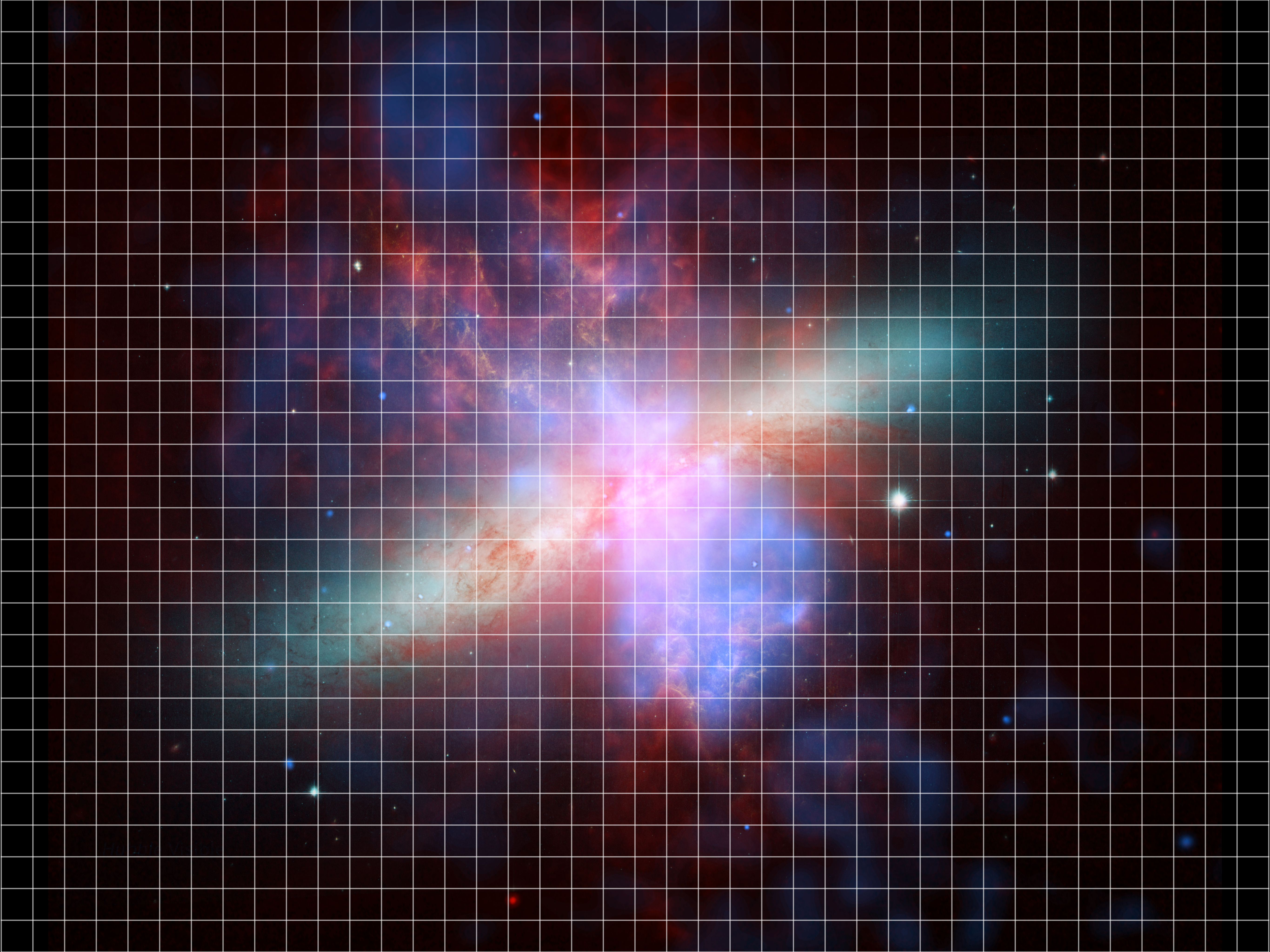
Cholla: A GPU-Native Hydrodynamics Code for Leadership Computing

Evan Schneider (Princeton)
Brant Robertson (UCSC)

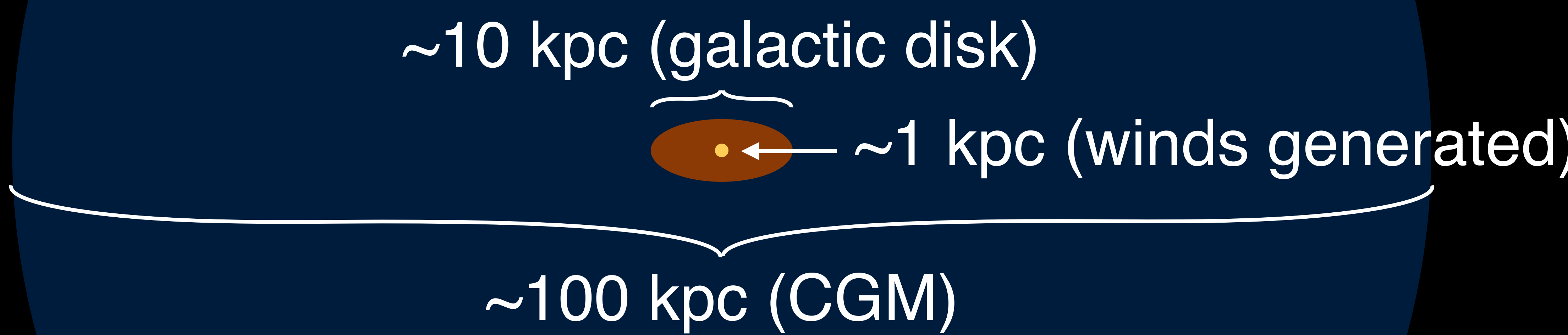
OLCF Users Group Meeting, May 15, 2018
Project ID AST 125

Oak Ridge National Laboratory | Leadership Computing Facility

Why did we need a new hydrodynamics code?

