Mon. Not. R. Astron. Soc. **000**, 1–10 (2011) Printed 10 May 2012 (MN LaTeX style file v2.2)

High Performance P^{3}M N-body code: CUBEP3M:

Joachim Harnois-Déraps[1,2] ⋆, Ue-Li Pen[1] †, Ilian T. Iliev[3] ‡ and Hugh Merz[4] §
[1]*Canadian Institute for Theoretical Astrophysics, University of Toronto, M5S 3H8, Canada*
[2]*Department of Physics, University of Toronto, M5S 1A7, Ontario, Canada*
[3]*Astronomy Centre, Department of Physics and Astronomy, Pevensey II Building, University of Sussex, BN1 9QH, Brighton, United Kingdom*
[4]*Department of XXX*

10 May 2012

ABSTRACT
This paper presents CUBEP3M, an upgraded version of PMFAST, a two-mesh gravity solver that was already among the fastest N-body codes. Among the principal changes, the Poisson solver includes particle-particles interactions at the sub-grid level and across neighbouring cells, plus each level of the volume decomposition is now cubical. Force kernel have been improved for enhanced matching of the two mesh contributions near the cutoff length. We discuss the structure of the code, its accuracy, and its scaling performance. In addition, many utilities have been added, including a runtime halofinder, particle identification tags and non-Gaussian initial conditions generator, and we briefly describe their implementation strategy and accuracy, when applicable .

Key words: N-body simulations — Large scale structure of Universe — Dark matter

1 INTRODUCTION

Many physical and astrophysical systems are subject to non-linear dynamics and rely on N-body simulations to describe the evolution of bodies. One of the main field of application is the modelling of large scale structures, which are driven by the sole force of gravity. Recent observations of the cosmic microwave background (**?**), of galaxy clustering (York et al. 2000; Colless et al. 2003; Schlegel et al. 2009; Drinkwater et al. 2010) of weak gravitational lensing (**??**) and of supernovae redshift-distance relations all point towards a standard model of cosmology, in which dark energy and collisionless dark matter occupy more than 95 per cent of the total energy density of the universe. In such a paradigm, pure N-body code are perfectly suited to describe the dynamics, as long as baryonic physics is not very important, or at least we understand how the baryonic fluid feeds back on the dark matter structure. The next generation of measurements aim at constraining the cosmological parameters at the per cent level, and the theoretical understanding of the non-linear dynamics that govern structure formation heavily relies on numerical simulations.

For instance, a measurement of the baryonic acoustic oscillation (BAO) dilation scale can provide tight constraints on the dark energy equation of scale (**??**). The most optimal estimates of the uncertainty requires the knowledge of the matter power spectrum covariance matrix, which is only accurate when measured from a large sample of N-body simulations (Rimes & Hamilton 2005; Takahashi et al. 2009, 2011). For the same reasons, the most accurate estimates of weak gravitational lensing signal is obtained by propagating photons in past light cones that are extracted from simulated density fields (Vale & White 2003; Sato et al. 2009; Hilbert et al. 2009). Another area where large-scale N-body simulations have in recent years been instrumental are in simulations of early cosmic structures and reionization (e.g. Iliev et al. 2006; Zahn et al. 2007; Trac & Cen 2007; Iliev et al. 2011). The reionization process is primarily driven by low-mass galaxies, which for both observational and theoretical reasons, need to be resolved in fairly large volumes, which demands simulations with a very large dynamic range.

The basic problem that is addressed with N-body codes is a time evolution of an ensemble of N particles that is subject to gravitational attraction. The brute force calculation requires $O(N^2)$ operation, a cost that exceeds the memory and speed of current machines for large problems. Solving the problem on a mesh (Hockney & Eastwood 1981) reduces to $O(N\log N)$ the number of operations, as it is possible to solve for the particle-mesh (PM) interaction with fast Fourier transforms techniques, typically using high performance libraries such as FFTW (Frigo & Johnson 2005).

With the advent of large computing facilities, parallel coding has now become common practice, and N-body codes have evolved both in performance and complexity. Many codes have opted for a tree algorithm, including Gadget (**??**), PMT (Xu 1995), GOTPM (Dubinski et al. 2004), and Hydra (**?**), in which the local resolution increases with the density of the matter field. These have the advantage to balance the load across the computing units, which enable the calculation of high density regions. The drawback is a significant loss in speed, which can be only partly recovered by turning off the tree algorithm. The same reasoning applies to the

⋆ E-mail: jharno@cita.utoronto.ca
† E-mail: pen@cita.utoronto.ca
‡ E-mail: i.t.iliev@sussex.ac.uk
§ E-mail: merz@sharcnet.ca

mesh-refine code of Couchman (1991) and the PM code by **?**. these are not designed to perform large scale PM calculations.

(PM code by japanese? Others codes I should mention? That of Hy Trac?)

PMFAST (Merz et al. (2005), MPT hereafter) is one of the first code that is designed such as to optimize the PM algorithm, both in terms of speed and memory usage. It uses a two-level mesh algorithm based on the gravity solver of Trac & Pen (2003), The long range gravitational force is computed on a grid four times coarser, such as to minimize the MPI communication time and to fit in system's memory. The short range is computed locally on a finer mesh, and only the local sub-volume needs to be stored at a given time, allowing for OPENMP parallel computation. This this setup enable the code to evolve large cosmological systems both rapidly and accurately, on relatively modest clusters. One of the main advantage of PMFAST over other PM codes is that the number of large arrays is minimized, and the global MPI communications are cut down to the minimum: for passing particle at the beginning of each time step, and for computing the long range FFTs. As described in MPT, access to particles is accelerated with the use of linked lists, deletion of 'ghost' particles in buffer zones is done at the same time as particles are passed to adjacent nodes, and the global FFTs are performed with a slab decomposition of the volumes via a special file transfer interface, designed specifically to preserve a high processor load.

Since its first published version, PMFAST has evolved in many aspects. The first major improvement was to transform the volume decomposition in multi-node configurations from slabs to cubes. One of the problem with slabs is that they do not scale well to large runs: as the number of cells per dimension increases, the thickness of each slab shrinks rapidly, until it reaches the hard limit of a single cell layer. With this enhancement, the code name was changed to CUBEPM, which soon after incorporated particle-particle (pp) interactions at the sub-grid level. The code was finally renamed CUBEP3M, and now includes a significant number of new features: the pp force can be extended to an arbitrary range, the size of the redshift jump can be constrained for improved accuracy during the first time steps, a runtime halo finder has been implemented, the expansion has also been generalized to include a redshift dependent equation of state of dark energy, there is a system of unique particle identification which can be switched on or off, and the initial condition generator has been generalized as to include non-Gaussian features **(anything else since PMFAST?)**.

