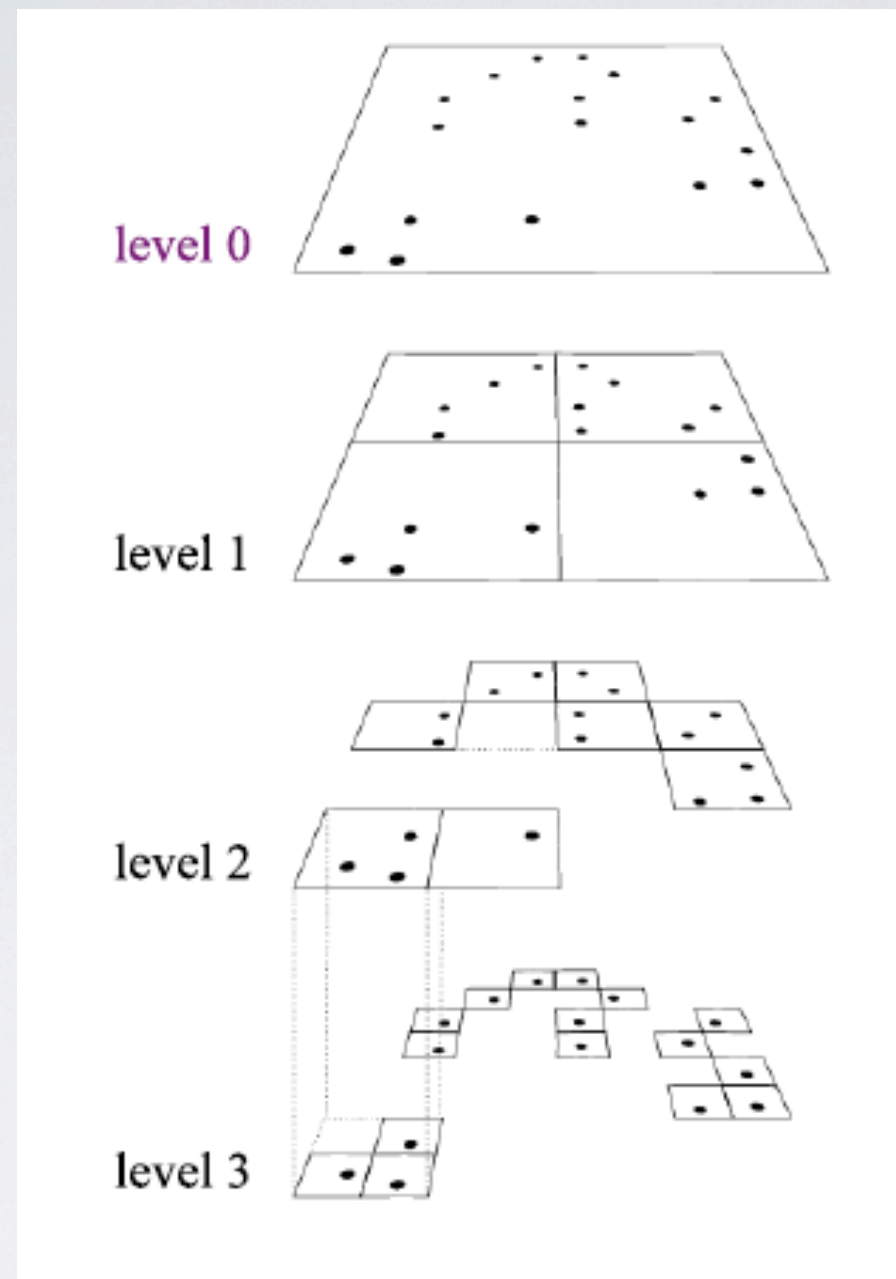


Tree structure and tree walk



Particles are organized in “nodes”, namely recursive divisions of the computational volume until individual particles are reached (buckets). The full domain is called level 0 node or “root”. Various types of trees, most commonly the oct-tree (Barnes & Hut 1996) and the binary tree. Above the graph of an oct-tree in 2D (in 3D each node is a cube, with a recursive division into eight sub-cubes until buckets are reached)

