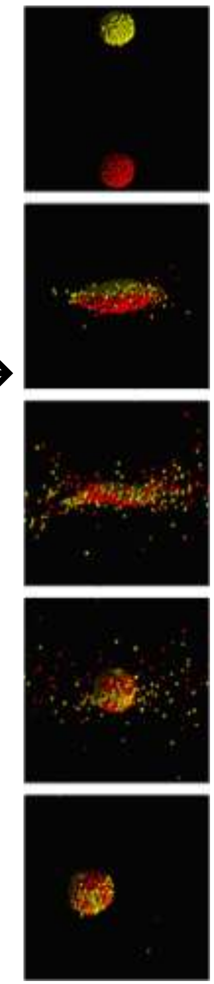
ChaNGa/Gasoline Tutorial

James Wadsley
(McMaster University)
Tom Quinn
(University of Washington)

Part I: Parallel Code, ChaNGa building, Simple test, Tipsy

Code History

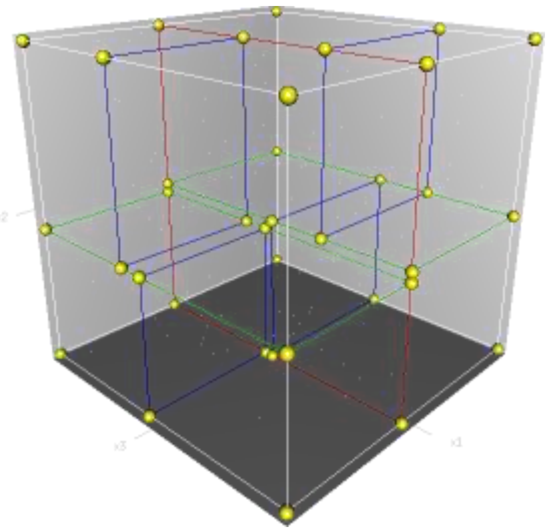
- Pkdgrav (~ 1990s, Stadel 2001)
  - C, KD-tree for DD, Binary Trees, Gravity only, Dark Matter Cosmology sims
  - Parallel via MDL library (MPI, threads & many more)
  - *Pkdgrav-Collisions SS* (Richardson, Quinn & Lake 1997) →
    Gasoline (~ 2000s, Wadsley, Stadel & Quinn 2004)
  - Density-Energy SPH, Cooling, Ionization/UV
  - Star formation+Feedback (Stinson+ 2006 -- Blastwave)
- Changa (~ 2010s, Menon+ 2015)
  - Charm++ (C++) for parallel, Space-filling curve, Oct Trees
  - SPH, SF, Cooling etc… same as Gasoline(2)
- Gasoline2 (~ 2010s, Wadsley, Keller & Quinn 2017)
  - Updated SPH, Diffusion, Superbubble Feedback



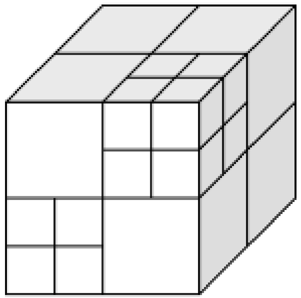
Charm++

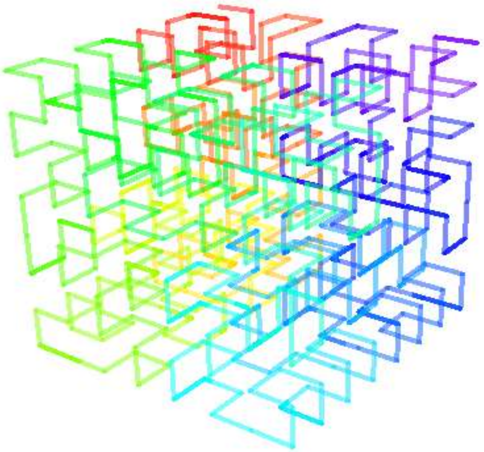
- C++-based parallel runtime system
  - Composed of a set of globally-visible parallel objects that interact
  - The objects interact by asynchronously invoking methods on each other (e.g. calling a function)
- Charm++ runtime
  - Manages the parallel objects and continuously (re)maps them to processes to balance work
  - Provides scheduling, load balancing, and a host of other features, requiring little user intervention