This paper aims at presenting and quantifying these new features that make CUBEP3M one of the most competitive public N-body code. It has already been involved in a number of scientific applications over the last few years, spanning the field of weak lensing (Vafaei et al. 2010; Lu & Pen 2008; Doré et al. 2009; Lu et al. 2010; Yu et al. 2010; **?**), BAO (Zhang et al. 2010; Ngan et al. 2011; Harnois-Deraps & Pen 2011; **?**), formation of early cosmic structures (Iliev et al. 2008, 2010), observations of dark stars (Zackrisson et al. 2010; Ilie et al. 2012) and reionization (Iliev et al. 2011; Fernandez et al. 2011; Friedrich et al. 2011; Mao et al. 2012; Datta et al. 2011; Friedrich et al. 2012), and it seems relevant for the community to have access to a paper that describes the methodology, the accuracy and the performance of this public code.

Since CUBEP3M is not a new code, the accuracy and systematic tests were performed by different groups, on different machines, and with different configurations. It is not an ideal situation in which to quantify the performance, and each test must be viewed as a separate measurement that quantifies a specific aspect of the code. We have tried to keep to a minimum the number of such different runs, and although the detailed numbers might vary across different runs, the general trends are common to all of them.

The paper is structured as follow: XXXX

2 REVIEW OF THE CODE STRUCTURE

An optimal large scale N-body code must address many challenges: minimize the memory footprint to allow larger dynamical range, minimize the passing of information across computing nodes, reduce and accelerate the memory accesses to the large scale arrays, make efficient use of high performance libraries to speed up standard calculations like Fourier transforms, just to name a few. In the realm of parallel programming, high efficiency can be assessed when a high load is balanced across all processors most of the time. In this section, we present the general strategies adopted to address these challenges[1]. We start with a walkthrough the code flow, and briefly discuss some specific sections that depart from standard N-body codes, while referring the reader to future sections for detailed discussions on selected topics.

As mentioned in the Introduction section, CUBEP3M is a FORTRAN90 N-body code that solves Poisson's equation on a two-level mesh, with sub-cell accuracy thanks to particle-particle interactions. The code has extensions that departs from this basic scheme, and we shall come back to these later, but for the moment, we adopt the standard configuration. The long range component of the gravity force is solved on the coarse grid, and is global in the sense that the calculations require knowledge about the full simulated volume. The short range force and the particle-particle interactions are computed in parallel on local volumes[2] or *tiles*, and for maximal efficiency, the number of tiles per node should be at least equal to the number of available processors. Given the available freedom in the parallel configuration, it is generally good practice to maximize the number of OPENMP threads and minimize the number of MPI processes, as the information exchange between cores that are part of the same motherboard is generally much faster. As mentioned in MPT, the computation of the short range force requires each tile to store the fine grid density in a buffer surface around the physical volume it is assigned. The thickness of this surface must be larger than the cutoff length, and we find that a 24 cells deep buffer is a good compromise between memory usage and accuracy **(Hugh, is that correct?)**

Because the coarse grid arrays require 4^3 times less memory per node, the bulk of the foot-print is concentrated in a handful of fine grid arrays. In addition, some of these are required for intermediate steps of the calculations only, hence it is possible to hide some of the coarse grid arrays[3]. We present here the largest arrays used by the code:

(i) `xv` stores the position and velocity of each particle
(ii) `ll` stores the linked-list that accelerate the access to particles in a local domain
(iii) `rho_f` and `cmplx_rho_f` store the local fine grid density in real and Fourier space respectively

[1] Many originate directly from MPT and were preserved in CUBEP3M; those will be briefly mentioned, and we shall refer the reader to the original PMFAST paper for greater details.
[2] To make this possible, the fine grid arrays are constructed such as to support parallel reading and writing. In practice, this is done by adding an additional dimension to the relevant arrays, such that each CPU accesses a unique memory location.
[3] This memory recycling is done with 'equivalence' statements in F90

```
program cubep3m
   call initialize
   do
       call timestep
       call particle_mesh
       if(checkpoint_step) then
          call checkpoint
       elseif(last_step)
          exit
       endif
   enddo
   call finalize
end program cubep3m
```

Figure 1. Overall structure of the code.

(iv) `force_f` stores the force of gravity (short range only) along the three Cartesian directions
(v) `kern_f` stores the fine grid force kernel in the three directions

The code flow is presented in Fig. 1 and 2. Before entering the main loop, the code starts with an initialization stage, in which many declared variables are assigned default values, the redshift checkpoints are read, the FFTW plans are created, and the MPI communicators are defined. The phase-space array is obtained from the output of the initial conditions generator, and the force kernels on both grids are constructed from the specific geometry of the simulation. For clarity, all these operations are collected under the subroutine call `initialize` in Fig. 1, although they are actually distinct calls in the code.

Each iteration of the main loop starts with the `timestep` subroutine, which proceeds to a determination of the redshift jump by comparing the step size constraints from each force components and from the scale factor. The cosmic expansion is found by Taylor expanding Friedmann's equation up to the third order in the scale factor, and can accommodate constant or running equation of state of dark energy. The force of gravity is then solved in the `particle_mesh` subroutine, which first updates the positions and velocities of the dark matter particles, exchange with neighbouring nodes those that have exited to volume, creates a new linked list, then solve Poisson's equation. This subroutine is conceptually identical to that of PMFAST, with the exception that CUBEP3M decomposes the volume into cubes (as opposed to slabs). The loop over tiles and the particle exchange are thus performed in three dimensions. **(Should I mention leapfrog here or anywhere else?)** When the short range and pp forces have been calculated on all tiles, the code exits the parallel OPENMP loop and proceeds to the long range. This section of the code is also parallelized in many occasions, but, unfortunately, the current MPI-FFTW do not allow multi-threading. There is thus an inevitable loss of efficiency during each global Fourier transforms, during which only the head processor is active. Other libraries such as P3DFFT (**?**) or Intel MKL-α (**?**) currently permit this extra level of parallelization, and it is our plan to migrate to one of these in the near future.

If the current redshift corresponds to one of the checkpoints, the code advances all particles to their final location and writes them to file. Similarly, the code can compute two-dimensional projections of the density field, halo catalogues (see section 7 for details), and can compute the power spectrum on the coarse grid at run time. The code exits the loop when it has reached the final redshift, it then wraps up the FFTW plans and clears the MPI communicators.

```
subroutine particle_mesh
   call update_position
   call link_list
   call particle_pass
   !$omp parallel do
   do tile = 1, tiles_node
      call rho_f_ngp
      call cmplx_rho_f
      call kernel_multiply_f
      call force_f
      call update_velocity_f
      if(pp = .true.) then
         call link_list_pp
         call force_pp
         call update_velocity_pp
         if(extended_pp = .true.) then
            call link_list_pp_extended
            call force_pp_extended
            call update_velocity_pp_extended
         endif
      endif
   end do
   !$omp end parallel do
   call rho_c_ngp
   call cmplx_rho_c
   call kernel_multiply_c
   call force_c
   call update_velocity_c
   delete_buffers
end subroutine particle_mesh
```

Figure 2. Overall structure of the two-level mesh algorithm. We have included the section that concerns the extended pp force calculation to show that it follows the same basic logic. We mention here that the three linked list arrays are distinct entities, and their structure differ slightly.

