ELSEVIER

Parallel Computing 22 (1997) 1931–1963

PARALLEL COMPUTING

Practical aspects and experiences

Scalable massively parallel algorithms for computational nanoelectronics [1]

Xiaodong Wang [a,*], Vwani P. Roychowdhury [b,2], Pratheep Balasingam [c]

[a] *Department of Electrical Engineering, Princeton University, Princeton, NJ 08540, USA*
[b] *Electrical Engineering Department, University of California, Los Angeles, Los Angeles, CA 90095-1594, USA*
[c] *Meta-Software Inc., 1300 White Oaks Road, Cambell, CA 95008, USA*

Received 28 March 1996; revised 22 June 1996, 19 September 1996

Abstract

There is at present a worldwide effort to overcome the technological barriers to nanoelectronics. Microscopic simulation can significantly enhance our understanding of the physics of nanoscale structures, and constitutes a valuable tool for designing nanoelectronic functional devices. In nanodevices, novel physics effects are used to attain logic functionality which conventional technology can not achieve. Therefore it is necessary to develop quantum-transport simulation methods which include novel physical effects. Moreover, simulation of realistic nanodevices require enormous computing resource, necessitating parallel supercomputing. In this paper, we present massively parallel algorithms for simulating large-scale nanoelectronic networks based on the single-electron tunneling effect, which is arguably the quantum effect of greatest significance to nanoelectronic technology. A MIMD implementation of our simulation algorithm is carried out on a 64-processor nCUBE 2, and a SIMD implementation is carried out on a 16,384-processor MasPar MP-1. By exploiting massive parallelism, both parallel implementations achieve very high parallel efficiency and nearly linear scalability. The result of this work is that we are able to simulate large-scale nanoelectronic network, within a reasonable time period, which would be impractical on conventional workstations.

Keywords: Nanoelectronic networks; Electron dynamics; Monte Carlo simulation; Parallel algorithm; MIMD; SIMD; Scalability analysis

* Corresponding author. Email: wang@princeton.edu.

[1] This work was supported in part by the NSF Grants No. ECS-9308814 and No. ASC-9211073, and NSF Parallel Infrastructure Grant CDA-9015696.

[2] Email: vwani@ee.ucla.edu.

PII S0167-8191(96)00069-5

1. Introduction

Nanoelectronics is a revolutionary integrated circuit (IC) technology that will permit the down scaling of minimum circuit geometries to continue beyond the limiting length scales set by the conventional electron devices [25]. In nanodevices, novel physics effects are used to attain logic functionality which conventional technology can not achieve [7,21,24,14]. Recently there has been a significant increase in experimental activity involving the fabrication of nanoelectronic devices with feature sizes in the 1–100 nm range [25,20,30]. In this regime the dimensions of the structure become comparable with electron wavelengths, and charge transport is dominated by the fundamental quantum properties of the electron. Therefore the physical principles underlying the operation of conventional devices can no longer be utilized, and new physical effects must be exploited.

Along with fabricational advances, nanoelectronics will also be enabled by realistic quantum device modeling [26,25]. Modeling provides a natural framework for understanding the physics of nanoscale structures. It also constitutes a practical tool of increasing significance to the design of quantum devices. Microscopic simulation is the most effective modeling technique, which can significantly enhance our understanding of the nonlinear, nonequilibrium dynamics in the nanostructures. "Computational Electronics" is a rapidly growing, interdisciplinary approach (including computer science, electrical engineering, physics and mathematics) to device simulations [15]. In recent years, device simulations have played a significant role in the development of very large-scale integrated circuit (VLSI) technology. For nanoelectronic devices, however, it is necessary to develop quantum-transport simulation techniques which include novel physical effects crucial to the operation of such devices. Furthermore, simulations of realistic nanoelectronic devices require enormous computing resources. It is thus crucial to exploit the massively parallel computing technology. This so-called "Parallel Computational Nanoelectronics" is the subject of this paper.

We have been carrying out research on the design of integrated nanoelectronic systems with information processing capabilities [6,31,8]. Our work has been unified under a particular technology based on the creation of array of nanometer-sized metallic dots. We have studied different types of network mechanisms for the transfer of electrons between these quantum dots. Depending on the types of network nonlinearities permitted by the network links, we have shown that it is possible to generate different kinds of global activities in these networks. Moreover, we have shown that it is possible to impart computational interpretations to these global activities, and thus this class of networks will in effect become "artificial solids" [19] which compute [6,31,8].

In this paper, we present the massively parallel simulation techniques we have developed for understanding the underlying dynamics of such nanoelectronic networks. Such networks can be implemented by fabricating arrays of conductive quantum dots that interact with each other capacitively [6]. The basic physical concept underlying such networks is the single-electron tunneling effects [18], which is a discrete stochastic process. Monte Carlo simulation technique that mimics in detail the physics of single-electronics is an important tool for investigating the characteristics of such networks. Moreover, the computational needs for simulating these networks are very high,

necessitating the use of parallel supercomputers. In this paper, we present efficient massively parallel algorithms for simulating large-scale nanoelectronic networks. A MIMD implementation is described and the corresponding experiment on a 64-processor nCUBE 2 parallel computer is reported. A SIMD implementation is also described and the corresponding experiment on a 16,384-processor MasPar MP-1 parallel computer is reported. By exploiting massive parallelism, both parallel algorithms achieve very high parallel efficiency and nearly linear scalability. Our parallel simulators accomplish impressive speedup and make it possible to simulate large-scale nanoelectronic networks within reasonable time period.

2. Single-electron threshold devices

The class of nanometer scale electron devices, based on *quantum interference effects* are prone to universal conductance fluctuations [33] and therefore may not be suitable for consideration as a viable technology. Moreover, quantum coherence which is needed for interference effects will require very low operating temperatures, and low applied biases [33]. Another family of nanometer scale devices, known as *single electron transistors* [18] offer greater flexibility, in terms of design and operation. These are essentially semi-classical threshold devices in which the electrons can be thought of as discrete charged particles. They are therefore not subject to the interference fluctuations affecting other mesoscopic phenomena. Furthermore, it is believed that single-electron tunneling phenomena hold out the greatest promise [32] as the technology that will overcome the very low operating temperatures, which restrict other nanoelectronic concepts.

