ELSEVIER

Computer Physics Communications 87 (1995) 155–178

Computer Physics Communications

Body-fitted electromagnetic PIC software for use on parallel computers

J.W. Eastwood [1], W. Arter, N.J. Brealey, R.W. Hockney

AEA Technology, Culham Laboratory, Abingdon, Oxfordshire, OX14 3DB, UK

Received 20 September 1994; revised 2 December 1994

Abstract

This paper describes the algorithm and implementation issues for the three-dimensional body-fitted Particle-In-Cell (PIC) software suite 3DPIC for modelling the time evolution of interactions between electromagnetic (EM) fields and flows of relativistic charged particles. 3DPIC is applicable to EM calculations in complex geometries where displacement fields are important, and was developed specifically for microwave device modelling. A description is given of the physical model, the numerical scheme and the software. The software was designed to run efficiently on both serial and distributed-memory parallel computers. The performance achieved by the software on parallel computers demonstrates the potential of this code for large scale time-dependent electromagnetic and electrodynamic calculations.

1. Introduction

The software described in this paper offers powerful new capabilities for electromagnetic PIC [1,2] modelling. It was designed for two-dimensional (2-D) and three-dimensional (3-D) modelling of microwave tubes and microwave transmission where the interaction of electromagnetic waves with charged particle flow is important [4–6], although it can equally well be applied to other time dependent problems normally tackled by finite difference, time domain (FDTD) codes. The software solves the relativistic Maxwell–Vlasov set of equations for the electromagnetic field and particle motion throughout the computational volume; this is appropriate for situations where the characteristic timescales are comparable to the transit time of light across the system. For phenomena on a slower timescale, electrostatic [1] or Darwin models [3] may be more appropriate. Volume filling lossy dielectrics, magnetic media, conducting grids and surfaces, surface physics and coupling to external circuits can all be incorporated into the simulations.

Particle-in-Cell (PIC) simulation methods have been used to solve the coupled Maxwell–Vlasov system of equations for over two decades. Early simulations (e.g. [7]) were restricted to simple geometries, and repeatedly solved Poisson's equation to correct the longitudinal part of the electric field to obtain charge conservation. Later codes, designed specifically for simulations in non-rectangular domains achieved charge conservation without the solution of Poisson's equation by using a particle integration scheme which moved particles along the principal axes of the grid. Mature codes using this approach include the MRC MAGIC (2-D) [8,9] and SOS (3-D) [10] codes, and the SAIC 3-D code ARGUS [11].

MAGIC, SOS and other codes of similar vintage were designed to exploit the large scientific com-

[1] E-mail: jim.eastwood@aeat.co.uk.

SSDI 0010-4655(94)00165-0

puters of the 1970s. More recent programs, such as QUICKSILVER [12] were developed to exploit shared-memory MIMD computers efficiently. Almost all of the mature codes use finite differences and are restricted to meshes which are unions of rectangular bricks. None are specifically designed to use body fitted coordinates and to exploit distributed-memory MIMD computers.

The new simulation suite 3DPIC differs from established PIC codes [8–12] for microwave modelling primarily in that

(i) it uses body-fitted finite elements rather than finite differences on an orthogonal grid,
(ii) the finite element (FE) derivation carries over the simplicity of the Yee FDTD algorithm [13] for the EM calculation to general geometry,
(iii) the general geometry FE derivation gives current assignment schemes which are charge conserving,
(iv) it was designed *ab initio* to use distributed-memory parallel computers [14] efficiently.

The last is achieved by a multiblock decomposition of the computational volume. Each block in the decomposition behaves as an independent PIC calculation, which communicates with neighbouring blocks by exchanging fields and particles at its boundaries; these exchanges are treated as message passing, thus allowing simple and efficient mapping of complex physical configurations onto distributed-memory parallel computers.

Blocks are curvilinear hexahedra, i.e. they are cuboid in some (in general non-orthogonal) coordinate system. The elegance of this method is that with appropriate choices of co- and contra- variant tensor components, the majority of the PIC algorithm for each block takes a coordinate invariant form which differs little from the "virtual particle" scheme [15] on regular Cartesian meshes. All geometrical and material properties are encapsulated in the constitutive equations for the fields and in coordinate transformations in the particle equations of motion.

The next section presents the physical model, followed in Section 3 by an outline of the numerical method. Our FE derivation extends the methods introduced by Lewis [16], in two directions: the employment of more general element shapes and the use of the FE approximation for time dependence in the field action integral [15]. The former allow accurate element fitting to complex shaped devices, and the latter automatically ensures charge conservation. The importance of exact charge conservation is that it eliminates the need to assign charge and solve Poisson's equation to correct the longitudinal electric field. Major issues are the description of objects and compact data storage for the computations; these issues are addressed in the Section 4. Implementing the code on parallel computers and benchmarking its performance are treated in Section 5. Section 6 contains concluding remarks.

2. Physical model

2.1. Basic equations

The mathematical model consists of the Maxwell–Vlasov set; Maxwell's equations for the electromagnetic fields and Vlasov's equation (possibly relativistic) for the dynamics of the charged particles. The charged particles may be electrons, ions, or both, although the discussion given below is for electrons only. The mathematical model of the physical system is completed by adding descriptions of the emission and absorption of radiation and of particles from boundaries.

If the particle distribution function f is represented by a set of sample points (i.e. "superparticles")

$$f(\boldsymbol{x},\boldsymbol{p},t) = \sum_i N_s \delta(\boldsymbol{x} - \boldsymbol{x}_i(t))\delta(\boldsymbol{p} - \boldsymbol{p}_i(t)) \quad (1)$$

where $(\boldsymbol{x}_i, \boldsymbol{p}_i)$; $i\epsilon[1, N_p]$ are the coordinates of the N_p superparticles, each of mass $M = N_s m_0$ and charge $Q = N_s q$, then the Maxwell–Vlasov set may be written in terms of the action integral

$$I = \int dt d\tau \left(\frac{\boldsymbol{D}\cdot\boldsymbol{E} - \boldsymbol{B}\cdot\boldsymbol{H}}{2} - \rho\phi + \boldsymbol{j}\cdot\boldsymbol{A} \right) - \int \sum_i \frac{Mc^2}{\gamma_i} dt \quad (2)$$

where symbols take their usual meanings; $\boldsymbol{A}$ is the magnetic vector potential, ϕ is the electric potential, $\boldsymbol{B} = \nabla \times \boldsymbol{A}$ is magnetic flux, $\boldsymbol{E} = -\nabla\phi - \dot{\boldsymbol{A}}$ is electric field, $\boldsymbol{D}$ is electric displacement, $\boldsymbol{H}$ is magnetic intensity, ρ is charge density, and $\boldsymbol{j}$ is current density. γ is the relativistic gamma, and the integrals are over

time t and space τ. Treating I as a functional of the vector potential $\boldsymbol{A}$, the scalar potential ϕ and particle coordinates $\{\boldsymbol{x}_i\}$ leads to Euler–Lagrange equations which reduce to Maxwell's equations and the relativistic equations of motion [16,17].

2.2. Material properties

General dielectric and magnetic media may be included in the model by adding polarisation, $\boldsymbol{P}$, and magnetisation, $\boldsymbol{M}$, terms to the action integral

$$I = \cdots + \int dt d\tau (\boldsymbol{M} \cdot \boldsymbol{B} + \boldsymbol{P} \cdot \boldsymbol{E}) \tag{3}$$

and substituting for $\boldsymbol{M}$ and $\boldsymbol{P}$ from the resulting evolutionary equation using their constitutive relationships. The non-lossy part gives the dielectric function relating $\boldsymbol{D}$ to $\boldsymbol{E}$, and the lossy parts give the magnetisation and conduction currents, $\boldsymbol{j}_l$:-

$$\boldsymbol{D} = \epsilon \boldsymbol{E}, \qquad \boldsymbol{j}_l = -\nabla \times \beta \dot{\boldsymbol{B}} + \sigma \boldsymbol{E} \tag{4}$$

where β is the inductive loss coefficient and σ is the conductivity.

A small magnetisation current can be used in simulations to control numerical noise, thereby allowing simulations to be performed with fewer simulation superparticles than would otherwise be possible.

2.3. Electromagnetic boundary conditions

2.3.1. Internal

Internal boundary conditions are used to treat periodicity, symmetry and to allow domain decomposition. The latter case arises when the computational domain is subdivided into blocks (cf. Section 3.2). Internal boundary conditions are implemented using "glue-patches" (Section 4.1.6).

2.3.2. External

External boundary conditions specify tangential electric fields or normal magnetic fields. They are typically Dirichlet conditions, and may apply at surfaces completely embedded within the computational domain (e.g. an internal conductor).

The full wave description within a device is linked to a lumped circuit approximation of power supplies and extraction waveguides by coupling elements at the surface of the active computational domain to external circuit elements. The external elements have no physical dimension, but they provide a relationship between surface current, I, and tangential electric fields, E. Thus, a perfect conductor would give $E = 0$, a purely resistive circuit element would give $E = ZI$ and a circuit with resistance and inductance (or capacitance) would give

$$\alpha_c \frac{dE}{dt} + \beta_c E = \gamma_c \frac{dI}{dt} + \delta_c I \tag{5}$$

where circuit coefficients α_c, β_c, γ_c, δ_c are chosen to describe the particular external circuit (see [18]). Similarly, a second order differential equation is used for coupling to an L-R-C circuit and so forth. For steady state circuits, Eq. (5) can be replaced by the complex impedance equation $E = ZI$.

Radiation boundary conditions can be treated using the volume wave absorption method used in MAGIC [8,9]. This involves the addition of a non-physical absorbing medium (giving terms $-\sigma \boldsymbol{E}$ in Ampere's Law and $-\sigma_B \boldsymbol{B}$ in Faraday's Law) at free space boundaries.

2.4. Particle boundary conditions

2.4.1. Internal

Internal particle boundary conditions follow the same classification as field boundary conditions, and like their field counterparts are implemented using glue-patches. These boundary conditions conserve particle number and involve transformation of particle position and/or momentum coordinates.

2.4.2. External

External particle boundary conditions specify either particle absorption or emission; for example beam injection, cathode emission and secondary emission. Electron emission from boundary surfaces is treated using standard Monte-Carlo procedures. Beam injection selects superparticle positions and momenta to fit the incoming beam distribution function. Space charge limited emission at cathodes is modelled by introducing sufficient free surface space charge (in the form of superparticles) to bring the normal electric field at the surface to zero. Secondary electrons at boundary surfaces are generated and emitted in response to the

locations and momenta of incident superparticles to fit the chosen energy and direction distribution.

3. Numerical scheme

3.1. Tensor formulation

This section summarises the definitions and identities in general curvilinear coordinates [19] used in the algorithm derivation and formulation.

Let the reference Cartesian coordinates have components (x^1, x^2, x^3) and unit vectors $(\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2, \hat{\boldsymbol{x}}_3)$, so that a vector, $\boldsymbol{x}$, may be written $\boldsymbol{x} = x^i \hat{\boldsymbol{x}}_i$, where summation over the repeated index i (= 1, 2, 3) is implied. If (x^1, x^2, x^3) are expressed as functions of the curvilinear coordinate components $(\bar{x}^1, \bar{x}^2, \bar{x}^3)$, then $x^1 = x^1(\bar{x}^1, \bar{x}^2, \bar{x}^3)$, etc, or more concisely $\boldsymbol{x} = \boldsymbol{x}(\bar{\boldsymbol{x}})$.

The *basis vectors* $\boldsymbol{e}_i$, and the *reciprocal basis vectors* $\boldsymbol{e}^i$ are defined by

$$\boldsymbol{e}_i = \frac{\partial \boldsymbol{x}}{\partial \bar{x}^i}, \qquad \boldsymbol{e}^i = \nabla \bar{x}^i \tag{6}$$

These vectors are orthogonal; $\boldsymbol{e}_i \cdot \boldsymbol{e}^j = \delta_i^j$ where δ_j^i is the Kronecker delta (= 1 if $i = j$, 0 otherwise). The reciprocal basis vectors can be written in terms of the basis vectors $\boldsymbol{e}^i = \boldsymbol{e}_j \times \boldsymbol{e}_k / J$ and vice-versa $\boldsymbol{e}_i = (\boldsymbol{e}^j \times \boldsymbol{e}^k) J$ where the Jacobian $J = \sqrt{g} = |\boldsymbol{e}_i \cdot \boldsymbol{e}_j \times \boldsymbol{e}_k|$ is the square root of the determinant g of the metric tensor.