We have collected those operations under the subroutine `finalize` for concision.

(Anything else to say in this section?)

3 POISSON SOLVER

This section reviews how Poisson's equation is solved on a double-mesh configuration. Many parts of the algorithm are identical to PMFAST, hence we refer the reader to section 2 of MPT for more details. In CUBEP3M, the mass default assignment scheme are a 'cloud-in-cell' (cic) interpolation for the coarse grid, and a 'nearest-grid-point' interpolation for the fine grid (Hockney & Eastwood 1981). Particle-particle interactions are not interpolated, but a sharp cutoff kills the force for pairs closer to a tenth of a grid cell.

The code units inherit from (Trac & Pen 2004), which we summarized here for completeness. In simulation units, the mean comoving mass density is set to unity, the length of a fine grid cell is set to one, and the Newton gravitational constant G is set to $1/(6\pi a)$, where a is the scale factor. With these three choices, one can transform any physical units into simulation units and vice versa.

The force of gravity on a mesh can be computed either with a gravitational potential kernel $\omega_\phi(\mathbf{x})$ or a force kernel $\omega_F(\mathbf{x})$. Gravity fields are curl-free, which allows us to relate the potential $\phi(\mathbf{x})$ to the source term via Poisson's equation:

$$\nabla^2\phi(\mathbf{x}) = 4\pi G\rho(\mathbf{x}) \tag{1}$$

G being Newton's constant. We solve this equation in Fourier space, where we write

$$\tilde{\phi}(\mathbf{k}) = \frac{4\pi G\tilde{\rho}(\mathbf{k})}{-k^2} \equiv \tilde{\omega}_\phi(\mathbf{k})\tilde{\rho}(\mathbf{k}) \tag{2}$$

The potential in real space is then obtained with an inverse Fourier transform, and the kernel becomes $\omega_\phi(\mathbf{x}) = -G/r$. Using the convolution theorem, we can write

$$\phi(\mathbf{x}) = \int \rho(\mathbf{x}')\omega_\phi(\mathbf{x}' - \mathbf{x})d\mathbf{x}' \tag{3}$$

Although this approach is fast, it involves a finite differentiation which enhances the numerical noise. We therefore opt for a force kernel, which is more accurate but has the inconvenient to require four extra Fourier transforms. In this case, we must solve the convolution in three dimensions:

$$F(\mathbf{x}) = -m\nabla\phi(\mathbf{x}) = \int \rho(\mathbf{x}')\omega_F(\mathbf{x}' - \mathbf{x})d\mathbf{x}' \tag{4}$$

The differentiation does not affect the density since it only acts on un-prime variables, and the force kernel is given by

$$\omega_F(\mathbf{x}) \equiv -\nabla\omega_\phi(\mathbf{x}) = -\frac{mG\hat{\mathbf{r}}}{r^2} \tag{5}$$

Following the spherically symmetric matching technique of MPT (section 2.1), we split the force kernel into two components, for the short and long range respectively, and match the overlapping region with a polynomial. Namely, we have:

$$\omega_s(r) = \begin{cases} \omega_F(r) - \boldsymbol{\beta}(r) & \text{if } r \leqslant r_c \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

and

$$\omega_l(r) = \begin{cases} \boldsymbol{\beta}(r) & \text{if } r \leqslant r_c \\ \omega_F(r) & \text{otherwise} \end{cases} \tag{7}$$

The vector $\boldsymbol{\beta}(r)$ is related to the fourth order polynomial that is used in the potential case by $\boldsymbol{\beta} = -\nabla\alpha(r)$. The coefficients are found by matching the boundary conditions at r_c up to the second derivative, and we get

$$\boldsymbol{\beta}(r) = \left[-\frac{7r}{4r_c^3} + \frac{3r^3}{4r^5} \right]\hat{\mathbf{r}} \tag{8}$$

Because these calculations are performed on two grids of different resolution, a sampling window function must be convoluted both with the density and the kernel (see [Eq. 7-8] of MPT). When matching the two force kernels, it was realized that close to the cutoff region, the long range force is always on the low side, whereas the short range force is scattered across the theoretical $1/r^2$ value. These behaviors are purely features of the CIC and NGP interpolation scheme respectively. We identified a small range surrounding the cutoff length, in which we empirically adjust both kernels such as to improve the match. Namely, for $14 \leqslant r \leqslant 16$, $\omega_s(r) \rightarrow \omega_s(r) * 0.985$, and for $12 \leqslant r \leqslant 16$, $\omega_l(r) \rightarrow \omega_l(r) * 1.2$. The two fudge factors were found by performing force measurements on two particles randomly placed in the volume.

Finally, a choice must be done concerning the longest range of the force. Gravity can be either a) an accurate $1/r^2$ force, as far as the volume allows, or b) modified to correctly match the periodicity of the boundary conditions. By default, the code is configured along to the second choice, which accurately models the growth of structure at very large scales. However, detailed studies of gravitational collapse would benefit from the first settings. **(Hugh, are the above two paragraphs correct?)**

Least square kernel?

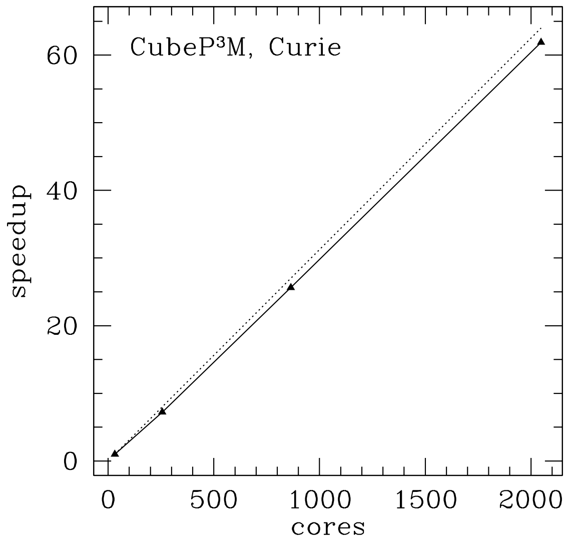
4 SCALING PERFORMANCES

The parallel algorithm of CUBEP^{3}M is designed for "weak" scaling, i.e. if the number of cores and the problem size increase in proportion to each other, then for ideal scaling the wall-clock time should remain the same, in contrast to "strong" scaling, whereby the same problem solved on more cores should take proportionately less wall-clock time. This weak scaling requirement is dictated by the problems we are typically aiming towards (very large and computationally-intensive) and our goals, which are to address such large problems in the most efficient way, rather than for the least wall-clock time. Furthermore, there is no explicit load balancing, thus the code is most efficient if the sub-domains contain roughly equal number of particles, which is true for most cosmological-size volumes, but not for e.g. simulations of a single highly-resolved galaxy.