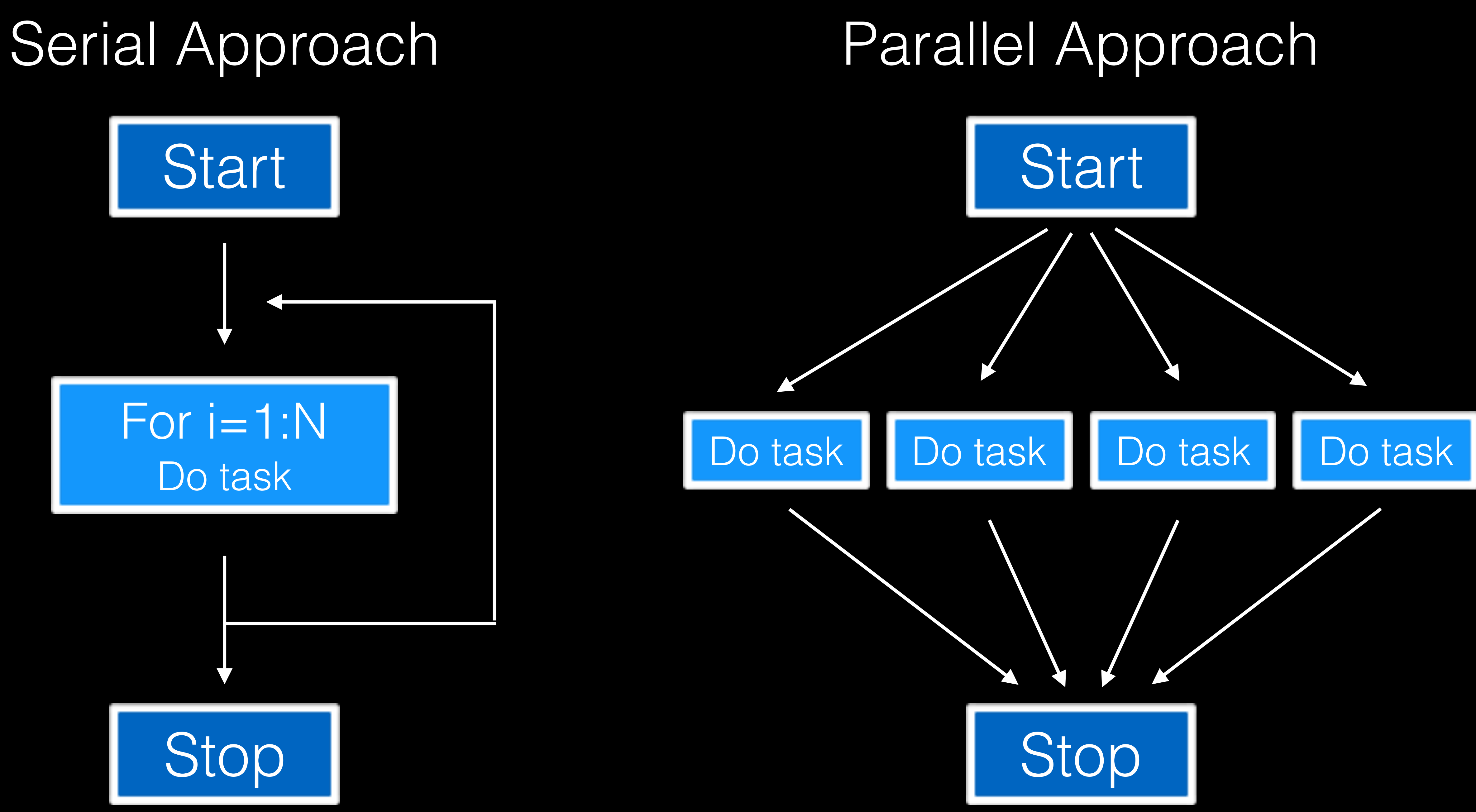

Hubble Visible

Simulating Galactic Winds Is Computationally Challenging

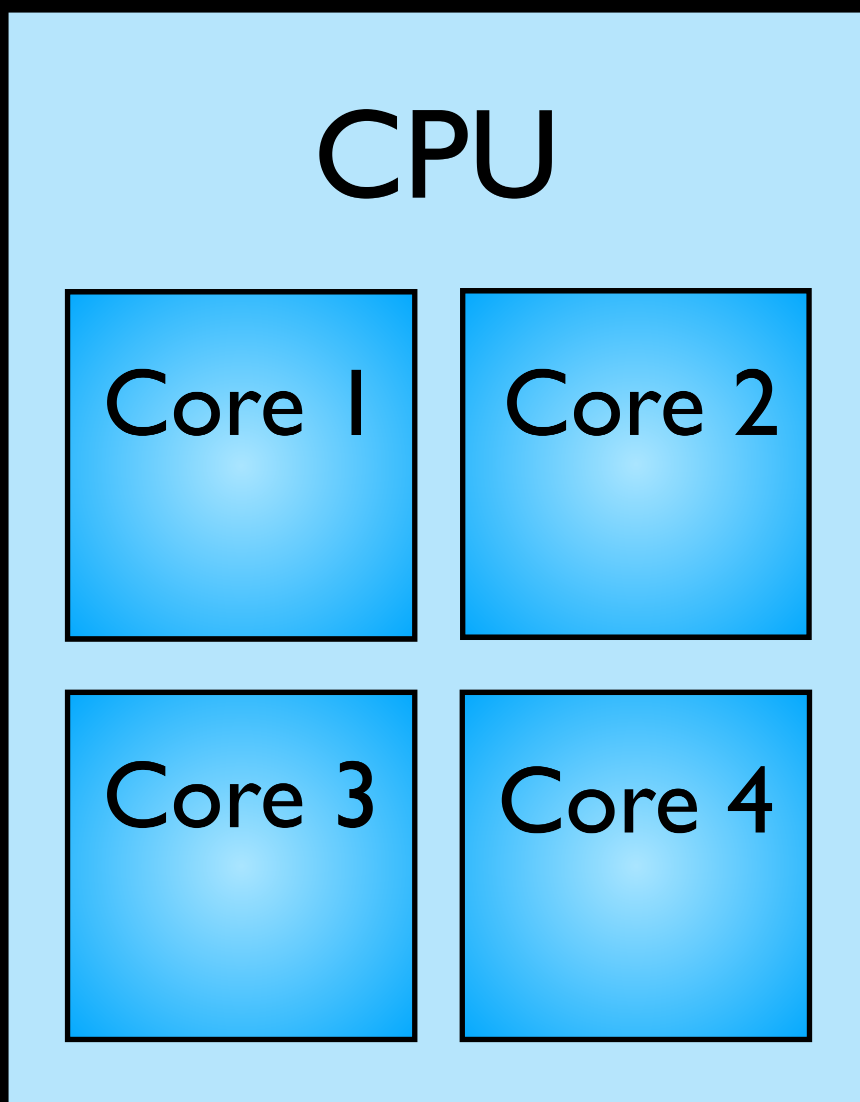


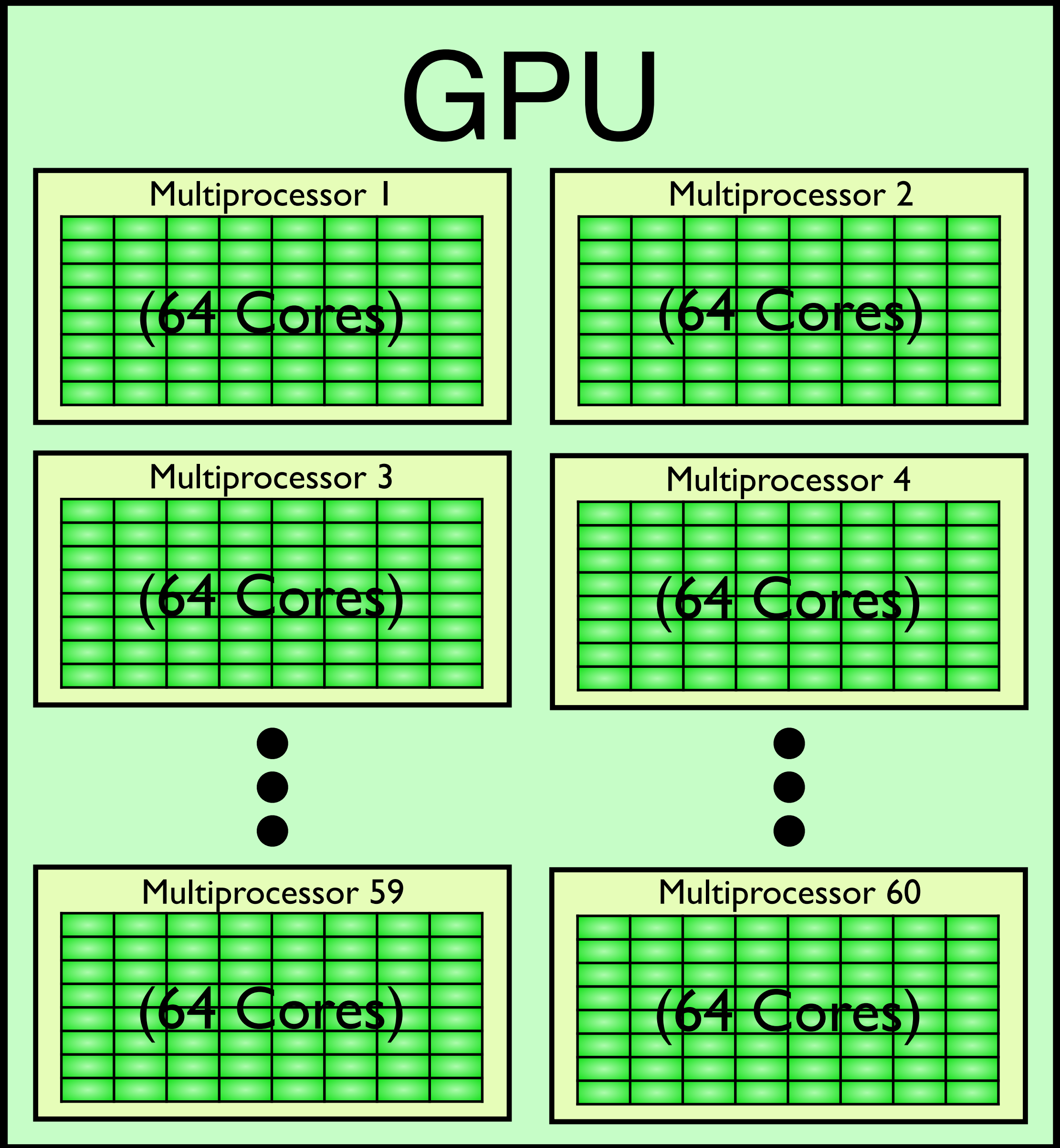
The scales involved in galactic wind evolution range from ~1-10 pc (cooling radius of supernova bubbles) to ~100 kpc (virial radius of halo).

Computer Architectures Have Changed



Computer Architectures Have Changed





Optimized for Serial Tasks

Optimized for Parallel Tasks

So, the goal was to build a *new* code, that could:

- achieve high resolution throughout the simulation volume (run simulations with large numbers of cells)
- take full advantage of new computing architectures
- address the limitations of the previous generation of hydrodynamics codes.

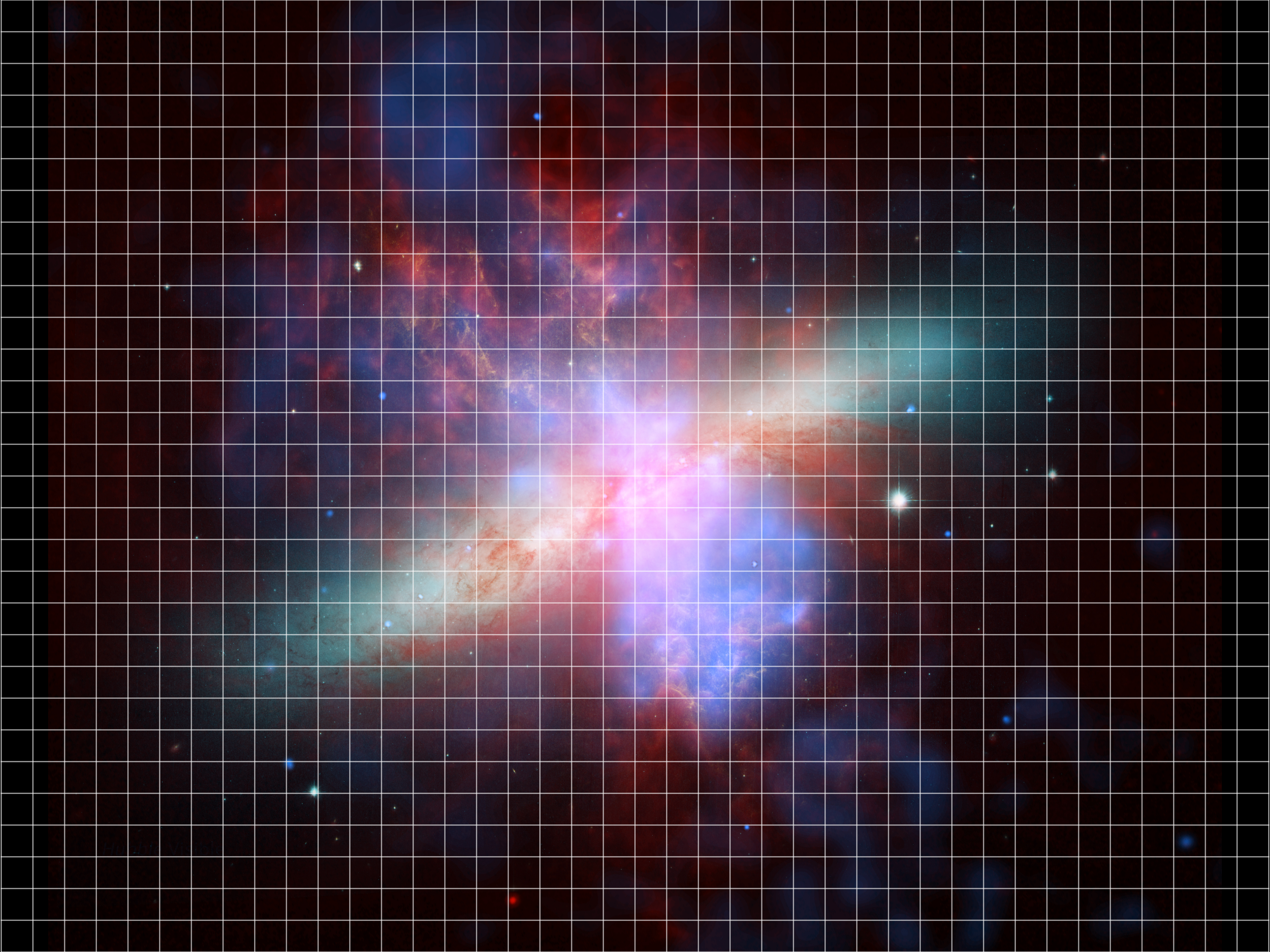
Cholla:

Computational hydrodynamics on ll architectures

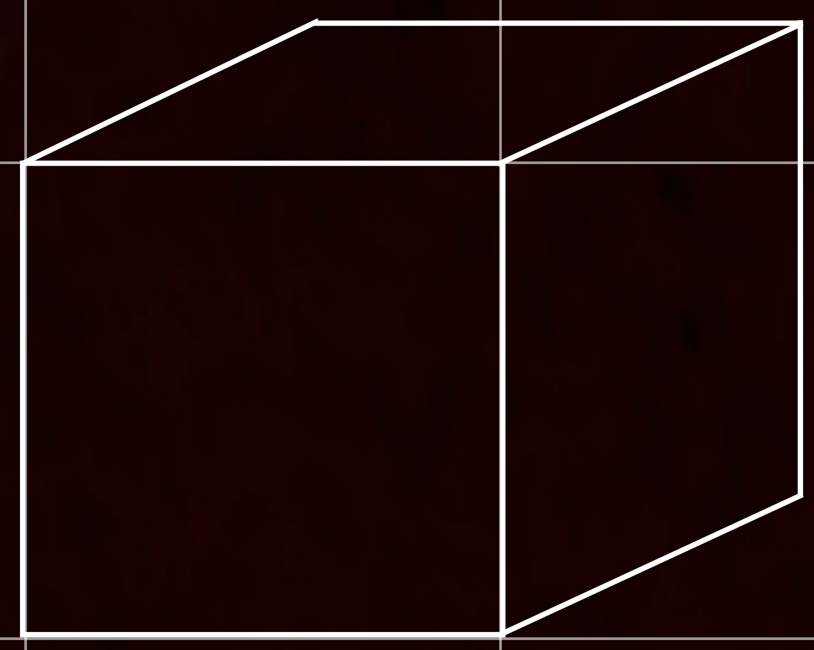
Cholla are also a group of cactus species that grows in the Sonoran Desert of southern Arizona.

- A GPU-native, massively-parallel, grid-based hydrodynamics code (publicly available at github.com/cholla-hydro/cholla)
- Incorporates state-of-the-art hydrodynamics algorithms (unsplit integrators, 3rd order spatial reconstruction, precise Riemann solvers, dual energy formulation, etc.)
- Also includes optically-thin cooling and photoionization heating based on precomputed rate tables, and static gravity.