Procedure (tree-walk) is the following:

for each force computation step on a given target particle one starts from the “root” node, examines the opening angle, and if that is not satisfied the node is “opened” and the next node level of the tree is examined. When the aperture angle is found small enough then the multipole expansion is performed and the force computation for the corresponding subset of particles finalized. Empty sub-nodes (sub-cubes) at any level need not to be stored (difference with grid-based methods where all cells stored!)

Clearly more nodes will have to be opened for closer particles until the opening angle criterion is satisfied. For the closest particles eventually individual particles are reached, for which direct summation is performed.

In modern TreeCodes, such as PKDGRAV2 (Stadel 2010), the “bucket” level contains not one but $N=8-10$ particles, to gain accuracy in high resolution simulations with strong clustering.

Note that *by construction* the tree algorithm is adaptive: in a given computational domain the tree is constructed based on the particle distribution rather than being a pre-defined structure like a grid and can in principle be adapted to an arbitrary “deep” clustering configuration.

In practice, one constructs the hierarchical grouping by sequentially inserting the particles into the tree and then computes multipole moments recursively for the different nodes (*tree-build step*)
This means individual particles belong simultaneously to different node levels -- depending on the target particle on which the forces have to be computed different nodes will be activated during the tree walk

For the standard tree-algorithm the cost is $O(N\log N)$ (see next slides). A faster algorithm can be obtained if one does a multipole expansion also for the potential of the *target* particle rather than just for the *sources*, and further exploits Newton's third law to the multipole-based force to reduce the number of interactions that have
----> *Fast Multipole Methods* (FFM - eg Dehnen 2002;2014), with cost even slightly better than $O(N)$

Cost of tree-based force computation

For simplicity assume N particles are distributed in an homogeneous sphere with radius R . The mean inter-particle separation will thus be:

$$d = \left[\frac{(4\pi/3)R^3}{N} \right]^{1/3}$$

We can define the computational cost as the number of nodes N_{nodes} that we need to compute the forces on a target particle at the center of the sphere (neglect direct summation part)

$$N_{\text{nodes}} = \int_d^R \frac{4\pi r^2 \mathrm{d}r}{l^3(r)},$$

The above expression is valid if we assume the nodes tessellate the whole sphere, and l is the size of a node with distance r from the target particle (while d is the characteristic distance of the nearest particle, whose force will be eventually computed using direct sum)

Assuming the nodes will be close to the maximum size allowed by the opening angle criterion, they will be “far enough” so that we can write $l \simeq \theta_c r$

With this last approximation and combining the two equations above we can then obtain:

$$N_{\text{nodes}} = \frac{4\pi}{\theta_c^3} \ln \frac{R}{d} \propto \frac{\ln N}{\theta_c^3}$$

which means for N particles the cost is $O(N \ln N)$ -- but note the actual number of nodes depends heavily on the choice of θ_c !

Typical force error

If we consider only the monopole term, just to have a sense how the error varies with the parameters used in the tree algorithm, the error in the force per unit mass given by just one node should be of order the truncation error:

$$\Delta F_{\text{node}} \sim \frac{GM_{\text{node}}}{r^2} \theta^2.$$

The errors from multipole nodes will add up in quadrature, which yields:

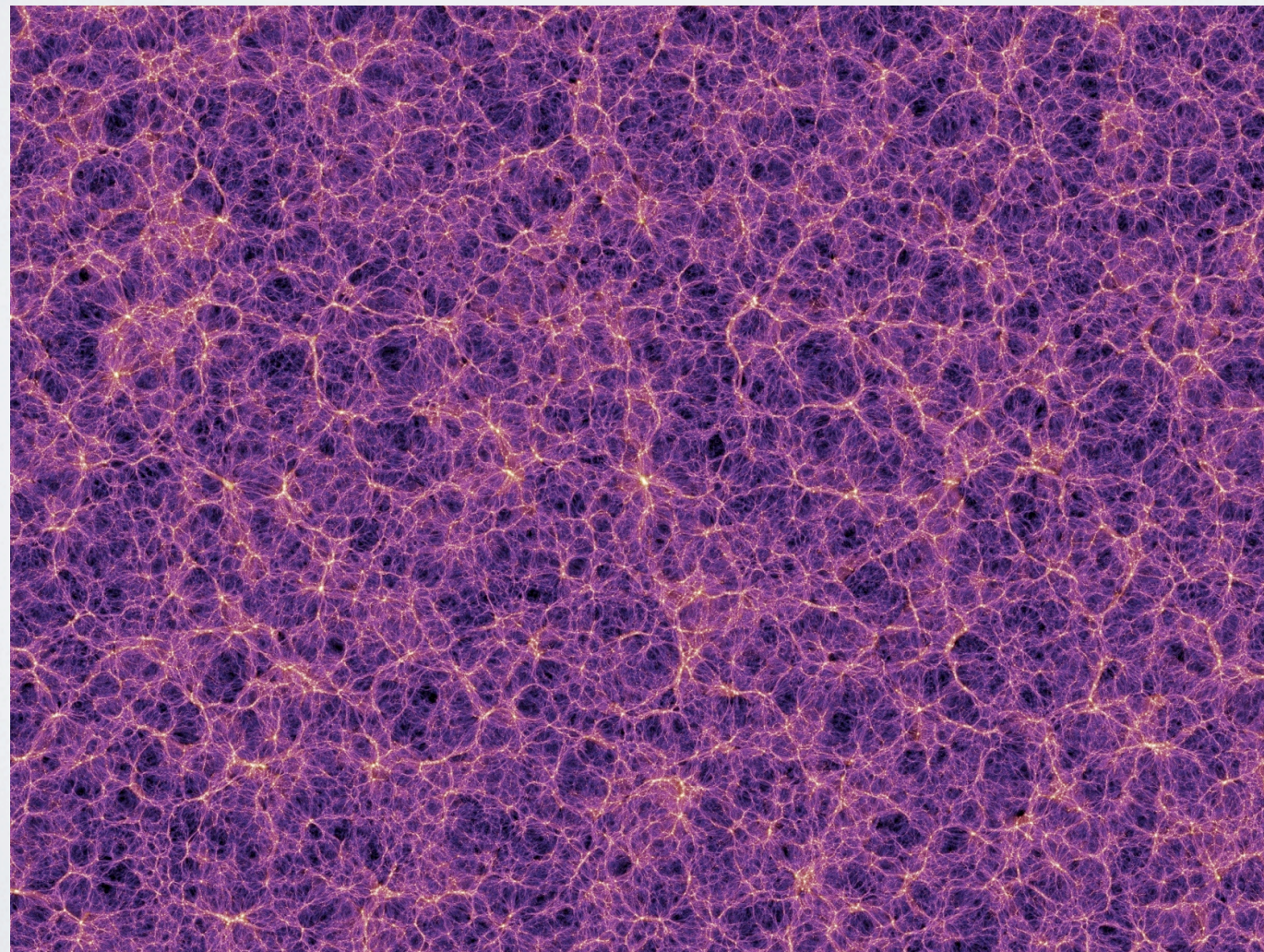
$$(\Delta F_{\text{tot}})^2 \sim N_{\text{node}} (\Delta F_{\text{node}})^2 = N_{\text{node}} \left(\frac{GM_{\text{node}}}{r^2} \theta^2 \right)^2 \propto \frac{\theta^4}{N_{\text{node}}} \propto \theta^7$$

