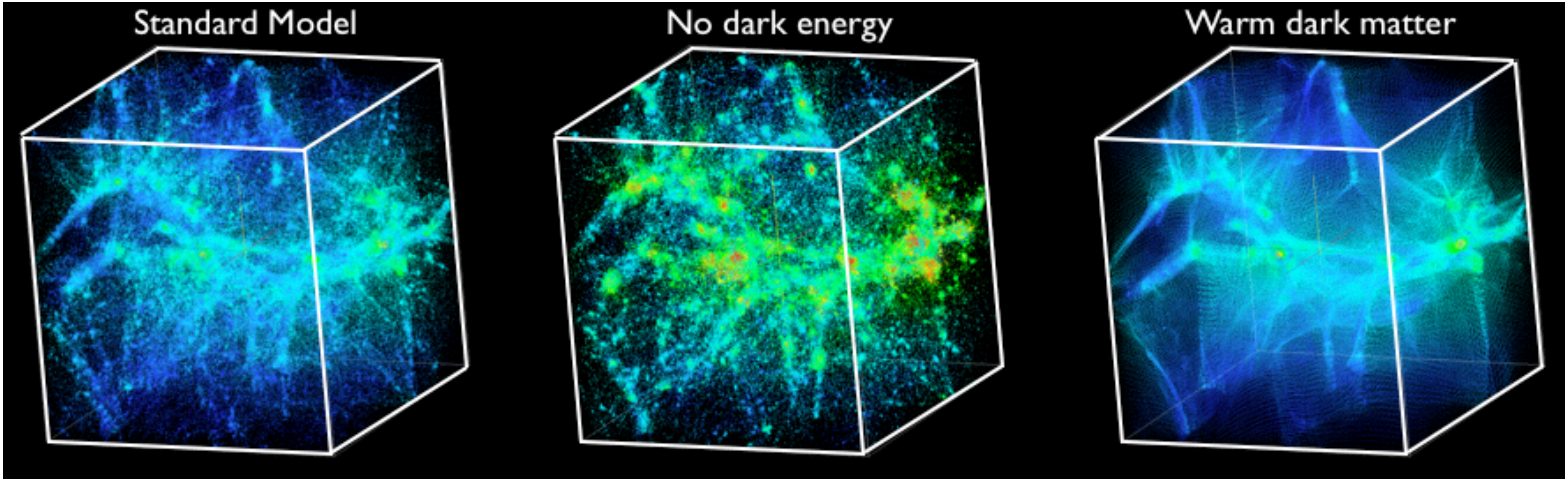
Cosmological Simulations: Under the Hood

Salman Habib
Argonne National Laboratory
Kavli Institute for Cosmological Physics



http://www.nobelprize.org/nobel_prizes/physics/laureates/2016/advanced-physicsprize2016.pdf

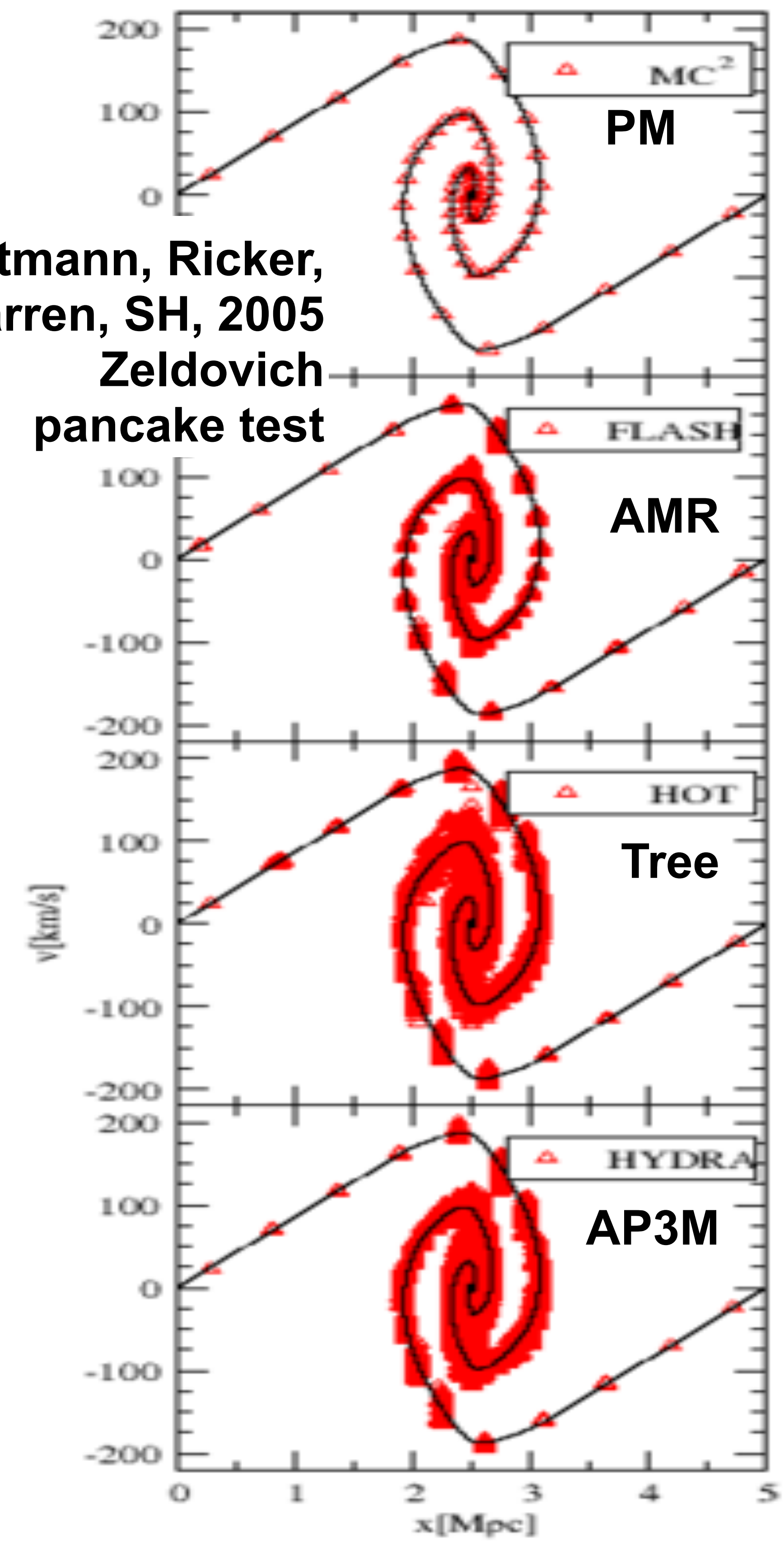
Haldane, Kosterlitz, Thouless (topological phase transitions)
Home assignment — read the description of their work!

Mexican Numerical Simulations School
Lecture 2, October 4, 2016

Particle Approaches to the VPE: Overview

- **General remarks about particle methods**
  - Note that we are interested in a kinetic description because we have a collisionless system — the obvious fluid description is simply wrong (CDM is not an ideal gas!)
  - Later on we will like to combine the dark matter description with a separate Euler description for the baryonic fluid — how to best combine these two descriptions? (also, don’t forget neutrinos!)
  - Because particle methods are intrinsically discrete, error analysis is subtle — many things can go wrong, and they often do!
  - Sometimes the best way to test results in complex simulation problems is to run multiple algorithms and compare results, so it is important to not focus on a single technique too much, but develop a suite of methods that have reasonable overlap in their domains of validity

Heitmann, Ricker, Warren, SH, 2005
Zeldovich pancake test



Particle Approaches: PM Method

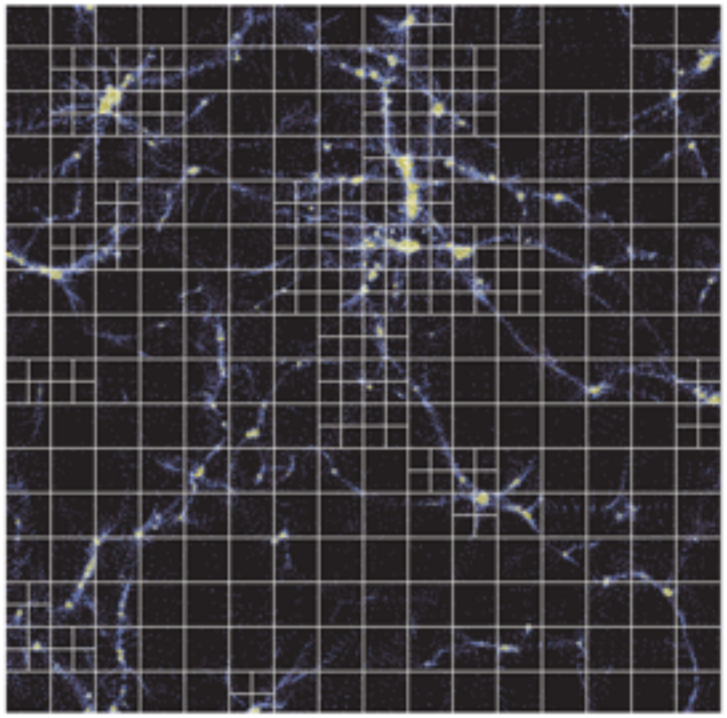
- **Particle-In-Cell (PIC)/Particle-Mesh (PM)**
  - Reminder: Use tracer particles to model ***collective*** effects
  - Sequence of events: 1) generate ICs (particle positions and velocities), 2) generate density field on a grid, 3) solve the Poisson equations, generate gradient of the potential, 4) move the particles, 5) repeat
  - Note all information is particle information, the grid is a temporary construct to smooth the particle distribution and to compute the smooth force (well, more or less smooth)
  - Different particle deposition and force interpolation strategies (NGP, CIC, TSC); note: need symmetry in the deposition and interpolation schemes to have explicit momentum conservation
  - Different levels of smoothing allow different orders of Poisson solvers, typically accuracy and spatial resolution are in opposition (this may seem counter-intuitive, but we will see why it is so)
  - If at fixed particle number, one increases the grid size arbitrarily, one gets back an "N-body" problem, what is the correct choice of N_p vs. N_g?