A particular realization of single-electron tunneling experiment is based on metallic dots which are deposited on a thin insulating layer, which in turn is grown on a conductive substrate [11]. The tip of a scanning tunneling microscope (STM) is then introduced as shown in Fig. 1, and the tunneling of electrons through the two potential barriers presented by the STM-dot gap, and the insulating layer between the dot and the substrate is studied. If the tunnel conductances are sufficiently small, the behavior of this

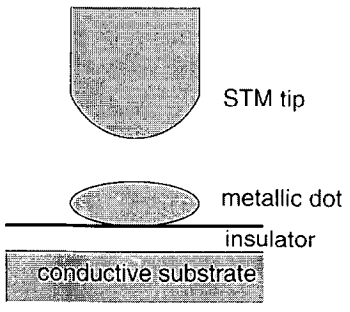


Fig. 1. STM-grain-substrate experiment for single-electron tunneling. The insulator is approximately 1 nm thick, and dots of diameter in the range 5–20 nm are typically studied.

system can be described using a semi-classical model, known as the "orthodox" theory [2]. In the framework of this theory there are well defined integer number of electrons on the dot. These charges are then changed discretely when single-electron tunneling events occur. At the absolute zero of temperature, such tunneling events can occur only if it is energetically favorable with respect to the electrostatic energy of the system. Now the novelty of these tunnel junctions arises from their small size. The capacitances of ultrasmall metallic dots can be very small, in the order of $C \sim 10^{-16}$ F, leading to significant capacitive contributions to the electrostatic energy through the term $e^2/2C$ [11]. So the voltage prevailing across a tunnel junction must cross the thresholds at $\pm e/2C$, before it can trigger a tunnel event. This *threshold* behavior is often referred to as Coulomb blockade, emphasizing the electrostatic origin of the phenomenon. As long as the capacitive energy $E_c = e^2/2C$, is not overshadowed by the thermal energy $E_T = kT$, small tunnel junction will exhibit this threshold behavior. A recent experiment [32] involving dots of diameter 4.5 nm, with pertinent $C \sim 10^{-18}$ F, has found evidence that the corresponding capacitive energy $E_c \sim 70$ meV, can indeed overcome the room temperature thermal energy $E_T \approx 26$ meV.

While there is optimism that nanoelectronic devices will lead to novel technologies, it is believed that they are not large enough to drive many other devices in subsequent stages. There have been suggestions that technological applications should be based, instead, on the cooperative behavior of arrays of such devices [18]. Indeed, there has been some theoretical and experimental work on arrays of laterally connected tunnel junctions [1,4,5]. In the next section we discuss the electron dynamics in the networks of such quantum dots.

3. Electron dynamics and Monte Carlo technique for voltage-driven networks of nanoscale metallic dots

3.1. Electron dynamics in voltage-driven nanoelectronic networks

We consider arrays of metallic dots each of small diameter in the 1–20 nm range, deposited onto an insulating layer, which has been grown on top of a conductive substrate [6]. The array is placed between two large electrodes – a source electrode and a drain electrode, as shown in Fig. 2(a). We investigate the electron transport across an array of such dots. We assume that an effective resistance parameter R_J of the tunnel junctions between any two adjacent dots is much larger than the quantum of resistance $R_Q = 2\pi\hbar/e^2$, and that the energy quantization within each dot is negligible compared to the charging energy of each dot. With these assumptions it is possible to use the "orthodox" theory [2] to describe the dynamics of electrons across an array of dots. The metallic dots are assumed to be sufficiently close to each other that lateral tunneling between near-neighbor occurs. Furthermore, we assume that the insulating layer of the substrate is sufficiently thick that vertical tunneling between an dot and the substrate will not occur.

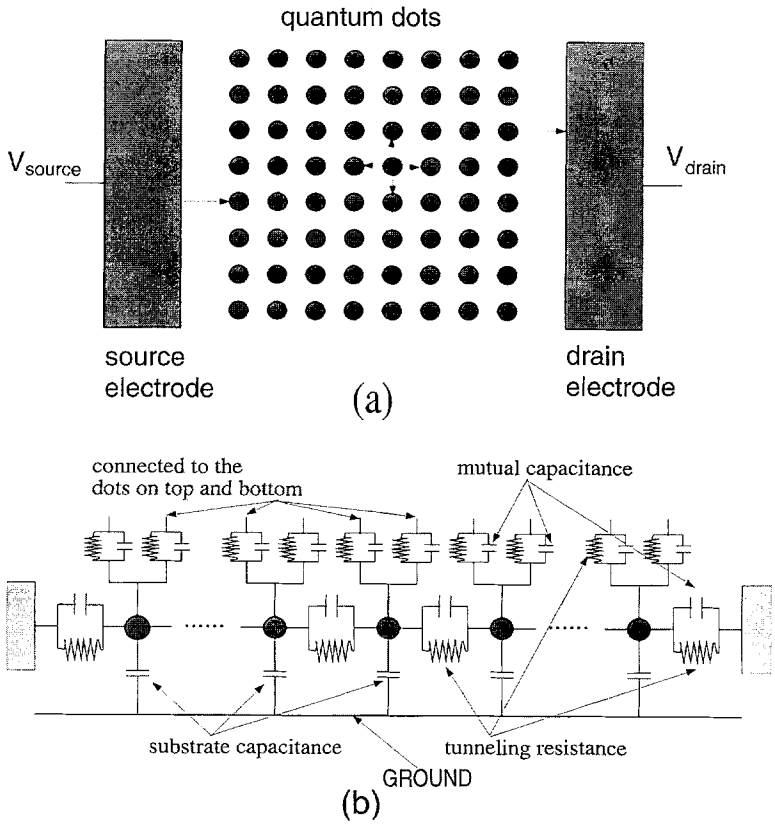


Fig. 2. (a) An array of metallic dots placed between a source electrode and a drain electrode. Single-electron tunneling event can happen only between neighbor dots, or between the electrode and the dots on the edge, as illustrated by the arrowed lines. (b) Cross-sectional view of the circuit model of (a), in which only the near neighbor capacitive coupling is assumed.

The dynamics of electrons across the array is simulated by means of an algorithm which carries out electron transitions between neighbor dots if that transition would in general minimize the total energy. At any given time, the entire array is scanned and the energy change associated with each electron transition is recorded. The rates of making those transitions are then determined, and a particular transition is picked at random with probability proportional to the tunneling rates. The method is due to Bakhvalov et al. [4], who showed that the simulation technique mimics exactly the behavior of the system as long as the tunnel resistance R_J between any two adjacent dots is much larger than the quantum of resistance R_Q.

Let C_{ij} denote the mutual capacitance between dots i and j, C_{is} denote the capacitance between dot i and the source electrode, C_{id} denote the capacitance between dot i and the drain electrode, and C_i denote the capacitance between dot i and the substrate. Suppose that at time t, the charge on dot i is $Q_i(t)$ and the dot voltage,

measured with respect to the substrate is $V_i(t)$. Let the source and drain voltages be V_s and V_d respectively. The charge can then be written in terms of the voltage as

$$Q_i(t) = C_i V_i(t) + \sum_{i \neq j} C_{ij}\big(V_i(t) - V_j(t)\big) + C_{is}\big(V_i(t) - V_s\big) + C_{id}\big(V_i(t) - V_d\big). \tag{1}$$

Given a particular charge configuration $\{Q_i(t)\}$ of electrons on each of the dots at time t, and the applied biases V_s and V_d at the source and drain electrodes, the corresponding set of voltages $\{V_i(t)\}$ on the dots can be derived by inverting Eq. (1)

$$V_i(t) = \sum_j [\mathscr{C}^{-1}]_{ij} \tilde{Q}_j(t), \tag{2}$$

where the off-diagonal elements of the capacitance matrix $\mathscr{C}$ are $\mathscr{C}_{ij} = -C_{ij}$, the diagonal elements are $\mathscr{C}_{ii} = C_i + \Sigma_{j \neq i} C_{ij} + C_{is} + C_{id}$, and $\tilde{Q}$ is an augmented charge vector of the system, $\tilde{Q}_j = Q_j + C_{js} V_s + C_{jd} V_d$.

