PHYSICAL AND NUMERICAL METHODS OF SPEEDING UP PARTICLE CODES AND PARALLELING AS APPLIED TO RF DISCHARGES

by

E. Kawamura, C. K. Birdsall and V. Vahedi

Memorandum No. UCB/ERL M99/58

1 December 1999

PHYSICAL AND NUMERICAL METHODS OF SPEEDING UP PARTICLE CODES AND PARALLELING AS APPLIED TO RF DISCHARGES

by

E. Kawamura, C. K. Birdsall and V. Vahedi

Memorandum No. UCB/ERL M99/58

1 December 1999

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Physical and Numerical Methods of Speeding Up Particle Codes and Paralleling as Applied to RF Discharges

E. Kawamura[†], C. K. Birdsall[†], and V. Vahedi[*]

[†] EECS Dept., 195M Cory Hall
University of California at Berkeley
Berkeley, CA 94720-1770

[*] Lam Research Corp.
4650 Cushing Parkway
Fremont, CA 94538-6470

Abstract

We demonstrate means, both physical and numerical, for speeding up particle-in-cell (PIC) simulations of RF discharges. These include implicit movers, longer ion timesteps, lighter mass ions, different weights for electrons and ions, and improved initial density profiles. By using these methods (singly or together) on Ar and O_2 RF discharges, we were able to achieve speedups of 6 to 30 times with single processor machines. In electrostatic 1d3v PIC simulations of RF discharges, the field solve is typically less than 1 percent of the work load. Even for 2d3v PIC simulations, the field solve can be a small percentage of the work load especially when FFT methods are used to solve for the field. Thus, we can obtain significant gains by just paralleling the particle processing (e.g., pushing/accumulating) without paralleling the field solve. We applied this simple scheme to conduct 1d3v and 2d3v PIC simulations of Ar RF discharges on 2 and 4 CPU symmetric multiprocessor (SMP) machines and on a distributed network of workstations (NOW). For a fixed number of grid points, the speedup for this parallel particle processing became more linear with increasing particle number. The combination of single processor methods and paralleling makes run-times for PIC codes more competitive with other types of codes.

1 Introduction

Particle-in-cell (PIC) simulations of RF discharges are attractive because the fields and energy distributions can be obtained self-consistently from first principles [1]. No assumptions need be made about the electric field or the bulk plasma velocity distributions, and the boundary conditions are realistic for both particles and fields. Collisions can be included in PIC simulations by coupling PIC methods with Monte-Carlo collisions (MCC). A detailed key reference for PIC-MCC is provided by V. Vahedi and M. Surendra (1995) [2]. The only disadvantage of PIC-MCC is that it tends to be computationally expensive compared to other numerical methods. Plasma simulations using fluid codes (solving several moments of the Boltzmann equation) tend to run faster than PIC-MCC codes, but make assumptions about the velocity distributions of the electrons and ions (e.g., assume velocity distributions are Maxwellian) and ignore kinetic effects such as stochastic electron heating. Our goal is to keep the first-principle, self-consistent approach in PIC-MCC while accelerating PIC-MCC codes so that they are more useful to modelers and experimentalists alike who are designing new plasma processing equipment.

The current "rule-of thumb" on PIC-MCC simulations of RF discharges is that one needs to run the simulations for about 1000 RF cycles in order to reach equilibrium. Depending on the simulation, this may take 10^6 or more timesteps, with from $10^4 - 10^6$ particles. Standard explicit PIC-MCC codes with no speedup applied, running on a single processor moderately fast workstation (e.g., a 233 MHz DEC Alpha workstation), do not begin to complete such a run in one hour.

In this paper, we will briefly review the standard PIC-MCC model used in our simulations. Next, we will discuss various methods of speeding up PIC-MCC codes on single processor machines. These include implicit movers, longer ion timesteps, lighter mass ions, different weights for electrons and ions, and improved initial density profiles. We will then apply these methods to three different 1d3v RF discharges (a 100 mT argon discharge, a 10 mT argon discharge, and a 100 mT oxygen discharge). By "1d3v", we mean one dimensional displacements (1d) with three velocity components (3v).

Finally, we will introduce and discuss a parallel particle processing scheme for PIC-MCC simulations. In a typical electrostatic 1d3v PIC-MCC simulation of an RF discharge, most of the time is spent in processing (e.g., pushing or accumulating) the particles rather than in the field solve. This is true even for 2d3v PIC-MCC simulations, especially when we use Fast Fourier Transform (FFT) field solve methods. Thus, we can obtain significant speed gains by just paralleling the particle processing without paralleling the field solve. We will apply this scheme to both 1d3v and 2d3v argon RF discharges.

1.1 Bounded plasma model

Figure 1 shows the Plasma Device Planar 1d3v model used in our well known and widely used code called PDP1 [1, 3]. The metal electrodes at $x = 0$ and $x = L$ bound the plasma region and have surface charge, both induced by the fields in the plasma and from charges deposited from the external circuit.

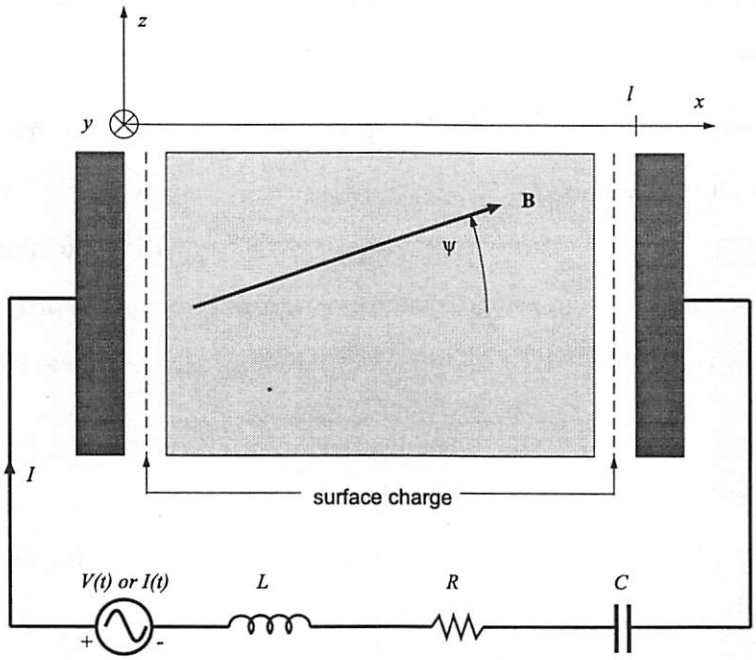

Figure 1: Model for the plasma device code PDP1, showing all of the elements of a whole device, with the plasma between the electrodes and the external driving circuit outside, all solved simutaneously. From Verboncoeur et al [4] (1993).

PIC-MCC codes follow particles in their own and applied electric (E) and magnetic (B) fields, from first principles, self-consistently, with the collisions done by the MCC method. Each computer particle is a "superparticle" which represents 10^6-10^9 real particles (electrons or ions). The simulation must run with a sufficient number of particles in order to minimize the discrete particle noise. For each timestep: (i) The charge and current densities ρ and J are obtained by a linear weighting of the particles to the spatial grid. (ii) ρ and J are used to solve for E and B on the grid. For electrostatic simulations, Poisson's equation is used to solve for E. For electromagnetic simulations, the full set of Maxwell's equations is used to solve for E and B. (iii) E and B are linearly weighted back to each particle position in order to determine the force on each particle. (iv) The equations of motion are used to advance the particles to new positions and velocities. (iv) The boundaries are checked, and out of bounds particles are removed while particles injected at the boundaries are added to the simulation. (v) A Monte-Carlo collision handler checks for collisions and adjusts the particle velocities accordingly. Figure 2 shows the PIC-MCC flow chart for the bounded plasma model.

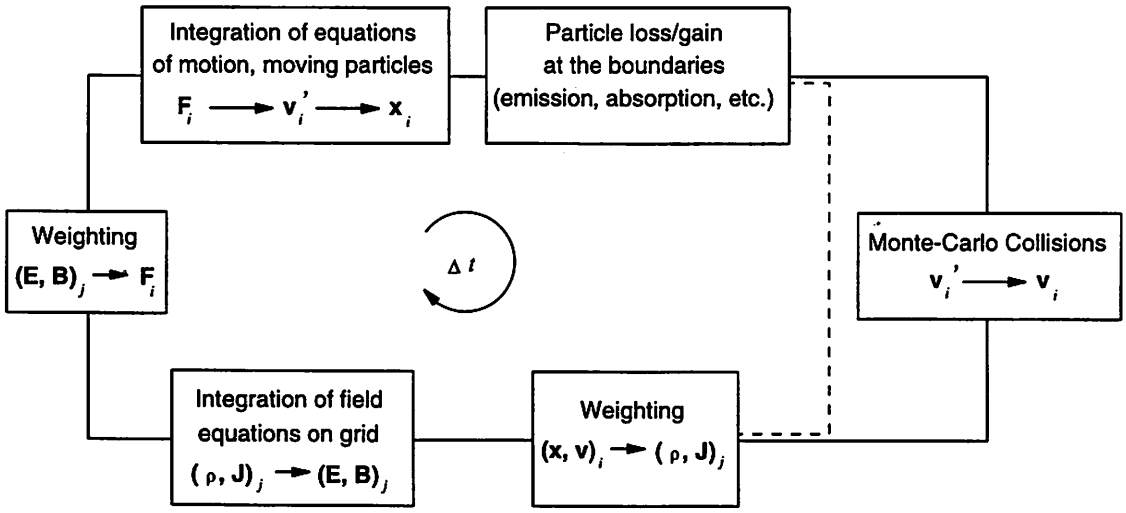

Figure 2: PIC-MCC Flow chart for bounded and collisional models. From Vahedi and Surendra [2] (1995).

