Particle-Mesh Techniques

P. MacNeice
Hughes STX, Goddard Space Flight Center
Greenbelt, MD 20771
macneice@alfven.gsfc.nasa.gov

July 5, 1996

Abstract

This is an introduction to numerical Particle-Mesh techniques, which are commonly used to model plasmas, gravitational N-body systems and both compressible and incompressible fluids. The theory behind this approach is presented, and its practical implementation, both for serial and parallel machines, is discussed. This document is based on a four hour lecture course presented by the author at the NASA Summer School for High Performance Computational Physics, held at Goddard Space Flight Center.

Contents

1 Introduction

Particle simulation techniques attempt to model many-body systems by solving the equations of motion of a set of particles or pseudo-particles which are used to represent the system. Particle-mesh(PM) techniques represent a popular variant on this theme, in which a numerical mesh is added to more effectively compute the forces acting on these model particles.

Otherwise known as Particle-In-cell(PIC), particle-mesh codes come in two basic flavors, pure particle-mesh, and a combination of particle-mesh and particle-particle known as P^3M [1]. In this chapter we will only consider pure particle-mesh.

The particle-mesh technique was originally invented at Los Alamos(circa 1955 - Harlow et al) for application in compressible fluid flows [6]. It was essentially reinvented in the mid sixties by Buneman and by Hockney for application to plasmas. Since then it has been applied most often to plasmas, but also to gravitational N-body systems, in solid state device design, compressible fluid flow, incompressible fluid flow(the vortex method), and MHD.

In the fluid and MHD applications, the particles are introduced as a numerical artifice to add an appealing lagrangian character to the model. We will mention these applications only briefly. Our main focus will be those applications which model true particle systems. More specifically we will concentrate on those in which the systems particles interact with each other through long range forces. In what follows, unless otherwise stated, we shall assume that our objective is to model a system of particles interacting through long range forces.

P^3M is used almost exclusively for modeling gravitational N-body systems.

Tracking particle trajectories enables us to explore physical effects which are inaccessible to other modeling techniques. For example the more common fluid modeling approaches average over the velocity distributions of the particles in the system. To do this they implicitly assume a form for this distribution, which is invariably close to equilibrium. But much of the interesting physics of these systems is associated with timescales before these equilibrium velocity distributions can be established. To explore this physics we need to resolve this structure in velocity space. There are two techniques we might use to study structure in velocity space, a continuum approximation based on a Boltzmann or Vlasov type equation, or a particle code. Particle codes are

- simpler to code up and analyze than Vlasov solvers
- more robust
- have a lagrangian character which lends them a certain economy
- can be applied in 2 or 3D where Vlasov solvers are generally impractical.

However they are generally noisier than Vlasov solvers.

The appeal of particle codes is rather obvious. By modeling a system, such as a plasma using particles, we are automatically incorporating much of the systems structure in our model. This has many computational advantages. Instead of working with a complex set of fluid equations, or a Fokker-Planck equation, we get to work with the equations of motion of the particles which are simple ODEs. We never have to worry about negative mass or energy densities developing. Also particle codes have a natural adaptivity in the sense that computational effort is automatically focused where the particles accumulate.

Of course particle codes have limitations of their own. For example in most cases the real physical system has many more particles than we can cope with on a computer. As a result the forces acting on the individual particles in a numerical model are much noisier[1] than in the real physical system and we are required to take measures to limit this noise. For systems in which the particles interact through long-range forces, the most naïve implementation scales in a prohibitively expensive way and we are forced to develop more efficient algorithms. These problems limit the range of physical systems to which particle methods can be successfully applied.

The best way to understand the limitations which apply to particle-mesh codes is by trying to construct a code. So let us consider how we would construct a particle model of an electrostatic plasma.

1.1 Particle-Particle

Let us assume that at time t we have N_p particles in our system located at $\boldsymbol{x}_i(t)$ with velocities $\boldsymbol{v}_i(t)$, where $1 \leq i \leq N_p$.[2] The force on particle i is

[1]Noisier in the sense that statistical fluctuations become more significant. We will explore this point in more detail later.

[2]Unless otherwise stated, variables subscripted with the letters i or j will be assumed to identify properties associated with particles.

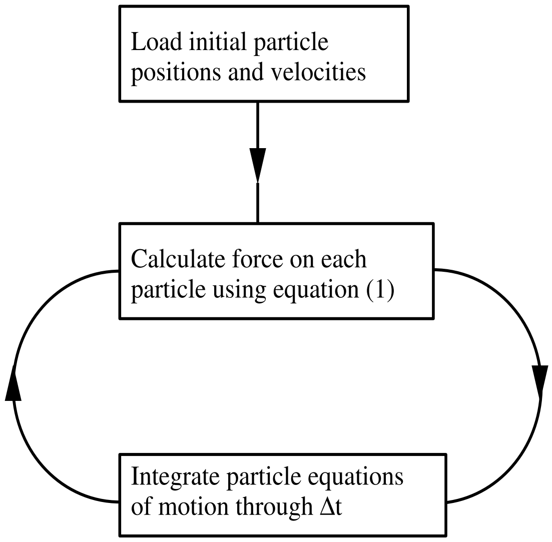

Figure 1: Flowchart for a Particle-Particle code.

given by

$$\boldsymbol{F}_i = q_i \sum_{j=1,\neq i}^{N_p} q_j \frac{\boldsymbol{x}_i - \boldsymbol{x}_j}{|\boldsymbol{x}_i - \boldsymbol{x}_j|^3} \tag{1}$$

where q_i is the charge on particle i. The equations of motion for particle i, with mass m_i are

$$\frac{d\boldsymbol{x}_i}{dt} = \boldsymbol{v}_i$$

$$\frac{d\boldsymbol{v}_i}{dt} = \frac{\boldsymbol{F}_i}{m_i}$$

The flowchart for a simple program to model this electrostatic system might look like figure 1. The force on particle i is computed by summing its interaction with every other particle. We would describe this as a particle-particle code. It has a severe limitation. The number of arithmetic operations required in the force evaluation scales as N_p^2. In a 3D simulation the interaction between 2 particles requires approximately 10 floating point operations. For a code running for N_t timesteps the force evaluation requires roughly $10 \times N_t \times N_p^2/2$ floating point operations. On one processor of a YMP C-90 which has a clock speed of 4 nanoseconds, a 1000 timestep calculation using a modest 10^6 particles would last more than 200 days.

So the particle-particle approach will only allow us to model a system in which the physics is determined by the interaction of a small($< 10^5$) number of particles. For any other system we need to reduce the scaling of the operation count in the force evaluation below order N_p^2. This is done in one of two ways, by using a particle-mesh technique, or by using a tree code. Particle-mesh techniques are most effective when the particle density distribution is relatively uniform. Tree codes are favored in systems with large density contrast.

1.2 Particle-Mesh

In the particle-mesh approach we replace the force equation (1) with an evaluation based on continuum representations of the charge density and electric field. Poisson's equation

$$\boldsymbol{\nabla}^2\phi = -\rho(\boldsymbol{x}) \tag{2}$$

relates the charge density $\rho(\boldsymbol{x})$ to the electric potential ϕ. The electric field is

$$\boldsymbol{E}(\boldsymbol{x}) = -\boldsymbol{\nabla}\phi \tag{3}$$

and the force on particle i is

$$\boldsymbol{F}_i = q_i\boldsymbol{E}(\boldsymbol{x_i}) \tag{4}$$

We then use finite difference approximations on a mesh to solve equations (2)-(4). The steps in this process are

- calculate ρ at each mesh point using the particle position information. This operation scales as order N_p.
- solve equation (2). If we use Fast Fourier Transforms(FFT's) this scales as $N_g \ln N_g$ where N_g is the number of mesh points.
- evaluate $\boldsymbol{E}$ at each mesh point using equation (3). This operation scales as N_g.
- use interpolation and equation (4) to evaluate the force on each particle. This scales as N_p.

Combining these steps we can see that the number of floating point operations for the complete scheme scales as

$$\alpha N_p + \beta N_g \ln N_g + \gamma N_g$$

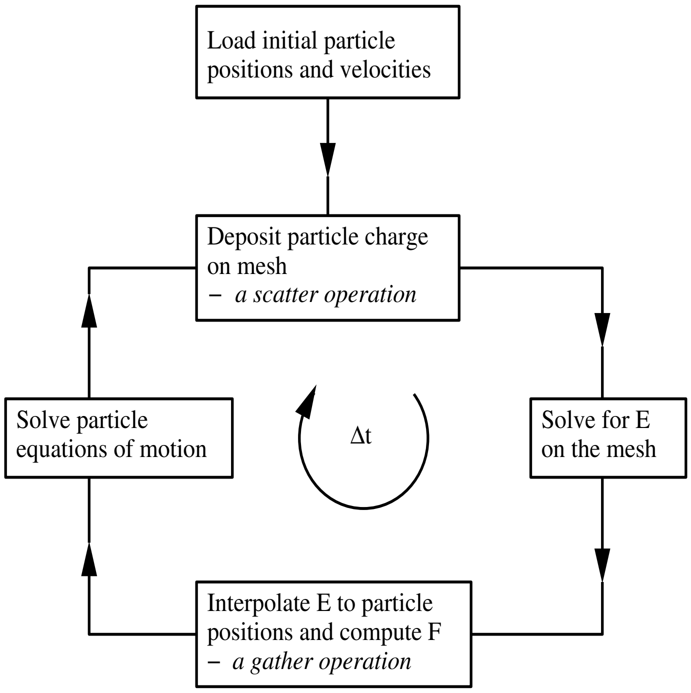

Figure 2: Flowchart for a Particle-Mesh code.

where α, β and γ are constants. The flow chart for this scheme is shown in figure 2.

1.3 For Which Systems Does PM Work?

Obviously when we introduced the mesh which we shall assume is uniformly spaced with cell size Δ, we sacrificed our ability to accurately model phenomena at length scales shorter than Δ.

Consider the field produced by a single charge. When we assign this charge to the mesh we must decide which mesh points in the particles vicinity acquire charge. The mesh spacing Δ is the length scale characterizing the coarseness of this assignment. In a sense the particle has acquired a finite size. The force field produced by this 'finite-sized' particle will accurately reproduce that of a point particle at distances from the particle which are large compared with Δ. But no matter how we assign the charge, the force will become more inaccurate as the distance from the particle decreases toward Δ.

Now let us pose the question, for what types of physical system are particle-mesh codes appropriate?

Suppose that the nature of the system is such that the force on any particle is dominated by the contributions from its nearest neighbors. Systems of this type are often called 'collisional' or 'correlated'. In these cases the force evaluation will be inaccurate unless we make Δ small compared with the typical distance of closest approach. Let us make the rather conservative assumption that the distance of closest approach (l_{close}) is as large as $1/10^{\text{th}}$ of the average inter-particle spacing. Then this accuracy criterion requires

$$\Delta \;<<\; l_{close} \;=\; \frac{L}{10\; N_p^{1/3}}$$

where $L = (N_g \Delta^3)^{1/3}$ is the characteristic physical dimension of the system. We can rewrite this as

$$\left(\frac{N_p}{N_g}\right)^{1/3} << .1.$$

For example if

$$\left(\frac{N_p}{N_g}\right)^{1/3} \leq .01$$

which produces a force error of $\leq 1\%$ at closest approach, then

$$N_g \geq 10^6 N_p,$$

ie. more than 10^6 grid cells per particle are required. Clearly particle-mesh is an expensive way to achieve even modestly accurate individual particle trajectories in a 'collisional' system, both in terms of the necessary flops and storage. It should come as no surprise that there are more efficient ways to achieve this accuracy (eg. tree codes which scale as $N_p \ln N_p$).

Now consider the opposite case, where close neighbors contribute very little to the force on a particle which is dominated by the sum of its interactions with distant particles. This type of system is called 'collisionless' or 'uncorrelated'. At first sight this may seem an unlikely circumstance for a force law which falls off as r^{-2}, but in fact plasmas, and most N-body gravitational systems fit this description. In these cases the important contributions to the force are accurately represented, since from the point of view of any particle most of the other particles in the system will be many cells away. So particle-mesh should be appropriate for collisionless, not collisional, systems.

1.4 Noise Reduction

The granularity of a particle representation inevitably introduces short-scale fluctuations into the force field, which are superimposed upon a more slowly varying component. The mean amplitude of these fluctuations is proportional to $\sqrt{n}$, where n is the particle number density. The ratio of the mean amplitudes of the fluctuations to the slowly varying component varies as $1/\sqrt{n}$. In the real system these fluctuations cause particles to be scattered at a frequency which we call the collision frequency. Because our numerical model typically uses far fewer particles than are present in reality, the effect of these fluctuations is greatly enhanced. This produces anomalously large collision frequencies.

We need to take steps to reduce the significance of these fluctuations. Fortunately we do not need to reduce the fluctuation amplitudes to their correct values, but merely to levels at which they no longer dominate the forces on the particles, or influence the particles significantly during the course of our simulation. Remember, each particle contributes charge to some number, C, of cells in its neighborhood. The average value of C depends on the number of spatial dimensions, d, in the model, and on the shape and size of the particles. Suppose each particle is a uniform charge cloud of length w in each dimension. Then for $w = \Delta$ in 1D we get $C = 2$, in 2D we get $C = 2^2 = 4$ and in 3D we get $C = 2^3 = 8$. In general we have

$$C = \left(1 + \frac{w}{\Delta}\right)^d.$$

The granularity with which the particles represent the charge distribution is reduced by increasing the number of charge contributions which are made in each cell. This requirement is expressed in the inequality

$$CN_p >> N_g,$$

or

$$\frac{N_p}{N_g} >> \frac{1}{C} = \left(1 + \frac{w}{\Delta}\right)^{-d}.$$

We can see that there are two ways to satisfy this inequality and reduce the importance of the statistical fluctuations, either by increasing N_p/N_g, the number of particles per cell, or by increasing w/Δ, the ratio of the particle size to the grid size.

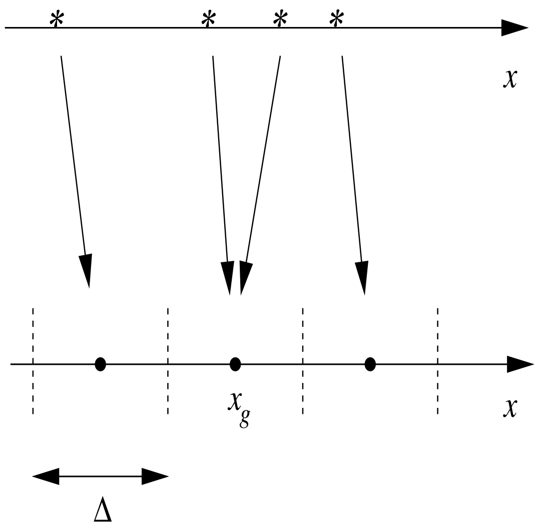

Figure 3: Charge assignment for NGP in 1D. The asterisks denote the location of particles, and the filled circles the mesh cell centers. The arrows associated with each particle indicate to which grid cell all the particles charge is to be assigned. The dashed vertical lines denote mesh cell boundaries.

1.5 Charge Assignment and Force Interpolation

Once we introduce the grid we can no longer view the particles as point particles. We have to specify how the particle charges are assigned to the mesh and how the electric field at the mesh points is used to determine the field acting on each particle. As we noted this leads naturally to the idea of a finite sized particle.

The simplest charge assignment is called "Nearest Grid Point" (NGP). As the name suggests this associates a particles charge with the grid point nearest to the particle, as shown in figure 3. The most popular scheme is "Cloud-in-Cell"(CIC). In this case each particle can be regarded as a uniform distribution of charge, of width w, centered about the particle's nominal location, as shown in figure 4. Usually w is set equal to the mesh cell size Δ. CIC is slightly more expensive computationally than NGP, but has better numerical properties.

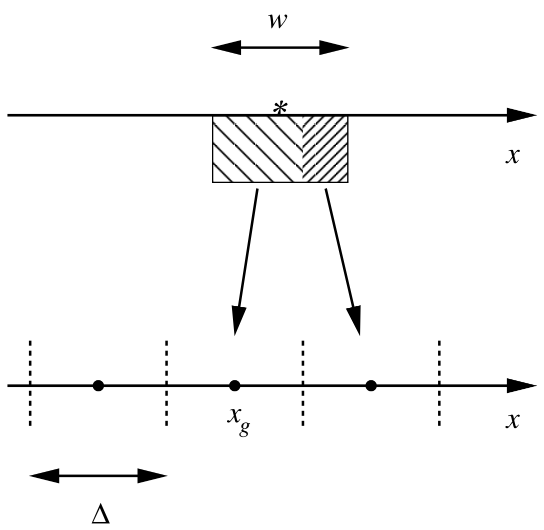

Figure 4: Charge assignment for CIC in 1D. As in the previous figure the asterisk denotes the particle location. The different hatched areas indicate the relative proportions of the particle's charge assigned to mesh cells g and $g+1$.

1.6 N-body Theory

Before we can understand why PM works for plasmas and some gravitational N-body systems, we need to understand a little about how these systems behave.

Our goal in N-body theory is to understand the evolution of macroscopic properties of the system. We can specify the initial conditions and boundary conditions for the macroscopic variables which interest us. The evolution of these macroscopic quantities depends on the behavior of the N bodies, but the macroscopic quantities do not uniquely identify the micro-state of the system. By micro-state we mean that the state of the system is defined in terms of the positions and momenta of every particle. There are an infinite number of micro-states which are consistent with a given macro-state.

The theory for this problem is a statistical one. It imagines an infinite number of copies(micro-states) of the system, each consistent with the macro-state. The description of a typical micro-state is based on an expansion about the average of this 'ensemble' of micro-states, the so called ensemble average. The ensemble average defines a smoothed continuous force field. A true micro-state has the granularity that accompanies a distribution of discrete charges. It can be represented by the sum of the average force field, and a term describing fluctuations in the force field. If $\boldsymbol{F}^M$ is the force field of the micro-state, and we use $\langle\ \rangle$ to denote ensemble averaged quantities, then

$$\boldsymbol{F}^M = \langle \boldsymbol{F}^M \rangle + \delta \boldsymbol{F}^M$$

where $\delta \boldsymbol{F}^M$ are the fluctuations in the micro-state. We are not interested in the exact form of $\delta \boldsymbol{F}^M$ for any particular micro-state. We are interested in the statistical properties of these fluctuations in the ensemble of micro-states.

Let $x = (\boldsymbol{r}, \boldsymbol{p})$ denote a point in 6 dimensional phase space, and $x_i = (\boldsymbol{r}_i, \boldsymbol{p}_i)$ the i^{th} particles coordinates. The micro-state is given by $X = (x_1, x_2, \ldots, x_N)$. The distribution function for X is denoted by $f_N(X,t)$, where $f_N(X,t)dX$ is the probability that the system is in a micro-state in the element dX at X. It evolves according to the Liouville equation

$$\frac{\partial f_N}{\partial t} + \sum_{1 \leq i \leq N} \left(\frac{\boldsymbol{p}_i}{m} \cdot \frac{\partial f_N}{\partial \boldsymbol{r}_i} + \boldsymbol{F}_i \cdot \frac{\partial f_N}{\partial \boldsymbol{p}_i} \right) = 0.$$

The microscopic phase density is

$$N^M(x,t) = \sum_{1 \leq i \leq N} \delta(x - x_i(t)).$$

The ensemble average of N^M is given by

$$\begin{aligned} \langle N^M(x,t) \rangle &= \int d^N y_i \sum_{1 \leq i \leq N} \delta(x - y_i(t)) f_N(y_1, y_2, \ldots, y_N) && (5) \\ &= \frac{N}{V} \int dy_1 \delta(x - y_1(t)) f_1(y_1, t) && (6) \\ &= \frac{N}{V} f_1(x,t), && (7) \end{aligned}$$

where V is the systems volume in $\boldsymbol{r}$-space. This equation defines $f_1(x,t)$ which is the probability that a particle will be found at x at time t, regardless of where all the other particles in the system are located. Many of the interesting macroscopic properties of the system can be determined from f_1, so we would like to solve for it. Since $dN^M/dt = 0$ we get

$$\frac{\partial N^M}{\partial t} + \frac{\partial \boldsymbol{r}}{\partial t} \cdot \frac{\partial N^M}{\partial \boldsymbol{r}} + \frac{\partial \boldsymbol{p}}{\partial t} \cdot \frac{\partial N^M}{\partial \boldsymbol{p}} = 0$$

or

$$\frac{\partial N^M}{\partial t} + \boldsymbol{v} \cdot \frac{\partial N^M}{\partial \boldsymbol{r}} + \boldsymbol{F} \cdot \frac{\partial N^M}{\partial \boldsymbol{p}} = 0.$$

Taking the ensemble average of this equation gives

$$\frac{\partial}{\partial t}\langle N^M \rangle + \boldsymbol{v} \cdot \frac{\partial}{\partial \boldsymbol{r}}\langle N^M \rangle + \langle \boldsymbol{F} \cdot \frac{\partial N^M}{\partial \boldsymbol{p}} \rangle = 0.$$

Writing $\boldsymbol{F}^M = \langle \boldsymbol{F}^M \rangle + \delta \boldsymbol{F}^M$ and $\langle N^M \rangle = (N/V) f_1$ gives

$$\frac{\partial f_1}{\partial t} + \boldsymbol{v} \cdot \frac{\partial f_1}{\partial \boldsymbol{r}} + \langle \boldsymbol{F}^M \rangle \cdot \frac{\partial f_1}{\partial \boldsymbol{p}} = -\langle \delta \boldsymbol{F}^M \cdot \frac{\partial}{\partial \boldsymbol{p}} \delta N^M \rangle.$$

If we set the right hand side of this equation, which is often referred to as the collision integral, to zero we get the Vlasov equation. This depends only on the ensemble averaged field $\langle \boldsymbol{F}^M \rangle$. It does not incorporate any effects due to the granularity which is present in any real micro-state. If we wish to include effects due to this granularity, we need to include the right hand side term which depends on the ensemble average of correlations in the fluctuations of the micro-states. For plasmas the simplest non-zero approximation for this term produces the Landau equation. A yet more detailed approximation leads to the Balescu-Lenard equation which incorporates the dynamic polarization effect known as Debye shielding.