The (covariant) metric tensor is defined as $g_{ij} = \boldsymbol{e}_i \cdot \boldsymbol{e}_j$ and its reciprocal tensor, the contravariant metric tensor is $g^{ij} = \boldsymbol{e}^i \cdot \boldsymbol{e}^j$. In Cartesian coordinates, the covariant, contravariant and physical components are identical and the metric tensor reduces to $g_{ij} = \delta_{ij}$. For orthogonal systems, $g_{ij} = 0$ for $i \neq j$. In general g_{ij} is symmetric with six distinct elements, $g_{ij} = g_{ji}$.

3.1.1. Vector identities

A vector $\boldsymbol{A}$ may be expressed in terms of its *contravariant* component A^i, *covariant* components A_i or *physical* components, $A(i)$:-

$$\boldsymbol{A} = A^i \boldsymbol{e}_i = A_i \boldsymbol{e}^i = A(i)\hat{\boldsymbol{e}}_i \tag{7}$$

where

$$\hat{\boldsymbol{e}}_i = \frac{\boldsymbol{e}_i}{|\boldsymbol{e}_i|} \tag{8}$$

The permutation symbols e^{ijk} and e_{ijk} are 1 for cyclic indices, -1 for anticyclic indices, and 0 otherwise, and are related to the permutation tensors by

$$\epsilon^{ijk} = e^{ijk}/\sqrt{g}; \qquad \epsilon_{ijk} = \sqrt{g} e_{ijk} \tag{9}$$

The permutation tensors satisfy the identity

$$\epsilon^{ijk}\epsilon_{klm} = \delta_l^i \delta_m^j - \delta_m^i \delta_l^j \tag{10}$$

Vector dot and cross products become:

$$\begin{aligned} \boldsymbol{a} \cdot \boldsymbol{b} &= a^i b_i = a_i b^i \\ (\boldsymbol{a} \times \boldsymbol{b})^i &= \epsilon^{ijk} a_j b_k \\ (\boldsymbol{a} \times \boldsymbol{b})_i &= \epsilon_{ijk} a^j b^k \end{aligned} \tag{11}$$

The common vector operations are:

$$(\nabla \rho)_i = \frac{\partial \rho}{\partial \bar{x}^i} \tag{12}$$

$$\nabla \cdot \boldsymbol{a} = \frac{1}{\sqrt{g}} \frac{\partial}{\partial \bar{x}^i} \sqrt{g} a^i \tag{13}$$

$$(\nabla \times \boldsymbol{a})^i = \epsilon^{ijk} \frac{\partial a_k}{\partial \bar{x}^j} \tag{14}$$

$$\boldsymbol{a} \cdot \nabla \rho = a^i \frac{\partial}{\partial \bar{x}^i} \rho \tag{15}$$

$$\nabla^2 \rho = \frac{1}{\sqrt{g}} \frac{\partial}{\partial \bar{x}^i} \left(g^{ij} \sqrt{g} \frac{\partial \rho}{\partial \bar{x}^j} \right) \tag{16}$$

3.1.2. Maxwell's equations

Using the formulae of Section 3.1.1 we can write Maxwell's equations in tensor form:-

$$\frac{\partial B^i}{\partial t} = -\epsilon^{ijk} \frac{\partial E_k}{\partial \bar{x}^j} \tag{17}$$

$$\frac{1}{\sqrt{g}} \frac{\partial}{\partial \bar{x}^i} \sqrt{g} B^i = 0 \tag{18}$$

$$\frac{\partial D^i}{\partial t} = \epsilon^{ijk} \frac{\partial H_k}{\partial \bar{x}^j} - j^i \tag{19}$$

$$\frac{1}{\sqrt{g}} \frac{\partial}{\partial \bar{x}^i} \sqrt{g} D^i = \rho \tag{20}$$

It is convenient to introduce extensive current and charge variables

$$I^i = \sqrt{g} j^i; \qquad Q = \sqrt{g} \rho \tag{21}$$

and volume scaled flux quantities b^i and d^i for magnetic and displacement fields

$$b^i = \sqrt{g}B^i; \qquad d^i = \sqrt{g}D^i \tag{22}$$

If in addition we write the permutation tensor in terms of the permutation symbol, Maxwell's equations become

$$\frac{\partial b^i}{\partial t} = -e^{ijk}\frac{\partial E_k}{\partial \bar{x}^j} \tag{23}$$

$$\frac{\partial b^i}{\partial \bar{x}^i} = 0 \tag{24}$$

$$\frac{\partial d^i}{\partial t} = e^{ijk}\frac{\partial H_k}{\partial \bar{x}^j} - I^i \tag{25}$$

$$\frac{\partial d^i}{\partial \bar{x}^i} = Q \tag{26}$$

Eqs. (23)–(26) have the particularly attractive feature that they have the same form irrespective of coordinate system, and contain no explicit reference to the metrics. This is also true of the relationships between the electromagnetic fields and potentials:

$$E_k = -\frac{\partial \phi}{\partial \bar{x}^k} - \dot{A}_k \tag{27}$$

$$b^i = e^{ijk}\frac{\partial A_k}{\partial \bar{x}^j} \tag{28}$$

The only explicit reference to the metrics appears in the constitutive relationships between fields, which in vacuo are

$$\mu_0 H_i = \frac{g_{ij}}{\sqrt{g}}b^j; \qquad \epsilon_0 E_i = \frac{g_{ij}}{\sqrt{g}}d^j \tag{29}$$

More generally, μ_0 and ϵ_0 are replaced by tensor permeabilities and permittivities.

A similar attractive simplicity appears in the Action Integral for the electromagnetic field equations:

$$I = \int dt\, d\bar{x}^1\, d\bar{x}^2\, d\bar{x}^3 \times \left(\frac{1}{2}(E_i d^i - H_i b^i) + I^i A_i - Q\phi\right) \tag{30}$$

This metric-free form simplifies the problem of writing a program module for solving Maxwell's equations in arbitrary non-orthogonal coordinate systems.

3.1.3. Equations of motion

The relativistic equations of motion of a charged particle are

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{v} \tag{31}$$

$$\frac{d\boldsymbol{p}}{dt} = q(\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B}) \tag{32}$$

where

$$\boldsymbol{p} = \gamma m_0 \boldsymbol{v} \tag{33}$$

$$\gamma^2 = 1 + \frac{|\boldsymbol{p}|^2}{(m_0 c)^2} = 1/\left(1 - \frac{|\boldsymbol{v}|^2}{c^2}\right) \tag{34}$$

In general coordinates $(\bar{x}^1, \bar{x}^2, \bar{x}^3)$, these become

$$\frac{d\bar{x}^k}{dt} = \bar{v}^k \tag{35}$$

and

$$\frac{d\bar{p}_s}{dt} - \Gamma^r_{sl}\bar{p}_r\bar{v}^l = q\left[E_s + e_{slr}\bar{v}^l b^r\right] \tag{36}$$

where the Christoffel symbol of the first kind is given by

$$\Gamma^i_{jk} = \boldsymbol{e}^i \cdot \frac{\partial \boldsymbol{e}_k}{\partial \bar{x}^j} \tag{37}$$

The particle dynamics can be included in the action integral, Eq. (30) by adding the extra particle Lagrangian term

$$-\int \frac{m_0 c^2}{\gamma} dt \tag{38}$$

and explicitly expressing the charges and currents in Eq. (30) as sums over particles, i, with charges q_i

$$Q = \sum_i q_i \delta(\bar{x}^1_i - \bar{x}^1)\delta(\bar{x}^2_i - \bar{x}^2)\delta(\bar{x}^3_i - \bar{x}^3) \tag{39}$$

$$I^m = \sum_i q_i \delta(\bar{x}^1_i - \bar{x}^1)\delta(\bar{x}^2_i - \bar{x}^2)\delta(\bar{x}^3_i - \bar{x}^3)\frac{d\bar{x}^m}{dt} \tag{40}$$

The equation of motion Eq. (36) arises from the Euler–Lagrange equation

$$\frac{\partial}{\partial \bar{x}^s}L - \frac{d}{dt}\frac{\partial L}{\partial \dot{\bar{x}}^s} = 0 \tag{41}$$

where L is the total Lagrangian density.

3.2. Element generation

3.2.1. Multiblock decomposition

The elements are generated using a multiblock decomposition; complex objects are divided into a set of curvilinear hexahedral blocks (where each face has four edges), which in turn are subdivided into hexahedral finite elements using transfinite interpolation. Metric tensors and basis vectors are defined on each element by using coordinate transformations isoparametric to the electric potential representation.

The user of the software controls the multiblock decomposition by describing objects in terms of a "kit-of-parts" (Section 4.2). In selecting the parts for the multiblock division, complex objects are divided into sufficiently convex volumes to avoid the creation of very small, or even negative, volume elements in the blending process; this could cause the numerical simulations to fail. However in the applications envisaged it is unlikely that singular elements will present serious difficulties. Computational efficiency dictates that wherever possible, orthogonal blocks (such as rectangular bricks and cylinder sections) be used as this greatly reduces computational costs and storage. To facilitate load balancing on MIMD computers, it may also be advantageous to subdivide blocks beyond the level dictated by the object's geometry.

Each block of the multiblock decomposition is described by its bounding curves; four intersecting curves in two dimensions and twelve curves in three dimensions. Alternatively, the 3-D block may be described by its bounding surfaces. The relationship of the block coordinates to the global coordinates of the object is described in Section 3.5. The locations of element nodes within a block are generated by blending the interpolants from the bounding curves or surfaces, as described in Sections 3.2.3.

In order to transform between curved space (barred variables) and physical coordinates, and to integrate the equations of motion, values of the basis vectors are required. The integration of the field equations and transformation between covariant and contravariant components requires metric tensor values. These may be evaluated from the finite element approximations to the coordinate transformations, as outlined below.

3.2.2. Isoparametric elements

An isoparametric element uses the same function for the coordinate mapping as for the finite element support of, in this case, the scalar potential ϕ. Fig. 1 shows an isoparametric quadrilateral element in physical and barred coordinate space. A point $\boldsymbol{x}$ is related to the barred coordinates $\bar{\boldsymbol{x}} = (\bar{x}^1, \bar{x}^2)$ by

$$\boldsymbol{x} = \boldsymbol{x}_1(1-\bar{x}^1)(1-\bar{x}^2) + \boldsymbol{x}_2\bar{x}^1(1-\bar{x}^2) + \boldsymbol{x}_3\bar{x}^1\bar{x}^2 + \boldsymbol{x}_4(1-\bar{x}^1)\bar{x}^2 \tag{42}$$

This gives a basis vector

$$\boldsymbol{e}_1 = \frac{\partial \boldsymbol{x}}{\partial \bar{x}^1} = (\boldsymbol{x}_2 - \boldsymbol{x}_1)(1-\bar{x}^2) + (\boldsymbol{x}_3 - \boldsymbol{x}_4)\bar{x}^2 = \boldsymbol{r}_S(1-\bar{x}^2) + \boldsymbol{r}_N\bar{x}^2 \tag{43}$$

and similarly for $\boldsymbol{e}_2$, from which the reciprocal basis vectors, metric tensor elements and Jacobian can be readily computed at any point within the element.