Architecture: Trees & Parallel



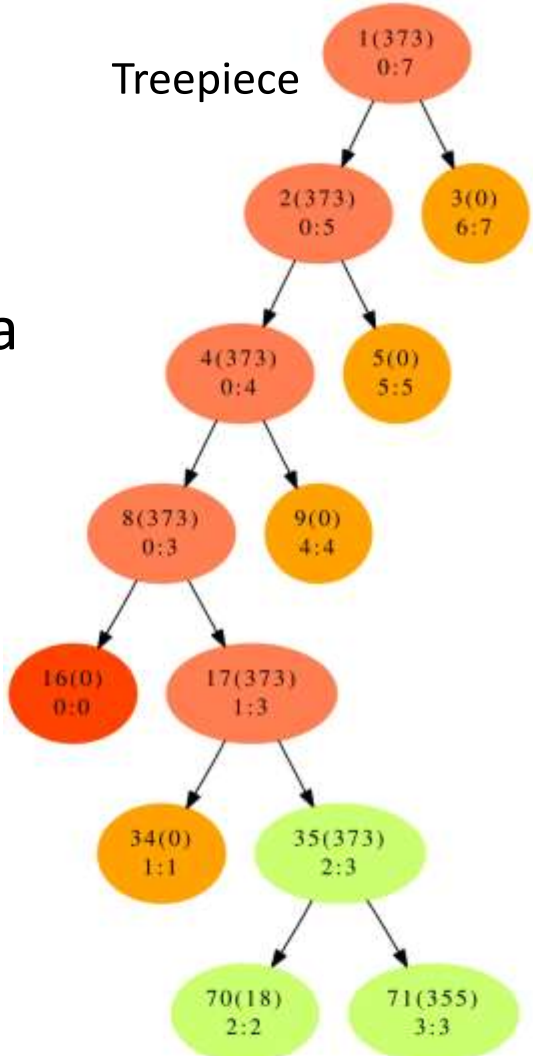
- p**kd**grav – KD-tree used for load balancing (dividing work/data), gravity (initially) and SPH
  - moderate work to build (2x)
  - hard to divide work >~ 100 cores
- ChaNGa – Oct tree, good for gravity, build once. Divide work with space-filling curve
  - fast, single tree-build
  - Scale to 100,000+ cores!





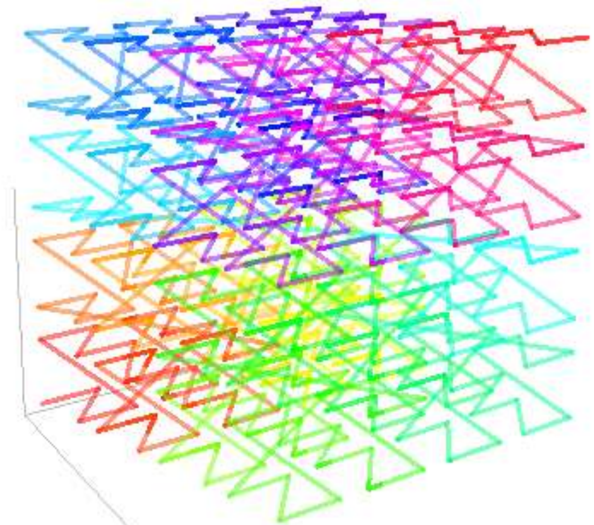
Parallel Design

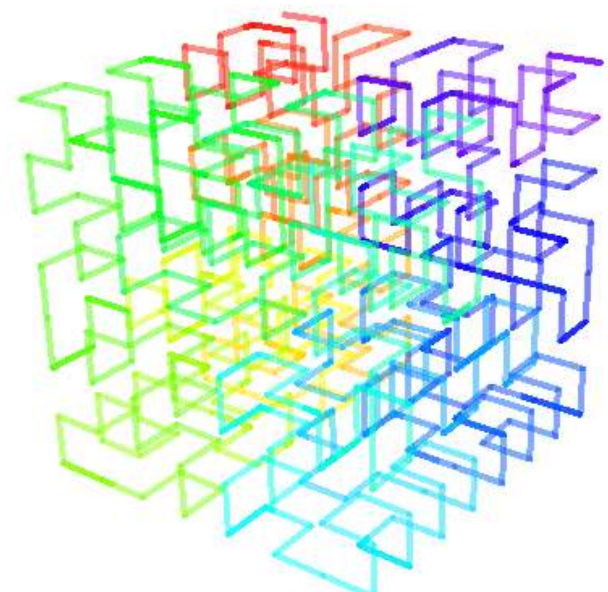
- Pkdgrav/Gasoline:
  C/object-oriented design: master, PST, PKD from tree. Load balance manually every few step. Each local domain rectangular and executed on one core. Cache for remote data (tree cells & particles)
- ChaNGa:
  Charm++ (parallel extension of C++, Kale & Krishnan 1993). Domains are “tree-pieces” (Charm++ objects) of the oct-tree. Many (~ 10) per core.
  - Automated work migration, checkpointing, load balance via Charm runtime. Remote data access, remote tasks (function calls)
  - Cache remote data, work overlap, GPUs



Changa: Domain Decomposition Options

- Space-filling curves
  - Morton ordering
  - Peano-Hilbert
- "Oct": fully contained nodes
  - Less communication
  - Harder load balancing
- ORB (orthogonal recursive bisection)
  - Poor gravity





Changa:

Tree Building

- Sort on Keys: particles are in tree order
- Determine count of particles in each Node
- Assign NodeKey: each bit a left-right branch
- Stop at “buckets”: each leaf contains a few particles.
- Construct multipole moments
  - Request moments of External Nodes
- Merge pieces on same address space.

