx, Ny, Nguard;
        double *x, *y;
        double **dens;
        double **ener;
        double **pres;
        double **temp;
        double **momx;
        double **momy;

        short int alloced;
        int   enerflag;
} domain_t;
```

domain.h

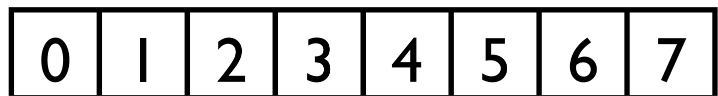
```
for (i=0; i<nxt; i++) {
        xx = d->x[i] - cx;
        xx2 = xx*xx;
        for (j=0; j<nyt; j++) {
                yy = d->y[j] - cy;
                r = sqrt(xx2 + yy*yy);

                if (r < psize) {
                        d->pres[i][j] = pexp;
                } else {
                        d->pres[i][j] = backgroundpres;
                }
                d->dens[i][j] = backgrounddens;
                d->momx[i][j] = 0;
                d->momy[i][j] = 0;
        }
}
```

ics.c line 87

Nguard Nx Nguard



```
d->dens[2][3];
d->momx[2][3];
d->momy[2][3];
d->ener[2][3];
```




```
typedef struct {
        int Nx, Ny, Nguard;
        double *x, *y;
        double **dens;
        double **ener;
        double **pres;
        double **temp;
        double **momx;
        double **momy;

        short int alloced;
        int   enerflag;
} domain_t;
```

domain.h

```
for (i=0; i<nxt; i++) {
        xx = d->x[i] - cx;
        xx2 = xx*xx;
        for (j=0; j<nyt; j++) {
                yy = d->y[j] - cy;
                r = sqrt(xx2 + yy*yy);

                if (r < psize) {
                        d->pres[i][j] = pexp;
                } else {
                        d->pres[i][j] = backgroundpres;
                }
                d->dens[i][j] = backgrounddens;
                d->momx[i][j] = 0;
                d->momy[i][j] = 0;
        }
}
```

ics.c line 87

Timestep routine

- For each row,
  - extract it
  - Do 1d hydro (X sweep)
- For each column
  - extract it
  - Do 1d hydro (Y sweep)
- Apply boundary conditions, EOS
- Y sweep, X sweep
- *kl_row_rk2()*: main 1d hydro routine, black box

```
init_row(&row, nc+2*ng, d->x[0], d->x[nc+2*ng-1], d->enerflag);
init_row(&col, nr+2*ng, d->y[0], d->y[nr+2*ng-1], d->enerflag);

/* xsweep along all rows */
for (i=ng; i<nr+ng; i++) {
        domain_extract_row(d, &row, i, XDIR);
        kl_row_rk2(&row, dt);
        domain_insert_row(d, &row, i, XDIR);
}

/* ysweep along all cols */
for (i=ng; i<nc+ng; i++) {
        domain_extract_row(d, &col, i, YDIR);
        kl_row_rk2(&col, dt);
        domain_insert_row(d, &col, i, YDIR);
}

apply_all_bcs(d,bcs,gridi,gridj);
domain_forward_eos(d);

for (i=ng; i<nc+ng; i++) {
        domain_extract_row(d, &col, i, YDIR);
        kl_row_rk2(&col, dt);
        domain_insert_row(d, &col, i, YDIR);
}

/* xsweep along all rows */
for (i=ng; i<nr+ng; i++) {
        domain_extract_row(d, &row, i, XDIR);
        kl_row_rk2(&row, dt);
        domain_insert_row(d, &row, i, XDIR);
}
apply_all_bcs(d,bcs,gridi,gridj);
domain_forward_eos(d);

free_row(&row);
free_row(&col);
```

kurganovlevy_timestep.c line 23-67

Timestep routine

- After first X,Y sweep, boundary conditions and the equation of state is applied;
- (can't be before all sweeps are done)
- Same with the second X,Y sweep
- All rows must be completed before the columns start and vice versa

```c
init_row(&row, nc+2*ng, d->x[0], d->x[nc+2*ng-1], d->enerflag);
init_row(&col, nr+2*ng, d->y[0], d->y[nr+2*ng-1], d->enerflag);

/* xsweep along all rows */
for (i=ng; i<nr+ng; i++) {
        domain_extract_row(d, &row, i, XDIR);
        kl_row_rk2(&row, dt);
        domain_insert_row(d, &row, i, XDIR);
}

/* ysweep along all cols */
for (i=ng; i<nc+ng; i++) {
        domain_extract_row(d, &col, i, YDIR);
        kl_row_rk2(&col, dt);
        domain_insert_row(d, &col, i, YDIR);
}

apply_all_bcs(d,bcs,gridi,gridj);
domain_forward_eos(d);

for (i=ng; i<nc+ng; i++) {
        domain_extract_row(d, &col, i, YDIR);
        kl_row_rk2(&col, dt);
        domain_insert_row(d, &col, i, YDIR);
}

/* xsweep along all rows */
for (i=ng; i<nr+ng; i++) {
        domain_extract_row(d, &row, i, XDIR);
        kl_row_rk2(&row, dt);
        domain_insert_row(d, &row, i, XDIR);
}
apply_all_bcs(d,bcs,gridi,gridj);
domain_forward_eos(d);

free_row(&row);
free_row(&col);
```

kurganovlevy_timestep.c line 23-67

SciNet High Performance Computing

Timestep routine

- All the work is being done in these nice big work-intensive loops.
- Just makes you want to OpenMP them!
- Remember: easiest to declare private variables in parallel section.

```
init_row(&row, nc+2*ng, d->x[0], d->x[nc+2*ng-1], d->enerflag);
init_row(&col, nr+2*ng, d->y[0], d->y[nr+2*ng-1], d->enerflag);

/* xsweep along all rows */
for (i=ng; i<nr+ng; i++) {
        domain_extract_row(d, &row, i, XDIR);
        kl_row_rk2(&row, dt);
        domain_insert_row(d, &row, i, XDIR);
}

/* ysweep along all cols */
for (i=ng; i<nc+ng; i++) {
        domain_extract_row(d, &col, i, YDIR);
        kl_row_rk2(&col, dt);
        domain_insert_row(d, &col, i, YDIR);
}

apply_all_bcs(d,bcs,gridi,gridj);
domain_forward_eos(d);

for (i=ng; i<nc+ng; i++) {
        domain_extract_row(d, &col, i, YDIR);
        kl_row_rk2(&col, dt);
        domain_insert_row(d, &col, i, YDIR);
}

/* xsweep along all rows */
for (i=ng; i<nr+ng; i++) {
        domain_extract_row(d, &row, i, XDIR);
        kl_row_rk2(&row, dt);
        domain_insert_row(d, &row, i, XDIR);
}
apply_all_bcs(d,bcs,gridi,gridj);
domain_forward_eos(d);

free_row(&row);
free_row(&col);
```

kurganovlevy_timestep.c
line 23-67

```
int domain_forward_eos(domain_t *d) {

        int i,j;
        int nxt, nyt, nguard;

        nguard = d->Nguard;
        nxt = d->Nx+2*nguard;
        nyt = d->Ny+2*nguard;

#pragma  omp for
        for (i=0; i<nxt; i++) {
                for (j=0; j<nyt; j++) {
                        d->pres[i][j] = (EOS_GAMMA - 1.) * d->dens[i][j] * d->ener[i][j];
                        d->temp[i][j] = d->ener[i][j] * ((EOS_GAMMA-1.)/EOS_GASCONST) * EOS_ABAR
                }
        }

        return 0;
}
```

- cp kurganovlevy_timestep.c kurganovlevy_timestep_omp.c
- Add a line to the makefile

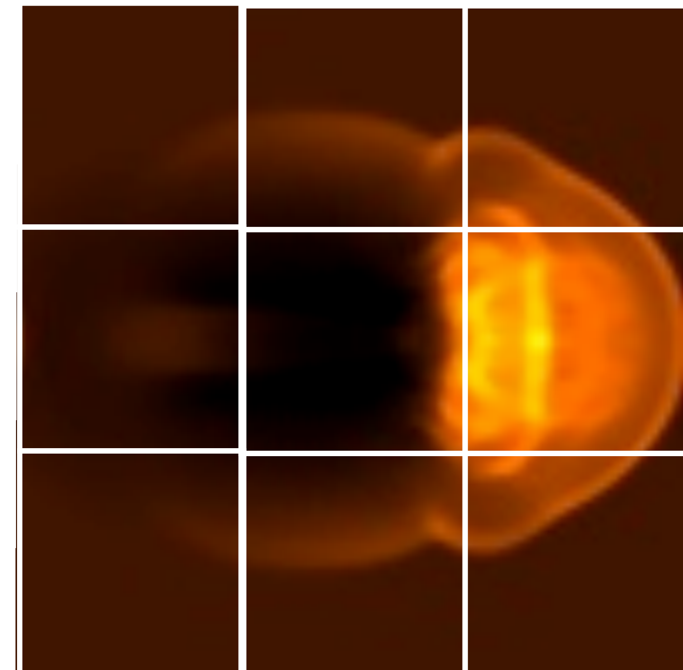
```
kurganovlevy_timestep_omp.o: kurganovlevy_timestep_omp.c
        $(OMPCC) $(CFLAGS) -c $<
```

- And copy the 2 lines for the 'fixed_dt_singleproc' target, but call it fixed_dt_omp, use the new _omp.c, and add -lgomp on the link line:

```
fixed_dt_omp: fixed_dt_singleproc.o kurganovlevy.o
kurganovlevy_timestep_omp.o domain.o domain_output.o row.o eos.o
reconstruction.o boundary_conds.o ics.o $(UTILS)
        ${LD} -o $@ -g fixed_dt_singleproc.o kurganovlevy.o
kurganovlevy_timestep_omp.o domain.o domain_output.o row.o eos.o
reconstruction.o boundary_conds.o ics.o $(UTILS) $(LDFLAGS)  $
(PGPLIBS) -lgomp
```

MPIing the code

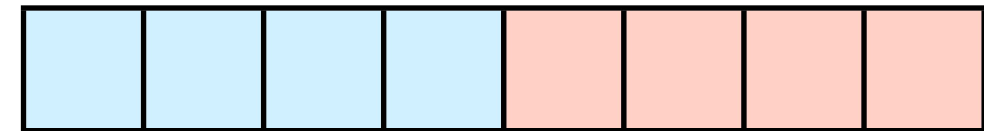
- Domain decomposition
- Lots of data - ensures locality
- How are we going to handle getting non-local information across processors?

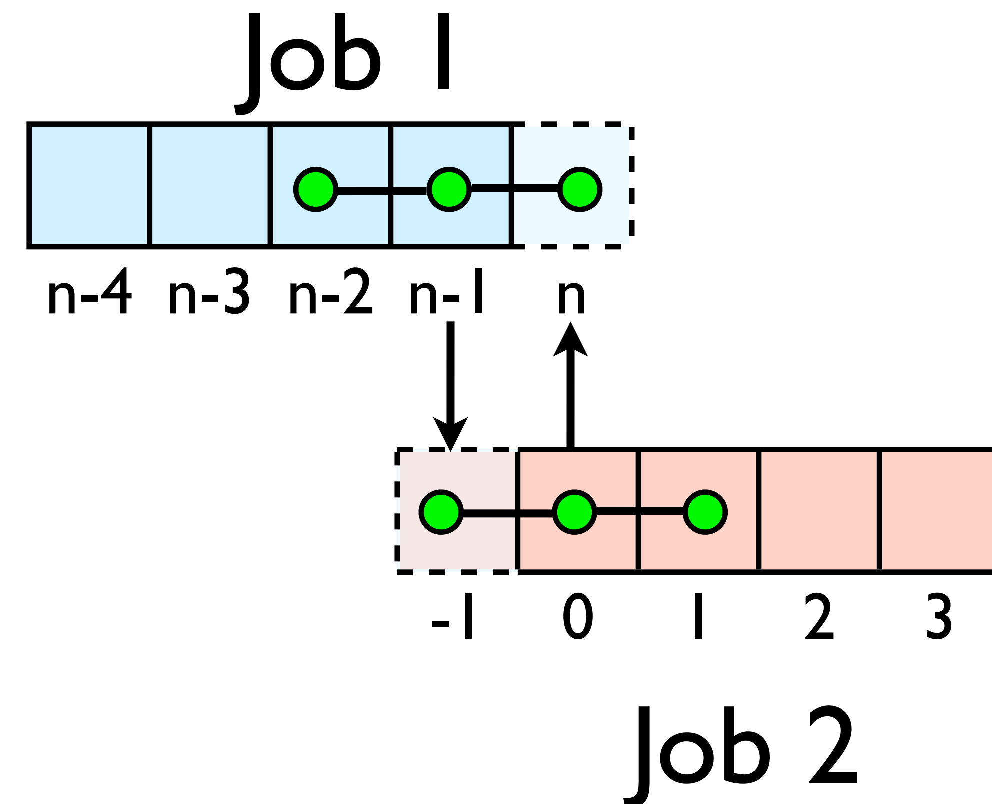


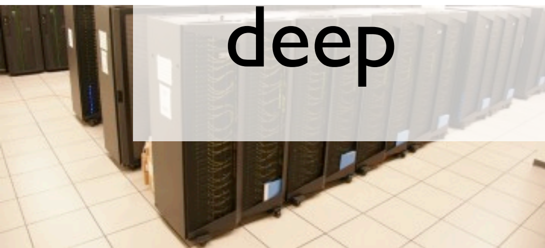
Guardcells

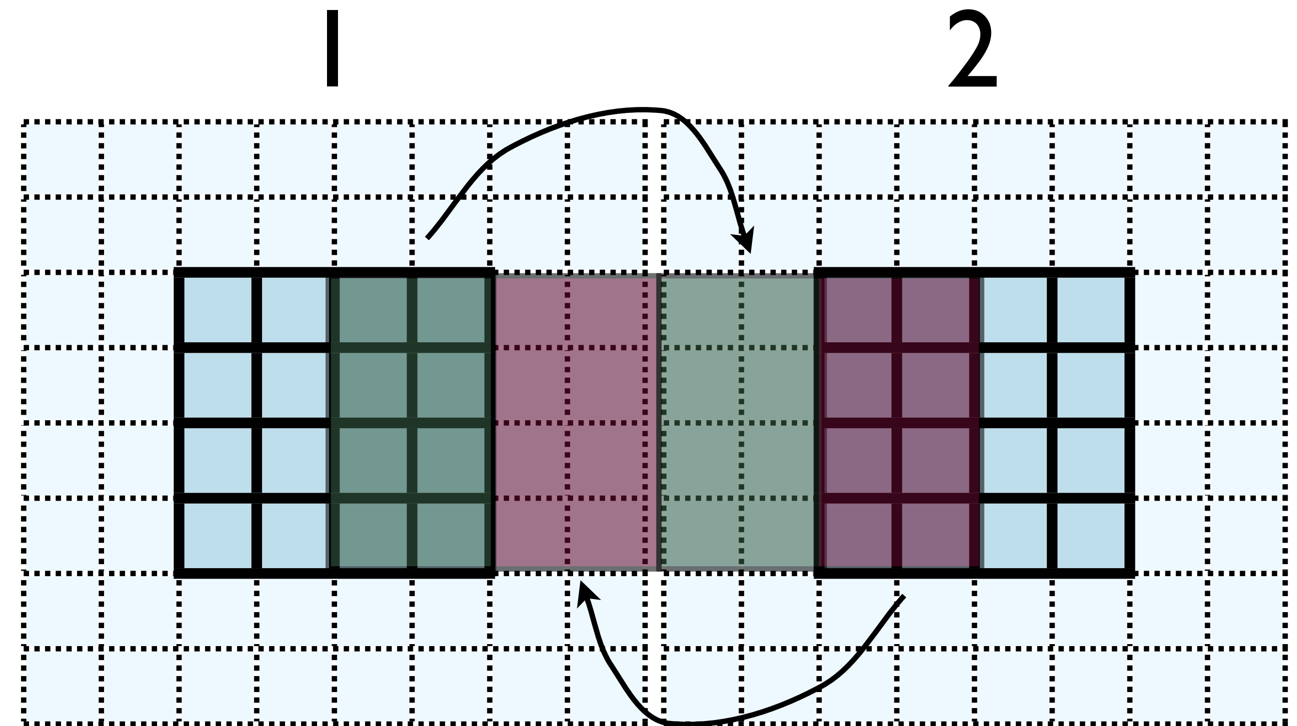
- Works for parallel decomposition!
- Job 1 needs info on Job 2s 0th zone, Job 2 needs info on Job 1s last zone
- Pad array with 'guardcells' and fill them with the info from the appropriate node by message passing or shared memory
- Hydro code: need guardcells 2 deep

Global Domain









Guard cell fill

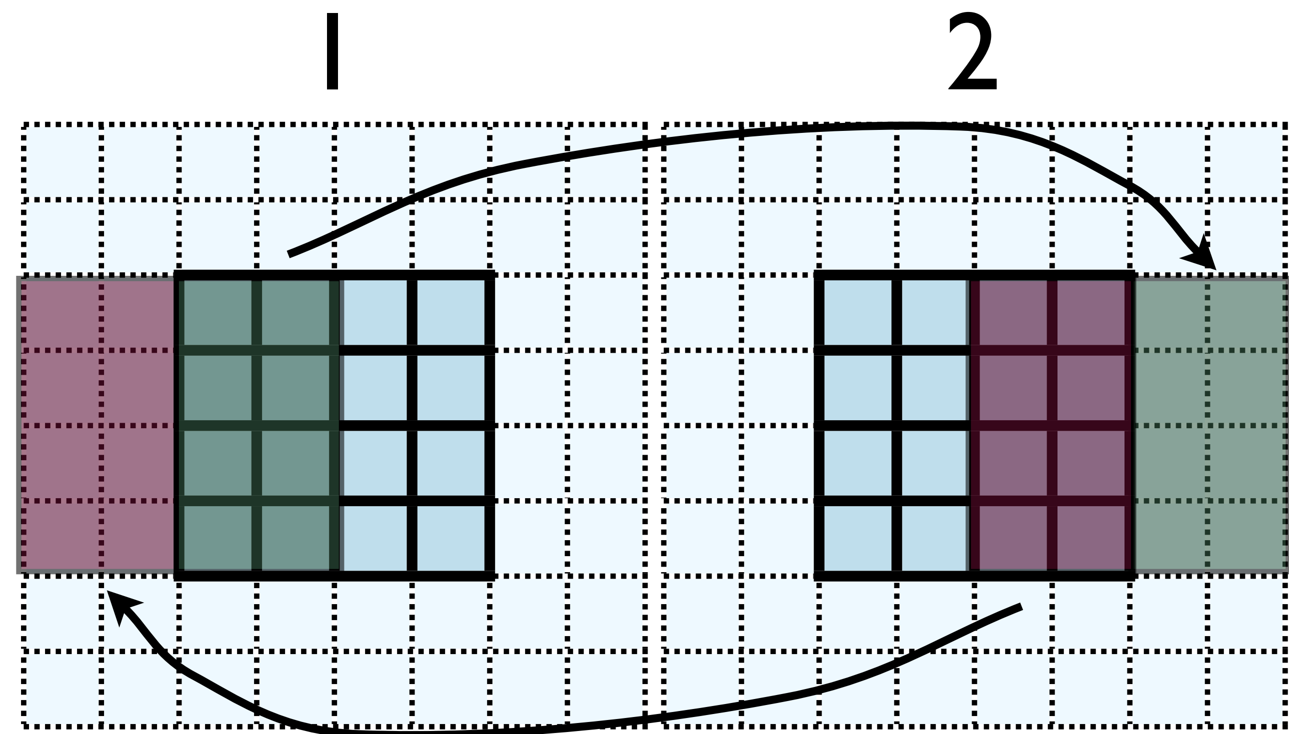
- When we're doing boundary conditions.
- If BC = -1,-2,-3, fill guardcells with usual BC stuff
- if BC = +P, then our neighbor is P; swap GC with P.