Schneider & Robertson (2015, 2017)

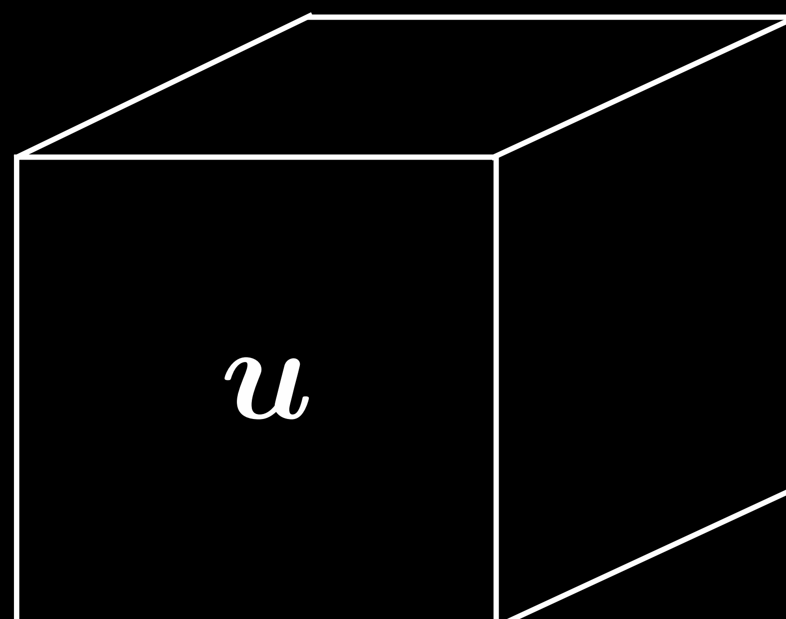

Hubble Visible

A (brief) introduction to finite-volume methods



A (brief) introduction to finite-volume methods

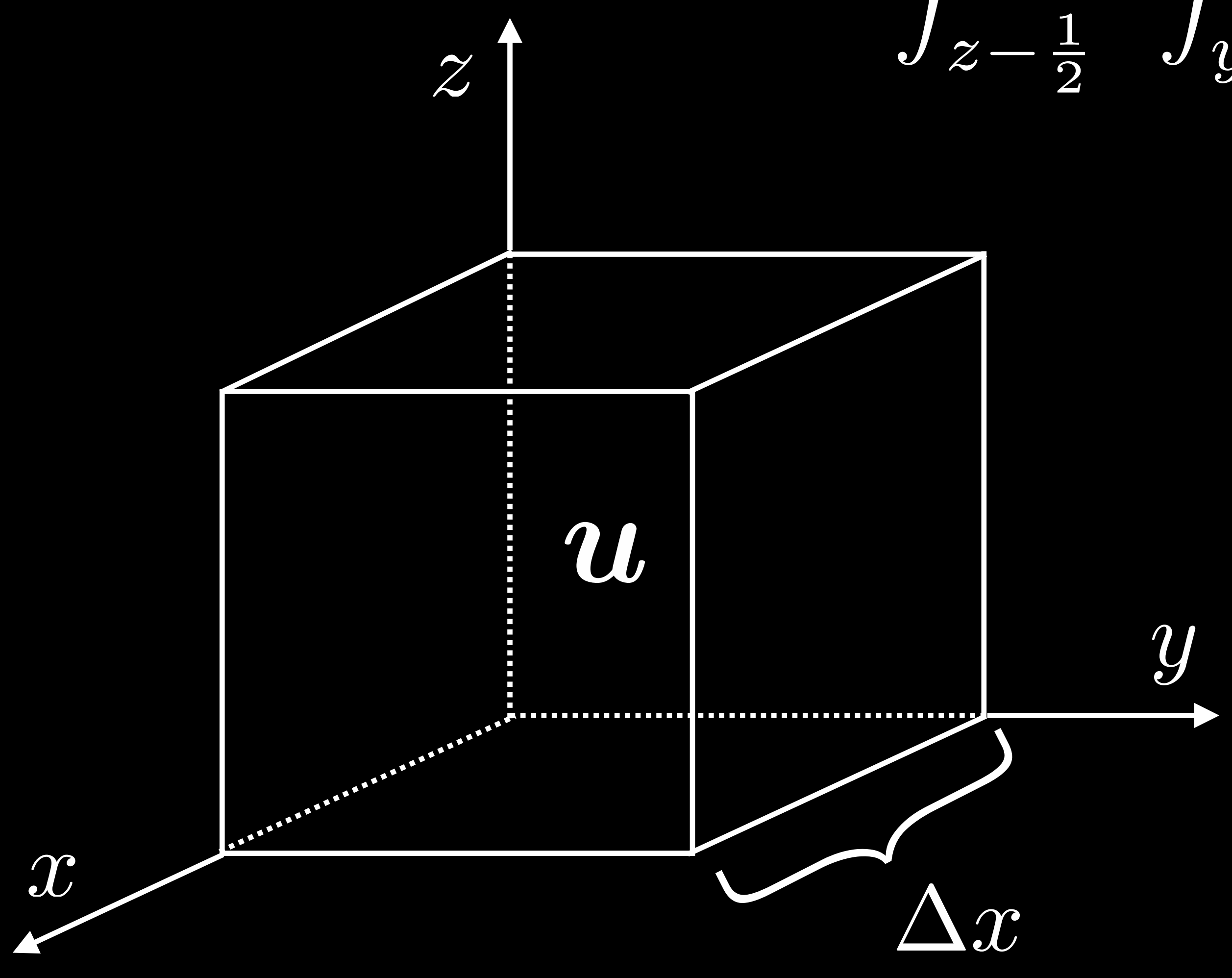
$\boldsymbol{u} = [\rho, \rho u, \rho v, \rho w, E]^{\mathrm{T}}$, a vector of conserved quantities



A (brief) introduction to finite-volume methods

$\boldsymbol{u} = [\rho, \rho u, \rho v, \rho w, E]^{\mathrm{T}}$, a vector of conserved quantities

$$\boldsymbol{u} = \int_{z-\frac{1}{2}}^{z+\frac{1}{2}} \int_{y-\frac{1}{2}}^{y+\frac{1}{2}} \int_{x-\frac{1}{2}}^{x+\frac{1}{2}} \frac{\boldsymbol{u}(x,y,z)}{\Delta x \Delta y \Delta z} \mathrm{d}x\mathrm{d}y\mathrm{d}z$$

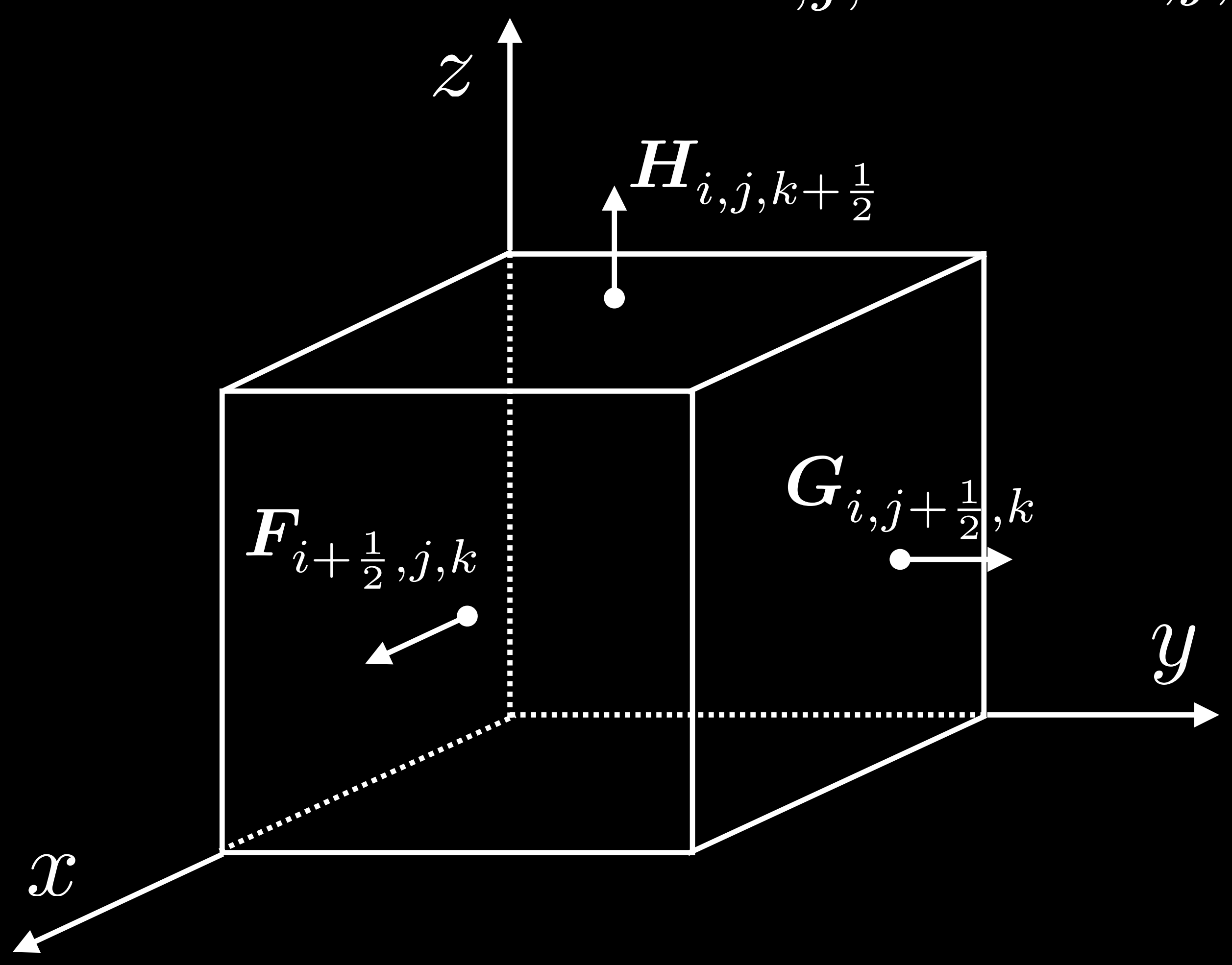


We want to go from $\boldsymbol{u}$ at time t, to $\boldsymbol{u}$ at time $t + \Delta t$.

A (brief) introduction to finite-volume methods

$\boldsymbol{u} = [\rho, \rho u, \rho v, \rho w, E]^{\mathrm{T}}$, a vector of conserved quantities

$$\boldsymbol{u}_{i,j,k}^{t+\Delta t} = \boldsymbol{u}_{i,j,k}^{t} + \frac{\Delta t}{\Delta x}\left(F_{i+\frac{1}{2},j,k} - F_{i-\frac{1}{2},j,k}\right) + \frac{\Delta t}{\Delta y}\left(G_{i,j+\frac{1}{2},k} - G_{i,j-\frac{1}{2},k}\right) + \frac{\Delta t}{\Delta z}\left(H_{i,j,k+\frac{1}{2}} - H_{i,j,k-\frac{1}{2}}\right)$$



Conserved Variable Update in C

```
// loop over each cell, updating density, momentum, and energy
for (i=0; i<nx; i++) {
  density[i]       += dt/dx * (F.d[i-1]  - F.d[i]);
  momentum_x[i] += dt/dx * (F.mx[i-1] - F.mx[i]);
  momentum_y[i] += dt/dx * (F.my[i-1] - F.my[i]);
  momentum_z[i] += dt/dx * (F.mz[i-1] - F.mz[i]);
  Energy[i]        += dt/dx * (F.E[i-1]  - F.E[i]);
}
```

Conserved Variable Update in Cuda

```
void Update_Conserved_Variables(double *dev_conserved, double *dev_F,
int nx, double dx, double dt)
{
  // get a global thread ID
  int id = threadIdx.x + blockIdx.x * blockDim.x;

  // update the conserved variable array
  if (id < nx) {
    dev_conserved[0*nx + id] += dt/dx * (dev_F[0*nx + id-1] - dev_F[0*nx + id]);
    dev_conserved[1*nx + id] += dt/dx * (dev_F[1*nx + id-1] - dev_F[1*nx + id]);
    dev_conserved[2*nx + id] += dt/dx * (dev_F[2*nx + id-1] - dev_F[2*nx + id]);
    dev_conserved[3*nx + id] += dt/dx * (dev_F[3*nx + id-1] - dev_F[3*nx + id]);
    dev_conserved[4*nx + id] += dt/dx * (dev_F[4*nx + id-1] - dev_F[4*nx + id]);
  }
}
```