Basis vectors and metrics are defined in the same manner for 3-D hexahedral elements. If we label the nodes of the element by the index triplet (i, j, k), $i, j, k = 0$ or 1, and let

$$w_0(z) = (1-z), \qquad w_1(z) = z \tag{44}$$

then the analogue of Eq. (42) is

$$\boldsymbol{x} = \boldsymbol{x}_{ijk} w_i(\bar{x}^1) w_j(\bar{x}^2) w_k(\bar{x}^3) \tag{45}$$

and the basis vectors are

$$\boldsymbol{e}_1 = (\boldsymbol{x}_{1jk} - \boldsymbol{x}_{0jk}) w_j(\bar{x}^2) w_k(\bar{x}^3) = \boldsymbol{r}_{\frac{1}{2}jk} w_j(\bar{x}^2) w_k(\bar{x}^3) \tag{46}$$

$$\boldsymbol{e}_2 = \boldsymbol{r}_{i\frac{1}{2}k} w_i(\bar{x}^1) w_k(\bar{x}^3) \tag{47}$$

$$\boldsymbol{e}_3 = \boldsymbol{r}_{ij\frac{1}{2}} w_i(\bar{x}^1) w_j(\bar{x}^2) \tag{48}$$

The reciprocal basis vectors and metric tensor elements follow by substituting $\boldsymbol{e}_i$ from Eqs. (45)–(48) into the appropriate formulae from Section 3.1.

3.2.3. Transfinite interpolation

Mesh generation for finite volume and finite element schemes, by blending the interpolants from intersecting pairs of curves such that the mesh exactly fits the boundary curves, was introduced by Gordon and Hall [20] and is now widely used in body-fitted fluid flow

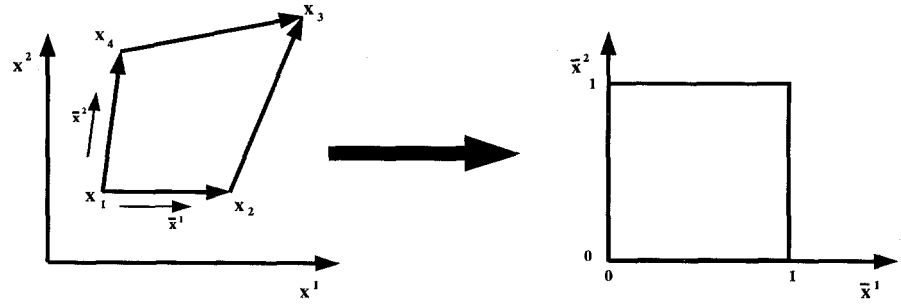

Fig. 1. The isoparametric linear quadrilateral element with corner nodes $x_1 \ldots x_4$ maps to the unit square in the barred coordinate space.

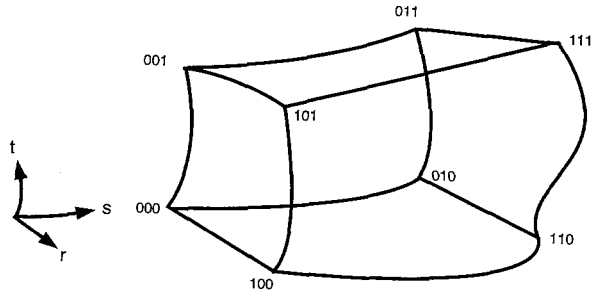

Fig. 2. The six faced volume is defined by the twelve bounding curves joining the corner nodes at positions x_{ijk}, $i, j, k = 0$ or 1

codes, such as Harwell-FLOW3D [21,22]. We summarise here the transfinite interpolation formulae that we use in our body-fitting software.

The aim is to divide the general curvilinear hexahedra as illustrated in Fig. 2 into a set of hexahedral elements which map to unit cubes in curved space. The bounding curve $X_{ij}(r)$, $r \epsilon [0,1]$ joins the corner node x_{0ij} at $r = 0$ to node x_{1ij} at $r = 1$. Similarly $Y_{ij}(s)$ joins x_{i0j} to x_{i1j} and $Z_{ij}(t)$ joins x_{ij0} to x_{ij1}, where $i, j = 0$ or 1. The interpolation functions for the r, s, and t coordinates are respectively λ, ψ and η.

Blending interpolants between the bounding curves to obtain exact fits at all eight edges gives the formula

$$\begin{aligned} x(r,s,t) = {} & X_{jk}(r)\psi_j(s)\eta_k(t) \\ & + Y_{ik}(s)\lambda_i(r)\eta_k(t) \\ & + Z_{ij}(t)\lambda_i(r)\psi_j(s) \\ & - 2x_{ijk}\lambda_i(r)\psi_j(s)\eta_k(t) \end{aligned} \tag{49}$$

where sums over $i, j, k = 0$ and 1 are implied. Generally, the functions λ, ψ and η are chosen to be linear.

3.3. Maxwell's equations

The discrete finite element approximations to Maxwell's equations are obtained, following Refs. [4,15], using virtual particle electromagnetic particle-mesh schemes for general 3-D coordinate systems. The present implementation uses the lowest order conforming elements, and employs lumping to maintain explicit time integration. If more accurate wave dispersion is required, then the lumping approximations could be replaced by "mass" matrix inversions.

Discrete equations are derived by introducing finite element approximations to the potentials

$$\phi = \Phi U \tag{50}$$

$$A_i = \mathsf{A}_{(i)} W_{(i)} \tag{51}$$

where Φ and $\mathsf{A}_{(i)}$ are nodal amplitudes. Eq. (50) is shorthand (Eq. (51) similarly) for

$$\begin{aligned} & \phi(\bar{x}^1, \bar{x}^2, \bar{x}^3, t) \\ & \quad = \sum \Phi(\boldsymbol{k}, n) U(\boldsymbol{k}, n; \bar{x}^1, \bar{x}^2, \bar{x}^3, t) \end{aligned} \tag{52}$$

where the sum is over spatial ($\boldsymbol{k}$) and temporal (n) node indices. The approximations (50) and (51) are substituted into the Action Integral Eq. (30), which is then varied with respect to the nodal amplitudes Φ and A to yield discrete approximations to the inhomogeneous Maxwell's Eqs. (25) and (26). (The homogeneous Maxwell Equations (23) and (24) follow by virtue of Eqs. (27) and (28), if U and W_i are appropriately chosen.)

3.3.1. Finite element field approximations

The lowest order conforming choice of element support for the potentials is a mixture of piecewise linear and piecewise constant functions. Eqs. (27) and (28) are satisfied in a strong sense for the lowest order conforming elements provided that the finite element approximations to the fields are

$$d^i = \mathsf{d}^{(i)} V_{(i)} \tag{53}$$

$$E_i = \mathsf{E}_{(i)} V_{(i)} \tag{54}$$

$$b^i = \mathsf{b}^{(i)} X_{(i)} \tag{55}$$

$$H = \mathsf{H}_{(i)} X_{(i)} \tag{56}$$

where

$$W_i(\bar{x}^1, \bar{x}^2, \bar{x}^3, t) = \Pi(\bar{x}^i)\Lambda(\bar{x}^j)\Lambda(\bar{x}^k)\Lambda(t) \tag{57}$$

$$U(\bar{x}^1, \bar{x}^2, \bar{x}^3, t) = \Lambda(\bar{x}^1)\Lambda(\bar{x}^2)\Lambda(\bar{x}^3)\Pi(t) \tag{58}$$

$$V_i(\bar{x}^1, \bar{x}^2, \bar{x}^3, t) = \Pi(\bar{x}^i)\Lambda(\bar{x}^j)\Lambda(\bar{x}^k)\Pi(t) \tag{59}$$

$$X_i(\bar{x}^1, \bar{x}^2, \bar{x}^3, t) = \Lambda(\bar{x}^i)\Pi(\bar{x}^j)\Pi(\bar{x}^k)\Lambda(t) \tag{60}$$

$$\Lambda(\bar{x}^i) = \begin{cases} 1 - |\bar{x}^i| \ ; & |\bar{x}^i| < 1 \\ 0 \quad\quad ; & \text{otherwise} \end{cases} \tag{61}$$

$$\Pi(\bar{x}^i) = \begin{cases} 1 \ ; & -1/2 < \bar{x}^i \le 1/2 \\ 0 \ ; & \text{otherwise} \end{cases} \tag{62}$$

and (i, j, k) are cyclic permutations of indices (1, 2, 3). Fig. 3 shows the location of the nodes at which amplitudes of these finite element field approximations are defined. The following two subsections discuss how this choice of FE approximation yields field equations of a similar form to the familiar staggered leapfrog finite difference equations [13] and charge assignment similar to the standard CIC scheme [1,2]. Higher order schemes could be generated by using higher order FE support functions, but are not considered further here.

3.3.2. Homogeneous equations

The homogeneous equations follow immediately from the definitions of the finite element fields and Eqs. (23), (24), (27) and (28). Magnetic flux is identically conserved and Faraday's law

$$\frac{\partial b^i}{\partial t} = -e^{ijk}\frac{\partial E_k}{\partial \bar{x}^j} \tag{63}$$

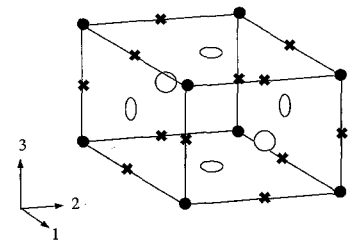

Fig. 3. The location of nodes on the unit cube element. Crosses give A_i, E_i, I^i and d^i node locations, open circles give H_i and b^i locations and solid circles give element corner and Φ node locations.

gives

$$\mathsf{b}^i \frac{\partial X_{(i)}}{\partial t} = -e^{ijk}\mathsf{E}_k \frac{\partial V_{(k)}}{\partial \bar{x}^j} \tag{64}$$

Using the above definitions of X_i and V_i shows that Eq (64) reduces to the standard Yee [13] finite difference form of Faraday's law, but for nodal amplitudes of field components b^i and E_k.

3.3.3. Inhomogeneous equations

Ampere's equation is obtained by assembling contributions to the variation of the Action from adjacent elements

$$\frac{\delta I}{\delta \mathsf{A}_i} = \int dt\, d\bar{x}^1\, d\bar{x}^2\, d\bar{x}^3 \left\{ -\mathsf{d}^i \frac{\partial W_{(i)}}{\partial t} V_{(i)} - \mathsf{H}_j e^{jm(i)} \frac{\partial W_{(i)}}{\partial \bar{x}^m} X_{(j)} + I^{(i)} W_{(i)} \right\} = 0 \tag{65}$$

The current term, which gives the prescription for assigning current from particles to the elements, is discussed further in Section 3.4.1.

For each node internal to a block, there are eight contributions (from four adjacent elements at two timelevels) to the assembled equation resulting from Eq. (65) for a displacement field component. At block surfaces, this number may be different. In the implementation of the software, element contributions are pre-assembled for nodes internal to blocks, but contributions across block faces are assembled by "glue-patch" data exchanges (cf. Section 4.1.6) only during the computation of the displacement fields. For surface nodes, there are contributions from elements within the computational volume, plus possibly

surface current terms from the external boundary conditions.

Variation of I with respect to the potential nodal amplitudes gives the discrete approximations to Gauss' Law:

$$\frac{\delta I}{\delta \Phi} = \int dt\, d\bar{x}^1\, d\bar{x}^2\, d\bar{x}^3 \left\{ -\mathrm{d}^i \frac{\partial U}{\partial \bar{x}^i} V_{(i)} - qU \right\} = 0 \tag{66}$$

The last term gives the prescription for charge assignment, and the remaining part of the integral gives contributions to the divergence of the displacement field. As for Ampere's equations, the number of contributions to a node at a block surface may differ from that for nodes interior to blocks, and boundary conditions are applied via surface patches.

For simplicity, the present implementation of the software uses the lumped approximations $\langle V_{(i)}, V_{(i)} \rangle = 1$, $\langle X_{(i)}, X_{(i)} \rangle = 1$, where the inner product notation

$$\langle a, b \rangle = \int_D dt\, d\bar{x}^1\, d\bar{x}^2\, d\bar{x}^3 a(\bar{x}^i, t) b(\bar{x}^i, t) \tag{67}$$

is used for the integral over the domain of interest D. The advantage of lumping is that it greatly reduces the amount of computational work per node per timestep, but at the cost of degrading dispersive properties and introducing the possibility of numerical Cerenkov instability [23]. With lumping, the assembled equation resulting from Eq. (65) becomes

$$\partial_t \mathrm{d}^i = e^{ijk} \partial_j \mathrm{H}_k - \mathrm{I}^i \tag{68}$$

where the operators ∂ denote centred difference. Lumping in time has the advantage of making timestepping explicit.