Gravity Algorithms

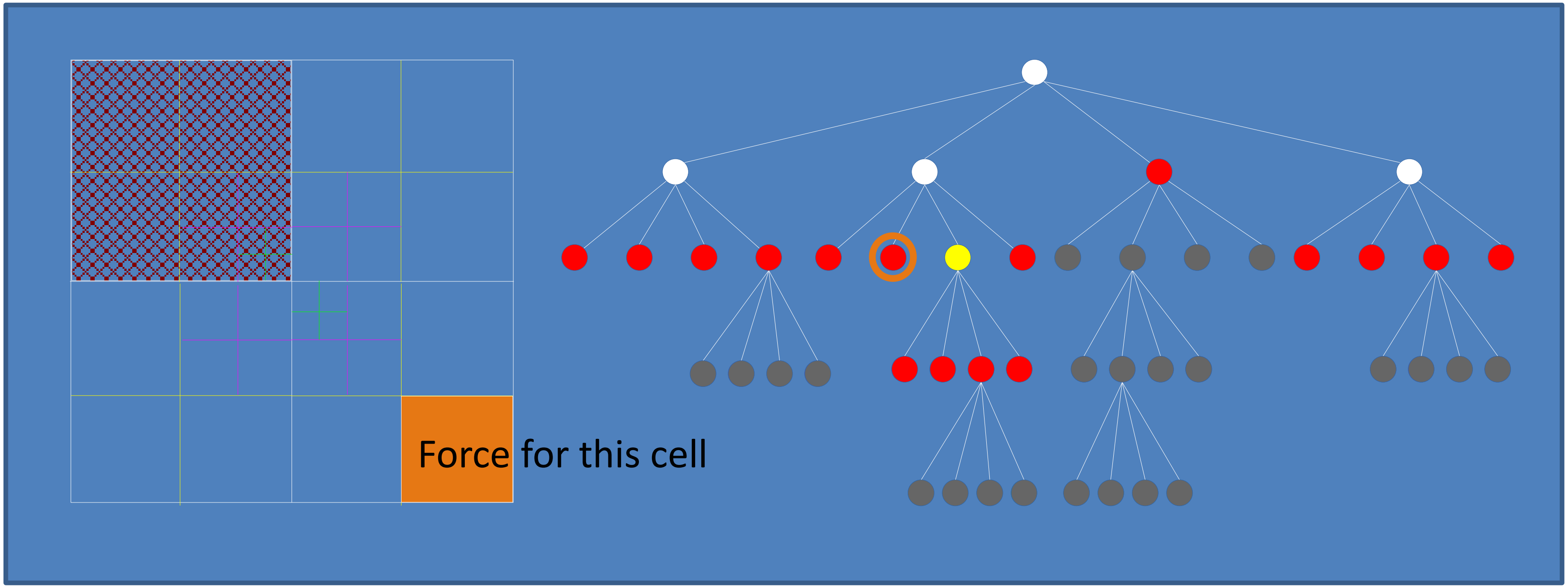
Newtonian gravity interaction

- Each mass is influenced by all others: O(n^2) algorithm

Barnes-Hut (1986) Tree approximation: O($n \log n$)

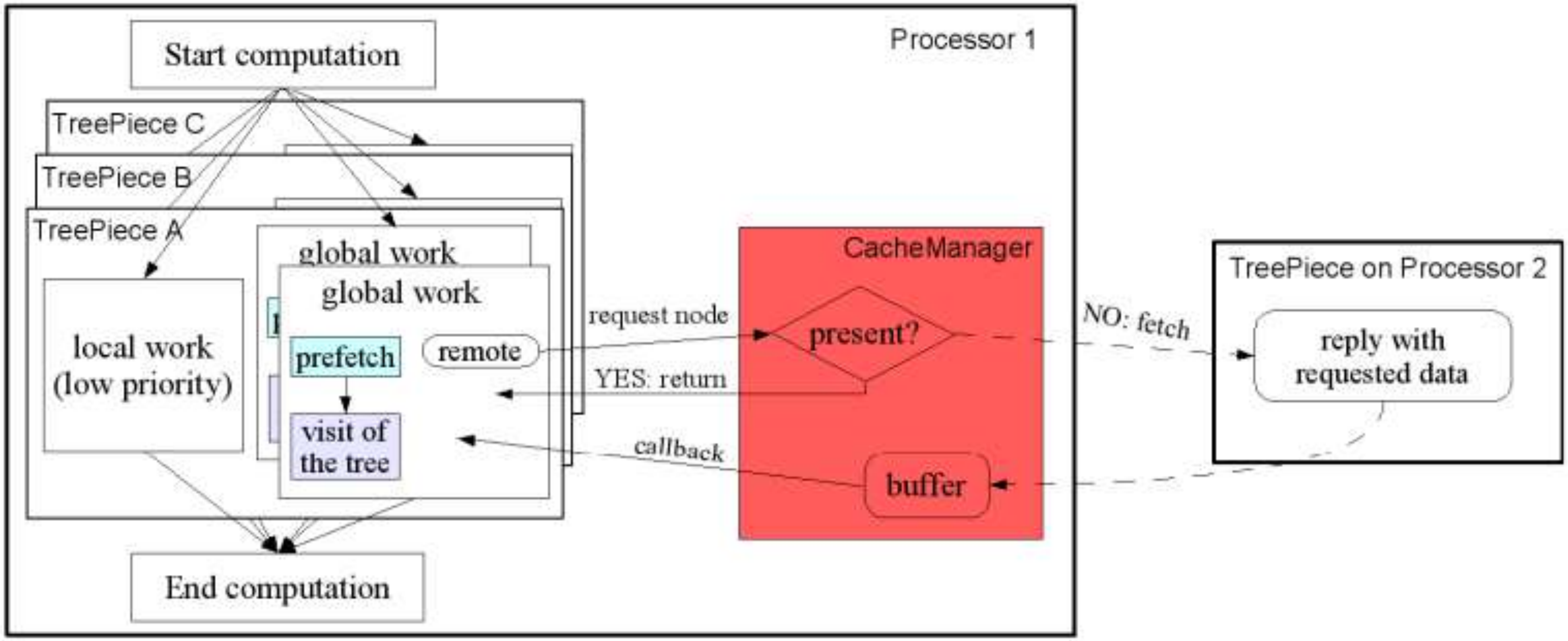
- Distant particles combined into center of mass of tree cell
- Expand to hexadecapole (4^{th}) order (Stadel 2001)
- Cells are "buckets" with up to nBucket particles O($n\, n_{bucket}$) part

