Simulation of the Hydrodynamic Device Model on Distributed Memory Parallel Computers

N. R. Aluru, K. H. Law, and R. W. Dutton, *Fellow, IEEE*

Abstract— **Stable and robust finite element methods for the convective hydrodynamic model of semiconductor devices are developed and implemented on distributed memory parallel computers. Specifically, a stable and accurate space-time and Galerkin/least-squares finite element formulation is developed for the hydrodynamic transport equations for the conservation laws. In addition, Galerkin finite element methods are employed for the Poisson and lattice thermal diffusion equations. The inclusion of the lattice thermal diffusion equation in the numerical solution of the convective hydrodynamic model is presented for the first time. Numerical results for a bipolar transistor are included to illustrate the effectiveness of the numerical methods. Numerical simulations of semiconductor devices using the convective hydrodynamic model require a significant amount of computations. A single-program-multiple-data (SPMD) programming model is proposed for the implementation of the hydrodynamic model on distributed memory parallel computers. Employing the SPMD programming model, a serial program developed on a workstation can be converted into a parallel program with minimal changes. The parallel program has been ported to a wide range of distributed memory parallel computers including the iPSC/860 hypercube, the Touchstone Delta machine, and the IBM SP-1. Parallel performance results are reported for a bipolar transistor, silicon MESFET and diodes. The results indicate that the parallel hydrodynamic device simulator exhibits excellent speedups and scalability on distributed memory parallel computers with minimum communication overhead.**

I. Introduction

NUMERICAL studies employing the convective hydrodynamic model (simply referred hereafter as the hydrodynamic model) cannot be conducted trivially since conventional numerical methods often fail when convective terms are included. The classical Scharfetter–Gummel method [1] for discretization, that can be extended for simplified hydrodynamic models [2], [3] and the energy-transport models [4], does not work well for the hydrodynamic model. Advanced finite difference- and volume-based schemes have been proposed for the single carrier simulations of the hydrodynamic model [5], [6]. We have introduced advanced, provably stable, and accurate finite element methods for the simulation of the hydrodynamic model [7]–[9]; however, the studies have ignored the lattice thermal diffusion effects so far. For comprehensive semiconductor device simulations, the inclusion of the lattice thermal diffusion equation is mandatory. Due to the intensive computation time required for the hydrodynamic device simulations, high performance parallel computers are inevitable. This paper presents stable and robust finite element methods for comprehensive hydrodynamic device simulations and an implementation of the finite element program, FInite Element Simulator for Transport Analysis (FIESTA), on distributed memory parallel computers.

The hydrodynamic transport model is discretized by employing the finite element methods. Specifically, the Poisson and the lattice thermal diffusion equations are discretized by a Galerkin finite element method. The standard Galerkin finite element method can be shown to be unstable for electron and hole hydrodynamic equations. Hence, they are discretized by employing a Galerkin/least-squares finite element method. The time dependence of the electron and hole hydrodynamic equations is discretized by employing a discontinuous Galerkin method in time. With a discontinuous Galerkin in time and a Galerkin/least-squares in space, the numerical method employed for the discretization of the electron and hole hydrodynamic equations is referred to as a space-time Galerkin/least-squares finite element method. The finite element discretization techniques for the hydrodynamic transport model are briefly summarized in this paper.

The two most popular parallel programming approaches are the data-parallel and the single-program-multiple-data (SPMD) models. The data-parallel computational model is most suitable for computers with a large number of processing elements and a global shared space, e.g., MASPAR, CM-2, or CM-5. The data-parallel programming model can be referred to as a synchronous model of execution in that parallelism is obtained through the simultaneous execution of the same set of operations across multiple sets of data. Data-parallel programs are generally easier to write and debug because of the synchronous nature of the model. However, in data-parallel programs, the original serial program may need significant changes in the data-structures. SPMD programming model is preferred when using distributed memory parallel computers such as iPSC/860 or IBM SP-1/2 for large engineering application software. In the SPMD programming model, the same program is assigned on every processor [10] and the processors are coordinated with each other at synchronization points. The SPMD programming model is referred to as an asynchronous model because between the synchronization points, every processor executes instructions at its own pace, and different instruction sets are executed by different processors depending upon the data available and the conditional statements they satisfy.

Manuscript received May 10, 1995; revised October 3, 1995, January 10, 1996, and May 21, 1996. This work was supported by ARPA through Contract DAAL 03-91-C-0043. This paper was recommended by Associate Editor S. Duvall.

The authors are with the Center for Integrated Systems, CISX 333, Stanford University, Stanford, CA 94305 USA.

Publisher Item Identifier S 0278-0070(96)06724-3.

SPMD program model offers the advantages of portability to a wide range of distributed memory computers, as well as requiring minimal changes to the original serial code and the data structures (except for the preprocessing module). An application of the SPMD programming model to the drift-diffusion based simulator PISCES has been reported in [11]. The development of general semiconductor device simulators using the SPMD model has been briefly presented in [12]. In this paper, an SPMD programming model is designed for the hydrodynamic device simulator, FIESTA. Implementations on distributed memory parallel computers, including the Intel's iPSC/860, Touchstone Delta, and the IBM SP-1 are described.

This paper is organized as follows: the hydrodynamic transport model comprising the Poisson, thermal lattice, electron hydrodynamic, and hole hydrodynamic equations is reviewed in Section II. In that section, the constitutive equations for the hydrodynamic model are summarized and the similarity of the electron and hole hydrodynamic equations to the equations of ideal gas is noted. Section III presents details on the finite element formulations for the lattice equation, Poisson equation, and the electron and hole hydrodynamic equations. The distributed memory parallel computers and the communication model employed in this study are discussed in Section IV. The serial and parallel program organizations are presented in Section V. The preprocessing steps that are needed to render the program executed effectively on the parallel computers are discussed in Section VI. The parallel finite element program and the parallel solution methods are described in Section VII. Numerical results for a bipolar transistor and the performance of the parallel program for several device simulations are presented in Section VIII. Section IX concludes the paper with a discussion on this study.

II. The Transport Model

This section briefly reviews the hydrodynamic transport model which consists of the Poisson equation, the lattice thermal diffusion equation, and the electron and hole hydrodynamic equations.

2.1. Poisson Equation

Derived from the Maxwell's equations [13], the Poisson equation is given as

$$\nabla \cdot (\varepsilon \nabla \psi) = q(c_1 - c_2 - N_D^+ + N_A^-) \tag{1}$$

where q is the charge, ε is the permittivity, ψ is the electrostatic potential, c_1 and c_2 are the concentrations of electrons and holes, respectively, and N_D^+ and N_A^- are the concentrations of ionized donors and acceptors, respectively. The electrostatic potential is related to the electric field $\boldsymbol{E}$ by the relation

$$\boldsymbol{E} = -\nabla \psi. \tag{2}$$

The electron and hole concentrations are related to the electrostatic potential by the relations

$$\begin{aligned} c_1 &= c_{\text{int}} e^{q(\psi - \varphi_n)/(k_b T_1)} \\ c_2 &= c_{\text{int}} e^{q(\varphi_o - \psi)/(k_b T_2)} \end{aligned} \tag{3}$$

where c_{int} is the intrinsic carrier concentration for the silicon material, k_b is the Boltzmann constant, φ_n and φ_p are the electron and hole quasi-fermi potentials, respectively, and T_1 and T_2 are the electron and hole carrier temperatures, respectively.

2.2. Lattice Equation

The lattice thermal diffusion equation describes the variation of the lattice temperature in the semiconductor device and is given by the following equation [14]

$$\nabla \cdot (\kappa_l \nabla T_l) + H = 0 \tag{4}$$

where κ_l is the lattice thermal conductivity, T_l is the lattice temperature and H is the net heat source for the lattice. The lattice thermal conductivity is related to the lattice temperature by the expression

$$\kappa_l(T_l) = \kappa_0 \left(\frac{T_l}{300} \right)^{-1.2} \tag{5}$$

where κ_0 is the thermal conductivity at $T_l = 300$ K and, in this study, takes the value of 1.45 W/K-cm. By considering the total energy conservation in the electrons, holes, and lattice, the expression for the net heat source H is written as

$$\begin{aligned} H \approx{}& \frac{3c_1}{2} \frac{k_b(T_1 - T_l)}{\tau_{w1}} + \frac{3c_2}{2} \frac{k_b(T_2 - T_l)}{\tau_{w2}} \\ &+ \left(\frac{3}{2} k_b (T_1 + T_2) + E_{\text{gap}} \right) R_{\text{SRH}} \end{aligned} \tag{6}$$

where R_{SRH} is the Shockley-Read-Hall recombination rate, E_{gap} is the band gap energy, τ_{w1} and τ_{w2} are the energy relaxation times for the electron and hole, respectively. In (6), the first and second terms represent the energy transfer from the hot carrier electrons and holes to the lattice due to their temperature differences, and the last term represents the energy released due to recombination processes.

2.3. Hydrodynamic Equations

The electron and hole hydrodynamic equations can be derived from the first three moments of the Boltzmann Transport Equation (BTE) [15]. The zeroth, first, and second moments of the BTE correspond to the particle continuity, conservation of momentum, and energy, respectively. The conservation laws for the particles can be stated as follows:

$$\frac{\partial c_\alpha}{\partial t} + \nabla \cdot (c_\alpha \boldsymbol{u}_\alpha) = \left[\frac{\partial c_\alpha}{\partial t} \right]_{\text{col}} \tag{7}$$

$$\begin{aligned} &\frac{\partial \boldsymbol{p}_\alpha}{\partial t} + \boldsymbol{u}_\alpha (\nabla \cdot \boldsymbol{p}_\alpha) + (\boldsymbol{p}_\alpha \cdot \nabla) \boldsymbol{u}_\alpha \\ &\quad = (-1)^\alpha q c_\alpha \boldsymbol{E} - \nabla (c_\alpha k_b T_\alpha) + \left[\frac{\partial \boldsymbol{p}_\alpha}{\partial t} \right]_{\text{col}} \end{aligned} \tag{8}$$

$$\begin{aligned} &\frac{\partial w_\alpha}{\partial t} + \nabla \cdot (\boldsymbol{u}_\alpha w_\alpha) = (-1)^\alpha q c_\alpha (\boldsymbol{u}_\alpha \cdot \boldsymbol{E}) \\ &\quad - \nabla \cdot (\boldsymbol{u}_\alpha c_\alpha k_b T_\alpha) - \nabla \cdot \boldsymbol{q}_\alpha^{\text{heat}} + \left[\frac{\partial w_\alpha}{\partial t} \right]_{\text{col}} \end{aligned} \tag{9}$$

where $\alpha = 1$ or 2, denoting the system for the electron or hole particles, respectively. In (7)–(9), c_α is the particle concentration, $\boldsymbol{u}_\alpha$ is the particle velocity vector, $\boldsymbol{p}_\alpha$ is the particle

momentum density vector, T_α is the particle temperature, w_α is the particle energy density, $\boldsymbol{q}_\alpha^{\text{heat}}$ is the particle heat flux vector and $[\ ]_{\text{col}}$ denotes the collision terms accounting for the particle-particle interactions, particle-lattice interactions, the transfer of energy between particle and lattice, and the generation and recombination processes. Within the context of this paper, the Greek subscript α designates the electron and hole systems according to the above stated convention and the repeated Greek subscript does not imply summation.

Constitutive Equations: Equations (7)–(9) represent an indeterminate system of equations because the number of unknowns are more than the number of equations. In order to facilitate a solution to the device model, a few constitutive approximations need to be made. The carrier momentum density vector can be written as

$$\boldsymbol{p}_\alpha = m_\alpha c_\alpha \boldsymbol{u}_\alpha \tag{10}$$

where m_α represents the particle mass. The carrier energy density can be expressed as

$$w_\alpha = \tfrac{3}{2} c_\alpha k_b T_\alpha + \tfrac{1}{2} m_\alpha c_\alpha |\boldsymbol{u}_\alpha|^2. \tag{11}$$

The heat conduction is assumed to be given by the Fourier law as

$$\boldsymbol{q}_\alpha^{\text{heat}} = -\kappa_\alpha \nabla T_\alpha. \tag{12}$$

The particle heat-conductivity κ_α is given by the Wiedemann–Franz law as

$$\kappa_\alpha = \left(\frac{5}{2} + \zeta\right) \frac{\mu_{0\alpha} c_\alpha k_b^2 T_l}{q} \tag{13}$$

where $\mu_{0\alpha}$ is the low-field mobility of the particle, T_l is the lattice temperature, and ζ is a parameter associated with the energy dependence of the momentum relaxation time. A standard value of $\zeta = -1$ was employed in [5], [7]. A calibrated value of $\zeta = -2$ was proposed in [16] and is employed in this study.

The collision term $[\partial c_\alpha / \partial t]_{\text{col}}$ in (7) describes the rate of change of particle concentration due to collisions. This term is neglected for single carrier devices. In the presence of both electrons and holes, however, this collision term has significant contribution to the transport equations and introduces coupling between the electron and hole transport systems. The collision term for the continuity equation describes the generation and recombination processes and has the following form

$$\left[\frac{\partial c_\alpha}{\partial t}\right]_{\text{col}} = G - R \tag{14}$$

where G is the avalanche generation term and R is the recombination term. The recombination term is the sum of Shockley-Read-Hall and Auger recombinations [13], i.e.,

$$R = R_{\text{SRH}} + R_{\text{AU}}. \tag{15}$$

The physical processes involved with the Auger recombination and the avalanche generation terms remain a subject of active investigation. These terms, although they can be easily included in the finite element program, are not modeled in this study. The Shockley-Read-Hall recombination is given by

$$R_{\text{SRH}} = \frac{c_1 c_2 - c_{\text{int}}^2}{\tau_2(c_1 + c_{\text{int}}) + \tau_1(c_2 + c_{\text{int}})} \tag{16}$$

where $\tau_\alpha (\alpha = 1, 2)$ is the life time of the particle and assumes a value of 10^{-7} s in this study.