1: dens[nx:nx+ng-1][ng:ng+ny-1]
→ 2: dens[0:ng-1][ng:ng+ny-1]

2:dens[ng:2*ng-1][ng:ng+ny-1]
→ 1: dens[nx+ng:nx+2*ng-1][ng:ng+ny-1]

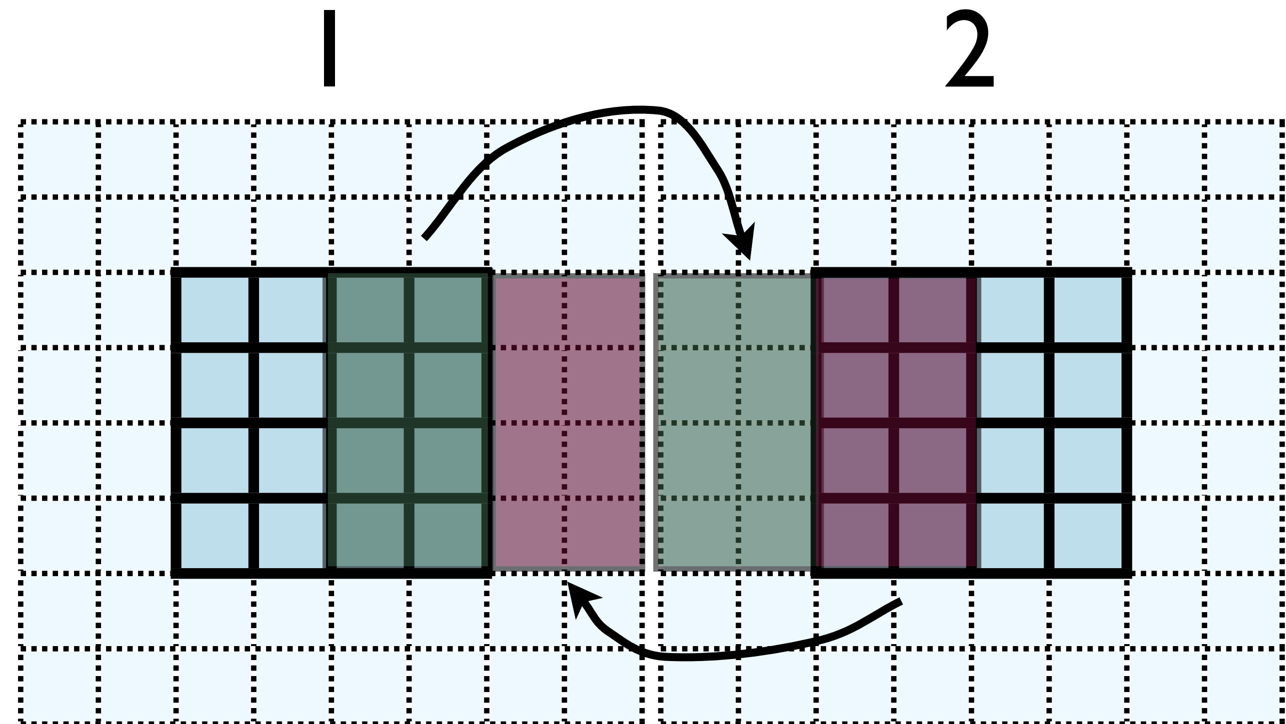
ny*ng values to swap

Cute way for Periodic BCs



- Actually make the decomposed mesh periodic;
- Make the far ends of the mesh neighbors
- Don't know the difference between that and any other neighboring grid

Implementing in MPI



- No different in principle than diffusion
- Just more values
- And more variables: dens, ener, temp....
- Simplest way: copy all the variables into an NVAR*Ny*ng sized buffer

1: dens[nx:nx+ng-1][ng:ng+ny-1]
→ 2: dens[0:ng-1][ng:ng+ny-1]

2:dens[ng:2*ng-1][ng:ng+ny-1]
→ 1: dens[nx+ng:nx+2*ng-1][ng:ng+ny-1]

ny*ng values to swap

Implementing in MPI

- No different in principle than diffusion
- Just more values
- And more variables: dens, ener, temp....
- Simplest way: copy all the variables into an NVAR*Ny*ng sized buffer

1

2

SciNet High Performance Computing

Implementing in MPI

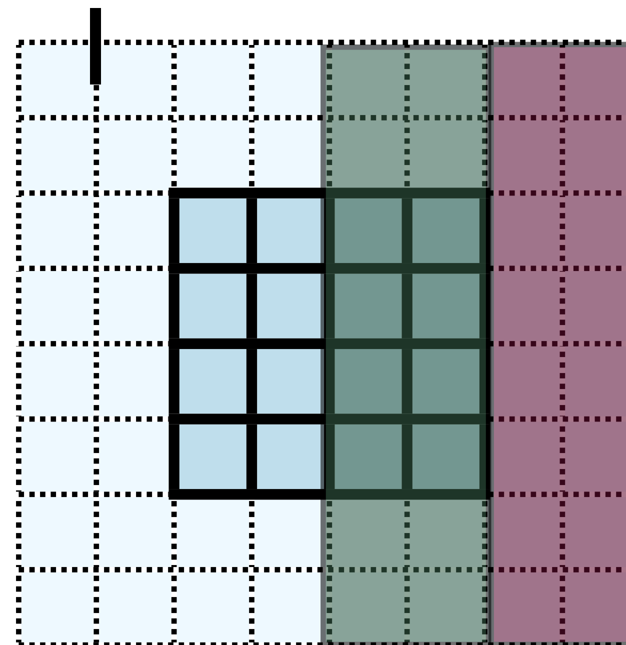
- No different in principle than diffusion
- Just more values
- And more variables: dens, ener, temp....
- Simplest way: copy all the variables into an NVAR*Ny*ng sized buffer