Fast Multipole Method (e.g. Dehnen, 2000, approximation: O(*n-ish*)



Changa:

Overall treewalk structure



Building the codes…

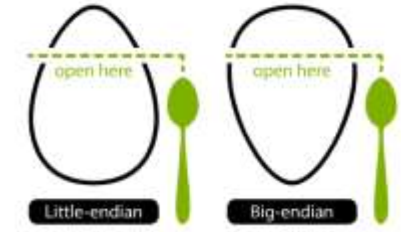
ChaNGa

https://github.com/N-BodyShop/changa/wiki/Flatiron-Quickstart

- Get charm, ChaNGa – build charm & ChaNGa
- Run ChaNGa included examples: teststep, testcosmo

Initial Condition & Conventions

Tipsy format



tipsydefs.h

```
#define MAXDIM 3
typedef float Real;
struct gas_particle {
    Real mass;
    Real pos[MAXDIM];
    Real vel[MAXDIM];
    Real rho;
    Real temp;
    Real eps;
    Real metals ;
    Real phi ;
} ;

struct dark_particle {
    Real mass;
    Real pos[MAXDIM];
    Real vel[MAXDIM];
    Real eps;
    Real phi ;
} ;

struct star_particle {
    Real mass;
    Real pos[MAXDIM];
    Real vel[MAXDIM];
    Real metals ;
    Real tform ;
    Real eps;
    Real phi ;
} ;
```

Header

```
struct dump {
    double time ;
    int nbodies ;
    int ndim ;
    int nsph ;
    int ndark ;
    int nstar ;
} ;
```

Tipsy binary
“native” little endian (=Intel)
Header 28 or 32 bytes (Annoying)

Tipsy Standard
“std” big endian via xdr libraries (=Sun)
(also annoying)
Header exactly 32 bytes
e.g. cube300.000128

Data not in this format output as tipsy arrays: just flat binaries with an integer (4 byte) size and then data

e.g. cube300.den
cosmo16.HI
cosmo16.HeI
…

Units

- G = 1 => only two of mass, distance, time specified.
- Solar System: D in AU, M in $M_\odot$, T in years/2π
- Galaxies: D in kpc, T in Gyr (.9778 km/s), M in 2.22306e5 $M_\odot$

Cosmology Units

- Recall: $H^2 = (8\pi\rho_c/3)$ and $\Omega = \rho/\rho_c$
  - If we choose $\rho_c = 1$, then
    - $H = (8\pi/3)^{1/2} = 2.894405$
  - Time is 2.894405*(Hubble time)
- Choose boxsize = 1 then $M_{box} = \Omega$, and velocity unit = (Hubble velocity across box)/2.894405
  - Mass unit is $H^2 * (\text{boxsize})^3 * 3/(8\pi G)$

Comoving → Code → Physical

physical units

$r_{phys} = a\, r_{code}$ x kpcunit
$v_{phys} = 1/a\, v_{code} + \mathrm{adot}\, r_{code}$
$phi_{phys} = 1/a\, phi_{code}$
$u_{phys} = u_{code}$
$\rho_{phys} = 1/a^3\ \rho_{code}$

Output (Tipsy file)

$r_{out} = r_{code}$
$v_{out} = 1/a^2\, v_{code}$
$phi_{out} = phi_{code}$
$T_{out} = T(u_{code}=u_{phys})$
$\rho_{out} = \rho_{code}$