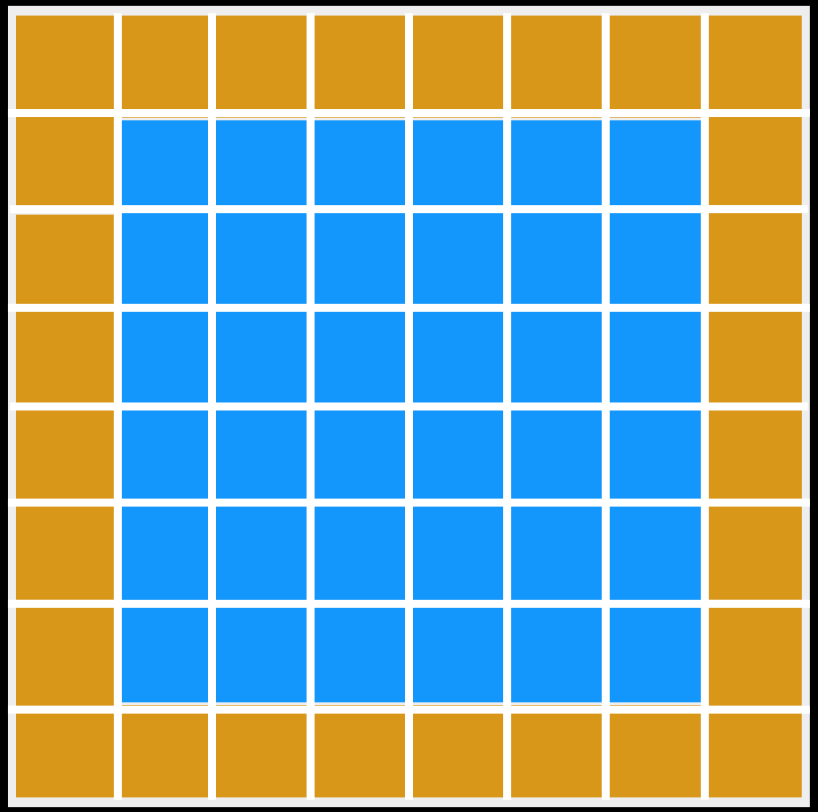
Conserved Variable Update in Cuda

```
// copy the conserved variable array onto the GPU
cudaMemcpy(dev_conserved, host_conserved, 5*n_cells*sizeof(double),
        cudaMemcpyHostToDevice);

// call cuda kernel
Update_Conserved_Variables<<<dimGrid,dimBlock>>>(dev_conserved,
dev_F, nx, dx, dt);

// copy the conserved variable array back to the CPU
cudaMemcpy(host_conserved, dev_conserved, 5*n_cells*sizeof(double),
        cudaMemcpyDeviceToHost);
```

What does Cholla do?

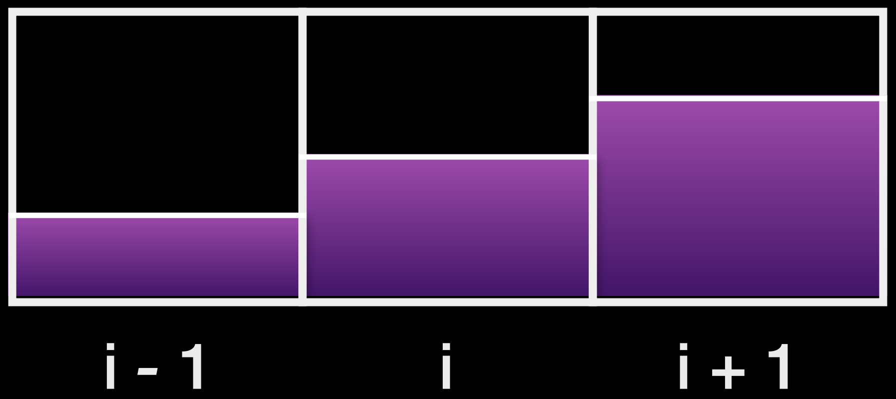
Models the equations of hydrodynamics on a static mesh in 1D, 2D, or 3D using either the 6-solve Corner Transport Upwind algorithm (Colella, 1990; Gardiner & Stone, 2008) or the Van Leer integration algorithm (Stone & Gardiner, 2009).

Apply initial conditions and boundary conditions to the grid.

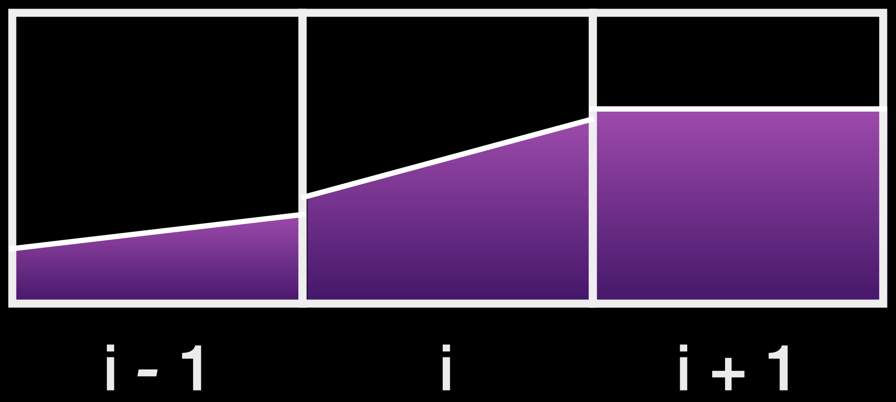


What does Cholla do?

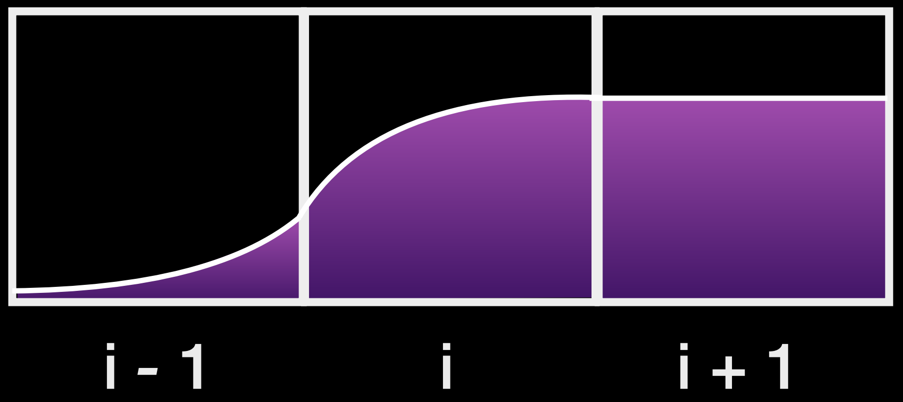
Reconstruct interface values using cell averages.



Choose either piecewise constant, piecewise linear, or piecewise parabolic reconstruction.

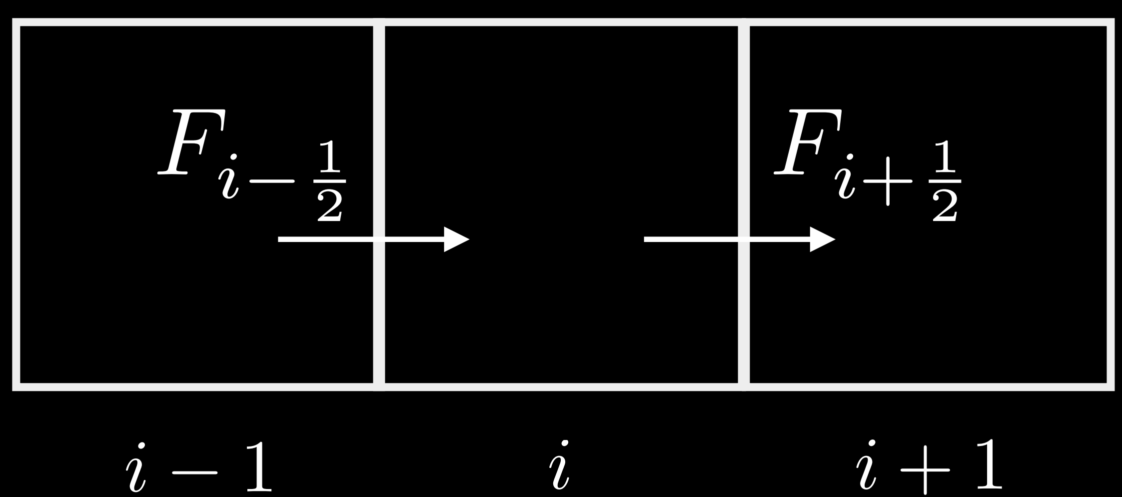


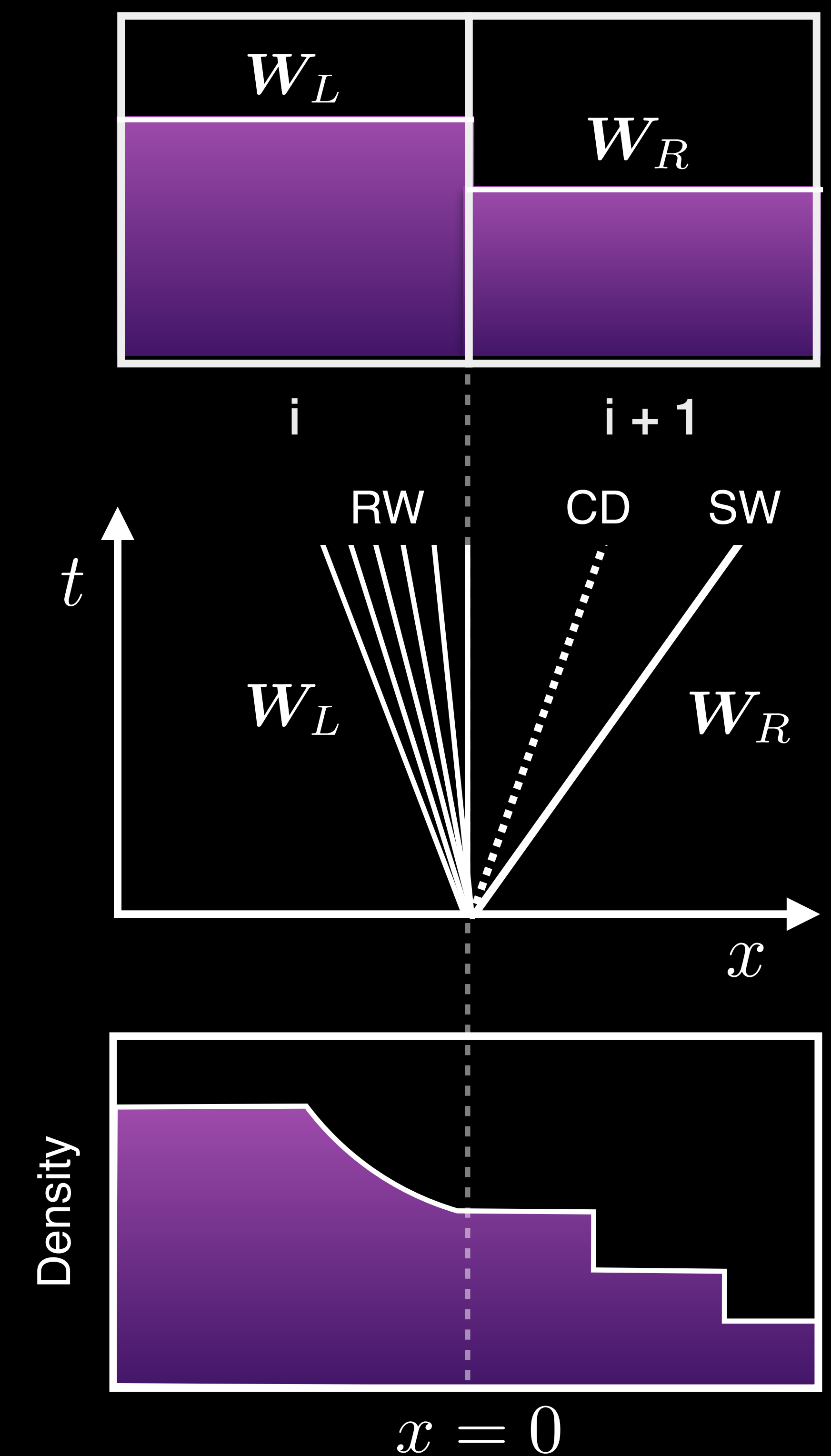
Piecewise linear and piecewise parabolic reconstruction can be done in either the primitive variables or the characteristic variables.