Therefore the error scales roughly inversely as the computational cost (see dependence of cost on θ_c on previous slide), namely

$$(\Delta F_{\text{tot}}) \propto \theta^{3.5},$$

Hybrid techniques

The treecode is fastest when high accuracy is required, but in general PM codes are the fastest. However, when clustering is strong, as in cosmological simulations at relatively small scale, PM would smooth the force excessively, losing accuracy (unless a very large number of cells is used, which contain only a few particles!). But at large scales, the same cosmological simulations show a more uniform density field, for which smoothing as PM does should be fine. A possible strategy is thus to combine different methods, using a *more accurate one for local forces* and a *faster, less accurate one for distant forces* (eg PM)



Various hybrid techniques:

P³M - direct summation + PM

AP³M - adaptive grid version of P3M, more efficient

TreePM - treecode + PM, possibly best combination of accuracy and efficiency

P3M

Decompose the total force in 2 components:

$$F_{\text{total}}(r) = F_{PM}(r, a) + F_{PP}(r, \epsilon)$$

a=cell size

The PP force is a short-range interaction for $r < r_c = 2-3a$:

$$F_{PP}(r, \epsilon) = \frac{1}{r^2 + \epsilon^2} - F_{ref}(r, a)$$

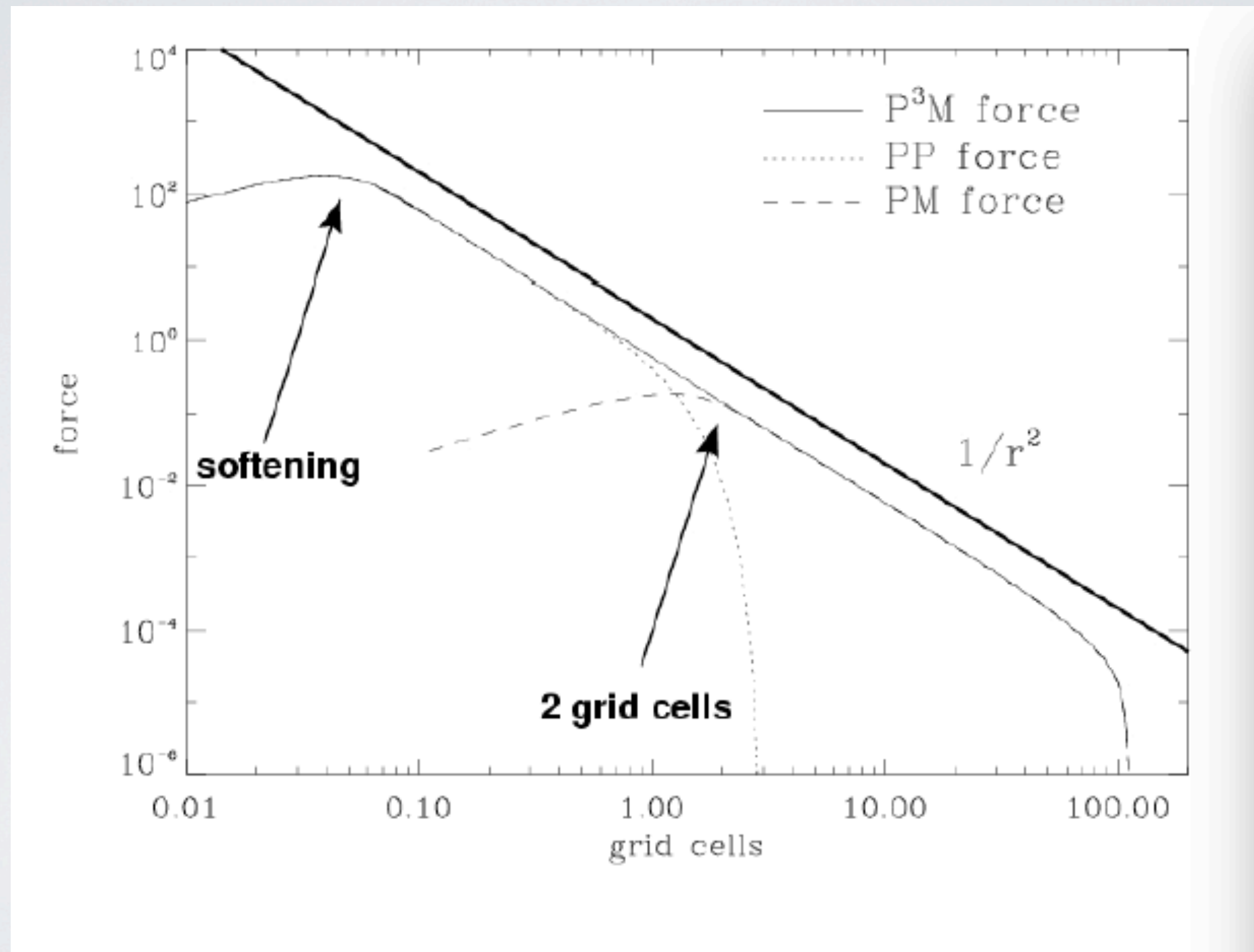
The PP force is computed with direct N^2 summation, for all particles within the cut-off radius.