What does $\delta \boldsymbol{F}^M$ look like for systems in which the particles interact through an r^{-2} force?

Consider a distribution of stars, all of mass m, with number density $n(r, \theta, \phi)$ in spherical coordinates. Let us assume for simplicity that n has no significant r dependence. The gravitational force on a star at $r = 0$ due to the stars in a volume element $r^2 dr \sin\theta d\theta d\phi$ at $\boldsymbol{r}$ is

$$m^2 G n(\theta, \phi) r^2 dr \sin\theta d\theta d\phi \;\; \hat{\boldsymbol{r}} \;\; \frac{1}{r^2},$$

where $\hat{\boldsymbol{r}}$ is a unit vector in the radial direction. The force on the star due to all the stars in a shell of thickness dr and radius r is

$$d\langle \boldsymbol{F}^M \rangle = dr \left[\int n(\theta, \phi) \hat{\boldsymbol{r}} \sin\theta d\theta d\phi \right] m^2 G.$$

The term in the square brackets has no dependence on distance. So in this idealized case all shells of a given thickness contribute equally to the force on the star at $r = 0$. Since there are more shells at large r than small r, distant interactions will dominate $\langle \boldsymbol{F}^M \rangle$. It is this behavior which introduces long wavelength collective modes to the system.

The fluctuation term $\delta \boldsymbol{F}^M$ has a different r dependence. The contribution to the fluctuations from a given volume element scales as the square root of the number of stars in that volume element,

$$m^2 G \sqrt{n(\theta,\phi) r^2 dr \sin\theta d\theta d\phi} \;\; \hat{\boldsymbol{r}} \;\; \frac{1}{r^2},$$

$$= \left[m^2 G \sqrt{n(\theta,\phi) \sin\theta d\theta d\phi} \;\; \hat{\boldsymbol{r}} \right] \frac{\sqrt{dr}}{r}.$$

Summing over θ and ϕ gives the contribution $d(\delta \boldsymbol{F}^M)$ from the shell dr at r. Once again the factor in the square brackets is independent of distance so

$$d(\delta \boldsymbol{F}^M) \propto \frac{\sqrt{dr}}{r}.$$

In a plasma close to equilibrium, Debye shielding adds an exponential factor e^{-r/λ_d} to this behavior causing even faster fall off with distance. λ_d is the Debye length.

Our conclusion from this crude hand-waving argument is that fluctuations in the force are a short scale phenomena. If they are significant then we will need to model them accurately in our simulation. This means resolving the interactions between close neighbors. We have seen however that PM codes are very inefficient for this type of system. On the other hand, if fluctuations in the force are not important, the system evolves under the influence of the long range ensemble average $\langle \boldsymbol{F}^M \rangle$. PM codes accurately represent forces on scales greater than a few mesh cell sizes, so we can expect them to do a good job for these 'collisionless' systems, for which they can be used to study collective modes in the wavelength range $\Delta \leq \lambda \leq N_g^{1/d} \Delta$.

How do we determine whether force fluctuations are important in a real plasma or gravitational N-body system? The more particles that are involved in representing a given mass or charge distribution, the less granular the resulting force field will be and the less significant $\delta \boldsymbol{F}^M$ will become. In an homogeneous plasma only those particles within a distance λ_d contribute to $\delta \boldsymbol{F}^M$. Therefore as $N_d = n\lambda_d^3$ increases, the plasma becomes more collisionless. We can assume

$$N_d >> 1$$

as a condition for collisionless behavior.

For gravitational N-body systems the criterion is less clear-cut. Imagine a galaxy of radius R with N stars. For simplicity let us assume that within the galaxy the distribution of stars is uniform. For a star at the center of the galaxy the mean force (ie. the ensemble averaged force) is zero. Only the fluctuations persist. If we sum the individual binary interactions of a test star with all the stars in the galaxy, assuming each interaction can be considered independent, we can derive an expression for the rate at which the test star is deflected from its trajectory. Using Rutherford scattering theory we derive an expression for $\Delta v_\perp^2$ due to a typical interaction with a field star. Here $\perp$ denotes the plane perpendicular to the initial velocity of the test star. Then we sum this expression over all possible interactions as the star crosses the galaxy once.[3] The result is

$$\frac{\Delta v_\perp^2}{v^2} \sim \frac{8 \ln N}{N}.$$

So the condition we need to satisfy is that $\Delta v_\perp^2 / v^2 << 1$, ie.

$$\frac{8 \ln N}{N} << 1.$$

This is a very crude and idealized estimate, but it gives us a more quantitative criterion for deciding if PM is an appropriate technique for a gravitational N-body problem.[4] From this expression it is clear that galaxies ($N \sim 10^{11}$) are collisionless, globular clusters ($N \sim 10^4 - 10^6$) are marginally collisionless, while stellar clusters ($N \sim 10^2$) are dominated by fluctuations.

There is a caveat to bear in mind with regard to gravitational systems. It is possible that some stars in the system might be unaffected by fluctuations while at the same instant other stars are being dominated by fluctuations. This occurs because of spatial clustering which tends to develop in gravitational systems but not in plasmas. Conditions in the neighborhoods of two spatially separated stars can be significantly different.

In both plasmas and gravitational N-body systems the particles interact with each other through a r^{-2} force, and so we might expect them to be similar in nature. However in the gravitational case the forces are always attractive. In plasmas the interactions between particles can be either attractive or repulsive. As a result these systems have major differences.

[3] This of course violates our assumption that the star is at the center of the galaxy, but our purpose here is to derive a crude estimate for which we consider this approximation acceptable.

[4] Less crude criteria can be developed from the Landau equation. See chapter 8 of Binney and Tremaine [11].

Gravitational N-body systems are subject to a collapse instability which does not occur for plasmas. When a density perturbation develops, the result is to more strongly attract surrounding material toward the denser region, reinforcing the original perturbation. Plasma density perturbations do not feed themselves in this way.

The result is that gravitational systems generally possess very high density contrast, with particles distributed in clusters which may be part of a hierarchy of clusters. Plasmas tend to be much more uniform.

In addition plasmas are affected by a phenomenon known as Debye shielding. Every particle in the plasma has a tendency to attract those particles in its neighborhood with charge of opposite sign and repel particles with charge of similar sign. This weakly correlated behavior has the effect of screening fluctuations on scales longer than the Debye length λ_d.

1.7 Key features of a collisionless plasma

Most of the pioneering studies of the properties of PM schemes were collisionless plasma calculations. So let us take a moment here to review some of the key features of collisionless plasmas in light of the points made in the previous section.

The interaction force(r^{-2}) is a long range force. In principle and in practice the system supports long wavelength modes involving coherent collective motion of many particles.

The highest characteristic frequency associated with collective modes is the electron plasma frequency

$$\omega_{pe} = \left(\frac{4\pi n e^2}{m_e}\right)^{\frac{1}{2}}.$$

The range of wavelengths of these coherent modes is bounded at the lower end by the Debye length λ_d. Electron thermal motions dampen modes with $\lambda < \lambda_d$ so strongly (Landau Damping), that we can say collective modes only exist for $\lambda > \lambda_d$.

Particles separated by less than λ_d see each other as individuals - particles separated by more than λ_d interact with each other as participants in collective wave modes.

The collision frequency $\nu_c(\equiv \tau_{coll})$ is the rate at which sub-λ_d fluctuations scatter a particle.

$$\frac{2\pi\nu_c}{\omega_{pe}} \propto \frac{n\ln\Lambda}{\sqrt{n}} \propto \frac{1}{N_d}\ln N_d$$

ie.

$$\frac{2\pi\nu_c}{\omega_{pe}} << 1 \rightarrow N_d >> 1$$

In collisionless systems, the $\lambda > \lambda_d$ coherent collective modes are the objects we wish to study.

A collisionless system has a very large number of particles in a Debye sphere. A collisional (fluctuation affected) system has relatively few. To properly model a collisional system we must faithfully reproduce N_d. For a collisionless system we can ignore sub-λ_d fields. The challenge there is to make sure that while grossly under-representing the number of particles we maintain

$$\frac{2\pi\nu_c}{\omega_{pe}} << 1.$$

One way to achieve this is by filtering out the sub-λ_d scale components of the electric field. Since $\lambda_{min} = 2\Delta$ is the shortest wavelength which the grid will support, we can arrange this if we set $\Delta \sim \lambda_d$.

2 ES1 - A 1D Electrostatic Code

ES1 is a well-known and documented 1D electrostatic code, written by Birdsall and Langdon, which is widely used as a teaching aid. For a more complete description of the code and a variety of simulation projects to which it can be applied, the reader is referred to reference [4]. At this point we shall present a quick outline.

2.1 Integration of the Equations of Motion

The particle equations of motion are

$$\frac{dx_i}{dt} = v_i$$

$$\frac{dv_i}{dt} = \frac{F_i}{m_i}$$

ES1 uses a leap-frog scheme which is second order accurate in time for constant integration timestep Δt. The discrete equations are

$$m_i \frac{v_i^{n+1/2} - v_i^{n-1/2}}{\Delta t} = F_i^n \qquad (8)$$

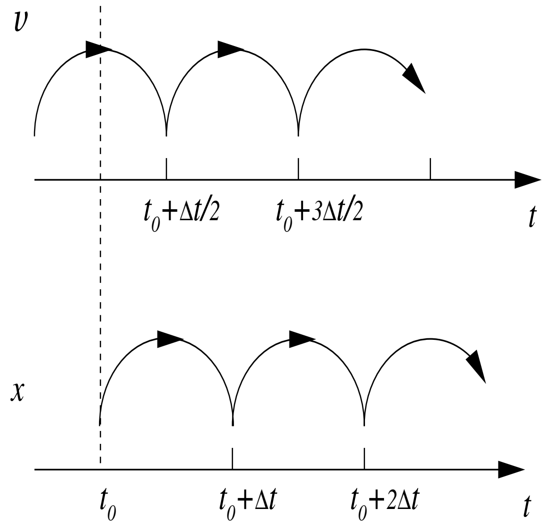

Figure 5: The leap frog time integration. Note that the times at which v and x are known are offset by $\Delta t/2$.

$$\frac{x_i^{n+1} - x_i^n}{\Delta t} = v_i^{n+1/2} \tag{9}$$

where superscripts denote time levels. This is illustrated in figure 5. Note that the times at which a particles position are known are offset by $\Delta t/2$ from the times at which its velocity is known. For an harmonic oscillator (ie. $F \propto x - x_0$) of frequency ω_0, leap frog has no amplitude error for $\omega_0 \Delta t \leq 2$, and has second order phase errors. Thus choosing Δt to satisfy $\omega_0 \Delta t = .3$ gives reasonable accuracy provided we do not run the integration beyond roughly $100\Delta t$.

2.2 Integration of the Field Equations

ES1 solves the 1D form of Poisson's equation

$$\frac{\partial^2}{\partial x^2}\phi = -\rho(x) \tag{10}$$

and computes the electric field using

$$E_x = -\frac{\partial \phi}{\partial x}. \tag{11}$$

Taking fourier transforms of these equations gives

$$\hat{\phi}(k) = \frac{\hat{\rho}(k)}{k^2} \tag{12}$$

and

$$\hat{E}(k) = -ik\hat{\phi}(k). \tag{13}$$

For a discrete periodic system of length L, the fourier transform and inverse transform are defined by

$$\hat{f}(k_l) = \Delta x \sum_{g=0}^{N_g-1} f(x_g) e^{-ik_l x_g}$$

$$f(x_g) = \frac{1}{L} \sum_{l=0}^{N_g-1} \hat{f}(k_l) e^{ik_l x_g}$$

where $k_l = 2\pi l/L$. We could form the discrete transform $\hat{\rho}(k_l)$ and then use equations (12) and (13) to find $\hat{E}(k_l)$, and finally apply the inverse transform to get $E(x_g)$. However this mixes discrete representations $\hat{\rho}(k_l)$ and $\hat{E}(k_l)$ with continuous representations k^2 and $-ik$ for the operators $\partial^2/\partial x^2$ and $\partial/\partial x$. [5] So instead we replace equations (10) and (11) with their finite difference approximations

$$\frac{\phi_{g+1} - 2\phi_g + \phi_{g-1}}{\Delta x^2} = -\rho_g.$$

and

$$E_g = -\frac{\phi_{g+1} - \phi_{g-1}}{2\Delta x}$$

where Δx is the mesh cell size. Taking discrete transforms of these finite difference approximations gives

$$\hat{\phi}(k_l) = \frac{\hat{\rho}(k_l)}{\mathrm{K}_l} \tag{14}$$

and

$$\hat{E}(k_l) = -i\kappa_l \hat{\phi}(k_l). \tag{15}$$

where

$$\kappa_l = k_l \frac{\sin k_l \Delta x}{k_l \Delta x}$$

and

$$\mathrm{K}_l = k_l^2 \left(\frac{\sin \frac{1}{2} k_l \Delta x}{\frac{1}{2} k_l \Delta x} \right)^2.$$

[5] We will explore this further in section 4.7

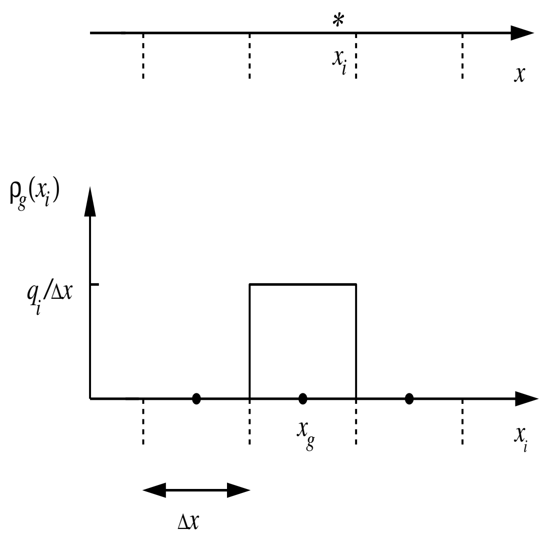

Figure 6: The variation of charge from particle i assigned to mesh point g as the particle's location x_i is altered, for NGP assignment.

2.3 Assignment of Particle Charge to the Mesh

We are given two choices by ES1. The lowest order and least expensive is Nearest Grid Point(NGP). In this case all of a particles charge is assigned to the mesh cell in which the particle is located, as shown in figure 6. As a particle crosses the boundary between two cells, the charge assigned to both cells jumps discontinuously. This produces some noise in both ρ and E.

The higher order alternative provided is Cloud-In-Cell(CIC). In this case each particle can be considered as a uniform charge cloud of width w. The default width is $w = \Delta x$, the mesh spacing. This scheme is illustrated in figure 7. If the particle position x_i is located between the two mesh cell centers x_{g-1} and x_g, the charge assigned to each of these cells is given by

$$\rho_g = q_i \frac{x_i - x_{g-1}}{\Delta x}$$

and

$$\rho_{g-1} = q_i \frac{x_g - x_i}{\Delta x}.$$

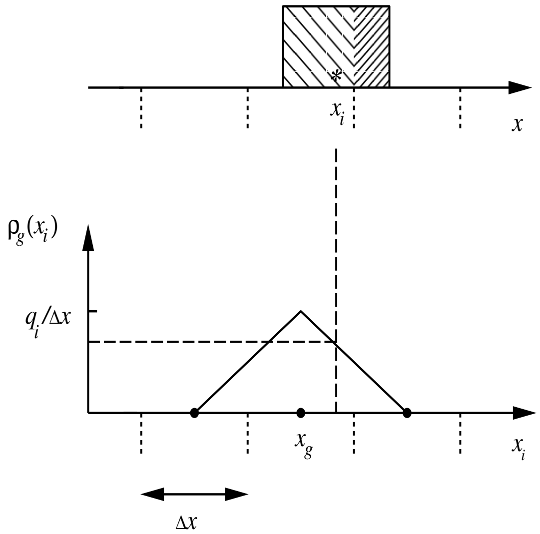

Figure 7: The variation of charge from particle i assigned to mesh point g as the particle's location x_i is altered, for CIC assignment.

2.4 Force Evaluation

To ensure momentum conservation the same interpolation scheme is used to compute the force on a particle as was used to perform the assignment of the particles charge to the mesh. If we used NGP then the force on the particle is simply the force evaluated at the mesh point nearest to the particle. For CIC it is given by the formula

$$F_i = q_i \left(\frac{(x_i - x_{g-1})}{\Delta x} E_g + \frac{(x_g - x_i)}{\Delta x} E_{g-1} \right)$$

for $x_{g-1} \leq x_i \leq x_g$. There is also an energy conserving option which uses CIC charge assignment and NGP force evaluation. However this does not conserve momentum.

3 Time Integration Schemes

In this section we provide a brief review of the properties of a number of the most frequently used time integration algorithms. We have followed the format adopted in chapter 4 of Hockney and Eastwood [1] to which the reader is referred for more detail. We will consider three algorithms of different order, the first order Euler scheme, also known as upwind differencing, the second order leapfrog algorithm which we have already encountered, and a fourth order Runge-Kutta algorithm. For the sake of brevity we will attempt to derive the properties of the leapfrog scheme only, while simply quoting the corresponding results for the two other schemes.

Applying the Euler scheme to the particle equations of motion gives

$$\frac{v_i^{n+1} - v_i^n}{\Delta t} = \frac{F_i^n}{m_i} \tag{16}$$

$$\frac{x_i^{n+1} - x_i^n}{\Delta t} = v_i^n. \tag{17}$$

The leapfrog scheme was given in equations (8) and (9). The fourth order Runge-Kutta scheme for a system of equations given by

$$\frac{d\boldsymbol{z}}{dt} = \boldsymbol{f}(\boldsymbol{z}, t)$$

where $\boldsymbol{z}$ and $\boldsymbol{f}$ are vectors (ie. in our case $\boldsymbol{z} = (x, v)$), is defined as

$$\boldsymbol{z}^{n+1} = \boldsymbol{z}^n + \frac{1}{6}(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4) + O(\Delta t^5)$$

with

$$\begin{aligned}
\boldsymbol{k}_1 &= \Delta t \boldsymbol{f}(\boldsymbol{z}^n, t^n) \\
\boldsymbol{k}_2 &= \Delta t \boldsymbol{f}(\boldsymbol{z}^n + \frac{1}{2}\boldsymbol{k}_1, t^n + \frac{1}{2}\Delta t) \\
\boldsymbol{k}_3 &= \Delta t \boldsymbol{f}(\boldsymbol{z}^n + \frac{1}{2}\boldsymbol{k}_2, t^n + \frac{1}{2}\Delta t) \\
\boldsymbol{k}_4 &= \Delta t \boldsymbol{f}(\boldsymbol{z}^n + \frac{1}{2}\boldsymbol{k}_3, t^n + \frac{1}{2}\Delta t).
\end{aligned}$$

There are four important criteria to be considered when choosing an algorithm to integrate the particle equations of motion,

- convergence
- accuracy
- stability
- efficiency.