The free energy which is minimized during the transport analysis can be written as

$$E = \frac{1}{2} \tilde{Q}^T \mathscr{C}^{-1} \tilde{Q} - Q_{in} V_s + Q_{out} V_d. \tag{3}$$

where the first term is the electrostatic energy stored in all of the capacitances in the network, the second term is the work done by the source electrode in injecting the charge Q_{in} into the network, and the third term is the work done by the drain electrodes in pulling the charge Q_{out} out of the network. (Q_{in} is the total charge injected into the network from the source electrode, and Q_{out} is the total charge pulled out of the network to the drain electrode. Therefore the net charge in the network is $Q_{in} - Q_{out}$.) Now for a charge q tunneling from dot i to dot j, the change in the free energy is

$$\Delta E_{ij} = E^a - E^b, \tag{4}$$

where the superscripts indicate whether the energies are taken *b*efore or *a*fter the tunneling event.

Once the energy change associated with a tunneling event is known, the tunneling rate Γ_{ij} corresponding to that event can be obtained by applying Fermi's Golden rule,

$$\Gamma_{ij} = \frac{2\pi}{\hbar} \int dE f\big(E - E_{f_i}\big)\Big\{1 - f\big(E - E_{f_j}\big)\Big\} D_i\big(E - E_{f_i}\big) D_j\big(E - E_{f_j}\big) \,|\, T_{ij}(E) \,|^2, \tag{5}$$

where f is the Fermi–Dirac distribution function giving the electron occupancies within each dot, D are densities of states within the dots, and T_{ij} is the matrix element for tunneling from dot i to dot j. In most experiments the biases needed to overcome the Coulomb blockade are sufficiently small, so that the energy dependence of the densities of states and the matrix element can be neglected over the energy interval of interest. Under these assumptions neither a detailed model of the tunneling barrier, nor of the

electron gas in the metallic dots is necessary, and the tunneling rate can be approximated as [6],

$$\Gamma_{ij} = \frac{\Delta E_{ij}}{q^2 R_{ij}\{\exp(\Delta E_{ij}/kT) - 1\}}, \tag{6}$$

where R_{ij} is an effective tunnel resistance which is composed of the densities of states, and the tunneling matrix element, all evaluated at the Fermi energy, q is the single electron charge, k is the Boltzman constant, and T is the absolute temperature. For transport considerations at the absolute zero of temperature, (i.e., $T = 0$) the rate equation can be simplified further to yield,

$$\Gamma_{ij} = \begin{cases} \Delta E_{ij}/q^2 R_{ij} & \text{if } \Delta E_{ij} < 0, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

3.2. Monte Carlo simulation technique

3.2.1. Simulating the single electron tunneling event

Next we will outline a Monte Carlo simulation technique [4,13] for the simulation of a voltage-driven network of quantum dots discussed above. The state of the system of quantum dots is fully described at time t by the vector of dot charges $Q(t)$. The entire system is swept, and a vector of numbers $\{\Delta E^m\}$ corresponding to the energy dissipated as a result of tunnel events (indexed by the superscript m), between each dot and its nearest neighbors (including the tunnel events between the edge dots and the electrodes). From the vector of dissipated energies, a vector of tunnel rates $\{\Gamma^m\}$ is then generated using Eq. (6) or Eq. (7). Another vector of cumulative tunnel rates is then generated by replacing each element of the vector $\{\Gamma^m\}$ with the sum of all previous rates (prefix sum):

$$S^m = \sum_{k=1}^{m} \Gamma^k. \tag{8}$$

The probability that any one of these tunnel events should proceed over a very small time interval δt is then determined by calculating

$$P(t + \delta t) = e^{-\delta t S^l}. \tag{9}$$

where l is the total number of possible tunnel events, and the total tunnel rate S^l is the last element in the vector of cumulative rates $\{S^m\}$.

Now a particular tunnel event k can be selected with probability proportional to the tunneling rates Γ^k. This is numerically implemented by picking a random number r_π distributed uniformly on the unit interval, and then selecting the tunnel event with the lowest index k, which meets the condition $(S^k/S^l) > r_\pi$. We still need to determine the length of the time interval between t and the time this tunnel event happens. Another random number τ_ϕ, distributed uniformly on the unit interval is now drawn, and the time interval δt is then,

$$\delta t = -\ln r_\phi / S^l. \tag{10}$$

After this time lapse δt, the system is ready for the next update. We call this Monte Carlo process for the nano-array a *SWEEP* process. The entire *SWEEP* process is repeated until stationary mean values $\{\langle Q_i \rangle\}$, corresponding to a local minimum of the energy set in, or until adequate numerical evidence has been accumulated.

3.2.2. Calculating the steady state current

The steady state current of the voltage-driven network is calculated in the following way. Suppose that totally ν electron tunneling events are simulated, i.e., the *SWEEP* process is iterated ν times. At the end of the transient period, e.g., after the ν_0th ($\nu_0 < \nu$) iteration of the *SWEEP* process, we calculate the total elapsed time and the charge flows at the two electrodes. The total elapsed time after the ν_0th iteration is $T(\nu_0) = \sum_{i=1}^{\nu_0} \delta t_i$, where δt_i is the time interval between the $(i-1)$th and the ith tunneling events, calculated by Eq. (10). The total charge flow at the source electrode after the ν_0th iteration is $Q_s(\nu_0) = Q_{in}(\nu_0) + \sum_j C_{js}(V_s - V_j(\nu_0))$, where $Q_{in}(\nu_0)$ is the total charge tunneling to quantum dots in the network from the source electrode during the ν_0 *SWEEP* processes, and $V_j(\nu_0)$ is the voltage at dot j at the end of the ν_0th *SWEEP* process. Similarly the total charge flow at the drain electrode after the ν_0th iteration is $Q_d(\nu_0) = Q_{out}(\nu_0) + \sum_j C_{jd}(V_d - V_j(\nu_0))$, where $Q_{out}(\nu_0)$ is the total charge tunneling from the quantum dots in the network to the drain electrode during the ν_0 *SWEEP* processes. At the end of the νth iteration we calculate the elapsed time and the charge flows at the two electrodes again: $T(\nu) = \sum_{i=1}^{\nu} \delta t_i$, $Q_s(\nu) = Q_{in}(\nu) + \sum_j C_{js}(V_s - V_j(\nu))$, and $Q_d(\nu) = Q_{out}(\nu) + \sum_j C_{jd}(V_d - V_j(\nu))$. Then the steady state current is calculated by

$$I = \frac{Q_s(\nu) - Q_s(\nu_0)}{T(\nu) - T(\nu_0)}, \tag{11}$$

or

$$I = \frac{Q_d(\nu) - Q_d(\nu_0)}{T(\nu) - T(\nu_0)}. \tag{12}$$

If at a particular *SWEEP* process during the simulation, there is no tunneling event feasible, then the system reaches metastable state, and the corresponding current is 0.