Problem: for highly clustered configuration, the PP force dominates the CPU time in the P3M scheme.

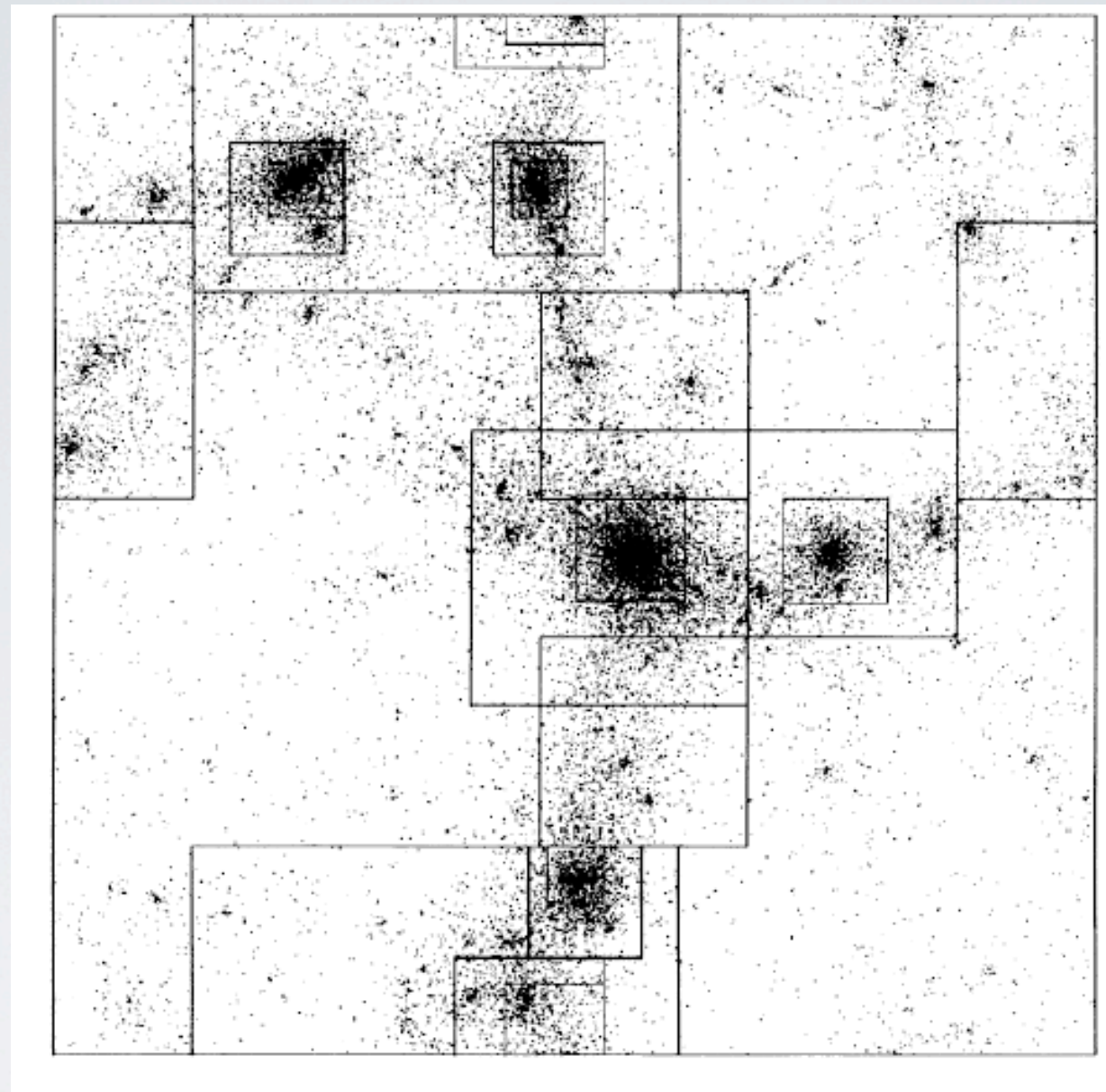
Solution: adaptive grids: the AP3M scheme.