1.2 Monte-Carlo collisions model

Let us briefly describe the MCC model used in our simulations. A more detailed discussion can be found in reference [2]. The kinetic energy $\mathcal{E}_i$ of the ith particle of species s is given by

$$\mathcal{E}_i = m_s v_i^2/2 = m_s(v_{ix}^2 + v_{iy}^2 + v_{iz}^2)/2. \tag{1}$$

The total collision cross-section $\sigma_T(\mathcal{E}_i)$ for this particle is given by

$$\sigma_T(\mathcal{E}_i) = \sum_{j=1,N} \sigma_j(\mathcal{E}_i), \tag{2}$$

where $j = 1$ to N denotes the different types of collisions for species s, and $\sigma_j(\mathcal{E}_i)$ is the cross-section for the jth type of collision. A constant maximum collision frequency ν_{max} for a species s is defined by

$$\nu_{max} = \max\{n_t \sigma_T v\}_{x,\mathcal{E}} = \max\{n_t(x)\}_x \max\{\sigma_T(\mathcal{E})(2\mathcal{E}/m_s)^{\frac{1}{2}}\}_{\mathcal{E}}, \tag{3}$$

where $n_t(x)$ is the density of the target species. If N_s is the total number of particles of species s in the simulation, then, the maximum number of particles of species s colliding in a time interval Δt is given by

$$N_{coll} = N_s P_s = N_s(1 - \exp(-\nu_{max}\Delta t)). \tag{4}$$

For each species s, N_{coll} particles are chosen randomly from the simulation to undergo collisions. A random number R between 0 and 1 is used to determine the type of collision in the following manner,

$$\begin{aligned}
R &\leq \nu_1(\mathcal{E}_i)/\nu_{max} && \text{(Collision Type 1)},\\
\nu_1(\mathcal{E}_i)/\nu_{max} < R &\leq (\nu_1(\mathcal{E}_i) + \nu_2(\mathcal{E}_i))/\nu_{max} && \text{(Collision Type 2)},\\
&\vdots && \\
\sum_{j=1,N} \nu_j(\mathcal{E}_i)/\nu_{max} &< R && \text{(No Collision)}.
\end{aligned} \tag{5}$$

Since some of the N_{coll} particles have a possibility of not colliding (i.e., undergoing a "null collision"), this model is called the "null collision" model. Typically, for our applications, $N_{coll} \ll N_s$, thus the null collision model has a significant speed advantage over other collision models which query each of the N_s particles for collisions every timestep.

1.3 Explicit leap frog mover

Our base PIC-MCC code uses the standard "leap frog" finite difference scheme to advance the particles:

$$\begin{aligned} v^{n+1/2} &= v_{n-1/2} + \Delta t a^n(x^n) \\ x^{n+1} &= x^n + \Delta t v^{n+1/2}, \end{aligned} \tag{6}$$

where $a^n(x^n) = qE^n(x^n)/m$ is the acceleration. The superscripts in the equations refer to the time level. For example, x^{n+1} refers to the value of x at time $t = (n+1)\Delta t$, where Δt is the timestep. Note that the leap frog scheme is "explicit" since x^{n+1} and $v^{n+1/2}$ are determined only from values of x, v and a at earlier time levels.

In the Birdsall and Langdon simulation text [5], the stability and accuracy of the leap frog scheme is tested by applying it to the simple harmonic oscillator model: $\ddot{x} = -\omega_0^2 x$. The exact acceleration $a^n = -\omega_0^2 x^n$ is substituted into the finite difference scheme (6) to obtain,

$$\frac{x^{n+1} - 2x^n + x^{n-1}}{\Delta t^2} = -\omega_0^2 x_n. \tag{7}$$

This equation is readily solved by assuming solutions of the form: $x^n = x_0 \exp(-in\omega\Delta t)$. Substituting this, into (7), we obtain,

$$\sin(\omega\Delta t/2) = \pm\omega_0\Delta t/2 \tag{8}$$

For $\omega_0\Delta t/2 \ll 1$, $\omega \approx \omega_0$, as desired. However, if $\omega_0\Delta t > 2$, the real solution for ω becomes complex with growing and decaying roots, implying numerical instability. Electron plasma oscillations are simple harmonic oscillations and are typically the highest frequency oscillations in our simulations. Thus, the stability criterion for the standard explicit leap frog mover is $\omega_0\Delta t \leq 2$. Accuracy for the leap frog scheme is also obtained from (8). For $\omega_0\Delta t \ll 1$,

$$\omega/\omega_0 = 1 + O(\omega_0\Delta t)^2, \tag{9}$$

showing a quadratic phase error term. For simulations which use the leap frog mover, we typically take $\omega_0\Delta t \lesssim 0.2$.

2 Speedup Methods for Single Processor Computers

Numerous methods for speeding up PIC-MCC codes on single processor machines have been published in detail and widely applied, but usually one at a time. See, for example, Table 1 in Vahedi et al [6]. We will apply several speedup methods together to electrostatic simulations of RF discharges. Our base PIC-MCC code uses an explicit leap frog mover to advance the particles, with no speedup applied, hence having a speed Gain = 1. "Gain" is defined as: (base run time)/(sped up run time); a number ≥ 1.

2.1 Implicit scheme

Implicit PIC-MCC codes allow much larger timesteps Δt and grid spacings Δx while maintaining stability and accuracy, due to attenuating fields at high frequencies and (usually) at short wavelengths. For detailed anaylses of implicit methods, please see the texts by Birdsall and Langdon (Chap. 9, 1985) [5] and by Brackbill and Cohen (Chaps. 9 and 11, 1985) [7]. The direct implicit scheme summarized below is also analyzed in detail in reference [8].

An example of a direct implicit particle advance is the following:

$$\begin{aligned} \bar{E}^n &= \frac{1}{2}[\bar{E}^{n-1} + E^{n+1}(x^{n+1})] \\ v^{n+1/2} &= v_{n-1/2} + q\Delta t\bar{E}^n/m \\ x^{n+1} &= x^n + \Delta t v^{n+1/2}. \end{aligned} \tag{10}$$

The scheme is "implicit" since E^{n+1} must be known to advance x from time level n to $n+1$, but x_{n+1} is needed to determine E^{n+1}. In otherwords, future time level information is needed to advance the particles. The scheme is "direct" since a direct solution for E^{n+1} can be obtained without iterating the particle advance, provided that $q\Delta t^2|\nabla E|/(2m) \ll 1$. See References [5, 7, 8] for the details.

The recursive filtering of the E field in (10) damps out high frequency modes. If we apply the simple harmonic oscillator model to this implicit scheme (as was done for the leap frog

mover), we obtain the following equations, which are valid for $\omega_0 \Delta t \ll 1$ [8]:

$$\begin{aligned} \mathrm{Re}(\omega)/\omega_0 &= 1 + O(\omega_0 \Delta t)^2 \\ \mathrm{Im}(\omega)/\omega_0 &= -O(\omega_0 \Delta t)^3. \end{aligned} \tag{11}$$

The real part shows a quadratic phase shift term while the negative and higher order imaginary part shows amplitude damping. High frequency ($\omega_0 \Delta t \gg 1$) oscillations are strongly damped while low frequency ($\omega_0 \Delta t \ll 1$) oscillations are hardly affected. Thus, the physics that is not accurately resolved (i.e., by the use of large timesteps) is removed by the damping.

The implicit movers are more complex and take more computations per timestep than explicit movers. However, since they do not rely only on past time level information to advance the particles, they are stable over a wider range of timesteps and grid spacings than explicit movers. Thus implicit movers can speedup simulations by allowing larger Δt and Δx than explicit movers. But by increasing Δt and Δx, information about high frequency or short wavelength phenomena is automatically lost. Additionally, implicit methods tend to damp out such phenomena anyway. Thus, implicit methods should only be applied in cases where high frequency or short wavelength phenomena are not important, (e.g., RF discharges where we need only resolve the sheath width and the RF frequency plus a few harmonics).

2.2 Subcycling, $\Delta t_i \gg \Delta t_e$

With the short timesteps used in order to satisfy stability and accuracy conditions for the electrons, the far heavier ions move hardly at all in one electron timestep Δt_e. Hence, the ions might be moved less frequently, i.e., every kth electron timestep ($\Delta t_i = k \Delta t_e$), where k may be 10 to 100, depending on the ion mass. This is called "subcycling". See the texts Birdsall and Langdon (Chap. 9, 1985) [5] and Brackbill and Cohen (Chap.10, 1985) [7] for the details. In electropositive discharges, the number of ions and electrons are about the same so that the maximum Gain achievable from subcycling is 2. However, for electronegative discharges, the number of ions can far exceed the number of electrons so that subcycling can significantly reduce the simulation time. We will see examples of this later.

2.3 Improved initial density profiles

We usually start our PIC-MCC simulations with spatially uniform ion and electron density profiles. The density profiles then evolve to their equilibrium states. This suggests that the time to reach equilibrium could be improved by starting off with non-uniform initial density profiles that are close to their final equilibrium values. These profiles may be calculated analytically or deduced from previous runs.

2.4 Light ions

One method of obtaining a better initial starting point for a simulation is the use of "light ions". The light ion speedup method is done in two steps. First, we replace the real ion masses M_{real} with the light ion masses M_{light}, and run the discharge until it comes to a "light ion" equilibrium. Reducing the mass of the ions increases their speed, which enables them to reach an equilibrium state in a smaller number of RF cycles, and hence less computer time. Then we restore the real masses M_{real}, (keeping the same kinetic energies, by decreasing the ion velocities by the factor $\sqrt{M_{real}/M_{light}}$), and run until equilibrium again. There is a short lived transient in restarting with the real masses, then a more gradual transition to the final "real ion" equilibrium. The hope is that the overall running time is less than that running with M_{real} throughout (it is). Since the speed of the light ions is faster by the factor $\sqrt{M_{real}/M_{light}}$, we expect the light ions to reach equilibrium faster than real ions by this factor. The maximum Gain possible is also $\sqrt{M_{real}/M_{light}}$, assuming that the second equilibrium is reached relatively quickly.

