

# Threads in the HACC cosmology framework

Hal Finkel

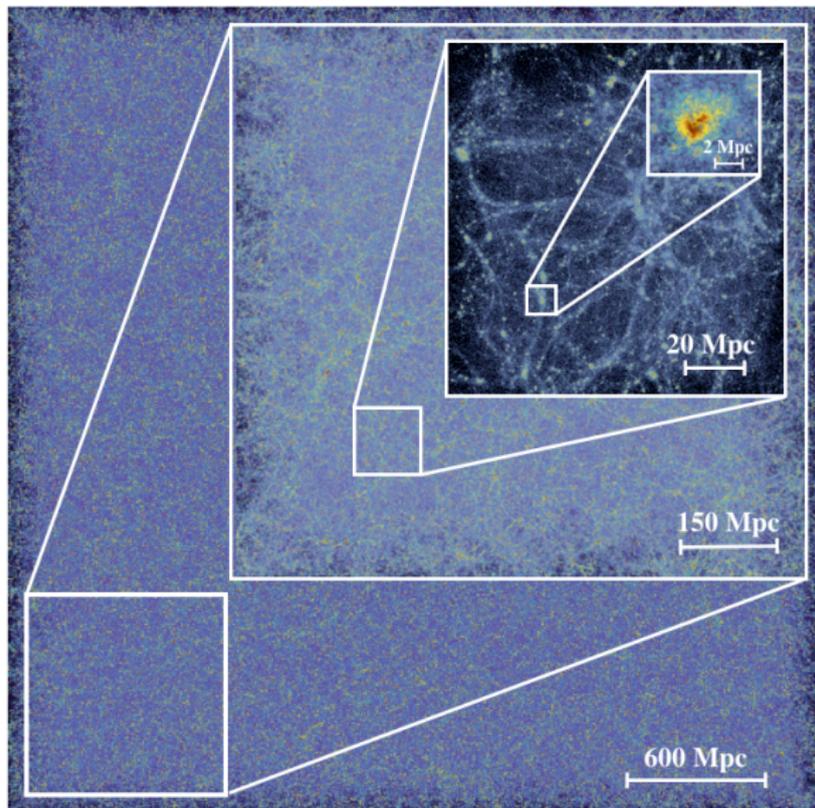
Salman Habib, Katrin Heitmann, Adrian Pope, Vitali Morozov, et al.