Couchman, H.M.P., "Mesh refined P3M: A fast adaptive N-body algorithm", ApJ, **368**, 23, (1991)

Graph of P³M force calculation through box

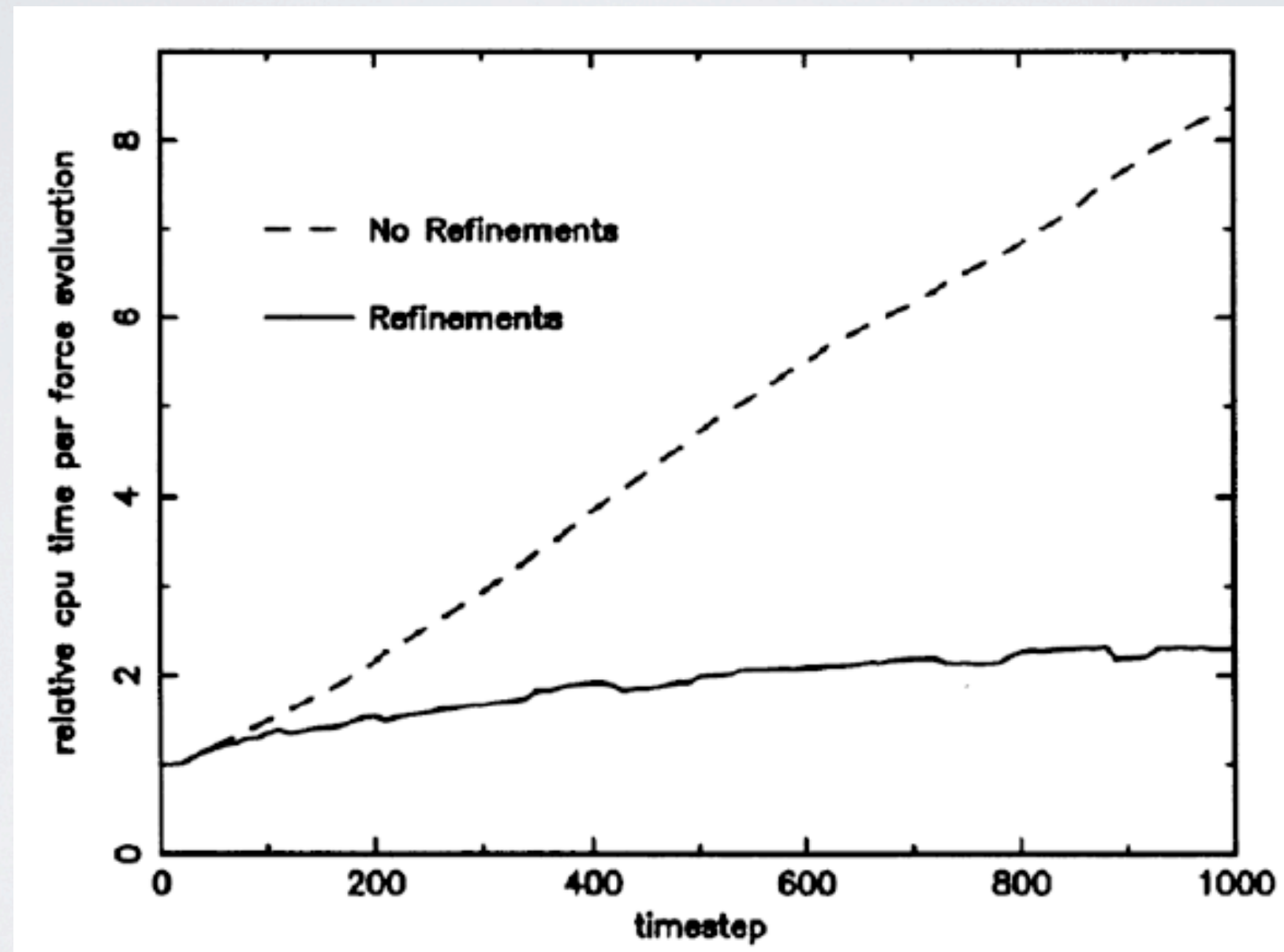


AP³M scheme



In this method the weight of the direct summation calculation is reduced by performing PM on finer grids till relatively small particle separations. Force is decomposed recursively in smaller and smaller scale components - only the coarsest grid used for long range force. Requires force shaping through grid levels solving Poisson for non-periodic boundaries. Not more accurate, just faster than P³M

Relative speed gain in AP³M relative to P³M for a cosmological box (low number of particles, at high number of particles where clustering stronger gain can be larger)



TreePM codes: idea and force shaping

Well known example is GADGET2 code (Springel 2005).

One starts from PM part and decomposes individual modes into long-range and short-range part:

$$\Phi_{\mathbf{k}} = \Phi_{\mathbf{k}}^{\text{long}} + \Phi_{\mathbf{k}}^{\text{short}},$$

$$\Phi_{\mathbf{k}}^{\text{long}} = \Phi_{\mathbf{k}} \exp(-\mathbf{k}^2 r_s^2),$$

$$\Phi_{\mathbf{k}}^{\text{short}} = \Phi_{\mathbf{k}} [1 - \exp(-\mathbf{k}^2 r_s^2)].$$

\mathbf{r}_s is the spatial scale of the force split (exponential cut-off).

Below a few cut-off radii a tree calculation is performed. This way the tree component of the calculation is efficient since it will cover only a limited region of the domain, allowing to consider only a low number of nodes.

Relative to PM the difference is that the Greens function in Fourier space gets an additional exponential smoothing, reducing grid-related anisotropies in the transition region.

For the short-range force one has to transform back into real space and use

$$\Phi^{\text{short}}(\mathbf{x}) = -G \frac{m}{r} \text{erfc} \left(\frac{r}{2r_s} \right)$$

This looks like the newtonian potential modified by a truncation factor that rapidly turns off the force at large distances. In practice the force is made $\sim 1\%$ of newtonian at $r \sim 4.5r_s$.

In practice one computes the above short-range potential with the treecode, namely not as single particle force but using the multipole expansion for a whole node. Clearly there will be no nodes above $r \sim 4.5r_s$.

In addition, this algorithm is ideal for cosmological simulations since it avoids the problem of introducing periodic boundaries in the treecode (Ewald summation) -- one simply exploits the natural periodicity of the long-range part performed by the PM code using FFT!