```
ioff = (dirn == XLEFT_DIR ? ng : nx);
count = 0;
for (i = ioff; i < ioff + ng; i++) {
        for (j=0; j<2*ng+ny; j++) {
                out[NVARS*count + DENSVAR] = d->dens[i][j];
                out[NVARS*count + PRESVAR] = d->pres[i][j];
                out[NVARS*count + ENERVAR] = d->ener[i][j];
                out[NVARS*count + TEMPVAR] = d->temp[i][j];
                out[NVARS*count + MOMXVAR] = d->momx[i][j];
                out[NVARS*count + MOMYVAR] = d->momy[i][j];
                count++;
        }
}
ierr = MPI_Sendrecv((void *)out, bufsize, MPI_DOUBLE_PRECISION, neigh, dirn,
                    (void *)in,  bufsize, MPI_DOUBLE_PRECISION, neigh, otherdirn,
                    MPI_COMM_WORLD, &status);


ioff = (dirn == XLEFT_DIR ? 0 : nx+ng);
count = 0;
for (i = ioff; i < ioff + ng; i++) {
        for (j=0; j<2*ng+ny; j++) {
                d->dens[i][j] = in[NVARS*count + DENSVAR];
                d->pres[i][j] = in[NVARS*count + PRESVAR];
                d->ener[i][j] = in[NVARS*count + ENERVAR];
                d->temp[i][j] = in[NVARS*count + TEMPVAR];
                d->momx[i][j] = in[NVARS*count + MOMXVAR];
                d->momy[i][j] = in[NVARS*count + MOMYVAR];
                count++;
        }
}
```

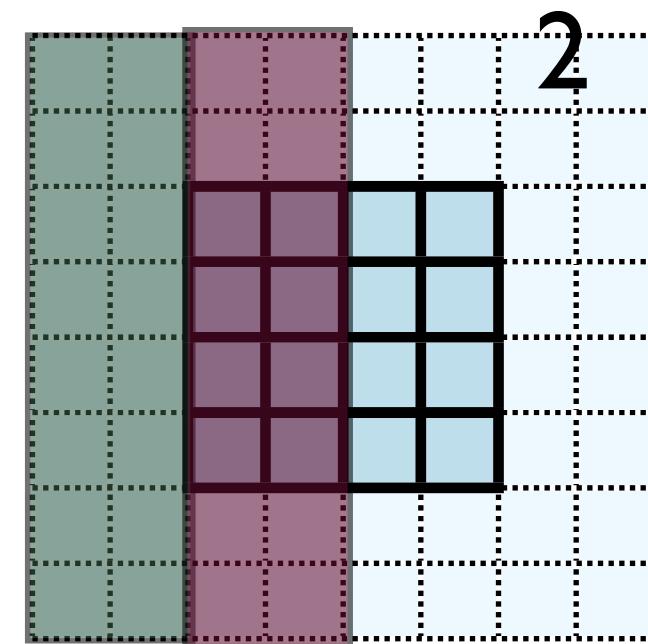
Implementing in MPI

- This approach is simple, but introduces extraneous copies
- Memory bandwidth is already a bottleneck for these codes
- It would be nice to just point at the start of the guardcell data and have MPI read it from there.

Implementing in MPI



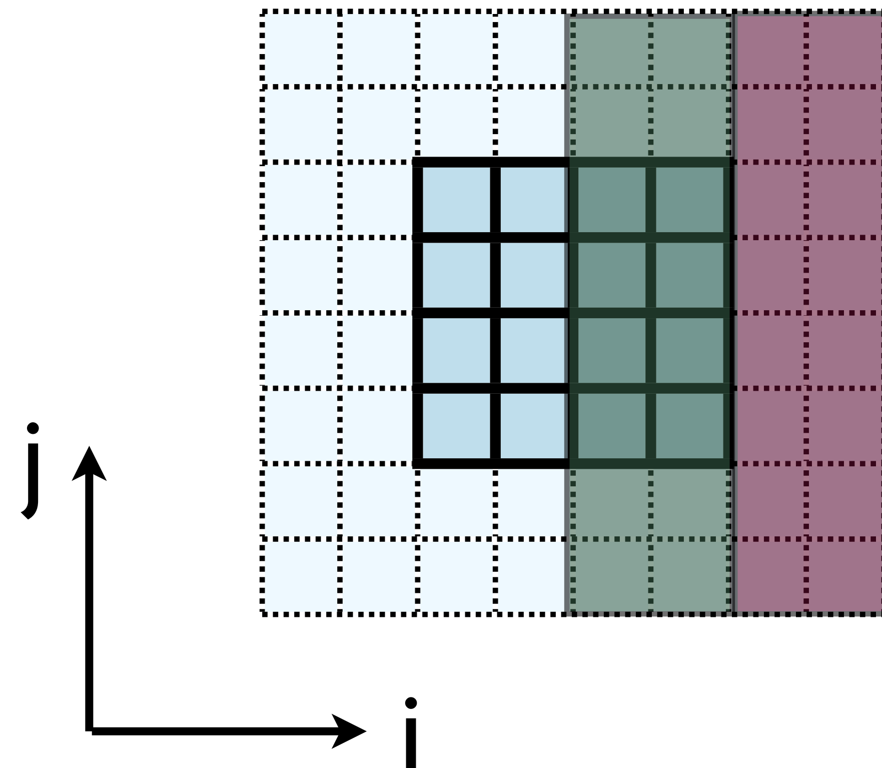
- Let me make one simplification for now; copy whole stripes
- This isn't necessary, but will make stuff simpler at first
- Only a cost of $2xNg^2 = 8$ extra cells (small fraction of ~200-2000 that would normally be copied)

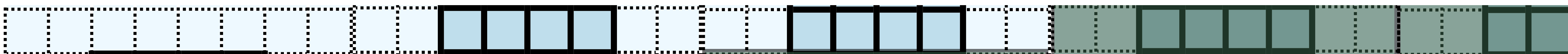


SciNet High Performance Computing

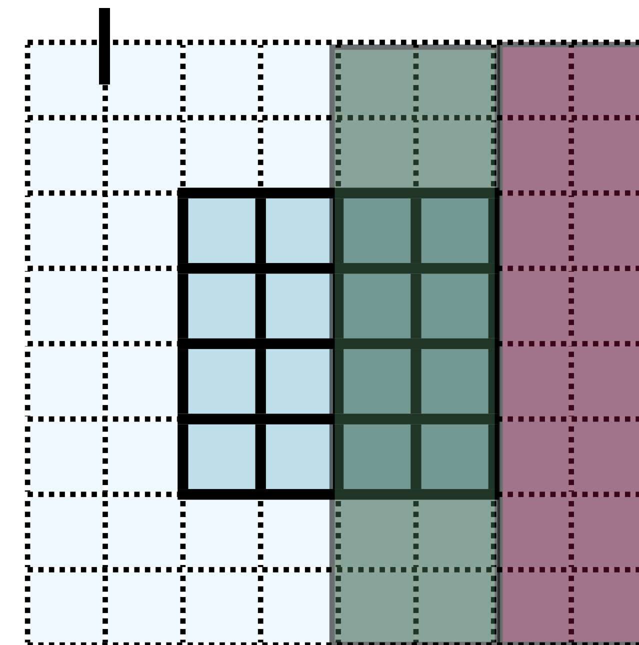
Implementing in MPI

- Recal how 2d memory is laid out in C
- x gcs or boundary values contiguous





Implementing in MPI



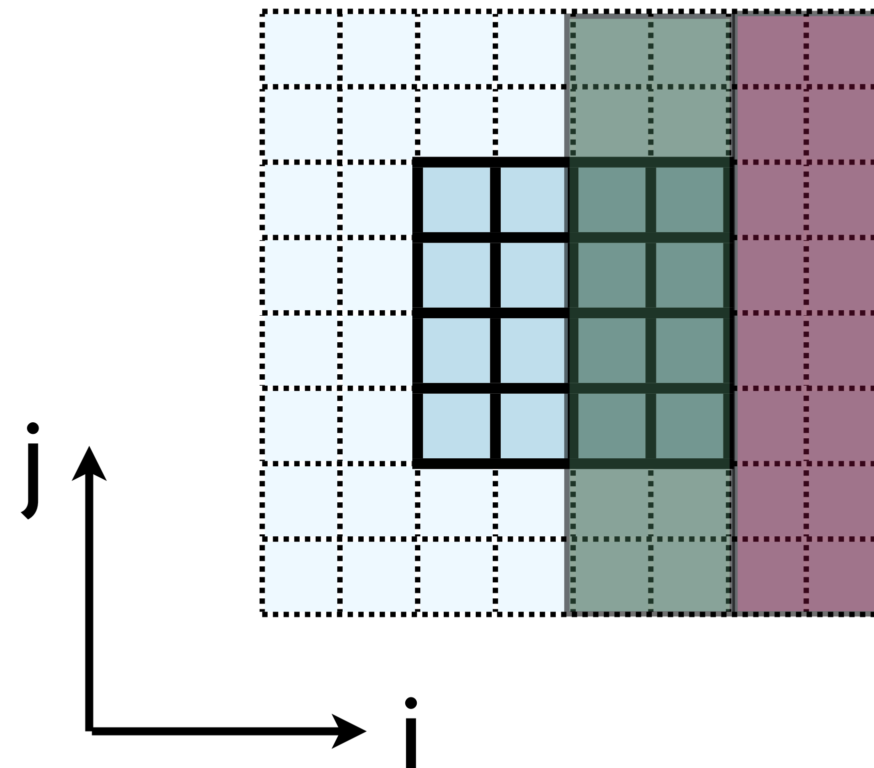
- Creating MPI Data types.
- MPI_Type_contiguous: simplest case. Lets you build a string of some other type.

Count   OldType &NewType

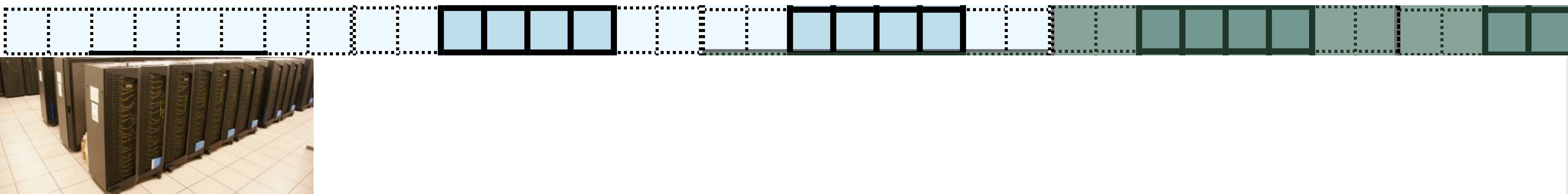
```
MPI_Datatype xbctype;