The collision term $[\partial \boldsymbol{p}_\alpha / \partial t]_{\text{col}}$ in (8) describes the particle rate of change of momentum due to collisions. This collision term can be treated by employing a relaxation time approximation as [17]

$$\left[\frac{\partial \boldsymbol{p}_\alpha}{\partial t}\right]_{\text{col}} = -\frac{\boldsymbol{p}_\alpha}{\tau_{p\alpha}} + \frac{\boldsymbol{p}_\alpha}{c_\alpha}\left[\frac{\partial c_\alpha}{\partial t}\right]_{\text{col}}. \tag{17}$$

In the above equation, $\tau_{p\alpha}$ denotes the momentum relaxation time given by

$$\tau_{p\alpha} = \frac{m_\alpha \mu_{0\alpha} T_l}{q T_\alpha}. \tag{18}$$

The collision term $[\partial w_\alpha / \partial t]_{\text{col}}$ in (9) describes the particle rate of change of energy due to collisions. This collision term can also be treated by employing a relaxation time approximation as

$$\left[\frac{\partial w_\alpha}{\partial t}\right]_{\text{col}} = -\frac{(w_\alpha - w_{0\alpha})}{\tau_{w\alpha}} + \frac{w_\alpha}{c_\alpha}\left[\frac{\partial c_\alpha}{\partial t}\right]_{\text{col}}. \tag{19}$$

In (19)

$$w_{0\alpha} = \tfrac{3}{2} c_\alpha k_b T_l \tag{20}$$

denotes the equilibrium energy density, $\tau_{w\alpha}$ denotes the energy relaxation time expressed as

$$\tau_{w\alpha} = \frac{3\mu_{0\alpha}}{2q v_{s\alpha}^2}\left(\frac{k_b T_\alpha T_l}{T_\alpha + T_l}\right) + \frac{\tau_{p\alpha}}{2} \tag{21}$$

and $v_{s\alpha}$ denotes the particle saturation velocity. The second term in (17) and (19) accounts for the rate of change of momentum and energy density due to particle generation and recombination processes, respectively. Since the validity of these terms remains a subject of active investigation, they are included in our model as an option, but they are not included in the simulation results presented in this paper.

Relationship of Device Models to Fluid Dynamic Equations: The convective hydrodynamic device model and other commonly employed semiconductor device models such as the drift-diffusion (DD) [13] and the simplified hydrodynamic models [18], which are obtained by considering moments of the Boltzmann equation with different assumptions, can be shown to resemble fluid equations. The following remarks can be made regarding the resemblance of device equations to fluid equations.

1) The DD model is the simplest of all device models and assumes isothermal conditions. This model can be shown to resemble the equations of isothermal, compressible Stokes flow with source terms as electric field and collision terms.
2) A simplified hydrodynamic model can be obtained from the more general form presented in this section by neglecting the convective terms in the momentum equation

and by assuming that the kinetic energy is negligible. This model can be shown to resemble the equations of compressible Stokes flow with source terms as electric field and collision terms.

3) The convective hydrodynamic equations considered in this study resemble the equations of compressible Euler and Navier-Stokes equations [5], [7].

III. Finite Element Formulation

A Galerkin finite element method and a new space-time Galerkin/least-squares finite element method are developed for the hydrodynamic transport model. The numerical methods proposed for the lattice equation [19] have been based so far on the finite difference and finite volumes techniques. We introduce a Galerkin finite element method for the lattice equation. A Galerkin finite element method has also been developed [7] for the elliptic Poisson equation. A space-time Galerkin/least-squares finite element method has been implemented for the simulation of single carrier hydrodynamic equations [7]. The generalization of this method to two-carrier devices is discussed in this paper.

3.1. Lattice Equation

A Galerkin finite element method is proposed for the elliptic lattice thermal diffusion equation. The Galerkin finite element formulation for the lattice equation can be summarized as follows.

Let the variational functional spaces $\mathcal{S}_l$ (subscript l denotes lattice equation) and ϑ_l both consist of continuous functions with square integrable first derivatives. The solution space $\mathcal{S}_l$ is the set of all such functions satisfying the essential boundary conditions. The weighting function space ϑ_l is made up of functions whose value is zero where essential boundary conditions are specified, i.e.,

$$\begin{aligned}\{\mathcal{S}_l &= T_l | T_l \in H^1(\Omega), T_l = g_l \text{ on } \Gamma_g\}\\ \{\vartheta_p &= \overline{w} | \overline{w} \in H^1(\Omega), \overline{w} = 0 \text{ on } \Gamma_g\}\end{aligned} \tag{22}$$

where g_l are the prescribed essential boundary conditions applied on the boundary Γ_g. In the discussion of the finite element formulation, we employ the following notation for the definition of the weak and Galerkin forms

$$\begin{aligned}a_l(u,v)_\Omega &= \int_\Omega u_{,i}\ \kappa_l(v)\ v_{,i}\ d\Omega\\ (u,v)_{\Gamma_{hi}} &= \int_{\Gamma_{hi}} uv\ d\Gamma\end{aligned} \tag{23}$$

where Ω denotes the multidimensional spatial domain with boundary Γ, u, v are arbitrary variables, $a_l(u,v)_\Omega$ defines the operator acting on u, v over the region Ω, and $(u,v)_{\Gamma_{hi}}$ denotes the inner product over Γ_{h_i}.

Weak Form: The weak form is stated as follows. Given f_{L1}, f_{L2}, and h_i, find $T_l \in \mathcal{S}_l$ such that for all $\overline{w} \in \vartheta_l$

$$B_l(\overline{w}, T_l) = L_l(\overline{w}) \tag{24}$$

where

$$\begin{aligned}B_l(\overline{w}, T_l) &= a_l(\overline{w}, T_l)_\Omega + (\overline{w}, f_{L1} T_l)_\Omega\\ L_l(\overline{w}) &= (\overline{w}, f_{L2})_\Omega + (\overline{w}, \kappa_l h_i)_{\Gamma_{hi}}.\end{aligned} \tag{25}$$

In the above equations, f_{L1} and f_{L2} are known constants given as

$$f_{L1} = (3/2)(c_1 k_b/\tau_{w1}) + (3/2)(c_2 k_b/\tau_{w2}) \tag{26}$$

$$\begin{aligned}f_{L2} = {}&(3/2)(c_1 k_b/\tau_{w1} T_1 + (3/2)(c_2 k_b/\tau_{w2}) T_2\\ &+ ((3/2) k_b (T_1 + T_2) + E_{\text{gap}}) R_{\text{SRH}}\end{aligned} \tag{27}$$

and $h_i = T_{l,i}\, n_i$ are the natural boundary conditions prescribed on boundary Γ_{h_i} with the unit outward normal n_i.

Galerkin Form: Let $\mathcal{S}_l^h$ and ϑ_l^h be the finite-dimensional approximations to $\mathcal{S}_l$ and ϑ_l, respectively. The Galerkin formulation can be stated as follows. Given f_{L1}, f_{L2}, and h_i, find $T_l^h \in \mathcal{S}_l^h$ such that for all $\overline{w}^h \in \vartheta_l^h$

$$B_l(\overline{w}^h, T_l^h) = L_l(\overline{w}^h). \tag{28}$$

The Galerkin finite element method can be shown to be stable for the lattice equation [20]. Approximating T_l^h and $\overline{w}^h$ by linear basis functions, a nonlinear equation is obtained

$$\boldsymbol{G}_l(\boldsymbol{d}) = 0 \tag{29}$$

where $\boldsymbol{d}$ is the unknown lattice temperature at each mesh node. The nonlinear system (29) can be solved using a Newton's method. The linearized form of (29) at iteration k can be written as

$$\boldsymbol{M}_l^{(k)} \Delta \boldsymbol{d} = -\boldsymbol{R}_l^{(k)} = -\boldsymbol{G}_l(\boldsymbol{d}^{(k)}) \tag{30}$$

where $\boldsymbol{M}_l^{(k)}$ is the tangent or the Jacobian matrix at iteration k, $\boldsymbol{R}_l^{(k)}$ is the residual vector at iteration k, and $\Delta \boldsymbol{d}$ is the solution increment.

3.2. Poisson Equation

For completeness, the finite element formulation for the Poisson equation is briefly summarized in this section. The solution and weighting function spaces are given as

$$\begin{aligned}\{\mathcal{S}_p &= \psi | \psi \in H^1(\Omega), \psi = g_p \text{ on } \Gamma_g\}\\ \{\vartheta_p &= \overline{\psi} | \overline{\psi} \in H^1(\Omega, \overline{\psi} = 0 \text{ on } \Gamma_g\}\end{aligned} \tag{31}$$

where $\overline{\psi}$ is the arbitrary weighting function, and g_p are the prescribed essential boundary conditions applied on the boundary Γ_g. Consider the following notation for the definition of the Galerkin form

$$a_p(u,v) = \int_\Omega u_{,i}\ \varepsilon\ v_{,i}\ d\Omega. \tag{32}$$

Let $\mathcal{S}_p^h$ and ϑ_p^h be the finite-dimensional approximations to $\mathcal{S}_p$ and ϑ_p, respectively. The Galerkin formulation can be stated as follows. Given ε, f_p, and h_i, find $\psi^h \in \mathcal{S}_p^h$ such that for all $\overline{\psi}^h \in \vartheta_p^h$

$$B_p(\overline{\psi}^h, \psi^h) = L_p(\overline{\psi}^h) \tag{33}$$

where

$$B_p(\overline{\psi}^h, \psi^h) = a_p(\overline{\psi}^h, \psi^h) \tag{34}$$

$$L_p(\overline{\psi}^h) = (\overline{\psi}^h, f_p)_\Omega + (\overline{\psi}^h, \varepsilon h_i)_{\Gamma_{hi}}. \tag{35}$$

In the above equations, $f_p = -q(c_1 - c_2 - N_D^+ + N_A^-)$ and $h_i = \psi_{,i}\, n_i$.

3.3. Hydrodynamic Equations

The finite element formulation of the hydrodynamic equations is briefly summarized in this section and a complete mathematical treatment can be found in [20]. The hydrodynamic equations (7)–(9) can be rewritten in a system form as

$$\boldsymbol{U}_{\alpha,t} + \boldsymbol{F}^{c}_{\alpha i,i} + \boldsymbol{F}_{\alpha} \tag{36}$$

where $\boldsymbol{U}_{\alpha}$ are conservation variables, $\boldsymbol{F}^{c}_{\alpha}, \boldsymbol{F}^{h}_{\alpha}$, and $\boldsymbol{F}_{\alpha}$ are convective, heat flux, and source vectors, respectively, and an inferior comma denotes differentiation. Equation (36) is stated in a quasi-linear form as

$$\boldsymbol{U}_{\alpha,t} + \boldsymbol{A}_{\alpha i}\boldsymbol{U}_{\alpha,i} = (\boldsymbol{K}_{\alpha ij}\boldsymbol{U}_{\alpha,j})_{,i} + \boldsymbol{F}_{\alpha} \tag{37}$$

where the advection and diffusion matrices are related to the convective and heat flux vectors by the relations $\boldsymbol{A}_{\alpha i} = \boldsymbol{F}^{c}_{\alpha i,\boldsymbol{U}_{\alpha}}$ and $\boldsymbol{K}_{\alpha ij}\boldsymbol{U}_{\alpha,j} = \boldsymbol{F}^{h}_{\alpha i}$, respectively. The matrices $\boldsymbol{A}_{\alpha i}$ do not possess the properties of symmetry or positiveness and, in general, are functions of $\boldsymbol{U}_{\alpha}$. A symmetrized system of equations can be developed for (37) by employing generalized entropy functions [7]. The symmetrized system of equations is written as

$$\tilde{\boldsymbol{A}}_{\alpha 0}\boldsymbol{V}_{\alpha,t} + \tilde{\boldsymbol{A}}_{\alpha i}\boldsymbol{V}_{\alpha,i} = (\tilde{\boldsymbol{K}}_{\alpha ij}\boldsymbol{V}_{\alpha,j})_{,i} + \boldsymbol{F}_{\alpha} \tag{38}$$

where the matrix operators $\tilde{\boldsymbol{A}}_{\alpha i}, \tilde{\boldsymbol{K}}_{\alpha ij}$ are symmetric and $\tilde{\boldsymbol{A}}_{\alpha 0}$ is symmetric and positive definite. $\boldsymbol{V}_{\alpha}$ are referred to as entropy variables. The definitions for the flux vectors, matrix operators, entropy variables, and generalized entropy functions can be found in [20]. The symmetry property is fundamentally important in the development of a stable numerical method [21].