Increasing the speed of the ions, however, increases the rate of loss of the ions to the walls. This means that we must increase the creation rate (ionization) in order to make up for this increased rate of loss. One way to increase the creation rate was to increase the electron-neutral cross-sections (including ionization) in our code by the factor $\sqrt{M_{real}/M_{light}}$ [9].

2.5 Variable particle weights

In electropositive discharges, the number of electrons is about the same as the number of ions; the charge of the positive ions is neutralized by the charge of the electrons. In contrast, for very electronegative discharges, the number of ions can greatly exceed the number of electrons; the heavy negative ions largely neutralize the heavy positive ions so that there is no need for a comparable electron population. In a typical oxygen RF discharge, there may be $\sim 10^2$ O_2^+ and O^- ions for every electron. Suppose we need atleast 10^3 electrons in our simulation to reduce numerical noise and to satisfy the condition that the number of electrons per Debye length λ_{De} be much greater than 1. Ordinarily, this would mean that we require at least $\sim 10^5$ O_2^+ and O^- ions in our simulation. Since processing the particles is typically the costliest part of the simulation, significant Gains can be achieved by reducing the number of computer particles. One way to reduce the number of computer particles is to weight the ion superparticles more heavily than the electron superparticles. For example, if an electron superparticle represents 10^8 real electrons, an ion superparticle can represent 10^9 real ions. Then, in the above example, we can reduce the number of required ions in the simulation from $\sim 10^5$ to $\sim 10^4$.

A "variable weighting" procedure for electronegative discharges has been developed and implemented by Cooperberg et al [10]. If there were no collisions, then the procedure would be fairly straightforward since there would be no interactions involving particles of different weights. But, because of collisions, the weight of each particle is a dynamic variable that is updated every time the particle undergoes a collision. Suppose in a chlorine discharge, we initially start with a weight $w = 1$ for electrons and $w = 10$ for Cl^- ions. Consider the reaction,

$$e^- + Cl_2 \rightarrow Cl^- + Cl.$$

Here, we must remove 1 electron and create 1/10 of a Cl^-. Even if the Cl^- particles start out with a high weight, the creation of low weight Cl^- particles, as in the above case, can still lead to the increase of Cl^- particles. The solution is to impose a cap on the number of particles per species. Once the maximum number of particles is reached for a species, instead of adding particles to the simulation, we increase the weight of existing particles. In

the above reaction, if the Cl^- number is at its maximum, we find two existing Cl^- close to the event and adjust the weights and velocities of these Cl^- to add the mass, momentum and kinetic energy required by the reaction. Two Cl^- are used since energy and momentum conservation cannot be both satisfied with only one Cl^-.

2.6 Constraints on Δt and Δx

We often try to speedup simulations by increasing the timestep Δt and the grid spacing Δx. But, in order to maintain stability and accuracy, we must obey certain constraints on Δt and Δx. Most of these limits are discussed in the simulation texts noted earlier. The higher Gains come when we run near these limits.

In a typical RF discharge, fast plasma phenomena may not be significant. Then, the highest frequency we need to resolve is the RF driving frequency ω_{rf}, plus a few harmonics, and the shortest length we need to resolve is the sheath width s. However, the accuracy criteria for PIC-MCC codes which use explicit movers require resolving the electron plasma frequency and the electron Debye length: $\omega_{pe}\Delta t \leq 0.2$, $\lambda_{De}/\Delta x \gtrsim 1$. (Typically, $\omega_{pe} \gg \omega_{rf}$ and $\lambda_{De} \ll s$.) On the other hand, by using implicit movers, we can relax these constraints to $\omega_{rf}\Delta t \ll 1$ and $\Delta x \ll s$.

An additional accuracy condition which must always be satisfied is the Courant condition: $v_s\Delta t_s/\Delta x < 1$, where v_s is the characteristic velocity of a particle of species s, and Δt_s is the timestep of a particle of species s. This ensures that most particles will not travel more than one cell per timestep and will sample the electric fields properly.

Finally, because our MCC model assumes one collision per particle per timestep, care must be taken to choose a Δt such that the probability of having a particle collide more than once per Δt is low. The collision probability of the jth particle in a timestep Δt is

$$P_j = 1 - \exp(-n_{gas}\sigma_T(\mathcal{E}_j)v_j\Delta t), \tag{12}$$

where v_j and $\mathcal{E}_j$ are the speed and kinetic energy of the jth particle, and $\sigma_T(\mathcal{E}_j)$ is its total collision cross section with the target density n_{gas}. Then, the collision probability for n

collisions in the same Δt is approximately P_j^n . The number of missed collisions per Δt is then proportional to

$$err = \sum_{k=2,\infty} P_j^k = P_j^2/(1-P_j). \tag{13}$$

For example, an $err = 0.01$ requires $P_j < 0.095$ and $n_{gas}\sigma_T(\mathcal{E}_j)\mathrm{v}_j\Delta t = \nu_j\Delta t < 0.1$. See Vahid and Surendra, 1995 [2] . In our Tables below, we calculate $\nu_{\max}\Delta t = n_{gas}\max\{\sigma_{T,s}(\mathcal{E})\mathrm{v}_s(\mathcal{E})\}_{\mathcal{E}}\Delta t$ which is larger than all the $\nu_j\Delta t$.

3 Applications of Speedup Methods for Single Processor Computers

In this section, we apply the speedup methods discussed in the previous section on three different models: a 100 mT argon discharge, a 10 mT argon discharge, and a 100 mT oxygen discharge. All of the simulations reported in this section were made on a modest 233 MHz DEC Alpha workstation using our 1d3v electrostatic PIC-MCC code PDP1. For each of the three models, we provide a Table which lists the Gains due to the speedup methods as well as the values of the various constraint terms (e.g., $\omega_{pe}\Delta t_e$) discussed in the previous section.

3.1 100 mT argon discharge

Our first step was to examine the 1000 RF cycle "rule of thumb" on our standard argon current driven model, as reported on earlier by V. Vahedi et al(1993) [6]. This 1d3v model consisted of a 100 mT parallel plate argon reactor with a 2 cm electrode spacing, driven by a sinusoidal current drive with an amplitude of 0.4 A and a frequency of 13.56 MHz. The runs were initiated with an equal number of electrons and ions, uniform in x, with Maxwellian electrons at $T_e = 2$ eV, and Maxwellian ions at $T_i = 0.03$ eV. These runs were made with about 2×10^4particles, half electrons, half ions. [See Table 1].

100 mT Ar

Model	Base	Implicit	Implicit, Subcycled
Total time(s)	7.1×10^4	1.75×10^4	1.1×10^4
GAIN	**1**	**4.1**	**6.5**
$v_{te}\Delta t_e/\Delta x$	0.43	0.69	0.69
$v_{imax}\Delta t_i/\Delta x$	0.013	0.021	0.42
$\nu_e\Delta t_e$	0.041	0.32	0.32
$\nu_i\Delta t_i$	0.0024	0.019	0.38
$\lambda_{De}/\Delta x$	2.7	0.53	0.53
$\omega_{pe}\Delta t_e$	0.16	1.3	1.3

Table 1: Argon 100 mT: base, implicit, implicit plus subcycling runs. Run time went from 20 hours to 3.1 hours.

3.1.1 Base run

We conducted the base run (explicit coding with no speedup applied) for about 1000 RF cycles $\approx$ 2 million Δt_{base}. Here, $\Delta t_{base} \equiv$ timestep used for base run. The computing time was about 7.1×10^4 seconds, roughly 20 hours. However, the actual time for the base model to reach equilibrium, judging from the behavior of the number of particles and other diagnostics, was less, at about 500 RF cycles. We assume that a simulation has reached equilibrium when the number of particles and other diagnostics (i.e., density, electric field, potential profiles, temperature, energy distributions) do not change "appreciably" when we run the simulation longer.

3.1.2 Implicit case

As mentioned earlier, implicit schemes allow a larger timestep Δt while maintaining numerical stability and accuracy. By using an implicit scheme, we were able to increase the timestep by a factor of 8 ($\Delta t_{implicit} = 8\Delta t_{base}$). When run for 1000 RF cycles, the simulation took

about 1.75×10^4 seconds (4.9 hours). Hence, the running time was reduced by a factor of $7.1\times10^4/1.75\times10^4 = 4.1$, or Gain = 4.1. We did not obtain a Gain of 8 as might be expected because running the code implicitly was roughly twice as costly as running it explicitly. A profile of the explicit and implicit codes showed that the percentage of time spent in the mover was 37% in both cases but the more complex implicit mover took 1.85 times longer per call than the explicit mover. Also even though the collision handler was called only every $8\Delta t_{base}$, it took 4.5 times as long per call in the implicit case as the explicit case because the number of collisions per call increases with Δt. [See (4).] This suggests that at lower pressures, where collision handlers play smaller roles, we can expect higher Gains for implicit runs.

3.1.3 Implicit and subcycling

We further reduced the run time from the implicit case by using longer ion timesteps. We chose a subcycling factor $k = 20$, which means that the ions were processed only every $20\Delta t_e$. When run for 1000 RF cycles, the computing time was about 1.1×10^4 seconds. This represents a Gain = $1.75\times10^4/1.1\times10^4 = 1.6$ over the implicit without subcycling case. The total Gain = $7.1\times10^4/1.1\times10^4 = 6.5$ over the base case.