3.3.4. Constitutive relations

For isotropic permeability and permittivity tensors the finite element approximations to the constitutive relations are

$$\langle \mathrm{E}_i V_{(i)}, V_{(i)} \rangle = \langle \frac{1}{\epsilon \sqrt{g}} g_{ij} \mathrm{d}^j V_{(j)}, V_{(i)} \rangle \tag{69}$$

and

$$\langle \mathrm{H}_i X_{(i)}, X_{(i)} \rangle = \langle \frac{1}{\mu \sqrt{g}} g_{ij} \mathrm{b}^j X_{(j)}, X_{(i)} \rangle \tag{70}$$

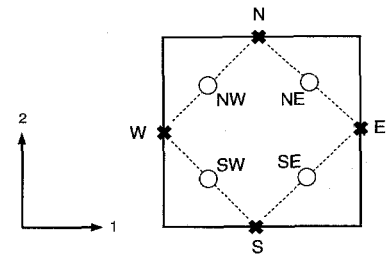

Fig. 4. Values of G_{12} at the NW, NE, SW and SE nodes couple values of d^1, E_1, and the N and S nodes to d^2, E_2 at the E and W nodes.

Using lumped approximations these reduce to

$$\mathrm{E}_i = \mathrm{G}^E_{ij} \mathrm{d}^j \tag{71}$$

$$\mathrm{H}_i = \mathrm{G}^H_{ij} \mathrm{b}^j \tag{72}$$

Elements of the symmetric tensors G^E_{ij} and G^H_{ij} are sparse matrices. More general permittivities and permeabilities are handled by replacing ϵ and μ by tensor functions in the computation of G^E_{ij} and G^H_{ij}, respectively.

The constitutive relations (71) and (72) resulting after assembly at an internal node generally have four off-diagonal terms for each element. Fig. 4 illustrates the location of the nodes. For example, contributions from the element to E_1 at the W node are

$$\delta \mathrm{E}_1 = \mathrm{G}^E_{11}(W) \mathrm{d}^1(W) + \mathrm{G}^E_{12}(NW) \mathrm{d}^2(N) + \mathrm{G}^E_{12}(SW) \mathrm{d}^2(S) \tag{73}$$

and to H_2 at the W node

$$\delta \mathrm{H}_2 = \mathrm{G}^H_{22}(W) \mathrm{b}^2(W) + \mathrm{G}^H_{12}(NW) \mathrm{b}^1(N) + \mathrm{G}^H_{12}(SW) \mathrm{b}^1(S) \tag{74}$$

Similar expressions hold for other components at the other nodes. Adding contributions of all elements to a given node implies a four point sum for the off-diagonal terms.

3.3.5. Metric computations

The metric terms are defined by (no implied summation)

$$\mathrm{G}^E_{ij} = \langle \frac{g_{ij}}{\epsilon \sqrt{g}} V_j, V_i \rangle \tag{75}$$

and

$$\mathsf{G}^H_{ij} = \langle \frac{g_{ij}}{\mu\sqrt{g}} X_j, X_i \rangle \tag{76}$$

The integrals in Eqs. (75) and (76) are evaluated element-by-element in the preprocessor (Section 4.2.1) using Gaussian quadrature. This metric computation assembles the terms within the interior of a uniblock, but leaves the surface coefficients unassembled to facilitate patch exchange.

3.3.6. Electromagnetic boundary conditions

The internal electromagnetic boundary conditions outlined in Section 2.3 are dealt with by glue-patch exchange (Section 4.1.6), and the external boundary conditions described in this section appear as extra terms in the evolutionary equations for fields at surface nodes.

For a surface of a block arranged so that its normal is parallel to $\boldsymbol{e}^3$, the vector equivalent of Ohm's Law $ZI = E$ is

$$\boldsymbol{e}^3 \times (Z\boldsymbol{I}_s - \boldsymbol{E}) = \boldsymbol{0} \tag{77}$$

where the surface current $\boldsymbol{I}_s$ has $I^3_s = 0$. Now for an arbitrary vector $\boldsymbol{f}$, straightforward manipulation gives

$$\begin{aligned}\boldsymbol{e}^3 \times \boldsymbol{f} = \sqrt{g}\{&\boldsymbol{e}^1(-g^{33}f^2 + g^{23}f^3)\\ &+\boldsymbol{e}^2(g^{33}f^1 - g^{13}f^3)\\ &+\boldsymbol{e}^3(-g^{23}f^1 + g^{13}f^2)\}\end{aligned} \tag{78}$$

Taking

$$\boldsymbol{f} = Z\boldsymbol{I}_s - \boldsymbol{E}$$

and noting that

$$f^3 = -E^3 = \frac{-d^3}{\epsilon\sqrt{g}} \tag{79}$$

it follows, from equating the components of Eq. (78) separately to zero that

$$I^i_s = \frac{1}{\epsilon Z\sqrt{g}}\left(d^i - \frac{g^{i3}d^3}{g^{33}}\right), \quad i = 1,2 \tag{80}$$

Geometrical considerations lead to the result that the volume current is given by

$$\boldsymbol{I} = \boldsymbol{I}_s \mid \boldsymbol{e}_1 \times \boldsymbol{e}_2 \mid \tag{81}$$

Algorithmically, the first term on the right-hand side of Eq. (80) presents few problems. The second vanishes if the 3-coordinate is normal to the surface in the sense that $g_{13} = g_{23} = 0$, since $g^{ij}g_{j3} = 0$ for $i = 1, 2$. Alternatively, if the coordinates within the surface are orthogonal so that $g_{12} = 0$, then the relations

$$\frac{g^{i3}}{g^{33}} = -\frac{g_{i3}}{g_{(i)(i)}}, \quad i = 1,2 \tag{82}$$

apply and $g^{i3}d^3/g^{33}$ can be replaced by

$$S_i = -\frac{\mathsf{G}^E_{i3}\mathsf{d}^3}{\mathsf{G}^E_{(i)(i)}} \tag{83}$$

The d^i on a resistive wall patch are thus updated using

$$\mathsf{d}^{i(n+1)} = \alpha\mathsf{d}^{i(n)} + \beta\dot{\mathsf{d}}^{i(n)} + \gamma_i, \quad i = 1,2 \tag{84}$$

where n and $n+1$ denote time levels. Representing d^i in Eq. (80) as $\theta\mathsf{d}^{i(n+1)} + (1-\theta)\mathsf{d}^{i(n)}$ it follows that

$$\alpha = \frac{\tilde{Z} + 2\theta - 2}{\tilde{Z} + 2\theta} \tag{85}$$

$$\beta = \frac{2\tilde{Z}}{\tilde{Z} + 2\theta} \tag{86}$$

and

$$\gamma_i = \frac{2S_i}{\tilde{Z} + 2\theta} \tag{87}$$

where

$$\tilde{Z} = \frac{\epsilon Z}{\Delta t\sqrt{g^{33}}} \tag{88}$$

Note that provided $\theta > 1/2$ the above can be used to represent a perfectly conducting wall. Boundary conditions involving the specification of normal magnetic field and tangential electric field are straightforwardly implemented; the latter are in many cases equivalent to specifying a potential difference across a block.

3.3.7. Timestep loop

The timestep loop for the integration of the field equations deals exclusively with field components. Input from the particle integration are the contravariant currents I^i, and the output to the particle integration are field components E_i and b^i. Details of the particle integration and boundary condition will be discussed later. The finite element field equations are assembled

within each block, and glue-patch data exchange is used to complete the assembly at block surfaces. The electromagnetic field integration routines advance the primary field arrays - the set of nodal amplitudes of d^i and b^i - by the electromagnetic integration timestep ($\leq$ particle timestep). This leads to the following operations in the timestep loop, starting from currents I^i and fields b^i at time level $n + 1/2$ and displacement fields at time level n:

(i) obtain H_i from b^i in each block (Eq. (72)).
(ii) compute the contributions $\dot{\mathsf{d}}^i$ to the displacement current in each block, using the right hand side of Eq. (68) for internal nodes, and its partially assembled equivalent for surface nodes.
(iii) assemble $\dot{\mathsf{d}}^i$ contributions at block surfaces by glue-patch data exchange. This step applies the "internal" boundary conditions (of periodicity, symmetry, neumann type and domain decomposition).
(iv) update d^i and apply external boundary conditions. At internal boundaries, the new d^i is computed from $\partial_t \mathsf{d}^i = \dot{\mathsf{d}}^i$. The treatment of external nodes is described in Section 3.3.6.
(v) obtain E_i from d^i in each block (Eq. (71)).
(vi) assemble E_i contributions across internal boundaries by glue-patch data exchange.
(vii) apply external boundary conditions to E_i
(viii) advance b^i in each block (Eq. (64)).

At the end of the electromagnetic timestep, nodal values d^i are known at time level $n + 1$ and b^i are known at time level $n + 3/2$.

3.4. Particle equations

The electromagnetic field equations take a particularly simple form when the appropriate choices of co- and contra- variant fields are made; the same is true for current assignment, force interpolation and the integration of the particle equations of motion. The only substantial difference between the algorithm in general geometry and in Cartesian coordinates arises in the update of particle positions.

3.4.1. Assignment

The approximation of representing the distribution function by a set of "superparticle" sample points reduces the velocity space integrals for charge and current density to sums over particles (Eqs. (39) and (40)). Substituting these sums into the source term integrals arising from the variations of the field Lagrangian with respect to Φ and A_i give respectively the prescriptions for charge assignment

$$\mathsf{Q} = \int dt \sum_p q_p U(\bar{x}_p^1, \bar{x}_p^2, \bar{x}_p^3, t) \tag{89}$$

and for current assignment.

$$\mathsf{I}^i = \int dt \sum_p q_p \dot{\bar{x}}_p^{(i)} W_{(i)}(\bar{x}_p^1, \bar{x}_p^2, \bar{x}_p^3, t) \tag{90}$$

The integrals for Q and I^i are evaluated in exactly the same manner as described in Ref. [15]. Motion is assumed to be impulsive and trajectory segments are assumed to be straight lines in the curved $(\bar{x}^1, \bar{x}^2, \bar{x}^3)$ space in which elements are unit cubes. The sums in Eqs. (89) and (90) are over particles p. Since the expressions are linear, there is no loss of generality in considering a single particle of charge q.

Charge assignment. If we label nodes on the unit cube element in curved space (i, j, k) as shown in Fig. 5 and write the assignment function in terms of its product parts

$$U_{ijk}(\bar{x}^1, \bar{x}^2, \bar{x}^3, t) = w_i(\bar{x}^1) w_j(\bar{x}^2) w_k(\bar{x}^3) \tag{91}$$

the charge assigned to element node (i, j, k) where $i, j, k = 0$ or 1 becomes

$$\mathsf{Q}_{ijk} = q w_i(\bar{x}^1) w_j(\bar{x}^2) w_k(\bar{x}^3) \tag{92}$$

where $w_0(x) = 1 - x$, $w_1(x) = x$ and $(\bar{x}^1, \bar{x}^2, \bar{x}^3)$ is the displacement of the particle position from the element corner $(0, 0, 0)$. Inspection of Eq. (92) reveals the assignment scheme to be the same as area weighting [1,2], but now applied on a uniform net in curved space rather than in Cartesian, physical space. Locating the element containing the particle is trivial, since the element index is given by the integer truncation of the particle coordinates $\bar{x}^i$, exactly as for simple Cartesian PIC.