ierr = MPI_Type_contiguous(Nguard*(Ny+2*Nguard), MPI_DOUBLE, &xbctype);
ierr = MPI_Type_commit(&xbctype);

MPI_Send(&(d->dens[nx][0]), 1, xbctype, ....)

ierr = MPI_Type_free(&xbctype);
```

Implementing in MPI

- Not super exciting; could just as easily done this without fancy types...

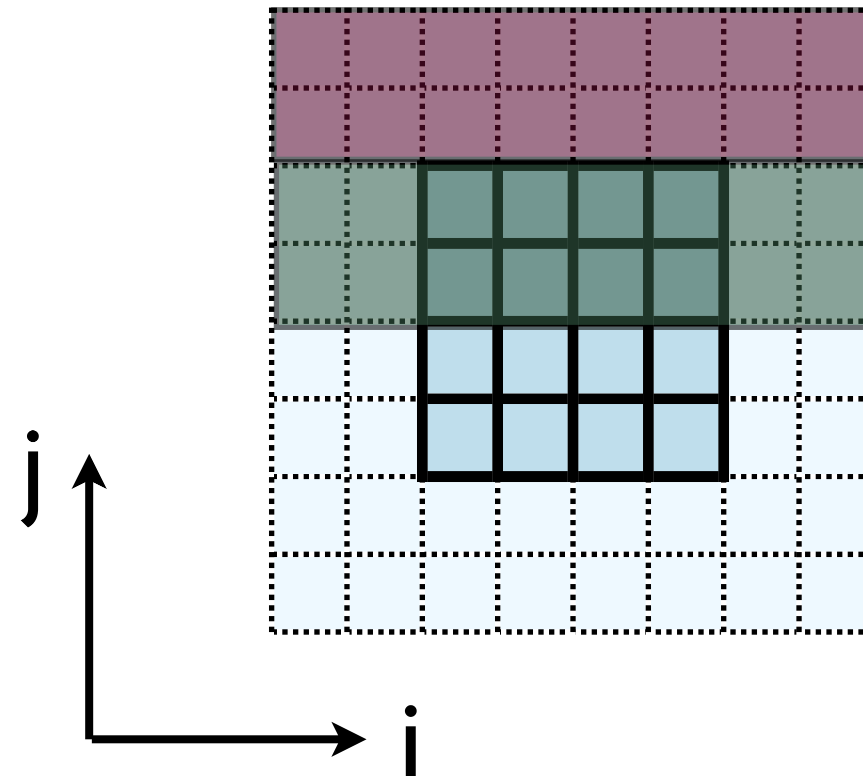