The scaling was tested with a dedicated series of simulations with increasing size and number of cores on the 'fat" (i.e. large-memory) nodes of the supercomputers Curie at TGCC in France. Our results are shown in Fig. 3 and in Table 1. For appropriate direct comparison, all simulations were performed using the same particle mass ($M_{\rm particle} = 1.07 \times 10^{11} M_\odot$) and force resolution (softening length 50 h^{-1}kpc). The box sizes used range from 256 h^{-1}Mpc to 2048 h^{-1}Mpc, and the number of particles from 256^3 to 2048^3. Simulations were run on 32 up to 2048 computing cores starting from redshift $z = 100$, and evolving until $z = 0$. Our results show excellent scaling, within $\sim$ 3% of the ideal one, for up to 2048 cores.

We have also ran CUBEP^{3}M on a much larger number of cores, from 8000 to up to 21,976, with 5488^3-6000^3 (165 to 216 billion) particles on Ranger at the Texas Advanced Computing Centre (Sun-Blade x6420 with AMD x86_64 Opteron Quad Core 2300 MHz (9.2 GFlops) "Barcelona" processors and Infiniband networking, a total of 62,976 computing cores and 125,952 GB of total memory. Its shared-memory nodes consist of 4 Quad Core processors and 32 GB of RAM.) and JUROPA at Jülich Supercomputing Centre in Germany (Intel Xeon X5570 (Nehalem-EP) quad-core, 2.93 GHz, Infiniband networking, a total of 17,664 computing cores and 52 TB of total memory. Its shared-memory nodes consist of 2 Quad Core processors and 24 GB of RAM). Since it is not practical to perform dedicated scaling tests on such large number computing cores, we instead use data extracted from production runs, as listed in Table 2. We have found the code to scale quite well, within 1.5% of ideal up to 4,096 cores. For very large number of cores (10,976) the scaling is slightly worse, but still within $\sim$ 20% of ideal, due to increased communication costs, I/O overheads (a single timeslice of 5488^3 particles is 3.6 TB) and load balancing issues. These first three Ranger runs were performed with 4 MPI processes and 4 threads per Ranger node ('4way') (using the specially-provided *tacc_affinity* NUMA script to bind the memory usage to local core, thus ensuring memory affinity. This is important because the 32 GB RAM/node on Ranger (and other computers with multi-processor sockets) are actually not equal-access and the local 8 GB memory of each processor has much shorter access time).

Furthermore, due to the very strong clustering of structures at those small scales, some of the cuboid sub-domains came to contain well above the average number of particles, thereby requiring more memory per MPI process in order to run. As a consequence, throughout most of the evolution the largest two of these simulations were run with 4096 and 21,952 cores and with only 2 MPI processes and 8 threads per Ranger node (2way), which on Ranger allows using up to 16 GB of RAM per MPI process.



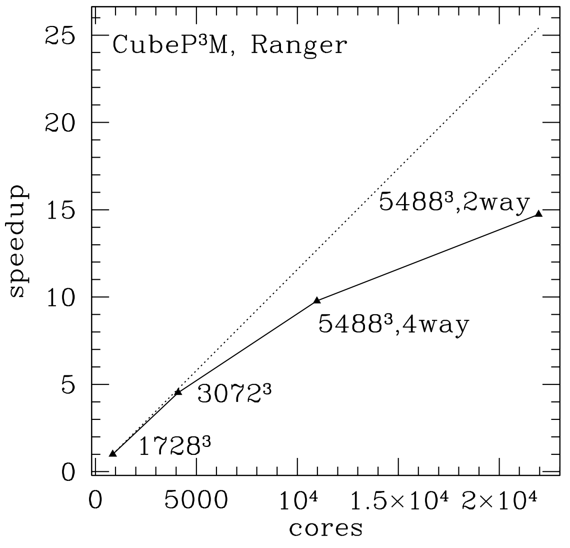


Figure 3. Scaling of CUBEP3M on Curie fat nodes (left) and on Ranger TACC facility for very large number of cores (right). Plotted is the code speedup ($N^3_{\rm particles}/t_{\rm wallclock}$) against core count, normalized by the smallest run in each case. Dashed line indicates the ideal weak scaling. The data is listed in Table 1.

Table 1. Scaling of CUBEP3M on Curie. Speedup is scaled to the smallest run.

number of cores	speedup	ideal speedup	absolute timing (min)	$N_{\rm particles}$	box size (h^{-1}Mpc)
32	1.00	-	3.2	256^3	256
256	7.21	8	3.55	512^3	512
864	25.63	27	4.8	864^3	864
2048	61.87	64	26.48	2048^3	2048

Table 2. Scaling of CUBEP3M for large number of cores (on Ranger). Speedup is scaled to the smallest run.

number of cores	speedup	ideal speedup	absolute timing (min)	$N_{\rm particles}$	box size (h^{-1}Mpc)
864	1.00	-	258	1728^3	6.3
4096	4.53	4.74	320	3072^3	11.4
10976	9.78	12.7	845	5488^3	20
21952	14.73	25.4	561	5488^3	20

In order to insure local memory affinity a special NUMA control script (which we called *tacc_affinity_2way*) was developed for us by TACC staff and was used by us to run more efficiently in this mode. However, this not-fully-local memory access does affect the performance somewhat, as does the imperfect load balancing in such situations, as seen by the rightmost point on the scaling plot. The scaling in this case is 42% below the ideal, although we note that we still get ~ 1.5 speedup from doubling the core count, even given these issues. Overall the code scaling performance is thus quite satisfactory and it clearly scales very well to extremely large number of cores and can therefore do very large problems efficiently, utilizing the next generation Petascale systems.

Finally, we note that several special fixes had to be developed by TACC and JUROPA staff in order for our largest runs to work properly and to be able to use the software libraries we required (MPICH and FFTW), as those latter exhibited serious problems when applied to runs of such unprecedented size.

Real time versus size, bottle neck, description of the largest runs, etc. **(Hugh, do you have anything you would like to share concerning the largest runs you did on Blue Gene?)**

5 ACCURACY

One of the most reliable way to assess the simulation's accuracy at evolving particles is to measure the density power spectrum at late time, and compare to non-linear prediction. For an over-density field $\delta(x)$, the power spectrum is extracted from the two point function in Fourier space as:

$$\langle|\delta(k)\delta(k')|\rangle = (2\pi)^3 P(k)\delta_D(k'-k) \qquad (9)$$

where the angle bracket corresponds to an ensemble (or volume) average. We plot in Fig. 4 the dimensionless power spectrum for a large run (4000^3 particles), which are evolved from a Gaussian initial distribution at $z = 100$ down to $z = 0$, in a box $3.2h^{-1}$Gpc per side. We observe that the agreement with the non-linear prediction of **?** is at the per cent level over a large dynamical range. The drop of power at high-k is caused by the finite resolution, and the fluctuations at low-k is caused by the white noise imposed in the initial conditions.