4. Parallelizing the simulation of electron dynamics in nanoelectronic networks

4.1. Structure of the simulation algorithm and domain decomposition

We are interested in the current–voltage (I–V) characteristics of the voltage-driven nanoelectronic network introduced in Section 3. Based on the preceding discussions, we can give the structure of the simulation algorithm, as shown in Fig. 3.

From Fig. 3 it is clear that the simulation algorithm consists of two major computation blocks: (1) calculate the synaptic weight matrix W, and (2) carry out the electron dynamics in the network of quantum dots through Monte Carlo simulation. Both of these

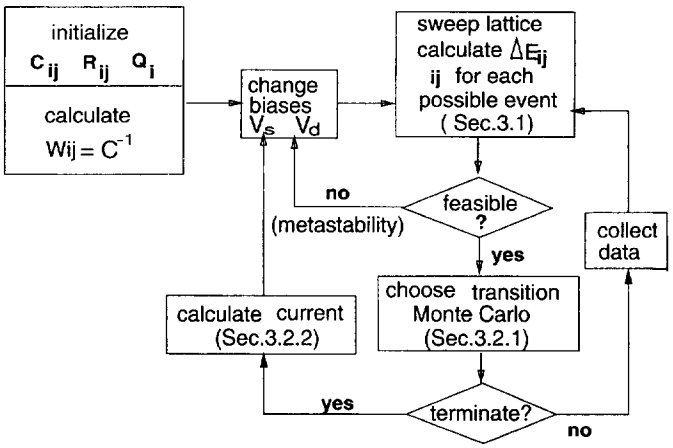


Fig. 3. The structure of the simulation algorithm.

two computation blocks exhibit high degree of data parallelism, i.e., parallelism achieved by multiple processors simultaneously processing different subsets of the data. Furthermore, the regularity of the data structure and the uniformity of the computation over the nano-array has important implications particularly for the parallel implementation. A careful consideration of the presence and efficient use of data parallelism can lead to very high parallel performance.

Our parallel simulation algorithm divides the network of $n = N_x \times N_y$ quantum dots into a number of regions equal to the number of processors p. The network is divided into rectangular regions by forming P_x strips in the x dimension and P_y strips in the y dimension, where $p = P_x \times P_y$, as shown in Fig. 4 for $N_x = N_y = 12$ and $P_x = P_y = 4$. We next present efficient parallel algorithms for the two computation blocks.

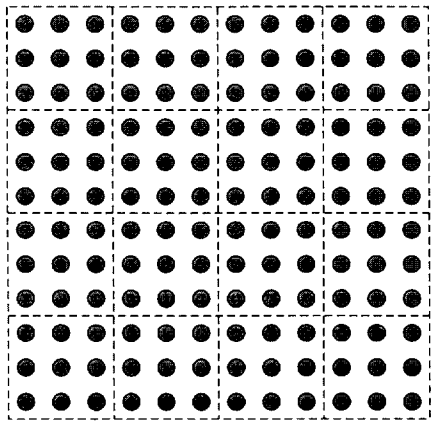


Fig. 4. Grid partition of the network of quantum dots for parallel simulation.

4.2. Parallel calculation of the synaptic weight matrix

The first computational task is calculating the synaptic weight matrix $W = \mathscr{C}^{-1}$ of the nano-array. Suppose that there are p processors, and the nano-array is of size $n = N_x \times N_y$. Then the capacitance matrix $\mathscr{C}$ is an $n \times n$ matrix. After partitioning the array each processor has n/p quantum dots and correspondingly, each processor has n/p rows of $\mathscr{C}$. Each row i is a capacitance vector consisting of the capacitances between dot i and all the other dots. After matrix inversion, the corresponding row i of W is a weight coupling vector consisting of the coupling weights between dot i and all the other dots j, $1 \leqslant j \leqslant n$.

Currently, the metallic dots on the nano-array under our consideration are assumed *locally* capacitively coupled. This leads to a multidiagonal structure of the synaptic weight matrix W, as derived below. A parallel Gauss–Seidel iterative algorithm can then be used for computing W approximately. Moreover, recent research on molecular networks [10] and nanotubes [9], are indicative of approaches which may lead to viable *global* coupling, i.e., any two dots on the array can be coupled with arbitrary capacitive strength, resulting in a dense matrix $\mathscr{C}$ and its inverse W. A parallel Gauss–Jordan elimination algorithm can be used for solving W in that case, which is also outlined in this section.

4.2.1. Structure of the synaptic weight matrix W

We study here the structure of the synaptic weight matrix when only *local capacitive coupling* is considered. If only near neighbor capacitive coupling is considered, then the capacitance matrix for an one-dimensional regular array is a tridiagonal matrix of the following form,

$$\mathscr{C} = \begin{bmatrix} C_0 + C & -C & 0 & 0 & \cdots & 0 \\ -C & C_0 + 2C & -C & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & -C & C + C_0 \end{bmatrix},$$

where C_0 is the capacitance to the substrate, and C is the inter-dot capacitance. Let us examine the first column of W, which satisfies

$$\begin{bmatrix} C_0 + C & -C & 0 & 0 & \cdots & 0 \\ -C & C_0 + 2C & -C & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & -C & C + C_0 \end{bmatrix} \begin{bmatrix} w_{11} \\ w_{12} \\ \vdots \\ w_{1n} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

The above matrix equation corresponds to a second-order linear difference equation of the following form,

$$w_{1i} = \delta(w_{1(i-1)} + w_{1(i+1)}) \quad \text{for } i = 2, \ldots, (n-1),$$

where $\delta = C/(C_0 + 2C)$, and $w_{10} = 1$. The solution can be given as

$$w_{1i} = a\alpha^{(i-1)} + b\alpha^{(n-i)} \quad \text{for } i = 1, \ldots, n,$$

where

$$\alpha = \frac{1}{2\delta}\left(1 - \sqrt{1 - 4\delta^2}\right) < 1,$$

and the constants a and b are determined from the boundary conditions (i.e., from the difference equations corresponding to $i = 1$ and $i = n$). Clearly, w_{1i}'s decrease exponentially as a function of i.

One can carry out a similar analysis for the other columns of W, and show that the elements in every row/column of W decrease exponentially with their distance from the diagonal.

In general, if every dot is capacitively coupled to k of its neighbors, then the preceding analysis can again be carried out. It will involve kth-order linear difference equations, whose solutions will determine the entries of W. One can then use this correspondence to show that every element w_{ij} of the synaptic weight matrix W, decreases exponentially with the distance between dot i and dot j. Hence, the synaptic matrix W can be modeled essentially as a banded matrix with a width of O(log n). That is, for any i, $w_{ij} \approx 0$ if dot j is more than c log n (where, c is determined by k) distance away from dot i.