```
MPI_Send(&(d->dens[nx][0]), Nguard*(Ny+2*Nguard), MPI_DOUBLE, ....)
```

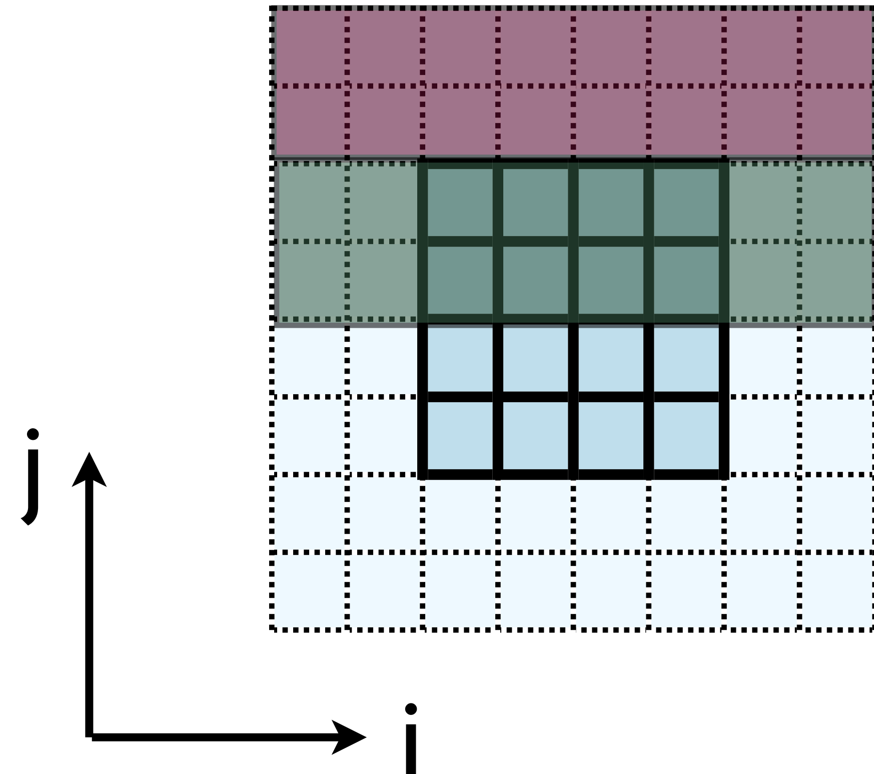


Implementing in MPI

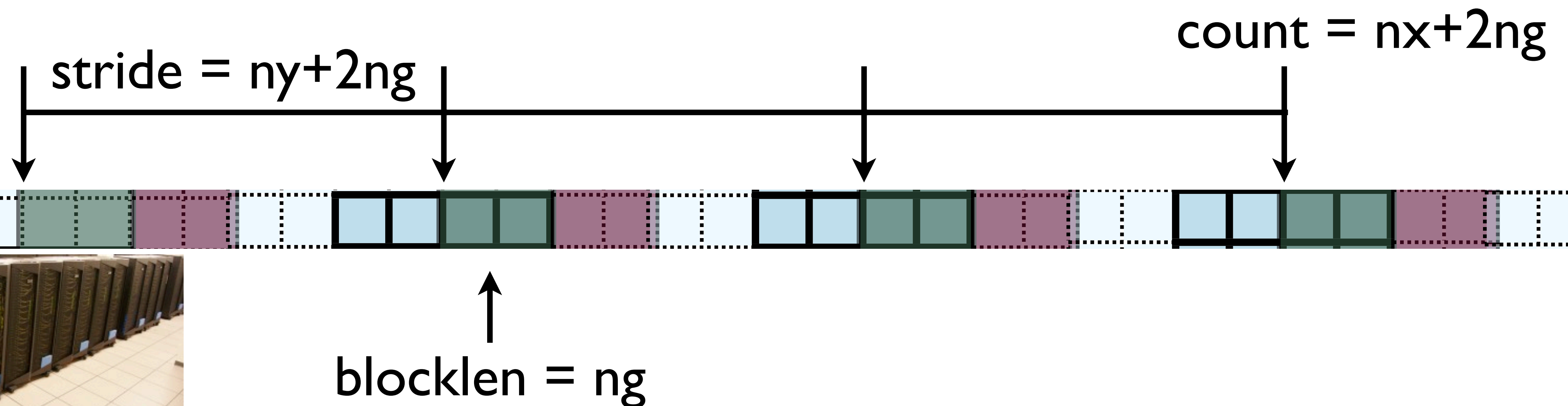
- But how do we do something like this for the y boundary conditions?
- Not contiguous - jumps around.   Ng zones every (Ny+2*Ng)



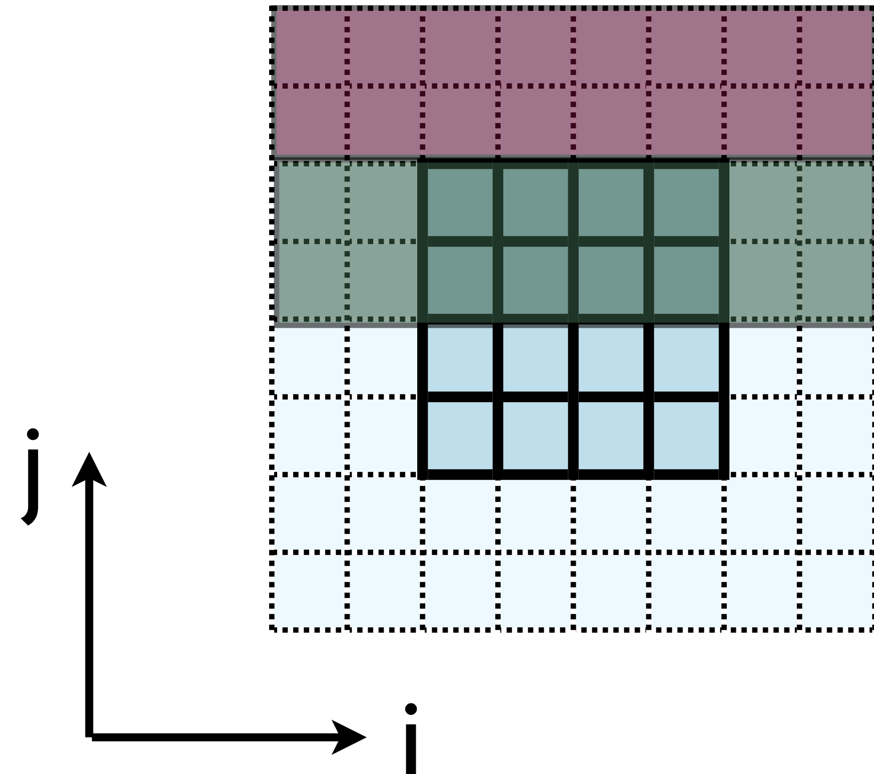
Implementing in MPI



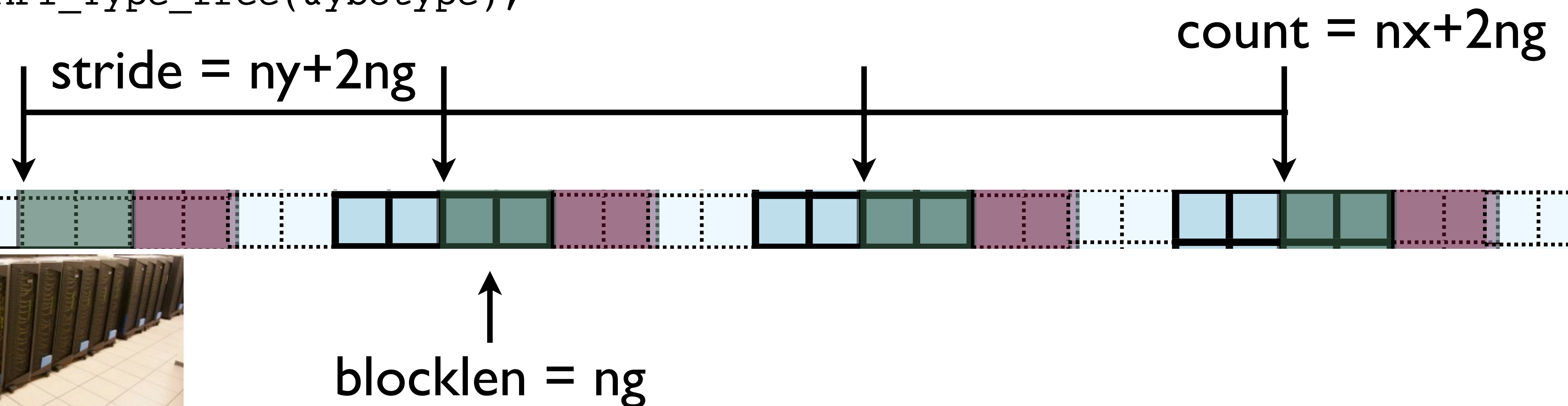

```
int MPI_Type_vector(
        int count,
        int blocklen,
        int stride,
        MPI_Datatype old_type,
        MPI_Datatype *newtype )
```



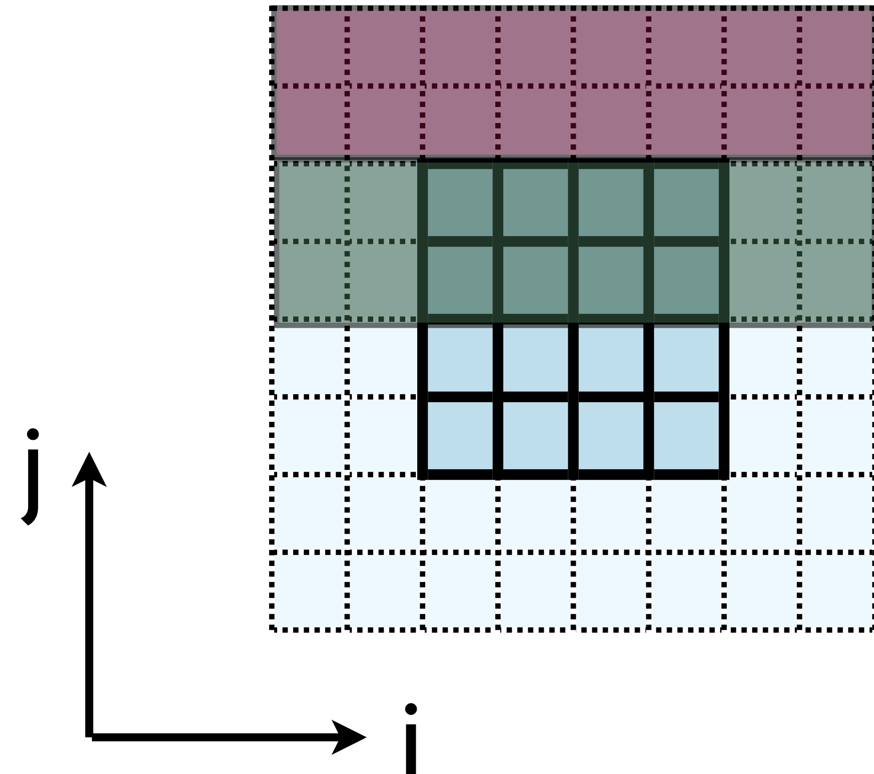
Implementing in MPI



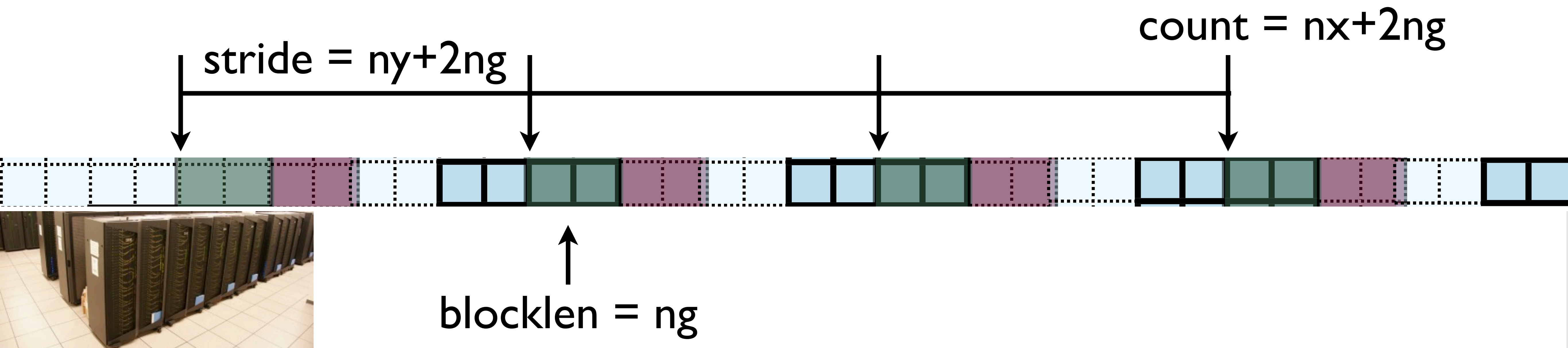