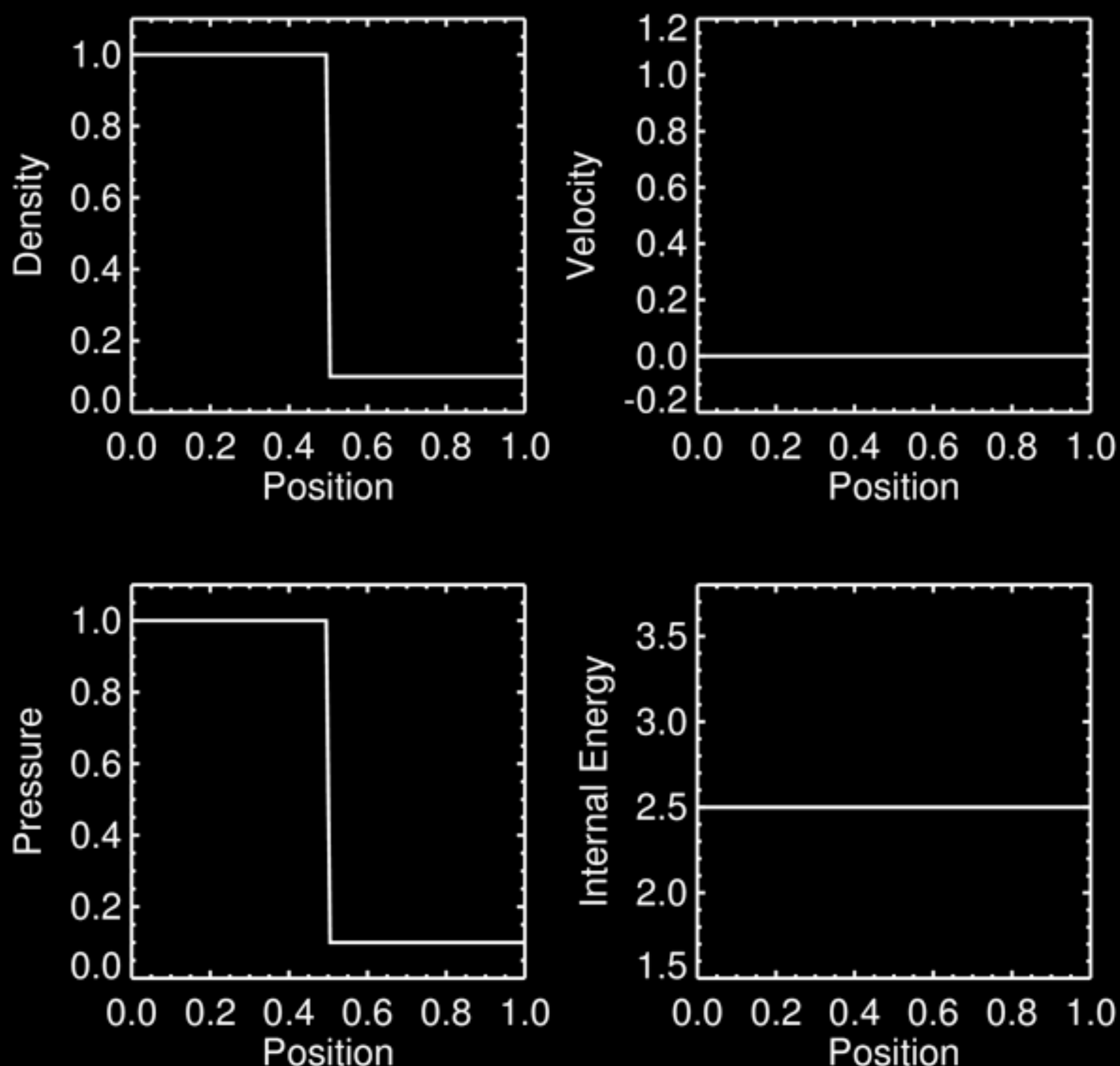
What does Cholla do?

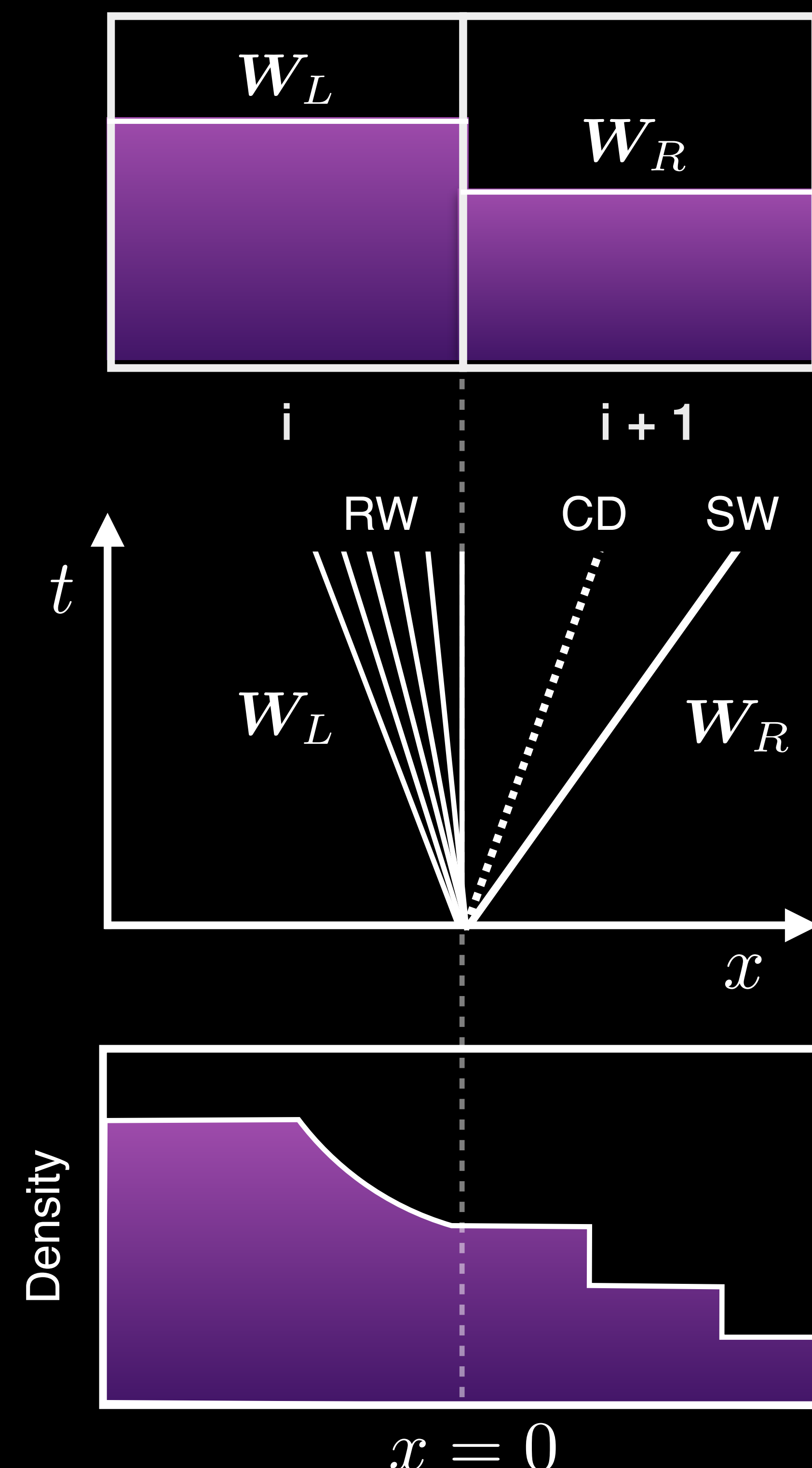
Calculate fluxes across cell interfaces using reconstructed interface values.





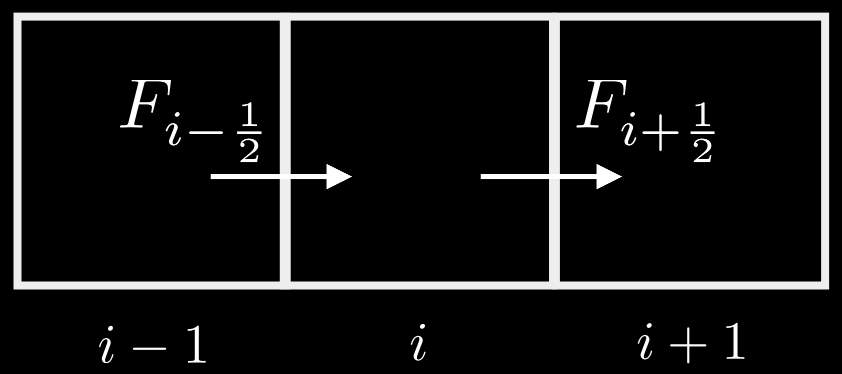
What does Cholla do?



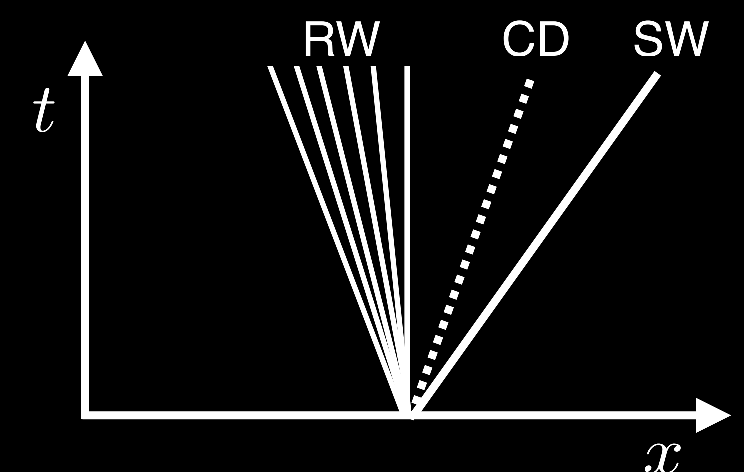


What’s the GPU advantage?

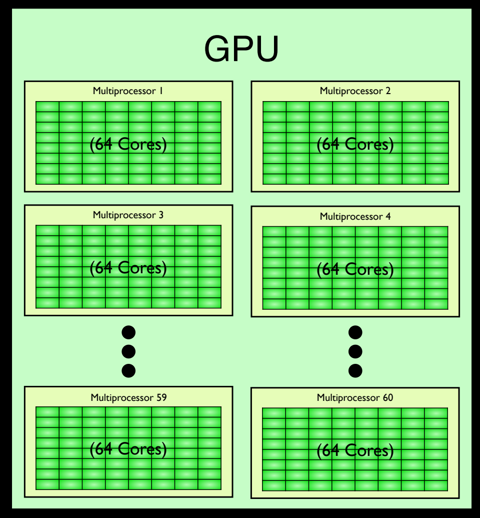
- Grid-based hydro codes are eminently parallelizable - each cell needs data from only a few nearby cells to reconstruct interface values, calculate fluxes across interfaces, and update conserved quantities.



- Hydro solvers are computationally expensive. Many unsplit algorithms require 6 Riemann problems per cell, per timestep (in 3D).