(Show plots of pairwise force (transverse and radial), density force, mention the work on covariance matrix by Ngan, Harnoisetal.)

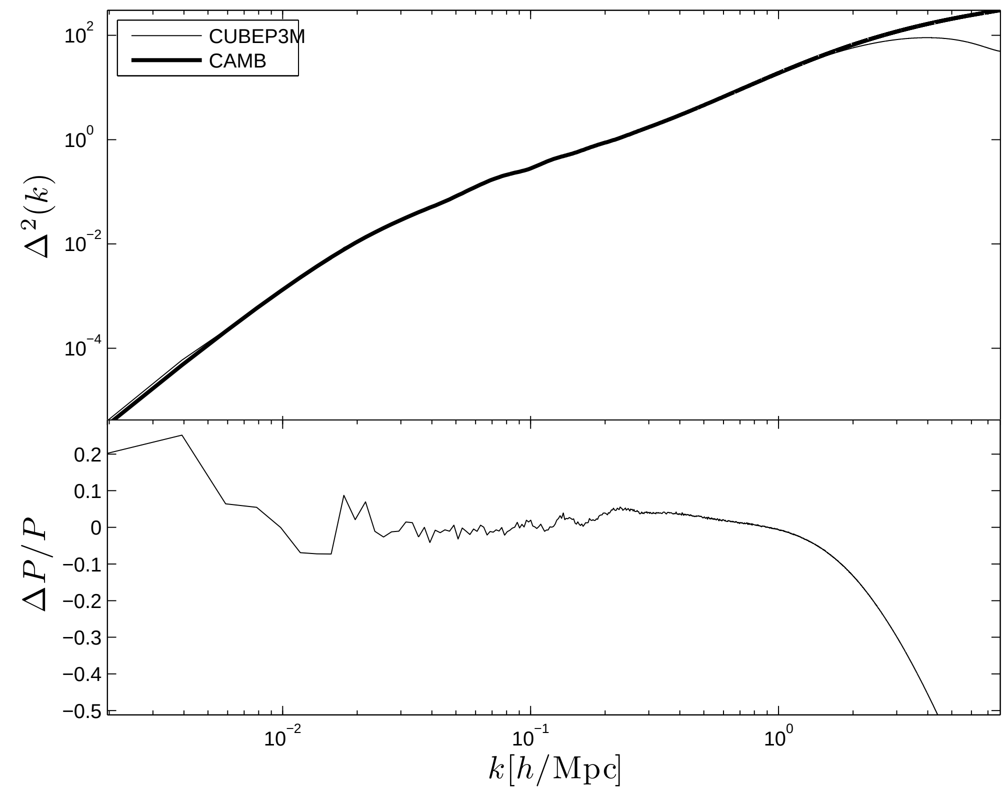


Figure 4. Dark matter power spectrum, measured at $z = 0$ in a volume $3.2h^{-1}$Gpc per side, from 4000^3 particles. The simulation ran on 4000 cores on Ranger.

6 SYSTEMATICS

The biggest problem with the pp force calculation is that it is anisotropic and depends on the location of the fine mesh with respect to the particles. An example are two particles on either side of a grid cell boundary. These particles will experience the 1-grid cell separation NGP mesh force, but if the mesh were shifted such that they were within the same cell, they would experience the (larger) pp force instead. This effect is especially pronounced at the early stages of the simulation where the density is more homogeneous, and leads to mesh artefacts appearing in the density field. In order to minimize this systematic effect, we randomly shift the particle distribution relative to the mesh by a small amount – up to 2 fine grid cells in magnitude – in each dimension and at each time step. This adds negligible computational overhead as it can be applied during the particle position update and suppresses the mesh behaviour over multiple time steps. It is possible to shift back the particles at the end of each time steps, which prevents a random drift of the whole population. This is convenient if one needs to correlate the initial and final positions of the particles.

We ensure that, on average, this solution balances out the mesh feature, by tuning the NGP force kernel such as to provide force as unbalanced as possible at grid cell distances. This adjustment is performed from the pairwise force test described in section 5. We note that this is one of the driving argument to extend the pp force outside the fine mesh cell, since the scattering of the NGP force about the actual $1/r^2$ law drops rapidly as the distance increases. As discussed in section 8.3, this gain in accuracy comes at a price, a choice that must be carefully balanced.

At early stages of the simulation, the density fields is rather homogenous, causing the force of gravity to be rather weak everywhere. In that case, the size of the redshift jumps is controlled by a limit in the cosmological expansion. If the expansion jump is too large, the size of the residual errors can become significant, and one can observe, for instance, a growth of structure that does not match the predictions of linear theory even at the largest scales. One therefore needs to choose a maximum step size. This is controlled by r_{max}, which is the fractional step size, $\mathrm{d}a/(a+\mathrm{d}a)$ and is set to 0.05 by default. It is possible to reduce this number and improve significantly the accuracy of the code, at the cost of increasing the total number of time steps. Fig. 5 shows how r_{max} affects the code. It is a comparison of late time power spectra of density fields that originate from the same initial conditions, and used the same random seeds to control the fine mesh shift (mentioned above). We observe that XXXXX. The CPU resources required to run these simulations increase rapidly as r_{max} decreases, as seen in Table 3.

Figure 5. Dark matter power spectrum, measured at $z = 0$ in a volume $XXXh^{-1}$Mpc per side, from 256^3 particles and a starting redshift of $z = 200$. The different curves show different values of r_{max}. The resources required to run these simulations increase rapidly as r_{max} decreases.

a good compromise seems to be XXXXX.

To discuss :

2) choice of initial redshift, which can lead to large truncation error if too early, but poor inaccurate Zel'dovich if too late **JD, here you could discuss the ra_max tempering...**

3) limits of resolution caused by softening length,

4) Poisson shot noise if too few particles or unrelaxed systems,

Table 3. Scaling in CPU resources as a function of the value of r_{max}. The tests were performed on the CITA Sunnyvale cluster, and general trends could vary slightly on other machines.

r_{max}	time (h)
0.1	8.74
0.05	11.47
0.01	14.30
0.005	18.87
0.002	22.52
0.001	29.62

5) finite box size effects (with beat coupling?)

7 RUNTIME HALO FINDER

We have implemented a halo finding procedure, which we have developed based on the spherical overdensity (SO) approach (Lacey & Cole 1994). In the interest of speed and efficiency the halo catalogues are constructed on-the-fly at a pre-determined list of redshifts. The halo finding is massively-parallel and threaded based on the main CUBEP3M data structures discussed in section 2. The code first builds the fine-mesh density for each sub-domain using CIC or NGP interpolation. It then proceeds to search for and record all local density maxima above certain threshold (typically set to 100 above mean density) within the local sub-domain's physical volume (excluding the sub-domain buffer zones). It then uses parabolic interpolation on the density field to determine more precisely the location of the maximum within the densest cell, and records the peak position and value. The halo centre determined this way agrees closely with the centre-of-mass of the halo particles discussed below.