3.1 Convergence

By convergence we mean that the numerical solution converges to the exact solution of the differential equation in the limits as Δt and Δx tend to zero. For linear schemes Lax has shown that consistency and stability are necessary and sufficient conditions for convergence. Consistency requires that the difference approximation should reduce to the differential equations in the limit as $\Delta t \to 0$. It should also preserve time reversibility. All three schemes reduce to the correct differential equations in the limit as $\Delta t \to 0$. However of the three only leapfrog satisfies the time reversibility requirement.

3.2 Accuracy

By accuracy we mean the truncation error associated with approximating derivatives with differences. If we combine the two equations defining the leapfrog approximation into one we get

$$\frac{x^{n+1} - 2x^n + x^{n-1}}{\Delta t^2} = \frac{F(x^n)}{m} \tag{18}$$

which can be compared with the exact expression

$$\frac{\partial^2 x}{\partial t^2} = \frac{F}{m}.$$

Using Taylor series expansions for x^{n+1} and x^{n-1} about x^n gives

$$x^{n+1} = x^n + \Delta t \frac{\partial x}{\partial t}\bigg|_n + \frac{1}{2!}\Delta t^2 \frac{\partial^2 x}{\partial t^2}\bigg|_n + \frac{1}{3!}\Delta t^3 \frac{\partial^3 x}{\partial t^3}\bigg|_n + O(\Delta t^4)$$

$$x^{n-1} = x^n - \Delta t \frac{\partial x}{\partial t}\bigg|_n + \frac{1}{2!}\Delta t^2 \frac{\partial^2 x}{\partial t^2}\bigg|_n - \frac{1}{3!}\Delta t^3 \frac{\partial^3 x}{\partial t^3}\bigg|_n + O(\Delta t^4).$$

Combining these gives

$$\frac{x^{n+1} - 2x^n + x^{n-1}}{\Delta t^2} = \frac{\partial^2 x}{\partial t^2}\bigg|_n + O(\Delta t^2).$$

Thus leapfrog is second order accurate in time.

It is relatively trivial to show that the Euler scheme is only first order. Showing that the Runge-Kutta scheme is fourth order is a very long and tedious task which will be left up to those readers with superhuman patience. It should be pointed out that order is generally but not always a reliable guide to the accuracy of a scheme. Each scheme has its complement of pathological applications which can cause it to break down.

3.3 Stability

Do errors grow in time? If round-off error (ie the error introduced because the computer only stores numbers up to a certain precision) grows in time then the scheme is unstable.

Consider again the leapfrog equation (18). Let x^n be the numerical solution at time t^n, and X^n be the exact solution (ie no round-off error) to this difference equation. We shall denote the numerical error at time t^n by

$$\epsilon^n = x^n - X^n. \tag{19}$$

Using equation (19) to replace x in equation (18) gives us an equation for the evolution of the error ϵ with time,

$$\begin{aligned}
\epsilon^{n+1} - 2\epsilon^n + \epsilon^{n-1} &= \frac{\Delta t^2}{m}\Big(F(X^n + \epsilon^n) - F(X^n)\Big) && (20)\\
&\simeq \frac{\Delta t^2}{m}\frac{\partial F}{\partial X}\bigg|_{X^n}\epsilon^n. && (21)
\end{aligned}$$

If we assume that we are looking at bounded oscillatory solutions of the form

$$\epsilon^n = (\lambda)^n = (e^{i\omega\Delta t})^n$$

then by substituting this into the previous equation we get

$$\lambda^2 - 2\lambda + 1 = -(\Omega\Delta t)^2\lambda \tag{22}$$

where we assume also that

$$\frac{1}{m}\frac{\partial F}{\partial X}\bigg|_{X^n} = -\Omega^2.$$

This has solutions

$$\lambda_\pm = 1 - \frac{1}{2}\Omega^2\Delta t^2\left(1 \mp \sqrt{1 - \frac{4}{\Omega^2\Delta t^2}}\right)$$

and the general solution is

$$\epsilon^n = a\lambda_+^n + b\lambda_-^n.$$

The scheme will be stable provided $|\lambda_\pm| \leq 1$. Figure 8 shows how $\lambda_\pm$ varies as a function of $\Omega\Delta t$. When $\Omega\Delta t < 2$, $\lambda_\pm$ has an imaginary part, but

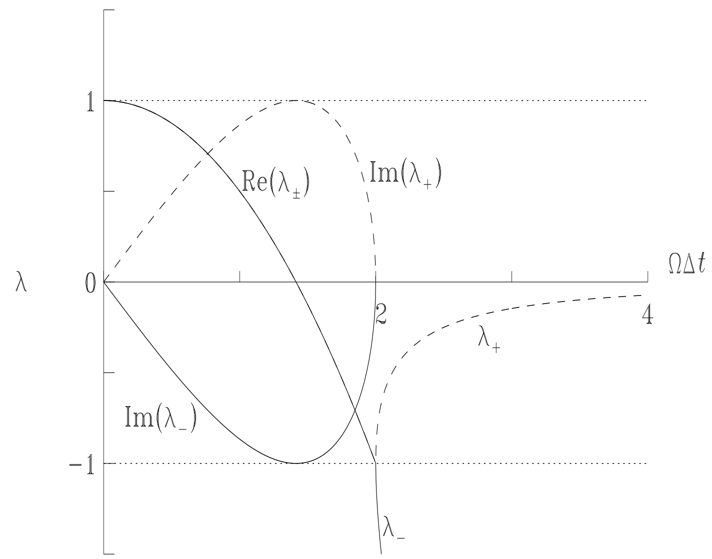

Figure 8: The roots of equation (22) as a function of $\Omega\Delta t$. For $\Omega\Delta t < 2$ both roots have an imaginary component, but both have magnitude $|\lambda| = 1$. For $\Omega\Delta t > 2$, both roots are real and $|\lambda_-| > 1$.

for $\Omega\Delta t \geq 2$ both solutions are real. For $\Omega\Delta t \leq 2$ we can easily show that $|\lambda_\pm| = 1$. This means that not only is the leapfrog scheme stable for $\Omega\Delta t \leq 2$, but it has the additional advantage that it suffers no amplitude dissipation. When $\Omega\Delta t > 2$ however $|\lambda_-| > 1$. Therefore to guarantee stability, we can calculate the largest value of $|m^{-1}\partial F/\partial X|$ and then set Δt such that

$$\left|\frac{1}{m}\frac{\partial F}{\partial X}\right|^{\frac{1}{2}} \Delta t < 2.$$

The stability equation for the Euler scheme as given in equations (16) and (17) has roots

$$\lambda_\pm = 1 \pm i\Omega\Delta t$$

which has $|\lambda_\pm| \geq 1$, and so is unconditionally unstable. The Runge-Kutta scheme is also unstable when applied to the particle equations of motion.

When we make the simplifying assumption that the force acting on the particle is proportional to the particle displacement, ie simple harmonic oscillation, the stability equation has solution

$$\lambda_{\pm} = 1 - \frac{5}{12}(\Omega\Delta t)^2 + \frac{1}{48}(\Omega\Delta t)^4 \pm \Omega\Delta t\Big(1 - \frac{3}{24}(\Omega\Delta t)^2\Big).$$

In this case $|\lambda_{\pm}| > 1$ for all $\Omega\Delta t$ except in a short interval about $\Omega\Delta t \simeq 2.5$.

This analysis is more complicated for non-oscillatory solutions. There we need to check that ϵ^n grows more slowly than X^n.

There is of course another requirement for accurate stable integration of the equations of motion. No particle can be allowed to move more than one mesh cell in distance during one timestep.

3.4 Efficiency

This is a critical consideration since whatever scheme we choose will be used for each particle at each timestep, a total which can be comfortably in excess of 10^{10} times. It is generally true of a lower order scheme that they

- involve fewer intermediate time levels per timestep
- require fewer stored intermediate values
- require fewer floating point operations per timestep
- have a greater stability range
- require finer timesteps to achieve the same accuracy

compared with a higher order scheme. The conventional wisdom is that the simple second order leapfrog achieves the best balance between accuracy, stability and efficiency. In ES1, with normalizations such that $\Delta t = 1$, and with the variable replacements

$$v \to \overline{v} = v\frac{\Delta t}{\Delta x}$$

$$x \to \overline{x} = \frac{x}{\Delta x}$$

then

$$\overline{v}_{new} = \overline{v}_{old} + A(\overline{x}_{old})$$

$$\overline{x}_{new} = \overline{x}_{old} + \overline{v}_{old}$$

where

$$A(\overline{x}_{old}) = \alpha E(\overline{x}_{old})$$

with $\alpha = q\Delta t^2/m\Delta x$. This implementation is remarkably efficient requiring just 4 fetches from memory, 2 additions, and 2 stores in memory per particle per timestep.

4 Spatial Discretization

The mesh is involved in three separate stages of the code,

1. assignment of the particle charge to the mesh
2. solving the field equations
3. interpolating mesh defined forces to the particle positions.

All three steps introduce some error. However we should remember that the only error that matters is the final combination of errors. As we shall see it is sometimes possible to manage these individual errors in ways which allow them to partially cancel each other and so produce a superior result.

In discussing the ramifications of spatial discretization we will limit ourselves to the two schemes which we have seen already, NGP and CIC. To illustrate discretization errors we will consider a simple test problem, a periodic 1D electrostatic system with boundaries at $x = \pm L/2$, and a uniform mesh with cell size Δx, and we shall assume the following discretized field equations,

$$\frac{\phi_{g+1} - 2\phi_g + \phi_{g-1}}{\Delta x^2} = -\rho_g$$

$$E_g = -\frac{\phi_{g+1} - \phi_{g-1}}{2\Delta x}.$$

Before we proceed there are a couple of function definitions which we need to establish. We define a cloud shape function $S(x)$ which gives the charge density associated with a finite-sized particle at $x = 0$. For NGP this is

$$S(x) = \frac{1}{\Delta x}\delta\Big(\frac{x}{\Delta x}\Big)$$

and for CIC,

$$S(x) = \frac{1}{\Delta x}\Pi(x).$$

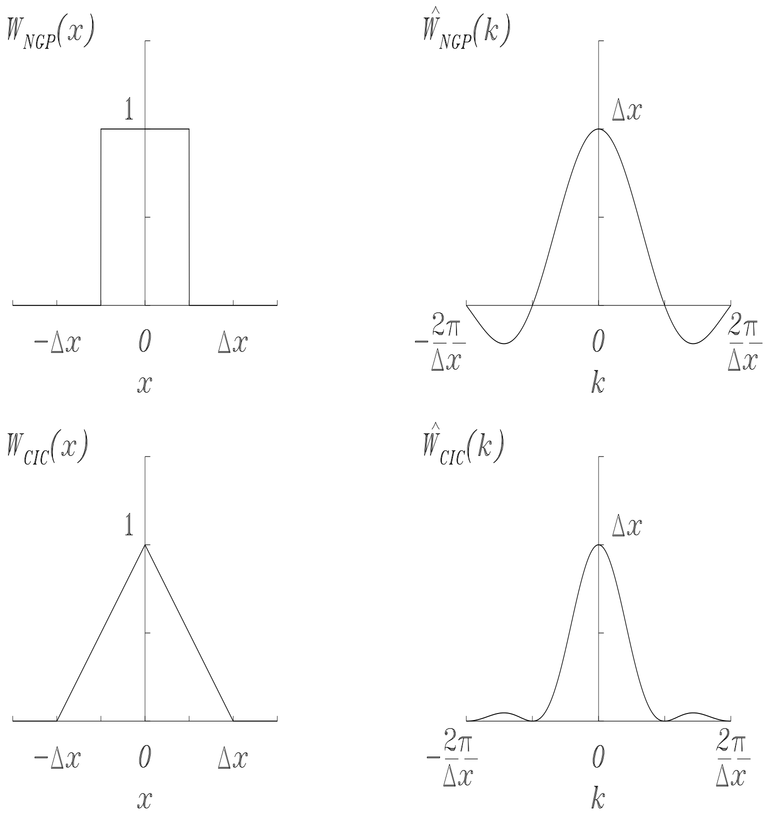

Figure 9: The weighting functions and their fourier transforms used in NGP and CIC schemes. The NGP weighting function $W_{NGP}(x)$ is the hat function $\Pi(x)$, and the CIC weighting function $W_{CIC}(x)$ is the triangle function $\bigwedge(x)$.

Π is the hat function shown in figure 9 and defined in equation (23). The charge assigned to mesh point g from particle i located at x_i is computed from the weighting function

$$W(x_i - x_g) = \int_{x_g - \frac{\Delta x}{2}}^{x_g + \frac{\Delta x}{2}} S(x_i - x')dx'.$$

The total charge at g is

$$\rho_g = \frac{q}{\Delta x} \sum_{i=1}^{N_p} W(x_i - x_g).$$

If the number density of particle centers is

$$n(x) = \sum_{i=1}^{N_p} \delta(x - x_i)$$

we can define a continuous charge density ρ_c by analogy

$$\rho_c = \frac{q}{\Delta x} \int_{-L/2}^{L/2} W(x' - x) n(x') dx'.$$

Note that $\rho_g = \rho_c(x_g)$, in other words ρ_g can be obtained by sampling the continuous density distribution ρ_c.

4.1 NGP

For NGP the weighting function is given by

$$W(x) = \Pi(x) = \begin{cases} 1 & \mid x \mid \leq \Delta x/2 \\ \\ 0 & \mid x \mid > \Delta x/2. \end{cases} \tag{23}$$

The mesh charge density is obtained by sampling the continuous density

$$\rho_g = \rho_c(x)|_{x_g} = \frac{q}{\Delta x} \int_{-L/2}^{L/2} W(x' - x_g) n(x') dx'.$$

where

$$n(x) = \sum_{i=1}^{N_p} \delta(x - x_i).$$

The force on particle i is given by the force evaluated at the nearest cell center. If $x_g - \Delta x/2 \leq x_i \leq x_g + \Delta x/2$ then

$$\begin{aligned} F(x_i) &= q_i E(x_g) && (24) \\ &= q_i \sum_{g=0}^{N_g - 1} W(x_i - x_g) E_g. && (25) \end{aligned}$$

Now, consider our simple test problem, a 1D electrostatic system with periodic boundary conditions imposed at $x = \pm L/2$, and just two particles, a charge $-q$ at x_1, and a charge $+q$ at x_2, with $x_2 > x_1$. We shall label

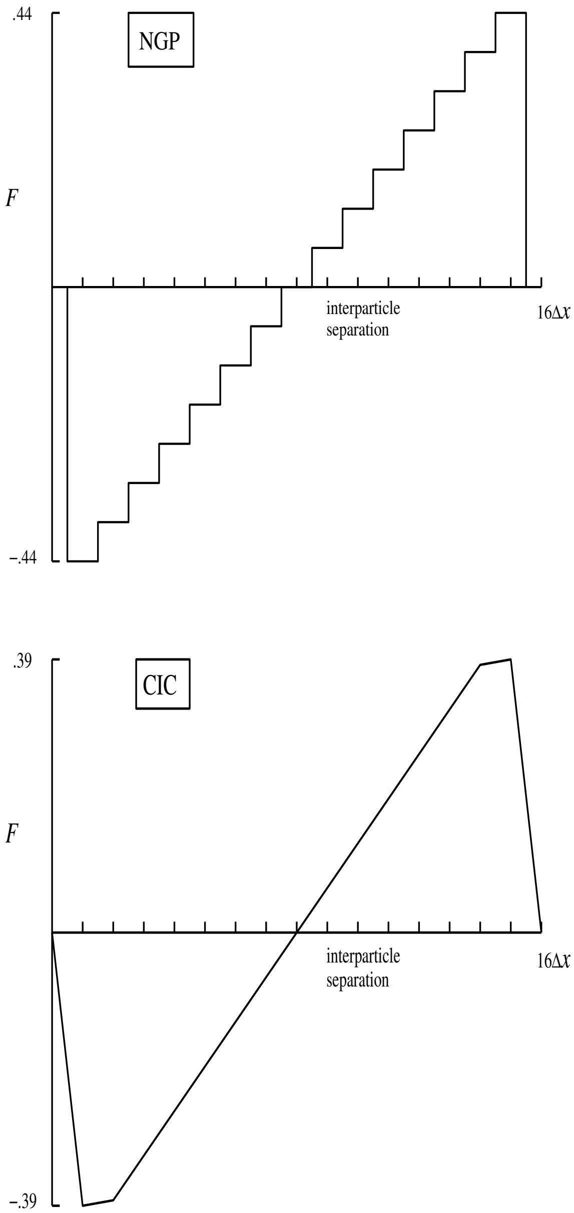

Figure 10: The mutual force between a positive and negative charge as a function of their spatial separation, in a periodic system of length $16\Delta x$, using NGP(top frame) and CIC(bottom frame) charge assignment and force evaluation.

the center of the mesh cell containing the negative charge $\overline{x}_1$, and the center of the mesh cell containing the positive charge $\overline{x}_2$. Using NGP charge assignment, the potential at the mesh points is given by

$$\phi_g = \begin{cases} \beta(x_g + L/2) & x_g \leq \overline{x}_1 \\ \beta(x_g + L/2) + q(x_g - \overline{x}_1) & \overline{x}_1 < x_g < \overline{x}_2 \\ \beta(x_g + L/2) + q(x_g - \overline{x}_1) - q(x_g - \overline{x}_2) & \overline{x}_2 \leq x_g \end{cases}$$

where $\beta = q(\overline{x}_1 - \overline{x}_2)/L$. The force on the particle at x_2 is

$$F_2 = q^2\Big(\frac{\overline{x}_2 - \overline{x}_1}{L} - \frac{1}{2}\Big).$$

This is plotted in figure 10 as a function of the inter-particle separation $x_2 - x_1$, for a system with $L = 16\Delta x$. Note that as we vary the particle separation the force jumps discontinuously when one of the particles crosses a mesh cell boundary. Another unfortunate property of this force is that it is not invariant under spatial translation. Suppose we vary $(x_1 + x_2)/2$, the mean position of the particles, while keeping their separation $x_2 - x_1$ fixed. Then the force fluctuates with period Δx, as shown schematically in figure 11. This fluctuation in the inter-particle force is greatest at small separations. Minimizing this loss of displacement invariance is one of the most significant improvements we can make in designing a particle code. As we shall now see using a higher order scheme, such as CIC, achieves this.

4.2 CIC

With NGP we used one mesh point to define charge assignment and force interpolation, and we got a noisy result. CIC uses two mesh points. In this case the particle can be viewed as a uniform density charge cloud of width Δx. The charge assignment scheme is illustrated in figure 7. The weighting function is given by

$$W(x_i - x_g) = \bigwedge(x_i - x_g) = \begin{cases} 1- \mid x_i - x_g \mid /\Delta x & \mid x_i - x_g \mid \leq \Delta x \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

$\bigwedge(x)$ is called the triangle function, and is shown in figure 9. We will leave it as an exercise to the reader to derive the electric potential on the mesh

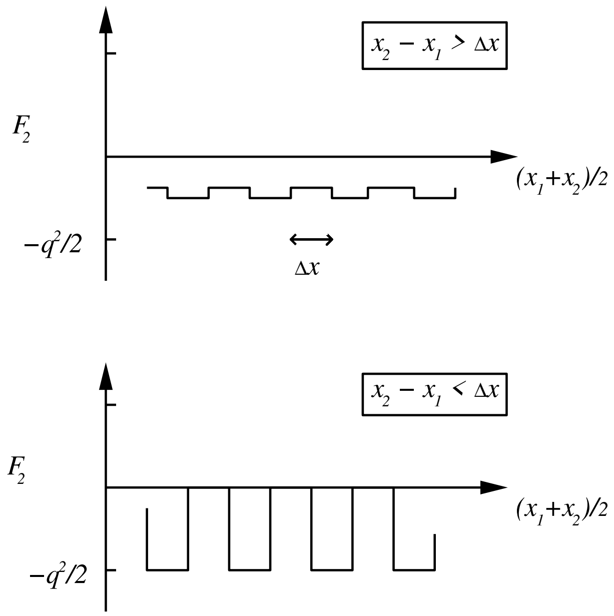

Figure 11: Variation of the mutual force between a positive and negative charge as a function of their mean position but with fixed separation, for the two distinct cases when the particles are separated by more than Δx(top), and by less than Δx(bottom). NGP charge assignment and force evaluation are used.

for the test problem considered in the previous section. An example of the interaction force between the two particle is plotted in the bottom frame of figure 10 as a function of the inter-particle separation. For $x_2 - x_1 \geq 2\Delta x$ we recover the exact analytic answer for the 1D problem.[6] For $x_2 - x_1 < 2\Delta x$, the inter-particle force depends also on the positions of the particles with respect to the mesh. Figure 10 illustrates one example, where x_1 was chosen to be $-0.4\Delta x$ from cell center while x_2 was varied.