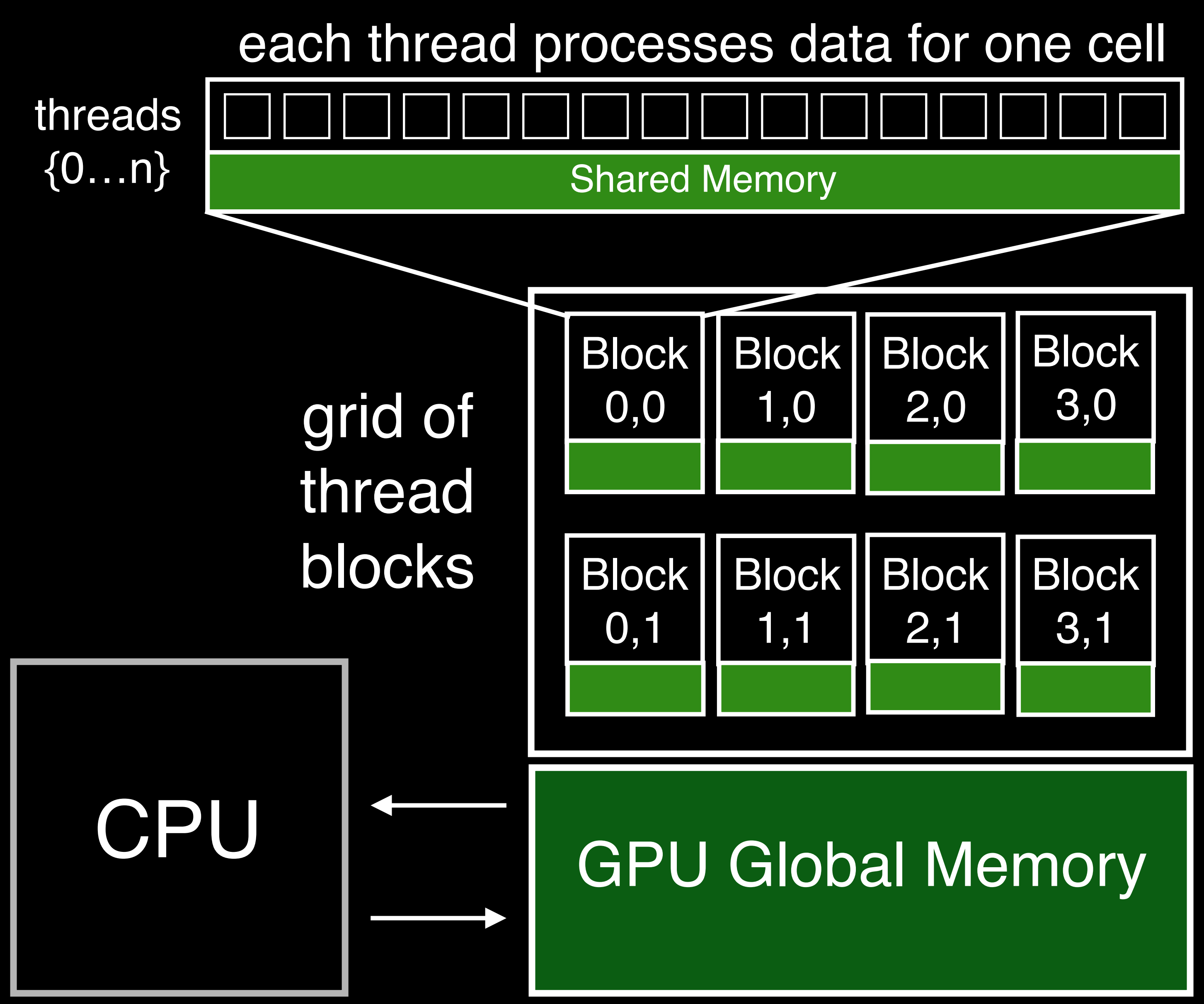


- With GPUs, we massively parallelize the calculation across many cores, allowing us to speed up computation by an order of magnitude as compared to similarly intensive CPU codes.



How does it work?

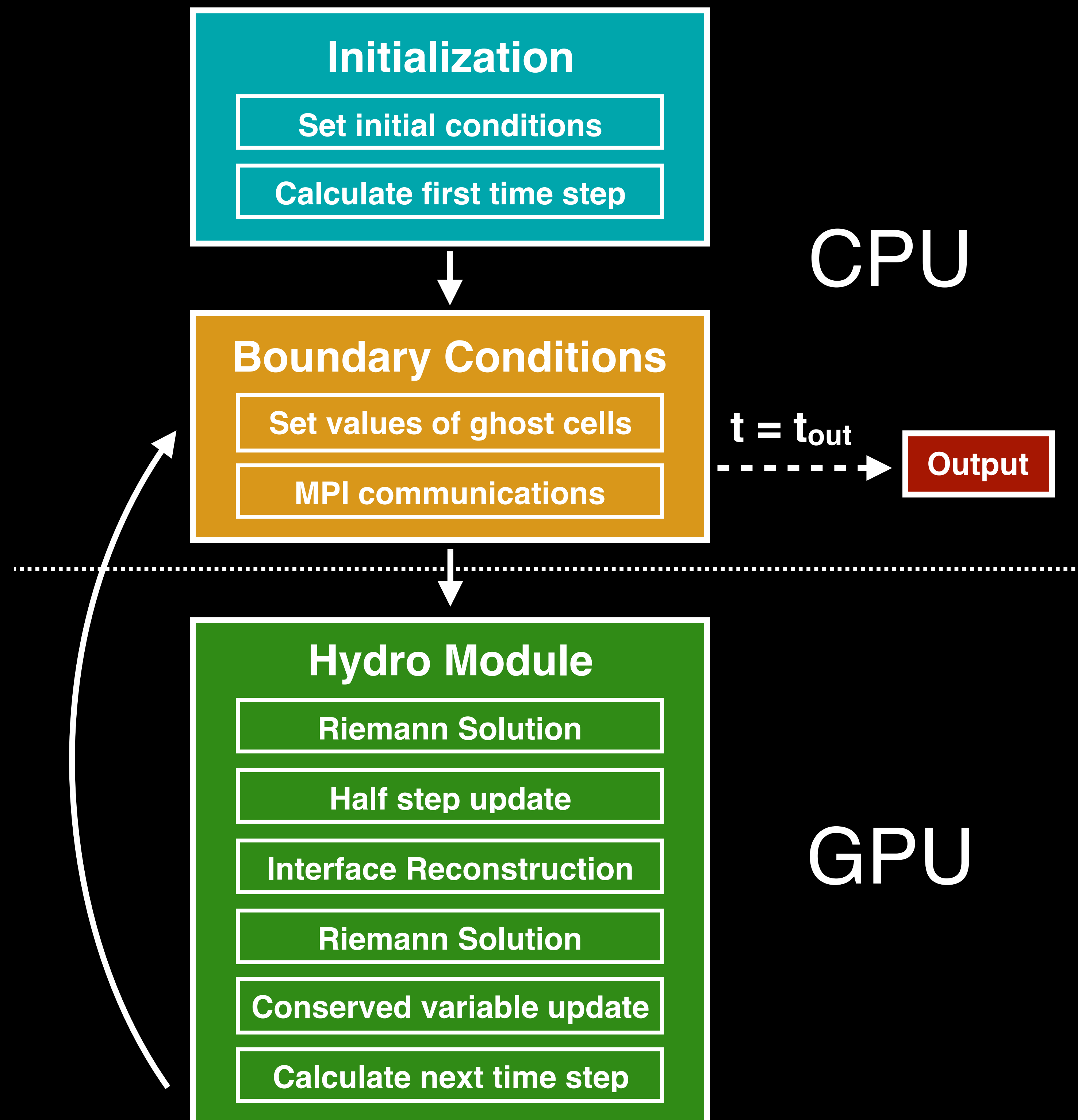
- GPU functions execute as CUDA kernels on a grid of thread blocks - each cell in the simulation is mapped to a single thread.
- Cholla is designed to minimize memory transfers between the CPU and GPU, reducing computational overhead.



How does it work?

Serial parts of code execute on the CPU

Parallel portions execute on the GPU

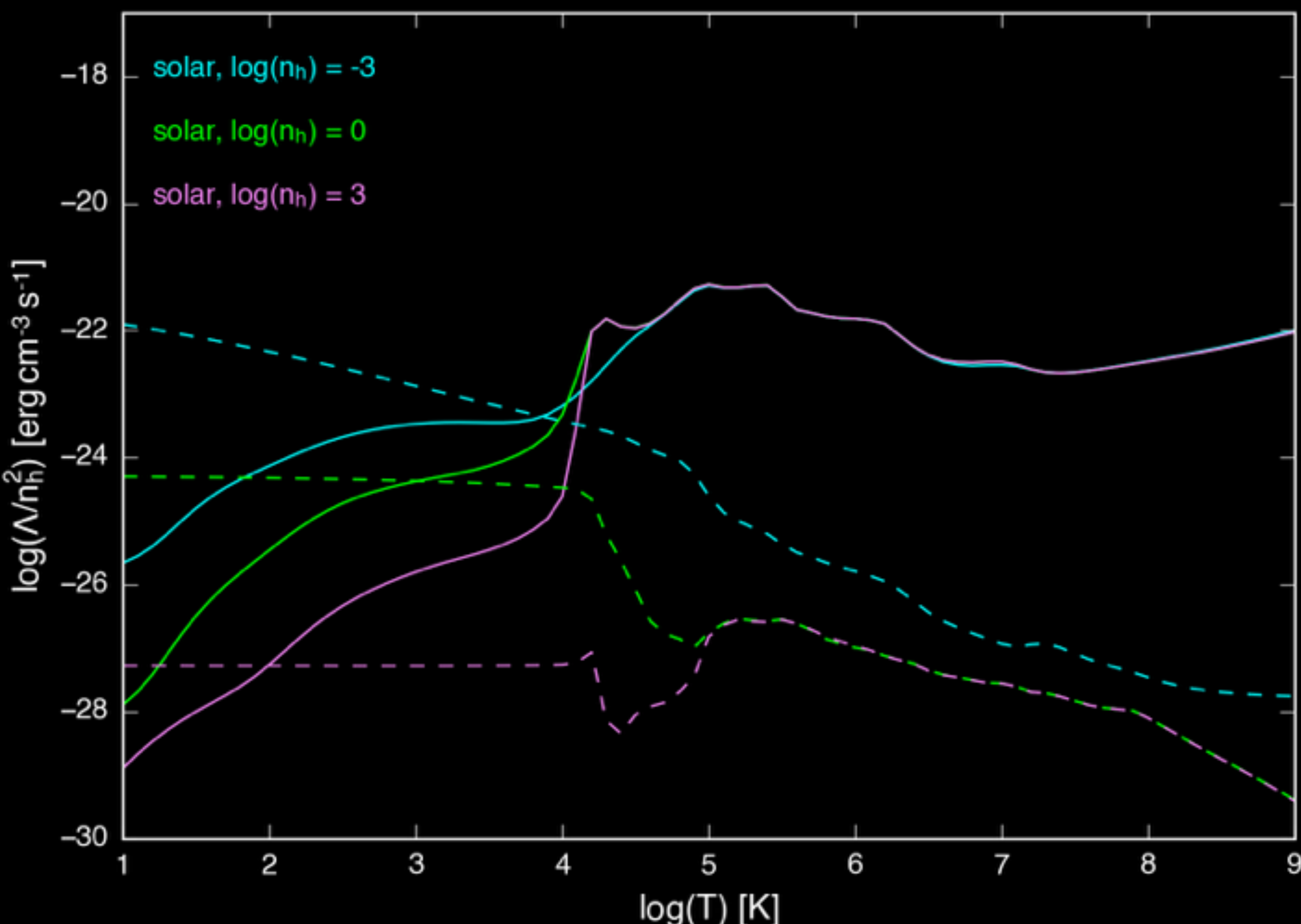


An aside: using texture mapping to accelerate cooling calculations.

What is texture mapping?

Skyrim, Bethesda Game Studios

Many astrophysical cooling calculations rely on multidimensional table lookups to calculate radiative cooling / heating rates.



Texture mapping speeds up the cooling implementation in Cholla

1. Copy 2D cooling tables to texture memory on the GPU
2. Calculate density and temperature of gas
3. “Fetch” cooling and heating rates from the texture - bilinear interpolation comes for free!

```
__device__ double Cloudy_cool(double n, double T)
{
  double lambda = 0.0; // cooling rate, erg s^-1 cm^3
  double H = 0.0;      // heating rate, erg s^-1 cm^3
  double cool = 0.0;   // cooling per unit volume, erg / s / cm^3

  // fetch cooling and heating rates
  lambda = tex2D<float>(coolTexObj, T, n);
  H      = tex2D<float>(heatTexObj, T, n);

  // cooling rate per unit volume
  cool = n*n*(lambda - H);

  return cool;
}
```

How does it compare to the CPU version?