The variational functional spaces $\mathcal{S}_n$ and ϑ_n both consist of continuous functions with square integrable first derivatives within each space-time slab. The solution space $\mathcal{S}_n$ is the set of all such functions satisfying the essential boundary conditions. While the weighting function space ϑ_n is made up of functions whose value is zero where essential boundary conditions are specified, i.e.,

$$\begin{aligned}\mathcal{S}_n &= \{\boldsymbol{V}_{\alpha} | \boldsymbol{V}_{\alpha} \in H^1(Q_n), \boldsymbol{D}_{\alpha}(\boldsymbol{V}_{\alpha}) = \boldsymbol{g}_{\alpha}(t) \text{ on } \Pi_n\} \\ \vartheta_n &= \{\boldsymbol{W}_{\alpha} | \boldsymbol{W}_{\alpha} \in H^1(Q_n), \boldsymbol{D}'_{\alpha}(\boldsymbol{W}_{\alpha}) = 0 \text{ on } \Pi_n\}\end{aligned} \tag{39}$$

where $Q_n = \Omega \times I_n$ is the space-time slab with boundary $\Pi_n = \Gamma \times I_n \cdot \boldsymbol{D}_{\alpha}$ and $\boldsymbol{D}'_{\alpha}$ denote the nonlinear boundary condition operators for the αth carrier and $\boldsymbol{g}_{\alpha}$denotes the vector of prescribed boundary conditions. $I_n =]t_n, t_{n+1}[$ denotes the nth time interval with t_n and t_{n+1} as the nth and $(n+1)$th time levels, respectively. Consider the following notation for the definition of weak and Galerkin forms

$$\begin{aligned}(\boldsymbol{W}_{\alpha}, \boldsymbol{V}_{\alpha})_{Q_n} &= \int_{Q_n} (\boldsymbol{W}_{\alpha} \cdot \boldsymbol{V}_{\alpha})\, dQ \\ (\boldsymbol{W}_{\alpha}, \boldsymbol{V}_{\alpha})_{\Omega} &= \int_{\Omega} (\boldsymbol{W}_{\alpha} \cdot \boldsymbol{V}_{\alpha})\, d\Omega \\ a(\boldsymbol{W}_{\alpha}, \boldsymbol{V}_{\alpha})_{Q_n} &= \int_{Q_n} (\boldsymbol{W}_{\alpha,i} \cdot \tilde{\boldsymbol{K}}_{\alpha ij}\boldsymbol{V}_{\alpha,i})\, dQ \\ (\boldsymbol{W}_{\alpha}, \boldsymbol{V}_{\alpha})_{\Pi_n} &= \int_{\Pi_n} (\boldsymbol{W}_{\alpha} \cdot \boldsymbol{V}_{\alpha}) n_i\, d\Pi\end{aligned}$$

$$(\boldsymbol{W}_{\alpha}, \boldsymbol{V}_{\alpha})_{Q_n^{\Sigma}} = \sum_{e=1}^{(n_{el})_n} \int_{Q_n^e} (\boldsymbol{W}_{\alpha} \cdot \boldsymbol{V}_{\alpha})\, dQ. \tag{40}$$

In the above equations, $(n_{el})_n$ denotes the number of space-time finite elements at time level n, $Q_n^e = \Omega_n^e \times I_n$ denotes the domain of element interior, and n_i denotes the unit outward normal. Note that the operators defined in (40) are symmetric, i.e., $(\boldsymbol{W}_{\alpha}, \boldsymbol{V}_{\alpha})_{\Omega} = (\boldsymbol{V}_{\alpha}, \boldsymbol{W}_{\alpha})_{\Omega}$.

Weak Form: The weak form is stated as follows. Within each $Q_n, n = 0, \cdots, N-1$, find $\boldsymbol{V}_{\alpha} \in \mathcal{S}_n$ such that for all $\boldsymbol{W}_{\alpha} \in \vartheta_n$ the following variational equation is satisfied

$$\boldsymbol{B}(\boldsymbol{W}_{\alpha}, \boldsymbol{V}_{\alpha})_{\alpha n} = \boldsymbol{L}(\boldsymbol{W}_{\alpha})_{\alpha n} \tag{41}$$

where

$$\begin{aligned}\boldsymbol{B}(\boldsymbol{W}_{\alpha}, \boldsymbol{V}_{\alpha})_{\alpha n} &= ((-\boldsymbol{W}_{\alpha,t}), \boldsymbol{U}_{\alpha}(\boldsymbol{V}_{\alpha}))_{Q_n} \\ &\quad - ((\boldsymbol{W}_{\alpha,i}), \boldsymbol{F}^{c}_{\alpha i}(\boldsymbol{V}_{\alpha}))_{Q_n} + a(\boldsymbol{W}_{\alpha}, \boldsymbol{V}_{\alpha})_{Q_n} \\ &\quad - (\boldsymbol{W}_{\alpha}, \boldsymbol{F}_{\alpha}(\boldsymbol{V}_{\alpha}))_{Q_n} \\ &\quad + (\boldsymbol{W}_{\alpha}(t^-_{n+1}), \boldsymbol{U}_{\alpha}(\boldsymbol{V}_{\alpha}(t^-_{n+1})))_{\Omega} \\ &\quad + (\boldsymbol{W}_{\alpha}, \boldsymbol{F}^{c}_{\alpha i}(\boldsymbol{V}_{\alpha}) - \boldsymbol{F}^{h}_{\alpha i}(\boldsymbol{V}_{\alpha}))_{\Pi_n} \\ \boldsymbol{L}(\boldsymbol{W}_{\alpha})_{\alpha n} &= (\boldsymbol{W}_{\alpha}(t^+_n), \boldsymbol{U}_{\alpha}(\boldsymbol{V}_{\alpha}(t^+_n)))_{\Omega}.\end{aligned} \tag{42}$$

Equation (42) is obtained by multiplying (36) with the weighting function and performing integration by parts. It can be observed that the operator $\boldsymbol{B}$ in (41) is nonsymmetric.

Time-Discontinuous Galerkin/Least-Squares Formulation: The space-time Galerkin/least-squares finite element formulation for the αth carrier hydrodynamic transport equations is stated as follows.

Within each $Q_n, 0 = 0, \cdots, N-1$, find $\boldsymbol{V}^h_{\alpha} \in \mathcal{S}^h_n$ such that for all $\boldsymbol{W}^h_{\alpha} \in \vartheta^h_n$ the following variational equation is satisfied:

$$\boldsymbol{B}_{\mathrm{GLS}}(\boldsymbol{W}^h_{\alpha}, \boldsymbol{V}^h_{\alpha})_{\alpha n} = \boldsymbol{L}_{\mathrm{GLS}}(\boldsymbol{W}^h_{\alpha})_{\alpha n} \tag{43}$$

where

$$\begin{aligned}\boldsymbol{B}_{\mathrm{GLS}}(\boldsymbol{W}^h_{\alpha}, \boldsymbol{V}^h_{\alpha})_{\alpha n} &= \boldsymbol{B}(\boldsymbol{W}^h_{\alpha}, \boldsymbol{V}^h_{\alpha})_{\alpha n} \\ &\quad + (\mathcal{L}_{\alpha}\boldsymbol{W}^h_{\alpha}, \tau_{\mathrm{GLS}\alpha}\mathcal{L}_{\alpha}\boldsymbol{V}^h_{\alpha})_{Q_n^{\Sigma}} \\ \boldsymbol{L}_{\mathrm{GLS}}(\boldsymbol{W}^h_{\alpha})_{\alpha n} &= (\boldsymbol{W}^h_{\alpha}(t^+_n), \boldsymbol{U}_{\alpha}(\boldsymbol{V}^h_{\alpha}(t^-_n)))_{\Omega}.\end{aligned} \tag{44}$$

A jump condition term of the following form

$$\int_{\Omega} \boldsymbol{W}^h_{\alpha}(t^+_n) \cdot \lceil\lfloor \boldsymbol{U}_{\alpha}(\boldsymbol{V}^h_{\alpha}(t_n)) \rfloor\rceil\, d\Omega \tag{45}$$

is added to the variational equation to enforce weak initial conditions for each space-time slab. The term

$$\lceil\lfloor \boldsymbol{U}_{\alpha}(t_n) \rfloor\rceil = \boldsymbol{U}_{\alpha}(T^+_n) - \boldsymbol{U}_{\alpha}(t^-_n) \tag{46}$$

denotes the jump in time of $\boldsymbol{U}_{\alpha}$ in the time slab.

The stability emanates from the addition of a least-squares term to the Galerkin formulation

$$\sum_{e=1}^{(n_{el})_n} \int_{Q_n^e} (\mathcal{L}_{\alpha}\boldsymbol{W}^h_{\alpha}) \cdot \tau_{\mathrm{GLS}\alpha}(\mathcal{L}_{\alpha}\boldsymbol{V}^h_{\alpha})\, dQ. \tag{47}$$

The least-squares term is proportional to the residual and therefore only contributes to regions where the Galerkin method

fails to resolve the transport of carriers. The governing differential operator $\mathcal{L}_\alpha$ is given by

$$\mathcal{L}_\alpha = \tilde{A}_{\alpha 0}\frac{\partial}{\partial t} + \tilde{A}_{\alpha i}\frac{\partial}{\partial x_i} - \frac{\partial}{\partial x_i}\left(\tilde{K}_{\alpha ij}\frac{\partial}{\partial x_j}\right) + \tilde{C}_\alpha \quad (48)$$

where $\tilde{C}_\alpha$ is a nonunique operator and is defined as

$$F_\alpha = -\tilde{C}_\alpha V_\alpha. \quad (49)$$

$\tau_{\mathrm{GLS}\alpha}$ is an $n_{\mathrm{dof}} \times n_{\mathrm{dof}}$ symmetric positive-semidefinite matrix of intrinsic time scales [21], [22]. Approximating V^h and W^h in (43) by linear basis functions in space and constant functions in time, a nonlinear system of equations can be obtained. The nonlinear system is linearized and a predictor-multicorrector algorithm is employed to find the solution at each time slab [22].

IV. Distributed Memory Parallel Computers

The finite element program for the hydrodynamic transport model has been implemented on Intel's iPSC/860, Touchstone Delta, and IBM's SP-1 computer which are MIMD (Multiple-Instruction–Multiple-Data) machines with distributed memory units. The iPSC/860 (also referred to as i860) system consists of a large number of processors or nodes interconnected with a hypercube topology. An n-dimensional cube has n communication channels and 2^n processors. Each processor has its own local memory of 16 MB. A node's memory is distinct from that of other nodes and the host. Each node runs the NX operating system and uses message passing to communicate with other nodes. The latency (the start-up overhead associated in sending a message from one processor to another) on this machine is about 210 μs for a typical message of length greater than 100 bytes and the bandwidth (the communication rate among interprocessors) 20 MB/s. While the iPSC/860 has a hypercube topology, the Touchstone Delta is a 2-D mesh. This alternative design has more wires among the neighboring connected processors, but less communication channels per node than the iPSC/860. The latency on this machine is about 167 μs and the bandwidth 56 MB/s. For both the i860 and the Delta machines, alongside with the nodes are I/O nodes, peripheral units and a front-end host. Since the memory available on each node is small, preprocessing front-end programs, if any, are executed on a separate workstation when executing parallel programs on the Intel machines. The results presented in this study are obtained on a 32 node i860 and 512 node Delta computers.

The 9076 SP-1 system provides flexible serial and parallel computing and is the first in a family of serial and parallel systems from IBM. Each processor is a RISC system/6000 with 128 MB of memory. The processors can be connected through an Ethernet or an optional high-speed switch and the high-speed switch is utilized in this paper. The structure of the high-speed switch is that of a multistage crossbar, which is also referred to as an Ω-switch. Optional high-speed switch provides higher bandwidth and is utilized for interprocessor communication. The latency on this machine is about 80 μs and the bandwidth 64 MB/s. A 16 processor system, of which eight are configured as interactive nodes and the remaining eight as batch nodes, is employed in this paper. The front-end programs, if any, can be executed on one of the nodes, designated as a host node, of SP-1 since the local memory available on each node is sufficient to hold a large scale program.

4.1. Communication Libraries

In distributed memory computers, data in the local memory of a processor are available to other processors through data communication. Data communication among the processors is managed by the user program. The user program typically utilizes the message-passing communication libraries available on the parallel machine. The communication library available on the Intel supercomputers (iPSC/860, Delta) is NX while MPL (Message Passing Library) is the communication library available on IBM SP-1/2 supercomputers. The primary communication functionalities utilized in both NX and MPL are point-to-point and collective communication libraries. To ensure portability of code to different distributed memory computers, advanced functionalities like communication among groups or context based communication provided by MPL are not used in our implementation on the SP-1. Point-to-point and collective communication libraries are the core of message passing and are supported on all distributed memory computers. In a point-to-point communication, a message is sent from one processor to another processor. A collective communication is performed among all processors and is utilized to perform broadcast, synchronization, or global operations. The performance of point-to-point and collective communication libraries can be measured by useful/important parameters such as latency, bandwidth, and overhead. The performance can be optimized by overlapping communication with computation, aligning data, and avoiding buffering. A few of the important communication functions that are employed in the parallel implementation are shown in Table I. More details on Intel communication libraries can be found in [23] and the details on IBM SP-1 can be found in [24].

4.2. A Manager-Worker Model for Data Communication

The SPMD programming model for a PDE solver requires that the domain or mesh is first decomposed into subdomains and each subdomain be assigned to a processor. Each processor executes a copy of the serial program with the changes made to the code primarily to obtain global (or correct) values for the mesh nodes (also termed as separator nodes) lying on the subdomain interface. A manager-worker model is designed for data communication among processors specifically to obtain global values assembled from the subdomains.

In a manager-worker model, each separator node on the interprocessor boundary is classified as either a manager or a worker to all the processors to which the node belongs. Only one processor will be identified as a manager to a separator node and all other processors will be identified as workers (see Fig. 1 for an example). The distinction of the processor as a manager or as a worker for a separator node is made for two important reasons.