It is immediately apparent by comparing the two frames in figure 10 that CIC gives much smoother forces than NGP. It is also obvious that the dependence of the inter-particle force on the mean particle position $(x_1 + x_2)/2$ is much weaker than in NGP. Finally, the errors that do remain are more localized spatially than for NGP.

[6] This occurs because in 1D the exact analytic solution for the potential is piece-wise linear in x. CIC interpolation is capable of representing this variation exactly. This fortuitous match does not occur in 2 or 3D.

4.3 Momentum Conservation

Both the NGP and CIC examples above conserve momentum. We can in fact show that momentum will be conserved provided we use centered differencing, and we use the same weighting function W for charge assignment and force interpolation. To prove that a scheme conserves momentum we need to demonstrate

- that no self-forces act on the particles
- that interacting particles impose equal but opposite forces upon each other.

It is convenient at this point to develop some additional array notation. Let us consider a vector $\boldsymbol{\rho}$ such that each element of $\boldsymbol{\rho}$ is the charge density at a different mesh point, ie $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_g, \ldots, \rho_{N_g})$. Let us also construct similar vectors $\boldsymbol{\Phi}$ and $\boldsymbol{\varepsilon}$ from the mesh values of the electric potential and electric field. Differencing Poisson's equation leads to a matrix equation

$$\boldsymbol{A}\boldsymbol{\Phi} = -\boldsymbol{\rho}$$

where $\boldsymbol{A}$ is an $N_g \times N_g$ matrix. Similarly, differencing

$$E = -\frac{\partial \phi}{\partial x}$$

produces

$$\boldsymbol{\varepsilon} = -\boldsymbol{B}\boldsymbol{\Phi}$$

where $\boldsymbol{B}$ is another $N_g \times N_g$ matrix. Combining these equations gives

$$\begin{aligned} \boldsymbol{\varepsilon} &= \boldsymbol{B}\boldsymbol{A}^{-1}\boldsymbol{\rho} && (27) \\ &= \boldsymbol{C}\boldsymbol{\rho}. && (28) \end{aligned}$$

Choosing the differencing scheme

$$\frac{\partial^2 \phi}{\partial x^2} \qquad \rightarrow \qquad \frac{\phi_{g+1} - 2\phi_g + \phi_{g-1}}{\Delta x^2}$$

means $\boldsymbol{A}$ and $\boldsymbol{A}^{-1}$ are symmetric. With

$$\frac{\partial \phi}{\partial x} \qquad \rightarrow \qquad \frac{\phi_{g+1} - \phi_{g-1}}{2\Delta x}$$

$\boldsymbol{B}$ is anti-symmetric. Therefore $\boldsymbol{C} = \boldsymbol{B}\boldsymbol{A}^{-1}$ is also anti-symmetric.

4.3.1 Self-Forces

Consider a charge q at x. Let us assume for now that the weighting functions used to assign charge to the grid and to interpolate the force from the mesh to the particles are not necessarily the same, and we will denote them by W^ρ, and W^F respectively. Consider a charge q at x. The force on this particle is

$$F(x) = q \sum_g W^F(x - x_g) E(x_g).$$

The field E is given by

$$E(x_g) = \sum_{g'} C_{gg'} \rho(x_{g'})$$

and so therefore

$$F(x) = \sum_g \sum_{g'} q W^F(x - x_g) C_{gg'} \rho(x_{g'}).$$

To calculate the self-force on the particle we consider only the contribution to ρ from the particle itself, ie.

$$\rho(x_{g'}) \to \delta\rho(x_{g'}) = q W^\rho(x - x_{g'}).$$

The resulting self-force is

$$F_{self}(x) = \sum_g \sum_{g'} q^2 C_{gg'} W^F(x - x_g) W^\rho(x - x_{g'}).$$

Now if we assume that $W^F \equiv W^\rho$, then the necessary condition to achieve $F_{self} = 0$ is that $\boldsymbol{C}$ be anti-symmetric. We saw above that centered differencing makes $\boldsymbol{C}$ anti-symmetric. Thus we have established that the combination of centered differencing and $W^F \equiv W^\rho$ eliminates self-forces.

4.3.2 Mixed NGP/CIC schemes

What if $W^F \not\equiv W^\rho$? Consider a single particle with charge q located at x in an infinite 1D system. Let us use x_m to denote the center of the cell containing the particle.

First consider the combination of NGP charge assignment and CIC force interpolation. We can easily show that the potential at mesh point g is

$$\phi_g = -\frac{q}{2} |x_g - x_m|.$$

Using CIC for the force evaluation we get

$$F = \begin{cases} q^2(x - x_m)/2\Delta x & x_m < x < x_m + \Delta x/2 \\ \\ -q^2(x_m - x)/2\Delta x & x_m - \Delta x/2 < x < x_m. \end{cases}$$

The self-force acts to drive the particle away from the cell center. This can produce instability.

Now consider CIC charge assignment and NGP force interpolation. Let us further assume that the particle lies in grid cell m as before and is located between x_m and $x_m + \Delta x/2$. Using CIC charge assignment gives

$$\rho_m = \Big(\frac{x_{m+1} - x}{\Delta x}\Big)\frac{q}{\Delta x}$$

$$\rho_{m+1} = \Big(\frac{x - x_m}{\Delta x}\Big)\frac{q}{\Delta x}.$$

Setting $\phi_m = 0$ and assuming equal but opposite potential gradients at $\pm\infty$, ie.

$$\frac{\phi_{g+1} - \phi_g}{\Delta x} = -\frac{\phi_{-g+1} - \phi_{-g}}{\Delta x}$$

as $g \to \infty$, we can easily show that

$$\phi_{m+1} = -\frac{q}{2}(2(x_m - x) + \Delta x)$$

$$\phi_m = -\frac{q}{2}\Delta x$$

which implies that

$$F = -q^2\Big(\frac{x - x_m}{2\Delta x}\Big).$$

The particle will perform simple harmonic oscillation about the nearest grid point with frequency

$$\omega_{self} = \sqrt{\frac{q^2}{2m\Delta x}}.$$

From our analysis of the leap-frog scheme we know this will be stable provided the timestep is chosen to satisfy $\omega_{self}\Delta t < 2$.

The general principle here is that we should always use a force interpolation scheme which is of no higher order that the scheme used for charge assignment.

4.3.3 Equal but opposite forces

Now consider two particles, q_1 at x_1 and q_2 at x_2. The force on particle 1 due to particle 2 is

$$F_{12} = q_2 q_1 \sum_g \sum_{g'} W(x_1 - x_g) C_{gg'} W(x_2 - x_{g'}).$$

Similarly, the force on particle 2 due to particle 1 is

$$F_{21} = q_1 q_2 \sum_g \sum_{g'} W(x_2 - x_g) C_{gg'} W(x_1 - x_{g'}).$$

But since $\boldsymbol{C}$ is anti-symmetric

$$\begin{aligned} F_{21} &= -q_1 q_2 \sum_g \sum_{g'} W(x_1 - x_{g'}) C_{g'g} W(x_2 - x_g) && (29) \\ &= -F_{12}. && (30) \end{aligned}$$

Note, this result again depend on $\boldsymbol{C}$ being anti-symmetric, and $W^F \equiv W^\rho \equiv W$.

We have highlighted three properties which we would like the charge assignment/force interpolation scheme to satisfy,

- at particle separations large compared with the mesh spacing, the fluctuations in the inter-particle force due to displacement relative to the mesh should become negligible

- as a particle moves across the mesh the force it experiences and the charge it assigns to the mesh should change smoothly

- momentum should be conserved.

Hockney and Eastwood [1] describe a hierarchy of schemes constructed on these principles (NGP, CIC, TSC, ...). Alternatively, schemes can be developed by using multi-pole expansions of the charge distribution of each finite sized particle about the nearest grid point to the particle. To zeroth order this approach returns NGP. Including the dipole gives a scheme similar to CIC.

4.4 Aliasing

The spurious fluctuations which appear as a result of the loss of displacement invariance, manifest themselves in k-space as non-physical mode couplings, known as "aliasing". Spatial structure on scales finer than Δx cannot be represented on the mesh. Some of the power in this fine scale is erroneously interpreted by the mesh as belonging to longer wavelength modes which the grid does support.

For a 1D infinite system the density of particle centers is given by

$$n(x) = \sum_{i=1}^{N_p} \delta(x - x_i)$$

and the continuous charge density is

$$\rho_c(x) = \frac{q}{\Delta x} \int_{-\infty}^{\infty} W(x' - x) n(x') dx'.$$

This has fourier transform

$$\hat{\rho_c}(k) = \int_{-\infty}^{\infty} dx \ \rho_c(x) e^{-ikx}$$

with

$$\rho_c(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} dk \ \hat{\rho_c}(k) e^{ikx}. \tag{31}$$

By introducing a mesh we reduced our representation of $\rho(x)$ from a continuous representation $\rho_c(x)$ to a sampled representation $\rho_s(x_g)$. The only wavelengths which can be represented on the mesh are $\lambda \geq 2\Delta x$, ie $k \leq \pi/\Delta x = k_{grid}/2$. The appropriate transform for this discrete representation is the discrete fourier transform

$$\hat{\rho_s}(k) = \Delta x \sum_{g=-\infty}^{\infty} \rho_s(x_g) e^{-ikx_g}$$

$$\rho_s(x_g) = \frac{1}{2\pi} \int_{-k_{grid}/2}^{k_{grid}/2} dk \ \hat{\rho_s}(k) e^{ikx_g}. \tag{32}$$

Equation (32) expresses $\rho_s(x_g)$ in terms of the transform of the sampled charge density, but we can also write it in terms of the transform of the continuous charge density (equation (31)),

$$\rho_s(x_g) = \rho_c(x_g) = \frac{1}{2\pi} \int_{-\infty}^{\infty} dk \ \hat{\rho_c}(k) e^{ikx_g}. \tag{33}$$

Breaking up this integral into intervals of length k_{grid}, and setting $k = k' + nk_{grid}$, equation (33) becomes

$$\rho_s(x_g) = \sum_{n=-\infty}^{\infty} \frac{1}{2\pi} \int_{-k_{grid}/2}^{k_{grid}/2} dk' \hat{\rho}_c(k' + nk_{grid}) e^{ik'x_g} e^{ink_{grid}x_g}. \tag{34}$$

Comparing equations (34) and (32) implies,

$$\hat{\rho_s}(k) = \sum_{n=-\infty}^{\infty} \hat{\rho}_c(k + nk_{grid}) e^{ink_{grid}x_g}.$$

But $ink_{grid}x_g = 2\pi ing$ and therefore

$$e^{ink_{grid}x_g} = 1$$

so that

$$\hat{\rho_s}(k) = \sum_{n=-\infty}^{\infty} \hat{\rho}_c(k + nk_{grid}). \tag{35}$$

When we represent ρ on a discrete mesh instead of a continuum, we limit the independent wavelengths which the solution can contain to the wavenumber range $-k_{grid}/2 \leq k \leq k_{grid}/2$, called the "principal zone" or "first Brillouin zone". The transform of the charge density in our discrete representation is given by the sum of copies of the transform of the continuous charge density, each offset by a different multiple of k_{grid}. This means of course that $\hat{\rho_s}(k)$ is periodic in k with period k_{grid}. The extra contributions (from $|n| > 0$) to $\hat{\rho_s}(k)$ inside the principal zone are called aliases.

Note that for the infinitely long mesh there is a continuum of allowed wavenumbers in the principal zone. In the case of a finite sized mesh there would be just a discrete set of wavenumbers given by $k = 0$ and $k = 2\pi/(n\Delta x)$ where $1 \leq n \leq N_g$.

Aliasing is generally worse for $k \sim k_{grid}/2$ than for $k \sim 0$, ie the shortest wavelengths are most affected. Also, the smoother ρ is, ie the less power there is at short wavelengths, the less likely it is that aliasing will be a problem. Recall that

$$\rho_c(x) = \frac{q}{\Delta x} \int n(x') W(x' - x) dx$$

which implies by the convolution theorem that

$$\hat{\rho}_c(k) = \frac{q}{\Delta x} \hat{n}(k) \hat{W}(k). \tag{36}$$

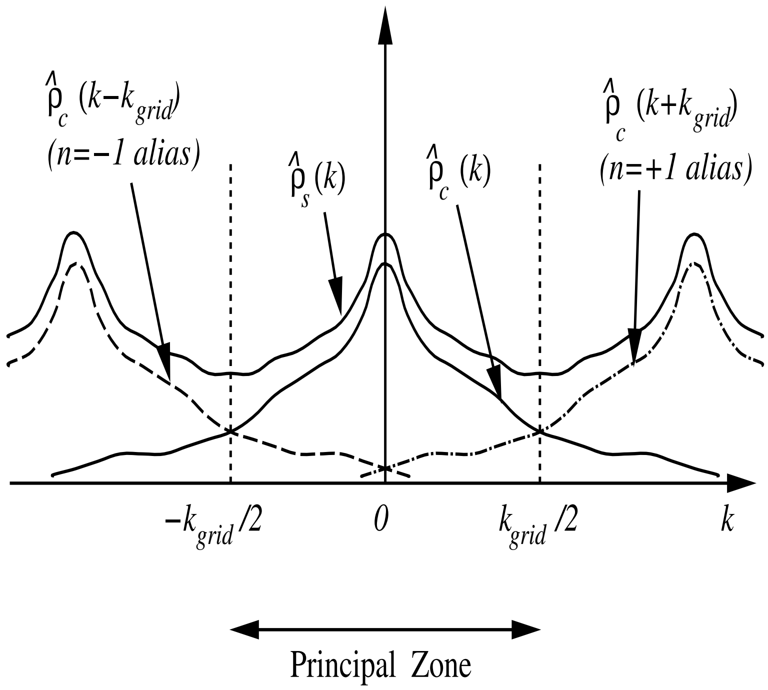

Figure 12: The fourier transform of the discrete sampled charge density, which is the sum of copies of the transform of the continuous charge density, each offset by a different integer multiple of k_{grid}.

Equation (35) implies that the narrower $\hat{\rho_c}(k)$ is in k-space, the less aliasing will occur.[7] But from equation (36) we can see that we can narrow $\hat{\rho_c}(k)$ by narrowing $\hat{W}(k)$, or in other words by making the charge assignment smoother.

For NGP we saw that $W_{NGP}(x) = \Pi(x)$, the hat function defined by equation (23) and shown in figure 9. The fourier transform of this is

$$\hat{W}_{NGP}(k) = \Delta x \ \mathrm{sinc}\Big(\frac{k\Delta x}{2}\Big) = \Delta x \ \frac{\sin(k\Delta x/2)}{k\Delta x/2}.$$

If we use CIC then $W_{CIC}(x) = \bigwedge(x)$, the triangle function defined in equa-

[7] If $\hat{\rho_c}(k)$ is band limited, in other words, a critical wavenumber k_{cr} exists such that $\hat{\rho_c}(k) = 0$ for $|k| \geq k_{cr}$, then no aliasing will occur provided $k_{grid} \geq k_{cr}$. Under these circumstances ρ can be represented exactly on the mesh. When $k_{grid} = k_{cr}$ the mesh samples ρ at the Nyquist frequency.

tion (26), and also illustrated in figure 9. This has a transform

$$\hat{W}_{CIC}(k) = \Delta x \ \mathrm{sinc}^2\Big(\frac{k\Delta x}{2}\Big).$$

In figure 9 we can see that $\hat{W}_{CIC}(k)$ is narrower than $\hat{W}_{NGP}(k)$ and so less susceptible to aliasing.

Aliasing will be produced by any mechanism which introduces wavelengths shorter than $2\Delta x$. Low order charge assignment schemes do this. Setting the particle size w smaller than the mesh size will also do this. We get strong aliasing if we set $w < \Delta x/10$. This limits the dynamic range of wavelengths which we can include in our model to $\lambda < 10N_g w$. This lower limit on w can be relaxed somewhat by using higher order assignment schemes.

Setting the Debye length much less than Δx will also introduce wavelengths shorter than $2\Delta x$. In a mono-energetic beam for example, $T = 0$ and so $\lambda_d = 0$. If we use ES1 to model this system we find that the beam begins to spread in v space. Aliasing is feeding energy from $\lambda < \Delta x$ modes into modes supported by the mesh, heating the beam and thereby raising both T and λ_d. This thermal instability persists until λ_d grows sufficiently that the condition $\lambda_d << \Delta x$ is no longer satisfied.

4.5 The Field Solver

Formally, we can write

$$\phi_s(x_g) = \Delta x \sum_{g'=-\infty}^{\infty} G_s(x_g - x_{g'})\rho_s(x_{g'})$$

or in k-space,

$$\hat{\phi}_s(k) = \hat{G}_s(k)\hat{\rho}_s(k). \tag{37}$$

The function G_s is a discrete representation of the Greens function for Poisson's equation. The transform of the exact(continuous) Greens function for $\nabla^2\phi = -\rho$ is $1/k^2$. If we use $\hat{G}_s(k) = 1/k^2$ then

$$G_s(x_{g''}) = \frac{1}{2\pi}\int_{-k_{grid}/2}^{k_{grid}/2} dk \frac{1}{k^2} e^{ikx_{g''}}$$

where $x_{g''} = x_g - x_{g'} = (g - g')\Delta x$. Therefore

$$G_s(x_{g''}) = \frac{i}{\pi}\int_0^{k_{grid}/2} dk \ \frac{\sin kx_{g''}}{k^2}.$$

For $g' = g$ this gives $G_s = 0$. When $g' \neq g$ we have $G_s(x_{g''}) = -G_s(-x_{g''}) \neq 0$, ie in x-space $1/k^2$ appears as a very non-local operator. If instead we use the finite difference approximation for $\nabla^2\phi$,

$$\phi_{g+1} - 2\phi_g + \phi_{g-1} = -\rho_g \Delta x^2$$

which transforms to

$$\left(e^{ik\Delta x} - 2 + e^{-ik\Delta x}\right)\hat{\phi}_s(k) = -\Delta x^2 \hat{\rho}_s(k)$$

where

$$\hat{\phi}_s(k) = \Delta x \sum_{g=-\infty}^{\infty} \phi_s(x_g) e^{-ikx_g},$$

giving

$$\begin{aligned}
\hat{\phi}_s(k) &= \frac{1}{k^2}\hat{\rho}_s(k)/\mathrm{sinc}^2\Big(\frac{k\Delta x}{2}\Big) && (38)\\
&= \hat{\rho}_s(k)/\mathrm{K}\Big(\frac{k\Delta x}{2}\Big). && (39)
\end{aligned}$$

Differencing exaggerates the amplitude of higher k modes.

4.6 Force Evaluation

Recall that

$$E = -\frac{\partial \phi}{\partial x}$$

and

$$\hat{E}(k) = -ik\hat{\phi}(k).$$

Again we can choose to use these exact operators, or finite difference approximations to them. If we replace $\partial\phi/\partial x$ with

$$\frac{\phi_{g+1} - \phi_{g-1}}{2\Delta x}$$

we replace the exact operator, $-ik$, in fourier space, with $-ik\sin(k\Delta x)/k\Delta x$,

$$\begin{aligned}
\hat{E}(k) &= -ik\ \mathrm{sinc}(k\Delta x)\ \hat{\phi}(k) && (40)\\
&= -i\kappa(k\Delta x)\ \hat{\phi}(k). && (41)
\end{aligned}$$

In this case differencing acts to dampen high k modes.

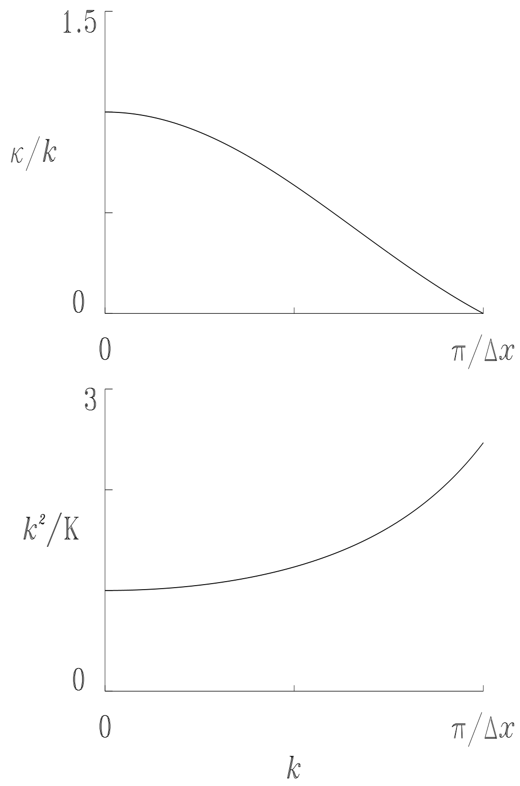

Figure 13: The function κ is introduced into the evaluation of the electric field when we use the centered second order finite difference approximation. It acts to dampen the higher k modes. Similarly k^2/K appears in the potential solver but this exaggerates the higher k modes.