The banded structure of W corresponds to having a nano-array with O(log n) synaptic connectivity. Therefore, for each dot i, we can set a square window of size $(2r+1)\times(2r+1)$, centered at dot i, where $r = c$ log n, and we assume that the synaptic weight between dot i and any dot inside the window is nonzero, and the weight between dot i and any dot outside the window is zero, as shown in Fig. 5. In this way, at each processor the memory requirement for storing W can be reduced to $(2r+1)^2 n/p = O((N^2/p)\log^2 N)$, for an $N \times N$ network. The nonzero elements of W can be calculated approximately by using the Gauss–Seidel algorithm.

4.2.2. Gauss–Seidel iteration

We need to get the synaptic matrix W by solving the following equation

$$\mathscr{C}W = I, \tag{13}$$

where I is an $n \times n$ identity matrix (recall that $n = N_x \times N_y$). Let $W = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n]$

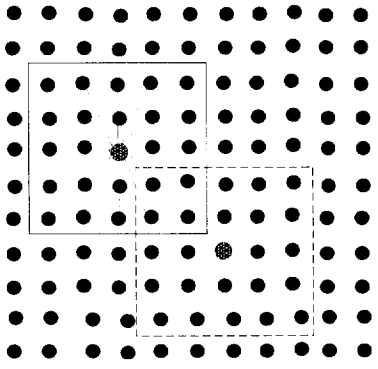


Fig. 5. For each quantum dot i in the network, set a square window of size $(2r+1)\times(2r+1)$, e.g., $r = 2$, centered at dot i, and we assume that the synaptic weight between dot i and any dot inside the window is nonzero, and the weight between dot i and any dot outside the window is zero.

and $I = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n]$, where $\mathbf{w}_i = [w_{1i}, w_{2i}, \ldots, w_{ni}]^T$ and $\mathbf{e}_i = [0, \ldots, 1, \ldots, 0]^T$, i.e., the ith element of $\mathbf{e}_i$ is 1 and all the other elements are 0's. Then we have n equations

$$\mathscr{C}\mathbf{w}_l = \mathbf{e}_l, \quad l = 1, 2, \ldots, n. \tag{14}$$

Gauss–Seidel (GS) iteration [29] can be applied to solve $\mathbf{w}_l$:

$$w_{il}^{k+1} = \frac{1}{c_{ii}}\left(-\sum_{j<i} c_{ij} w_{jl}^{k+1} - \sum_{j>i} c_{ij} w_{jl}^{k} + e_{il} \right), \quad i = 1, \ldots, N, \ k = 0, 1, \ldots. \tag{15}$$

An important modification to the Gauss–Seidel iteration is the successive overrelaxation (SOR) iteration [29]. Let $\hat{w}_{il}^{k+1}$ denote the right-hand side Eq. (15) and define

$$w_{il}^{k+1} = w_{il}^{k} + \omega\left(\hat{w}^{k+1 il} - w_{il}^{k} \right). \tag{16}$$

The introduction of the parameter ω is to enhance the rate of convergence of the Gauss–Seidel iteration. When we assume that the quantum dots are near-neighbor coupled capacitively, the capacitance matrix $\mathscr{C}$ is a diagonally dominant sparse matrix, and consists of only five nonzero diagonals. Matrices of this form are called Poisson matrix [29]. For Poisson matrix ω is chosen as

$$\omega = \frac{2}{1 + \left(1 - \mu^2\right)^{1/2}}, \tag{17}$$

$$\mu = \cos \frac{\pi}{n+1}. \tag{18}$$

Let $\{\mathbf{w}_l^k\}$ is the sequence of iterates for the lth column of W, then the convergence test is given by

$$\| \mathbf{w}_l^{k+1} - \mathbf{w}_l^k \|_\infty \leqslant \varepsilon, \tag{19}$$

where $\| \mathbf{w}_l \|_\infty = \max_{1 \leqslant i \leqslant n} | w_{il} |$ is the l_∞ vector norm.

Since we assume that for each dot, the synaptic weight between this dot and any dot outside a $(2r+1) \times (2r+1)$ square window centered at this dot is zero, then each $\mathbf{w}_l$ vector is of length $(2r+1)^2$ instead of n. Moreover, since we assume that there is capacitance coupling only between two near neighbor dots, in Eq. (15) there are at most four c_{ij}'s that are nonzero. These capacitances can be represented as c_u, c_d, c_l, c_r, i.e., the capacitance couplings between dot i and its upper, lower (down), left and right neighbors, respectively. Let these four neighbors be indexed as i_u, i_d, i_l, i_r, then Eq. (15) can be rewritten as

$$w_{il}^{k+1} = \frac{1}{c_{ii}}\left(c_u w_{i_u l}^{k+1} + c_l w_{i_l l}^{k+1} + c_d w_{i_d l}^{k} + c_r w_{i_r l}^{k} + e_{il} \right). \tag{20}$$

The data parallelism lies in the fact that the n linear equations in Eq. (14) can be solved independently by using equation Eq. (20) iteratively. Each processor can solve n/p columns of W in parallel with the other processors. Note that the parallel Gauss–Jordan elimination solves W among the processors by rows, while the parallel

Gauss–Seidel iteration solves W among the processors by columns. Since $\mathscr{C}$ is symmetric, so is W, the two methods actually obtain the same results. The pseudo code for the Gauss–Seidel iteration at each processor is as follows:

```
for each dot l in the region
  for each dot i inside the square window centered at dot l
  calculate i_u, i_d, i_l, i_r and c_u, c_d, c_l, c_r, c_ii
  s = 1/c_ii(c_u w_{i_u l} + c_l w_{i_l l} + c_d w_{i_d l} + c_r w_{i_r l} + e_il)
  δ = s − w_il
  η_l = max(η_l, | d |)
  w_il = w_il + ω · δ
Until all η_l's ⩽ ε
```

Although all the processors solve the same number (i.e., n/p) equations, the computation load among the processors may be unbalanced, due to the different convergence rates of the Gauss–Seidel iterations on these equations. This load imbalance is caused by the numerical property of the problem, and it can not be predicted in advance. The load imbalance is determined by factors including the structure of the matrix, the error tolerance for convergence, and the partition of W among the processors, etc.

There is no communication overhead when using the parallel Gauss–Seidel iteration to solve the synaptic weight matrix W. All the processors calculate their shares of W independently, without any inter-processor communication or synchronization necessary.