Particle Approaches: P3M Method

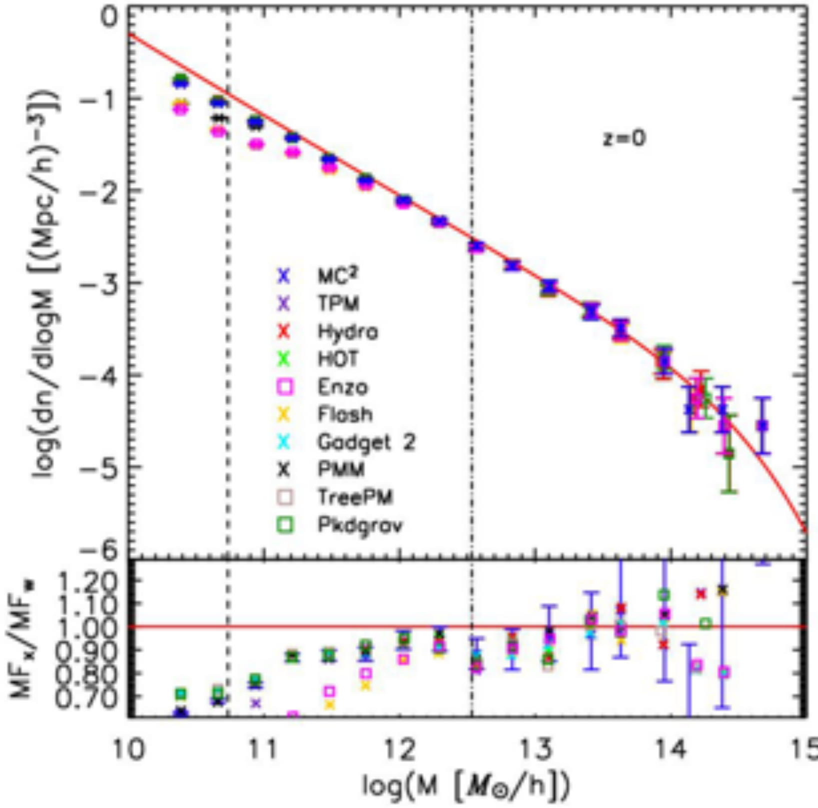
- **Particle-Particle Particle-Mesh (P3M)**
  - Fundamental problem of the PM method is the memory cost of the grid
  - If we can have a sufficiently large number of particles, increasing N_g to get enhanced resolution is potentially very expensive
  - In P3M, one splits the force computation into two parts, a long-range force computed via PM (which also leaks into small scales) and a short-range force computed via direct particle-particle interactions
  - Need to introduce a force smoothing scale for the particle-particle interaction to make sure we are still in the VPE limit (this is messy)
  - To basic PM need to add another construct to reduce the particle-particle computational costs, the chaining mesh
  - The particle-particle computations are expensive — for clustered problems, P3M can be potentially problematic
  - Until recently, the general view was that P3M is not competitive for cosmological simulations, but GPUs have changed this picture

Particle Approaches: AMR Techniques

- **Adaptive Mesh Refimement (AMR)**
  - Fundamental problem of the PM method is the memory cost of the grid
  - AMR attacks this by changing the size of the grid depending on the mass distribution
  - Need to understand how different AMR levels interact
  - Need to figure out a criterion for deciding the level of refinement (nontrivial, see bottom figure)
  - More complex data structures needed
  - Particle-mesh interaction complex (variable softening)
  - Need a multi-scale Poisson solver, unlike PM or P3M.
  - In high-resolution cosmological problems, because the resolution is needed “everywhere”, deep AMR requirements lead to high memory requirements
  - Currently, deep AMR methods are mostly used for cosmological hydrodynamics simulations

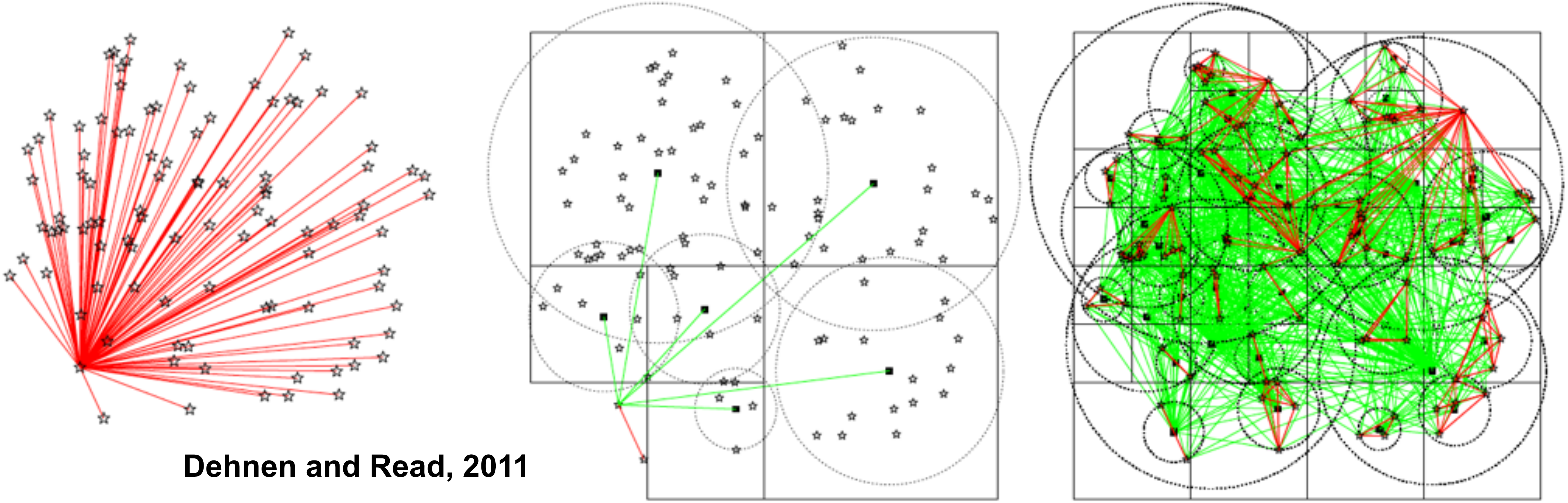


FLASH AMR hierarchy showing first two levels only, halo mass fn AMR issue below



Particle Approaches: Tree and FMM Techniques

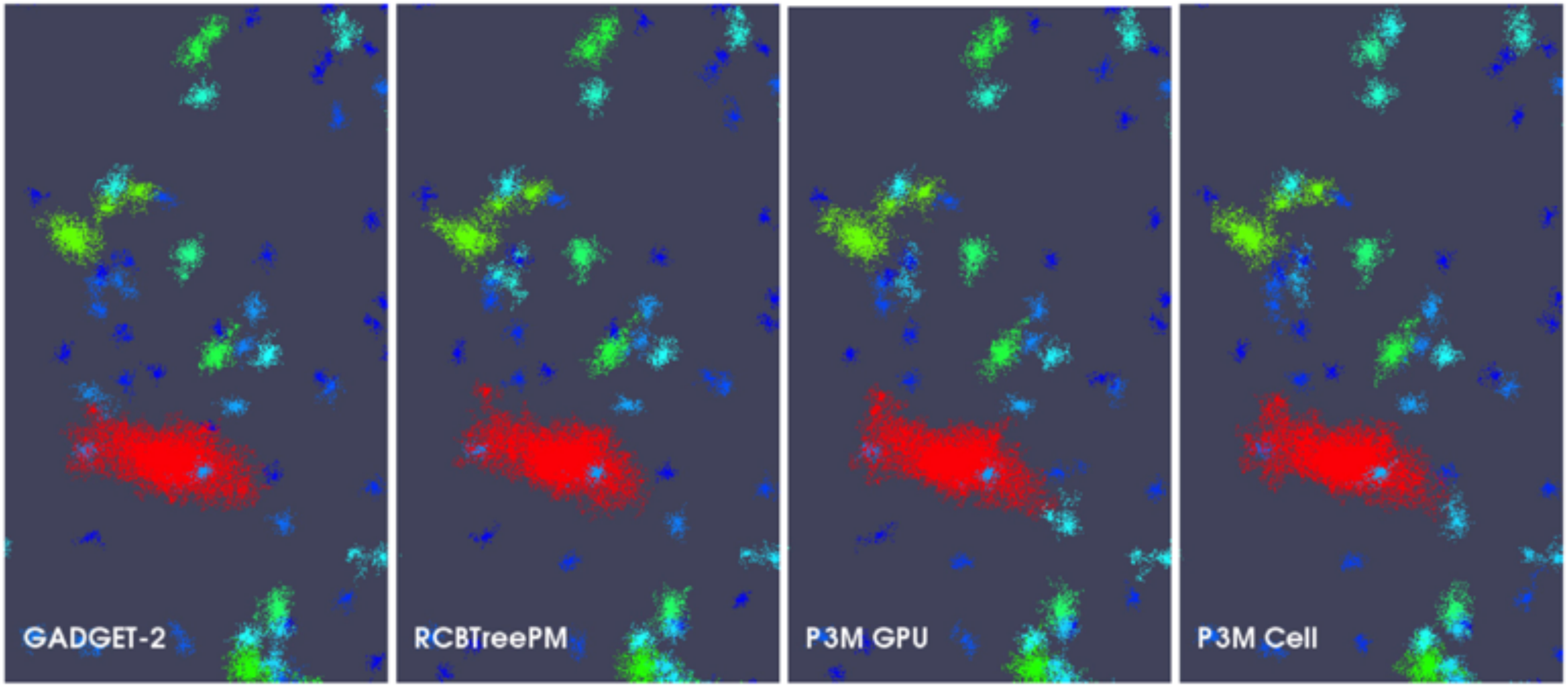
- **Tree and Fast Multipole Algorithms**
  - Avoid grids altogether and exploit multipole expansions (particularly useful for clustered situations)
  - Need error control criteria (e.g., opening angle, expansion order) and appropriate data structures (RCB trees, oct-trees, space-filling curves, etc.)
  - Fast and efficient (FMM has a double expansion of the Green's function)
  - Not naturally periodic, need to add periodic BCs via Ewald sums



Dehnen and Read, 2011

Particle Approaches: Hybrid Methods

- **Mostly PM + X (X=Tree or FMM)**
  - At large scales, PM methods are very convenient and fast, also good for evolving cosmological simulation at early times when clustering is small
  - Address weakness of PM codes at small scales via tree/FMM algorithms
  - Need to use force matching as in P3M but can be more relaxed because the efficiency of tree methods allows the matching point to be at a larger distance scale
  - HACC uses PM + X (low-order FMM) to address multiple architectures