4.7 Optimal Schemes

In the last few sections we have used fourier analysis to study why spatial discretization errors occur and how they may affect the overall solution. We can use this type of analysis to develop improved schemes in which the errors introduced at each stage can partially cancel each other.

All four stages in our code which contribute to the evaluation of the force on the particles, ie charge assignment to the mesh, solving Poisson's equation to obtain the potential, differencing the potential to derive the electric field and force values at the mesh points, and finally interpolating force values at the particle positions from the mesh values, can be expressed as convolutions in x-space and so as products in k-space. For example, rewriting

$$E_g = -\frac{\phi_{g+1} - \phi_{g-1}}{2\Delta x}$$

as

$$E_g = -\sum_{g'} (\delta_{g+1-g'} - \delta_{g-1-g'})\phi_{g'}/2\Delta x \tag{42}$$

$$= -\Delta x \sum_{g'} D(x_g - x_{g'})\phi_{g'} \tag{43}$$

where the operator D is defined as

$$D(x_g - x_{g'}) = -(\delta_{g+1-g'} - \delta_{g-1-g'})/2\Delta x^2.$$

The fourier transform of equation (43) is

$$\hat{E}(k) = -\hat{D}(k)\hat{\phi}(k).$$

Similarly interpolation of force to particle positions is given by

$$F(x_i) = \frac{q}{\Delta x} \sum_{g'} W(x_i - x_{g'})E(x_{g'})$$

which has transform

$$\hat{F}(k) = -\frac{q}{\Delta x^2}\hat{W}(k)\hat{E}(k). \tag{44}$$

Combining equations (43),(44),(36) and (37), gives

$$\hat{F}(k) = -\frac{q}{\Delta x^2}\hat{W}(k)\hat{D}(k)\hat{G}(k)\frac{q}{\Delta x}\hat{n}(k)\hat{W}(k)$$

and so

$$F(x) = -\frac{q^2}{\Delta x^3}\frac{1}{2\pi}\int dk\hat{W}(k)\hat{D}(k)\hat{G}(k)\hat{W}(k)\hat{n}(k)e^{ikx}.$$

This equation enables us to combine the separate numerical steps in a way which produces the most "accurate" expression for $F(x)$.

5 Energy Conservation and Collision Times

The schemes we have considered so far are momentum conserving, but do not conserve energy, although the misconservation can be kept to acceptably small levels. Energy conserving schemes do exist, but they fail to conserve momentum. The crucial difference in energy conserving schemes is that ∇W or ∇S, when needed, are evaluated analytically rather than numerically. We should point out that the term "energy conserving" is a little misleading-leading in this context, since these schemes only conserve energy in the limit as $\Delta t \to 0$. In other words, introducing time discretization breaks this conservation.

In momentum conserving schemes the misconservation of energy is due to aliasing. Likewise in energy conserving schemes the misconservation of momentum is also due to aliasing. If we eliminate aliasing by using a band limited cloud shape function $S(x)$, we can conserve both energy and momentum. However this is an expensive option because it generally requires a very high order weighting function $W(x)$ which is highly non-local, and for this reason is almost never used.

We have seen that aliasing is less of a problem for schemes which use higher order charge assignment. Therefore, for the momentum conserving schemes which we have considered, we would expect the CIC scheme to conserve energy better than the NGP scheme.

5.1 Heating Time

The principal symptom of energy misconservation is particle heating. Hockney [8] made a systematic study of heating and collision times for a 2D electrostatic plasma. Let $h_i(t)$ be the deviation of the kinetic energy of the i^{th} particle from its initial value. The average over all particles is

$$\langle h(t) \rangle = \frac{1}{N_p} \sum_{i=1}^{N_p} h_i(t).$$

We define a heating time τ_H such that

$$\langle h(\tau_H) \rangle = \frac{1}{2} kT.$$

How do we determine $\langle h(\tau_H) \rangle$? Assume that the errors in our model contribute to a stochastic error field $\delta \boldsymbol{E}$. For simplicity we shall assume $\delta \boldsymbol{E}$ is constant in magnitude but varies randomly in direction. For one timestep it introduces a momentum change

$$m\delta\boldsymbol{v} = q\delta\boldsymbol{E}\Delta t$$

for each particle. Each particle describes a random walk in $\boldsymbol{v}$-space from its initial velocity $\boldsymbol{v}_0$. Writing $\Delta\boldsymbol{v} = \boldsymbol{v} - \boldsymbol{v}_0$, after n timesteps

$$\langle \Delta\boldsymbol{v} \rangle = 0$$

$$\langle |\Delta\boldsymbol{v}|^2 \rangle = n\frac{q^2\Delta t^2}{m^2}|\delta\boldsymbol{E}|^2$$

which implies

$$\begin{aligned}
\langle h \rangle &= \frac{1}{2}m\langle |\boldsymbol{v}_0 + \Delta\boldsymbol{v}|^2 \rangle - \frac{1}{2}m\langle |\boldsymbol{v}_0^2| \rangle && (45)\\
&= \frac{1}{2}m\langle |\Delta\boldsymbol{v}_0^2| \rangle && (46)\\
&= n\frac{q^2\Delta t^2}{2m}|\delta\boldsymbol{E}|^2. && (47)
\end{aligned}$$

Stochastic heating increases linearly with the number of timesteps.

Hockney has shown that the heating timescale is a complicated function of both $\Delta x/\lambda_d$ and $\omega_{pe}\Delta t$. The heating rate increases as either $\Delta x/\lambda_d$ or $\omega_{pe}\Delta t$ increases. CIC has a slower heating rate than NGP, by a factor of roughly 20.

5.2 Collision Times

The effective collision frequency ν_c was determined in the 2D model by measuring the deflection $\varphi_i(t)$ of particles from their original direction.

$$\langle \varphi^2(t) \rangle^{1/2} = \Big(\frac{1}{N_p}\sum_{i=1}^{N_p}\varphi_i^2(t)\Big)^{1/2}.$$

The collision time τ_C is defined by

$$\langle \varphi^2(\tau_C) \rangle^{1/2} = \frac{\pi}{2}.$$

Hockney [8] found the relationship

$$\frac{\tau_C}{\tau_{pe}} \simeq n(\lambda_d^2 + w^2).$$

Recall that τ_{pe} is the shortest timescale associated with collective modes in the system. As the particle size $w \to 0$ the collisionality of the system is determined by the number of particles in the Debye circle $(n\lambda_d^2)$. The finite size of particles helps to increase the collision time. Note that since w is the same for both NGP and CIC, both schemes have the same collisionality.

6 Higher Dimensions

The basic steps are the same. However everything is more complicated to program and more costly to run. Anisotropies appear due to the use of square or cubic particle shapes, and due to directional dependencies in truncation errors of finite difference approximations. Waves propagate with differing ease in directions aligned with or between axes. These anisotropies can be reduced by using smoother cloud shape and assignment functions.

Visualization of results is considerably more difficult, particularly in 3D.

7 P^{3}M

We recall from section 1.2 that the mesh was introduced into particle codes as a means of avoiding the N^2 operation scaling count associated with particle-particle(PP) schemes. Because of this N^2 scaling, particle-particle codes only work well for systems with small N, or for systems with highly localized interaction forces.

Particle-mesh(PM) on the other hand works efficiently for large N. However we quickly realized that we could not afford to resolve the typical inter-particle spacings with the particle-mesh approach. As a result we only apply it to 'collisionless' systems where we already know from physical arguments that near neighbor interactions are not significant and can be ignored in favor of smoothly varying long range interactions.

What can we do if we have a system in which short range interactions are important (ie we need to resolve inter-particle distances), but the number of particles contributing to these interactions is too large for the particle-particle approach ?

One answer is to use a combination of PP and PM called particle-particle-particle-mesh(P^{3}M). This was invented by Hockney et al[2][3].

The basic idea is straightforward. Use PP to capture the near neighbor interactions out to some cutoff distance r_c, chosen to include a sufficiently small number of particles that PP is still tractable. Then use PM to calculate

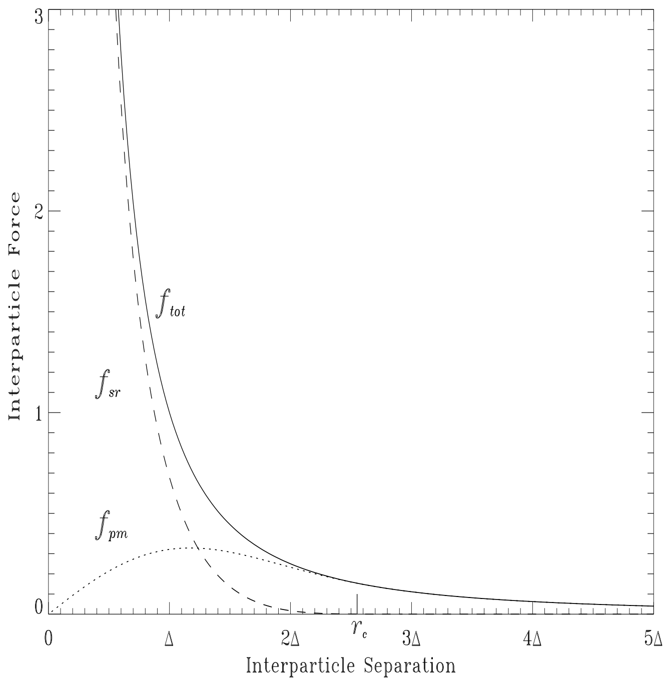

Figure 14: An example of the decomposition of the total $1/r^2$ inter-particle force in a 3D P^3M model.

the contributions to the force from more distant particles. This is illustrated schematically in figure 14 for particles in a 3D model interacting with a r^{-2} force law. The total force on a particle due to another at a distance r is given by

$$f_{tot}(r) = \frac{1}{r^2}$$

and can be decomposed into two distinct contributions, $f_{pm}(r)$ from the PM calculation, and a short range correction term $f_{sr}(r)$ introduced by the PP calculation, where

$$f_{sr}(r) = f_{tot}(r) - f_{pm}(r)$$

for $r < r_c$ and $f_{sr}(r) = 0$ for $r \geq r_c$. What we need is a reliable estimate for $f_{pm}(r)$.

7.1 Optimizing the PM Part

In PM codes we know that the force on particle i at $\boldsymbol{r}_i$ due to particle j at $\boldsymbol{r}_j$ is given by

$$f^*(\boldsymbol{r}_i, \boldsymbol{r}_j) = \frac{1}{V} \sum_{\boldsymbol{k}} \hat{D}(\boldsymbol{k}) \hat{G}(\boldsymbol{k}) \sum_{\boldsymbol{n}} \hat{W}(\boldsymbol{k_n}) e^{i\boldsymbol{k_n} \cdot \boldsymbol{r}_i} \sum_{\boldsymbol{m}} \hat{W}(\boldsymbol{k_m}) e^{-i\boldsymbol{k_m} \cdot \boldsymbol{r}_j}$$

The symbol $\hat{}$ denotes the fourier transform of a continuous operator. The vector $k = \boldsymbol{l} k_{grid}$ lies inside the principal zone of the discretely sampled periodic mesh. $\boldsymbol{l}$ is a vector of integer components. For example in 3D $\boldsymbol{l} = (l_x, l_y, l_z)$ where $-L_i/2 < l_i \leq L_i/2$ and the system volume is $V = L_x L_y L_z$. The vectors $\boldsymbol{n}$ and $\boldsymbol{m}$ are vectors of integer components, like $\boldsymbol{l}$, but without bounds. The summations over $\boldsymbol{n}$ and $\boldsymbol{m}$ are sums over aliases, with $\boldsymbol{k}_n = \boldsymbol{k} - 2\pi \boldsymbol{n}$, and $\boldsymbol{k}_m = \boldsymbol{k} - 2\pi \boldsymbol{m}$.

Can we use $f^*(\boldsymbol{r}_i, \boldsymbol{r}_j)$ to represent $f_{pm}(r)$?

The simple answer is no. The reason is that f^* is not a function of r, but of $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$, the specific positions of the particle pair that is being considered. Different particle pairs can have the same scalar separation r but different values of f^*. This is a consequence of the loss of translational invariance which is one manifestation of the aliasing errors. Also in 2 or 3D, directional anisotropies are introduced if square or cubic particle shapes are used. The result of this is to transform the curve representing $f_{pm}(r)$ in figure 14 into a set of points (one for each particle pair) which outline a similar curve, but which have a spread which indicates the significance of the aliasing error, as shown in figure 15. Since we will be investing considerable effort using PP to compute the short range contributions to the forces accurately, we must ensure that the aliasing error in the PM calculation is small enough not to undo this good work. In effect this means that we must minimize the spread in these points.

In plain PM codes we know that the shortest range interactions, where aliasing effects are most significant, are not important. We used finite sized particles and the low pass filtering effect of the mesh to minimize them. We were not overly concerned with the form of the interaction force at very short wavelengths provided aliasing errors were not overwhelming and the effective interaction force law dropped smoothly to zero for $r = 0$. However in $\mathrm{P}^3\mathrm{M}$ we require much finer control of the error associated with the PM part.

We achieve this finer control by defining a reference force $R(r)$ which is the force between two spherically symmetric charge clouds with a smooth

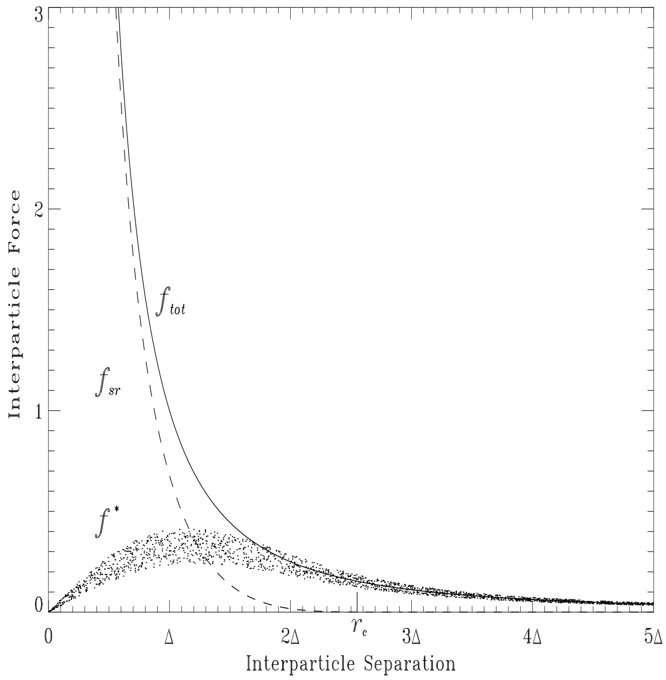

Figure 15: Same as figure 14 except that the idealized representation of $f_{pm}(r)$ is replaced by a set of points representing the forces $f^*(\boldsymbol{r}_i, \boldsymbol{r}_j)$ for particle pairs as they might be returned by the PM calculation.

higher order shape function. $R(r)$ is computed analytically, without the use of a mesh. We implement the PM section of our code,

- using higher order charge weighting and force interpolation functions
- using a higher order (ie smoother) differencing stencil for the gradient operator
- and by adjusting the Green's function used in the Poisson solver to minimize the difference between f^* and $R(r)$.

Then we can use

$$f_{sr}(r) = f_{tot}(r) - R(r)$$

as the short range correction provided by the PP component.

The particle cloud shape used most commonly to calculate the reference force $R(r)$ is

$$S_2(r) = \begin{cases} 48(\frac{a}{2} - r)/\pi a^4 & r < \frac{a}{2} \\ \\ 0 & r \geq \frac{a}{2} \end{cases}$$

where a is the particle diameter. The resulting inter-particle force is

$$R(r) = \begin{cases} (224\xi - 224\xi^3 + 70\xi^4 + 48\xi^5 - 21\xi^6)/35a^2 & \xi \leq 1 \\ \\ (12\xi^{-2} - 224 + 896\xi - 840\xi^2 + 224\xi^3 + \\ \qquad 70\xi^4 - 48\xi^5 + 7\xi^6)/35a^2 & 1 < \xi \leq 2 \\ \\ r^{-2} & \xi \geq 2 \end{cases}$$

where $\xi = 2r/a$. The continuous fourier transform of R is

$$\hat{R}(\boldsymbol{k}) = \frac{-i\boldsymbol{k}\hat{S}_2^2}{k^2}.$$

For charge assignment and force interpolation we use the triangular shaped cloud(TSC) weighting function $W(\boldsymbol{r}) = w(x)w(y)w(z)$ with

$$w(x) = \begin{cases} \frac{3}{4} - x^2 & |x| \leq \frac{1}{2} \\ \\ \frac{1}{2}(\frac{3}{2} - |x|)^2 & \frac{1}{2} \leq |x| \leq \frac{3}{2} \\ \\ 0 & |x| > \frac{3}{2} \end{cases}$$

The use of this product of triangular shaped functions, which is not spherically symmetric, provides a reasonable approximation to the spherically symmetric shape S_2, while avoiding the need to compute radius vectors, as would be required with a spherically symmetric cloud. The fourier transform of w is

$$\hat{w}(k) = \left(\sin\frac{k}{2} \Big/ \frac{k}{2}\right)^3.$$

We can use a 10 point differencing scheme to represent the gradient operator, D, in 3D. The x component is

$$\begin{aligned} (D_x\phi)_{i,j,k} &= \frac{1}{6}(\phi_{i-1,j,k} - \phi_{i+1,j,k}) \\ &+ \frac{1}{12}(\phi_{i-1,j+1,k} - \phi_{i+1,j+1,k} + \phi_{i-1,j-1,k} - \phi_{i+1,j-1,k} \\ &+ \phi_{i-1,j,k+1} - \phi_{i+1,j,k+1} + \phi_{i-1,j,k-1} - \phi_{i+1,j,k-1}). \end{aligned}$$

This is considerably more accurate than the 2 point differencing scheme we used in plain PM, provided ϕ is sufficiently smooth.

Finally, we select the optimal Green's function by minimizing the function

$$Q = \frac{1}{\Delta^3}\int_{\Delta^3} d\boldsymbol{r}_1 \int_V d(\boldsymbol{r}_2 - \boldsymbol{r}_1)|f^*(\boldsymbol{r}_1, \boldsymbol{r}_2) - R(|\boldsymbol{r}_2 - \boldsymbol{r}_1|)|^2$$

with respect to G. The resulting Green's function will produce the PM force $f^*(\boldsymbol{r}_1, \boldsymbol{r}_2)$ with the least squares difference from $R(r)$ when averaged over all the possible pair positions $(\boldsymbol{r}_1, \boldsymbol{r}_2)$. This is given by [1]

$$\hat{G}(\boldsymbol{k}) = \hat{D}(\boldsymbol{k}) \cdot \sum_{\boldsymbol{n}} \hat{R}(\boldsymbol{k_n})\hat{W}^2(\boldsymbol{k_n})|\ \hat{D}(\boldsymbol{k})|^{-2}\Big[\sum_{\boldsymbol{n}} \hat{W}^2(\boldsymbol{k_n})\Big]^{-2}.$$

We now have a PM component which returns the reference force $R(r)$ to an accuracy determined by the error estimate Q. This error is a function of the ratio of the particle size, a, used to compute R and the mesh spacing Δ. As we increase a/Δ we push the point at which the R curve peaks and turns down out to larger multiples of Δ. In effect we are adjusting the reference force to make it vary very smoothly out to many times Δ. This is a force law that we know we can achieve very accurately with PM with higher order weighting and differencing and the optimal Green's function. Of course we cannot increase a/Δ without penalty. The price we pay is that we have to increase the cut-off distance r_c associated with the short range correction in the PP code to compensate for the fact that R now turns down at larger r. This means there are more particles contributing to the PP forces. The relative values of a, r_c and Δ are set as a result of this tradeoff between accuracy and efficiency. For the algorithm components described here, typical values would be $0.7 \leq r_c/a \leq 1$, and $a/\Delta \sim 3$ – 4.