4.2.3. Gauss–Jordan elimination for computing dense synaptic weight matrix

As mentioned earlier, research progress in molecular wiring technology may make the fabrication of nano-array with a dense capacitance coupling matrix $\mathscr{C}$ feasible. We next outline a parallel Gauss–Jordan (GJ) elimination algorithm [16] for computing the inverse of such a *dense* matrix $\mathscr{C}$. By construction, the capacitance matrix $\mathscr{C}$ is symmetric and diagonally dominant, therefore, GJ algorithm is numerically stable for computing $W = \mathscr{C}^{-1}$ without using partial pivoting. The serial pseudo code for inverting the $n \times n$ matrix $\mathscr{C} = [c_{ij}]$ using GJ elimination is as follows:

```
for k = 1 to n
  for j = 1 to n and j ≠ k
    c_kj = c_kj / c_kk
    for i = 1 to n and i ≠ k
      c_ij = c_ij − c_ik · c_kj
  for i = 1 to n and i ≠ k
    c_ik = − c_ik / c_kk
  c_kk = 1 / c_kk
```

In the i loop, the same reduction operations are applied to all the rows of matrix $\mathscr{C}$, this is the data parallelism in the GJ algorithm. Let each processor have n/p rows of $\mathscr{C}$. At the kth iteration of the reduction, all the processors get the elements of the kth row from the processor that has this pivot row through interprocessor communication. After that, all the processors can apply the reduction operations on their shares of data in parallel. The communication requirement for the parallel GJ elimination is a one-to-all broadcasting of the pivot row at every iteration.

4.3. Parallel Monte Carlo simulation of the electron dynamics of the nanoelectronic networks

4.3.1. The local updating property

We next derive the local updating formulae for calculating the energy differentials (ΔE_{ij}'s) and for updating the dot voltages (V_i's).

Fact 1 [Local calculation of ΔE_{ij}]. *When a charge q tunnels from dot i to dot j, the change in the free energy can be written as*

$$\Delta E_{ij} = \frac{q}{2}\left[\left(V_j^b + V_j^a\right) - \left(V_i^b + V_i^a\right)\right], \tag{21}$$

where the subscript of V denote the indices of the dots involved in the transition, while the superscripts indicate whether the voltage are taken before or after the tunneling event. Eq. (21) also holds when charge q tunnels from or to an electrode. If q tunnels from source (or drain) electrode to dot j, then $V_i^a = V_i^b = V_s$ (or $V_i^a = V_i^b = V_d$). If q tunnels to source (or drain) electrode from dot i, then $V_j^a = V_j^b = V_s$ (or $V_j^a = V_j^b = V_d$).

Proof. Let $\tilde{Q}^b = [q_1\ q_2 \ldots q_n]$ be the augmented charge vector of the system before the tunneling event. Then the augmented charge vector of the system after a single electron tunnel event involving the charge q tunneling from dot i to dot j has occurred can be represent as

$$\tilde{Q}^a = \tilde{Q}^b + \Delta Q = [q_1, \ldots, q_i - q, \ldots, q_j + q, \ldots, q_n].$$

where $\Delta Q = q(\mathbf{e}_j - \mathbf{e}_i)$, $\mathbf{e}_i$ is a vector where all the elements are zeros, except the ith element, which is one.

Using Eqs. (3) and (4) we have

$$\begin{aligned}
\Delta E_{ij} &= E^a - E^b \\
&= \frac{1}{2}\tilde{Q}^{a^T}\mathscr{C}^{-1}\tilde{Q}^a - \frac{1}{2}\tilde{Q}^{b^T}\mathscr{C}^{-1}\tilde{Q}^b \\
&= \frac{1}{2}\left(\tilde{Q}^{b^T} + \Delta Q^T\right)\mathscr{C}^{-1}\left(\tilde{Q}^b + \Delta Q\right) - \frac{1}{2}\tilde{Q}^{b^T}\mathscr{C}^{-1}\tilde{Q}^b \\
&= \tilde{Q}^{b^T}\mathscr{C}^{-1}\Delta Q + \frac{1}{2}\Delta Q^T\mathscr{C}^{-1}\Delta Q \\
&= \frac{1}{2}\left[\tilde{Q}^{b^T} + \left(\tilde{Q}^b + \Delta Q\right)^T\right]\mathscr{C}^{-1}\Delta Q \\
&= \frac{1}{2}\left(\tilde{Q}^{b^T}\mathscr{C}^{-1} + \tilde{Q}^{a^T}\mathscr{C}^{-1}\right)\Delta Q \\
&= \frac{q}{2}\left(V^{b^T} + V^{a^T}\right)(\mathbf{e}_j - \mathbf{e}_i) \\
&= \frac{q}{2}\left[\left(V_j^b + V_j^a\right) - \left(V_i^b + V_i^a\right)\right].
\end{aligned}$$

Now suppose that a charge q tunnels from the source electrode to dot j. In this case $\Delta Q = q\mathbf{e}_j$. Then using Eqs. (3) and (4) we get

$$\begin{aligned}\Delta E_{ij} &= E^a - E^b \\ &= \frac{1}{2}\tilde{Q}^{a^T}\mathscr{C}^{-1}\tilde{Q}^a - \frac{1}{2}\tilde{Q}^{b^T}\mathscr{C}^{-1}\tilde{Q}^b - qV_s \\ &= \frac{1}{2}\left(\tilde{Q}^{b^T}\mathscr{C}^{-1} + \tilde{Q}^{a^T}\mathscr{C}^{-1}\right)\Delta Q - qV_s \\ &= \frac{q}{2}\left[\left(V_j^b + V_j^a\right) - 2V_s\right].\end{aligned}$$

Similarly we can prove the other cases involving charge q tunneling between a dot and the electrode. □

Let $W = \mathscr{C}^{-1}$, we call W the synaptic weight matrix of the nanonetwork. From Section 3, by the construction of $\mathscr{C}$, it is symmetric and diagonally dominant, therefore, its inverse W exists and is also symmetric.

Fact 2 [Local updating of V_i]. *Suppose that a tunnel event is charge q tunneling from dot i to j, then for any dot k,*

$$V_k^a = V_k^b + qw_{kj} - qw_{ki}. \tag{22}$$

In particular,

$$V_i^a = V_i^b + qw_{ij} - qw_{ii}, \tag{23}$$

$$V_j^a = V_j^b + qw_{jj} - qw_{ji}. \tag{24}$$

Eqs. (22), (23) *and* (24) *also hold when charge q tunnels from or to an electrode in the following way: If q tunnels from an electrode to dot j, then $w_{ki} = 0$; If q tunnels to an electrode from dot i, then $w_{kj} = 0$.*

Proof. From Eq. (2)

$$\begin{aligned}V_k^a &= \left[\mathscr{C}^{-1}\tilde{Q}^a\right]_k \\ &= \left[\mathscr{C}^{-1}\left(\tilde{Q}^b + \Delta Q\right)\right]_k \\ &= \left[\mathscr{C}^{-1}\tilde{Q}^b\right]_k + q\left[\mathscr{C}^{-1}(\mathbf{e}_j - \mathbf{e}_i)\right]_k \\ &= V_k^b + q\left[W\mathbf{e}_j\right]_k - q\left[W\mathbf{e}_i\right]_k \\ &= V_k^b + qw_{kj} - qw_{ki}.\end{aligned}$$

If charge q tunnels from the source electrode to dot j, then $\Delta Q = \mathbf{e}_j$, therefore

$$\begin{aligned}V_k^a &= \left[\mathscr{C}^{-1}\tilde{Q}^a\right]_k \\ &= \left[\mathscr{C}^{-1}\left(\tilde{Q}^b + \Delta Q\right)\right]_k\end{aligned}$$

$$= \left[\mathscr{C}^{-1}\left(\tilde{Q}^b + \mathbf{e}_j\right)\right]_k$$
$$= V_k^b + q\left[W\mathbf{e}_j\right]_k$$
$$= V_k^b + qw_{kj}.$$

Similarly we can prove other cases involving charge q tunneling between a dot and the electrode. □

After substituting Eqs. (23) and (24) into (21), we have

$$\Delta E_{ij} = q\left(V_j^b - V_i^b\right) + q^2\left[w_{ij} - \frac{1}{2}(w_{ii} + w_{jj})\right]. \tag{25}$$

Eqs. (22) and (25) indicate both the calculation of the energy differentials associated with a tunnel event and the updating of the dot voltages after a tunnel event has happened can be carried out *locally*. This local updating property makes the simulation suitable for implementation on parallel computers.