Once the list of peak positions is generated, they are sorted from the highest to the lowest density value. Then each of the halo candidates is inspected independently, starting with the highest peak. The grid mass is accumulated in spherical shells of fine grid cells surrounding the maximum, until the mean density within the halo drops below a pre-defined overdensity cutoff (usually set to 178 in units of the mean, in accordance to the top-hat collapse model). As we accumulate mass we remove it from the mesh, so that no mass element is double-counted. This method is thus inappropriate for finding sub-halos as within this framework those are naturally incorporated in their host halos. Because the halos are found on a grid of finite-size cells, it is possible, especially for the low-mass halos, to overshoot the target overdensity. When this occurs we use an analytical halo density profile to correct the halo mass and radius to the values corresponding to the target overdensity. This analytical density profile is given by Truncated Isothermal Sphere (TIS) profile (Shapiro et al. 1999; Iliev & Shapiro 2001) for overdensities below ~ 130, and $1/r^2$ for lower overdensities. The TIS density profile has a similar outer slope (the relevant one here) to the Navarro, Frenk and White (NFW Navarro et al. 1997) profile, but extends to lower overdensities and matches well the virialization shock position given by the Bertschinger self-similar collapse solution (Bertschinger 1985).

Once the correct halo mass, radius and position are determined, we find all particles which are within the halo radius. Their positions and velocities are used to calculate the halo centre-of-mass, bulk velocity, internal velocity dispersion and the three angular momentum components, all of which are then included in the final halo catalogues. We also calculate the total mass of all particles within the halo radius, also listed in the halo data. This mass is very close, but typically slightly lower, than the halo mass calculated based on the gridded density field. The particle centre-of-mass corresponds very well to the halo centre found based on the gridded mass distribution.

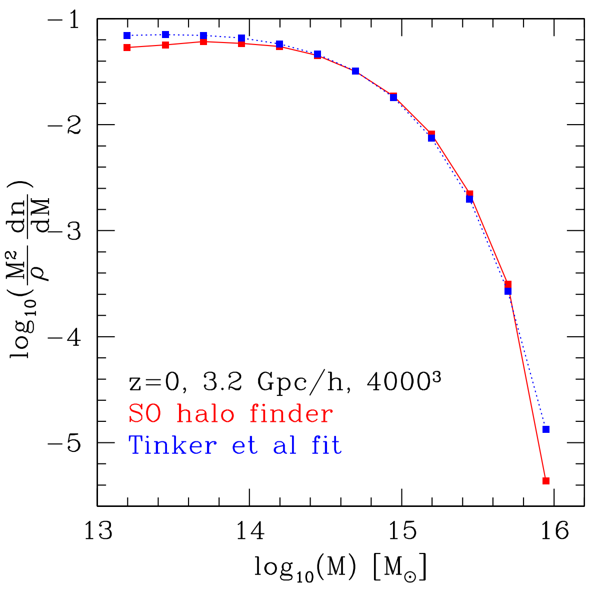


Figure 6. Simulated halo multiplicity function, $\frac{M^2}{\rho}\frac{dn}{dM}$ based on a simulation with $3.2\,h^{-1}$Gpc box and 4000^3 particles (red, solid). For reference we also show a widely-used precise fit by Tinker et al. (2008) (blue, dashed).

A sample halo mass function produced based on our inlined SO halo finder at redshift $z = 0$ from a simulation with $3.2\,h^{-1}Gpc$ box and 4000^3 particles is shown in Fig. 6. We compare our result to the precise fit presented recently by Tinker et al. (2008). Unlike most other widely-used fits like the one by Sheth & Tormen (2002), which are based on friends-of-friends (FOF) halo finders, the Tinker et al. (2008) fit is based on the SO search algorithm, whose masses are systematically different from the FOF masses (e.g. Reed et al. 2007; Tinker et al. 2008), making this fit a better base for comparison here. Results show excellent agreement, within $\sim$ 10 per cent for all halos with masses corresponding to 1000 particles or more. Lower-mass halos are somewhat under-counted compared to the Tinker et al. (2008) fit, by $\sim$ 20 per cent for 400 particles and by $\sim$ 40 per cent for 50 particles. This is due to the grid-based nature of our SO halo finder, which misses some of the low-mass halos. Using more sophisticated halo finders (available through post-processing due to their heavier memory footprint) recovers the expected mass function.

II: do we need to say anything more about having the option to use other halo finders like AHF and FOF?

A second test of the accuracy of the halo finder algorithm is to extract the halo power spectrum $P_h(k)$ and compare the halo bias with theoretical predictions. The halo bias is defined as $b(k) = \sqrt{P(k)/P_h(k)}$ and is shown in Fig. 7.

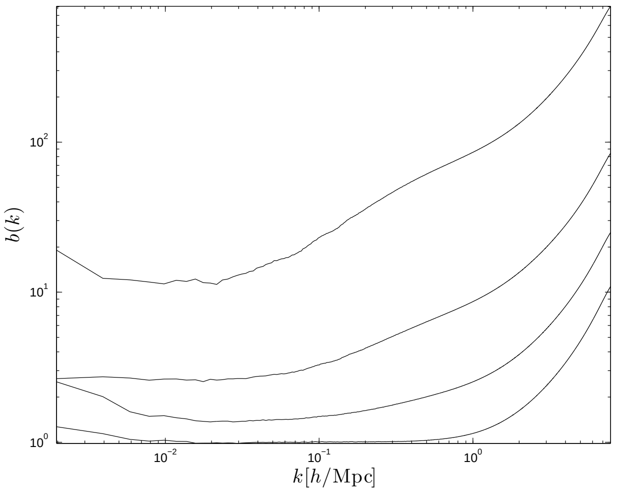


Figure 7. Halo bias, measured at $z = 0$ from a 4000^3 particles simulations with a side length of $3.2h^{-1}$Gpc. Bottom to top curves correspond to haloes of different masses, from light to massive. The four bins are decades of halo masses; the first contains haloes made of 20-200 particles, the second of 200-2,000 and so on.

8 BEYOND THE STANDARD CONFIGURATION

The preceding descriptions and discussions apply to the standard configuration of the code, as described at the beginning of section 2. A few extensions have been recently developed in order to enlarge the range of applications of CUBEP3M, and this section briefly describe the most important improvements.

8.1 Initial Conditions

As mention in section 2, the code start off by reading a set of initial conditions. These correspond to a $6 \times N$ phase-space array, where N is the number of particles in the local node. Although most applications so far were based on initial conditions that follow Gaussian statistics, we have developed a non-Gaussian initial condition generator that we briefly describe in this section.