TABLE I
COMMUNICATION LIBRARIES ON INTEL AND IBM SUPERCOMPUTERS

iPSC/860, Delta	SP-1	Functionality
csend *(type, buf, len, node, pid)*	***mpc_bsend*** *(outmsg, msglen, dest, type)*	*sends a message from one processor to another; waits until the application buffer in the sending processor is free for reuse*
crecv *(typesel, buf, len)*	***mpc_brecv*** *(inmsg, msglen, source, type, nbytes)*	*receive a message and wait for completion*
numnodes *()*	***mpc_environ*** *(numtask, taskid)*	*obtain number of nodes allocated in the parallel machine*
gdsum *(x, n, work)*	***mpc_reduce*** *(outmsg, inmsg, msglen, dest, func, gid)* *func = d_vadd*	*global vector double precision sum operation*
gisum *(x, n, work)*	***mpc_reduce*** *(outmsg, inmsg, msglen, dest, func, gid)* *func = i_vadd*	*global vector integer sum operation*
mynode *()*	***mpc_environ*** *(numtask, taskid)*	*node ID of calling process*
gsync *()*	***mpc_sync*** *(gid)*	*global synchronization operation*

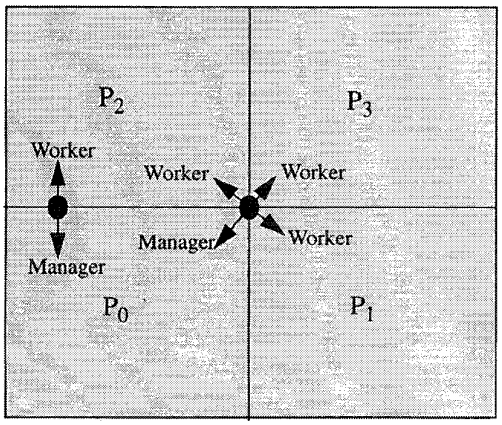


Fig. 1. Illustration of manager-worker assignment for an s_i shared by four processors.

- A manager processor P_i for a separator node s_i is the only processor responsible for s_i.
- Even though the manager processor P_i is the only processor responsible for s_i, P_i informs all other worker processors P_j (j different from i) that contain s_i when a globalization operation is performed.

The data structures necessary to identify the separator nodes and to classify the list of processors sharing the separator nodes as manager and worker processors are discussed in Section VI-2.

V. PROGRAM ORGANIZATIONS

In order to properly implement a parallel software, it is of utmost importance to understand the organization and the basic data flow of the serial application program. With a good understanding of the program structure, efficient parallel algorithms can be designed. Furthermore, communication requirements and synchronization steps can be identified and often minimized. More significantly, parallel programs can be developed with minimal changes to the serial code. The abstractions of the serial and parallel finite element programs are discussed in this section.

5.1. Serial Finite Element Program Structure

A typical organization of a serial nonlinear finite element program is depicted in Fig. 2 and contains five modules: a preprocessor, an element library, determination of the local and global residual vector and tangential matrix, a linear equation solver, and a postprocessor. The preprocessor performs the task of reading and generating the input data needed for the finite element analysis. An element library provides the flexibility to perform multidimensional (one, two, or three dimensions) finite element analysis of partial differential equations (Poisson, electron hydrodynamic, hole hydrodynamic, or lattice equations). The element generation process also depends on the solution method such as direct, iterative, or matrix-free iterative methods and these are optional. Based on the strategy chosen in the element library, computations are performed to determine the local residual vectors and tangential matrices of the finite elements. The local contributions are then assembled to obtain the global residual vector and the tangential matrix of the system. Note that the global tangential matrix is not formed if an iterative or a matrix-free iterative method is chosen. A linear equation solver is then employed to solve the linearized system. The postprocessor module

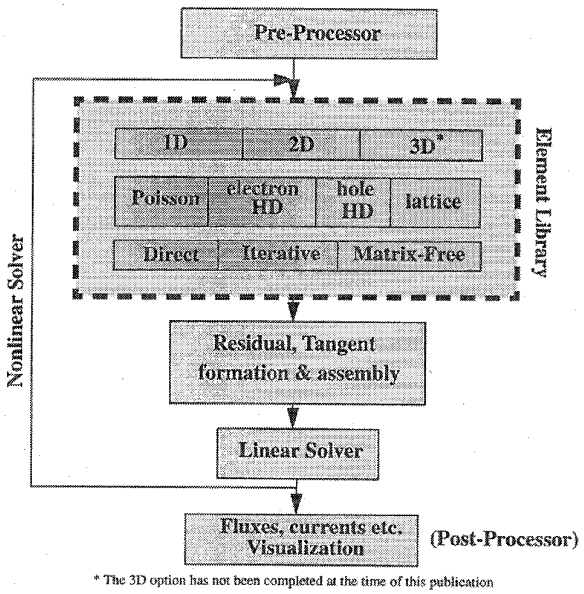


Fig. 2. Serial finite element program organization.

performs the calculation of fluxes, currents, and visualization of data.

5.2. Parallel Finite Element Program Organization

Ideally, all five modules of the serial program depicted in Fig. 2 should be parallelized. Computationally, the most intensive step in the solution of partial differential equations is the finite element analysis (element library, residual, tangent formation and assembly, and linear solver) while preprocessing and postprocessing form only a small fraction of the total CPU time. This paper focuses on the parallel numerical computations for the finite element analysis. In addition, a number of steps in the postprocessor (such as calculation of fluxes, currents) are performed trivially in parallel.

The preprocessor performs several tasks, and one of the tasks is to prepare the input data required for each parallel processor. In the case of the i860 and the Delta machines, the input data are prepared on a host workstation and sent to each processor of the parallel machine through TCP/IP [25]. For the IBM SP-1, the data are prepared on a designated host node (typically node 0) and sent to other processors using message passing communication routines. The parallel program organization is shown in Fig. 3 and note that the direct solver option is currently not implemented.

VI. Preprocessor

In a serial program for the device simulator, a preprocessor performs the following tasks:

- generating a mesh for the domain (Ω) on which the partial differential equation needs to be solved;
- specifying the boundary conditions for the contacts and artificial boundaries;
- specifying the initial conditions, doping profiles;
- reading the numerical solution parameters, such as quadrature rules, the type of linear solver to be employed, and convergence tolerances for the stopping criteria.

In addition to the above tasks, a preprocessor in a parallel program is required to:

- partition the original domain Ω into N subdomains, $\Omega_i, i = 1, \cdots, N$, where N is typically equal to the number of processors to be employed;
- prepare the required input data for each parallel processor;
- create additional data structures necessary for data communication among processors;
- transfer the data to each of the parallel processors.

The additional preprocessing tasks required by the parallel finite element program are briefly discussed in this section.

6.1. Domain Decomposition

Domain decomposition is one of the most important steps in designing parallel numerical software for solving partial differential equations. Much of the coarse grain parallelism is achieved by considering a decomposition of the original domain (or grid) into subdomains. Ideally, all the subdomains should contain approximately the same number of mesh nodes and elements to ensure a good load balance. In addition, the number of mesh nodes lying on the interface between the subdomains should be kept small to ensure minimum communication. Good load balance and low communication overhead are both equally important to achieve good performance. Neither a good load balance algorithm with high communication costs nor a poor load balance algorithm with very little communication costs could give good performance.

Several domain decomposition algorithms are available in the literature. Generally speaking, the domain decomposition algorithms can be grouped into three categories: engineering [26], optimization [27], and graph theoretic based algorithms [28]. The domain decomposition algorithm employed in this paper is based on the graph representation of a finite element mesh and was originally proposed in [28]. This graph theoretic based domain decomposition algorithm consists of three basic steps.

Step 1: For a given mesh of n vertices, an $n \times n$ Laplacian matrix $Q_{n\times n}$ can be constructed by knowing the adjacency information. The adjacency structure can be constructed as follows:

i) for each finite element, connect each vertex with all other vertices of the element;
ii) the finite element graph is constructed by the union of the edges of the finite elements in the mesh.

The Laplacian matrix Q is then given by

$$q_{ij} = \begin{cases} -1 & \text{for an edge connection between } i \text{ and } j \\ 0 & \text{for no edge connection between } i \text{ and } j \\ \deg(i) & \text{for } i = j \end{cases}$$

where $\deg(i)$ denotes the degree of vertex i and is defined as the number of vertices connected to vertex i. It can be observed that Q is a singular M-matrix.

Step 2: Compute the eigenvalues of Q and order them as $\lambda_1 = 0 \leq \lambda_2 \leq \cdots \leq \lambda_n$.

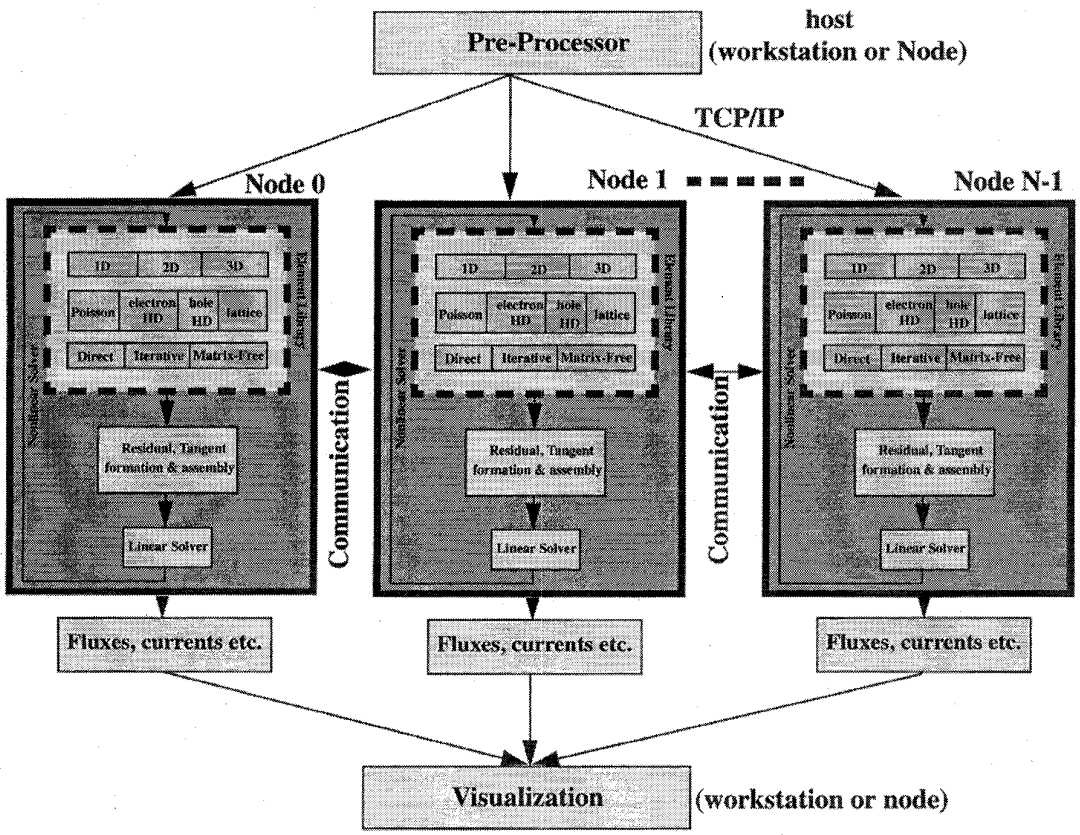


Fig. 3. Parallel finite element program organization on i860/Delta/SP-1.

Compute the second eigenvector corresponding to the smallest positive eigenvalue.

Step 3: The components of the eigenvector computed in Step 2 are used to partition the vertices into subsets. All vertices with eigenvector components above the median value are put in one subset and the rest are put in a second subset. A vertex separator is computed by employing a matching algorithm.

The above three steps are repeated until the number of subsets obtained matches the desired number of subdomains.

Remarks:

1) The expensive step in the above algorithm is to compute the second eigenvector corresponding to the smallest positive eigenvalue.
2) An analysis of the Laplacian eigenvectors of graphs shows that the second eigenvector tends to partition the graph at its weakest-link.
3) Instead of partitioning the eigenvector using the median value (as was done in Step 3), one can partition by using the sign of the eigenvector components. The key idea, however, is to obtain subsets that are approximately equal in size.

6.2. Data Structures for the Parallel Program

Once the domain decomposition is complete, additional data structures need to be defined for the parallel program. The additional data structures are necessary to map the subdomains to the original domain, to store the separator nodes, the list of processors sharing the separator nodes, and the manager and worker processor necessary for the manager-worker data communication model discussed in Section IV-2. Denote n_v^p and n_e^p to be the number of mesh nodes and mesh elements, respectively, in subdomain Ω_p. The following two array structures are defined to map the subdomain and global domain information.

- $local\_to\_global\_nodes[1 \cdots n_v^p]$: This array structure maps the local node numbering in the subdomain to the global node numbering and is available on the processor to which the subdomain Ω_p is assigned.
- $local\_to\_global\_elements[1 \cdots n_e^p]$: This array structure maps the local element number in the subdomain to the global element numbering and is available on the processor to which the subdomain Ω_p is assigned.

In addition, the following array structures are used to identify the separators to facilitate the data communication among the processors.

- $mesh\_node\_owner[1 \cdots n_v^p]$: This array structure identifies the manager processor of each local node in the subdomain. For mesh nodes that are interior in the subdomain, the manager processor is the processor to which the subdomain is assigned. For those mesh nodes that are

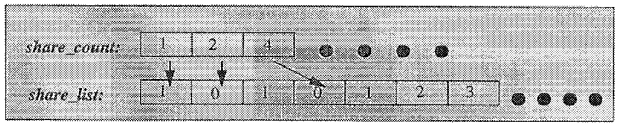


Fig. 4. Definitions of $share\_list$ and $share\_count$ data structures.

on the boundary, the manager processor could be any of the processors sharing the node.

- $share\_count[1 \cdots n_v^p]$: This array structure, in conjunction with the array $share\_list$, provides the information about the interactions with other processors for each shared mesh node. The data are stored using a quotient list structure [29]. For each mesh node, $share\_count$ points to the starting location in the $share\_list$ of that node's list of involved processors (Fig. 4). The number of processors sharing mesh node i is given by $share\_count[i+1] - share\_count[i]$.
- $share\_list[1 \cdots share\_count[n_v^p+1]]$: This data structure (Fig. 4) contains a set of lists, one for each mesh node, of the processors sharing a mesh node. It is indexed by $share\_count$.