7.2 Implementing the PP Part

The main task here is to identify which particles are separated by less than r_c. To compute the force on a chosen particle we could cycle through the entire particle list testing to see which particles were within a distance r_c of the chosen particle. However this approach does not avoid the N^2 operation count scaling. We need to limit our search to a much smaller subset of the complete particle list.

To do this we can use a second mesh, coarser than the mesh used for the potential solver, which is usually called a 'Chaining Mesh'. A typical example is shown in figure 16. We identify which cell a particle is in and

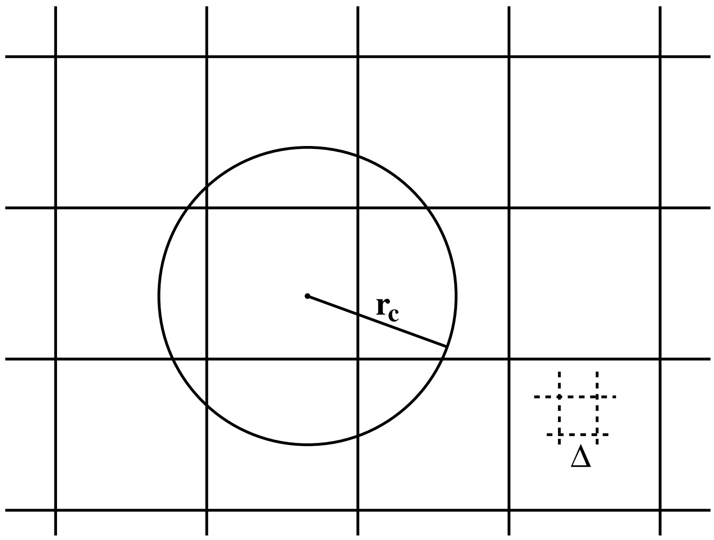

Figure 16: An example of a chaining mesh. The finely spaced dashed lines indicate the relative size of the mesh used PM field solver.

then link this particle to a list of any other particles which we have found in the same cell. The following FORTRAN code shows how these linked lists might be constructed.

```
integer   pp(n_coarse_grid)     ! particle pointer for each cell
integer   ll(npart)             ! particle linked list


do j=1, n_coarse_grid          ! initialize a particle pointer
    pp(j) = 0                   ! in each coarse mesh cell.
enddo


do i=1, npart                  ! cycle through the particles.

    icell = .......            ! determine which cell particle i is in.

    ll(i) = pp(icell)          ! link particle i to the last particle
                               ! found in cell icell.
```

```
        pp(icell) = i                ! update this cells pointer.

    enddo
```

Then for a given particle we need only test those particles in the same linked list, or the linked lists for the cells immediately surrounding the cell containing the chosen particle.

For the actual evaluation of the short range force between a particle pair it is often best to use a look-up table. We construct a table of values of $f_{sr}(r)$ at a discrete set of regularly spaced values of r in the range $0 < r \leq r_c$. This is done only once before the time integration begins. Then when we need to know f_{sr} for a particular particle separation we can compute it very efficiently by interpolation from the look-up table. For $1/r^2$ forces in 3D we would tabulate $\mathcal{F}(r) = f_{sr}(r)/r$. Then the velocity update for particle i becomes

$$\boldsymbol{v}_i^{new} = \boldsymbol{v}_i^{new} + \frac{\alpha}{m_i} \sum_{j=1}^{N_{pp}} \beta_j \mathcal{F}(|\boldsymbol{r}_i - \boldsymbol{r}_j|)(\boldsymbol{r}_j - \boldsymbol{r}_i) + \delta \boldsymbol{v}_i^{pm}$$

which avoids the need to evaluate any square roots. Here N_{pp} is the number of particles within distance r_c of particle i, and α and β_j are the appropriate constants.

We must honor the stability criterion associated with the time integration of the particle equations of motion. For the example of the $1/r^2$ inter-particle force law this means that as we attempt to resolve finer and finer particle separations we must use smaller and smaller timesteps. If we allowed point particles to approach infinitely close to one another we would see the inter-particle force go to infinity and the allowed timestep would be cut to zero. We can avoid having to cut the timestep below a certain minimum value and still not have to worry about particles at very short separations violating the stability constraint. The way we do this is by defining $f_{tot}(r)$ using finite-sized rather than point particles. That filters out wavelengths shorter than the particle size, which we will denote by a_s. Of course in this case a_s is very much smaller than the mesh size associated with the PM section and the particle size a used to define the reference force R. It is the resolution limit of the P^{3}M calculation, and in gravitational models is often referred to as the softening length.

Couchman[9] has developed an adaptive P^{3}M code which enables the technique to simultaneously encompass a much greater range of length scales.

8 PIC for Compressible Fluid Flow

PIC was originally invented by Harlow for this application. His motivation was to develop a tool to study highly distorted or sheared flows, or strongly shocked flows involving material interfaces and contact discontinuities in 2 or 3D. Here the particles represent fluid elements. In time the approach was dropped in favor of improved fluid codes because it was

- too noisy
- had high numerical viscosity (momentum diffusion)
- suffered from large heat conduction (energy diffusion).

It has been revived recently in codes like FLIP which are low dissipation PIC codes.

In its original incarnation, fluid PIC was a partially lagrangian method (mass was the only lagrangian variable). Modern fluid PIC is a fully lagrangian method (ie. mass, momentum and energy are all lagrangian variables).

For old style fluid PIC a typical timestep would be as follows:

1. load N_p particles, each with mass m_i, velocity $\boldsymbol{u}_i$ and internal energy e_i.
2. set up a mesh.
3. construct mesh point values of the fluid density ρ_g, velocity $\boldsymbol{U}_g$ and internal energy E_g by assigning the equivalent particle quantities to the mesh using eg.
$$\rho_g = \frac{1}{V_g} \sum_p m_i W(\boldsymbol{x}_i - \boldsymbol{x}_g).$$
4. solve the navier stokes equations on the mesh using finite differences.
5. construct a velocity and energy for each particle using eg.
$$\boldsymbol{u}_i = \sum_g \boldsymbol{U}_g W(\boldsymbol{x}_g - \boldsymbol{x}_i).$$
6. move the particles by solving
$$\frac{d\boldsymbol{x}_i}{dt} = \boldsymbol{u}_i.$$

7. begin cycle again at 1.

Mass density information goes just one way, from the particles to the grid. Mass is a lagrangian variable so mass diffusion is eliminated. However particle momentum and energy information is replaced every timestep by new values interpolated from the grid solution of the fluid equations. It is this transfer backwards and forwards which made the old style PIC so diffusive.

Modern fluid PIC, like FLIP, makes momentum and energy lagrangian variables also. This is achieved by solving fluid equations on a lagrangian grid, (ie. a grid which moves with the local fluid velocity), then updating rather than replacing the particle velocities and energies using the grid solution.

FLIP(Fluid Implicit Particle, Brackbill et al) is an implicit lagrangian code with adaptive re-zoning. It is less accurate than finite difference methods where they apply. It is also more expensive. However it still retains its advantage over finite difference methods where contact discontinuities and material interfaces are important.

EPIC(Ephemeral PIC, Eastwood) is an alternative low dissipation fluid PIC approach. It is finite element based, using an anti-diffusion step to remove momentum and energy diffusion.

9 Application to Incompressible Fluid Flow - the Vortex Method

By definition, for incompressible flow

$$\boldsymbol{\nabla}\rho = 0$$

and by implication from the mass continuity equation

$$\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0.$$

We can therefore write u as

$$\boldsymbol{u} = \boldsymbol{\nabla} \times \boldsymbol{\psi}$$

where $\boldsymbol{\psi}$ is the stream function. The momentum equation is

$$\rho \frac{d\boldsymbol{u}}{dt} = \boldsymbol{\nabla} p$$

and the vorticity is defined as

$$\boldsymbol{\omega} = \boldsymbol{\nabla} \times \boldsymbol{u}$$

We can show that vorticity is a conserved property of a fluid element in incompressible flow, ie.

$$\frac{d}{dt}\boldsymbol{\nabla} \times \boldsymbol{u} = -\frac{1}{\rho}\boldsymbol{\nabla} \times \boldsymbol{\nabla} p$$

and since $\boldsymbol{\nabla} \times \boldsymbol{\nabla} f = 0$ for any function f we have

$$\frac{d\boldsymbol{\omega}}{dt} = 0$$

In 2D, $u_z = 0$ and $\partial/\partial z \equiv 0$, so

$$\boldsymbol{\omega} = \boldsymbol{\nabla} \times \boldsymbol{u} = \boldsymbol{\nabla} \times \Big(\frac{\partial \psi_z}{\partial y}, -\frac{\partial \psi_z}{\partial x}, 0\Big)$$

$$\boldsymbol{\omega} = -\nabla^2(\psi_z \hat{\boldsymbol{k}}).$$

So for an element of fluid to which we assign a vorticity $\omega_z \hat{\boldsymbol{k}}$, the equations of motion are

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{u}$$

$$\boldsymbol{u} = \boldsymbol{\nabla} \times (\psi_z \hat{\boldsymbol{k}})$$

$$\nabla^2 \psi_z = -\omega_z$$

ie. the same equation structure as for a 2D electrostatic plasma. 3D vortex methods involve tracking vortex tubes. This is not a simple generalization of the 2D method. A comprehensive review of the vortex method is given by Leonard [7].

10 Parallel PIC

To understand how to optimize a PIC code for a particular architecture we need to understand how the processors and data memory are configured. We can think of our parallel computer as a system of data memory which feeds data to and receives data from a set of arithmetic processing units. This memory system can include

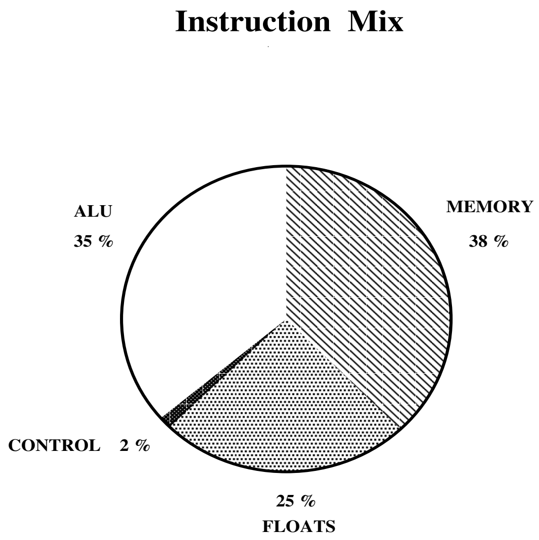

Figure 17: Cost of various instruction types for a 3D electrostatic code run on a Dec Alpha workstation.

- registers on processor
- cache on processor
- distributed RAM.

Each of these components have significantly different access times.

A common misconception is that raw processor speed is the key to faster performance. However on modern machines the performance of most real applications is dominated by the time to service memory accesses. In figure 17 we show the cost of the various types of instructions for a 3D electrostatic PIC code run on a Dec Alpha workstation. A profiling tool called PIXIE[12] was used to make these measurements. Execution of floating point operations accounts for only 25% of the run time, and higher level control constructs such as if statements accounts for only 2%. The bulk of the execution time is spent performing fetches from or stores to memory, and in integer arithmetic (ALU time), much of which is associated with address computations required for memory accesses.

The trick in optimizing the code is to tailor the algorithm and data layout to the machine in such a way as to keep the largest possible number of processors working effectively while reducing their access times to the data which they need.

In the most abstract sense, we have an algorithm and a data structure to map to the architecture. The four basic steps in a PIC algorithm are

1. assign particle charge(mass) to the mesh
2. solve for the force field on the mesh
3. interpolate force from the mesh to the particle positions
4. push the particles.

In combination these four steps involve computation and communication between two different data structures. The field data has the qualities of an ordered array in the sense that each element has specific neighbors. The particle data has the qualities of an unstructured vector, in which element i refers to particle i, and no element has any special relationship to its neighbors in the vector.

Steps 2 and 4 are parallelizable in rather obvious ways, since they involve only simple and completely predictable data dependencies, and do not couple the two data structures. Steps 1 and 3 however do couple the two data structures, with complicated and unpredictable data dependencies which evolve during the simulation. It is these steps which invariably dominate the execution times of parallel PIC codes.

There is one further observation to make before we discuss implementation specifics. On a serial machine our code can execute its computational workload in a time which is independent of any correlations in the spatial locations of the particles. This is not true on parallel machines. Particle clustering can create communication and/or computational hot-spots which impair performance. For example, an algorithm which works very efficiently for an homogeneous plasma application may be very inefficient for a highly clustered gravitational N-body problem. This can be an important factor in choosing an algorithm.

On a specific parallel machine, many factors will influence a codes performance. Machine architecture is unquestionably the most important. It would be appealing to present the best techniques for each of the broad architecture classes, such as SIMD (Single Instruction Multiple Data), MIMD (Multiple Instruction Multiple Data) with distributed memory and MIMD with shared memory. However such a clean presentation would be misleading. Even within these broad classes there are more minor architectural differences which can cause us to favor different algorithms. Differences in compilers, both in terms of functionality and maturity,[8] add to the problem.

So we will not attempt such a general description of preferred parallel algorithms. Instead we will examine how PIC codes have been parallelized on specific machines. These techniques should serve as suggestions rather than rules, when we approach other machines. The parallelization tasks we will consider are vectorizing the code for a powerful vector machine such as the Cray YMP series, implementing it on the MasPar's SIMD architecture, on a distributed memory MIMD machine, the Intel Touchstone Delta using message passing, and finally a MIMD shared memory implementation on the Convex Exemplar.

10.1 Vectorization

An efficient Cray YMP series code has two important characteristics. The most important is that it enables effective vectorization to the greatest extent possible. The second consideration is to minimize memory bank conflicts.

Three of the four basic steps of a PIC code (steps 2,3 and 4 above) are almost trivially vectorizable.

The particles are pushed by looping over the particles in sequence and pushing each in turn. Since each particle push is independent these loops vectorize. Highly efficient vectorized field solvers (eg fft's or multigrid) exist in various libraries(eg IMSL,NAG,ELLPACK) and we need not consider this any further. The force interpolation is a gather operation. Again we can loop over the list of particles, fetching the force values for the cells in the neighborhood of the particle. There are no data dependencies which will inhibit vectorization.

The only PIC code step which is not easily vectorizable is the charge deposition step. In a serial implementation we would loop over the list of particles in turn, determining where each particle is located in the mesh and then distributing contributions to the charge density of the mesh cells in the immediate vicinity of the particle. This loop will not vectorize automatically because it is possible that two particles might try to add charge to the same mesh cell at the same time. We can solve this problem in more than one way. The choice of solution depends on how often particles in the vector pipeline will try to add charge to the same mesh cell. If this is a rare occurrence then

[8] Many of the parallel machines and parallel compilers are in the earliest stages of their development.

we can test for when this happens and only inhibit vectorization when those particles are in the vector pipeline. However if it is a regular occurrence then the best solution is either to pre-sort the particles, or to use multiple copies of the charge density array.

10.1.1 Pre-sorting

We will illustrate a simple sorting algorithm devised by Horowitz [10].

For simplicity consider a 1D model using NGP weighting. Assume there are n_g particles in mesh cell number g. In this approach these particles are numbered from 1 to n_g. This is repeated for all the cells in the mesh. From these lists we form a number of particle groups. Group one contains all the particles numbered 1, group two all the particles numbered 2, and so on. Now loop over particle group 1. Each particle deposits its charge into the mesh cell to which it belongs. Since no two particles in group 1 belong in the same mesh cell we know that two particles can never be writing to the same array element at the same time, and we can force the loop over particles in the group to vectorize. We repeat this process in turn for all the particle groups. The only part of this algorithm which does not vectorize is the original grouping of the particles. It is easily extended to higher dimensions and to cope with higher order charge assignment schemes such as CIC.

Additional memory space is required to store the particle lists (and extra charge density arrays in the case of higher order charge assignment). Horowitz discusses how the algorithm can be tuned to trade-off between CPU performance and memory usage as resources might dictate.

10.1.2 Multiple Charge Density Arrays

Another simple vectorization technique is to use multiple charge density arrays. If we use 10 (say) charge density arrays, then we can vectorize a loop over 10 particles in an NGP scheme without any fear of dependencies, because each particle is updating elements of a different array[13]. After all the particles deposit their charge the 10 charge density arrays are combined to form the final master array. This approach buys vectorization at the expense of the extra memory allocated to store the multiple charge density arrays, and only achieves short vector lengths, in this example 10.

Héron and Adam[14] suggested a way to reduce memory needs and increase effective vector length. If we use CIC weighting, then in 3D each particle makes 8 separate contributions to distinct elements of the charge density array. We could compute the 80 contributions from these 10 particles in a vectorizable loop which stored the 80 data words into a temporary vector of length 80. Then these 80 contributions could be written into the 10 charge density arrays in another vectorizable loop. We would cycle through the particle list in strides of 10, and then finally, the 10 charge density arrays would be combined into a final summed charge density array by another vectorizable loop. In this case we would still require the same memory allocation but would have achieved a vector length of 80. If we used only 4 charge density arrays, we would more than halve the memory allocation, but would still have a vector length of 32, more than 3 times longer than the original scheme.

This approach works best for higher order weighting schemes, in higher spatial dimensions and for electromagnetic codes which deposit the current density vector in place of, or in concert with, the scalar charge density.

Anderson and Shumaker[15] have combined this scheme with particle sorting and data interleaving, designed to achieve more efficient memory access.

10.1.3 Dependency Testing

Again, let us consider the 1D model with NGP weighting. The charge deposition would be achieved by the Fortran 77 loop

```
do i=1,npart
rho(index(i)) = rho(index(i)) + q(i)
enddo
```

where `index(i)` is the mesh cell to which particle `i` with charge `q(i)` belongs, and `rho(j)` is the charge in mesh cell `j`. This scatter-with-add loop has potential vector dependencies, since `index` is not known until run time and changes during the simulation.

We break this loop into sections of length `nblock`.

```
do i1=1,npart-nblock+1,nblock
      do i=i1,i1+nblock
      rho(index(i)) = rho(index(i)) + q(i)
      enddo
enddo
```

Now consider one of these short inner blocks. We will test this inner loop to see if any vector dependencies actually occur within it. First we set up a temporary integer array, `itemp`, which has as many elements as there are cells in the mesh. We also set up two temporary integer arrays `ia` and `ib` of length `nblock`, with `ia(i)=i`. Now we use the appropriate `nblock` elements of `index` to scatter `ia` into `itemp`, forcing vectorization and accepting any overwrites which might occur.

```
cdir$ivdep
        do i=i1,i1+nblock
        itemp(index(i)) = ia(i)
        enddo
```

Then we try to reverse the scatter operation, by gathering elements of `itemp` into `ib`, under the influence of `index`.

```
        do i=i1,i1+nblock
        ib(i)=itemp(index(i))
        enddo
```

If no overwrites occured during the scatter operation, then the gather step exactly reverses the scatter and so `ia` and `ib` should be identical. In that case, for this particular block, we can safely force the inner loop to vectorize using a compiler directive. However if `ia` and `ib` differ anywhere, we have detected a vector dependency and the inner loop must execute in serial order.

Based on the success or failure of this test for each block, we can branch to a copy of the inner loop either with or without a preceeding compiler directive to ignore vector dependencies.

This scheme enables us to vectorize the charge deposition task over vector lengths `nblock`. Of course the test involves an additional overhead of a vectorized scatter and a gather step, so we would only consider it if we expected the test to find no vector dependencies much of the time. For a random spatial distribution of particles the chance that no two or more particles in the same block will try to add charge to the same cell is determined by the ratio of `nblock` to the number of cells in the mesh[9]. For performance reasons we would like to keep `ia` and `ib` in vector registers which means that `nblock` will be set to the vector pipeline length. On the Cray C90 that means `nblock` $= 128$. `nblock` will be much smaller than the number of mesh points for all but the smallest 1D meshes. As a consequence, the benefit of

[9]The actual value is $N_g!/N_g^b(N_g-b)!$ which is approximately $1-b^2/2N_g$ for $b << N_g$, where $b =$ `nblock`.

the improved vectorization should far outweigh the overhead associated with the test. [10]

No changes are required to apply this scheme in 2 or 3D, and minor modifications can accomodate higher order charge assignment schemes.