Loops/ Threads	GSL (CPU)	CUDA (GPU)
10^2	0.006 ms	0.023 ms
10^4	0.31 ms	0.023 ms
10^6	30 ms	0.023 ms

Cholla Test Suite

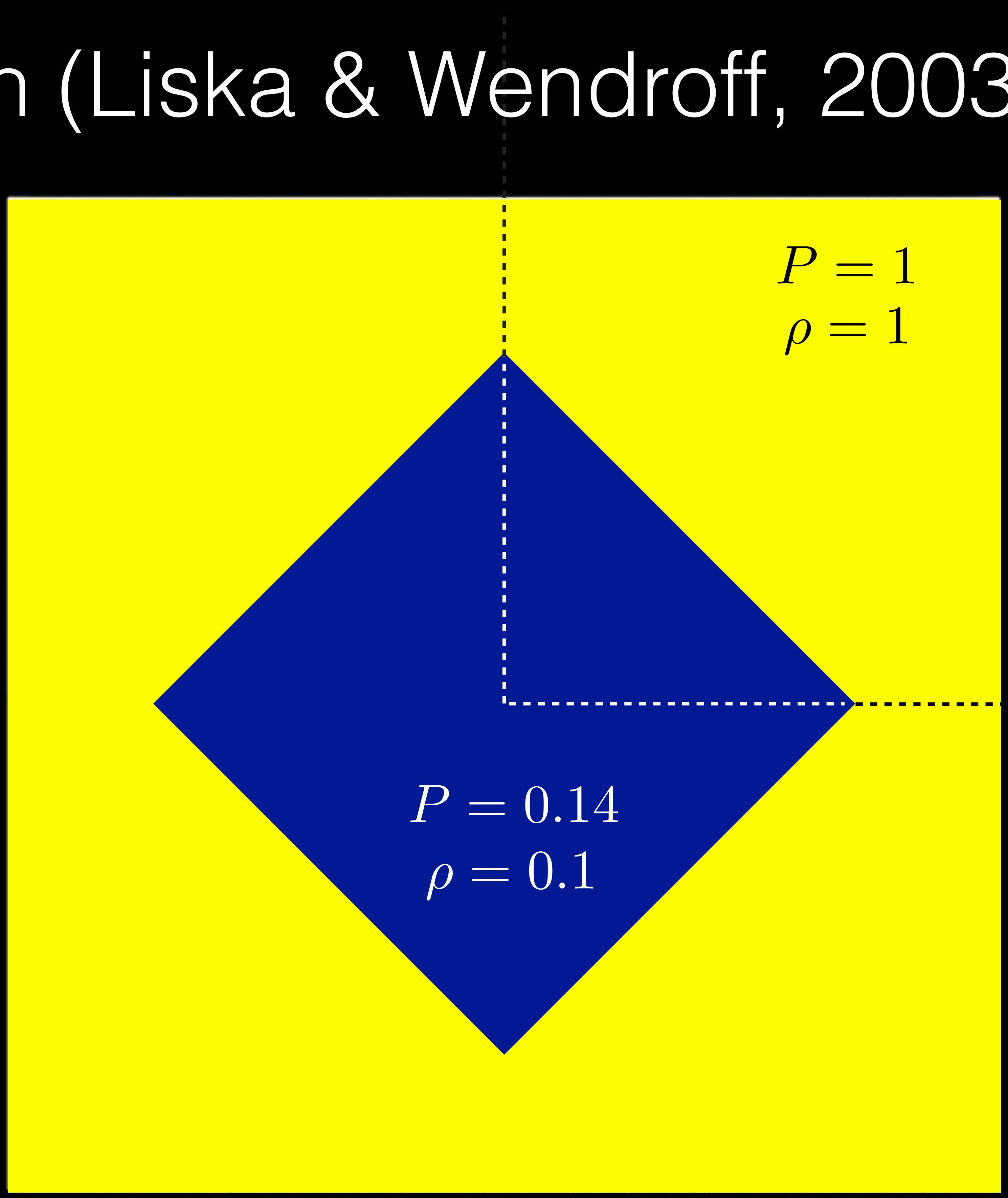
- Suite of 1, 2, & 3D hydro tests
- 1D: advection problem, Sod shock tube, strong shock problem, Shu & Osher shock tube, strong rarefaction problem, interacting blast waves, etc.
- 2D: advection problem, Sod shock problem (diagonal), implosion test, Kelvin Helmholtz instability, Rayleigh-Taylor instability, Noh's strong shock
- 3D: advection problem, Sod shock problem, Sedov-Taylor blast wave, Noh's strong shock

2D Implosion (Liska & Wendroff, 2003)

Example test calculation: Implosion test (1024^2)

55,804,166,144 cell updates

symmetric about y=x to roundoff error



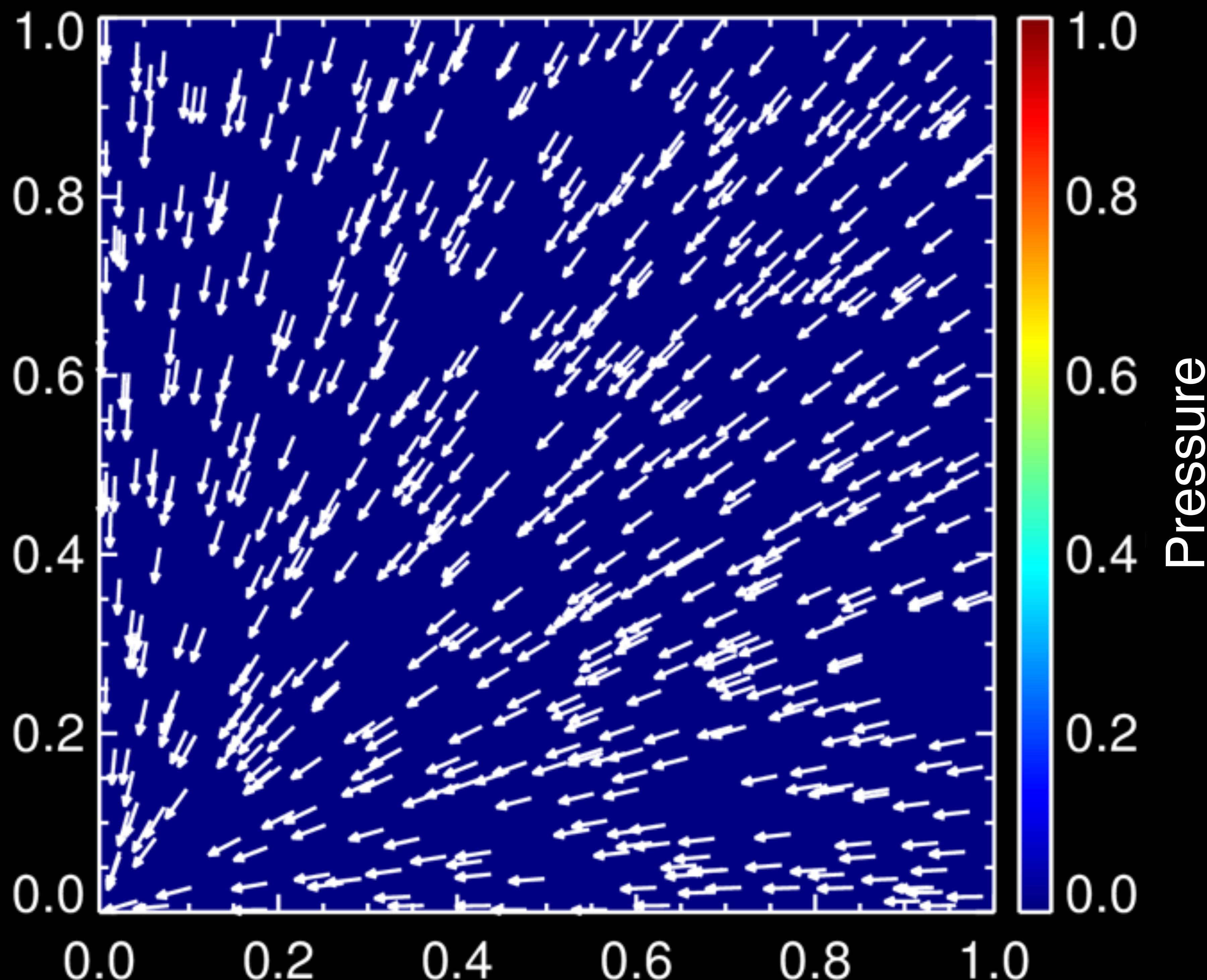
40 million cell updates/second on a single NVIDIA P100 GPU

3D Noh Strong Shock

1D version, Noh (1987)

d=1, P=0, |V| = -1, reflecting inner boundaries

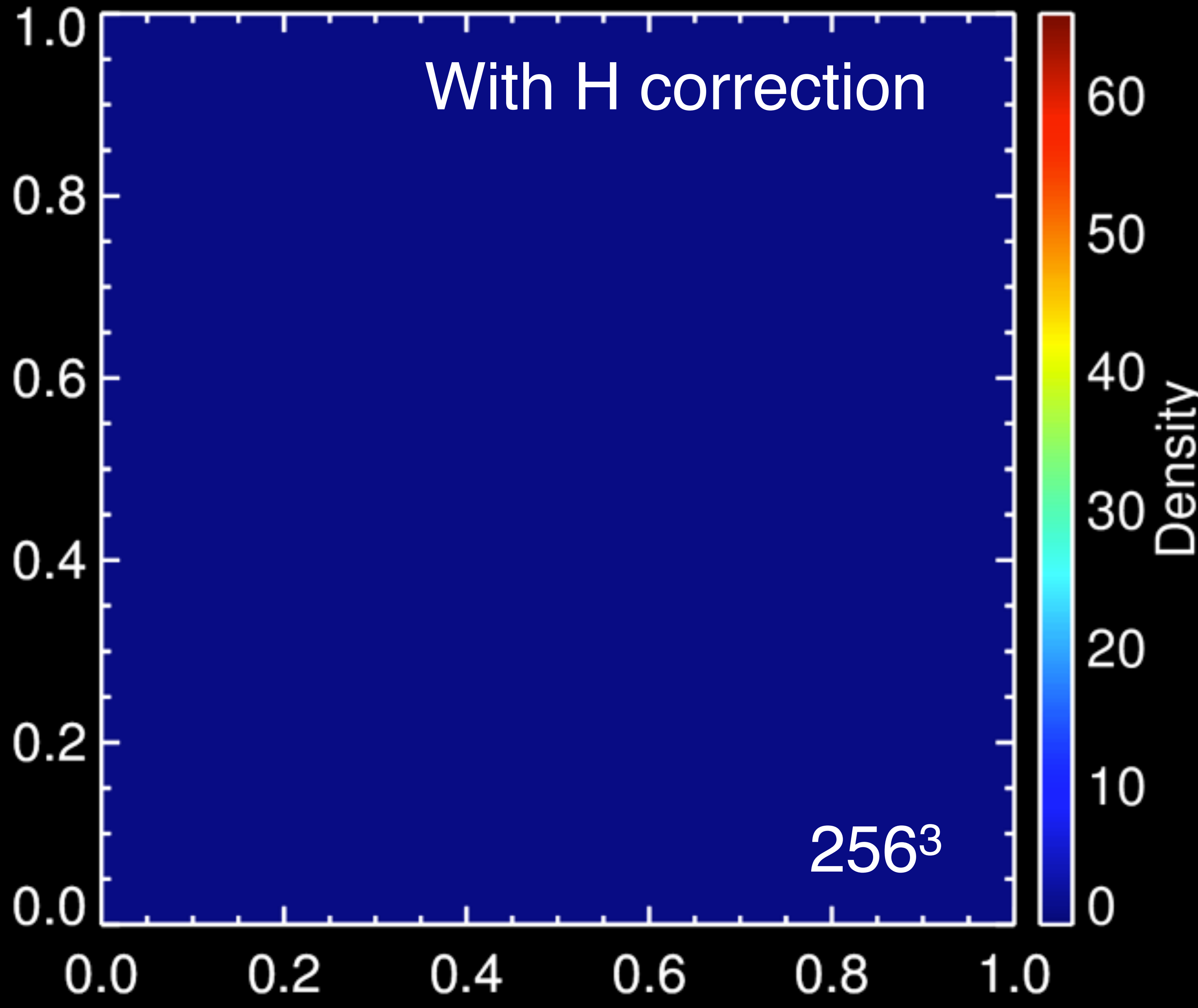
Formally infinite shock reflecting from origin.