The original code that provides Gaussian initial conditions for CUBEP3M is extended to include non-Gaussian features of the "local" form, $\Phi(\mathbf{x}) = \phi(\mathbf{x}) + f_{\rm NL}\phi(\mathbf{x})^2 + g_{\rm NL}\phi(\mathbf{x})^3$, where $\phi(\mathbf{x})$ is the Gaussian contribution to the Bardeen potential $\Phi(\mathbf{x})$ (see **?** for a review). We adopted the CMB convention, in which Φ is calculated immediately after the matter- radiation equality (and not at redshift $z = 0$ as in the large scale structure convention). For consistency, $\phi(\mathbf{x})$ is normalized to the amplitude of scalar perturbations inferred by CMB measurements ($A_s \approx 2.2 \times 10^{-9}$). The local transformation is performed before the inclusion of the matter transfer function, and the initial particle positions and velocities are finally computed from $\Phi(\mathbf{x})$ according to the Zel'dovich approximation, as in the original Gaussian initial condition generator.

This code was tested by comparing simulations and theoretical predictions for the effect of local primordial non-Gaussianity on the halo mass function and matter power spectrum (Desjacques, Seljak & Iliev 2009). It has also been used to quantify the impact of local non-Gaussian initial conditions on the halo power spectrum (Desjacques et al. 2009; Desjacques & Seljak 2010) and bispectrum (Sefusatti et al. 2010), as well as the matter bispectrum (Sefusatti et al. 2011).

8.2 Particle identification tags

A system of particle identification can be turned on, which basically allows to track each particle's trajectory between checkpoints. Such a tool is useful to a number of applications, from reconstruction of halo merging history to **(what else?)** The particle tag system has been implemented as an array of double integers, `PID`, and assigns a unique integer to each particle during the initialization stage. The location of the tag on the `PID` array matches the location of the corresponding particle on the `xv` array, hence it acts as if the latter array had an extra dimension. The location change only when particles exist the local volume, in which case the tag is sent along with the particle in the `pass_particle` subroutine. Deleted particles results in deleted flags, and the `PID` array gets written to file at each particle checkpoint. **(Anything else to say here?)**

8.3 Extended range of the pp force

One of the main source of error in the calculation of the force occurs when on the smallest scales of the fine grid. The approximation by which particles in a neighbouring mesh grid can be placed at the centre of the cell is less accurate, which cause a maximal scatter around the exact $1/r^2$ law. A solution to minimize this error consists in extending the pp force calculation outside a single cell, which inevitably reintroduces a N^2 number of operations. Our goal is to add the flexibility to have a code that runs slower, but produces results with a higher precision.

To allow this feature, we have to choose how far outside a cell we want the exact pp force. Since the force kernels on both meshes are organized in terms of grids, the simplest way to implement this feature is to shut down the mesh kernels in a region of specified size, and allow the pp force to extend therein. Concretely, these regions are constructed as cubic layers of fine mesh grids around a central cell; the freedom we have is to choose the number of such layers.

To speed up the access to all particles within the domain of computation, we construct a thread safe linked list to be constructed and accessed in parallel by each core of the system, but this time with a head-of-chain that points to the first particle in the current fine mesh cell. We then loop over all fine grids, accessing the particles contained therein and inside each fine grid cells for which we killed the mesh kernels, we compute the separation and the force between each pairs and update their velocities simultaneously with Newton's third law. To avoid double counting, we loop only over the fine mesh neighbours that produce non-redundant contributions. Namely, for a central cell located at (x_1, y_1, z_1), we only consider the neighbours (x_2, y_2, z_2) that satisfy the following conditions:

- $z_2 \geqslant z_1$ always
- if $z_2 = z_1$, then $y_2 \geqslant y_1$, otherwise we also allow $y_2 < y_1$
- if $z_2 = z_1$ and $y_2 = y_1$, then we enforce $x_2 > x_1$

The case where all three coordinates are equal is already calculated in the standard configuration of the code.

To quantify the accuracy improvement versus computing time requirements, we performed the following test. We generate a set of initial conditions at a starting redshift of $z = 100$, with a box size equal to $200h^{-1}$Mpc, and with 128^3 particles. We evolve the particles to $z = 0$ with different ranges for the pp calculation, and compare the resulting power spectra. For the results to be meaningful, we also need to use the same random seed for the random number generator, such that the only difference between different runs

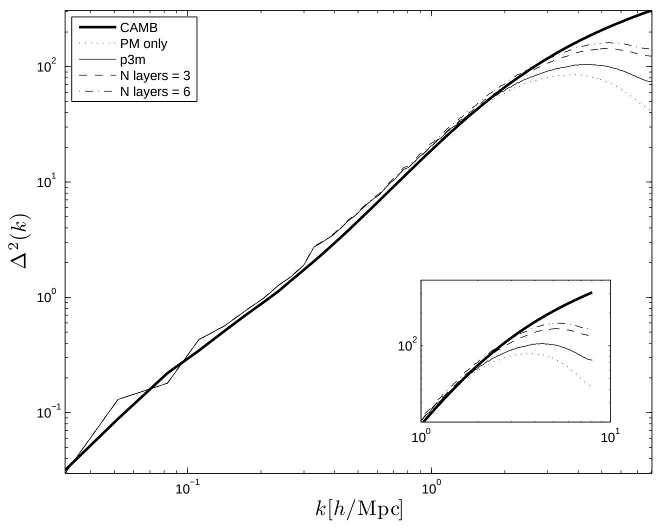


Figure 8. Dimensionless power spectrum for varying range of the exact pp force, compared to CAMB (Lewis et al. 2000).

Table 4. Scaling in CPU resources as a function of the range of the pp interaction. N_{layers} refers to the number of fine mesh layers around a given cell, inside of which the force calculation is purely given by the pp contribution.

Type	time (h)
PM	1.77
P^{3}M	2.09
N_{layers}	
1	8.74
2	11.47
3	14.30
4	18.87
5	22.52
6	29.62
7	34.82
8	47.00

is the range of the pp force. Fig. 8 shows the dimensionless power spectrum of the different runs, where we see a significant gains in resolution when extending PM to P^{3}M first, and when adding successive layers of fine cell mesh where the pp force is extended. We have not plotted the results for higher numbers of layers, as the improvement becomes milder there: the fine grid calculations are more accurate as the distance increases. For this reason, it seems that a range of a few layers, between 3 and 6, suffices to reduce most of the undesired NGP scatter.

Extending the pp calculation comes at a price, since the number of operation scales as N^2 in the sub-domain. This cost is best capture by the increase of real time required by a fixed number of dedicated CPUs to evolved the particles to the final redshift. Table 4 presents this usage.