Current assignment. Using the impulse approximation to particle motion as described in [15] and the product form of W reduces Eq. (90) for the contribution of a single particle to

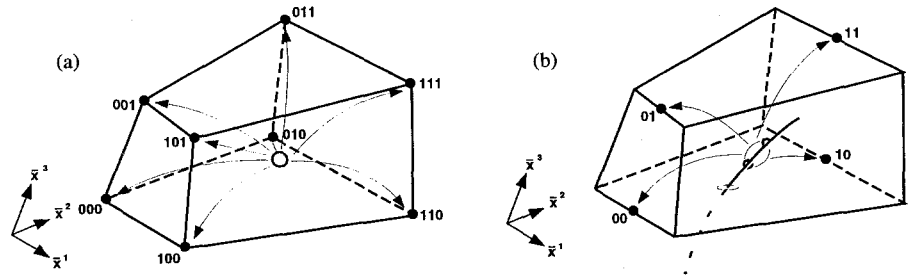

Fig. 5. (a) Charge assignment from a single particle and (b) I^1 current assignment from a single particle trajectory segment for linear potential test functions in physical space.

$$I^1_{\frac{1}{2}jk} = \int_{-1/2}^{1/2} dsqs\Delta\bar{x}^1 w_j(\bar{X}^2 + s\Delta\bar{x}^2) w_k(\bar{X}^3 + s\Delta\bar{x}^3) \tag{93}$$

The quadratic integral is evaluated exactly by point Gaussian quadrature, so Eq. (93) can be written as the contribution from "virtual particles" at $\boldsymbol{x}_\pm$:-

$$I^1_{\frac{1}{2}jk} = q\frac{\Delta\bar{x}^1}{2}\left[w_j(\bar{x}^2_+)w_k(\bar{x}^3_+) + w_j(\bar{x}^2_-)w_k(\bar{x}^3_-)\right] \tag{94}$$

where indices j and k take values 0 or 1. The positions of the virtual particle have coordinates

$$\bar{x}^k_\pm = \bar{X}^k \pm \frac{\Delta\bar{x}^k}{2\sqrt{3}} \tag{95}$$

where $\bar{X}^k = (\bar{x}^k_f + \bar{x}^k_i)/2$, $\Delta\bar{x}^k = \bar{x}^k_f - \bar{x}^k_i$, and $k = 1, 2$ or 3. Coordinates $(\bar{x}^k_i, \bar{x}^k_i)$ and $(\bar{x}^k_f, \bar{x}^k_f)$ are respectively the start (i) and end (f) of the trajectory segment. The corresponding expressions for current components $I^2_{k\frac{1}{2}j}$ and $I^3_{jk\frac{1}{2}}$ are given by simultaneously cyclically permuting component indices (123) and node indices $(\frac{1}{2}jk)$ in Eq. (94). Fig. 5 gives a Cartesian space sketch of assignment according according to (a) Eq. (92) for charge and (b) Eq. (94) for current.

In practice, it more convenient to evaluate explicitly the integrals for I^i rather than use quadrature. Eq. (94) then becomes

$$I^1_{\frac{1}{2}jk} = q\Delta\bar{x}^1\left[w_j(\bar{X}^2)w_k(\bar{X}^3) + (j-\frac{1}{2})(k-\frac{1}{2})\frac{\Delta\bar{x}^2\Delta\bar{x}^3}{3}\right] \tag{96}$$

and similarly for I^2 and I^3 by cyclic permutation of indices.

3.4.2. Charge conservation

By linear superposition, conservation for one trajectory moving through a single time step implies the same for the sum of all trajectories. Summing all contributions to Eqs. (93) from a single particle, gives the same as the difference of Eqs. (92) at start and end points, i.e. the linear hexahedral element case satisfies

$$\partial_t Q = -\partial_k I^k \tag{97}$$

where the symbol ∂ denotes a centred difference arising from assembling the finite element contributions, and Q and I^k are nodal amplitudes. Equations of the form (97) can be shown to be generally satisfied for virtual particle algorithms.

The great benefit of Eq. (97) is computational speed-up; charge assignment and solution of Poisson's equation for the electrostatic potential correction are not needed in the timestep loop.

3.4.3. Force interpolation

Fields at particle positions are given by the finite element fields (Eqs. (53)–(56)) sampled at the particle positions, where the interpolation functions are given by Eqs. (57)–(62).

3.4.4. Momentum equation

Particle momenta and positions are stored in terms of their curvilinear coordinates local to the block which contains them. The momentum coordinates are updated in two stages. First, the momentum components are updated at the old particle position, then are transformed to components measured at the new

position; this eliminates the explicit appearance of the Christoffel symbols terms in the discrete approximation to Eq (36).

The established centred time approximation [7,1,2] is used for the momentum update at the old particle position. Covariant momentum components k at time level $n+1/2$, $\bar{p}_k^{(n+1/2)}$, measured at the current particle positions $\bar{x}^{k(n)}$ are found using

$$\bar{p}_k^{(n+1/2)} - \bar{p}_k^{(n-1/2)} = q\Delta t \left(E_k^{(n)} + e_{klm}(\bar{v}^{l(n-1/2)} + \bar{v}^{l(n+1/2)}) b^{m(n)}/2 \right) \quad (98)$$

where $\bar{v}^l = \bar{p}^l/\gamma m_0$ are contravariant velocity components and e_{klm} is the permutation symbol. Eq. (98) is solved using the method due to Boris [7]; apply a half electric acceleration, compute the relativistic γ, apply a two stage rotate and then complete the electric acceleration. Fields are evaluated at the particle positions using Eqs. (54) and (55).

3.4.5. Position equation

Positions are updated to new time level $n+1$ and the momentum components at the new positions are computed using a predictor-corrector scheme to solve

$$\bar{x}^{k(n+1)} - \bar{x}^{k(n)} = \left(\bar{v}^{k(n+1/2)}(\bar{x}^{(n)}) + \bar{v}^{k(n+1/2)}(\bar{x}^{(n+1)}) \right) \Delta t/2 \quad (99)$$

and

$$\bar{p}^{l(n+1/2)}(\bar{x}^{(n+1)}) = e^l(\bar{x}^{(n+1)}) \cdot e_k(\bar{x}^{(n)}) \bar{p}^{k(n+1/2)}(\bar{x}^{(n)}) \quad (100)$$

In Cartesians, this scheme reduces to the usual Lorentz force leapfrog integration scheme and no iteration is required. In cylindrical coordinates, the predictor-corrector can be replaced by direct matrix inversion in the manner proposed by Boris [7]. Both Cartesian and curvilinear coordinates have been tried for the particle integration, and for orthogonal and near orthogonal meshes they give comparable results. The curvilinear option outlined here is favoured because it requires considerably less work per particle. However, its accuracy on distorted elements has not yet been examined.

3.4.6. Particle boundary conditions

Coordinates of particles leaving a block are collected and sorted into tables of particle passing through given glue-patches or boundary condition patches. Internal boundary conditions are applied at the glue-patches by transforming the particle coordinates to those used in the target block. External boundary conditions are applied by deleting outgoing particles and adding incoming particles according to the surface particle source model (e.g. beam, space charge limited emitter, secondary emitter, etc).

3.5. Coordinate transformations

The coordinate systems used in the software are:

(i) global Cartesians
(ii) block Cartesians
(iii) block orthogonal
(iv) block contravariant
(v) block covariant

Global positioning information comprises two parts: physical position and logical position. The physical position of a block is described in terms of its origin $\boldsymbol{x}_{000}$ and its rotation matrix R. Global physical positioning is used to graphically display the data. Logical position, described by the glue-patches joining blocks together, is used to transform curvilinear field and particle coordinate components between blocks. This is described in Section 4.1.6.

In most cases, the local Cartesian and local orthogonal coordinates are the same. However, for special cases (e.g. polar mesh segment block), it becomes advantageous for metric data compression to store the basis vectors $\boldsymbol{e}_i$ in terms of components of a reference orthogonal system which is not Cartesian. When local orthogonal coordinates are not Cartesian, their scale factors need to be stored.

The basis vectors define the transformations between orthogonal coordinate components and curvilinears, and the covariant / contravariant transformations use the metric tensor. The orthogonal components are used only for input and output.

4. Software

The management of complexity has been the major challenge in developing the PIC software suite 3DPIC;

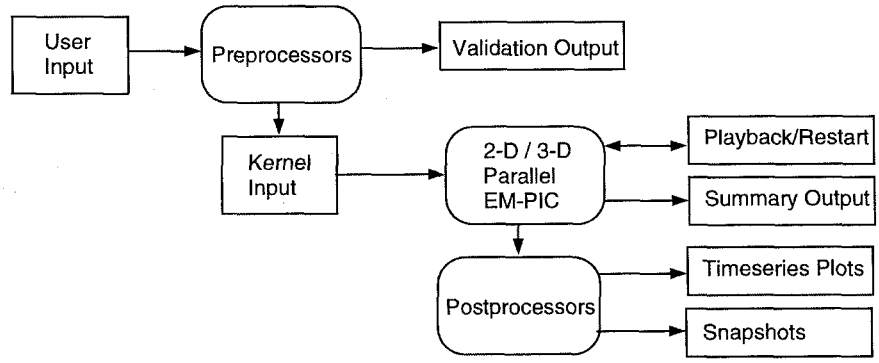

Fig. 6. A schematic of the division of the 3-D electromagnetic PIC software suite 3DPIC into preprocessor, simulation and postprocessor program units.

to produce a general geometry body-fitted code that is both portable and efficiently utilises multiprocessor distributed-memory parallel computers has required careful hierarchical structuring of both data and program units.

The first division is into the interactive (and serial) programs for preprocessing (PEGGIE / P3DTWT) and postprocessing (PICTIM / PICVIS), and into the simulation program (PIC3D) which runs non-interactively on the parallel computer. This is shown schematically in Fig. 6. The preprocessors take compact descriptions of the device geometry and material properties, the external circuit boundary conditions, the initial values for the simulation and the selections of output datasets to be generated and saved. They validate the input for consistency, and produce initialisation summaries and input datasets for the kernel simulation program. Additionally, data may come from restart files from previous runs and from playback files (e.g. beam data) from other software. The preprocessors exchange some flexibility for simplicity, so the kernel input dataset is in human readable form to allow the insertion of special conditions not catered for by the preprocessors. The simulation code PIC3D produces summary output and binary datasets for the postprocessors. These postprocessors take the time series and snapshot datasets and analyse them to produce data in the form suitable for human interpretation. The preprocessors, simulation program and postprocessors share a number of program units, and this is catered for by grouping the program units into a set of libraries. A discussion of the multiblock decomposition is presented in the next section, followed by a brief description of the preprocessors and postprocessors in Sections 4.2 and 4.3.

4.1. Multiblock decomposition

The multiblock decomposition of the computational domain into a set of curvilinear hexahedral bricks (uniblocks), where each uniblock has its own fields and particles and communicates with other blocks and boundaries via its glue-patches, provides our route to efficiency, portability and flexibility. The resulting numerical algorithm within each block is almost as simple as that for Cartesian PIC in a rectangular brick. The regular (i, j, k) addressing within each block allows data compression of basis vector and metric storage (Section 4.1.5).

Fig. 7 uses the simple case of a cylinder to illustrate the multiblock decomposition of a complex object into a set of curvilinear uniblocks connected by glue-patches. The cylinder (a) is divided into five blocks (b). These blocks are treated as separate objects with their own coordinate systems, and communicate by passing field and particle data via the glue-patches (denoted by dashed lines in (c)). In (d) the curvilinear quadrilateral (hexahedral in 3-D) blocks are meshed by transforming them to rectangles (bricks in 3-D) in curved space and dividing the rectangles (bricks) into unit side square (cube) elements.

The simulation program treats a device as a collection of uniblocks connected together by a network of glue-patches. Each glue-patch sticks together two ad-

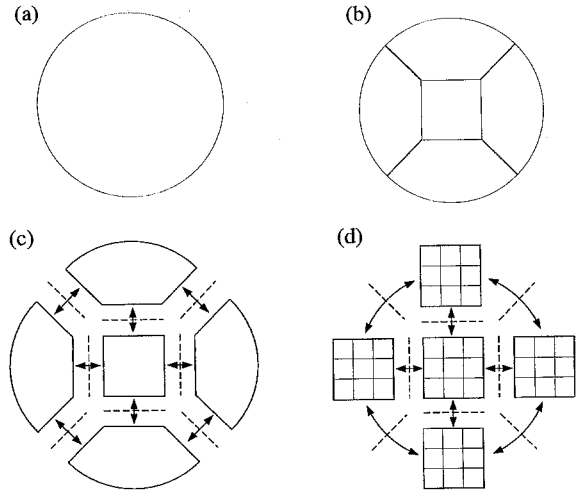

Fig. 7. An illustration of the multiblock decomposition and meshing of a cylinder.

jacent block faces (or block edges in the case of line patches). Different device are constructed by glueing different shaped multiblocks together and by changing boundary condition patches at the uniblock surfaces. Different material types are treated by changing the data associated with the block type.