Since all array structures discussed above are distributively assigned to the processors, the actual storage for these extra data structures is kept to the minimum. The additional storage requirements are on the order of O(*Nnode*), where *Nnode* is the total number of mesh nodes in the original mesh. Furthermore, these additional data structures can be constructed in a trivial manner based on the domain decomposition information.

6.3. Partitioning Global Input Data

The serial program data structures and the input data are distributively stored as each processor requires data only for its subdomain. The data structures and the input data required by the program on the parallel processor are made available by the preprocessor.

Two approaches can be used to assign data to each processor. In the first approach, the subdomain data can be written to separate files and each processor would read its data by opening the respective file. This approach is not efficient as the number of files can be quite large and the I/O on most parallel machines (comparing with the CPU in particular) is slow. In the second approach, the preprocessor would perform the task of collecting all the data for each subdomain and sending it to the parallel processor. In this approach, one has to pay attention to the additional storage required in the host processor. The second approach is adopted in this paper, and the additional storage required is estimated to be that required for one subdomain.

The data for the original domain are read into a common array structure (Fig. 5). The data are referenced by the starting memory location identified with each array structure. The domain decomposition information and the subdomain data structures discussed in the previous section are utilized to prepare the input data for each subdomain. For each mesh node belonging to the subdomain, the input values (coordinates, initial values, doping, and so on) are extracted from the

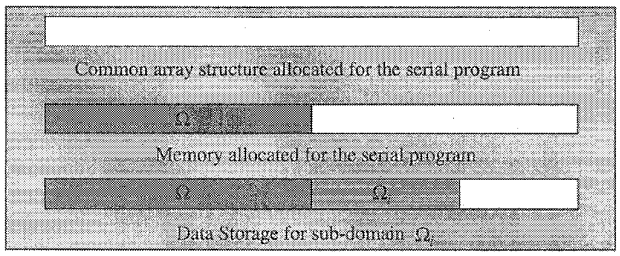


Fig. 5. Data storage strategies using common array structures.

```
for each mesh node belonging to Ω_i
    find the global node corresponding to the local node
    (use local_to_global_nodes array)
    get the coordinates for the global node
    store the coordinates in a new local coordinates array
end for
```

Fig. 6. Pseudocode for extracting coordinates for Ω_i.

common array structure stored in region Ω and moved into a separate location identified with Ω_i. An example is shown in Fig. 6 to extract the coordinates for a subdomain. Similar procedures are used to extract the remaining data.

The data stored in region Ω_i belongs to subdomain i and is transferred to the parallel processor. As the data for each subdomain is prepared, it is sent to the corresponding parallel processor and the same storage location is used for the next subdomain. Since the procedure of extracting data for each subdomain is performed by a preprocessor, the operation is executed in a serial manner.

6.4. Data Transfer

The data prepared for each subdomain are transferred to the parallel processor using either TCP/IP and/or message passing communication libraries. If the preprocessor is run on a workstation (as in the case of iPSC/860 and Delta) the data are transferred to node "0" of the parallel machine and node "0" redirects the data to appropriate processor using message-passing libraries. In the case of SP-1, for which the preprocessor is run on a designated host node, the data are directly sent from the host node using the message-passing communication libraries.

The issues to be considered while employing the TCP/IP communication links are summarized as follows:

- TCP/IP provides a general environment, but the data transfer rate is slow.
- TCP/IP may introduce another layer of complexity in order to accommodate the different machine-word byte orderings of the Intel hypercube and the host, such as a SUN workstation.
- It is typically difficult to maintain a separate TCP/IP connection from each processor to the host workstation. Hence, a TCP/IP connection is provided only to node '0' and an extra communication is needed to transmit the data from node '0' to the processor using the communication

protocols provided by the parallel machines, which are generally faster than the TCP/IP connection.

The above issues do not apply on the SP-1 since TCP/IP connection is not used.

VII. Parallel Finite Element Analysis

There are two main computationally intensive steps in the finite element analysis program, they are:

- the formation of the left-hand side (LHS) tangential matrix (or the Jacobian) A and the right-hand side (RHS) (or residual) vector b;
- the solution of the linear system of equations $Ax = b$.

In a parallel finite element program, the LHS matrix A_i and the RHS vector b_i can be formed for each subdomain, $\Omega_i, 1 \leq i \leq N$, with no interprocessor communication. This is a perfectly parallelizable step and provides very high efficiencies. In the solution of linear equations, interprocessor communication is needed for data exchange. For the iterative solvers employed in this work, interprocessor communication is primarily restricted to computing dot-products (vector-vector products) and matrix-vector products. By employing efficient data communication models, such as the manager-worker model discussed in Section IV-2, the communication overhead can be minimized and good overall efficiencies can be obtained. In this section, the solvers employed for Poisson, electron and hole hydrodynamic, and lattice equations are discussed followed by a discussion on the steps involved in computing dot and matrix-vector products.

7.1. Iterative Solvers

Finite element discretization of the Poisson and the lattice equations leads to symmetric systems of equations while that of the electron and hole hydrodynamic equations gives rise to nonsymmetric systems of equations. In this paper, conjugate gradient algorithm [30] with a diagonal preconditioner is employed to solve the symmetric systems of equations and a generalized minimal residual solver (GMRES) [31] with a block diagonal preconditioner is employed to solve the nonsymmetric systems of equations. Note, however, that no preconditioner is employed for the matrix-free approach.

The conjugate gradient and the GMRES algorithms as employed in the serial program are shown, respectively, in Figs. 24 and 25 of the Appendix. In the conjugate gradient algorithm, first the initial residual and the search direction vectors are formed. The conjugate gradient iterations are then performed until the solution to the system of equations is obtained to a given tolerance. The GMRES algorithm is based on the modified Gram–Schmidt procedure, and uses restarts to control storage requirements. The restart value is identified by the variable k in the GMRES iteration loop (see Fig. 25). Parallel conjugate gradient algorithms have been studied extensively in literature, for example see [32] and [33]. The parallel versions of the conjugate gradient and GMRES iterative methods are shown in Figs. 7 and 8, respectively. The basic operations that require communication are the matrix-vector and vector-vector products. The communication operation in a vector-vector product involves summing a scalar variable across all the processors and a matrix-vector product involves globalizing the resultant vector. A globalization operation assembles the contributions from neighboring processors to obtain global (or complete) values for all the separator nodes.

7.2. Parallel Dot and Matrix-Vector Computations

The two computations that require interprocessor communication are the dot and matrix–vector products. In a dot product, involving two global vectors, the global vectors are distributed across all the processors and care must be exercised in accounting for the separator nodes in the summation. In a matrix–vector product, techniques are needed to obtain global values for the separator nodes.

Parallel Dot Product—Computation of $u \cdot v$: Global vectors $\boldsymbol{u}, \boldsymbol{v}$ of size *Nnode* are distributed across N processors such that $\boldsymbol{u} = \cup_{i=1}^{N} \boldsymbol{u}_i$ and $\boldsymbol{v} = \cup_{i=1}^{N} \boldsymbol{v}_i$, where $\boldsymbol{u}_i, \boldsymbol{v}_i$ are vectors available in the ith processor. A straight forward computation of $\alpha = \sum_{i=1}^{N} \boldsymbol{u}_i \cdot \boldsymbol{v}_i$ gives an incorrect answer because the values for the separator nodes have been included more than once in the summation. A solution to this problem is to define a weight associated with each mesh node in the subdomain. The dot product within each processor $(\boldsymbol{u}_i \cdot \boldsymbol{v}_i)$ is then computed by the expression

$$\alpha_i = \sum_k w^k u_i^k v_i^k$$

where w^k is the weight associated with each mesh node given by

$$w^k = \begin{cases} 1 & \text{if a mesh node is classified as an interior to the subdomain} \\ 1 & \text{if a mesh node is on the separator and the processor to which the subdomain is assigned is a manager to the mesh node} \\ 0 & \text{if a mesh node is on the separator and the processor to which the subdomain is assigned is a worker to the mesh node} \end{cases} \tag{50}$$

The computation of $(\boldsymbol{u} \cdot \boldsymbol{v})$ is performed by summing α_i in all the processors, i.e.,

$$\alpha = \sum_{i=1}^{N} \alpha_i.$$

It can be observed that the separator nodes are accounted for only once in the summation.

Parallel Matrix–Vector Product: Computation of $A \cdot u$: The matrix $\boldsymbol{A}$ and the vector $\boldsymbol{u}$ are distributively stored across N processors such that each processor contains information about $\boldsymbol{A}_i$ and $\boldsymbol{u}_i$ for subdomain Ω_i. The vector $\boldsymbol{u}_i$ is global, i.e., the values for the separator nodes have been updated through interprocessor communication. However, the matrix $\boldsymbol{A}_i$ is not updated and takes into account only the information from subdomain Ω_i for the separator nodes. The tangent matrix $\boldsymbol{A}_i$ is stored on an element by element basis and

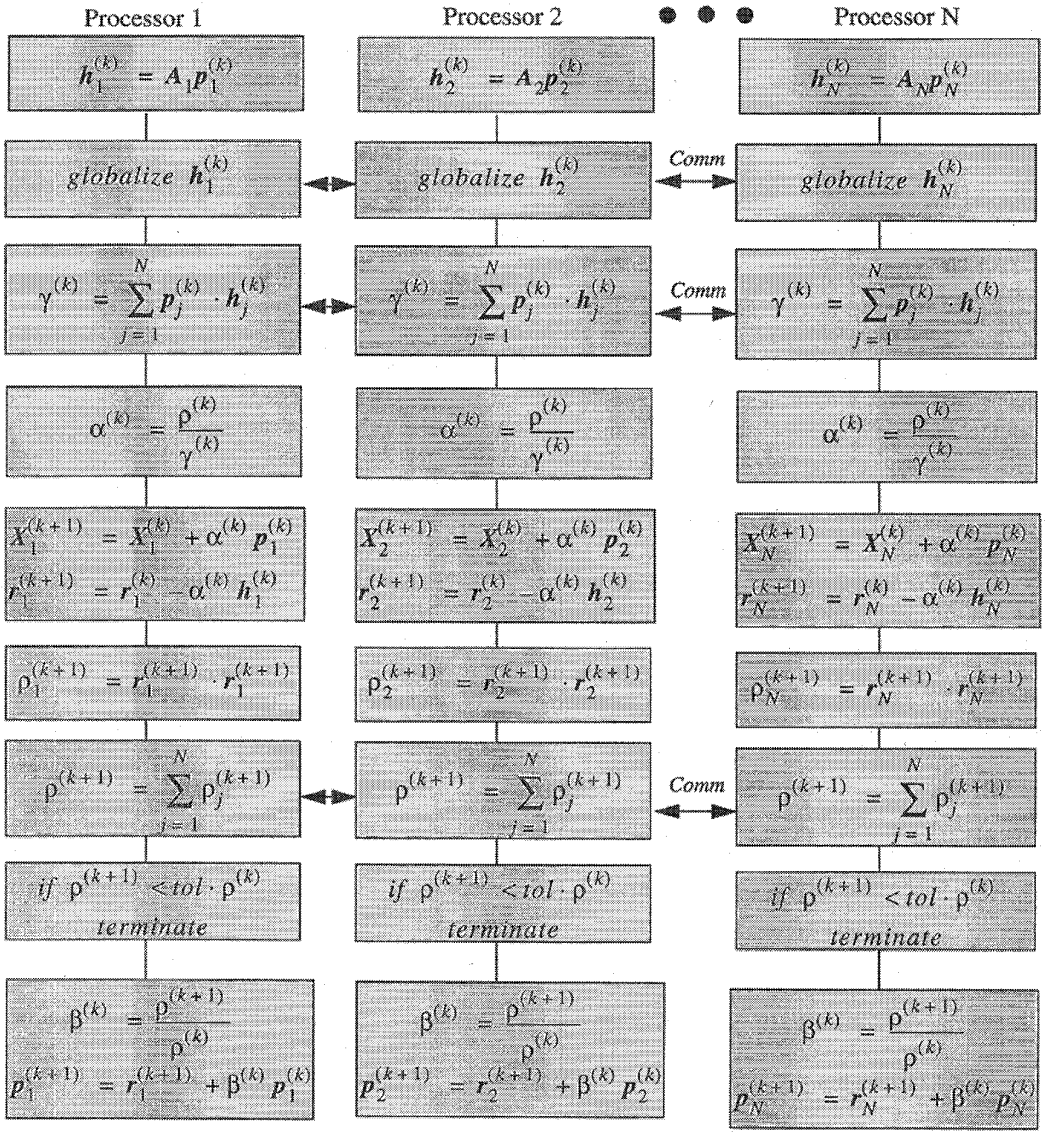


Fig. 7. Inner loop of the parallel conjugate gradient algorithm.

is never assembled. The tangent matrix is formed for each element ($\boldsymbol{A}_i^e$) in the subdomain and $\boldsymbol{A}_i$ is the set of all $\boldsymbol{A}_i^e$, i.e., $\boldsymbol{A}_i = \{\boldsymbol{A}_i^1, \boldsymbol{A}_i^2, \cdots, \boldsymbol{A}_i^{\text{nel}}\}$, where nel is the number of elements in subdomain Ω_i. See Fig. 9 for an illustration.

The matrix vector product is performed locally on each element and the result is added to the corresponding global node number. The pseudocode for computing matrix-vector product in subdomain Ω_i is shown in Fig. 10. In this approach, the separator nodes belonging to subdomain Ω_i take into account only the contributions from the elements belonging to subdomain Ω_i. The procedure to obtain the global values for the separator nodes consists of four steps:

1) If a processor k is classified as a worker for a separator node m, then send the matrix-vector product of node m computed in processor k to node m in the manager processor l.
2) If a processor l is classified as a manager for node m, then receive data from all processors $j(j \neq l)$ that contain node m, and add the data to the value of node m stored in processor l.
3) If a processor l is classified as a manager for node m, then send the value (global value) stored for node m in processor l to all processors $j(j \neq l)$ that contain node m.
4) If a processor k is classified as a worker for node m, then receive the data of node m from the manager processor $l(l \neq k)$. The value is stored as the global value for node m.