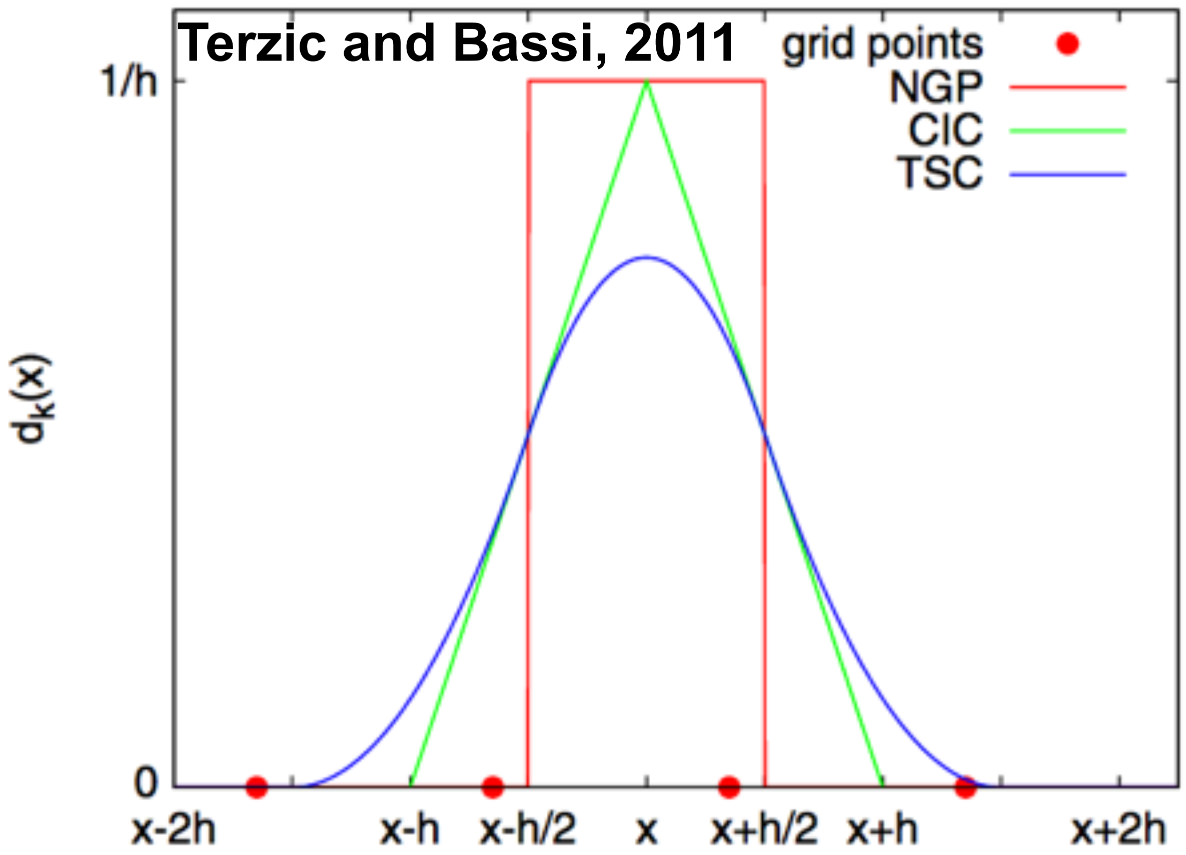


Comparison of multiple algorithms on the same problem, halos are color-coded according to the number of particles (blue~100, green~1000, red~10,000

SH et al. 2016

Case Study: PM

- **Particle deposition (density field)**
  - Use various weighting schemes to divide up the particle mass on nearby grid points (NGP — nearest grid point, CIC — volume-weighting to 8 grid points, TSC — quadratic on three nearest cells, 27 grid points)
  - NGP (field discontinuous), CIC (field continuous, gradient discontinuous), TSC (field and gradient both continuous)
  - NGP is rarely used (too noisy), CIC is most common
  - In Fourier space, the density weighting corresponds to sinc filters



1-D representation of NGP, CIC, and TSC; h is the grid spacing

$$[\sin(\pi k\Delta/L)/(\pi k\Delta/L)]^n$$

In Fourier space, n=1 is NGP, n=2 is CIC, n=3 is TSC

Case Study: PM (FFT-Based Poisson Solver)

- **Poisson equation on a grid**
  - 1-D:

$$\frac{\phi(x+\Delta)+\phi(x-\Delta)-2\phi(x)}{\Delta^2}=\frac{\partial^2\phi}{\partial x^2}+O(\Delta^2)$$

  - Use trignometric collocation to show that the Influence function is

$$G_2(k)=\frac{\Delta^2}{2}\frac{1}{\cos(2\pi k\Delta/L)-1}$$

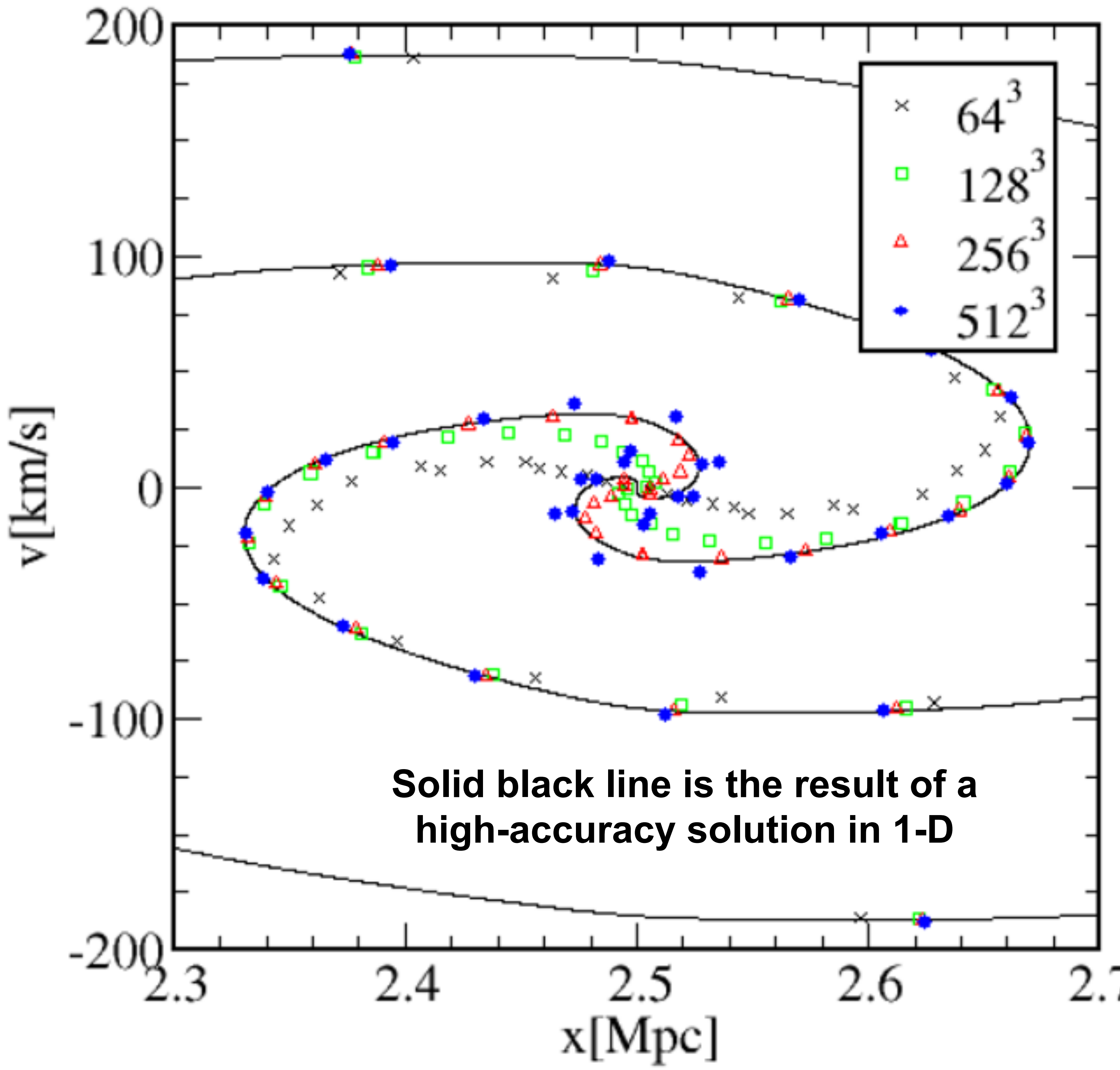
  - Note that this is "hotter" than the continuous Influence function, but it only appears in the Fourier integral multiplied by the CIC filtered density
  - It is trivial to show that G_2(k) multiplied by the CIC filter is

$$-\frac{L^2}{4\pi^2k^2}$$

  - This is the continuous Influence function
  - Usually one obtains the potential and differentiates it with a 2nd order stencil and then interpolates the gradient on the particle with inverse CIC (preserves momentum)

Case Study: PM (Planar Pancake Collapse Test)

- **Evidence of Collisionality**
  - Run the Zeldovich pancake collapse test at multiple grid resolutions with particle number fixed
  - Convergence must fail at some point (solution accuracy should improve for a while and then diverge)
  - Convergence can be tracked until failure near the mid-plane at 512^3 grid points



Solid black line is the result of a high-accuracy solution in 1-D

Case Study: PM (Higher-Order)

- **Higher-order Influence functions:**
  - 4th:

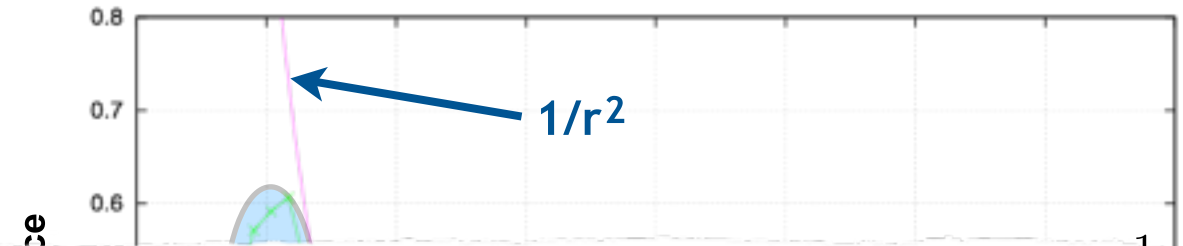
$$G_4(k) = \frac{3\Delta^2}{8} \frac{1}{\cos(2\pi k\Delta/L) - \frac{1}{16}\cos(4\pi k\Delta/L) - \frac{15}{16}}$$

  - 6th:

$$G_6(k) = \frac{45\Delta^2}{128} \frac{1}{\cos(2\pi k\Delta/L) - \frac{5}{64}\cos(4\pi k\Delta/L) + \frac{1}{1024}\cos(8\pi k\Delta/L) - \frac{945}{1024}}$$

  - These higher order functions should be used with appropriately smoothed density fields for their formal accuracy to be relevant
  - The smoothing can be performed in Fourier space with a sinc-Gaussian filter (to isotropize the force)
  - Higher-order gradients can be obtained directly in k-space using Super-Lanczos derivatives
  - Time-stepping is done with symplectic integrators

HACC PM Implementation



- **Spectral Particle-Mesh Solver:** Custom (large) FFT-based method -- uses 1) 6-th

$$G_6(\mathbf{k}) = \frac{45}{128}\Delta^2 \left[\sum_i \cos\left(\frac{2\pi k_i \Delta}{L}\right) - \frac{5}{64}\sum_i \cos\left(\frac{4\pi k_i \Delta}{L}\right) + \frac{1}{1024}\sum_i \cos\left(\frac{8\pi k_i \Delta}{L}\right) - \frac{2835}{1024}\right]^{-1}$$