Uniblocks map naturally onto processes on a MIMD computer architecture; the multiblock decomposition may be thought of as setting up a set of independent PIC calculations on each processor, with the (time dependent) particle and field boundary conditions being provided by messages from the other processors. All the field and particle data for a given block is local, so the only data that must be exchanged for the calculation to proceed is the boundary data.

To handle complex shapes, some uniblocks will need to be small, whereas for simple shapes, physical boundary condition constraints allow large blocks. If desired, large blocks can be subdivided to facilitate mapping onto the MIMD computer. The data organisation allows several small blocks to be collected together on a given process to obtain effective load balance and allows uniblock data to be moved from one process to another.

Without compromising the subdivision of the spatial mesh into blocks to get efficient MIMD processing, one can further demand that boundary conditions

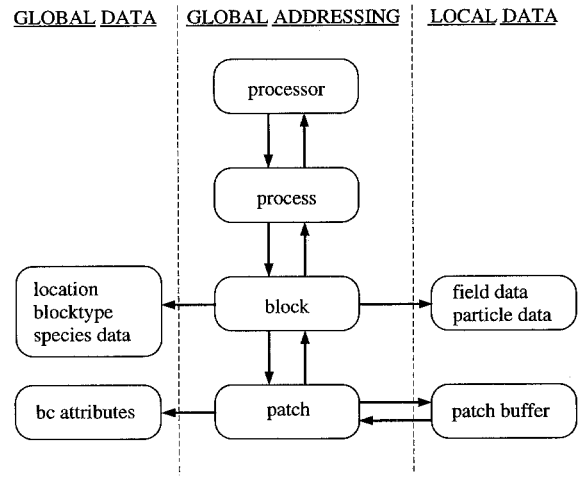

Fig. 8. The data in the main simulation program PIC3D is divided into global addressing, global data and local data.

only apply at the surfaces of blocks [2]. This completely eliminates the addressing problems caused by embedding surfaces within blocks, and allows surface data to be passed to the control program through the glue-patch tables. When a large number of small blocks is needed to describe a complex object, load balancing is achieved by assigning several blocks to one processor.

4.1.1. *Global addressing*

The global addressing in PIC3D describes the distribution of blocks over the processors of the MIMD computer, and the logical connection of the blocks; this addressing is used to pass information from the surface of one block on one processor to the surface of another block which may be on another processor. The complete global addressing network can be described by the positioning of the two faces of each glue-patch on the uniblocks. The four levels in this network, the processor, process, block and patch are shown schematically in Fig. 8.

Processor: To aid portability a distinction is made between the logical processes used in the program and the physical processors of the MIMD computer; tables are set up to provide the mapping between processes

[2] This condition is not strictly applied in the program, as it was found advantageous to allow special blocks with embedded structures to describe the geometry of helical travelling wave tubes.

and processors. The simulation program is written in terms of processes, which are translated to physical processor numbers by these tables. When the software is run on single processor workstations, there is simply one process and one processor.

Process: Each process corresponds to a running (master or slave) copy of the simulation program, and may have one or more blocks assigned to it. This allows the computational load to be balanced by moving blocks between processes.

Block: To each block corresponds global block data, namely the block's location in space relative to some global coordinates, its type and global attributes of the particles within it. The block type data contains metric information needed by the field solver and by the particle integrator.

Patch: All exchange of data between blocks is performed through the patch exchange subroutine. This recognises whether a target patch to which it is to send data is on a block belonging to its process, or on a block being computed by another process. In the former case, a memory-to-memory copy of the data is performed, and in the latter a message is sent to the target process.

Addressing in the direction of the downward arrows in Fig. 8 (processor → process → block → patch) uses ordered tables, and the reverse connections use linked lists.

4.1.2. Global data

Physical and numerical data which spans many blocks is stored as global data; examples of this are particle species data, boundary condition data, block type data (e.g. metrics) and global Cartesian positions of the blocks.

4.1.3. Local addressing

Local addressing manages the storage of the principal field and particle arrays, and the transfer of field and particle data to and from the patch buffers used in the glue-patch exchanges. One-dimensional addressing is used for the principal field and particle arrays to allow the same compact coding to be used for both 2-D and 3-D computations. Linked lists are used to sort particles into the glue-patch buffers.

4.1.4. Local data

The principal local data arrays are those for nodal amplitudes I^i, d^i and b^i, for the particle coordinates $(\bar{x}^i, \bar{p}^i)$, a workspace array to hold alternately E_i and H_i, and input / output buffers for the glue-patch exchanges. On a given process, the nodal amplitude data for blocks are stored as a sequence of fixed length records in the appropriate field or current array. Particle and buffer data use variable length records.

4.1.5. Memory usage

PIC codes are always large consumers of computer memory, and the extra data needed to describe general geometry can greatly inflate this need. This has been avoided in PIC3D by using data compression for metric and basis vector storage, and dynamic memory allocation for the large local data arrays.

In most cases, symmetries and congruences greatly reduce the amount of metric data to be stored. For instance, the example shown in Fig. 7 extended to 3-D perpendicular to the plane shown has two block types only; the central rectangular brick block type, which requires only six numbers to completely specify its basis vectors and metric tensor elements, and the surrounding wedge shaped block type, which requires a single plane of values for metrics. The metric data compression for this case has reduced the memory requirement from the full metric storage for the block $\sim 75N^3$ to $\sim 7+3N^2$, assuming an $N \times N \times N$ mesh in each block. These savings are typical, and similarly apply to the basis vector storage. Physical coordinates of element nodes are not stored, but are constructed when needed (for diagnostic output only).

The preprocessors compute the amount of memory required for the field arrays and estimate how much is needed for particle coordinates and patch exchange buffers; the simulation code allocates this space when it starts up.

4.1.6. Patch exchange

Patch exchanges are used to transfer field and particle data between blocks. Field glue-patch exchanges correspond to the transfer of fixed length records, whereas the length of records for particle exchange varies depending on the particle flux.

Field glue-patches. For all cases, the application of interior field boundary conditions involves assembling

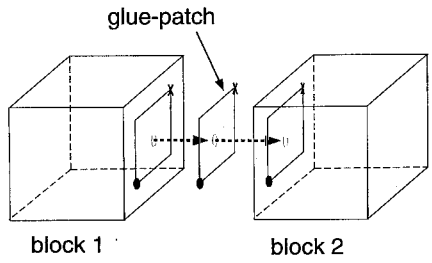

Fig. 9. All interior boundary conditions are applied by passing data between blocks via glue-patches.

contributions to d or E across adjacent block boundaries. This assembly is performed by adding the glue-patch contributions. Fig. 9 illustrates the glue-patch data transfer. Data stored on the logical (i, j, k) lattice of nodes in block 1 are copied to the glue-patch, and then data is added from the glue-patch to the corresponding nodes on block 2. On both source and target blocks are "O' and "X" reference points (solid circles and crosses in Fig. 9) which specify the location and orientation of the blocks with respect to the glue-patch. The reference points are used to perform the field component and indexing transformations between blocks.

Two types of field glue-patch are required:

- *surface patches* to exchange the two tangential field components at nodes common to the faces of two blocks;
- *line patches* to exchange the one tangential field component at nodes common to the edges of four or more blocks. If a node is common to M blocks, then it has associated with it $M(M-3)/2$ line patches.

Interior boundary conditions are applied by adding the glue-patch values d^i_{glue} to the corresponding node accumulator

$$\mathsf{d}^i := \mathsf{d}^i + \mathsf{d}^i_{glue} \tag{101}$$

A simple 2-D example of the use of glue-patches is to apply doubly periodic boundary conditions to a rectangular domain. Four glue-patches are required: two surfaces patches to connect the north to south boundary and the east to west boundary, and two line patches to connect the NE to SW corner and the SE to NW corner. Glue-patch exchanges are required twice each timestep, once for d^i to complete the integration of Ampere's equation, and once for E_i to complete the computation of E from d using the constitutive relationship (71).

Particle glue-patches. Particle exchange requires only surface patches. Each field surface glue-patch has a corresponding particle glue-patch. Particles passing through a glue-patch from a source block to a target block are passed from the storage areas of the source block particle tables to that of the target block via the glue-patch buffer. Position coordinates are transformed from the curved space components of the source block to those of the target block by using the locations of the "O" and "X" keys on the two connecting blocks (cf. Fig. 9). Particles passing through more than one block in a timestep are handled by repeated application of the surface glue-patch exchange.

4.2. Data input

4.2.1. Preprocessors

The generic preprocessor to treat general device geometries is called PEGGIE, for Parallel Electromagnetic Grid Generator Interface and Extensions. As implied by the word Interface, PEGGIE is not primarily intended to be a grid generator, although it has powerful in-built capabilities. The word Extensions relates to the ability to set most of the numerical, physical and housekeeping variables used by the PIC3D code.

The current version of PEGGIE accepts command line input. The use of a small number of tags, combined with FORTRAN 90 NAMELIST emulation makes it easy to specify complex objects in a compact way, set control and diagnostic parameters in a consistent manner, etc.

The in-built grid generator uses the concept that the device to be modelled is made by joining together simpler objects or blocks, each of which is cuboid in a suitably chosen, and in general non-orthogonal, coordinate system. Since element node positions are determined by transfinite interpolation, the point distribution along three adjacent edges defines the grid within the block. If the points are distributed uniformly in some (parametric) sense, then three integers n_i, $i = 1, 2, 3$ serve to define the local mesh. There are global constraints, arising from the need for adjacent objects to have the same point distribution on their common boundary, so the user has the ability to suggest n_i and let PEGGIE try to obtain a consistent set.

The algorithm employed is most easily explained for the case where the constituent blocks have entire faces in common (although partial joins are supported). Essentially a trail is started for each of the three adjacent edges that define a block. The connectivity or patch information on each surface implies a meshing in two coordinate directions for the adjacent block, i.e. two trails extend into it, and in general, another trail starts there. Successive blocks are examined until every trail has ended, typically by extending from one boundary to another. The trails are checked for consistency and where several suggested n_i lie upon a path, the one chosen is that corresponding to the earliest block in order of assembly.

It will be noted that blocks may have different orientations, e.g. the 1- and 2- coordinates in one block may correspond to respectively the 3- and 1- coordinates in an adjacent block. Such alignments are recorded by use of integer 3 vectors $(a\ b\ c)$ corresponding to rotation matrices

$$\begin{pmatrix} \boldsymbol{i}_a \\ \boldsymbol{i}_b \\ \boldsymbol{i}_c \end{pmatrix}$$

where $\boldsymbol{i}_a$ is a unit 3-vector with one entry, the a^{th}, non-zero and having the same sign as a, e.g. (1 -3 2) corresponds to the rotation matrix

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}.$$

A similar technique has been devised independently [24]. The importance of the $(a\ b\ c)$ notation is that quantities such as the n_i may be transformed simply by permutation and sign change.

Once blocks have been gridded, PEGGIE proceeds to generate first the covariant basis vectors $\boldsymbol{e}_i$ and then the coefficients $\mathsf{G}_{ij}^{E,H}$ that describe the combination of block geometry and (in general) tensor permeability and permittivity as explained in Section 3.3.4. To reduce computing time, block symmetries are employed, so for example in an orthogonal cuboid only $\mathsf{G}_{(i)(i)}$, i = 1, 2, 3 for one cell are actually calculated.

The G_{ij} serve to set a maximum timestep. Provided ϵ and μ are uniform, isotropic tensors, it may rigorously be shown for a grid consisting of identical parallelograms, that a necessary and sufficient criterion for stability is

$$c\Delta t\sqrt{\frac{g_{11}+g_{22}}{g}} \le 1 \tag{102}$$

This contrasts with the criterion in Ref. [25] that has explicit off-diagonal terms, apparently because the 4-point sums involving off-diagonal G_{ij} have not been fully taken into account. (The off-diagonal terms appear implicitly in the volume element g.) In 3-D, for general but still homogeneous G_{ij}, it is necessary to resort to more heuristic arguments, which suggest that stability is achieved provided

$$c\Delta t\sqrt{\Gamma} < \frac{1}{\sqrt{3}} \tag{103}$$

where

$$\Gamma = \frac{1}{c^2}\max(\mathsf{G}_{11}^E\mathsf{G}_{22}^H, \mathsf{G}_{22}^E\mathsf{G}_{11}^H, \mathsf{G}_{11}^E\mathsf{G}_{33}^H, \mathsf{G}_{33}^E\mathsf{G}_{11}^H, \mathsf{G}_{22}^E\mathsf{G}_{33}^H, \mathsf{G}_{33}^E\mathsf{G}_{22}^H) \tag{104}$$

and this is used to specify Δt.