7.3. Matrix-Free Techniques

In solving a system of equations of the form $Ju = -R$ using iterative schemes, the tangential matrix or the Jacobian J need not be formed explicitly. A technique can be

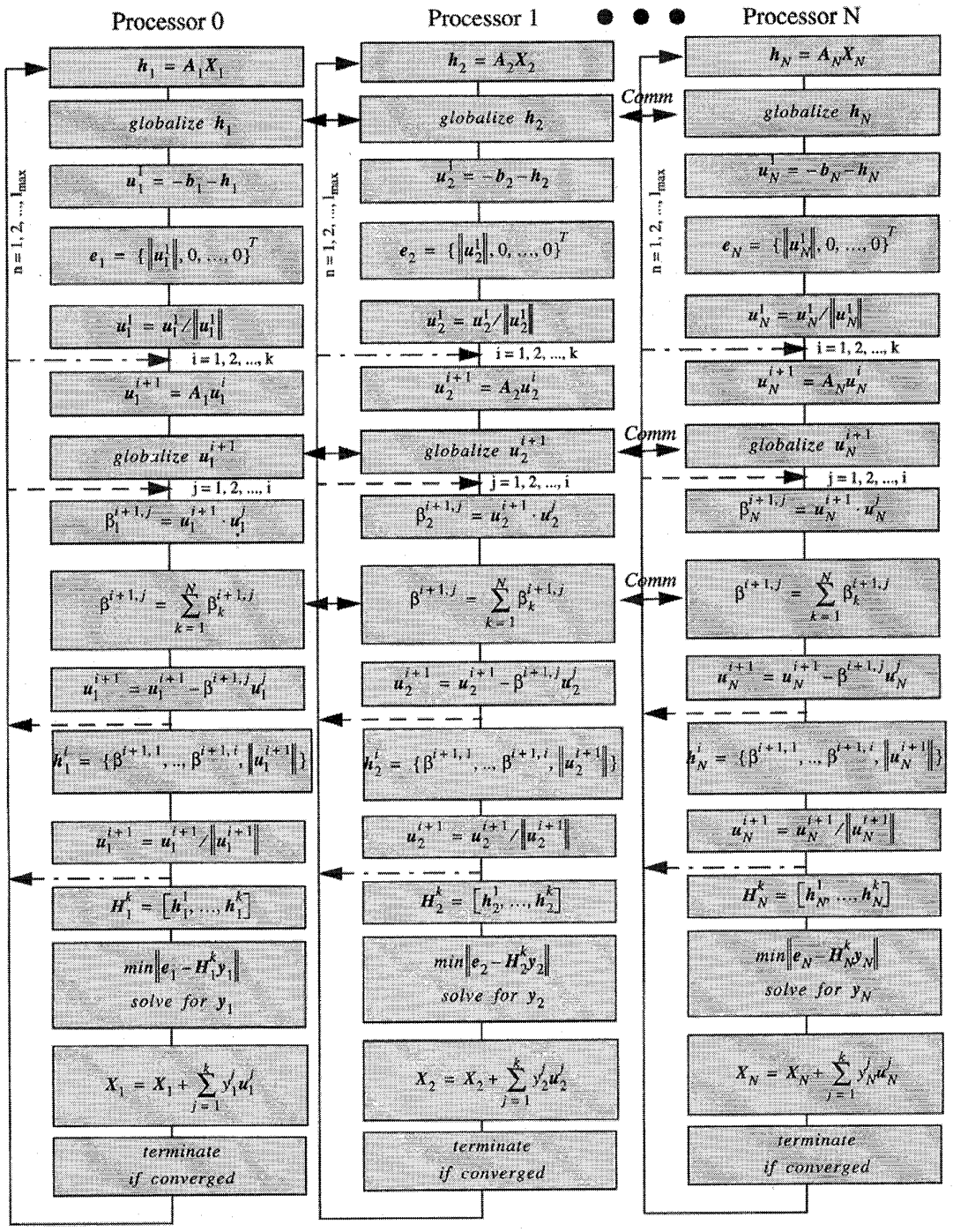


Fig. 8. Parallel GMRES algorithm.

developed for determining the product Ju without explicit formation of the Jacobian J. This technique is referred to as the matrix-free technique [34], [35]. Matrix-free techniques offer the advantage of less storage, typically faster convergence and does not require any knowledge about the components of the LHS matrix. The matrix-free technique can be especially useful for solving the hydrodynamic equations where significant saving in the memory requirements can be obtained.

Since the matrix J is a Jacobian ($J = \partial R/\partial v$) it is possible to substitute the matrix vector product with a forward difference approximation by using the residual vector R, such that

$$J(v)u = \lim_{\hat{\varepsilon} \to 0} \frac{R(v + \hat{\varepsilon}u) - R(v)}{\hat{\varepsilon}}. \tag{51}$$

A formula for evaluating the matrix-vector product $J(v)u$ is inferred from (51) where $\hat{\varepsilon}$ is taken as a small but finite value,

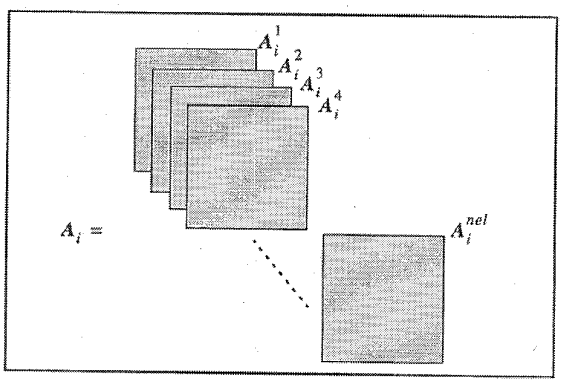


Fig. 9. Tangent matrix for subdomain i stored in an element by element form.

```
for each element j in sub-domain i
  perform  v^j = A_i^j · u_i^j
  for each node k in element j
    add  v_k^j  to corresponding global node number
  endfor
endfor
```

Fig. 10. Pseudocode for matrix-vector computation.

viz.

$$J(v)u \approx \frac{R(v+\hat{\varepsilon}u) - R(v)}{\hat{\varepsilon}}. \tag{52}$$

The choice of $\hat{\varepsilon}$ is important for accurate determination of the matrix-vector product $J(v)u$ and is discussed in greater detail in [34]. An optimal value for $\hat{\varepsilon}$ can be computed through an optimization problem as

$$\hat{\varepsilon}_{\text{opt}} = 2\left(\frac{\hat{\varepsilon}_A}{\|DJ[u]u\|}\right)^{1/2} \tag{53}$$

where

$$\begin{aligned} DJ[u]u &= \frac{R(v+\hat{\varepsilon}_{SD}u) - 2R(v) + R(v-\hat{\varepsilon}_{SD}u)}{\hat{\varepsilon}_{SD}^2} \\ \hat{\varepsilon}_{SD} &= \hat{\varepsilon}_M^{1/4}/\|u\| \\ \hat{\varepsilon}_A &= \hat{\varepsilon}_M\|R\| \end{aligned} \tag{54}$$

and $\hat{\varepsilon}_m$ is the machine precision.

VIII. Computational Results

8.1. Simulation of a Bipolar Transistor

Numerical results are shown for a 3.2 × 1.2 μm^2 bipolar transistor. The geometry of the bipolar device is shown in Fig. 11. The device consists of an n$^+$-emitter region, a p-base region and an n-collector region. The contacts are indicated by dark lines. The emitter and base contacts are 0.8 μm in length and the collector contact is 3.2 μm long. The emitter contact is placed at a distance of 0.8 μm from the top left corner, and the emitter and base contacts are separated by a distance of 0.8μm. The n$^+$ buried layer is approximately 0.4 μm in the

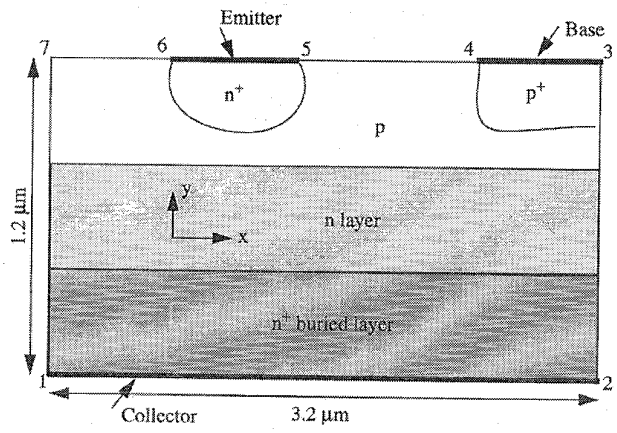


Fig. 11. Geometry of a 3.2 × 1.2 μm^2 BIPOLAR device.

y-direction. The semiconductor substrate is uniformly doped with a donor concentration of $2.5 \times 10^{16}/\text{cm}^3$. The emitter or n$^+$-region doping is a Gaussian function with a peak value of $2 \times 10^{20}/\text{cm}^3$ and $\sigma_x = \sigma_y = 0.0174$. The base or p-region doping is a Gaussian profile with a peak value of $1.55 \times 10^{18}/\text{cm}^3$ and $\sigma_x = \sigma_y = 0.082$. The p$^+$-region doping is a Gaussian profile with a peak value of $1.5 \times 10^{19}/\text{cm}^3$ and $\sigma_x = \sigma_y = 0.16$. The n$^+$-buried layer is doped with a peak value of $5 \times 10^{18}/\text{cm}^3$ and $\sigma_x = \sigma_y = 0.202$. This example utilizes a nonuniform finite difference mesh consisting of 21 868 elements and 22 165 nodes with 155 nodes along the x-direction and 143 nodes along the y-direction. This is a much finer mesh than is required for the computational domain (the mesh is chosen finer to investigate the scalability of the parallel algorithms, discussed in Section VIII-2, on large number of processing elements). The boundary conditions used for this experiment are summarized as follows:

- for the emitter contact: $c_1 = 1.984\,750 \times 10^{20}/\text{cm}^3, u_1 = 0, T_1 = T_l = T_0$, and $\psi = \psi_b(N_d) + \psi_{\text{appl}}$;
- for the base contact: $c_2 = 1.6525 \times 10^{19}/\text{cm}^3, u_2 = 0, T_2 = T_l = T_0$, and $\psi = -\psi_b(N_A) + \psi_{\text{appl}}$;
- for the collector contact: $c_1 = 5.025 \times 10^{18}/\text{cm}^3, c_2 = 4.184 \times 10^1/\text{cm}^3, u_1 = u_2 = 0, T_1 = T_2 = T_l = T_0$, and $\psi = \psi_b(N_D) + \psi_{\text{appl}}$;
- on all other boundaries: $J_n = J_p = 0$ and $\partial\psi/\partial n = 0$

Numerical experiments are performed for an applied bias of zero volts on the emitter, 0.7 V on the base and 1.5 V on the collector. A continuation method is employed to step up the base and collector voltages starting from 0 V. The results for this example are shown in Figs. 12–17.

Figs. 12 and 13 show, respectively, the electron and hole concentrations in log scale. The electron concentration is the highest in the emitter region, decreases in the p-base region and increases again in the n-collector region. The hole concentration is the highest in the p$^+$-base region and is several orders lower in the emitter and collector regions. The electron and hole concentration profiles contain very steep gradients and as is evident from the results, the numerical methods produce extremely robust solutions. Figs. 14 and 15 show, respectively, the electron and hole temperature profiles. The electrons are heated more significantly compared

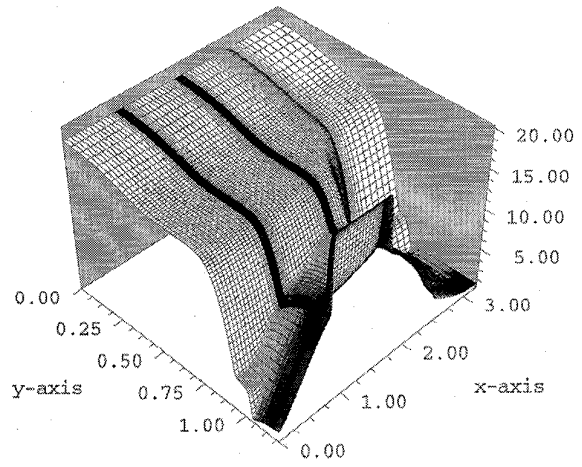


Fig. 12. Electron concentration (cm^{-3}) in steady state for a bipolar transistor.

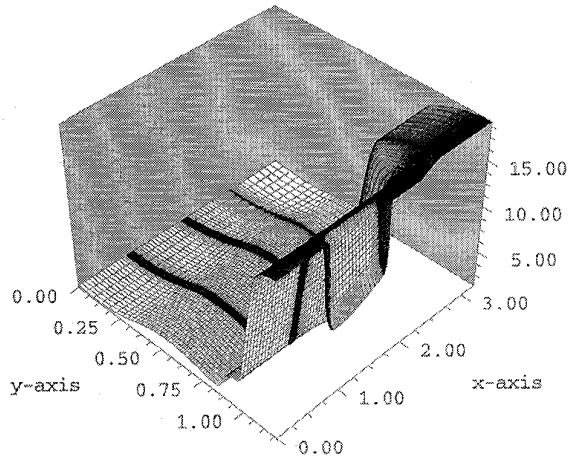


Fig. 13. Hole concentration (cm^{-3}) in steady state for a bipolar transistor.