In the non-subcyling case, for every $20\Delta t_e$, we process $20(1\times10^4) + 20(1\times10^4) = 4\times10^5$ particles. In the subcycling case, for every $20\Delta t_e$, we process $20(1\times10^4) + 1\times10^4 = 2.1\times10^4$ particles. This means that we expect a maximum Gain of about $4\times10^5/2.1\times10^4 = 1.9$ from subcycling. (As the subcycling factor k increases, the ion move becomes negligible and the Gain approaches 2.) However, only about 1.6 of the expected 1.9 was realized because, like the implicit case, the subcycling did not reduce appreciably the total time spent with the collision handler. Even though the collision handler is called less frequently for the ions (every $\Delta t_i = k\Delta t_e$ rather than every Δt_e), the number of ion collisions per call increases with increasing Δt_i. As with the implicit case, at lower pressures we should expect higher Gains due to subcycling since the collision handler would play a lesser role. We should also note that albeit small in 1d3v simulations, the time to calculate fields is also not reduced by subcycling.

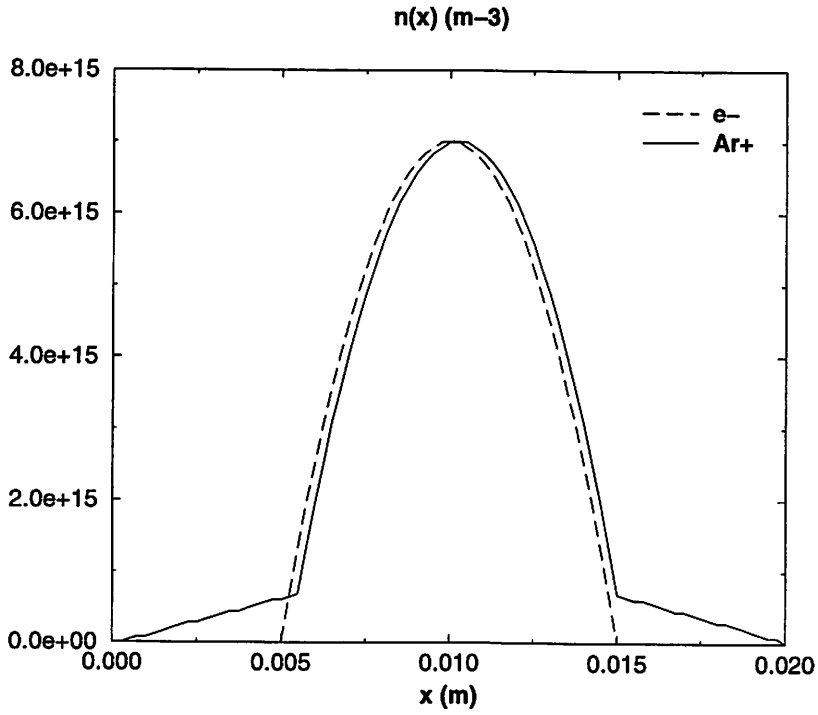

Figure 3: The initial density profiles used in simulating a 100 mT argon discharge. In the bulk, the ion and electron densities are equal and parabolic. In the sheath, the ion density drops off linearly and the electron density is zero.

3.1.4 Improved initial density profile

We repeated the base run described above but starting with an improved initial density profile shown in Fig. 3. As in the previous runs, the initial electron and ion velocity distributions are Maxwellian with $T_e = 2$ eV and $T_i = 0.03$ eV. In the bulk, the electron and ion densities are equal and parabolic in shape. In the sheath, the electron density is zero, while the ion density falls off linearly. We chose bulk parabolic profiles since they are similar to the cosine curves which are solutions to the diffusion equations. We chose to start off with zero electron density in the sheath since the electron density drops off sharply in the sheath. For the ions, the density drop off in the sheath is less abrupt, and we chose to model it with a straight line. Because of our improved starting point, our simulation of the 100 mT argon discharge reached equilibrium in about 125 RF cycles rather than 500 RF cycles. This represents a Gain = 4. So, good initial estimates of the density profiles can significantly shorten the simulation time.

10 mT Ar

Model	Base	Light ions	Light ions, implicit
Total time(s)	5×10^4	1.2×10^4	2.7×10^3
GAIN	**1**	**4.1**	**19**
$v_{te}\Delta t_e/\Delta x$	0.13	0.13	1.0
$v_{imax}\Delta t_i/\Delta x$	6.7×10^{-3}	0.12	0.96
$\nu_e\Delta t_e$	4.1×10^{-3}	4.1×10^{-2}	0.32
$\nu_i\Delta t_i$	2.4×10^{-4}	2.4×10^{-3}	0.019
$\lambda_{De}/\Delta x$	1.0	1.0	1.0
$\omega_{pe}\Delta t_e$	0.16	0.16	1.3

Table 2: Argon 10 mT: base, light ion, light ion plus implicit runs. Run time went from 13.9 hours to 45 minutes.

3.2 10 mT argon discharge

Our second model was also an argon discharge but with a lower pressure of 10 mT, a larger electrode spacing of 5 cm, and a sinusoidal current drive of amplitude 0.6 A and frequency 13.56 MHz. As with the 100 mT argon discharge, the runs were initiated with an equal number of electrons and argon ions uniform in x, with Maxwellian electrons and ions at T_e = 2 eV and T_i = 0.03 eV. The number of computer particles used in this simulation was roughly 1.3×10^4 instead of the 2×10^4 used in the 100 mT simulation. [See Table 2].

3.2.1 Base run

For the base run, the discharge reached equilibrium in about 1000 RF cycles. The run took about 5×10^4 seconds or 14 hours, slightly longer than $(1.3\times10^4/2\times10^4)\times$(time to run 100 mT argon model for 1000 RF cycles). In 1d3v PIC-MCC simulations of RF discharges, most of the time is spent processing particles so that the computation time is roughly proportional to the number of particles.

3.2.2 Light ions

As discussed before, the light ion speedup method requires two steps. First, we ran the discharge with light ions until we reached a "light ion" equilibrium. We used $M_{light}/M_{real} = 100$. In addition to increasing the electron-neutral cross-sections by the factor $\sqrt{M_{real}/M_{light}} = 10$, we needed to double the amplitude of the current drive to 1.2 A in order to maintain the discharge. This required increase in current can be estimated from analysis as shown in Appendix A. The light ions reached equilibrium at about 128 RF cycles $\ll$ 1000 RF cycles. Next, we switched back to the real masses, the original current drive and cross-sections. But we, kept the same kinetic energies as the light ions by decreasing the ion velocities by a factor of $\sqrt{M_{real}/M_{light}} = 10$. The simulation only needed an additional 128 RF cycles to reach a final "real ion" equilibrium. The total running time of this two step simulation was 1.2×10^4 seconds which is a factor of 4.1 less than the base simulation running time of 5×10^4 seconds; Gain = 4.1.

3.2.3 Light ions and implicit

Further reduction in run time can be made by using an implicit scheme together with the light ion speedup. This enabled us to increase the timestep by a factor of 8: $\Delta t_{implicit} = 8\Delta t_{base}$. The running time was 2.7×10^3 seconds, representing a Gain of 4.4 over the light ions only case. As in the 100 mT argon discharge, we did not get a Gain of 8 because the implicit code is roughly twice as costly as the explicit code. As expected, the Gain of 4.4 in the 10 mT argon model is larger than the Gain of 4.1 in the 100 mT argon model because, at the lower pressure, less time is spent with the collision handler. The total Gain in time from using a combination of light ions and an implicit code is Gain = $4.4\times4.1 = 19$.

3.3 100 mT oxygen discharge

Our third model is a 100 mT current-driven oxygen discharge. As with the 100 mT argon model, the length of the system was 2 cm, and the amplitude and frequency of the current

100 mT O_2

Model	Base	Subcycling	Subcycling, Weighting
Total time(s)	3×10^5	1.5×10^4	1×10^4
GAIN	**1**	**20**	**30**
$v_{te}\Delta t_e/\Delta x$	0.22	0.22	0.22
$v_{imax}\Delta t_i/\Delta x$	6.2×10^{-3}	0.5	0.5
$\nu_e\Delta t_e$	0.071	0.071	0.071
$\nu_i\Delta t_i$	8.9×10^{-4}	0.071	0.071
$\lambda_{De}/\Delta x$	5.0	5.0	5.0
$\omega_{pe}\Delta t_e$	0.043	0.043	0.043

Table 3: Oxygen 100 mT: base, subcycling, subcycling and variable weights runs. Run time went from 3.5 days to 2.8 hours.

drive was 0.4 A and 13.56 MHz respectively. The oxygen model had three species, electrons, O_2^+ ions and O^- ions. Because oxygen is electronegative, we expect the O_2^+ and O^- population to be much larger than the electron population. Hence, we started the simulation with about 5×10^4 O_2^+, 5×10^4 O^- ions and 5×10^3 electrons. The initial densities were all uniform in x, with Maxwellian electrons at $T_e = 2$ eV and Maxwellian $O_2^+$and O^- at $T_i = 0.03eV$. [See Table 3].

3.3.1 Base run

We conducted the base run for 1000 RF cycles. Because of the large number of particles, the run took 3×10^5 seconds or 3.5 days (slightly less than $(1.05\times10^5/2\times10^4)\times$(time to run 100 mT argon model for 1000 RF cycles). But, even after 1000 RF cycles, the density profiles were still evolving, suggesting that the simulation did not reach equilibrium. The number of ions in the simulation did not change appreciably over the 1000 RF cycles, but the number of electrons did drop sharply to about 1.5×10^3 electrons.