Another way to quantify the improvement of the calculation is to look at the halo mass function for these different runs. **(To do...)**

9 CONCLUSION

This paper describes CUBEP3M, a P^{3}M N-body public code that is massively parallel, memory local, and that scales well to 20,000 cores, thus raising the limits of the cosmological problem size one can handle. We summarize the code structure, the two-mesh Poisson solver algorithm, we present scaling and systematic tests that have been performed. We also describe a few utilities and extensions that come with the public release, including a run time halo finder, an extended pp force calculation and a non-Gaussian initial condition generator.

The code is publicly available on github.com under `cubep3m`, and extra documentation about the structure, compiling and running strategy is can be found at `www.wiki.cita.utoronto.ca/mediawiki/index.php/CubePM`.

ACKNOWLEDGEMENTS

TI was supported by The Southeast Physics Network (SEPNet) and the Science and Technology Facilities Council grants ST/F002858/1 and ST/I000976/1. The authors acknowledge the TeraGrid and the Texas Advanced Computing Center (TACC) at The University of Texas at Austin (URL: http://www.tacc.utexas.edu) for providing HPC and visualization resources that have contributed to the research results reported within this paper and Partnership for Advanced Computing in Europe (PRACE) grant 2010PA0442 to ITI used for the code scaling studies.

REFERENCES

Albrecht A., et al., 2006, ArXiv e-prints (astro-ph/060959)
Bertschinger E., 1985, ApJS, 58, 39
Colless M., et al., 2003, ArXiv e-prints (astro-ph/0306581)
Couchman H. M. P., 1991, ApJ, 368, L23
Datta K. K., Mellema G., Mao Y., Iliev I. T., Shapiro P. R., et al., 2011, arXiv:1109.1284
Desjacques V., Seljak U., 2010, Phys. Rev. D, 81, 023006
Desjacques V., Seljak U., Iliev I. T., 2009, MNRAS, 396, 85
Doré O., Lu T., Pen U.-L., 2009, ArXiv e-prints
Drinkwater M. J., et al., 2010, MNRAS, 401, 1429
Dubinski J., Kim J., Park C., Humble R., 2004, NewA, 9, 111
Fernandez E. R., Iliev I. T., Komatsu E., Shapiro P. R., 2011, arXiv:1112.2064
Friedrich M. M., Datta K. K., Mellema G., Iliev I. T., 2012, ArXiv e-prints
Friedrich M. M., Mellema G., Alvarez M. A., Shapiro P. R., Iliev I. T., 2011, MNRAS, 413, 1353
Frigo M., Johnson S., 2005, in Proceedings of the IEEE Vol. 93, The Design and Implementation of FFTW3. pp 216–231
Harnois-Deraps J., Pen U.-L., 2011, ArXiv e-prints
Hilbert S., Hartlap J., White S. D. M., Schneider P., 2009, A&A, 499, 31
Hockney R. W., Eastwood J. W., 1981, Computer Simulation Using Particles
Ilie C., Freese K., Valluri M., Iliev I. T., Shapiro P. R., 2012, MNRAS, p. 2794
Iliev I. T., Ahn K., Koda J., Shapiro P. R., Pen U.-L., 2010, proceedings paper for Moriond 2010 meeting (ArXiv:1005.2502)
Iliev I. T., Mellema G., Pen U.-L., Merz H., Shapiro P. R., Alvarez M. A., 2006, MNRAS, 369, 1625

Iliev I. T., Mellema G., Shapiro P. R., Pen U.-L., Mao Y., Koda J., Ahn K., 2011, ArXiv e-prints
Iliev I. T., Shapiro P. R., 2001, MNRAS, 325, 468
Iliev I. T., Shapiro P. R., Mellema G., Merz H., Pen U.-L., 2008, in refereed proceedings of TeraGrid08, ArXiv e-prints (0806.2887)
Lacey C., Cole S., 1994, MNRAS, 271, 676
Lewis A., Challinor A., Lasenby A., 2000, Astrophys. J., 538, 473
Lu T., Pen U.-L., 2008, MNRAS, 388, 1819
Lu T., Pen U.-L., Doré O., 2010, Phys. Rev. D, 81, 123015
Mao Y., Shapiro P. R., Mellema G., Iliev I. T., Koda J., Ahn K., 2012, MNRAS, 422, 926
Merz H., Pen U.-L., Trac H., 2005, New Astronomy, 10, 393
Navarro J. F., Frenk C. S., White S. D. M., 1997, ApJ, 490, 493
Ngan W.-H. W., Harnois-Déraps J., Pen U.-L., McDonald P., MacDonald I., 2011, ArXiv e-prints (astro-ph/1106.5548)
Reed D. S., Bower R., Frenk C. S., Jenkins A., Theuns T., 2007, MNRAS, 374, 2
Rimes C. D., Hamilton A. J. S., 2005, MNRAS, 360, L82
Sato M., Hamana T., Takahashi R., Takada M., Yoshida N., Matsubara T., Sugiyama N., 2009, ApJ, 701, 945
Schlegel D., White M., Eisenstein D., 2009, ArXiv e-prints (astro-ph/0902.4680)
Sefusatti E., Crocce M., Desjacques V., 2010, MNRAS, 406, 1014
Sefusatti E., Crocce M., Desjacques V., 2011, ArXiv e-prints
Shapiro P. R., Iliev I. T., Raga A. C., 1999, MNRAS, 307, 203
Sheth R. K., Tormen G., 2002, MNRAS, 329, 61
Takahashi R., et al., 2009, ApJ, 700, 479
Takahashi R., et al., 2011, ApJ, 726, 7
Tinker J., Kravtsov A. V., Klypin A., Abazajian K., Warren M., Yepes G., Gottlöber S., Holz D. E., 2008, ApJ, 688, 709
Trac H., Cen R., 2007, ApJ, 671, 1
Trac H., Pen U., 2003, in American Astronomical Society Meeting Abstracts Vol. 35 of Bulletin of the American Astronomical Society, Out-of-Core Hydrodynamic Simulations of the IGM. p. 1365
Trac H., Pen U.-L., 2004, NewA, 9, 443
Vafaei S., Lu T., van Waerbeke L., Semboloni E., Heymans C., Pen U.-L., 2010, Astroparticle Physics, 32, 340
Vale C., White M., 2003, ApJ, 592, 699
Xu G., 1995, ApJS, 98, 355
York D. G., et al., 2000, AJ, 120, 1579
Yu H., Zhang T., Harnois-Déraps J., Pen U., 2010, ArXiv e-prints
Zackrisson E., Scott P., Rydberg C.-E., Iocco F., Sivertsson S., Östlin G., Mellema G., Iliev I. T., Shapiro P. R., 2010, MNRAS, 407, L74
Zahn O., Lidz A., McQuinn M., Dutta S., Hernquist L., Zaldarriaga M., Furlanetto S. R., 2007, ApJ, 654, 12
Zhang T., Yu H., Harnois-Déraps J., MacDonald I., Pen U., 2010, ArXiv e-prints