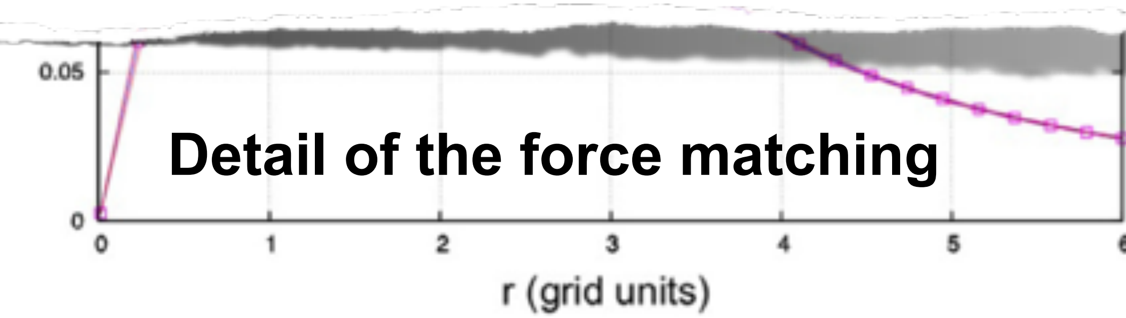
$$\left.\frac{\Delta f}{\Delta x}\right|_4 = \frac{4}{3}\sum_{j=-N+1}^{N} iC_j e^{(2\pi j x/L)} \frac{2\pi j \Delta}{L}\frac{\sin(2\pi j\Delta/L)}{2\pi j \Delta/L} - \frac{1}{6}\sum_{j=-N+1}^{N} iC_j e^{(2\pi j x/L)} \frac{2\pi j \Delta}{L}\frac{\sin(4\pi j\Delta/L)}{2\pi j \Delta/L}$$

where the C_j are the coefficients in the Fourier expansion of f

$$S(k) = \exp\left(-\frac{1}{4}k^2\sigma^2\right)\left[\left(\frac{2k}{\Delta}\right)\sin\left(\frac{k\Delta}{2}\right)\right]^{n_s}$$

$$f_{grid}(r) = \frac{1}{r^2}\tanh(br) - \frac{b}{r}\frac{1}{\cosh^2(br)} + cr\left(1+dr^2\right)\exp\left(-dr^2\right) + e\left(1+fr^2+gr^4+lr^6\right)\exp\left(-hr^2\right)$$

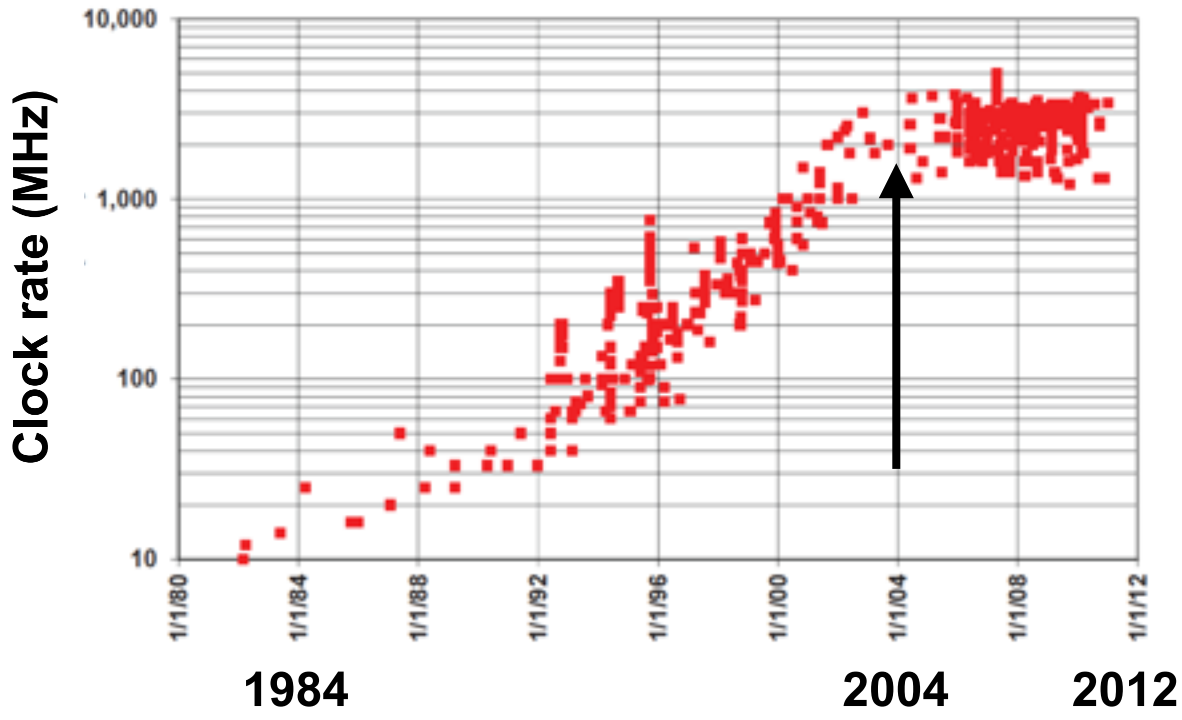
this later)



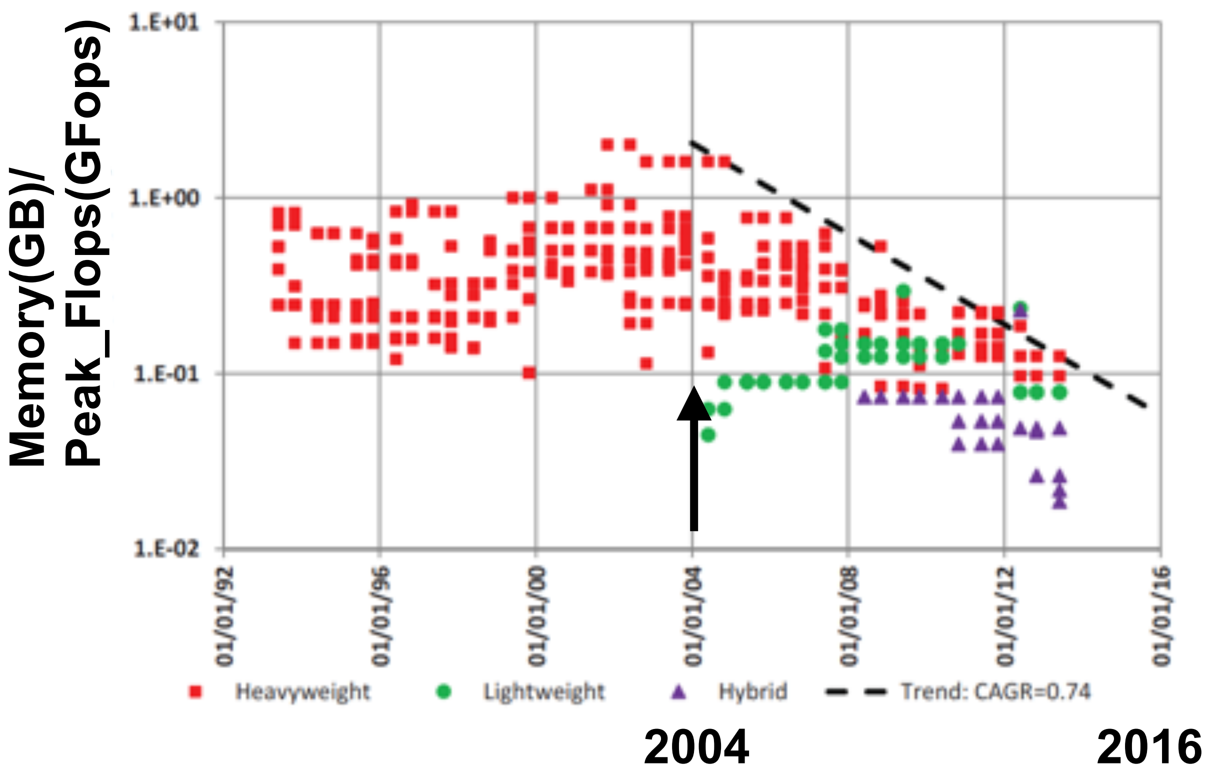
Detail of the force matching

Motivational Interlude: Hardware Evolution

- **Power is the main constraint**
  - Target: 30X performance gain by 2020
  - ~10-20MW per large system
  - Power/Socket roughly const.
- **Only way out: more cores**
  - Several design choices (e.g., cache vs. compute vs. interconnect)
  - All lead to more complexity
- **Micro-architecture gains sacrificed**
  - Accelerate specific tasks
  - Restrict memory access structure (SIMD/SIMT)
- **Machine balance sacrifice**
  - Memory/Flops; comm BW/Flops — all go in the "wrong" direction