PEGGIE proceeds to calculate line patch information, needed where four or more blocks have a common edge. An important feature here is the representation of a block as the cube $[-1,1]^3$ whose faces may be labelled by 3-vectors (i_1,i_2,i_3) = $(\pm1,0,0),(0,\pm1,0)$ and $(0,0,\pm1)$ according their surface normals. Edges can be represented by the 3-vectors that are the sums of two adjacent face vectors, and corners similarly as sums over the three faces that meet. These 3-vectors can each be condensed to one integer $i = 6i_3 + 2i_2 + i_1$ and the i values provide a natural ordering for respectively faces, edges and corners, e.g. DSWENU for faces where D, S, W, E, N and U represent the Down, South, West, East, North and Up faces of the cube $[-1,1]^3$ (so D corresponds to the face with normal (0, 0, -1), etc). A single integer operation decides which is the other cube face that shares a specified edge with a given face.

Answering the latter question is the basis of the line patch calculation: the need is, for each a block edge, to find a list of blocks which are coincident along at least part of that line, given only surface connectivity information. The idea is to describe a trail about the edge through the adjacent blocks, which may be a closed path, or may end at device boundaries, or even in general may split. The first stage of the algorithm is to rearrange the surface patch information, so in particular that for every edge of every block, there is

a list of contiguous surface patches. Starting with an unmarked edge a trail is pursued, marking each block edge (including the starting edge) that is found to be coincident with the first one, until it terminates as described above, or splits, in which case two separate trails are begun and pursued.

The last large set of non-local quantities to be computed is of the positions (x_{000}) of the blocks and their orientations (R) which respect to a global coordinate system. The position and orientation of one specified block defines this frame.

At this point PEGGIE calculates the scalings required to convert fields to the dimensionless units used by PIC3D. These are used at once in the generation of coefficients that treat boundary conditions of an applied electric field, resistive wall etc (see Section 3.3.6).

The timestep and scaled block dimensions are also employed to define control parameters; if the length of a PIC3D run is specified by the number of timesteps, PEGGIE works out suitable times for the saving of diagnostic information to disc. Line, surface and volume integral diagnostics (cf. Section 4.3) may be demanded, that may extend over different blocks.

Making full use of the compressed storage technique (Section 4.1.5), scalings are applied to e_i and G_{ij}^{EH}. Finally the remaining tables, such as the block to process mapping, (essential for PIC3D operations) are set, then the kernel data file suitable for immediate input to PIC3D is written to disc.

4.2.2. *TWT preprocessor*

The travelling wave tube (TWT) preprocessor is the first of a family of preprocessors which are designed for specialist application areas. It builds on existing experience of modelling TWTs in 2-D [26]. This preprocessor, which calls upon the library of routines developed for PEGGIE, provides a much simpler but more restricted method of initialisation.

The preprocessor was developed in collaboration with TWT tube engineers from EEV Ltd, and has been designed to take parameters with which the tube engineers prefer to work and generates the detailed input description that the main simulation code requires. The engineer is not required to set any parameters to define the multiblock and element subdivision of the tube interaction space, or the number of superparticles to be used, or any other numerical parameter to be used, although they can override the defaults if they wish.

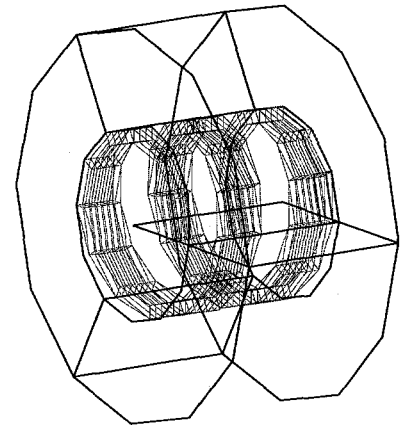

Fig. 10. A preprocessor output showing a section of the blocking of a TWT. Shown are a central cylindrical block containing three turns of the helix, surrounded by three further blocks. The thicker lines are block boundaries, and the thinner are helix element boundaries.

The user input file (`.uif`) used by the TWT preprocessor uses the FORTRAN 90 `NAMELIST` format. An input file can include other files to allow users to set their own defaults, allowing a series of related calculations to be performed with the minimum amount of input. One important feature of the preprocessor is the ability to produce validation output to confirm that the main simulation code is correctly initialised; an example of one such output is shown in Fig. 10.

4.3. *Data output*

The main simulation code uses a hierarchical approach for specifying the output required; plot sets define the output file format, the field set, the domain set and the time set. The field set specifies a set of quantities to be plotted for each domain in the domain set, at the times listed in the time set. Each domain is a union of subdomains, each of which belong to a single block. The implementation is designed to allow new operations and output formats to be added easily.

The present version of the software has snapshot output for a limited range of geometries, and time-series output field specifications for point quantities, line integrals (e.g. $\int \boldsymbol{E} \cdot d\boldsymbol{l}$ or $\int \boldsymbol{H} \cdot d\boldsymbol{l}$), surface inte-

grals ($\int \boldsymbol{D}\cdot d\boldsymbol{S}$, $\int \boldsymbol{B}\cdot d\boldsymbol{S}$ or $\int \boldsymbol{j}\cdot d\boldsymbol{S}$) and volume integrals ($\int \psi dV$ where ψ may be for example particle number, $\boldsymbol{B}\cdot\boldsymbol{H}/2$, $\boldsymbol{E}\cdot\boldsymbol{D}/2$ or $\boldsymbol{j}\cdot\boldsymbol{E}$).

The implementation of the diagnostic output produces separate output for each process. Post processors can then combine data from several processes into a single dataset. This is satisfactory provided that the subdomains of a diagnostic domain set reside on the same process, but for more general situations message passing will also be needed for diagnostics.

5. Parallel benchmarking

The message passing model used in PIC3D matches the structure of massively parallel processors (MPPs) such as the Intel Paragon, IBM SP/2 and Meiko CS-2, and is also implemented efficiently on the parallel vector shared-memory computers such as the Cray Y-MP and C-90. PIC3D currently uses the Intel NX/2 library of communication subroutines (mostly CSEND and CRECV) because these are implemented with minimum overhead on many computers and are widely used. A version of the code using the popular PVM message passing language [27] has been written, and tested on an IBM SP/2. This gives maximum portability, but the scaling with increasing number of processors is unsatisfactory if Ethernet and the the Internet Protocol (IP) are used, due to their high message latency. However the PVM code is expected to scale satisfactorily when run under IBM's customized implementation PVMe [28].

Both 2-D and 3-D test cases have been used to evaluate parallel performance. The former, LPM2 benchmark uses the realistic device geometry of a magnetically insulated line oscillator (MILO), and the latter, LPM3 benchmark uses a 3-D periodic plasma to provide a clearly scalable test which does not depend on non-reproducible details such as random number generators.

5.1. LPM2 benchmarking

The choice of a 2-D MILO case for LPM2 was made because the geometry reflects realistic engineering calculation requirements and the results could be compared with those from other PIC codes [29,30]. Fig. 11 shows the electron scatter plot for a 20 block subdivision of a MILO simulated using PIC3D in its 2-D mode.

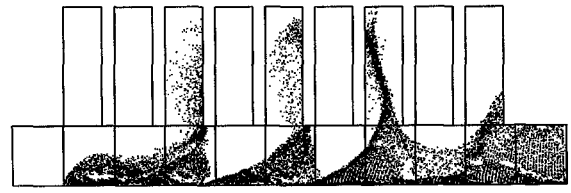

Fig. 11. An example of snapshot output of 2-D MILO simulation by PIC3D. This case is divided into 20 blocks; 9 cavities and 11 interaction space blocks.

LPM2 benchmarking runs undertaken on the Intel iPSC for a 64 block decomposition of a MILO, using a total of approximately 10,000 elements and 13,000 particles showed better than eighty percent of the theoretical speed-up for up to 16 processors, despite the non-uniform distribution of particles and small problem size. However, beyond this limit (i.e. < 4 blocks per process) the number of timesteps per second began to saturate (clearly, beyond 64 processors further speed-up is impossible without further problem subdivision!).

5.2. LPM3 benchmarking

The three-dimensional periodic plasma test case LPM3 (Local Particle Mesh benchmark # 3) was run on the Intel Paragon and Intel iPSC/860. This benchmark problem was chosen to be a representative test of the features of PIC3D on the available MPPs, to be easily scalable in size and to be physically realistic. LPM3 explores the capabilities of MPPs with more than a thousand processors.

In LPM3, the plasma space is divided into blocks, each of which consists of 512 particles representing the electrons, and 64 elements on which the fields are calculated. From the point of view of load balancing the parallel computer, the block is the smallest unit that can be allocated amongst the processors. The problem size is measured by the number of blocks N_b, and four problem sizes have been used with respectively $N_b = 8, 64, 512, 4096$. These correspond respectively to numbers of particles $N_p = 4K$, $32K$, $256K$, $2M_2$ where $K = 1024$ and $M_2 = K^2$. The timestep is such that about ten percent of the particles transit to neighbouring blocks during a timestep. A run of 100

timesteps is chosen as the benchmark test because this takes only a few minutes for the problem sizes and number of processors of interest, and the conservation of total number of particles is used as a validity check.

There are two different versions of the benchmark code, which are selected by an input variable. The *per-patch* version sends a separate message for every patch in the system, and there are 9 patches for every block. In the *per-process* version, on the other hand, the patch messages are assembled and sorted in a buffer so that only one message is sent to every other process to which a given process is attached. The per-process code may send 10 or 100 times fewer messages than the per-patch version, and so should be significantly faster than the per-patch version on computers with a high message start-up time or latency. This difference is clearly seen when the code is run on the IBM SP/2 using the high latency PVM-IP message passing interface. For computers with low latency such as the Intel Paragon using NX/2 under SUNMOS, there is little difference between the versions.

Results: All the benchmark measurements obtained are shown in Fig. 12. The results are expressed as temporal performance R_T in units of timestep per second ($tstep\ s^{-1}$), an unambiguous metric which expresses exactly what a program user needs to maximise [31]. We have deliberately not used the sometimes popular metrics of Parallel Speed-up and Efficiency because these are not absolute measures and cannot be used correctly to compare the performance of different computers [31–33]. Neither have we expressed the results as Megaflop per second, because this would make the curves lie closely on top of each other, and disguise the actual time of computation from the reader. In our units of timestep per second, calculations which take the same time lie at the same height in the graph; faster calculations lie higher and slower lie lower. These statements would not necessarily be true if the results were expressed as speed-up or in megaflop per second.

For each block size there are two pairs of curves. The pair closest to the theoretical dotted line is that for the Intel Paragon running under the Sandia SUNMOS 1.4.8 operating system, and the pair about 60 percent lower and slower are for the iPSC/860 running under the NX release 3.3.2 operating system. For each computer the open symbols are the measured values using the per-process code and the corresponding filled symbols for the per-patch code. The fact that the curves for both versions are almost the same on both the computers, means that message latency is not a problem with the Paragon under SUNMOS or with the iPSC/860. The latency has separately been measured using the COMMS1 "pingpong' benchmark [33] to be about 80 μs for both these computers.