3.3.2 Subcycling

In an argon discharge, where there is roughly an equal number of electrons and ions, the maximum Gain due to subcycling is limited to 2. However, in an oxygen discharge the number of ions can be much larger than the number of electrons so that the Gain due to subcycling can be far greater. We used a subcycling factor of $k = 80$; that is, the ions were processed only every $80\Delta t_e$. When run for 1000 RF cycles, the computing time was 1.5×10^4 seconds, representing a Gain of $3\times10^5/1.5\times10^4 = 20 \gg 2$. Towards the end of our oxygen simulation, we had 5×10^4 O_2^+ ,5×10^4 O^-, and 1.5×10^3 electrons. In the non-subcycling case, for every 80 Δt_e, we process $80(5\times10^4) + 80(5\times10^4) + 80(1.5\times10^3) = 8.12\times10^6$ particles. In the subcyling case, for every 80 Δt_e, we process $5\times10^4 + 5\times10^4 + 80(1.5\times10^3) = 2.2\times10^5$ particles. This means that we expect a maximum Gain of about $8.12\times10^6/2.2\times10^5 = 37$. We obtained a lesser gain because we initiated the simulation with 5×10^3 electrons. Also, as mentioned in the 100 mT argon case, although subcycling reduces the time spent pushing and accumulating particles (typically the costliest procedures), it does not reduce the time spent handling collisions or calculating fields.

3.3.3 Subcycling and variable weights

Next, we did a simulation using both subcycling and variable weights. As before, we used a subcycling factor $k = 80$. We used a variable weight factor $w = 10$; that is, each ion computer particle represented 10 times as many real particles as an electron computer particle. This reduced the number of O_2^+ and O^- from 5×10^4 each to 5×10^3 each. So instead of processing, $5\times10^4 + 5\times10^4 + 80(1.5\times10^3) = 2.2\times10^5$ particles every $80\Delta t_e$, we only had to process $5\times10^3 + 5\times10^3 + 80(1.5\times10^3) = 1.3\times10^5$ particles every $80\Delta t_e$. This implies a Gain = $2.2\times10^5/1.3\times10^5 = 1.7$. The subcycling run with variable weights took 1×10^4 seconds to run 1000 RF cycles. Compared to the subcycling run without variable weights the Gain = $1.5\times10^4/1\times10^4 = 1.5$, very close to the expected gain of 1.7. The total Gain from subcycling and variable weights is Gain = $20\times1.5 = 30$. Note that increasing Δt_e by using an implicit scheme will add little Gain here, as there are relatively few electrons to move, and the implicit

code is roughly twice as costly as the explicit one.

4 A Parallel PIC-MCC Speedup Method

Parallel PIC methods have been previously studied by various researchers using different schemes. See for example Ref. [11] and its references. Most parallel PIC schemes employ a "Eulerian" decompostion scheme in which each processor is assigned a fixed spatial partition of the grid as well as all the particles within that partition. The processors also share the field solve computation.

Our parallel PIC scheme is complementary to the Eulerian decomposition scheme described above. In our 1d3v PIC-MCC simulations of RF discharges, the field solve typically takes only $\sim$1 % of the total computation time. Even in 2d3v, the field solve can be a small fraction of the total workload when fast field solve techniques such as FFT schemes are used. This suggests that significant speed Gains can be achieved by a simple scheme in which only the particle processing (e.g., pushing and accumulating) is paralleled without paralleling the field solve.

4.1 Description of parallel particle processing

In our scheme, the particles are randomly and equally divided among the CPUs regardless of their positions within the grid. Each CPU sees the entire spatial grid but only a random portion of the particles. The steps to the parallel particle processing scheme are as follows:

1. Each CPU advances its allotment of particles and linearly weights its particles to the grid.

2. The contributions of all the CPUs are summed to find the total densities of each species on the grid.

3. These densities are received by one of the CPUs. This CPU (designated the "root" CPU), uses the information to calculate the total charge density and solve for the

electric field. Then, it broadcasts the result to the other CPUs.

Alternatively, the total densities of the species can be broadcast to all the CPUs. Then, each CPU can conduct its own field solve to advance its particles.

Note that this parallel particle processing method yields significant gains only if the field solve is a small fraction of the total work load.

As described above, the physical region for simulation is the same for each processor; (i.e., each processor sees the entire spatial grid). Also, all the particles are randomly and equally divided among the processors at the outset of the simulation. This implies that we get an automatic static load balancing: the amount of work done by each CPU is roughly the same even after many iterations.

This contrasts with the Eulerian decomposition scheme in which the physical region of simulation is divided among the processors. Any particles in a physical region are handled by the processor corresponding to that region. In such a scheme, a dynamic load balancing method must also be developed for optimal effect since some spatial partitions may accumulate more particles than others.

In our case, since the field solve is a small fraction of the simulation time, it is done by one processor which broadcasts the results to the other processors.(Or, alternatively, the total electron and ion densities are broadcast among all the processors, and the field solve is done individually by each processor). However, problems in which the field solve is a significant percentage of the workload will greatly benefit from a parallel field solve.

It is often desirable to store the state of a system in a dump file, so that a simulation can be restarted from the dump point rather than the starting point. To dump the state of the system in our parallel run, each CPU sends the positions and velocities of its particles to a root CPU which gathers the information and writes it to a file. When restoring, the root CPU reads the dump file and distributes the particles randomly among the CPUs.

Diagnostics are an essential part of the simulation. It is very desirable to view graphical displays of various diagnostics. In a parallel run, displaying graphics tends to be costly since

information must be gathered from all of the CPUs. One solution is to run the parallel PIC-MCC with the graphics turned off until equilibrium is reached, upon which the state of the system is dumped. Then, restart the run from the dump file using the single processor PIC-MCC with the graphics turned on. Another solution is for root to gather the diagnostics information from each CPU and then display the diagnostics. This is very costly to do at each timestep, so root can be constrained to gather and display a diagnostic only if the diagnostic window is open. In conjunction with this, root can refresh the graphics after many timesteps (e.g., 100) rather than after every timestep in order to further cut down the time spent doing graphics.

4.2 MPI: Message Passing Interface Library

The Message Passing Interface Library (MPI) contains a suite of useful functions for writing parallel codes in both C and Fortran. We used the MPICH implementation developed at Argonne National Lab and Mississipi State University [12]. The text by Pacheco [13] is an excellent introduction to parallel programming with MPI and contains many useful references. The following is a brief description of the MPI routines used in our parallel particle processing code. The routines use either a point to point communication procedure or a tree scheme. That is, a source CPU sends data directly to each of its target CPUs (point to point scheme), or all the CPUs collectively participate in distributing the data (tree scheme). These schemes are displayed in Fig. 4. Note that for a point to point scheme the communication time increases as $N_{proc} - 1$, where N_{proc} is the number of processors, while, for a tree scheme, the time increases as $\log_2 N_{proc}$.

- **MPI_Bcast**: Broadcasts data collectively from a source CPU to many target CPUs (tree scheme).
- **MPI_Reduce**: Sums the data of CPUs together and sends the result to root (tree scheme).
- **MPI_Allreduce**: Sums the data of CPUs together and sends the result to *all* the CPUs (tree scheme).

- **MPI_Gatherv**: Gathers variable size arrays from different CPUs and concatenates them together (tree scheme).
- **MPI_Send & MPI_Receive**: Send and receive data from a source CPU to a target CPU (point to point scheme).

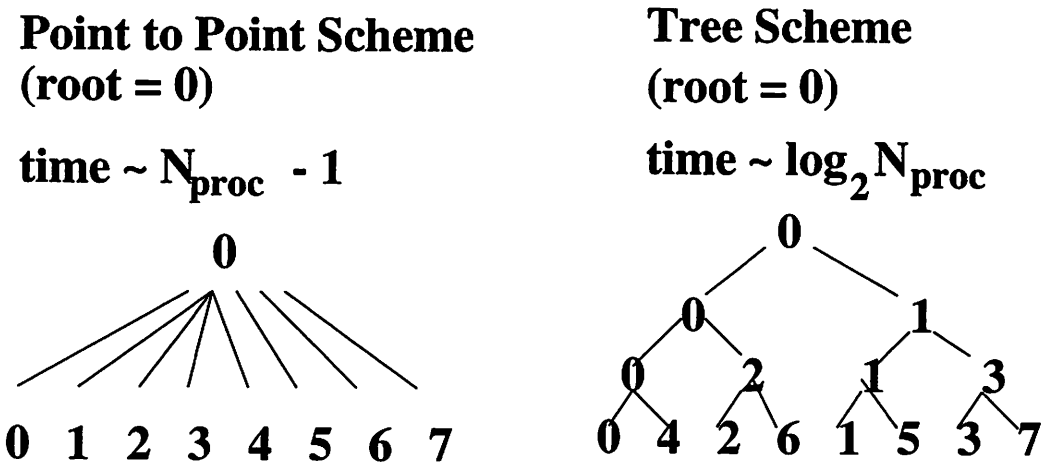

Figure 4: A point to point scheme communication model compared to a tree scheme communication model. From Pacheco [13].

The MPI libraries can be configured for a particular computing environment. For example, the MPICH distribution of MPI comes with libraries configured for shared memory systems such as symmetric multiprocessors (SMPs). Also members of the UC Berkeley Network of Workstations (NOW) project, headed by Prof. David Culler, have written MPI libraries optimized for their distributed network of Sun Ultra 170 workstations[14]. Using the libraries especially configured for the computing environment significantly reduces the communication time among the CPUs.

In our simulation, MPI_Send and MPI_Receive are used only in the initial stage of the simulation, when the particles are divided up among the CPUs. MPI_Gatherv is only used when gathering data to display diagnostics or to store in a dump file.

If we use the first method in step 3 for calculating the electric field E, then MPI_Reduce is used to sum the electron and ion density data of all the CPUs and send the result to root. Then, root calculates E after which MPI_Broadcast is used to send the result to all

the CPUs. If we use the alternative method, then MPI_Allreduce is used to sum the electron and ion density data of all the CPUs and send the result to *all* the CPUs. Then, each CPU individually calculates the field solve to advance its particles. Note that except for the initial stage of the simulation, we primarily use a tree communication scheme.