10.2 SIMD Implementation

The MasPar MP-2 has a SIMD architecture with up to 16384 (128×128) processors. The nominal peak performance of a 128×128 machine is 6.2Gflops. Each processor has 64Kb of dedicated data memory. The processors are arranged in a 2D array with dimensions which are integer increments of 32, ie 32, 64, 96 or 128. Straightline connections, known collectively as the X-net, exist between processors in the north, south, east, west, north-east, south-east, south-west and north-west directions. At the edges of the processor array the X-net wraps around so that the array has the same topology as the surface of a torus. Inter-processor communications can be achieved in one of two ways. The global router can be used for more complex patterns or for communication between widely spaced processors, while for regular patterns over short distances the X-net communications are much more efficient. The MasPar series broadens the definition of SIMD in at least one important way. It enables indirect addressing within a processor memory.

On a distributed memory machine, such as the MasPar, data layout across processor memory is an integral part of algorithm design. For PIC codes, once we have chosen the layout of the field arrays and particle data we have essentially set the computational and communication workload for each processor during each of the steps of the code. The challenge on the MasPar is to spread the computation and communication workload as evenly as possible, while minimizing the amount of global router communication required.[11]

The field arrays will be laid out so as to optimize the field solver routine. We do not need to consider the fine details of this layout, which will vary depending on the exact size of the physical mesh and the size of the processor array. All we really need to recognize is that any acceptable layout will establish a mapping between physical mesh points and the processor array so that it includes most if not all the processors, and that nearest neighbor

[10] Cray's cf77 compilation system automatically implements this solution.

[11] A plural floating point multiply takes 40 clocks on the MP-2, an X-net operation sending a real number a distance of 1 processor takes 41 clocks, and a random communication pattern using the global router, with all processors participating takes ~ 5000 clocks.

mesh cells will map to processors which are no further apart than nearest neighbors. For example if we have a 3D mesh of size $128 \times 128 \times 128$ and a processor array of size $N_{proc} = 128 \times 128$, we could map cell (i, j, k) into processor (i, j).

The major design question which faces us is how to distribute the particle data. There are some obvious choices which focus either on computational load balance or on efficient interprocessor communication [19].

10.2.1 Uniform Load Balance - with communication hotspots

The first option is to parcel the particles out evenly amongst the processors, paying no attention to their physical locations. This achieves the best computational load balance during the particle push and during the purely computational parts of the charge deposition and force interpolation tasks. It also makes memory management easier, since we know exactly how much memory we will need in every processor.

However it makes very heavy use of the global router for interprocessor communication. Any given particle can potentially seek to deposit charge on any of the processors in the processor array. For example if we use CIC for a 3D model, each particle has 8 charge contributions to distribute to a $2 \times 2 \times 2$ block of elements somewhere in the charge array. We can pack these 8 components into a message which is then sent by the global router to one of the processors storing the $2 \times 2 \times 2$ block, which then distributes them, as required among its neighboring processors using the X-net.

Similarily during the force interpolation the particle needs to interpolate between the 24 field components associated with the same $2 \times 2 \times 2$ block (ie 8 components for each of the x, y and z directions). At this point we have a choice to make.

We could pack the particles coordinates into a message, send them to a processor in the $2 \times 2 \times 2$ block, collect the 24 components there using X-net, compute the interpolated field values, pack them into a reply, and send the reply back to the originating processor using the router. This sends just 6 data words per particle over the router. However, on the MasPar, the global router has to cycle repeatedly to resolve message traffic contentions. Each processor belongs to a 4×4 group of processors which share a connection to the router. At most $1/16^{\text{th}}$ of the processors can communicate successfully each router cycle[12]. Since the force interpolations are inextricably bound

[12] For a random interprocessor communication pattern the router requires roughly 50 cycles

within the router cycling, no more than $1/16^{\text{th}}$ of the processors can perform interpolations simultaneously.

It is actually more efficient for each particle to fetch the 24 data words that it needs, allowing the router to cycle until all particles have fetched successfully, and then perform the force interpolation simultaneously in every processor. In this case the interpolation executes at least 16 times faster, more than compnsating for the cost of the longer messages.

This uniform load balanced scheme is slow because of its extensive use of the global router, and it scales poorly in situations where clustering occurs. Communication hotspots occur when a large number of messages are being sent to the same processor at the same time. The processors can only process one router message at a time. If n^p particles are actually located in cells which map into processor p, then processor p will need to receive n^p messages during that timesteps charge deposition. This algorithm therefore will scale as n^p_{max}, the maximum value of n^p across the processor array.

10.2.2 Uniform Load Balance - without communication hotspots

We can improve the scaling as a function of particle clustering of the charge deposition by using a combination of a sort and a segmented vector scan-add [5].

The basic approach is to sort the particle list based on the mesh cell in which the particle is located. The sorted list is then composed of segments. Each segment contains those, and only those particles located within the mesh cell associated with that segment. We can then use efficient algorithms to sum within these segments. If a segment is stored entirely within the memory of one processor we use a trivial serial algorithm to perform the sums. If the segment is distributed across multiple (ie > 1) contiguous processors then we can use a segmented vector scan-add to parallelize the summation.

This approach is easiest to explain for NGP charge assignment in the simple case where we have N_p particles and an equal number of processors.

Before we start we associate a unique id number $ID_{ij} = i + (j-1)N_x + (k-1)N_xN_y$ with each mesh cell (i,j,k), where N_x and N_y are the x and y dimensions of the mesh. As before we distribute the particles uniformly amongst the processors, with, in this example, one particle per processor. We think of the particle data and the mesh as long vectors. We set up another integer vector containing the particle cell ids. This vector is sorted in order of increasing cell id and the permutation required to sort it is recorded.

Now we apply the same permutation to the remaining particle data vectors. This permutation rearranges the one to one association between particles and processors. Because this is a permute operation it has no communication hotspots. It is performed using the global router. When this step is complete the particle vector is composed of a sequence of blocks of varying lengths. Each block is a contiguous list of particles whose spatial locations map to the same mesh cell, and each block is stored in a contiguous block of processors with one particle per processor. We can now use a segmented vector scan-add operation to sum the charge in each block. This can be written using X-net calls, and scales as $\ln n^p_{max}$. A simple vector scan-add example is described in appendix A. One possible and efficient choice of sorting algorithm for this case with one particle per processor is the bitonic sort[16] which scales as $\ln_2 N_p$.

The force interpolation can also be handled using a scan function to achieve a scaling with $\ln n^p_{max}$. The first particle in each block fetches the components of the field from the mesh cell corresponding to the particle's cell id. Since no processor receives more than one request there are no communication hotspots during this fetch. Then we use a segmented scan-copy to copy these values to every other particle in that particle block.

This scheme is easily extended to accomodate higher order charge deposition algorithms and cases where the number of particles and processors differ. Typically $N_p >> N_{proc}$, and then it is better to store segments by filling each processor memory before moving to the next processor. In this case the fastest available sorting algorithms are the Flashsort algorithms [17].[13]

10.2.3 A Particle Migration Strategy

The schemes we have outlined so far all make heavy use of the global router. We can avoid the router completely if we distribute the particles amongst the processors according to the same mapping used for the field arrays.

If a particle lies in cell (i, j, k) we store it on the processor to which we mapped cell (i, j, k). During the charge deposition, no particle will need to send charge any further than to a neighboring processor, and during the force interpolation the mesh field values which the particle needs are either on processor or stored by a neighbor. This enables us to use X-net communications exclusively. To maintain this locality we are required to

[13] Coded versions of these algorithms are currently available on the world wide web at http://www.cs/unc.edu/ prins/algs.html courtesy of Jan Prins of the University of North Carolina.

migrate particle information from processor to processor as the particles move between mesh cells. Since our timestep constraint limits the distance any particle can travel during that timestep to less than one mesh cell width, the migration can be achieved efficiently using the X-net.

There are of course drawbacks associated with this scheme. The additional code needed to perform the particle data migration makes the algorithm more difficult to program and debug. It also suffers from load imbalance as particle clustering develops. In this case both the communication and computation costs scale as n^p_{max}. Processor memory management is tricky. We have to allocate enough memory that the most heavily populated processor does not run out of memory. However this means that a lot of memory space in other processors will be allocated and never used.

This scheme has proven to be significantly faster than the others we have described when applied to relatively uniform spatial particle distributions. This is a testament to the relative efficiency of X-net communications when compared to global router communications.

One possible solution to its memory management weakness is to supplement this scheme with a backup routine similar to that used in the uniformly load balanced technique above. Two distinct particle populations are identified, those which have been migrated successfully(population I) and those which have not(population II). We use the migration strategy wherever possible. Any population I particle which tried to migrate to a processor whose memory was already full is left where it is and relabelled as population II. After we deposit the population I charge to the mesh, we use the global router to deposit charge from any population II particles. Similarily, when we have completed force interpolation for the population I particles we use the router approach to find the forces for any population II particles. At regular intervals (ie every 10 timesteps perhaps), we test to see if the population II particles might now be placed into the correct processors and so transferred back to population I. This hybrid scheme enables us to minimize memory wastage.

10.2.4 Which Algorithm Wins?

As we have mentioned, some of these algorithms will suffer from computational and/or communication hot spots when spatial clustering of particles occurs.

To illustrate this we have developed the following idealized test case[18], run on a 16384 processor MP-2. N_p particles have been placed into n_{cl}

clusters, whose centers are located at randomly chosen coordinates within the 128^3 mesh. Each cluster contains N_p/n_{cl} particles. The clusters have a radius R given by

$$\frac{R}{\Delta} = 128 \times n_{cl}^{-1/3} f$$

and the particle locations within a cluster are chosen randomly, subject to an r^{-2} density distribution, where r is the particles distance from the center of its cluster. The factor f is used to adjust the radius of the clusters. This test case mimics the clustering which might be encountered modeling a small number of interacting galaxies. By increasing f sufficiently we can make the particle distribution completely random.

The load balanced(both MPFortran and MPL versions), hybrid migration(MPL) and sort/scan-add(MPL) algorithms were tested. The sort/scan-add algorithm used a bitonic sort. The clusters were assigned a randomly chosen translational velocity to ensure that in the hybrid case the particle migration routines were exercised.

We chose $N_p = 10^6$ and $n_{cl} = 10$, and adjusted the degree of clustering by varying f. Results are presented in figure 18. Figure 18 shows average execution times per timestep for the combination of charge deposition and force evaluation tasks for all 3 algorithms. These times are plotted against the time averaged value of n^p_{max}/n^p_{avg}, where n^p is the number of particles contributing charge to processor p, n^p_{max} is the maximum value across the processor array, and n_{avg} is the average value.

This plot clearly shows the falloff in efficiency of the load balanced and particle migration strategies as spatial clustering of the particles develops, and the superior scaling of the sort/scan-add algorithm.

10.3 MIMD Implementation

MIMD systems present us with a more coarse grained parallelism. A MIMD machine will have somewhere in the range of 10 to 2000 microprocessors, each capable of running their own instruction stream. In principle, this introduces the possibility of using control decomposition(ie farming out separate tasks to different processors) as well as data decomposition. In practice the mandatory time ordering of the separate tasks in a PIC code leaves too little flexibility to make much use of control decomposition. However the separate instruction streams do enable SPMD (Single Program Multiple Data) style programming, and as we shall see this can prove useful in optimizing load balance.

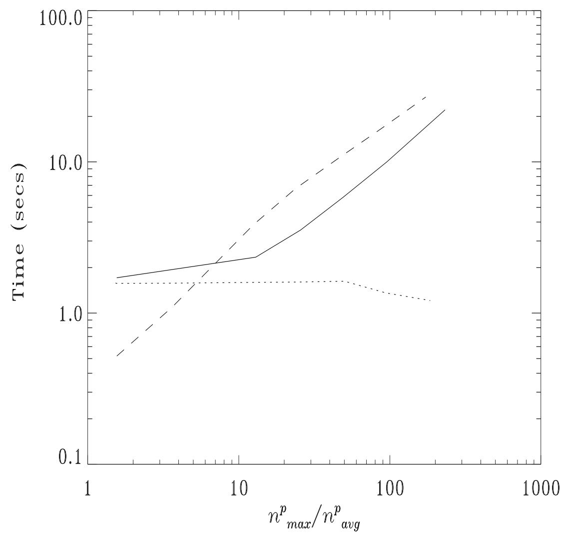

Figure 18: Average execution time per timestep for the combination of charge deposition and force evaluation tasks for the MPFortran load balanced(solid), sort/scan-add(dotted) and MPL hybrid(dashed) algorithms as a function of the degree of clustering.

10.3.1 Single Processor Optimizations

Almost all of the machines in this class use commodity RISC (Reduced Instruction Set Computer) processors which support some degree of instruction pipelining, and relatively slow Dynamic Random Access Memory(DRAM).

In an attempt to better match the processor and memory speeds, a small intermediate buffer (known as Cache) of fast Static Random Access Memory(SRAM) is provided between the processors and memory. When a processor requests a data word from memory a block of data (usually 4 or 8 data words) is moved from main memory into the processor's cache. The requested data word is than passed onto one of the processor's registers.

The advantage that cache provides is that if the next data word requested by the processor is in that same data block, the data is now available from cache on the fast SRAM access time rather than the slow DRAM time. [14] Cache based systems reward data locality. Data should be stored in memory based on the order in which the processors will need it, if possible.

Decyk et al [20] have performed a nice study of the influence of various single processor optimizations on the performance of a 2D code. They show that in general, for RISC/Cache machines, cache use is improved by avoiding large memory strides, and by occassionally sorting the particles according to their locations with respect to the mesh (see section 10.3.3). Optimizations for effective use of the arithmetic pipeline include removing IF statements, precalculation of any integer to float conversions, using one dimensional referencing for arrays rather than two or three dimensional, avoiding unnecessary divides and square roots, and unrolling loops in the deposition routine to help overlap fast arithmetic operations with slow loads and stores.

10.3.2 Message Passing Implementations

As with SIMD, our design goals are to keep as many of the processors as we can busy doing productive work. In choosing a data decomposition strategy we must bear in mind that we have many fewer processors than in the SIMD machine, that these processors are considerably more powerful computationally than their SIMD counterparts, and come with larger local memory banks. As a result, the balance between computation and interprocessor communication implied by our data decomposition must be struck somewhat differently than in our SIMD approach. It may also be necessary to dynamically reconfigure the data decomposition as the solution evolves, since the load imbalance penalty for a poorly fitting decomposition becomes more severe when we are working with fewer processors. The overhead for this dynamic load balancing must be factored into our analysis.

We will discuss two specific MIMD message passing PIC implementations. The first uses separate domain decompositions for particle related operations and for the field solver, at the cost of the extra communication required to share data between the two. The second approach uses only one decomposition. Both can perform dynamic load balancing.

(a) A Dual Decomposition

[14] For example, on the Convex Exemplar SPP1000 a data request serviced from cache takes 10ns while a request serviced from local memory takes 500ns.

The first example we have chosen to study is the 'General Concurrent PIC code' developed by Liewer et al [21] [22]. They have run this code on a number of machines including the Intel Touchstone Delta.

The Delta features 576 i860 microprocessors, each placed at a node of a regular 2D communication grid. Each processor has 16Mbytes of RAM, and in principle can achieve a peak of 80Mflops (single precision), although sustained performance is typically $\leq 10\%$ of peak.

Interprocessor communication is achieved by exchanging packaged messages. This message passing can be either synchronized amongst the processors or asynchronous. It is not significantly more expensive for a processor to communicate with a distant processor than with a near neighbor. There is a significant setup overhead associated with each message sent, so it is better to send a few long messages than a lot of short ones. The cost of a message increases with the length of the message, so it is advantageous to eliminate any unnecessary communication.

For simplicity, we will assume a 2D model covering a rectangular physical domain. The algorithm uses two separate data decompositions, the first to ensure that the particle push, charge deposition and force interpolation tasks are effectively load balanced, and the second to ensure that the field solve is load balanced.

First the physical domain is divided into sub-domains, with one sub-domain assigned to each processor, and with roughly equal numbers of particles in each sub-domain. For non-uniform particle distributions these sub-domains will not have equal areas and will contain different numbers of mesh points. Each processor is responsible for storing the data describing the particles in its sub-domain and for integrating their equations of motion. When a particle moves from one sub-domain to another the information describing the particle must be migrated to the appropriate processor. The processors also store the values of the electric field and charge density at the mesh points in their sub-domain, including any guard cells immediately outside the sub-domain boundaries which may be required. As the simulation evolves and particles move between sub-domains the sub-domain boundaries are adjusted at regular intervals to maintain roughly equal numbers of particles in each sub-domain. This guarantees that the particle push will be very evenly load balanced.

Because we are almost certain to have unequal numbers of mesh points in each of these 'primary' sub-domains, the field solver will not be load balanced. Therefore a secondary domain decomposition is established to suit the requirements of the field solver. Exactly what this decomposition is will depend on the technique used to solve Poisson's equation, but it is most likely to divide the mesh cells equally amongst the processors, in contiguous blocks. These 'secondary' sub-domains do not change during the simulation.

At the start of a timestep the particles deposit their charge to the mesh cells in their primary sub-domain. Because this information is needed by the field solver, and because the primary and secondary sub-domains do not necessarily coincide, this requires some inter-processor communication. The data which processor A must send to processor B is packaged together into a send buffer at A and then sent to B. When it arrives it is unpacked and stored appropriately. The field solver executes and the electric field is determined for every mesh point in the secondary sub-domains. The communication pattern is then reversed and the field values for the mesh points in the primary sub-domains are updated. At this point the forces on the particles can be evaluated by interpolation from the mesh using local data only. Then the particles are pushed. The final step is to transfer particle information between processors for any particles which have moved to a different primary sub-domain. Again this is done by packaging the information into longer messages in order to amortize the message start-up costs.

The selection of sub-domains is chosen so as to minimize inter-processor communication. There is no unique preferred solution to this problem. It is model dependent. Generally speaking, increasing the ratio of sub-domain area to boundary length (or volume to surface area in 3D) minimizes the percentage of particles which migrate between sub-domains. However if this reduces the overlap between primary and secondary sub-domains there will be more data to be communicated before and after the field solver is called. In some simulations particles tend to move in a preferred direction and then it pays to use long thin sub-domains aligned along this direction.

The details of the dynamic adjustment of the primary sub-domain boundaries obviously depend on how we choose to configure the sub-domains. Because there is an overhead associated with this adjustment, it should only be done when the load imbalance has exceeded some threshold value for which the resulting gain in performance outweighs the overhead. Since we wish to equalize the number of particles in each domain we need to know how many particles are in each sub-domain. This information can be accumulated with almost no extra effort during the charge deposition task, by also calculating the particle number densities at the mesh points. Then each processor sums the number densities over all the mesh cells in its sub-domain.

(b) A Single Decomposition

In this case a single domain decomposition is used for both particles and fields [23]. Since we do not keep a second copy of the field information this requires less memory than the dual decomposition strategy. Inter-processor communication is required to migrate particle information when particles change sub-domains, to exchange guard cell field and charge density values, and in the exchange of any field information required within the field solver.

The load on processor i is assumed to be of the form

$$L_i = AN_i + BM_i$$

where N_i is the number of particles and M_i is the number of mesh points in sub-domain i. A and B are constants which reflect the relative computational and communication costs of particle related and mesh related operations respectively. The objective is to adjust the sub-domain boundaries, and therefore N_i and M_i so that L_i is the same for all sub-domains. It is possible therefore that one processor might spend more of its time on particles and less on mesh points than another processor, and yet both would complete the timestep at the same time. This benefit cannot be realized for electrostatic codes because the global character of Poisson's equation requires that the processors be synchronized both at the start and finish of the field solver calculation. However electro-magnetic codes can be built that solve just local finite difference approximations to Maxwell's equations [15]. These require less stringent processor synchronization. For example, at the beginning of a timestep processors could accumulate current density (current density not charge density is required), update interior field values and push interior particles without requiring any information from neighboring sub-domains. Since this accounts for most of the computational effort a combined field/particle load balance is effectively possible. The remaining steps, exchanging guard cell information, updating sub-domain boundary field values and particles, and migrating particles would require some synchronization between processors.

10.3.3 Shared Memory Implementation

The Convex Exemplar provides a particularly interesting shared memory environment. It has an hierarchical NUMA(Non-Uniform Memory Access) structure, supplying data to as many as 128 HP PA-RISC 7100 processors. The system also provides two levels of cache support (cache on processor,

[15] Electro-magnetic models can be set up so that the solutions to $c\nabla \times \boldsymbol{E} = -\partial \boldsymbol{B}/\partial t$ and $c\nabla \times \boldsymbol{B} = \partial \boldsymbol{B}/\partial t + \boldsymbol{J}$ satisfy both Poisson's equation and $\nabla \cdot \boldsymbol{B} = 0$, provided both are satisfied by the initial fields [24].

and cache for sharing data between hypernodes) with full hardware support for global cache coherence.