Comoving

$r\_{com} = r_{out} = r_{code}$
$v\_{com} = v_{out} = 1/a^2\, v_{code}$

Physical Units

$r_{phys} = a\, r_{out}$ x kpcunit
$v_{phys} = a(v_{out} + H_{out}\, r_{out})$ x kmsunit
$H_{out} = H(a)/H0 \times \mathrm{sqrt}(8\,\pi/3)$
(G=1)
$Ha/H0 = \mathrm{sqrt}(\Omega_M (1+z)^3 + \Omega_\Lambda)$

$phi_{phys} = 1/a\, phi_{out}$ x ergpergmunit
$\rho_{phys} = 1/a^3\ \rho_{out}$ x gmperccunit(z=0)
$m_{phys} = m_{out}$ x MsolUnit

Runtime Parameters

https://github.com/N-BodyShop/changa/wiki/ChaNGa-Options

Every run requires a parameter file to start it

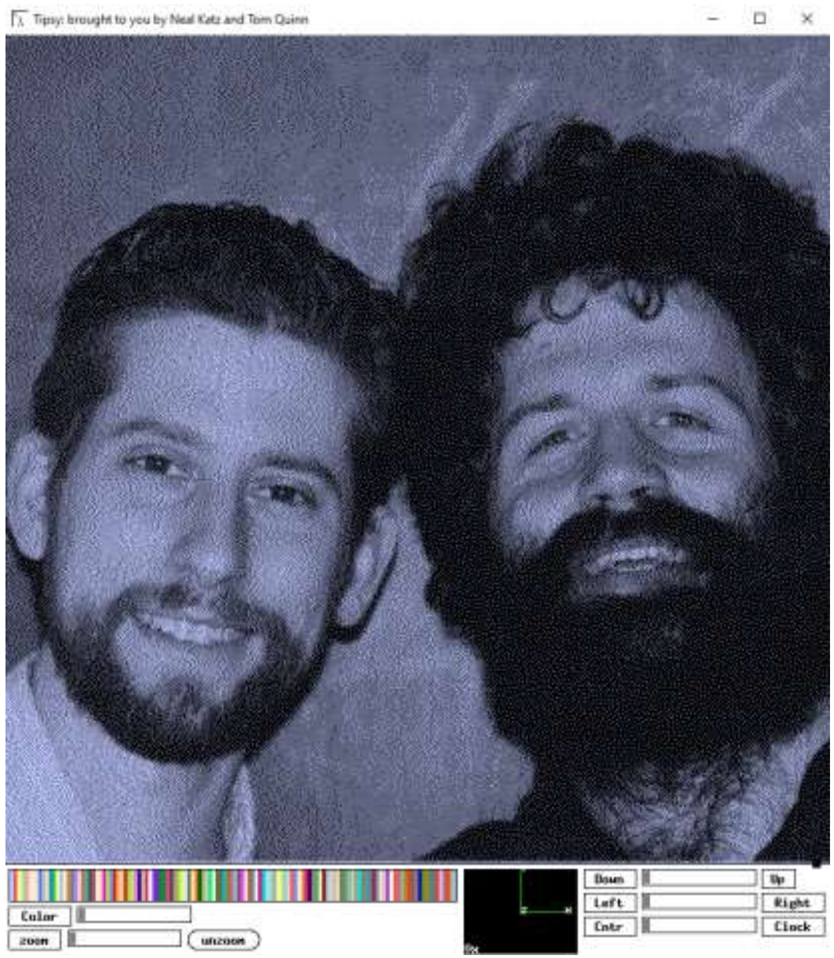
- ach – characters, e.g. achInFile (Tipsy format input file)
- b – binary on/off, e.g. bPeriodic, bDoGas (do hydro/SPH forces)
- i, n – integer, e.g. nSteps (number of timesteps)
- d – real value, e.g. dESN (supernova energy, ergs)

Tend to favour dimensionless units or code units except cooling/SF which is in CGS. Defaults are usually given. All parameter choices recorded in .log files (always save your .log file!). All parameters described in master.c (Gasoline) or ParallelGravity.cpp (ChaNGa) using “prmAddParam” functions.

Can also be specified on command line and saved in checkpoint files.

Continuing with Tipsy…

https://github.com/N-BodyShop/changa/wiki/Flatiron-Quickstart

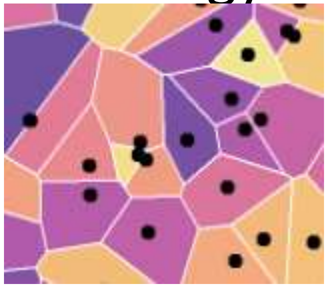


Part II: Hydrodynamics, Gasoline2/ChaNGa, Parameters, Test Problems, Python

Hydro Basic Methods

Two major flavours:

- Lagrangian = SPH (particle based), gradients for fluxes, (e.g. Lucy 1977, Monaghan & Gingold 1977)
  - 2nd order (noisy), naturally adaptive, good dynamics (orbits)
- Eulerian = Finite Volume (fixed grid based) with approximate Riemann Solvers for fluxes: 1st order (Godunov 1959), 2nd order (van Leer, e.g. RAMSES (Teyssier 2002)), 3rd order "PPM" (Colella & Woodward 1988, e.g. ENZO Bryan & Norman 1997, ATHENA Stone+ 2008)
  - Really good shocks/instabilities (low diffusion), diffusive for orbits/advection, need adaptivity (AMR) for Astro/cosmology problems (e.g. RAMSES, ENZO), dynamics bad
- Hybrid: (GIZMO, Hopkins 2015 mostly SPH, AREPO: moving Voronoi mesh (2nd order similar to Ramses))



SPH: Smoothed Particle Hydrodynamics

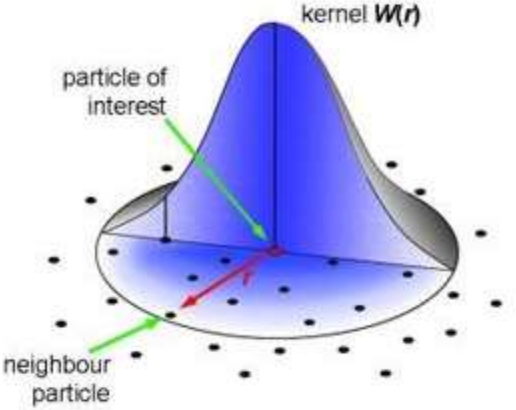
Basis: optimal density estimators for disordered points (can use for any data, e.g. astronomical data) using Kernel Functions

Smoothed density or any physical field, can take gradients for terms in fluid (Euler equations)

Symmetrize force expressions for momentum, angular momentum & energy conservation BUT adds noise

> Smooth glass state still small forces (cannot model arbitrarily small perturbations less than grid noise, very subsonic turbulence, streaming instability SI). Equivalent problem: SPH only approximately divides volume among particles (unlike a grid)

see: Monaghan 1992, Springel & Hernquist 2002, Springel 2009 (AREPO), Read+ 2010



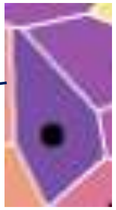
$$\int \nabla P \, dV = \oint P.n \, dS$$

Exact volumes+ΔS => if P = constant then Integral exactly zero (e.g. Finite Volume, AREPO). If ΔS approximate, integral zero only on average