3D Noh Strong Shock

Strong, grid-aligned shocks lead to Carbuncle instability.

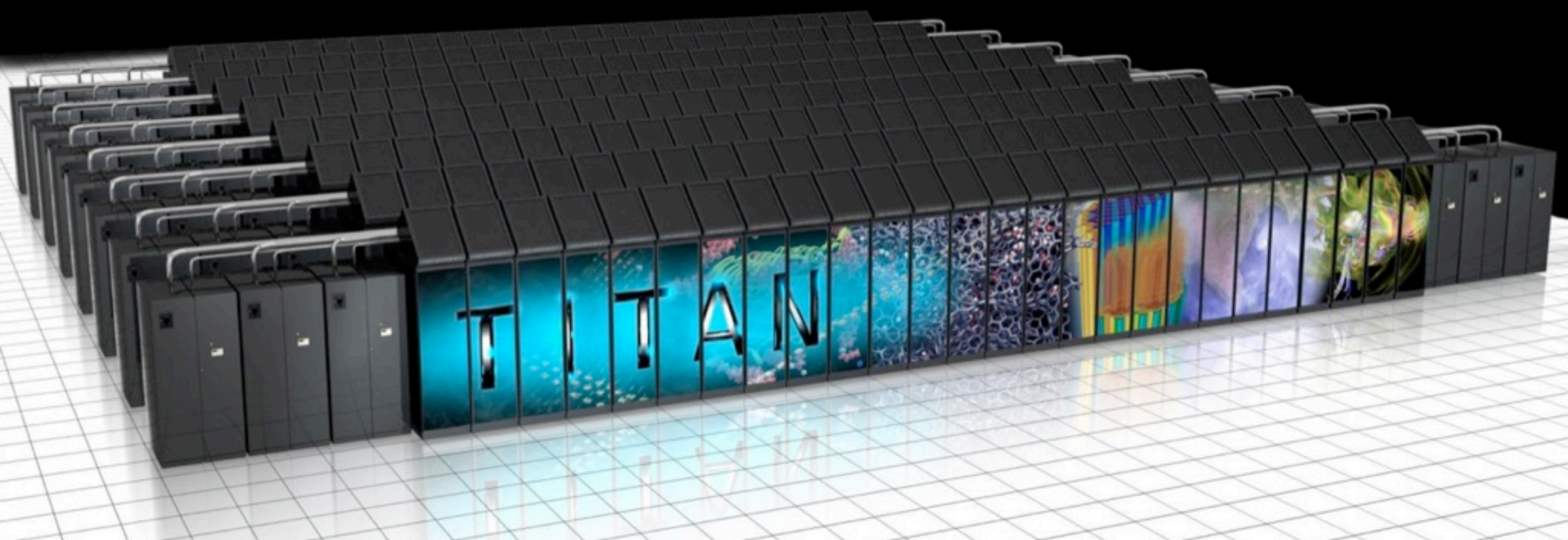
The H correction (Sanders,1998) uses information about transverse wave speeds to fix the problem.



Cholla takes advantage of the world's most powerful supercomputers.

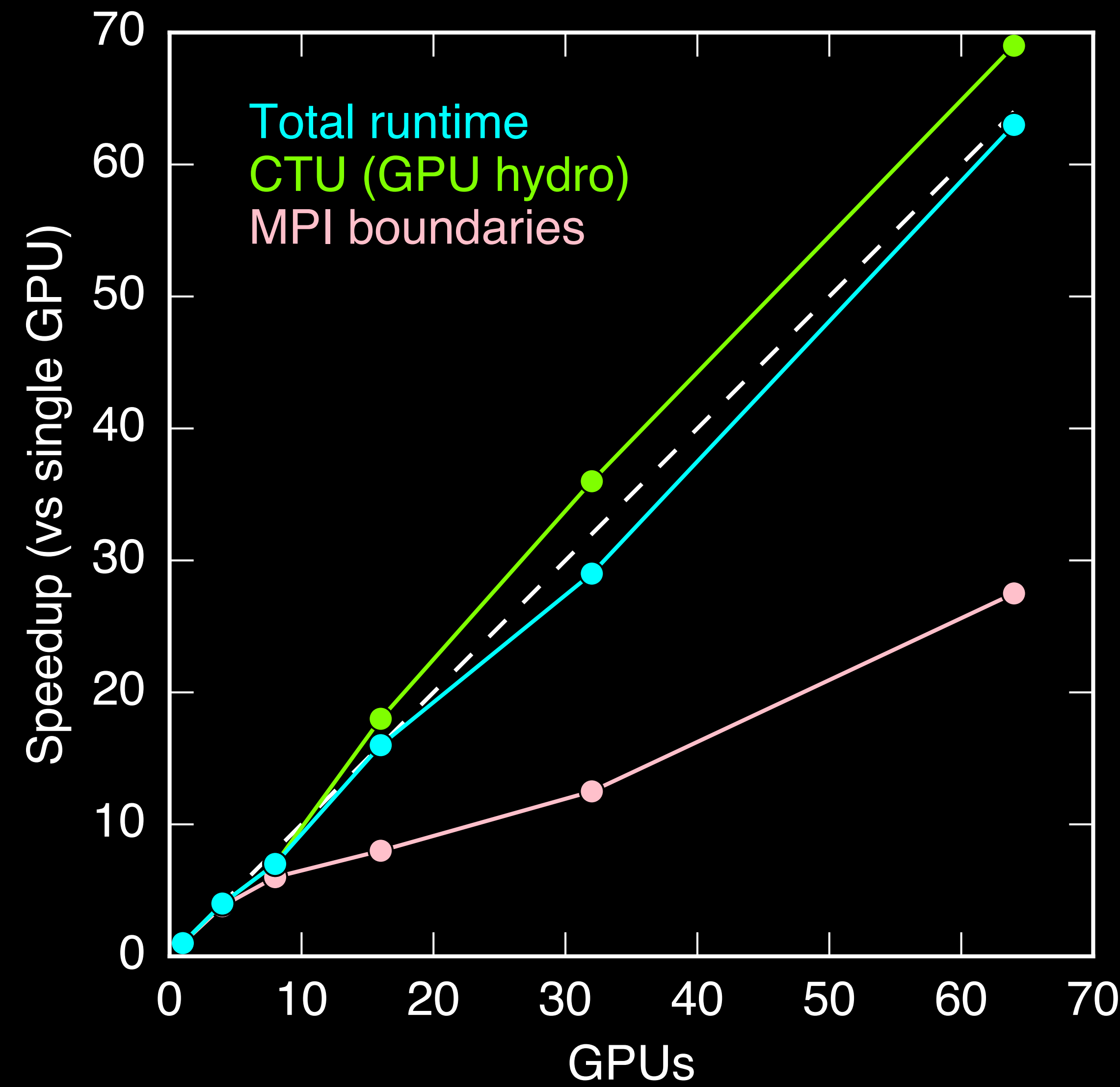
Titan: Largest Open Science Supercomputer in the US

Flagship accelerated computing system | 200-cabinet Cray XK7 supercomputer |
18,688 nodes (AMD 16-core Opteron + NVIDIA Tesla K20 GPU) |
CPUs/GPUs working together – GPU accelerates | 20+ Petaflops

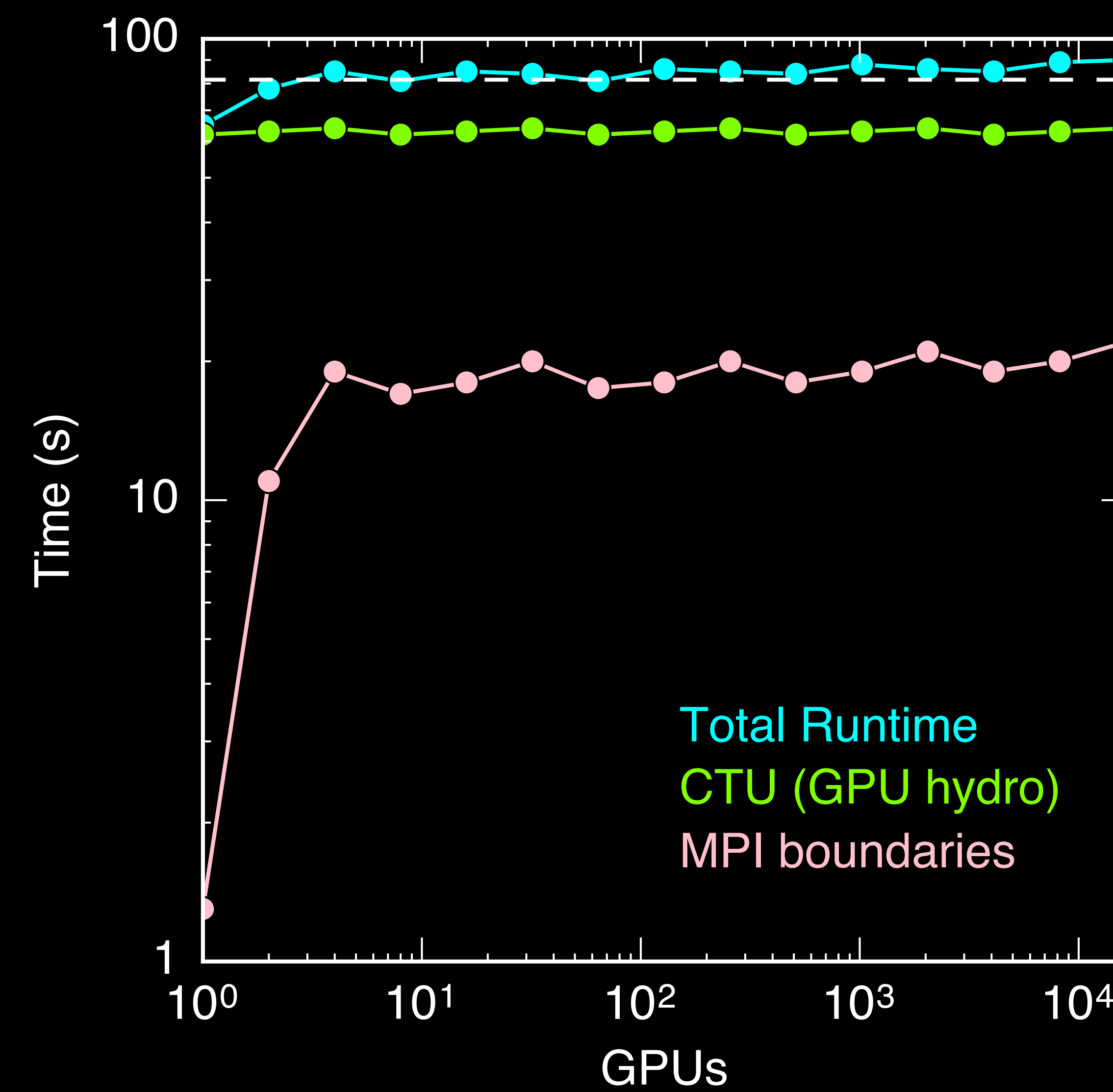


Cholla Achieves Excellent Scaling

Strong Scaling test, 512^3 cells



Weak Scaling test, ~322^3 cells / GPU



Schneider & Robertson (2015)

Graphics Processors as a Scientific Tool

Advantages

- Optimized for fast execution of parallel tasks
- Blocked architecture easily transitions to new hardware models
- Specialized hardware functions are FAST
- Offloading computation to GPU leaves CPU free to perform other tasks
- Energy efficient!

Challenges

- Limited memory on GPU
- Need lots of computation to make up for data transfer and memory latency
- Blocked architecture not optimal for some problems

2D Kelvin Helmholtz test

3840 x 2160 Resolution

Thanks!