4.3.2. Parallel Monte Carlo simulation

The second computational task is the Monte Carlo simulation of the electron dynamics, which carries out a series of random tunnel events in a manner that minimizes the overall energy of the whole system. Starting from the initial state s_0, the Monte Carlo simulation algorithm generates a sequence of states $s_1, s_2, \ldots, s_\nu$. Each state corresponds to a configuration of charge distribution on the nano-network. The state s_{i+1} results from randomly choosing an energetically favorable tunneling event in state s_i. To make the transition $s_i \to s_{i+1}$, all the possible tunneling events on the nano network need to be evaluated, and the corresponding energy differential and the transition rate of each event are calculated.

Data parallelism can be exploited at this level by assigning a different region of the network to each of the processors. The simulation of each individual transition $s_i \to s_{i+1}$ is made faster by breaking up the task of evaluating all the possible tunnel events into subtasks and allocating various subtasks to different processors. At each state s_i, each processor evaluates all the possible electron tunnel events in its share of the network. Since only the tunnel events between near neighbor quantum dots are considered, and from Section 4.3.1 the energy differential for each event can be evaluated locally, thus only near neighbor interprocessor communication is required when calculating the energy differentials for all the possible electron tunnel events.

As discussed in Section 3, after calculating the transition rate of each tunnel event, the cumulative transition probabilities of all the events, $P_1, P_2, \ldots, P_l$, are calculated, such that $0 < P_1 \leqslant P_2 \leqslant \cdots \leqslant P_{l-1} P_l = 1$, where l is the total number of possible events. Then a random number r, $0 < r < 1$, is generated and a tunneling event k is chosen such that $P_{k-1} < r \leqslant P_k$, and the system moves from state s_i to state s_{i+1}. When implemented in parallel, each processor calculates all the transition rates of the events within the region using Eq. (6) or (7). Based on the transition rates, the calculation of the cumulative transition probabilities of all the events can be done by parallel prefix

addition and parallel summation. Assume that processor j has a number x_j, $j = 1, 2, \ldots, p$, then by making parallel prefix addition, processor j will get a result $y_j = \sum_{i=1}^{j} x_i$, $j = 1, 2, \ldots, p$, and parallel summation enables each processor get the global sum $\sum_{i=1}^{p} x_i$. The communication requirement for the parallel prefix addition and parallel summation depends on the parallel architecture, and we will discuss it in the next two sections. After the parallel prefix addition and parallel summation, each processor has the global transition probabilities of all the possible tunneling events within its share of the network. All the processors then will generate a same random number, and this random number must fall into the probability ranges of the tunneling events at one processor. This processor will identify the tunneling event corresponding to this random number and then broadcast this event information to all the other processors, so that all processors can update their local information (i.e., charges $Q(i)$ and voltages $V(i)$) corresponding to this selected tunneling event. This updating can be done locally using Eq. (22).

5. MIMD implementation on nCUBE 2 parallel computer

5.1. Architecture of nCUBE 2

The nCUBE 2 system [28] is an MIMD machine design that can support up to 8,192 PEs. Each PE includes general-purpose 64-bit CPU with a 64-bit integer and 64-bit floating point execution unit, local memory, and a network of communication unit that includes 14 bidirectional DMA ports. Each PE operates independently. The nCUBE 2 employs a circuit-switched, partitionable hypercube network for inter-PE communications.

A mesh can be embedded in a hypercube using 2-D Gray code mapping [17]. Let the d-bit Gray code sequence of $p = 2^d$ integers be $Gray(d)$, i.e., $Gray(d) = (g_0^d, g_1^d, \ldots, g_{2^d-1}^d)$, where $g_i^d \in \{0, 1\}^d$ is the ith Gray code. The binary-reflected Gray code on d bits is recursively defined as follows [17]. $Gray(d+1) = (0g_0^d, 0g_1^d, \ldots, 0g_{2^d-1}^d, 1g_{2^d-1}^d, \ldots, 1g_1^d, 1g_0^d)$. Alternatively, $Gray(d+1) = (g_0^d0, g_0^d1, g_1^d1, g_1^d0, g_2^d0, g_2^d1, \ldots, g_{2^d-1}^d1, g_{2^d-1}^d0)$. As discussed in Section 4.1, the nano-network is divided into $P_x \times P_y$ regions, where $P_x = 2^k$ and $P_y = 2^{d-k}$. The region assignment to different processors on the hypercube is performed using 2-D Gray code mapping. The region (I, J) is assigned to the processor $P(I, J) = g_I \cdot 2^{d-k} + g_J$, $0 \leqslant I < P_x$, $0 \leqslant J < P_y$, where g_I is the ith Gray code. The assignment of regions in the partition in Fig. 4 to processors of the hypercube is shown in Fig. 6.

5.2. Parallel Gauss–Seidel on nCUBE 2

As discussed in Section 4.2.2, there is no communication overhead in the parallel Gauss–Seidel algorithm. But due to the different convergence rates, there is computation imbalance among the processors. Each processor needs to compute n/p vectors $\mathbf{w}_l$, each of which is of length $L = (2r+1)^2$ Assume that in processor k, it takes m_l^k iterations before vector $\mathbf{w}_l$ converges, $l = 1, 2, \ldots, n/p$ and $k = 0, 1, \ldots, p-1$. Also

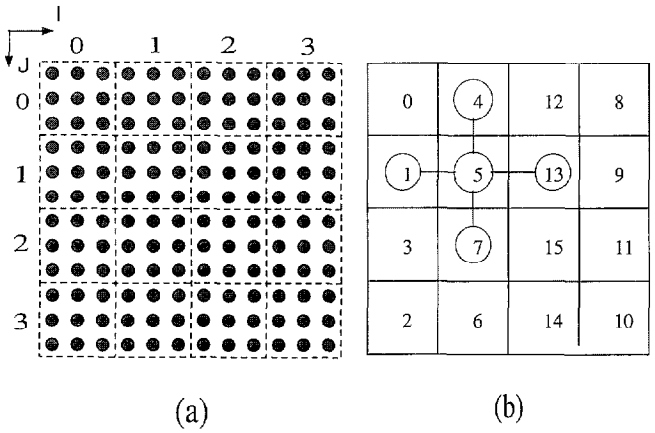


Fig. 6. Mapping the grid partitioning of the network on hypercube. (a) Grid partition the network. (b) Data distribution on the hypercube. The number inside each grid cell represents the address of the processor on the hypercube.