Voronoi cell (AREPO)
Exact Volumes+Surfaces

$\hat{n}$

ΔS



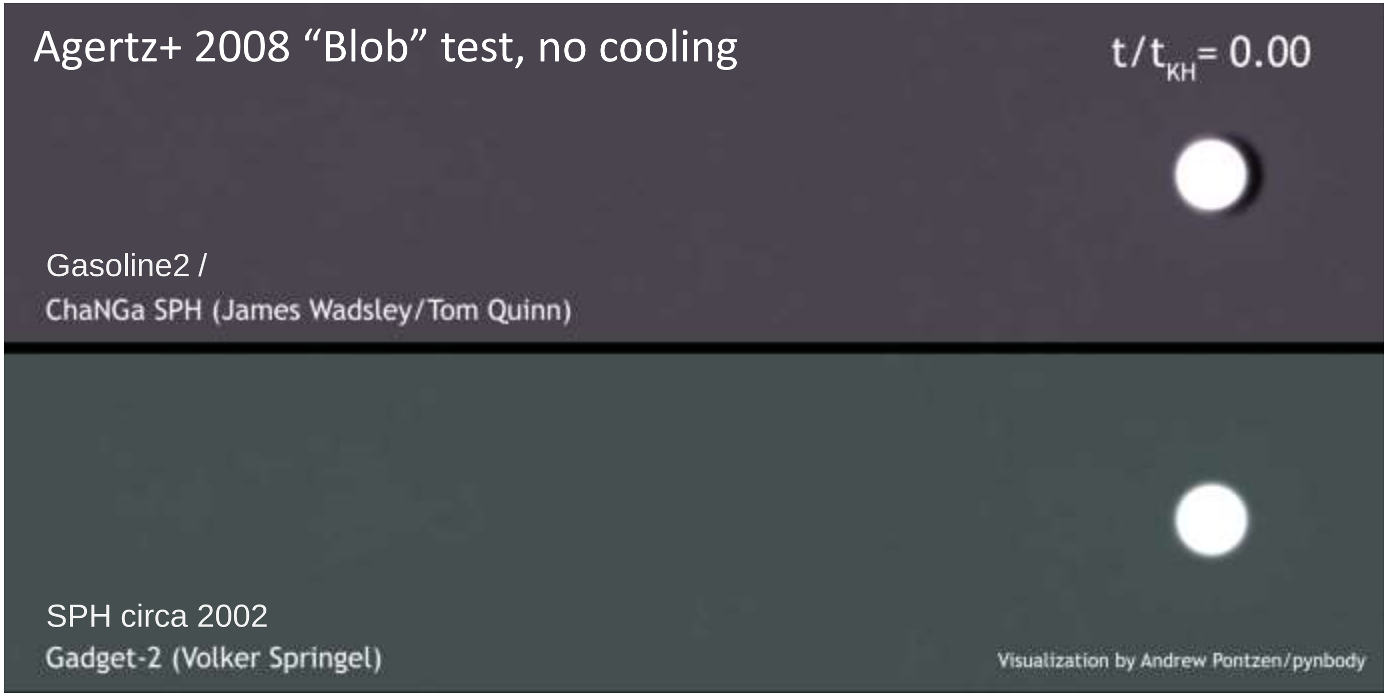
$$\sum \Delta S \, \hat{n} = 0$$

Lagrangian Hydro History

- PM-SPH (Evrard 1988)/ TreeSPH (Hernquist & Katz 1989) – vanilla "Traditional" SPH (see Monaghan 1992 ARAA): Energy and density formulation
- Gadget-2 (Springel Hernquist 2002) – Entropy SPH (from a Lagrangian) iterate for constraint: density = K mass/h^3
- Problems: No diffusion → extreme metallicities, strange entropies
  - Fix: Turbulent diffusion (Wadsley+ 2008, Shen+ 2010)
- Agertz 2008 "blob" test: Surface tension problems with SPH (need for "modern" SPH)
- Modern SPH: Must have diffusion, must remove surface tension
  - Fix: GIZMO (Hopkins 2015): SPH densities but forces from 2nd order Finite Volume-like approach (still not zero in uniform density but better)
  - Fix: Gasoline2: Geometric Density Forces: minimal surface tension (see Wadsley+ 2017, K. Dolag SPH)
  - Fix: AREPO: Voronoi Cells => Volume partitioned perfectly, forces zero in uniform density
  - No Fix: Gadget 2,3,4 (2022) never fixed this (yet)

Agertz 2008 “blob” test

- Test 3 of the Wengen comparison project, cold equal pressure blob (10x density) in supersonic wind tunnel
- Movie shows density vs. time. Blob catches up to wind speed, mixing fast (KH instability rate ~ Δv k) then slows