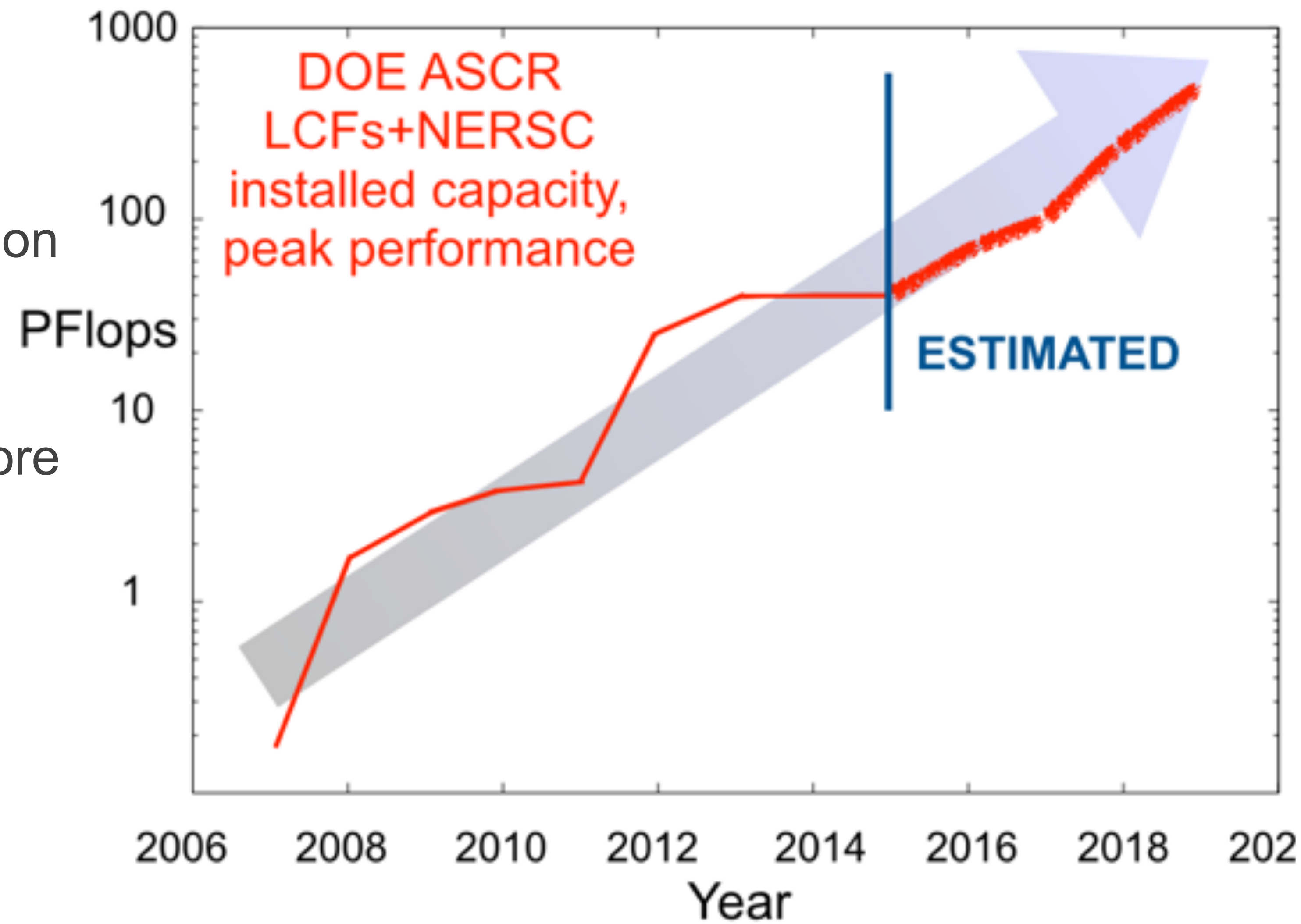


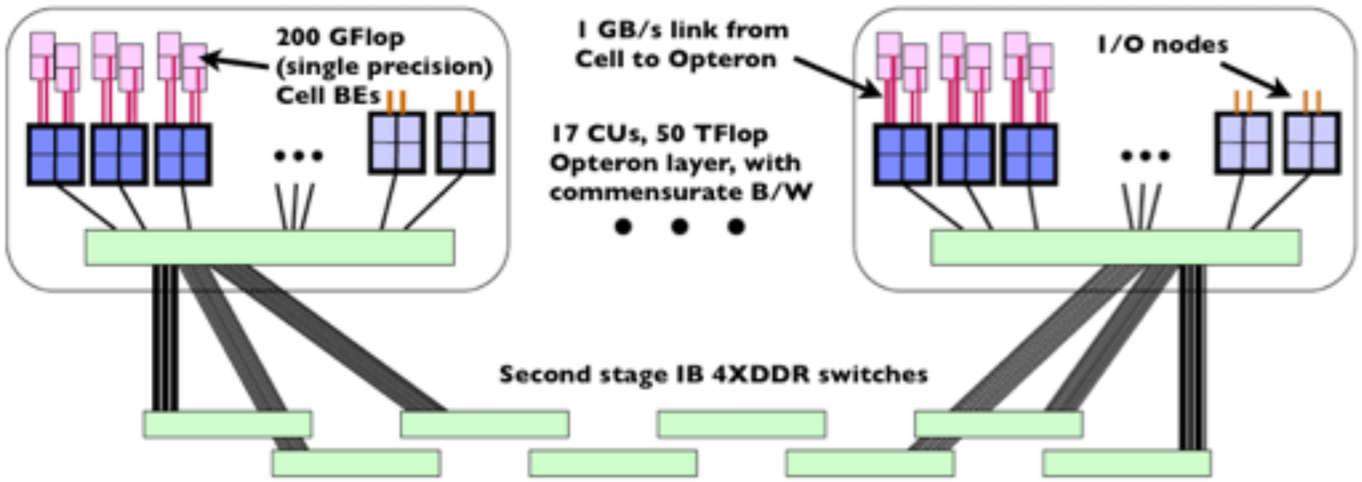
Kogge and Resnick (2013)



Emerging Architectures are Not New!

- **HPC systems:** “faster = more”
  - More nodes
    - Separate memory spaces
    - Relatively slow network communication
  - More complicated nodes
    - Architectures
      - Accelerators, multi-core, many-core
    - Memory hierarchies
      - CPU main memory
      - Accelerator main memory
      - High-bandwidth memory
      - Non-volatile memory
- **Portable performance**
  - Massively parallel/concurrent
  - Adapt to new architectures
    - Organize and deliver data to the right place in the memory hierarchy at the right time
    - Optimize floating point execution
  - Not possible with off-the-shelf codes

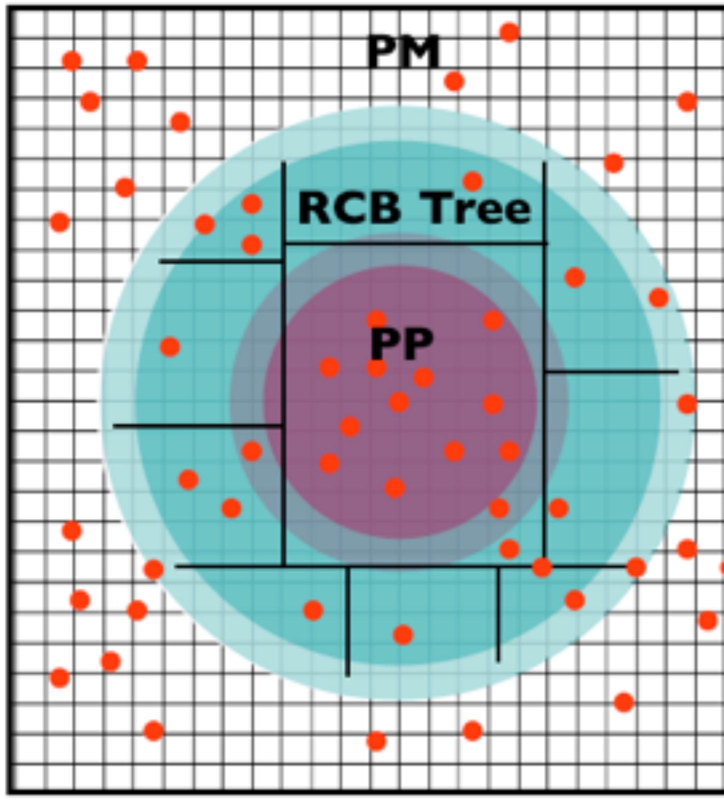




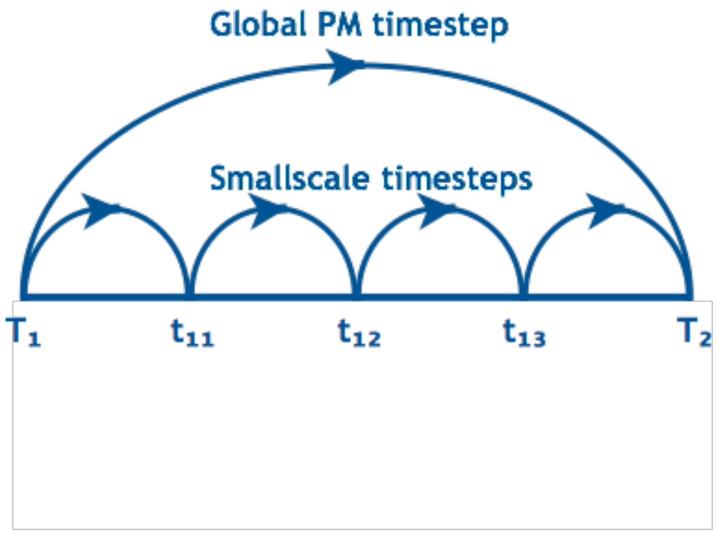
Roadrunner Architecture (2008)

HACC: Design Principles

- **Optimize Code ‘Ecology’:** Numerical methods, algorithms, mixed precision, data locality, scalability, I/O, in situ analysis -- life-cycle significantly longer than architecture timescales
- **Framework design:** ‘Universal’ top layer + ‘plug-in’ optimized node-level components; minimize data structure complexity and data motion -- support multiple programming models
- **Absolute Performance:** Scalability, low memory overhead, and platform flexibility; minimal reliance on external libraries
- **Optimal Splitting of Gravitational Forces:** Spectral Particle-Mesh melded with direct and RCB tree force solvers, short hand-over scale (dynamic range splitting ~ 10,000 X 100)
- **Compute to Communication balance:** Particle Overloading
- **Time-Stepping:** Symplectic, sub-cycled, locally adaptive
- **Force Kernel:** Highly optimized force kernel dominates compute time (90%), ***no look-ups*** due to short hand-over scale
- **Production Readiness:** runs on all supercomputer architectures; Gordon Bell Award Finalist 2012 and 2013, first production science code to break ***10PFlops*** sustained



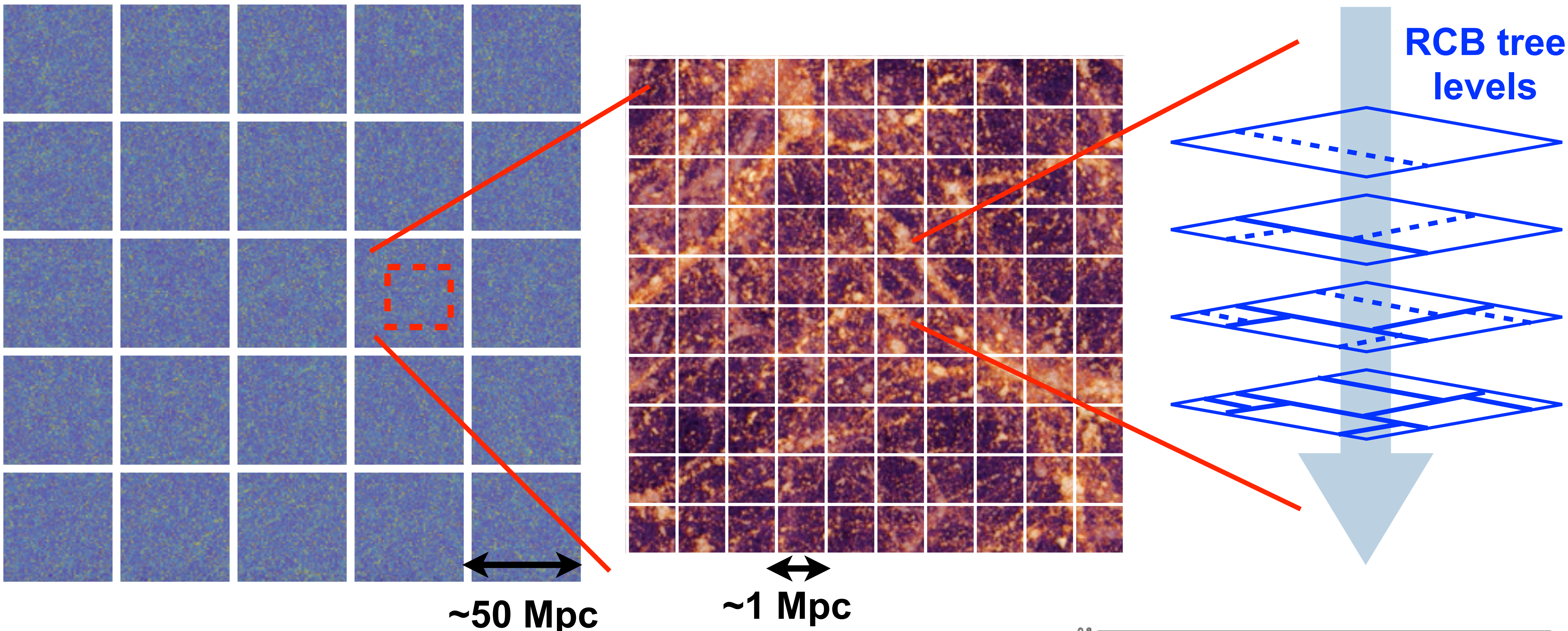
HACC force hierarchy (PPTreePM)