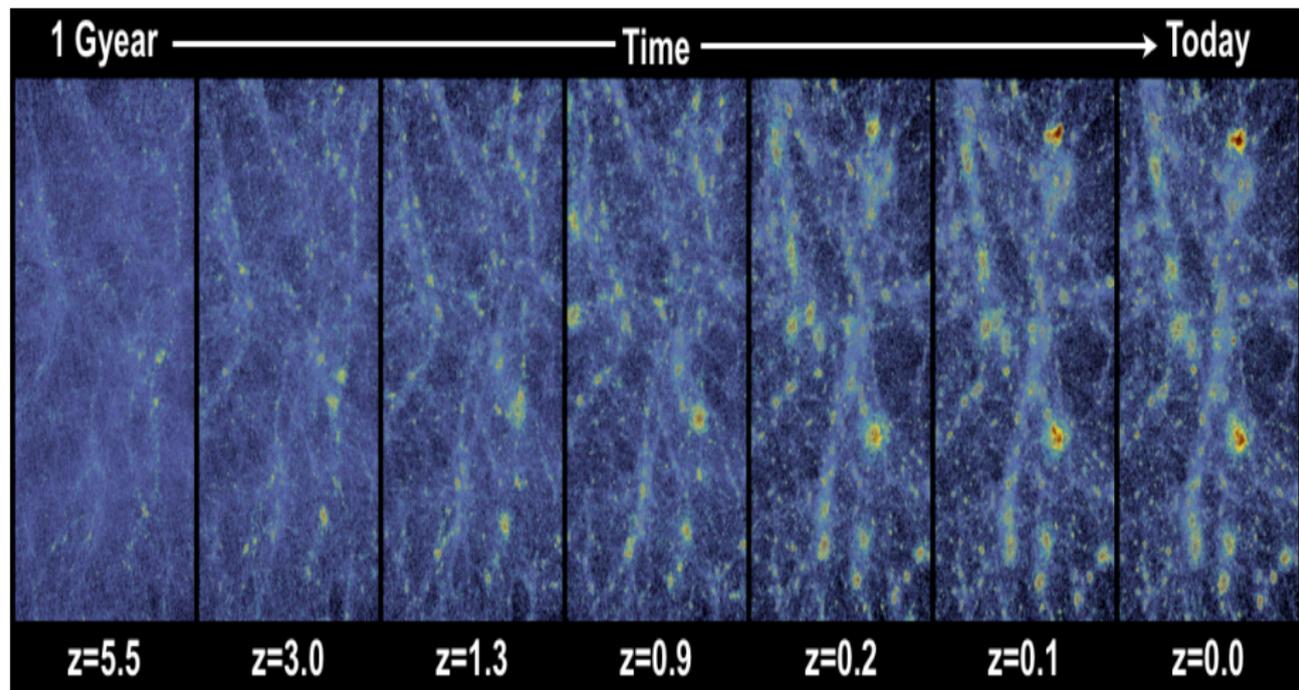
March 6, 2013



# The Big Picture



# Structure Evolution



# Large-Scale Structure Formation

Why study large-scale structure formation?

- Cosmology is the study of the universe on the largest measurable scales.
- Dark matter and dark energy compose over 95% of the mass of the universe.
- The statistics of large-scale structure help us constrain models of dark matter and dark energy.
- Predictions from simulations can be compared to observations from large-scale surveys.
- Observational error bars are now approaching the sub-percent level, and simulations need to match that level of accuracy.

# What We See

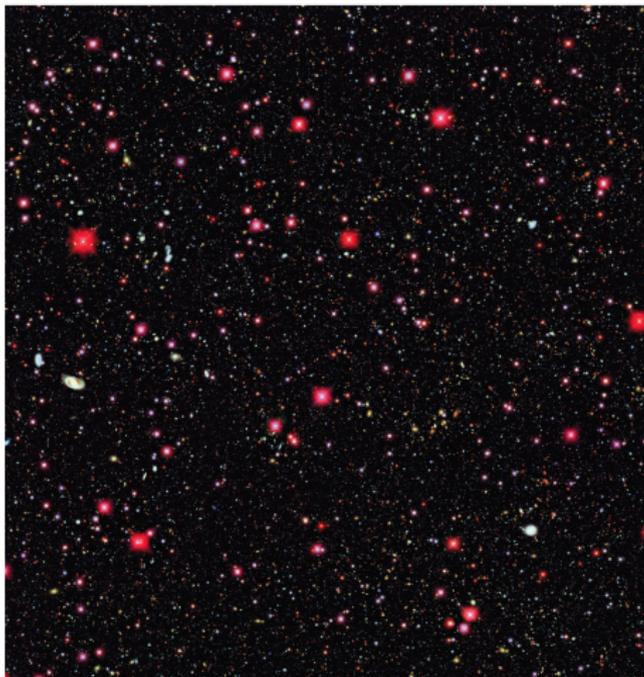
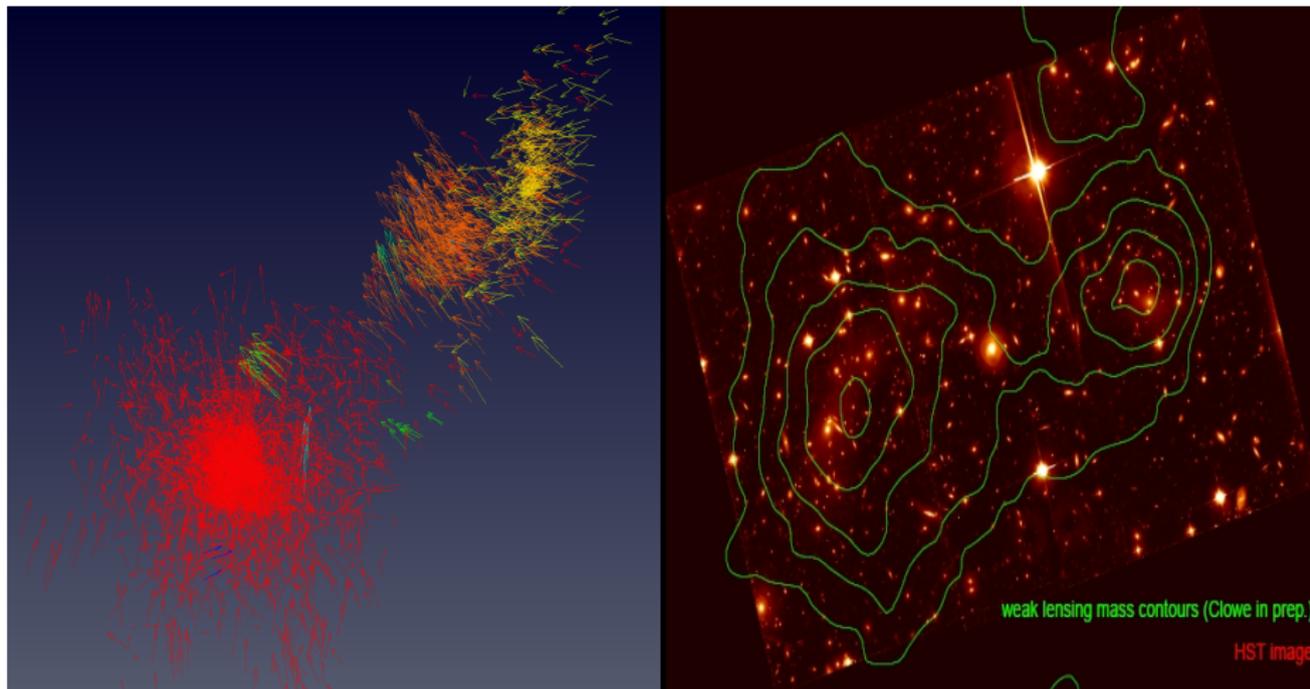