Each processor has a 1 Mbyte direct mapped data cache and a 1 Mbyte instruction cache. A group of 8 processors along with 4 main memory banks forms a hypernode. A crossbar switch connects the processors within a hypernode. Multiple hypernodes (up to a limit of 16) can be connected together, using 4 parallel ring networks. Cache coherence is implemented in hardware within each hypernode and across hypernodes.

Memory is configured to provide a hierarchy of latencies. Data designated as 'thread-private' can only be accessed by the processor executing that thread, thereby minimizing the sharing management cost and giving the fastest response time for a cache miss. 'Node-private' data is the next fastest and is available only to processors on the same hypernode, provided they are members of the same sub-complex. 'Near-shared' memory can be accessed by any processor in the sub-complex but is stored entirely on the hypernode on which it was allocated. 'Far-shared' memory is accessible from any processor within the sub-complex, is distributed across all the hypernodes in the sub complex but has the largest latency. 'Block-shared' is a variation of far-shared but distributed in contiguous blocks with one block per processor. Near-shared latencies are typically of order 500 ns, while far-shared latencies are about 2000 ns. By comparison a local cache hit costs just 10 ns.

In designing parallel code for this machine what are the key properties of the algorithm and the machine?

Firstly, the particle data structure is very much larger than the mesh data structure. This means that good load balancing requires that each processor work with roughly the same number of particles. As with all other MIMD machines, good single processor performance is an essential foundation for good parallel performance. In this case good single processor performance requires good cache reuse. Good cache reuse is easy to achieve in the particle push or the field solver where we can cycle through the appropriate data structure in regular order. It is much harder to achieve during the charge deposition and force interpolation tasks where the particle and mesh data structures interact with a large degree of indirect data referencing. The hierarchical memory structure of the Exemplar potentially adds yet another penalty for irregular data referencing, due to the variable access times.

In the best performing code[25] the particles are evenly distributed amongst the processors without regard to their physical positions. The particle data is stored in node private memory blocks with one block per processor. This avoids any use of the inter-hypernode connection when servicing requests for particle data.

In the routine DEPOSIT each processor is allocated a thread-private copy of the charge density array and its particles deposit their charge into this copy. These copies are then summed to produce the final complete charge density array. This eliminates any possibility that two processors might simultaneously attempt to update the same element of the charge density array, and the hardware latencies involved in enforcing it.

In the routine FORCE, node private copies of the electric field solution returned by FIELD_SOLVER are made. These are then used to compute the forces on the particles, and the particle momenta are advanced in time. Finally in MOVER the particle positions are advanced in time which is a completely parallel task.

Figure 19 shows how this strategy scales[25] with processor number for a test problem which grows in size in proportion to the number of processors used. In this case an average of 18 particles per mesh cell was used, beginning with a mesh size of $N_g = 30 \times 24 \times 30$ on 1 processors, $N_g = 30 \times 48 \times 30$ on 2 processors, $N_g = 30 \times 48 \times 60$ on 4 processors and so on. Parallel efficiency on N_{proc} processors is defined here as the time to solution on N_{proc} processors divided by the time to solution of the application on 1 processor.

The most striking result in Figure 19 is the rapid degradation in the performance of FORCE. There is also a noticeable degradation with DEPOSIT above 8 processors. Only MOVER shows impressive scaling.

How do we explain the poor performance in this scaling test?

Consider FORCE first. In the routine FORCE, node private copies of the electric field solution returned by FIELD_SOLVER are made. These are then used to compute the forces on the particles, and the particle momenta are advanced in time. This makes good reuse of cache when referencing the particle data, but not when referencing the field arrays. For each particle, with cloud-in-cell interpolation, 24 data words are needed from the field components in the particles immediate vicinity. This means that cache misses resulting from irregular referencing of the field arrays can be quite costly. This is the principal reason for the poor scaling of FORCE in Figure 19. The operation of setting up node private copies of the field arrays has a maximum potential speedup of 8. However it is relatively inexpensive and it guarantees that all cache misses in FORCE can be resolved on the local hypernode.

In the routine DEPOSIT each processor is allocated a thread-private copy of the charge density array and its particles deposit their charge into

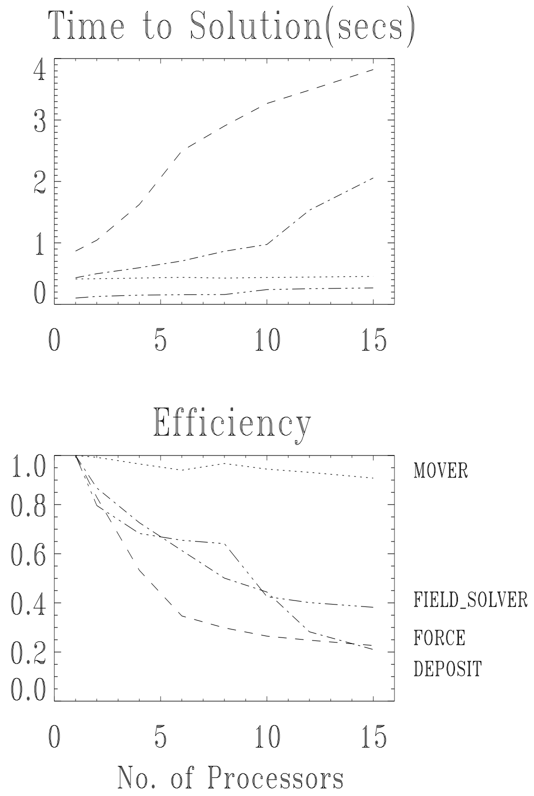

Figure 19: Time to solution per timestep and parallel efficiency scaling curves for a test case in which the workload per processor is kept constant.

this copy. These copies are then summed to produce the final complete charge density array. This is the most straightforward way to parallelize the charge deposition task while avoiding race conditions. It is perfectly load balanced and makes very good reuse of cache while dealing with the particle data. However the charge density array elements are referenced in a random order which can produce significant cache misses. Although the summation of the private copies of the charge density array into one master copy is implemented with completely parallel code, its timing is 'effectively' serial. Since N_{proc} copies of the charge density array (size N_g) are combined - a total of $N_{proc} \times N_g$ data words - by N_{proc} processors, the execution time for this step scales as $(N_{proc} \times N_g)/N_{proc} = N_g$.

A disadvantage with this approach is that extra memory must be allocated for the thread private arrays. However this extra memory overhead is small in typical PIC implementations, where the particle data structure's size dwarfs that of the grid.

Since the cost of cache misses is such a large factor in the poor scaling behavior, the obvious question to ask is 'can a sort of the particle data according to the particles mesh location enable more ordered referencing of the mesh data in FORCE and DEPOSIT?'

Figure 20 shows that sorting can make a significant difference. In this case the same simulations as in Figure were done but every 100 timesteps the particle data was sorted. The parallel sort which was used was based on the idea of a sample-sort [26]. A small sample of the particle data is taken and sorted using a serial sort. This is then used to set up N_{proc} bins into which the full particle list is coarsely sorted, with roughly equal numbers of particles in each bin. This is a partial sort. At this point one bin is assigned to each processor which can use a fast serial sort to complete a full sort of the particle data. The sorting routines actually produce an index list which defines the permutation required to reorder the particle data.

The addition of an occasional sort produces almost fourfold improvement in FORCE. DEPOSIT shows a less dramatic factor of 2 improvement. For 15 processors one application of SORT costs 12 seconds, or 0.12 seconds per timestep. The time saved in FORCE and DEPOSIT is 4 seconds per timestep, which implies that SORT could be implemented up to 30 times more frequently (ie every 3 timesteps) before it becomes a losing strategy.

How often should we sort?

For simulations in which the particles change mesh cell more often, the sorting frequency would have to be increased.

The routines whose timings are affected by the frequency with which SORT is called are FORCE, DEPOSIT and SORT itself. In Figure 21 the aggregate times per timestep for these routines as a function of the SORT calling frequency are plotted, for a test problem with a high migration rate in which about 10% of all the particles change mesh cell each timestep. It is clear that the optimal frequency is approximately once every 40 timesteps for this case. At this frequency the cost for these three routines for one timestep is 2.05 seconds. The equivalent time with 0.1% particle migration and sorting every 100 timesteps was 1.92 seconds, of which only 0.12 seconds was taken in doing the SORT. Clearly the algorithm is relatively insensitive to the rate at which particles migrate across the mesh. Also the shape of the curve in Figure 21 indicates that erring on the low frequency side of the optimal frequency is less costly than erring on the high frequency side. It would be easy to build an automatic search for this optimal frequency into

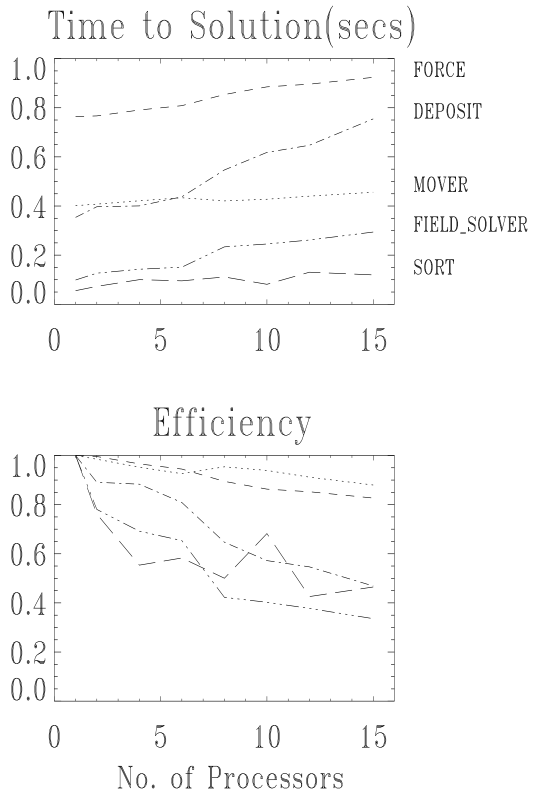

Figure 20: Same as figure 19 but with the particle data sorted according to the particle mesh locations every 100 timesteps. Note the scale change in the 'Time to Solution' axis.

the code.

For most 3D PIC simulations, the winning strategy is to sort the particle data no more than once every 20 timesteps. Particle migration across the mesh produces a slow, not sudden, degradation in performance because the particles have a finite size and overlap multiple cells. In this respect the architecture is somewhat forgiving.

In spite of the shared memory mechanisms that the Exemplar incorporates, great care was required in organizing the computation in order to achieve good performance and scaling characteristics. Latency management was the most demanding aspect. This diminished significantly many advantages that the shared memory hardware provides towards ease-of-use. Basically, data and work partitioning for processor and cache locality is nec-

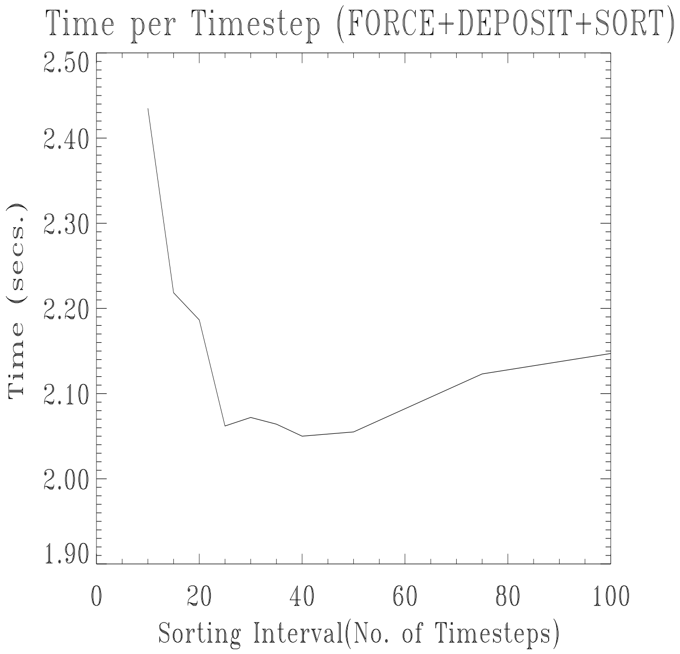

Figure 21: The sum of the average time per timestep for the routines FORCE, DEPOSIT and SORT as a function of the interval between calls to SORT, for the $30 \times 90 \times 120$ problem size on 15 processors.

essary for high performance on the Exemplar, giving the coding a 'message passing' feel.

Appendices

A A Parallel Vector Scan-add.

Suppose we have eight processors, each storing a different element of an array `a`. We want to sum the elements of `a` and store the sum in the first processor. In a naive implementation processor 8 would pass `a(8)` to processor 7 where it would be combined with `a(7)` and the sum would be passed on to processor 6. Eventually after seven of these communication steps the correct total would be accumulated in processor 1.

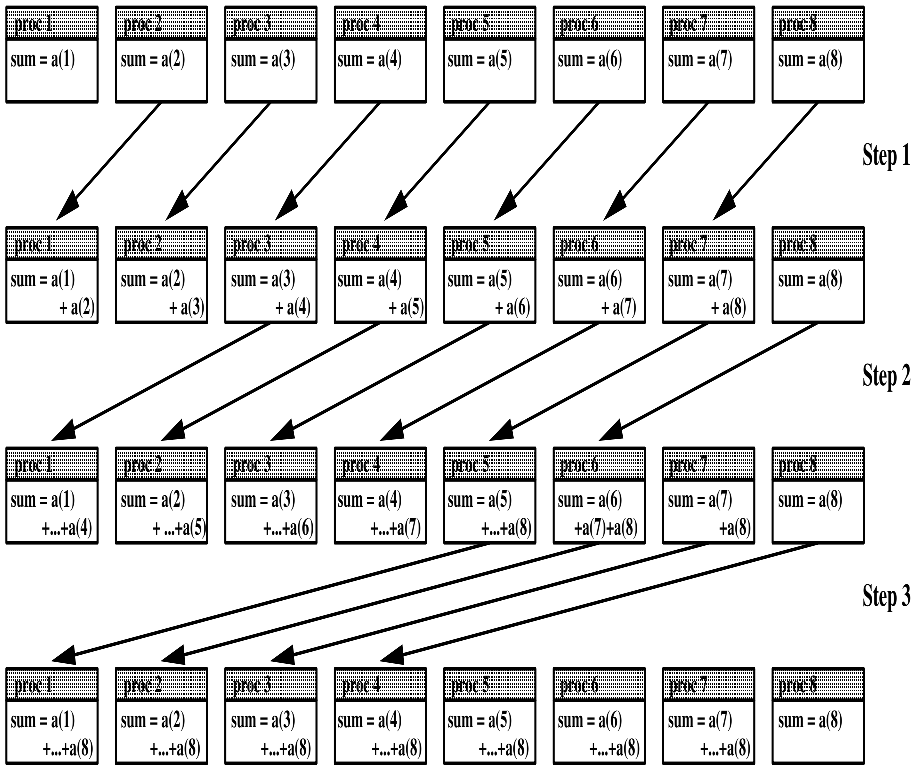

Figure 22: A scan-add for eight processors.

A vector scan-add does this faster, in a way which exploits the parallelism provided by the eight processors. This is illustrated in figure 22. In each processor we establish a variable called `sum`, and copy the local element of `a` into it. Now each processor sends its value of `sum` to its left neighbor, where it is added to the existing value of `sum`. We repeat this process, but this time the processors send their `sum` two processors to the left. Finally, we repeat the process sending `sum` four processors to the left. Processor 1 now contains the correct sum as required. This took $\ln_2 8 = 3$ steps. In general, if the execution time for a communication step is independent of the communication distance, this procedure scales as $\ln_2 N$ where N is the number of processors involved in the summing operation.

On the MasPar, using Xnet communications, the execution times are dependent on the communication distance. However because of the Xnet topology the communication distance is limited by the linear dimensions of the processor array.

Acknowledgments

Thanks are due to a number of people who took the time to review the manuscript and suggest improvements. These include Dr. Farzad Kaziminezhad.

References

[1] R.W. Hockney and J.W. Eastwood, Computer Simulation Using Particles, Institute of Physics, (1988).

[2] R.W. Hockney, S.P. Goel, and J.W. Eastwood, "A 10,000 particle Molecular Dynamics Model with Long-Range Forces", *Chem. Phys. Lett.*, **21**, 589, (1973).

[3] R.W. Hockney, S.P. Goel, and J.W. Eastwood, "Quiet High-Resolution Computer Models of a Plasma", *Journal of Computational Physics*, **14**, 148, (1974).

[4] C.K. Birdsall and A.B. Langdon, Plasma Physics via Computer Simulation, McGraw-Hill Inc., (1981).

[5] G.E. Blelloch, Vector Models for Data-Parallel Computing, MIT Press, (1990).

[6] F.H. Harlow, "The Particle-in-cell Computing Method in Fluid Dynamics", *Methods Comput. Phys.*, **3**, 319, (1964).

[7] A. Leonard, "Vortex Methods for Flow Simulation", *Journal of Computational Physics*, **37**, 289, (1980).

[8] R.W. Hockney, "Measurement of Collision and Heating Times in a Two-Dimensional Thermal Computer Plasma", *Journal of Computational Physics*, **8**, 19, (1971).

[9] H.M.P. Couchman, "Mesh-refined P^3M : a fast adaptive N-body algorithm", *Astrophysical Journal Letters*, **368**, 23, (1991).

[10] E.J. Horowitz, "Vectorizing the Interpolation Routines of Particle-in-Cell Codes", *Journal of Computational Physics*, **68**, 56, (1987).

[11] J. Binney and S. Tremaine, Galactic Dynamics, Princeton University Press, (1987).

[12] M.D. Smith, "Tracing with PIXIE", (1991).

[13] A. Nishiguchi, S. Orii and T. Yabe, "Vector Calculation of Particle Code", *Journal of Computational Physics*, **61**, 519, (1985).

[14] A. Héron and J.C. Adam, "Particle Code Optimization on Vector Computers", *Journal of Computational Physics*, **85**, 284, (1989).

[15] D.V. Anderson and D.E. Shumaker, "Hybrid Ordered Particle Simulation (HOPS) code for plasma modeling on vector-serial, vector-parallel,and massively parallel computers", *Computer Physics Communications*, **87**, 16, (1995).

[16] K.E. Batcher, "Sorting Networks and their applications", *Spring Joint Computer Conf., AFIPS Proc.*, **32**, 307, (1968).

[17] W.L. Hightower, J.F. Prins, and J.H. Reif, "Implementations of Randomized Sorting on Large Parallel Machines", *Proc. 4th ACM Symposium on Parallel Algorithms and Architectures*, (1992).

[18] P. MacNeice, C. Mobarry, and K. Olson, "Particle-Mesh Techniques on the MasPar" *NASA Technical Memorandum* 104632, (1996).

[19] D. W. Walker, "Characterizing the parallel performance of a large scale, particle-in-cell plasma simulation code", *Concurrency: Practice and Experience*, **2**, 257, (1990).

[20] V. K. Decyk, S. R. Karmesin, A. de Boer and P. C. Liewer, "Optimization of Particle-in-Cell Codes on RISC Processors", *http://olympic.jpl.nasa.gov/Reports/SNOptPics.html*,(1996).

[21] P. C. Liewer and V. K. Decyk, "A General Concurrent Algorithm for Plasma Particle-in-Cell Simulation Codes", *Journal of Computational Physics*, **85**, 302, (1989).

[22] R. D. Ferraro, P. C. Liewer and V. K. Decyk, "Dynamic Load Balancing for a 2D Concurrent Plasma PIC Code", *Journal of Computational Physics*, **109**, 329, (1993).

[23] P. M. Campbell, E. A. Carmona and D. W. Walker, "Hierarchical Domain Decomposition With Unitary Load Balancing For Electromagnetic Particle-In-Cell Codes", Proceedings of the Fifth Distributed Memory Computing Conference, 943, (1990).

[24] J. Villasenor and O. Buneman, "Rigorous charge conservation for local electro-magnetic field solver", *Computer Physics Communications*, **69**, 306, (1992).

[25] P. MacNeice, C.M. Mobarry, J. Crawford and T.L. Sterling, "Preliminary Insights on Shared Memory PIC Code Performance on the Convex Exemplar SPP1000" *submitted to Frontiers'96*, (1996).

[26] J. S. Huang and Y. C. Chow. "Parallel Sorting and data partitioning by sampling", *Proceedings of the IEEE Computer Society's Seventh International Computer Software and Applications Conference*, 627, (1983).