HACC Design: Portability Philosophy

- **Focus on Absolute Performance/Throughput:** High performance is a first-class requirement for HACC (portability here implies portability with ***absolute*** performance, not ***relative*** performance) — compute-intensive components control the performance of the code (not data motion)
- **Algorithmic Flexibility:** Allow for multiple algorithms in order to obtain the best possible performance on a given architecture (e.g., PPTreePM for BG/Q and Xeon Phi and P3M for GPUs)
- **Expert Tuning:** The code is designed so that the cost of obtaining peformance is limited to experts tuning small subsets of node-level plug-in code (via particle overloading in HACC) — this is a ***microkernel*** based approach (the microkernel is specific to the code, not a general purpose routine)
- **Portable Top Layer:** Maximize portability of non-performance critical framework within which the compute-intesive kernels reside (in HACC, the spectral PM method is “soft” portable in this sense)
- **Limit External Dependencies:** Minimize reliance on non-vendor supported libraries that could impact performance, portability, and time to implementation on new platforms (the main HACC simulation path is entirely free of such libraries)

Return from Interlude: ‘HACC In Pictures’

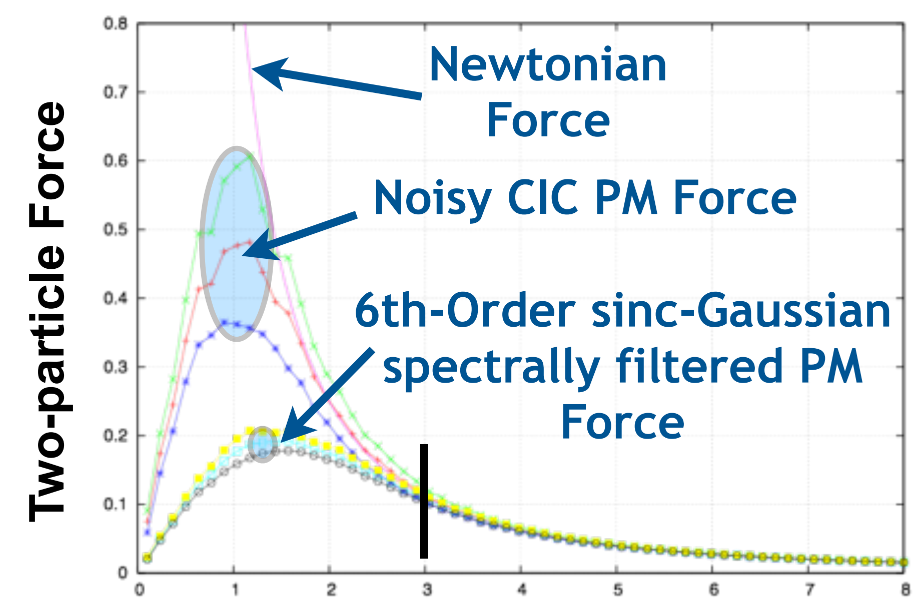


HACC Top Layer:
3-D domain decomposition with particle replication at boundaries (‘overloading’) for Spectral PM algorithm (long-range force)

Host-side

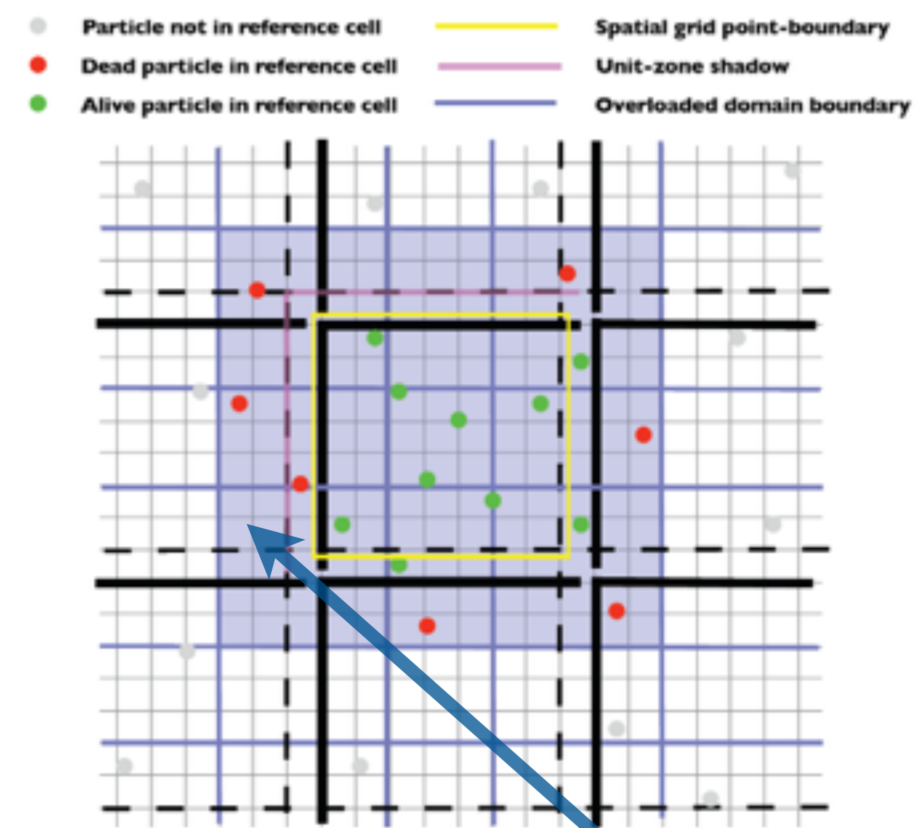
HACC ‘Nodal’ Layer:
Short-range solvers employing combination of flexible chaining mesh and RCB tree-based force evaluations

GPU: two options, P3M vs. TreePM



Particle Overloading and Short-Range Solvers

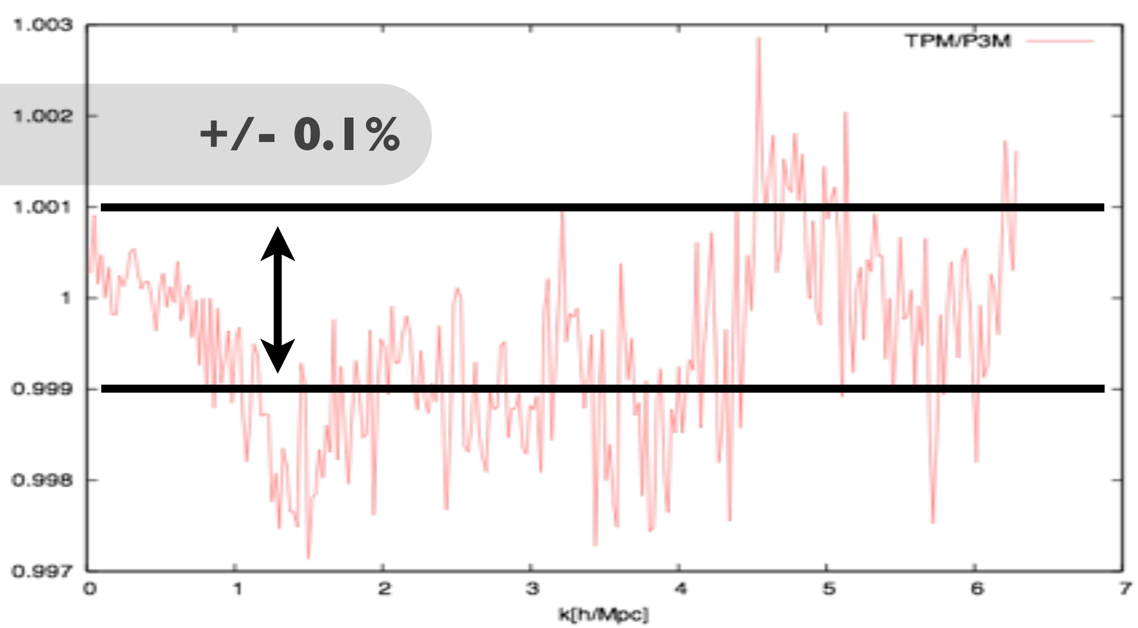
- **Particle Overloading:** Particle replication instead of conventional guard zones with 3-D domain decomposition -- minimizes inter-processor communication and allows for swappable short-range solvers **(IMPORTANT)**
- **Short-range Force:** Depending on node architecture switch between P3M and PPTreePM algorithms (pseudo-particle method goes beyond monopole order), by tuning number of particles in leaf nodes and error control criteria, optimize for computational efficiency
- **Error tests:** Can directly compare different short-range solver algorithms
- **Load-balancing:** Passive + Active task-based



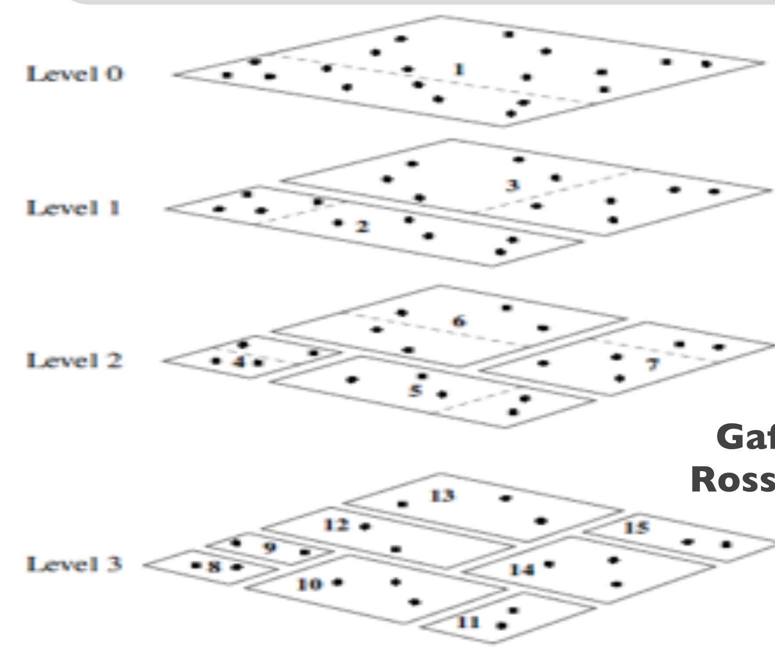
Overload Zone (particle 'cache')