The size of the electric field array is equal to the number of grid points N_g while the size of the density array is equal to the number of species times the number of grid points $(N_s \times N_g)$. This is because the electric field and the electron and ion densities are calculated at the grid points, and each species has its own density data. The two alternative field solve methods took about the same time in a 1d3v simulation, where the sizes of the arrays passed among the processors are relatively small ($\sim 10^2$ elements). However, for a 2d3v simulation, where the sizes of arrays passed among the processors can be significantly larger ($\sim 10^4$), the second method provided better communication speeds. For the simulations below, we used the first method to do 1d3v simulations while we used the second method for 2d3v simulations.

4.3 Estimating Gains in parallel particle processing

The total time to run the simulation is given by

$$t_{tot} = t_{init} + N_{steps} t_{phys}, \tag{14}$$

where t_{init} is the initial time spent in setting up the simulation, t_{phys} is the time it takes to complete one physics loop as depicted in Fig. 2, and N_{steps} is the number of timesteps. Since $N_{steps} \gg 1$, typically,

$$t_{tot} \approx N_{steps} t_{phys}. \tag{15}$$

The time to execute a physical loop can be divided into three parts:

$$t_{phys} = t_{part} + t_{field} + t_{comm}, \tag{16}$$

where t_{part} is the time to process (e.g., push or accumulate) particles, t_{field} is the time to conduct the field solve, and t_{comm} is the communication time among the processors. t_{part} is proportional to the number of particles N_{part}, while t_{field} is proportional to the number of

grid points N_g in the system. Also, since we use a tree scheme to communicate among the processors, t_{comm} is proportional to $\log_2 N_{proc}$. Then, we can write:

$$t_{phys} = c_1 N_{part}/N_{proc} + c_2 N_g + c_3 \log_2 N_{proc}, \tag{17}$$

where c_1, c_2 and c_3 are constant coefficients that depend on machine architecture and configuration. Note that as the number of processors N_{proc} increases, t_{part} decreases, t_{fields} remains the same, and t_{comm} increases. In order, to obtain any benefit from the parallel particle processing, we require both t_{field} and t_{comm} to be much less than t_{part}. The Gain for using n processors can be estimated by:

$$\text{Gain} = \frac{t_{phys}(N_{proc} = n)}{t_{phys}(N_{proc} = 1)} \tag{18}$$

Note that for a fixed N_g, the larger the number of particles N_{part}, the more the first term in (17) dominates so that the Gain (18) becomes more linear with N_{proc}.

5 Applications of Parallel Particle Processing Method

In this section, we apply the parallel particle processing method described in the previous section on 1d3v and 2d3v RF argon discharge models. For the 1d3v simulations, we used a modified version of PDP1 in which we incorporated the parallel particle processing scheme. For the 2d3v simulations, we used a similarly modified version of our 2d3v electrostatic PIC-MCC code PDP2[15]. We used a 2 CPU Intel Pentium II symmetric multiprocessor (SMP) machine ("the Dual"), a 4 CPU Intel Pentium Pro SMP machine ("the Quad"), and a distributed network of Sun Ultra 170 workstations ("the NOW"). The Dual machine contained two 400 MHz Pentium II CPUs, 512 KB level-2 cache and 512 MB of memory. The Quad machine contained four 200 MHz Pentium Pro CPUs, 256 KB of level-2 cache, and 256 MB of memory. Each workstation on the NOW contained a 167 MHz Ultra1 CPU, 512 KB level-2 cache, and 128 MB of memory. In general, we found that when passing large arrays (as in the 2d3v models) among processors, larger cache and memory sizes on the machines significantly reduced the communication times.

Gains 1d

CPUs	Dual			Quad			NOW		
	156K	78K	39K	156K	78K	39K	156K	78K	39K
1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	2.00	1.98	1.95	1.97	1.98	1.96	1.99	1.92	1.88
3				2.92	2.94	2.79	2.96	2.72	2.61
4				3.95	3.87	3.57	3.80	3.58	3.11
5							4.43	4.11	3.72
6							5.19	4.58	3.77
7							5.83	5.39	4.30
8							6.80	5.56	4.50

Table 4: Gains for 1d3v model with number of particles varying from 39,000 to 156,000 running on the Dual and Quad SMPs, and the NOW.

5.1 Application to 1d3v RF discharges

We applied the parallel particle processing to a 1d3v 100 mT parallel plate argon discharge with a plate separation of 5 cm driven by a sinusoidal voltage source of 1000 V at a frequency of 13.56 MHz. The total number of grid points in the system was 601. The total initial number of particles was varied from 39,000 to 156,000. We took timings after running the simulation for 5000 timesteps. The resulting Gains are listed in Table 4. Figure 5 is a plot of the Gains for the Quad and the NOW. We see that for a fixed number of grid points N_g, the Gain becomes more linear as the number of particles N_{part} in the simulation increases.

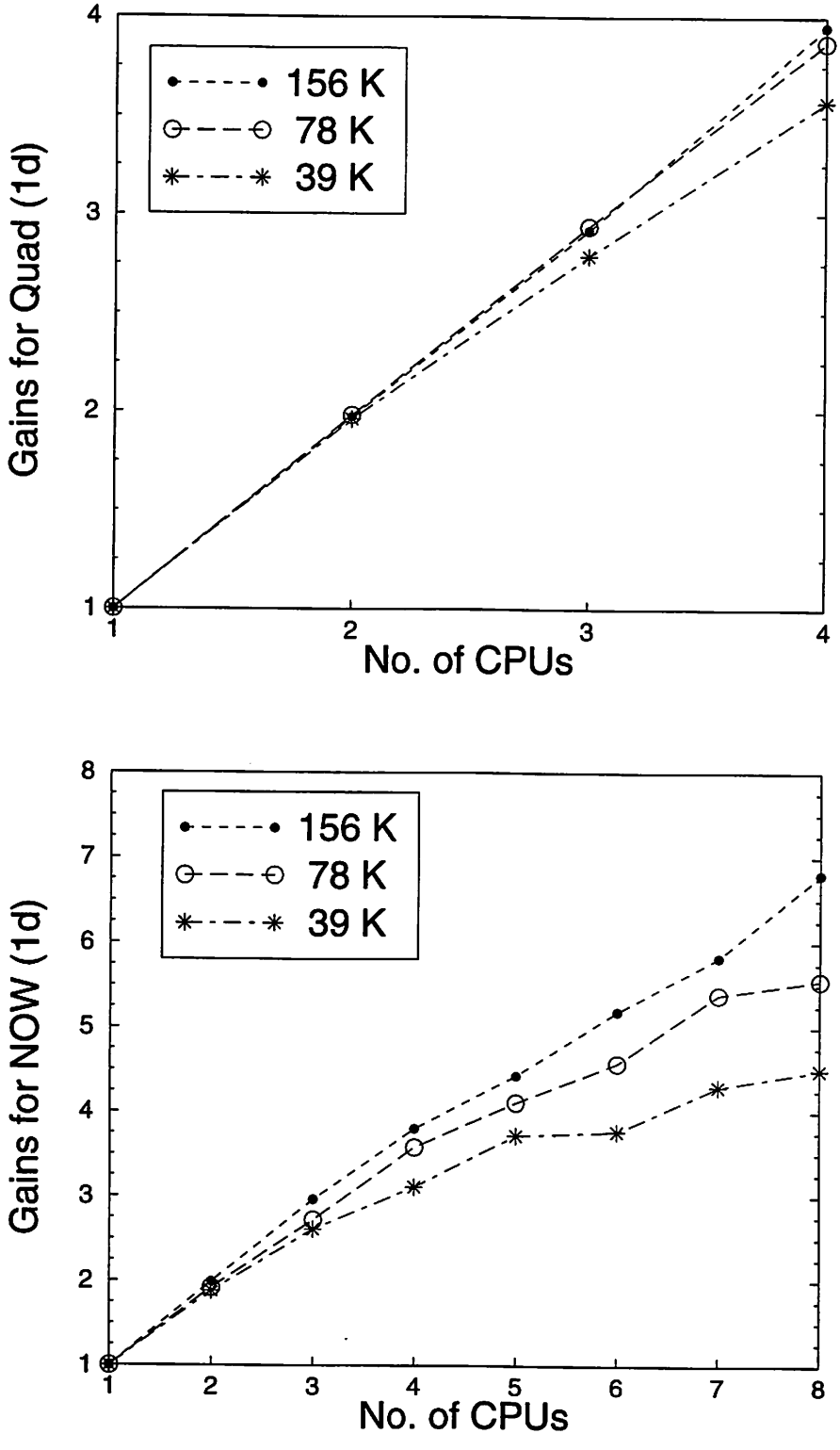

Figure 5: Gains for the 1d3v model for the Quad SMP, and for 8 nodes of the NOW.

Gains 2d

CPUs	Dual			Quad			NOW		
	812K	490K	360K	812K	490K	360K	812K	490K	360K
1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	1.93	1.93	1.92	1.89	1.88	1.88	1.93	2.00	1.88
3				2.64	2.66	2.68	2.78	2.86	2.79
4				3.30	3.37	3.35	3.80	3.88	3.52
5							4.77	4.38	4.31
6							5.22	5.08	4.86
7							6.04	6.00	5.05
8							6.89	6.51	5.82

Table 5: Gains for 2d3v model with number of particles varying from 360,000 to 812,000 running on the Quad and Dual SMP, and the NOW.

5.2 Application to 2d3v RF discharge

2d3v PIC-MCC simulations typically have a much larger number of particles than 1d3v PIC-MCC simulations. But they also usually have a much larger number of grid points and a much more complex field solve than their 1d3v counterparts. Thus, the field solve can be a much larger fraction of the total workload in 2d than in 1d. But by using FFT techniques to solve for the E field, we managed to get the field solve down to less than a few percent of the total workload in our 2d3v PIC-MCC simulations. FFT methods are fast since they are of order $O(N_g \log_2 N_g)$ while other matrix solves are typically of order $O(N_g^2)$. However, FFT field solve methods become very complex and cumbersome if there are internal structures such as conductors and dielectrics inside the computational grid. So, our current 2d3v FFT field solve requires that there be no such internal structures inside the grid. For a discussion of FFT techniques, see Chapter 12 and Section 19.4 in "Numerical Recipes in C" [16].