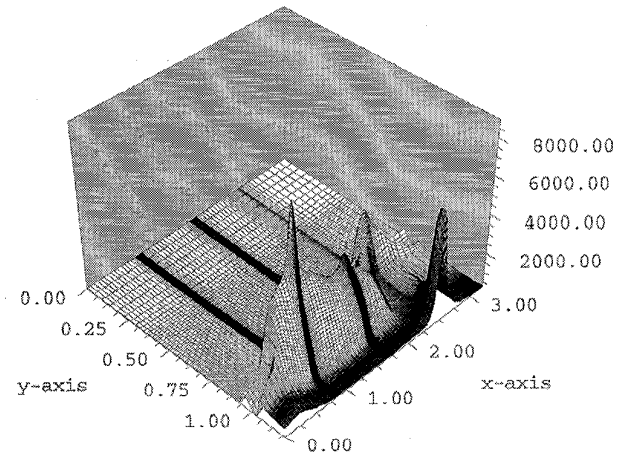


Fig. 14. Electron temperature (K) in steady state for a bipolar transistor.

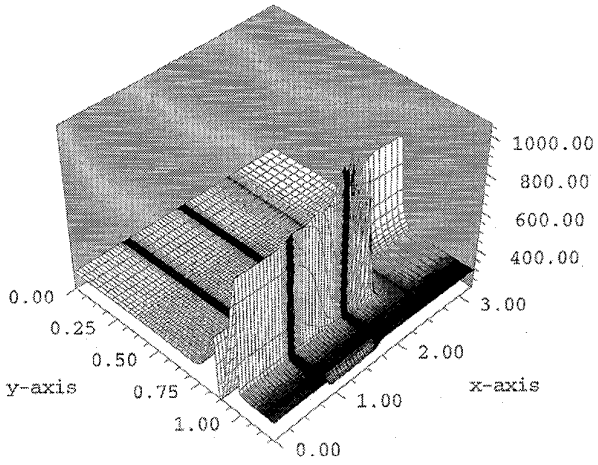


Fig. 15. Hole temperature (K) in steady state for a bipolar transistor.

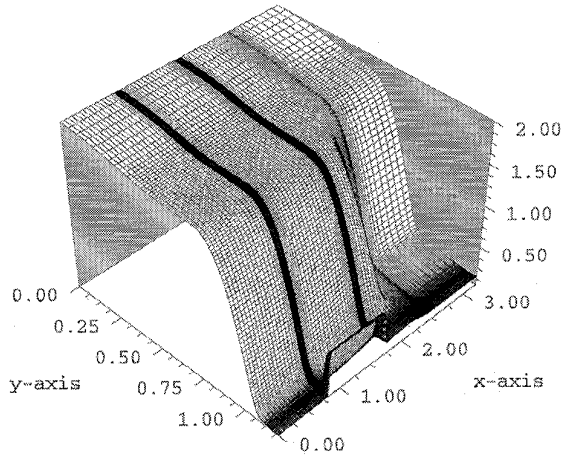


Fig. 16. Electrostatic potential (V) in steady state for a bipolar transistor.

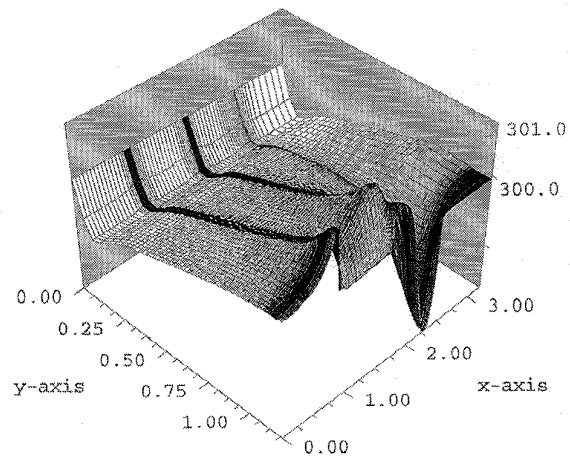


Fig. 17. Lattice temperature (K) in steady state for a bipolar transistor.

to holes and peak temperatures of about 8000 K and 1000 K are measured for the electrons and the holes, respectively. Fig. 16 shows the electrostatic potential. The profile is easily understood by the applied voltages on the base and the collector. Fig. 17 shows the lattice temperature. The lattice temperature is close to the thermal equilibrium value, and small deviations from the room temperature can be observed

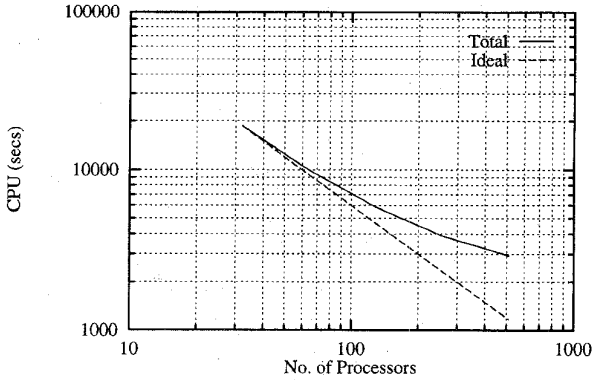


Fig. 18. Total CPU time plotted against ideal CPU time on the Touch-stone Delta machine for a bipolar transistor.

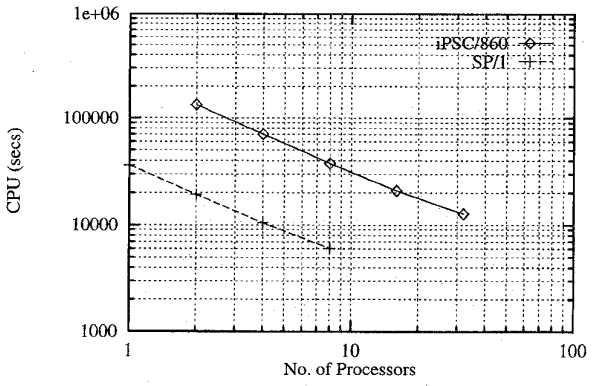


Fig. 19. CPU time comparison on iPSC/860 and SP-1 to solve 2-D *pn* silicon diode.

in the figure. Carrier velocities and electric field profiles for this bipolar transistor are shown in [20].

8.2. Parallel Performance

This section summarizes the performance of the parallel finite element program for the simulation of several devices including the bipolar transistor, pn diode and a MESFET. Performance results for several other devices can be found in [20]. All computations are done employing double precision.

Fig. 18 shows the parallel performance for the bipolar transistor example on the 512 node Touchstone Delta machine at Caltech. The plot shows the CPU time (per bias point) measured and also the ideal CPU time obtained neglecting the interprocessor communication. The difference between the two curves indicates the communication overhead. As is evident from the results, the communication overhead increases as the number of processors increases. The result, however, shows good scalability. The example requires a minimum of 32 iPSC/860 processors because of the limited memory available on each processor. During each time step approximately 250 000 linear equations are solved and a total of about 1000 time steps are employed before a steady state solution is obtained. A speedup of 60 on 64 processors, 102 on 128 processors, 154 on 256 processors, and 205 on 512 processors is obtained. On an IBM RS/6000 530H workstation, it takes approximately 40 h to obtain the results for each bias point. Using 512 nodes, the results are obtained in about 45 min. On an eight node SP-1, it takes 2.9 h to obtain steady state results. This is a powerful result and demonstrates the role parallel computers can play for practical hydrodynamic device simulations. The convergence of the parallel program for this and other examples presented in this paper is identical to the convergence rate of the serial program.

Fig. 19 shows the parallel performance characteristic for a 2D pn silicon diode, shown in Fig. 20, on iPSC/860 and SP-1. This example is simulated with 3008 mesh points. The results indicate good scalability and a reduction in CPU time as the number of processors increases. This example requires a minimum of 2 iPSC/860 processors and the speedups obtained on iPSC/860 and SP-1 are shown in Tables II and III, respectively. Compared to iPSC/860, the CPU time on the SP-1 is reduced by a factor of 6-7. The communication overhead is primarily limited to the matrix-vector and vector-vector products and is a small fraction of the total CPU time.

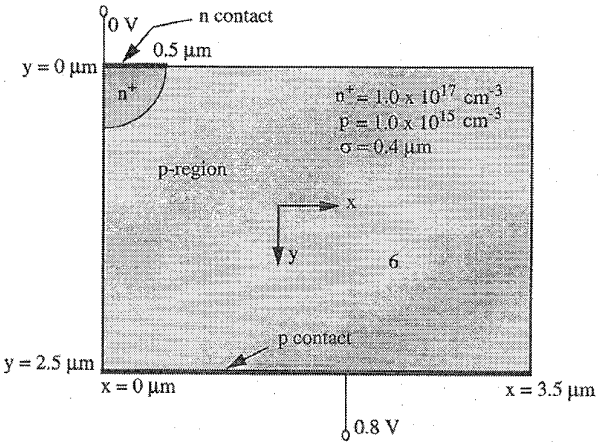


Fig. 20. Geometry of a 2-D *pn* diode.

TABLE II
SPEEDUP MEASUREMENTS ON i860 EMPLOYING 4, 8, 16, AND 32 PROCESSORS

Example	4	8	16	32
pn-2D	3.8	7.1	12.7	21.0
MESFET	3.7	6.4	11.0	16.5

TABLE III
SPEEDUP MEASUREMENTS ON SP-1 EMPLOYING 2, 4, AND 8 PROCESSORS

Example	2	4	8
pn-2D	1.9	3.5	6.0
MESFET	1.9	3.5	4.2

Fig. 21 shows the parallel performance obtained on the iPSC/860 and the SP-1 parallel computers for a 2-D silicon MESFET shown in Fig. 22. The device is discretized using 3072 mesh points and the results for this example were reported earlier in [8]. The problem size is too big for one processor of iPSC/860. The speedups obtained on the iPSC/860 and SP-1 are shown in Tables II and III, respec-

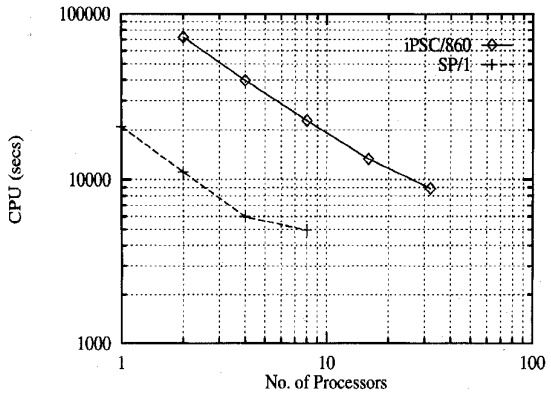


Fig. 21. CPU time comparison on iPSC/860 and SP-1 to solve 2-D silicon MESFET.

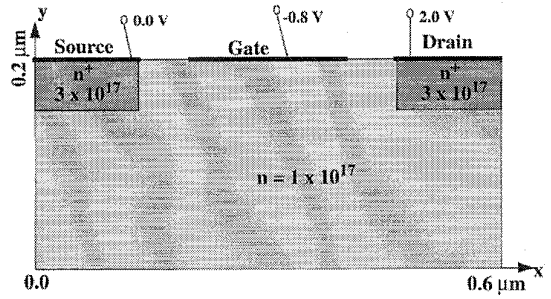


Fig. 22. Geometry of a silicon MESFET.

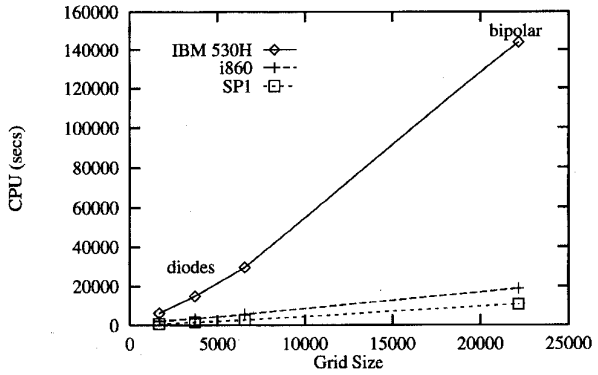


Fig. 23. Comparison of CPU times on IBM 530H, i860 (32 processors) and SP-1 (8 processors) for several diodes and a bipolar transistor.

tively. Observe that for SP-1 the speedup on eight processors is not significant as the problem size is small for eight processors and interprocessor communication time starts to dominate. This is a typical result on parallel computers as communication overhead becomes significant when the subdomain assigned to each processor is small.

Finally, Fig. 23 compares the performance of serial and parallel computers for several different device examples. The data points in the lower left portion of the figure are for diodes with increasing grid sizes. The data point with over 20 000 mesh points is for the bipolar transistor example. The CPU time on the serial machine, for increasing grid sizes,

(1) $\mathbf{r}^{(0)} = \mathbf{p}^{(0)} = \mathbf{b} - \mathbf{A}\,\mathbf{X}^{(0)}$
$\rho^{(0)} = \mathbf{r}^{(0)} \cdot \mathbf{r}^{(0)}$

(2) for k = 0, 1, . . . repeat;
$\alpha^{(k)} = \mathbf{r}^{(k)} / (\mathbf{p}^{(k)} \cdot \mathbf{A}\mathbf{p}^{(k)})$
$\mathbf{X}^{(k+1)} = \mathbf{X}^{(k)} + \alpha^{(k)}\,\mathbf{p}^{(k)}$
$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha^{(k)}\,\mathbf{A}\mathbf{p}^{(k)}$
$\rho^{(k+1)} = \mathbf{r}^{(k+1)} \cdot \mathbf{r}^{(k+1)}$
if $\rho^{(k+1)} <= tol \cdot \rho^{(0)}$ then terminate the loop
$\beta^{(k)} = \mathbf{r}^{(k+1)} / \mathbf{r}^{(k)}$
$\mathbf{p}^{(k+1)} = \mathbf{r}^{(k+1)} + \beta^{(k)}\,\mathbf{p}^{(k)}$

Fig. 24. Conjugate gradient algorithm for symmetric linear systems.

increases almost exponentially. The exponential behavior is both due to the cache and memory performance of the serial machine for increasing grid sizes and due to the physical nature of the problem being solved, i.e., the bipolar is a two-carrier problem and the diodes are solved by a single-carrier approximation. The CPU time obtained employing 32 processors of iPSC/860 increases sublinearly. Employing just eight processors of SP-1, the CPU time is shown to increase in a much slower pace. The sublinear behavior is due to the small communication overhead, good load balancing, and superior performance of cache and memory because of the decomposition of the original problem. The result signifies that as the grid size increases, the serial computers become impractical for device simulations. Parallel computers can help significantly in reducing the CPU time and can provide a platform for practical device simulations.