+/- 0.1%

P(k) Ratio



HACC Force Algorithm Test: PPTreePM vs. P3M



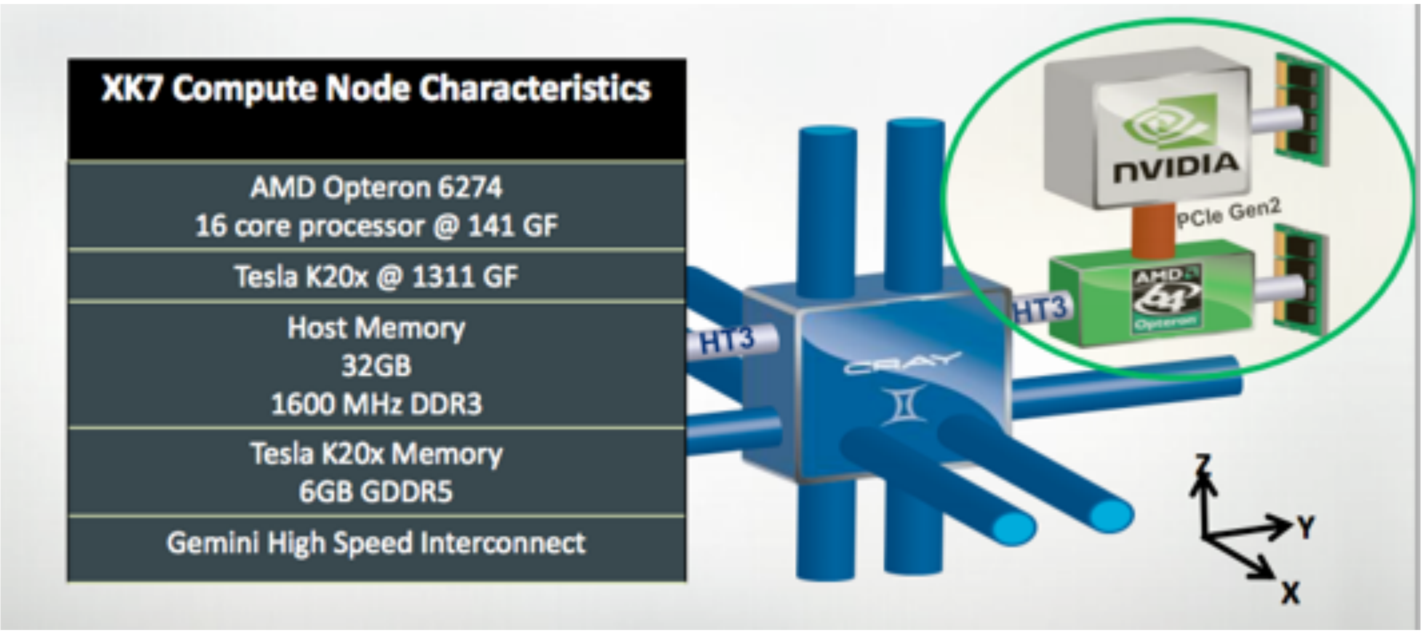
Gafton and Rosswog 2011

RCB Tree Hierarchy

Accelerated Systems: Specific Issues

Imbalances and Bottlenecks

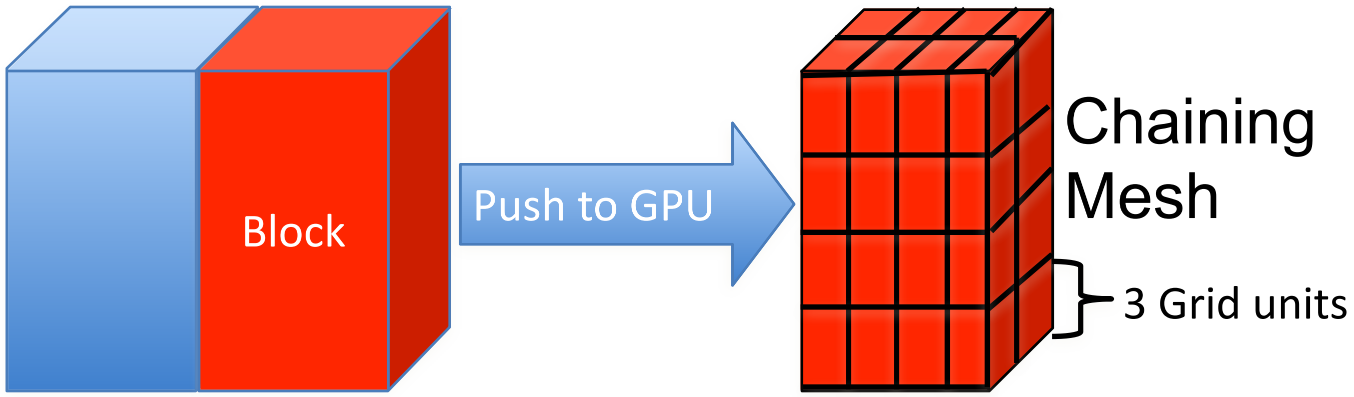
- Memory is primarily host-side (32 GB vs. 6 GB) (against Roadrunner's 16 GB vs. 16 GB), important thing to think about (in case of HACC, the grid/particle balance)
- PCIe is a key bottleneck; overall interconnect B/W does not match Flops (not even close)
- There's no point in 'sharing' work between the CPU and the GPU, performance gains will be minimal -- GPU must dominate
- The only reason to write a code for such a system is if you can truly exploit its power (2 X CPU is a waste of effort!)



Strategies for Success

- It's (still) all about understanding and controlling data motion
- Rethink your code and even approach to the problem
- Isolate hotspots, and design for portability around them (modular programming)
- Like it or not, pragmas will never be the full answer

HACC on Titan: GPU Implementation (Schematic)



P3M Implementation (OpenCL & CUDA)

- 1D-decomposed data pushed to GPU in large blocks; data sub-partitioned into chaining-mesh cubes
- Compute inter-particle forces within cubes and neitghboring cubes
- Large block size ensures computational time far exceeds memory transfer latency
- Natural parallelism provides high performance wrt book-keeping required for tree algorithms

New Implementations/Improvements

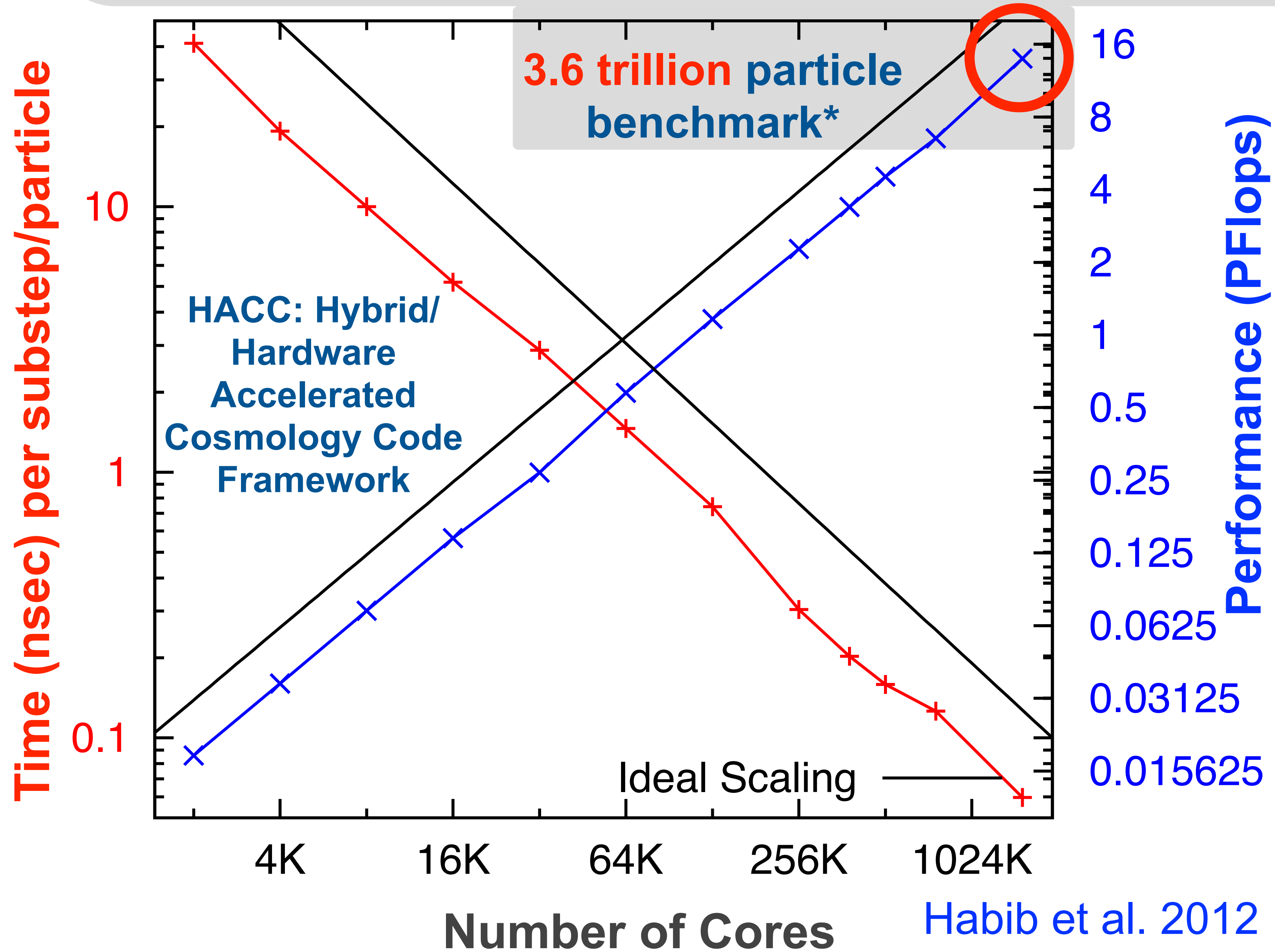
- P3M data-push once every long time-step, with 'soft boundary' chaining mesh, completely eliminates latency
- TreePM analog of BG/Q code written in CUDA also provides high performance
- Each block is an independent work-item; timing the blocks is used in a load-balancing scheme — blocks are transferred to lightly loaded ranks during execution

HACC on the BG/Q ('pre-manycore')

13.94 PFlops, 69.2% peak, 90% parallel efficiency on 1,572,864 cores/MPI ranks, 6.3M-way concurrency

HACC BG/Q Version

- **Algorithms:** FFT-based SPM; PP+RCB Tree
- **Data Locality:** Rank level via 'overloading', at tree-level use the RCB grouping to organize particle memory buffers
- **Build/Walk Minimization:** Reduce tree depth using rank-local trees, shortest hand-over scale, bigger p-p component
- **Force Kernel:** Use polynomial representation (no look-ups); vectorize kernel evaluation; hide instruction latency