Image of galaxies covering a patch of the sky roughly equivalent to the size of the full moon (from the Deep Lens Survey)

# What We See (cont.)



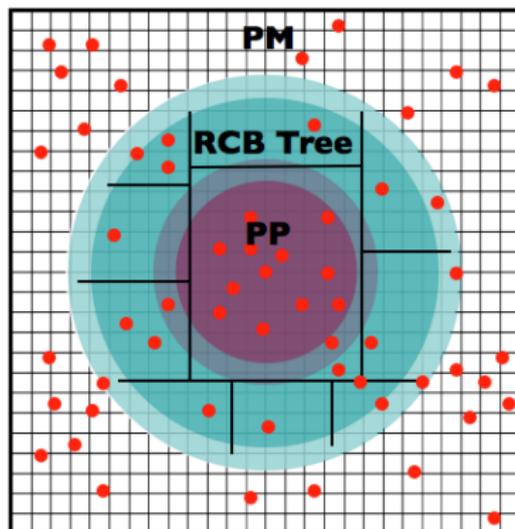
# Requirements

The requirements for our simulation campaigns are driven by the state of the art in observational surveys:

- Survey depths are of order a few Gpc (1 pc=3.26 light-years)
- To follow typical galaxies, halos with a minimum mass of  $\sim 10^{11} M_{\odot}$  ( $M_{\odot}$ =1 solar mass) must be tracked
- To properly resolve these halos, the tracer particle mass should be  $\sim 10^8 M_{\odot}$
- The force resolution should be small compared to the halo size, i.e.,  $\sim$ kpc.

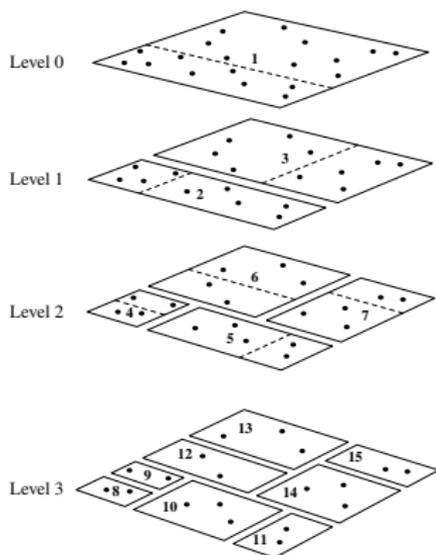
This implies that the dynamic range of the simulation is a part in  $10^6$  ( $\sim$ Gpc/kpc) everywhere! With  $\sim 10^5$  particles in the smallest resolved halos, we need hundreds of billions to trillions of particles.