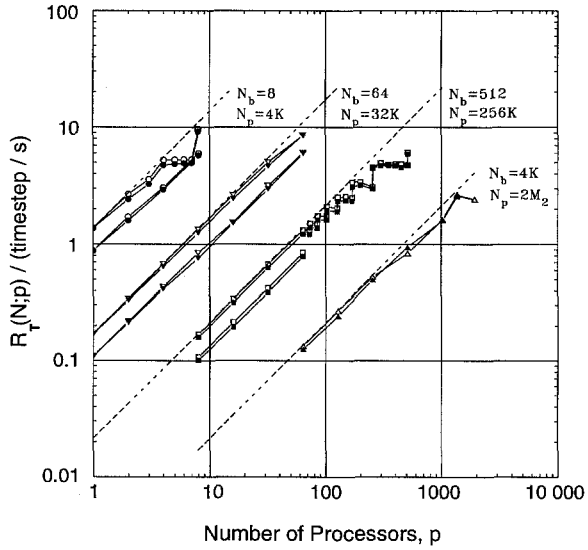

Fig. 12. Temporal Performance of the LPM3 benchmark measured in units of timestep per second, for four problem sizes and up to 1840 processors on the Sandia Laboratory's Intel Paragon XP/S 140 (upper pair of curves which are close to the dotted lines), and up to 64 processors on the Intel iPSC/860 (lower pair of curves). See text for further details.

The dotted lines are the theoretically perfect linear scaling predictions for the Paragon, calculated from the one-processor performance on the eight block problem. The theoretical scaling assumes that performance is proportional to the number of processors, and inversely proportional to the problem size; ideally, p-times as many processors should compute an existing problem p-times faster (the speed-up), or a problem p-times bigger in the same time (the size-up). Alternatively, of course, the increase in number of processors should be able to be used to obtain a combination of speed-up and size-up. The stepwise nature of the measured performance curves arises due to load imbalance when the number of blocks is not exactly divisible by the number of processors. If we

concentrate attention on the best performance figures which correspond to perfect load balance, when every processor is computing the same number of blocks, we find that the measured performance is within 80 percent of the ideal linear scaling except for the 512 block case with greater than 256 processors, and for the 4096 block case with more than 1366 processors. For these cases with larger numbers of processors, performance saturation is beginning to be seen.

For the set of cases shown in Fig. 12 it is evident that there is not enough parallelism in the problem to obtain more than about 10 $tstep\ s^{-1}$. The 8 block problem can at most use 8 processors, and there is no possibility of using more processors to speed up that problem. The best we can do with the extra processors is to solve bigger problems in about the same time, and this is seen for the cases of N_b = 64, 512, 4096, none of which exceed 10 $tstep\ s^{-1}$ however many processors are used.

6. Final remarks

We have devised and implemented new algorithms for electromagnetic Particle-in-Cell (PIC) calculations to enable the simulation of realistic devices with complex geometry. These algorithms build on existing experience with the electromagnetic PIC approach; indeed in simple geometries they reduce to state-of-the-art algorithms. They are economical in that under such circumstances they cost no more, and in that they perform efficiently on virtually all types of computer, i.e. on single and multiple processor machines which may or may not share memory. Moreover many features of the algorithms serve to reduce software costs, by making it easy to change the properties of a model by, say, altering its dielectric properties or surface material.

More precisely, our algorithms extend the finite element method of Lewis [16] to more general element shapes and to the time variable. These methods avoid the quadrature problems encountered in earlier generalisations [34], whilst automatically ensuring charge conservation. The variational formulation gives algorithms with the "energy conserving" rather than the "momentum conserving" property. If desired, similar momentum conserving schemes can be obtained in general geometry without losing the charge conservation property by introducing a field reconstruction step before interpolating force to particle positions.

We have focused in this paper on elements which transform to unit cubes in some general coordinate system; these were chosen for reasons of elegance and computational efficiency. However elements of higher order and of different shape could be treated. With cuboid elements, the tensor formulation provides simple expressions, differing little from those obtained using finite differences on a Cartesian mesh, for the update of fields and particle motion, yet allows fully body-fitted subdivision of the computational domain. The problems of locating the element in which a particle lies, of computing its acceleration and associated current, and of integrating Maxwell's equations, are no more difficult than in Cartesian PIC.

The form of the numerical algorithm allows data to be organised straightforwardly into a form suitable for distributed-memory parallel computers. This has made it possible for 3DPIC to be coded in FORTRAN 77, rather than have to submit to the stricter programming regime of C++ and endure its shortcomings regarding particularly speed of execution [35].

The physics of Maxwell's equations provides a natural organisation of the data, in that interactions at a point in space only involve information from its light cone in space-time. Our software uses this fact to divide a large PIC computations into many smaller ones which communicate only with their immediate neighbour. Hence work can be shared evenly amongst the processors of a MIMD machine, whilst minimising the amount of global data and interprocessor message passing. In addition, the simple logical cube addressing within each block leads to fast serial processing within each slave process.

Benchmarking confirms that our multiblock data organisation offers large computational intensity and weak Amdahl limit on parallel processor speed-up [14]. In particular, we have shown that doubling the size of the computer allows simulations of twice the size to be undertaken in the same elapse time. A similar approach [36] to dividing the computational domain has demonstrated very high efficiencies for PIC calculations in simple geometries.

An alternative to the hierarchical multiblock approach is to use a completely unstructured mesh [37–41]. This has the advantage that topologically difficult objects can be more readily meshed, but the disadvan-

tages that the large data compressions and the ease with which particles can be tracked in our multiblock method are not readily attainable.

The preprocessor, simulation and postprocessor software packages described in this paper are not yet fully mature, but are sufficiently well advanced to show that they will provide effective tools for computer simulation studies of not only parts of microwave devices, but given suitable MPPs, of entire microwave systems with hitherto inaccessible scale lengths.

Acknowledgements

Research sponsored in part by Air Force Office of Scientific Research (AFSC) under contract F61708-93-C-001, by AEA Corporate Research Funding and through subcontract by EEV Ltd under DRA Malvern contract number A 376 415677 E5W. Some of the work reported was undertaken under earlier AFSC contract F49620-92-C-0035 and under contract RAE 1B/7 sponsored by DRA Farnborough.

References

[1] R.W. Hockney and J.W. Eastwood, Computer Simulation using Particles (Adam-Hilger, Bristol, 1988).
[2] C.K. Birdsall and A.B. Langdon, Plasma Physics via Computer Simulation (Adam-Hilger, Bristol, 1991).
[3] D.W. Hewett, Comput. Phys. Commun. 84 (1994) 243–277.
[4] J.W. Eastwood, in: Proc. 7th Ann. Rev. Prog. App. Computational Electromagnetics (Naval Postgraduate School, Monterey, March 1991) pp. 655–660.
[5] J.W. Eastwood, Computer modelling of microwave sources, in: Proc. 18th Annual Conf. Plasma Phys. (IoP, Colchester, 1991) pp. 35–42.
[6] W. Arter and J.W. Eastwood, Electromagnetic modelling in arbitrary geometries by the virtual particle particle-mesh method, in: Proc. 14th Int. Conf. Num. Sim. Plasmas (APS, Annapolis, September 1991) Paper PWE15.
[7] J.P. Boris, Relativistic plasma simulation – optimization of a hybrid code, in: Proc. 4th Conf. Num. Sim. Plasmas (NRL, WA, 1971) p. 3.
[8] B. Goplen, L. Ludeking, J. MacDonald, G. Warren and R. Worl, MAGIC Reference Manual–Algorithms, MRC/WDC-R-201 (Mission Research Corporation, October 1989).
[9] B. Goplen, L. Ludeking, D. Smithe and G. Warren, User-configurable MAGIC for electromagnetic PIC calculations, Comput. Phys. Commun. (1995), this issue.
[10] B. Goplen, J. MacDonald and G. Warren, SOS Reference Manual, MRC/WDC-R-190 (Mission Research Corporation, March 1989).
[11] A.T. Drobot, C.-L. Chang, K. Ko, A. Mankofsky, A. Mondelli, L. Seftor and P. Vitello, IEEE Trans. Nucl. Sci. NS-32 (1985) pp. 2733–2737.
[12] D.B. Seidel, M.L. Kiefer, R.S. Coats, T.D. Pointon, J.P. Quintenz and W.A. Johnson, Multitasking the 3-D electromagnetic PIC code QUICKSILVER, in: Proc. 13th Conf. Num. Sim. Plasmas (Santa Fe, NM, September 1989) Paper IM6.
[13] K.S. Yee, IEEE Trans. Antennas Propagat. 14 (1966) 302–307.
[14] R.W. Hockney and C.R. Jesshope, Parallel Computers 2: Architecture, Programming and Algorithms (Adam Hilger, Bristol, 1988).
[15] J.W. Eastwood, Comput. Phys. Commun. 64 (1991) 252.
[16] H.R. Lewis, Methods Comput. Phys. 9 (1970) 307.
[17] H. Goldstein, Classical Mechanics (Addison-Wesley, Reading, MA, 1950).
[18] W. Arter and J.W. Eastwood, in: Mathematical and Numerical Aspects of Wave Propagation Phenomena, eds. G. Cohen, L. Halpern and P. Joly (SIAM, Philadelphia, 1991) p. 719.
[19] M.R. Spiegel, Vector Analysis (Schaum, New York, 1959).
[20] W.J. Gordon and C.A. Hall, Int. J. Numer. Methods Eng. 7 (1973) 461.
[21] A.D. Burns, N.S. Wilkes, I.P. Jones and J.R. Kightley, FLOW3D: body-fitted coordinates, AERE-R12262 (AEA Harwell, UK, 1986).
[22] A.D. Burns and N.S. Wilkes, A finite difference method for the computation of fluid flows in complex three-dimensional geometries, AERE-R12342 (AEA Harwell, UK, 1986).
[23] B.B. Godfrey, J. Comput. Phys. 15 (1974) 504–521.
[24] A. Rizzi, P. Eliasson, I. Lindblad, C. Hirsch, C. Lacor and J. Haeuser, Comput. Fluids 22 (1993) 341.
[25] J.-F. Lee, R. Palandech and R. Mittra, IEEE Trans. Microwave Theory Tech. 40 (1992) 346.
[26] N.J. Brealey, in: High Power Microwave Generation and Applications, eds. E. Sindoni and C. Wharton (SIF, Bologna, 1992) p. 549.
[27] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam, PVM 3.0 User's Guide and Reference Guide, ORNL report (Oak Ridge National Laboratory, Oak Ridge, TN, 1993).
[28] R.W. Hockney, The communication challenge for MPP: Intel Paragon and Meiko CS-2, Parallel Computing 20 (1994) 389–398.
[29] J.W. Eastwood, Computer modelling of high power microwave sources, in: High Power Microwave Generation and Applications, eds. E. Sindoni and C. Wharton (SIF, Bologna, 1992) pp. 539–548.
[30] M. Amman, private communication (1993).
[31] R.W. Hockney, A framework for benchmark performance analysis, Supercomputer 48 IX (1992) 9–22. (Also published in: J.J. Dongarra and W. Gentzsch, eds, Computer Benchmarks (Elsevier, Amsterdam, 1993) pp. 65-76.)
[32] D.H. Bailey, Twelve ways to fool the masses in performance evaluation, Supercomputer 45 VII (1991) 4–7.

[33] R.W. Hockney and M. Berry, eds, PARKBENCH report: public international benchmark for parallel computers, Sci. Program. 3 (1994) 101—146.
[34] B.B. Godfrey, Application of Galerkin's method to particle in cell plasma simulation, in: Proc. 8th Conf. Num. Sim. Plasmas (Monterey, CA, 1978).
[35] D.W. Forslund, C. Wingate, P. Ford, J.S. Junkins, J. Jackson and S.C. Pope, in: Proc USENIX C++ Conference (USENIX Associates, Berkeley, 1990) p. 177.
[36] P.C. Liewer and V.K. Decyk, J. Comput. Phys. 85 (1989) 302.
[37] D. Seldner and T. Westermann, J. Comput. Phys. 79 (1988) 1.
[38] R. Lohner and J. Ambrosiano, J. Comput. Phys. 91 (1990) 22.
[39] F. Assous, P. Degond and J. Segre, Comput. Phys. Commun. 72 (1992) 105.
[40] F. Hermeline, J. Comput. Phys. 106 (1993) 1.
[41] F. Assous, P. Degond, E. Heintze, P.-A. Raviart and J. Segre, J. Comput. Phys. 109 (1993) 222.