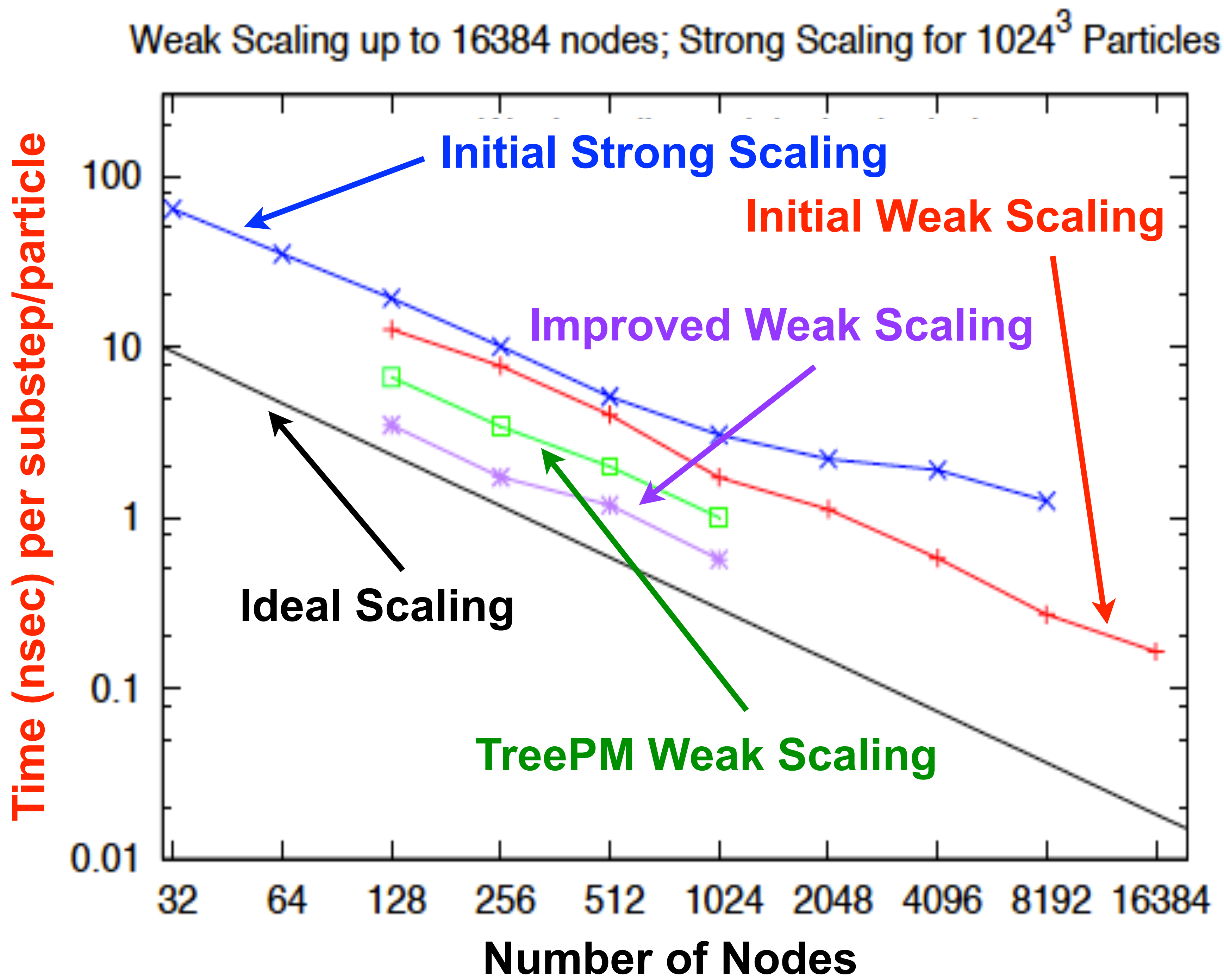
Habib et al. 2012

*largest ever run

HACC weak scaling on the IBM BG/Q (MPI/OpenMP)

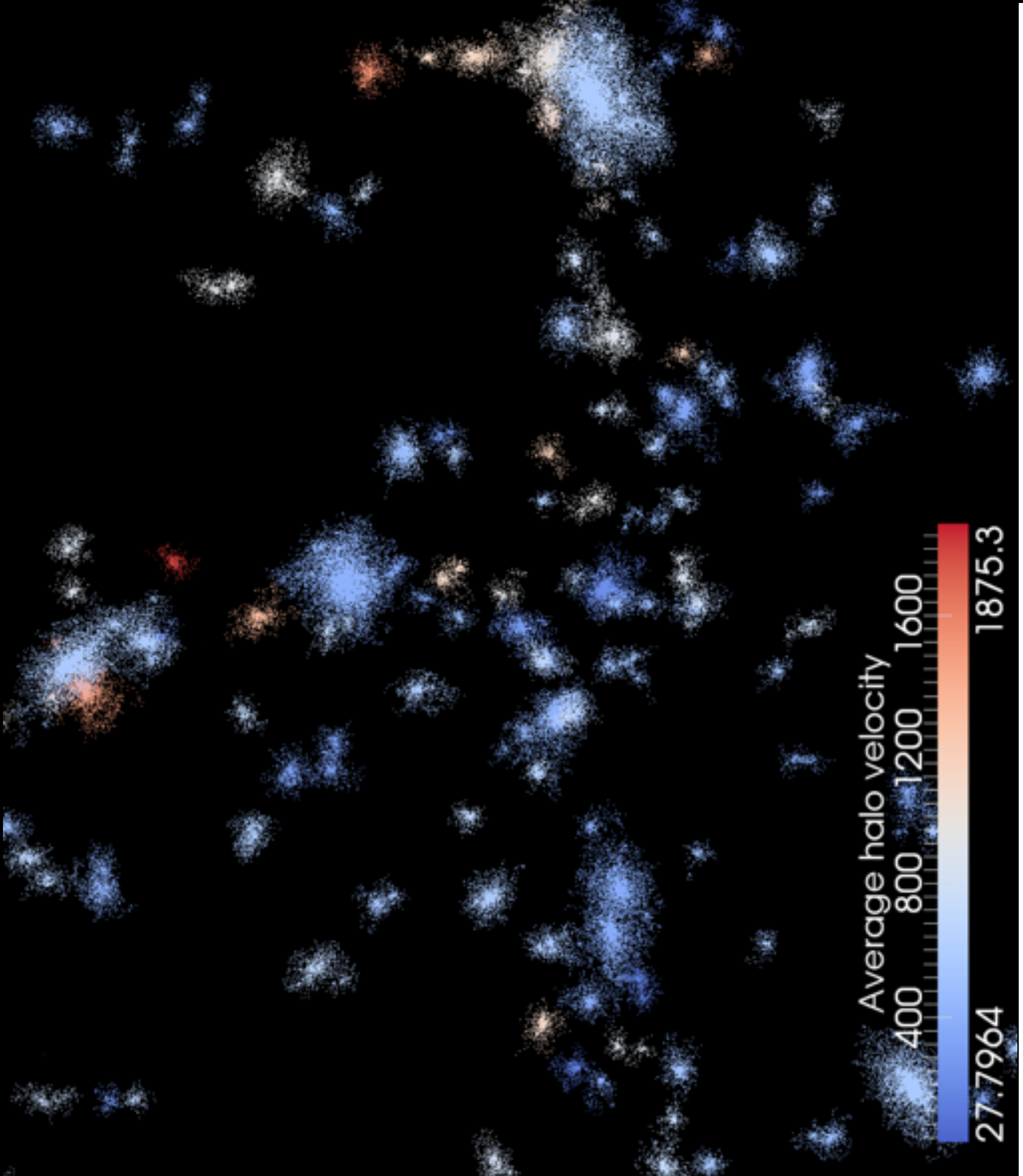
HACC on Titan: GPU Implementation Performance

- P3M kernel runs at 1.6TFlops/node at 40.3% of peak (73% of algorithmic peak)
- TreePM kernel was run on 77% of Titan at 20.54 PFlops at almost identical performance on the card
- Because of less overhead, P3M code is (currently) faster by factor of two in time to solution

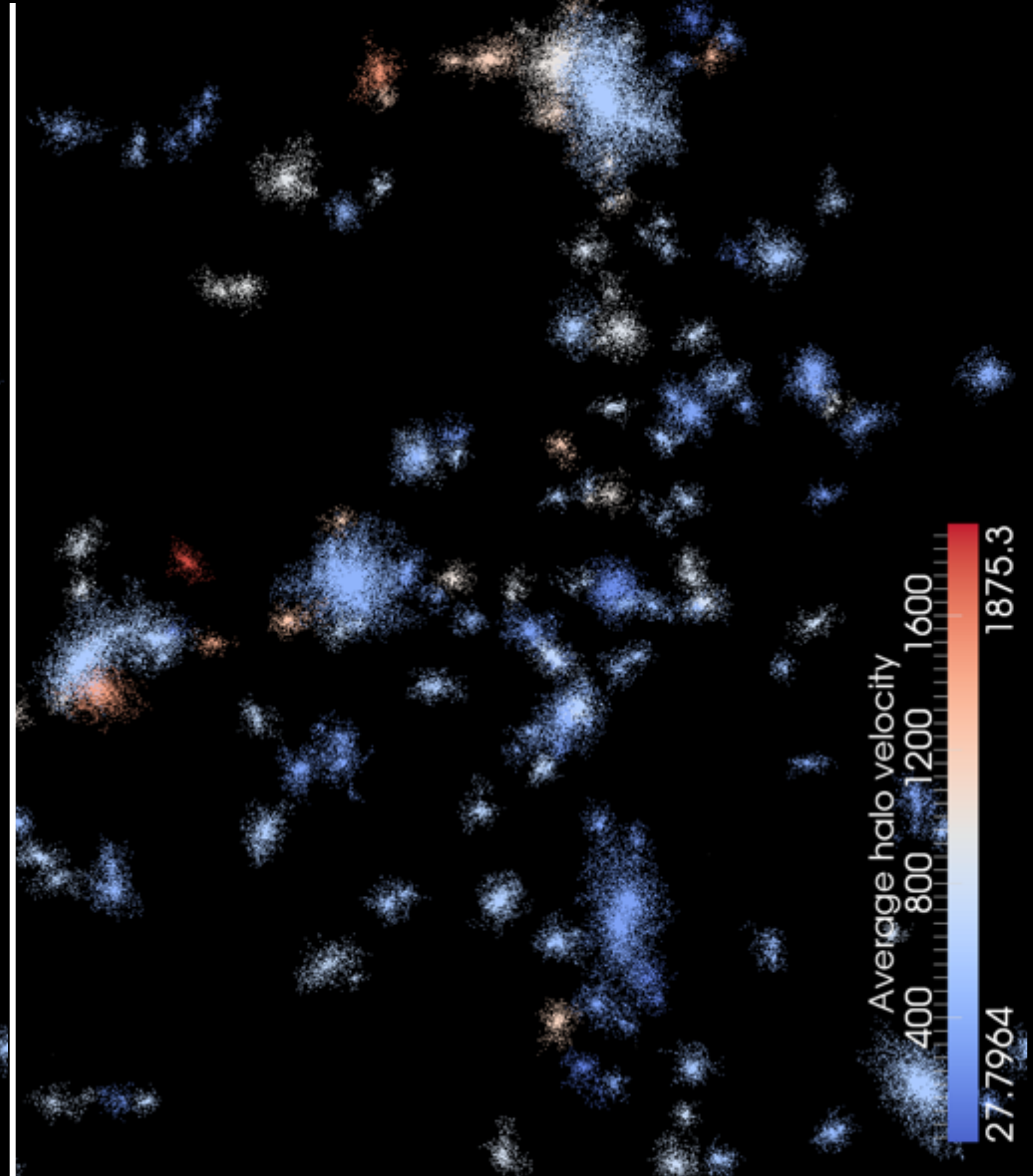


99.2% Parallel Efficiency

MC³ (HACC precursor)



Gadget-2



Snapshot from Code Comparison simulation, ~25 Mpc region; halos with > 200 particles, b=0.15
Differences in runs: P³M vs. TPM, force kernels, time stepper: MC³: a; Gadget-2: log(a)
Power spectra agree at sub-percent level