```
ierr = MPI_Type_vector((nx+2*ng),
        ng, (ny+2*ng), MPI_DOUBLE,
        &ybctype);
ierr = MPI_Type_commit(&ybctype);

ierr = MPI_Send(&(d->dens[0][ny]), 1, ybctype, ....)

ierr = MPI_Type_free(&ybctype);
```



Implementing in MPI

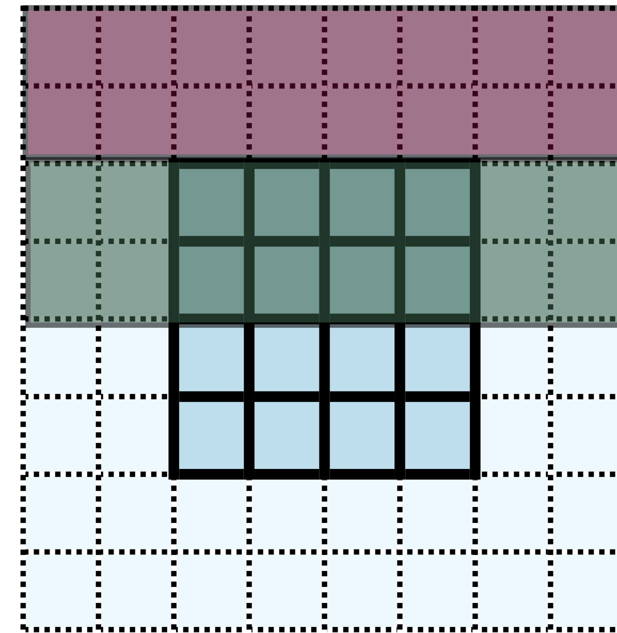


- Check: total amount of data = blocklen*count = ng(nx+2ng)
- Skipped over stride*count = (nx+2 ng) (ny+2 ng)



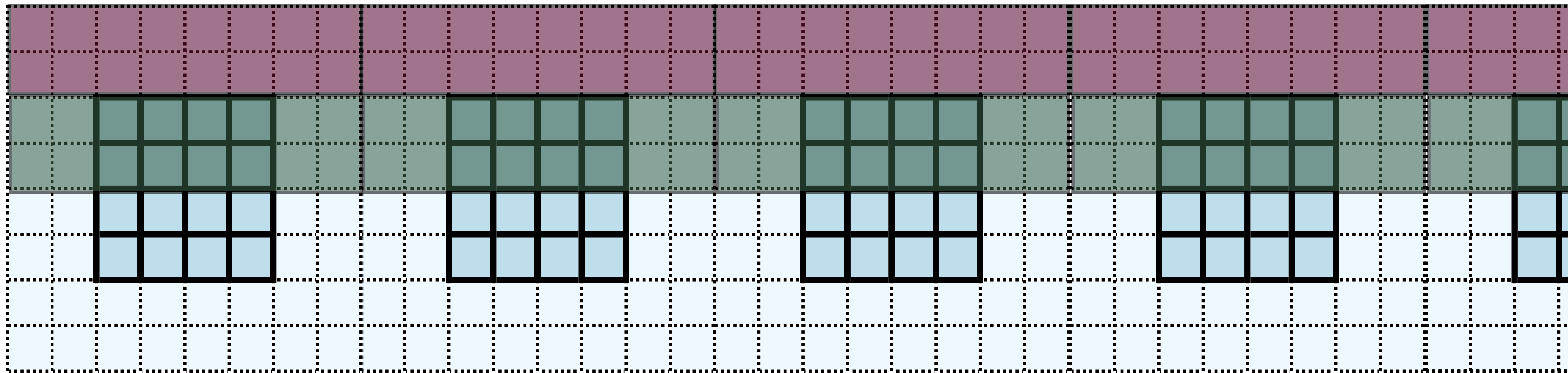
SciNet
High Performance Computing

Implementing in MPI

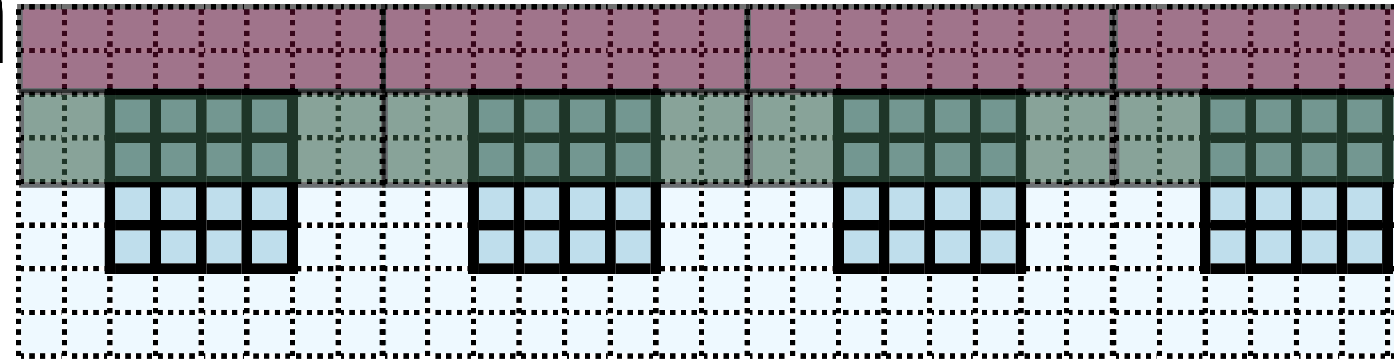


- So we could do this once per variable and send NVARS messages.
- NVARS * latency hit.
- However, I know something about how the memory is laid out...

Implementing in MPI



Implementing in MPI

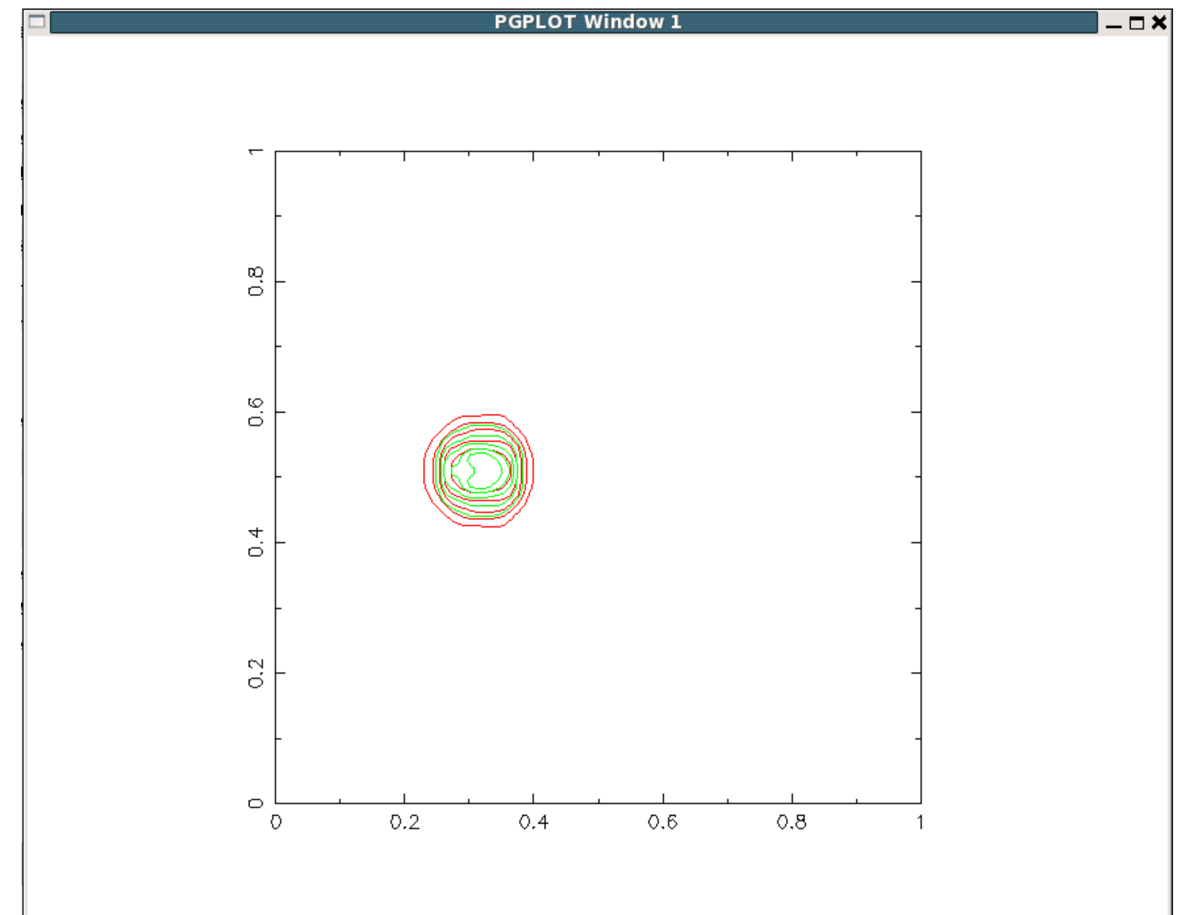


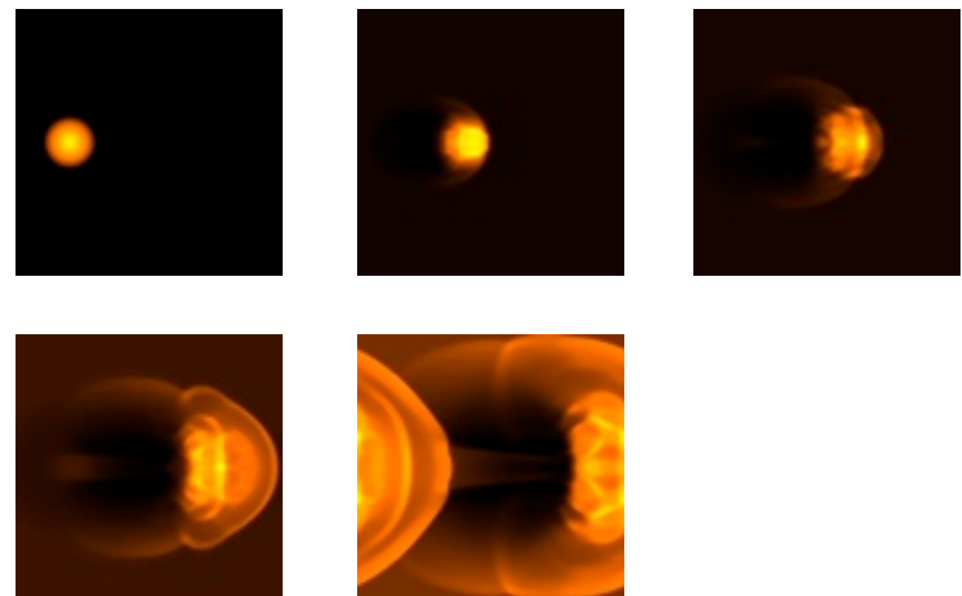
- So this makes extending the previous type to include all variable straightforward.
- One call, no memory copies, but a little thinking.

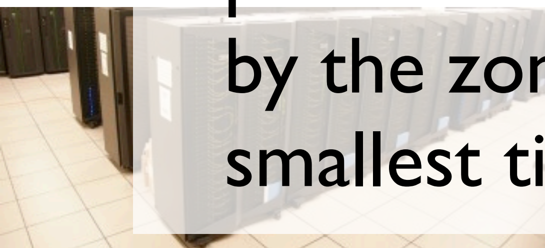
SciNet
High Performance Computing

MPI Hydro code

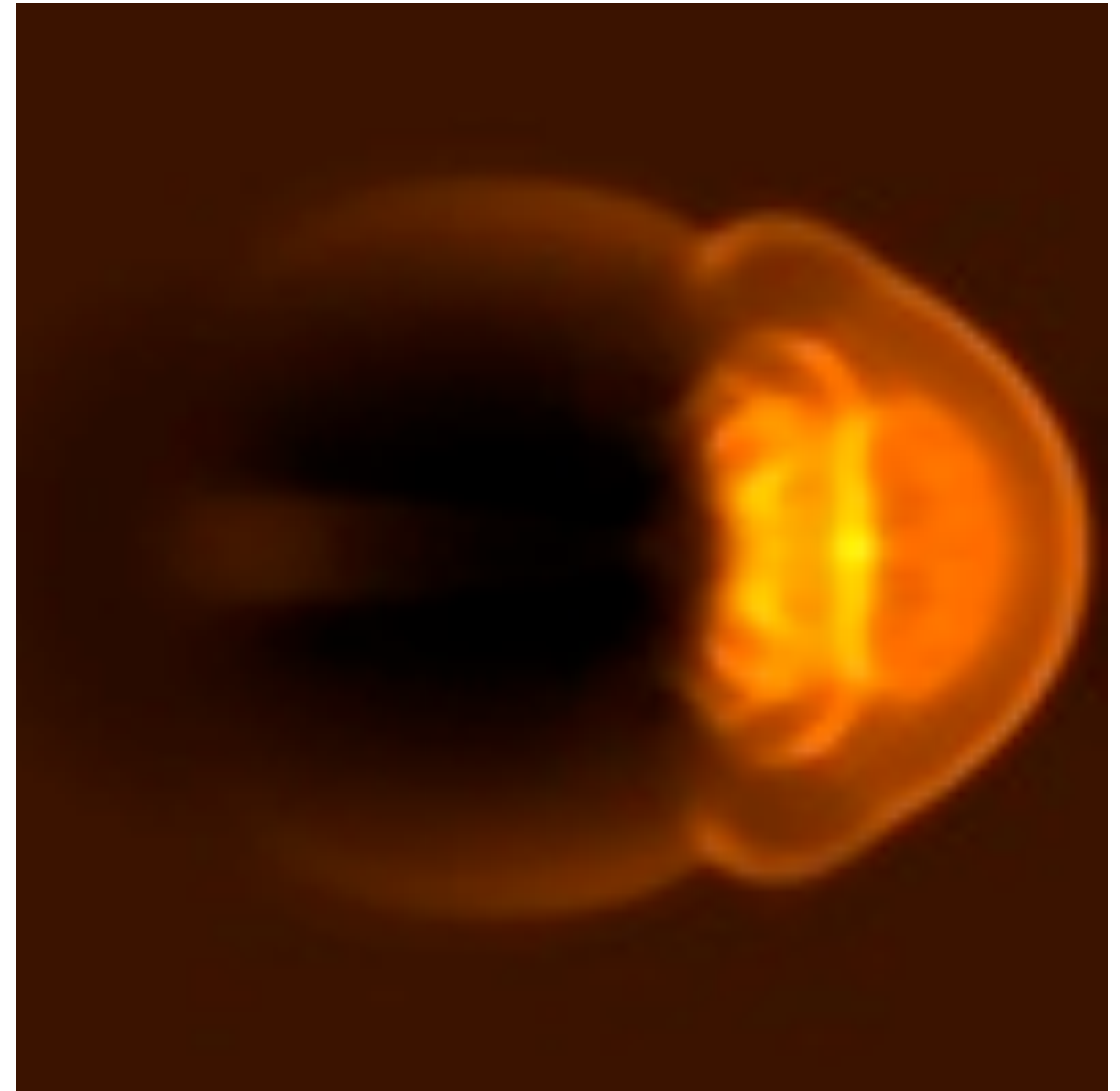
- cd ~pca/src/hydro/completedexecutables
- mpirun -np 3 varying_dt_mpi --nxproc=3
- Takes options:
  - --nxproc=N
  - --nyproc=N
- Better timestepping; takes largest possible stable timestep - limited by the zone that allows the smallest timestep.







MPI-IO



- When you run the parallel code, you get one file that has the sum of the domain
- At no time does one node have the whole domain
- How do we do this?