The HACC (Hybrid/Hardware Accelerated Cosmology Code) Framework meets these requirements using a  $P^3M$  (Particle-Particle Particle-Mesh) algorithm on accelerated systems and a Tree  $P^3M$  method on CPU-only systems (such as the BG/Q).



# RCB Tree

The short-range force is computed using recursive coordinate bisection (RCB) tree in conjunction with a highly-tuned short-range polynomial force kernel.



(graphic from Gafton and Rosswog: arXiv:1108.0028)

## RCB Tree (cont.)

- At each level, the node is split at its center of mass
- During each node split, the particles are partitioned into disjoint adjacent memory buffers
- This partitioning ensures a high degree of cache locality during the remainder of the build and during the force evaluation
- To limit the depth of the tree, each leaf node holds more than one particle. This makes the build faster, but more importantly, trades time in a slow procedure (a “pointer-chasing” tree walk) for a fast procedure (the polynomial force kernel).

# Running Configuration: Fine-Grained Threading

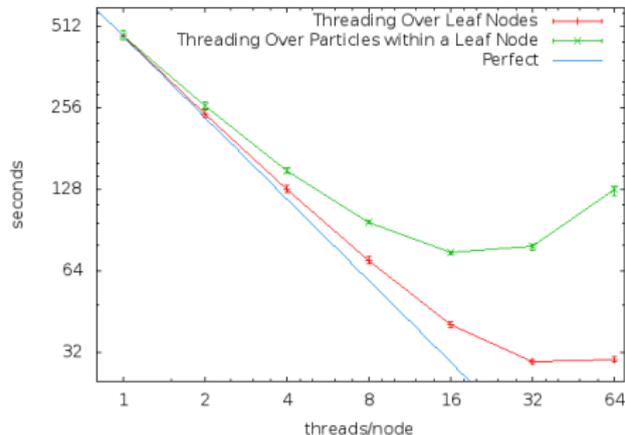
- Using OpenMP, the particles in the leaf node are assigned to different threads: all threads share the interaction list (which automatically balances the computation)
- We use either 8 threads per rank with 8 ranks per node, or 4 threads per rank and 16 ranks per node
- The code spends 80% of the time in the highly optimized force kernel, 10% in the tree walk, and 5% in the FFT, all other operations (tree build, CIC deposit) adding up to another 5%.

This code achieves over 50% of the peak FLOP rate on the BG/Q!

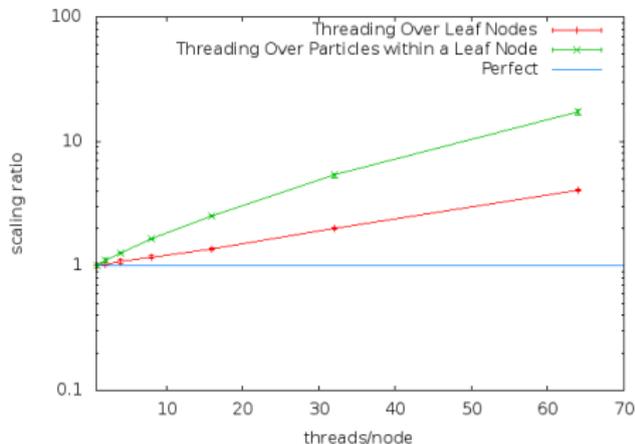
# Running Configuration: Threading over Leaf Nodes

- A work queue is formed of all leaf nodes, and this queue is processed dynamically using all available threads.
- Not limited by the concurrency available in each leaf node (which has only a few hundred particles with a collective interaction list in the thousands).

Step 0 Thread Scaling



Step 0 Thread Scaling



## Balanced Concurrency!

Divide the problem into as many computationally-balanced work units as possible, and distribute those work units among the available threads. These units need to be large enough to cover the thread-startup overhead.

When using OpenMP, don't forget to use dynamic scheduling when the work unit size is only balanced on average:

```
#pragma omp parallel for schedule(dynamic)
for (int i = 0; i < WQS; ++i) {
    WorkQueue[i].execute();
}
```