IX. Conclusion

A comprehensive convective hydrodynamic model comprising of the Poisson, electron hydrodynamic, hole hydrodynamic and the lattice thermal diffusion equations is simulated on distributed memory parallel computers. The hydrodynamic model is solved by employing space-time, Galerkin, and Galerkin/least-squares finite element methods. The proposed numerical methods are stable, accurate, convergent, and offer new avenues to traditional device modeling approaches. Results are shown for a bipolar transistor and steep layers and multiple scales encountered in the device are captured very effectively by the numerical method. Simulation results for the example device presented in this paper showed no substantial variation in the lattice temperature. Typically, the device dimensions need to be extended several times, as was done in [19], and Dirichlet temperature boundary conditions be carefully specified to observe significant variation in the lattice temperature.

The hydrodynamic device simulator is implemented on distributed memory parallel computers employing an SPMD programming model. The parallel code was originally developed on the Intel iPSC/860 and subsequently ported to Delta and IBM SP-1. Porting the code to the Delta machine

```
X = 0
(GMRES Cycles)
For n = 1, 2, . . ., l_max
    u^1 = -b - A X
    e = {|| u^1 ||, 0, . . . , 0}^T
    u^1 = u^1 / ||u^1||
    (GMRES iteration)
    For i = 1, 2, . . . , k
        u^{i+1} = A u^i
        (Modified Gram-Schmidt Procedure)
        For j = 1, . . . , i
            β^{i+1,j} = u^{i+1} . u^j
            u^{i+1} = u^{i+1} - β^{i+1,j} u^j
        (End Modified Gram-Schmidt Procedure)
        h^i = {β^{i+1,1}, . . . , β^{i+1,i}, ||u^{i+1}||}^T
        u^{i+1} = u^{i+1} / ||u^{i+1}||
    (End GMRES Iteration)
    H^k = [h^1, . . . , h^k]
    Solve min || e - H^k y || for y (use QR algorithm)
    X = X + Σ_{j=1}^{k} y^j u^j
    if || e - H^k y || <= δ, Exit l loop
(End GMRES Cycles)
```

Fig. 25. GMRES algorithm for nonsymmetric linear systems.

took less than a day and to the IBM SP-1 took less than a week. The small amount of time needed to port the code to different distributed memory machines demonstrates the wide portability of the code. Parallel performance results, shown for a bipolar transistor, silicon MESFET and diodes, indicate good scalability and speedup. The communication overhead is shown to be a small fraction of the total CPU time for a few number of processors and increases as the number of processors increases. The scalability results are better on the iPSC/860 compared to the SP-1 as the iPSC/860 uses a slower processor compared to SP-1. The CPU times on the SP-1 are reduced by a factor of about six compared to the CPU times on iPSC/860.

Finally, as the results indicate, the state-of-the-art parallel computers enable the solution of complex device structures employing the hydrodynamic model a reality. While 1-D simulations are possible on serial workstations, parallel computers are inevitable for the solution of complex 2- and 3-D structures, as is demonstrated in this paper for 2-D bipolar and other devices.

APPENDIX

Fig. 24 describes the sequential version of the conjugate gradient algorithm for solving a symmetric linear system of equation. Fig. 25 summarizes the sequential version of the GMRES algorithm for solving a nonsymmentric linear system of equations.

ACKNOWLEDGMENT

The authors wish to thank Dr. A. Raefsky, Professor P. M. Pinsky, and B. Herndon for their helpful discussions.

REFERENCES

[1] D. L. Scharfetter and H. K. Gummel, "Large-signal analysis of a silicon read diode oscillator," *IEEE Trans. Electon Devices*, vol. ED-16, pp. 64–77, 1969.

[2] M. Rudan and F. Odeh, "Multi-dimensional discretization scheme for the hydrodynamic model of semiconductor devices," *COMPEL*, vol. 5, pp. 149–183, 1986.

[3] M. Rudan, F. Odeh, and J. White, "Numerical solution of the hydrodynamic model for a one-dimensional device," *COMPEL*, vol. 6, pp. 151–170, 1986.

[4] D. Chen, E. C. Kan, U. Ravaioli, Z. Yu, and R. W. Dutton, "A self-consistent discretization scheme for current and energy transport equations," presented at the IV Int. Conf. Simulation Semiconductor Devices Processes, Zurich, Switerland, Sept. 12–14, 1991.

[5] C. L. Gardner, J. W. Jerome, and D. J. Rose, "Numerical methods for the hydrodynamic device model: Subsonic flow," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 501–507, 1989.

[6] E. Fatemi, J. W. Jerome, and S. Osher, "Solution of the hydrodynamic device model using high order nonoscillatory shock capturing algorithms," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 232–244, 1991.

[7] N. R. Aluru, A. Raefsky, P. M. Pinsky, K. H. Law, R. J. G. Goossens, and R. W. Dutton, "A finite element formulation for the hydrodynamic semiconductor device equations," *Comput. Meth. Appl. Mechanics Eng.*, vol. 107, pp. 269–298, 1993.

[8] N. R. Aluru, K. H. Law, P. M. Pinsky, A. Raefsky, R. J. G. Goossens, and R. W. Dutton, "Space-time Galerkin/least-squares finite element formulation for the hydrodynamic device equations," *IEICE Trans. Electron.*, vol. E77-C, no. 2, pp. 227–235, 1994.

[9] N. R. Aluru, K. H. Law, A. Raefsky, P. M. Pinsky, and R. W. Dutton, "Numerical solution of two-carrier hydrodynamic semiconductor device equations employing a stabilized finite element method," *Comput. Meth. Appl. Mech. Eng.*, vol. 125, pp. 187–220, 1995.

[10] A. Karp, "Programming for parallelism," *IEEE Trans. Comput.*, vol. C-36, pp. 43–57, May, 1987.

[11] B. P. Herndon, A. Raefsky, R. W. Dutton, and R. J. G. Goossens, "Parallelizing a PDE solver: Experiences with PISCES-MP," in *Proc. Aizu Int. Symp. Parallel A/A Syn.*, Mar. 15–18, 1995. pp. 287–296.

[12] B. P. Herndon, N. R. Aluru, A. Raefsky, R. J. G. Goossens, K. H. Law, and R. W. Dutton, "A methodology for parallelizing PDE solvers: Application to semiconductor device simulation," in *Proc. VII SIAM Conf. Parallel Process. Scientific Comput.*, San Francisco, CA, 1995, pp. 239–240.

[13] S. Selberherr, *An Analysis and Simulation of Semiconductor Devices*. New York: Springer, 1984.

[14] C. T. Wang, "A new set of semiconductor equations for computer simulation of submicron devices," *Solid-State Electron.*, vol. 28, no. 8, pp. 783–788, 1985.

[15] K. Blotekjaer, "Transport equations for electrons in two-valley semiconductors," *IEEE Trans. Electon Devices*, vol. 17, pp. 38–47, 1970.

[16] A. Gnudi, F. Odeh, and M. Rudan, "Investigation of nonlocal transport phenomena in small semiconductor devices," *Euro. Trans. Tel.*, vol. 1, pp. 307–313, 1990.

[17] G. Baccarani and M. R. Wordeman, "An investigation of steady-state velocity overshoot in silicon," *Solid-State Electron.*, vol. 28, pp. 407–416, 1985.

[18] B. Meinerzhagen and W. L. Engl, "The influence of the thermal equilibrium approximation on the accuracy of classical two-dimensional numerical modeling of silicon submicrometer MOS transistors," *IEEE Trans. Electron Devices*, vol. 35, pp. 689–697, 1988.

[19] S. Szeto and R. Reif, "A unified electrothermal hot-carrier transport model for silicon bipolar transistor simulations," *Solid-State Electron.*, vol. 32, no. 4, pp. 307–315, 1989.

[20] N. R. Aluru, "Parallel and Stabilized finite element methods for the hydrodynamic transport model of semiconductor devices," Ph. D Dissertation, Stanford Univ., Stanford, CA, June 1995.

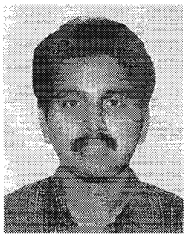
[21] T. J. R. Hughes, L. P. Franca, and G. M. Hulbert, "A new finite element formulation for computational fluid dynamics: VIII. The Galerkin/least-squares method for advective-diffusive systems," *Comput. Meth. Appl. Mech. Eng.*, vol. 58, pp. 173–189, 1986.

[22] F. Shakib, "Finite element analysis of the compressible Euler and Navier-Stokes equations," Ph.D dissertation, Stanford University, Stanford, CA, Nov. 1988.

[23] *Intel iPSC/860, Programmers Reference Manual*, Intel Corp., Order Number 311708-003, Beaverton, OR, 1990.

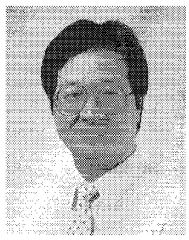
[24] *IBM User's Guide, Parallel Programming Subroutine Reference*, SH26-7228-00, IBM Corp., Kingston, NY 1993.

[25] *iPSC CIO Ethernet Reference Manual*, Intel Scientific Computers, Beaverton, OR, 1990.
[26] C. Farhat, "A simple and efficient automatic FEM domain decomposer," *Comput. Structures*, vol. 28, pp. 579–602, 1988.
[27] J. Flower, S. Otto, and M. Salama, "Optimal mapping of irregular finite element domains to parallel processors," in A. K. Noor, Ed., *Parallel Computations and Their Impact on Mechanics*. New York: The American Society of Mechanical Engineers, 1987, AMD-vol. 86, pp. 239–252.
[28] A. Pothen, H. D. Simon, and L. Wang, "Spectral nested dissection," Dept. Comput. Sci., Pennsylvania State Univ., University Park, PA, Tech. Rep. CS-92-01, 1992.
[29] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*. London, Eng.: Oxford University Press, 1986.
[30] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Baltimore, MD: The Johns Hopkins University Press, 1989.
[31] Y. Saad and M. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Statist. Comput.*, vol. 7, pp. 856–869, 1986.
[32] B. Nour-Omid, A. Raefsky, and G. Lyzenga, "Solving finite element equations on concurrent computers," in A. K. Noor, Ed., *Parallel Computations and Their Impact on Mechanics*. New York: ASME, 1987, AMD-vol. 86, pp. 209–228.
[33] K. H. Law, "A parallel finite element solution method," *Comp. Str.*, vol. 23, pp. 845–858, 1986.
[34] Z. Johan, "Data parallel finite element techniques for large-scale computational fluid dynamics," Ph.D dissertation, Stanford University, Stanford, CA, 1992.
[35] P. N. Brown and Y. Saad, "Hybrid Krylov methods for nonlinear systems of equations," *SIAM J. Sci. Comp.*, vol. 11, no. 3, pp. 450–481, 1990.



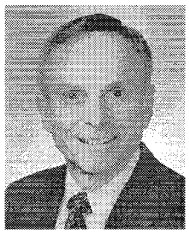
N. R. Aluru received the Bachelor of Engineering degree with honors and distinction from the Birla Institute of Technology and Science, Pilani, India, in 1989, the Master of Science degree from Rensselaer Polytechnic Institute, Troy, NY, in 1991, and the Ph.D. degree from Stanford University, Stanford, CA, in 1995.

He is currently a post-Doctoral Associate with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge. His research interests include micro-electro-mechanical-systems, efficient numerical methods, semiconductor device and process simulation, computational mechanics, and parallel processing.



K. H. Law received the B.S. degree in civil engineering with honors and the B.A. in mathematics with distinction from the University of Hawaii in 1976. He received the M.S. and Ph.D. degrees in civil engineering from Carnegie Mellon University, Pittsburgh, PA, in 1979 and 1981, respectively.

Prior to joining Stanford University, Stanford, CA, he served as a faculty member with Rensselaer Polytechnic Institute, Troy, NY, from 1982–1988. His research interests have focused on the application of advanced computing technology in engineering for over 14 years. His research has been involved in applied mathematical concepts and software engineering to develop effective methodologies for computationally intensive problems. His current work also includes parallel and distributed computing, with applications to semiconductor device simulations and fluid-structure interaction problems. He has authored and coauthored more than 100 publications in advanced engineering computing.



R. W. Dutton (S'67–M'70–SM'80–F'84) received the B.S., M.S., and Ph.D. degrees form the University of California, Berkeley, in 1966, 1967, and 1970, respectively.

He is a Professor of Electrical Engineering with Stanford University, Stanford, CA, and the Director of Research in the Center for Integrated Systems. He has held summer staff positions with Fairchild, Bell Telephone Laboratories, Hewlett-Packard, IBM Research, and Matsushita during 1967, 1973, 1975, 1977, and 1988, respectively. His research interests focus on integrated circuit process, device, and circuit technologies, especially the use of computer-aided design and parallel computational methods. He has published more than 200 journal articles and instructed more than four dozen Doctoral students.

He was the Editor of the IEEE COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. He received the 1987 IEEE J. J. Ebers Award and the 1988 Guggenheim Fellowship to study in Japan. He was elected to the National Academy of Engineering in 1991.