Mon. Not. R. Astron. Soc. **000**, 1–4 (2011)   Printed 17 April 2012   (MN LaTeX style file v2.2)

CUBEP3M: High Performance P3M N-body code

Joachim Harnois-Déraps[1,2] ⋆, Ue-Li Pen[1] †, Hugh Merz[3] ‡ and Ilian T. Iliev[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]*Department of XXX*
[4]*Department of XXX*
[5]*Department of XXX*

17 April 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 subgrid level and across neighboring cells. The volume decomposition is now cubical, and most of the calculations on the fine mesh are `openmp` parallel. Many utilities have been added and are briefly described, including a runtime halofinder and particle identification tags. We discuss the structure of the code, its accuracy, and its scaling performance.

Key words: To specify...

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 (**????**) 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 we understands where and 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 (**???**). 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 (**???**). **(Ilian, could you say something about reionization here? Just general references as to why one needs N-body codes to perform accurate predictions/forecasts, etc.)**

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 (**?**) 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 (**?**).

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 (**????**), 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. Unlike Hydra (**?**) and the PM code by **?**, these are not designed to perform large scale PM calculations.

(Describe briefly Hydra, PM code by japanese, and others...)

PMFAST (**?**, 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 **?**, 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 OPENMPparallel computa-

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

tion. This this setup enable the code to evolve very 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 came a new name, CUBEPM, which soon after incorporated particle-particle (pp) interactions at the sub-grid level, and the code was renamed CUBEP3M. It now includes a significant number of new features since MPT : 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 halofinder 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?)**.

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 (**?????**), BAO (**?????**), re-ionization (**?**), **(anymore cubep3m papers to cite?)**, 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 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
(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

[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.

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 **??** for details), and can compute the power spectrum on the coarse grid at run time **(is this correct?)**. The code exits the loop when it has reached the final redshift, it then wraps up the FFTW plans and clears the MPI communicators. 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 (**?**). Particle-particle interactions are not interpolated, but a sharp cutoff kills the force for pairs closer to a tenth of a grid cell.

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

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

```
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.

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:

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

and

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

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

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

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 cut-off 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 behaviours 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 $14 \leqslant r \leqslant 16$, $\omega_s(\mathbf{r}) \rightarrow \omega_s(\mathbf{r}) * 0.985$, and for $12 \leqslant r \leqslant 16$, $\omega_l(\mathbf{r}) \rightarrow \omega_l(\mathbf{r}) * 1.2$. The two fudge factors were found by performing force measurements on two particles randomly placed in the volume.

In addition, the long range force is subject to a correction that fixes **(what exactly fixes the LRCKCORR flag?)**.

(Anything else here?)

4 ACCURACY OF THE CODE

Pairwise Force (transverse and radial), density force, power spectrum, initial Redshift, growth factor, limits of resolution, softening length, random offset ...

5 SCALING PERFORMANCES

Real time versus size, bottle neck, extended pp, trials on scinet and ranger, p3dfft,

6 SYSTEMATICS

Random shift, resolution, initial redshift...

JD, here you could discuss the ra_max tempering...

7 SPERICAL OVERDENSITY HALO FINDER

Spherical overdensities, search algorithm, provided halo information, Comparison to PS and ST.

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. **Vincent, here you go!**

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

(JD, here will be one of your contribution!)

8.4 Generalization of the dark energy equation of state

9 CONCLUSION

10 ACKNOWLEDGEMENTS

REFERENCES

This paper has been typeset from a TEX/ LATEX file prepared by the author.