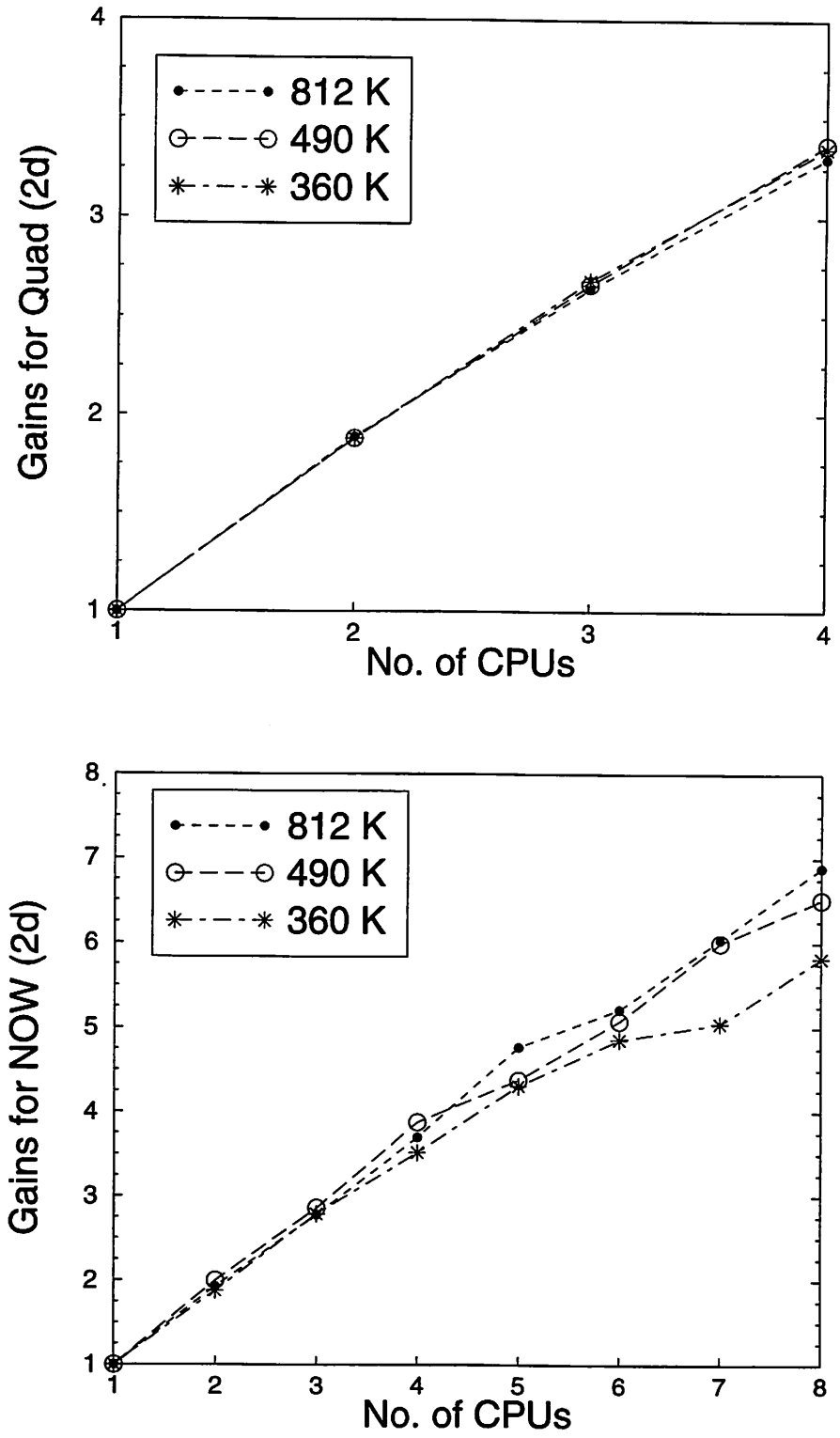

Figure 6: Gains for the 2d3v model for the Quad SMP, and for 8 nodes of the NOW.

We applied the parallel particle processing to a 2d3v 100 mT rectangular argon discharge with a size of 3 cm by 6 cm. The discharge was driven by a sinusoidal voltage source of 250 V at a frequency of 13.56 MHz. The grid size was 65 by 129 (or 8385 grid points). The total initial number of particles was varied from 360,000 to 812,000. We took timings after running the simulation for 500 timesteps. The resulting Gains are listed in Table 5. Figure 6 is a plot of the Gains for the Quad and the NOW. In general, we see that parallel particle processing can yield significant Gains even in 2d3v models, provided that the number of particles N_{part} is much greater than the number of grids N_g, and the communication time does not become too significant. In other words, t_{field} and t_{comm} must be much less than t_{part} in (17).

As expected, as N_{part}/N_g increases, so does the linearity of the Gain on the NOW. However, this is only true to some extent on the Dual and Quad. The Gain does become more linear on the Dual and Quad as the above ratio increases, but only to some extent. Eventually, the Gain saturates because in an SMP machine all the CPUs must share the bus lines which communicate with the memory and cache. When we increased the scale of the problem by increasing N_{part}, these communication lines became saturated. In contrast, for the NOW, each CPU is on a different machine with its own communication bus to its own memory and cache. Thus, when we increased the scale of the problem, the bus lines did not saturate, and the linearity of the Gain increased. This saturation of linearity was only noticeable in the 2d3v models. In the 1d3v models, the scale of the problem is so much smaller that we were not able to saturate the communication bus in the SMP machines.

5.3 Actual vs. estimated Gains

Recall that by using (17) and (18), we can estimate the Gains of the parallel particle processing scheme for our 1d3v and 2d3v models. Let us do this for the Quad SMP machine.

We found the coefficients c_1 and c_2 by profiling our PIC-MCC codes when running our models on one processor of the Quad. (Recall that c_1 is the particle processing coefficient and c_2 is the field solve coefficient.) From this we learned that for the 1d3v code, c_1=1.4

CPUs	1d						2d					
	156K		78K		39K		812K		490K		360K	
	Act.	Est.	Act.	Est.	Act.	Est.	Act.	Est.	Act.	Est.	Act.	Est.
1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	1.97	1.98	1.98	1.97	1.96	1.94	1.89	1.98	1.88	1.97	1.88	1.96
3	2.92	2.95	2.94	2.90	2.79	2.80	2.64	2.95	2.66	2.92	2.68	2.89
4	3.95	3.89	3.87	3.79	3.57	3.58	3.30	3.90	3.37	3.83	3.35	3.77

Table 6: A comparison of actual vs. estimated Gains for the 1d3v and 2d3v models running on the Quad SMP.

μsec and c_2=1.0 μsec. Whereas, for the 2d3v code, c_1=6.9 μsec and c_2=3.6 μsec. The communication coefficient c_3 can be estimated by writing a test program to measure how long it takes to pass the relevant-sized arrays among the processors in the Quad. From this, we learned that for the Quad Pentium-Pro SMP using the shared memory MPI libraries, c_3=0.55 ms for the 1d3v models, and c_3=7.6 ms for the 2d3v models. Then by using (17) and (18), we obtained the estimated Gains for the 1d3v and 2d3v models from using 2, 3, and 4 processors respectively.

The estimated and actual Gains for the Quad are listed in Table 6 for both the 1d3v and 2d3v models. From the Table, we see that the estimated and actual gains match up very well in the 1d3v models but not in the 2d3v models. This is because we are underestimating t_{comm} for the 2d3v models. Recall that the communication bus becomes saturated in SMP machines when the scale of the problem is very large. This effect is not incorporated into $t_{comm} = c_3 \log_2 N_{proc}$ which only measures how long it takes to pass relevant-sized arrays among the processors. We also neglect any synching time among the processors. These effects are small in 1d3v models, where the size of the arrays passed among the processors as well as the general scale of the problem are much smaller than in 2d3v models.

6 Other Developments

Other approaches are also being made to speedup PIC-MCC codes. For example Cartwright et al[17] have developed a hybrid Boltzmann-PIC-MCC scheme to speedup simlations of DC discharges. The bulk of the electrons reach thermodynamic equilibrium with the ions each timestep, using the Boltzmann relation for the electrons with a PIC ion source term. A small number of high energy electrons (e.g., higher than the lowest excitation energy threshold) are treated as PIC electrons. The Boltzmann electron approximation neglects effects faster than ion time scales, decreasing the computation time by about the square root of the mass ratio between the ion and the electron. Thus, it is ideal for investigating long time-scale low frequency phenomena in DC discharges.

Bowers[18] has developed a particle sorting algorithm in order to improve memory bandwidth in a 2d3v PIC simulation. In a typical PIC simulation, for every timestep, we need to (1) Load the particle information, (2) Load the electric field, (3) Load the charge density, (4) Store the particle information, and (5) store the charge density. Typically, about 100 bytes of data movement is needed for every particle. If the particles are located randomly with respect to the computational grid, all the memory accesses described above are likely to be cache misses. This cache "thrashing" problem increases with larger grid size as less of the grid will be able to fit in the cache. However, if the particles are sorted with respect to the grid, then accesses 2,3,5 (and maybe 4) will be likely cache hits. This reduces memory bandwidth and increases performance. The sort can be done while pushing the particles so that no extra passes through the particle list are needed. Once the particle list is sorted, one can further speedup up the simulation by doing the push and accumulate in a single pass rather than in two separate passes. In other words, we need only access the particle list once per timestep. But care must be taken to assure that there are no holes in allocation due to particle loss at walls. This combined push/sort/accumulate scheme has produced speedup on typical 2d3v simulations by 30-50 %. In 1d3v simulations cache thrashing is not a major problem because the entire computational grid can usually fit in the cache.