```
static MPI_Datatype blocktype;

ierr = MPI_Type_vector(ny, 3*nx, 3*nxproc*nx, MPI_UNSIGNED_CHAR, &blocktype);

MPI_Barrier(MPI_COMM_WORLD);

offset = headersize + (nyproc-1-gridj)*(3*rowsize*ny) + gridi*(3*nx);
ierr = MPI_File_open(MPI_COMM_WORLD,filename, MPI_MODE_WRONLY | MPI_MODE_APPEND , MPI_INFO_NULL, &file);
ierr = MPI_File_set_view(file, offset, blocktype, blocktype, "native", MPI_INFO_NULL);
ierr = MPI_File_write(file, data, 3*nx*ny, MPI_UNSIGNED_CHAR, &status);
ierr = MPI_File_close(&file);
```

Homework

- OMP the hydro code
- Fill in the MPI pieces for varying_dt_mpi (search for HW in boundary_conds_mpi.c, varying_dt_mpi.c)
- Do one of the GC-filling - copy into a buffer or use MPI_Vector type (2 separate routines: *mpi_bc* or *mpi_bc_vector*). Make sure *apply_bc* calls the right one (line 278, boundary_conds_mpi.c)
- Make an MPI call to find the minimum of all allowed timesteps in varying_dt_mpi.c

Timings

- use --nooutput --nsteps=50 as a standard test problem for timing.
- Time OpenMP on one and two threads on desktop
- Time MPI version on 1,2,4,8 cores on single node, and up to 32 cores using same qsub script from yesterday. How many zones per core are there at 32? What happens if you increase the problem size? (Don't forget to reduce the timestep if you haven't implemented varying timestep.)