What is going wrong? Mixing & Surface Tension

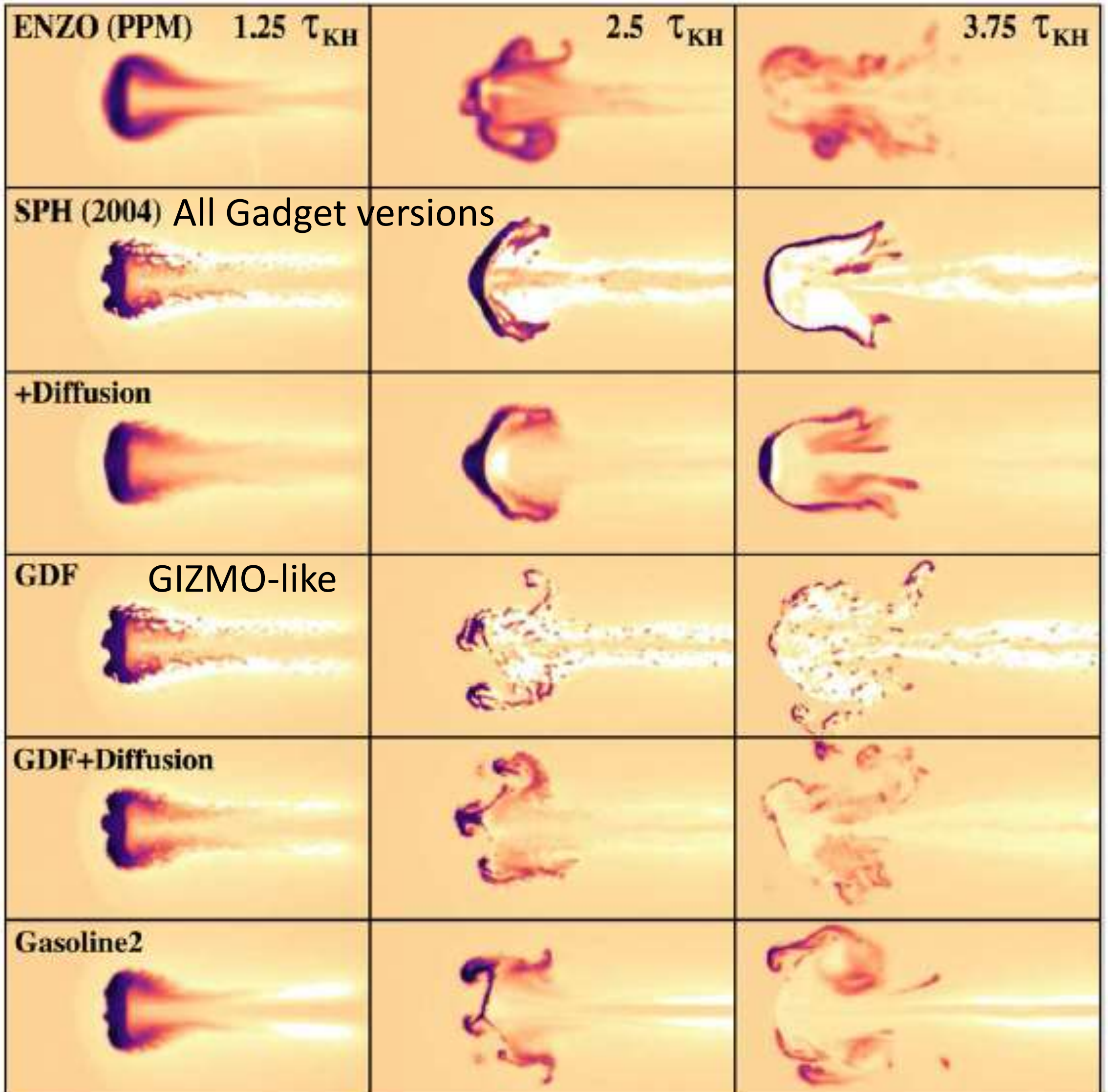


Figure 5 from Gasoline2 paper (Wadsley+ 2017) Blob test in **entropy**. Shows mixing (or not) clearly.

(1) Diffusion on/off
SPH has no intrinsic mixing

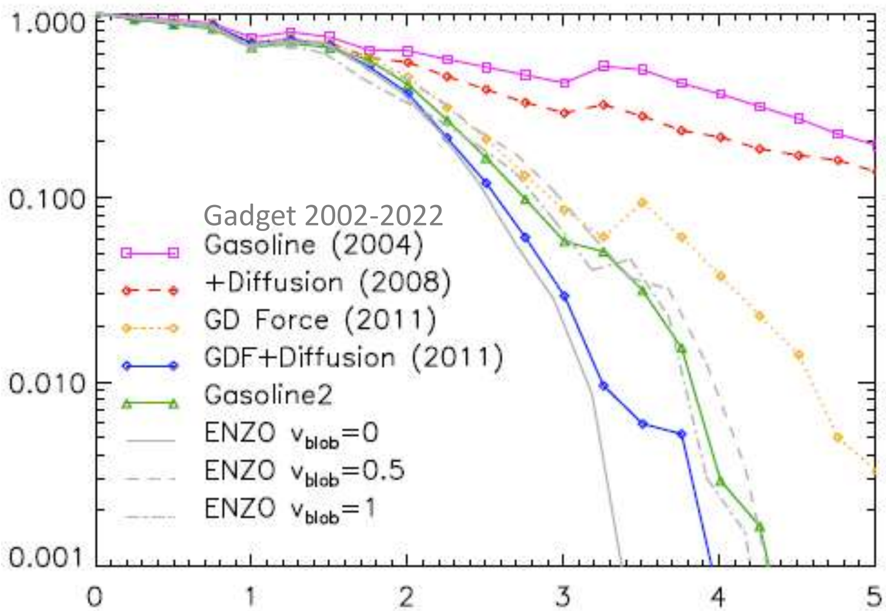
Small scale turbulence mixes all quanties (entropy, velocity, metals, etc…) at rates ~ Δv k (i.e. faster with small scales $t_k \sim k^{-1/2}$ so all scales mix in finite time)

Surface tension inhibits KH instability but absent in Astro, must avoid numerical surface tension!

(2) GD Force on/off
Force expressions can do poorly at high density gradients (due to lopsided particle numbers). Geometric Density (Gasoline2/ChaNGa) does well even in high density gradients

Mixing vs. time: Open questions

High Density gas Left vs. time



Exact answer tricky – even high order finite volume (PPM) depends on whether blob is moving v_{blob} = 1 (freezes out) or wind is moving v_{blob} = 0 (mixes continuously)

Real physics is Galilean invariant – moving blob or moving wind should give same answer

Gasoline2 (green + blue lines) within uncertainty of HR grid result

Exact behaviour depends on cooling in mixing layers. Hot topic for feeding gas fuel to galaxies, e.g. Fielding+ 2022

Food dye in water “freeze out” of mixing

Running the codes…

Hydro

https://github.com/N-BodyShop/flatiron-tests

Hydro test cases

- Evrard collapse – fast
- Sod shocktube – fast
- cosmo16 – make your own cosmo IC and run it
- blob test – slow, may want to look at outputs provided
- AGORA test – slow, may want to look at outputs provided

Manipulating tipsy format/ICs

- For test problems: pytipsy is great. Generate particle data and write in std Tipsy format
  - Starting with a glass IC is a good idea for tests
- Command line: **printtipsy** to view basic info
  - https://github.com/N-BodyShop/how-to
- For cosmological ICs: MUSIC (Hahn+Abel 2011) https://www-n.oca.eu/ohahn/MUSIC/
  - Fancy Cosmo ICS: GenetIC (Stopyra, Pontzen+ 2020)
- For isolated galaxies : MAKEDISK (Springel +2005) and others

- Thanks