7 Summary

As stated earlier, a major objective is to make first-principles, self-consistent PIC-MCC codes run fast enough to be useful to plasma processing machine designers and experimentalists, on their time scale. We take this to mean running at least as fast as experimentalists can "cut metal," say, in a matter of hours.

We have shown that PIC-MCC RF discharge simulations can be accelerated by large factors on single processor machines by employing both physical and numerical methods. Note that acceleration schemes and Gains tend to be model dependent. On a modest 233 MHz Dec Alpha machine, we reduced the run time from (a) 20 hours to 3.1 hours (Gain = 6) for the 100 mT argon runs, (b) 14 hours to 45 minutes (Gain = 19) for the 10 mT argon runs, and (c) 3.5 days to 2.8 hours (Gain = 30) for the 100 mT oxygen runs. We also found that an improved density profile can further reduce the run time for the 100 mT argon model by a factor of 4.

We also found that paralleling the particle processing can lead to significant Gains in 1d3v and 2d3v PIC-MCC simulations even when the field solve is not paralleled. We avoided the conventional Eulerian decomposition approach in which the computational grid is divided among the processors. Instead, each processor sees the entire grid, and all the particles are randomly divided among the processors. Our relatively simple scheme also has the advantage of static automatic load balancing. We found that as the ratio of the number of particles to the number of grids increased, our Gains became more linear with processor number. An additional requirement for our scheme is that the field solve is a small percentage of the overall simulation. This is easily realized in 1d3v simulations. In 2d3v simulations, we required fast Poisson solvers such as the ones which employ FFT schemes. The only disadvantage of the FFT schemes is that they do not usually allow for internal dielectrics and conductors within the grid. Finallly, we should add that we often find it necessary to use a large number of particles to reduce the noise level in our diagnostics. In this case, the parallel particle processing scheme is ideal since the larger the number of particles, the better the performance.

8 Acknowledgments

We are especially grateful to DOE Contract DE-FG03-97ER54446 and ONR Contract N00014-97-1-0241. There also has been continued support from LLNL, plus collaborations in the plasma processing area, for most of a decade, for which we are very grateful. We are also grateful to the Millennium Project and NOW Project at the University of California at Berkeley for the use of their computers and to Kevin J. Bowers for informative discussions on field solve techniques in 2d3v PIC-MCC simulations.

A Current increase needed for light ion runs

As mentioned above, we needed to increase the amplitude of the RF current drive in the light ion runs of the 10 mT argon model in order to maintain the discharge and produce density profiles similar to the base runs. The required increase in current can be estimated by using the expression for bulk density derived in Chapter 11 of Lieberman and Lichtenberg's text [19] for a homogeneous discharge model. In this simple model, the ion density is uniform everywhere in the plasma and sheath regions while the electron density is uniform and equal to the ion density in the bulk region but zero in the sheath region.

In this case, the bulk density n is found to be,

$$n = \frac{1}{2}\left[\frac{m_e(\nu_m L_p + 2\bar{v}_e)}{u_B e^2(\mathcal{E}_c + \mathcal{E}_e)}\right]^{1/2} J \tag{19}$$

J is the applied current density, ν_m is the electron momentum transfer collision frequency, $\bar{v}_e = \sqrt{8k_BT_e/(\pi m_e)}$ is the average electron thermal velocity, L_p is the bulk plasma length, $u_B = \sqrt{k_BT_e/M_{ion}}$ is the Bohm velocity, $\mathcal{E}_c$ is the collisional energy loss per electron-ion pair created, and $\mathcal{E}_e$ is the mean kinetic energy lost per electron lost.

We assume $T_e \approx 2$ eV and the sheath width $s \approx 1$ cm. For our 10 mT argon simulations, the electrode spacing $L = 0.05$ m. So, $L_p = L - 2s \approx 0.03$ m.

Let us first calculate n for our 10 mT argon base case with $M_{ion} = M_{real}$. Then, $u_B = (u_B)_{real} = \sqrt{k_BT_e/M_{real}}$ and $\bar{v}_e = 9.5\times10^5$ m/s. $\nu_m = n_{gas}K_{el}$, where K_{el} is the rate constant

for electron-neutral elastic collisions. $K_{el}(T_e = 2eV) \approx 7.1 \times 10^{14}$ m^{-3}/s for an argon discharge. At 10 mT, $n_{gas} = 3.2 \times 10^{20}$ m^{-3}. Thus, $\nu_m L_p = 6.8{\times}10^5$ m/s, and $\nu_m L_p + 2\bar{v}_e = 2.6{\times}10^6$ m/s. So, for the base case,

$$n_{real} = \frac{1}{2}\left[\frac{m_e(2.6 \times 10^6 m/s)}{(u_B)_{real} e^2(\mathcal{E}_c + \mathcal{E}_e)}\right]^{1/2} J_{real} \tag{20}$$

For the light ion case, we have $M_{ion} = M_{light} = M_{real}/100$. Then, $u_B = \sqrt{k_B T_e / M_{light}} = 10(u_B)_{real}$, while $\bar{v}_e = 9.5{\times}10^5$ m/s is unchanged. Recall that we increased the electron-neutral cross sections in our light ion runs by the factor $\sqrt{M_{real}/M_{light}} = 10$. Thus, $(\nu_m)_{light} L_p = 10(\nu_m)_{base} L_p = 6.8{\times}10^6$ m/s, and $\nu_m L_p + 2\bar{v}_e = 8.7{\times}10^6$ m/s. So, for the light ion case,

$$n_{light} = \frac{1}{2}\left[\frac{m_e(4.2 \times 10^6 m/s)}{10(u_B)_{real} e^2(\mathcal{E}_c + \mathcal{E}_e)}\right]^{1/2} J_{light} \tag{21}$$

Dividing (20) by (21), we obtain

$$\frac{n_{real}}{n_{light}} = 1.7\frac{J_{real}}{J_{light}} \tag{22}$$

Thus, in order to get the densities to be similar in the 10 mT argon base and light ion runs, we need to roughly double the current drive in the light ion runs.

References

[1] C. K. Birdsall. Particle-in-Cell Charged-Particle Simulations, Plus Monte Carlo Collisions with Neutral Atoms, PIC-MCC. *IEEE Transactions on Plasma Science*, 19(2):65–85, 1991.

[2] V. Vahedi and M. Surendra. Monte-Carlo Collision Model for Particle-in-Cell method: Application to Argon and Oxygen Discharges. *Comp. Phys. Comm.*, 87:179–198, 1995.

[3] Some of our PIC-MCC codes can be downloaded from our website at http://ptsg.eecs.berkeley.edu.

[4] J. P. Verboncoeur, M. V. Alves, V. Vahedi, and C. K. Birdsall. Simultaneous Potential and Circuit Solution for 1d Bounded Plasma Particle Simulation Codes. *J. Comput. Phys.*, 104:321–328, 1993.

[5] C. K. Birdsall and A. B. Langdon. *Plasma Physics via Computer Simulation.* McGraw-Hill, 1985.

[6] V. Vahedi, G. DiPeso, C. K. Birdsall, M. A. Lieberman, and T. D. Rognlien. Capacitive RF Discharges Modelled by Particle-in-Cell Monte Carlo Simulation. I: Anaylsis of Numerical Techniques. *Plasma Sources Sci. Technol.*, 2:261–272, 1993.

[7] J. U. Brackbill and B. I. Cohen. *Multiple Time Scales.* Academic Press, 1985.

[8] S. E. Parker. *Particle Simulation of Bounded Plasmas with a Wide Range of Space and Time Scales.* PhD thesis, Department of Nuclear Engineering at the University of California at Berkeley, 1990.

[9] I. Kouznetsov. University of California at Berkeley. Private communication.

[10] D. J. Cooperberg, V. Vahedi, and C. K. Birdsall. PIC-MCC with Variable Particle Weights. In *Proc. International Conf. on the Numerical Simulation of Plasmas, Valley Forge, PA*, 1994.

[11] E. Akarsu, K. Dincer, T. Haupt, and G. C. Fox. Non-linear Hybird Boltzmann-PIC Acceleration Scheme. In *Proc. SuperComputing'96 Conf., Pittsburgh, PA*, 1996. The article may be downloaded from the website http://www.supercomp.org/sc96/proceedings.

[12] The MPICH implementation of the MPI libraries can be downloaded from ftp://info.mcs.anl.gov.

[13] P. S. Pacheco. *Parallel Programming with MPI.* Morgan Kaufmann Publishers, Inc., 1997.

[14] D. E. Culler, A. Arpaci-Dusseau, R. Arpaci-Dusseau, B. Chun, S. Lumetta, A. Mainwaring, R. Martin, C. Yoshikawa, and F. Wong. Parallel Computing on the Berkeley NOW. In *Proc. of ISPP'97 (9th Joint International Symposium on Parallel Processing), Kobe, Japan*, 1997. The article may be downloaded from the website http://now.CS.Berkeley.EDU/Papers2.

[15] V. Vahedi, C. K. Birdsall, M. A. Lieberman, T. D. Rognlien, and G. DiPeso. Verification of Frequency Scaling Laws for Capacitive Radio-Frequency Discharges Using Two Dimensional Simulations. *Phys. Fluids B*, 5:2719–2729, 1993.

[16] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C, The Art of Scientific Computing, 2nd Edition.* Cambridge University Press, 1992.

[17] K. L. Cartwright, J. P. Verboncoeur, and C. K. Birdsall. Non-linear Hybird Boltzmann-PIC Acceleration Scheme. In *Proc. International Conf. on the Numerical Simulation of Plasmas, Santa Barbara, CA*, 1998.

[18] K. J. Bowers. University of California at Berkeley. Private communication.

[19] M. A. Lieberman and A. J. Lichtenberg. *Principles of Plasma Discharges and Materials Processing.* John Wiley and Sons, Inc., 1994.