assume that the cost for updating one element is t_0, then the total serial computation load is

$$T(n) = \frac{nL}{p} t_0 \sum_{k=0}^{p-1} \sum_{l=1}^{N/p} m_l^k, \tag{26}$$

and the parallel execution time is

$$T_p(n, p) = \frac{nL}{p} t_0 \max_k \left\{ \sum_{l=1}^{N/p} m_l^k \right\}. \tag{27}$$

Therefore the speedup is

$$S(n, p) = \frac{T}{T_p} = \frac{\sum_{k=0}^{p-1} \sum_{l=1}^{N/p} m_l^k}{\max_k \left\{ \sum_{l=1}^{N/p} m_l^k \right\}}. \tag{28}$$

Fig. 7 depicts the run-time node utilizations of the parallel GS on nCUBE 2. We can see the loads are unbalanced among the processors in the parallel GS inversion, which is caused by the different convergence rates at different processors. Table 1 shows the measured speedups. The measured speedup is calculated in the following way. By using the nCUBE 2 profiling software we can measure the computation time T_i at each processor i. The sequential computation time is then $T = \Sigma_i T_i$ (because there is no redundant computation when implemented in parallel). We can also measure the parallel execution time T_p. Then the measured speedup is $S = T/T_p = \Sigma_i T_i / T_p$. Table 2 shows the parallel execution times of the parallel GS on nCUBE 2.

We have also implemented parallel Gauss-Jordan algorithm for inverting the *dense* matrix C on nCUBE 2. Fig. 8 is the run-time node utilization profile for $n = 32 \times 32$ and $p = 8$. It is seen that the computation and communication loads are quite balanced

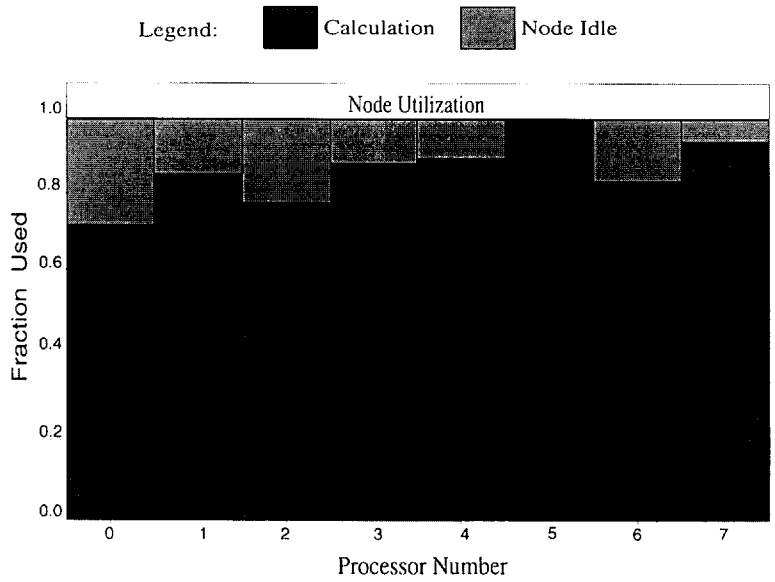


Fig. 7. Node utilizations profile of the parallel Gauss–Seidel inversion algorithm on nCUBE 2, $n = 32 \times 32$, $p = 8$. Nearest neighbor capacitance coupling is assumed, and the window size parameter $r = 10$, convergence tolerance $\varepsilon = 10^{-6}$.

Table 1
Speedups of the parallel Gauss–Seidel inversion algorithm on nCUBE 2, when inverting an $n \times n$ Poisson matrix using p processors ($r = 2 \log n$, $\varepsilon = 10^{-6}$)

$n = N_x \times N_y$	p			
	8	16	32	64
32×32	6.97	13.54	26.69	52.60
40×40	6.06	13.46	25.79	52.54
48×48	–	13.34	26.36	52.16
56×56	–	–	26.23	51.72
64×64	–	–	–	51.47

Table 2
Execution times of the parallel Gauss–Seidel inversion algorithm on nCUBE 2, when inverting an $n \times n$ Poisson matrix using p processors ($r = 2 \log n$, $\varepsilon = 10^{-6}$)

$n = N_x \times N_y$	p			
	8	16	32	64
32×32	5′1″	2′37″	1′21″	0′43″
40×40	15′12″	7′45″	3′57″	2′4″
48×48	–	19′0″	9′39″	5′0″
56×56	–	–	20′36″	10′36″
64×64	–	–	–	20′26″

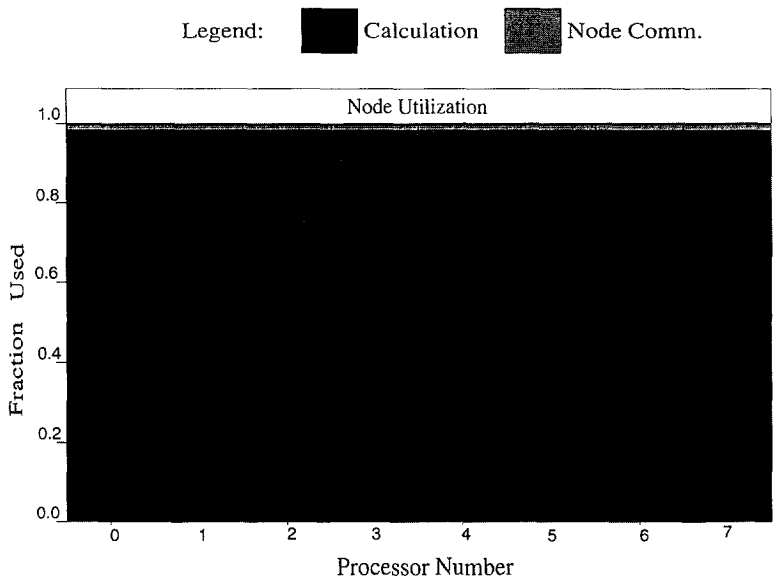


Fig. 8. Node-utilization profile of the parallel Gauss–Jordan inversion algorithm on nCUBE 2, $n = 32 \times 32$, $p = 8$. Random capacitance coupling between any two dots is assumed.

Table 3
Speedups of the parallel Gauss–Jordan inversion algorithm on nCUBE 2, when inverting an $n \times n$ *dense* matrix using p processors

$n = N_x \times N_y$	p			
	8	16	32	64
32×32	7.86	15.54	30.69	60.32
40×40	7.88	15.61	30.91	61.13
48×48	–	15.64	31.05	61.63
56×56	–	–	31.07	61.74
64×64	–	–	–	61.82

Table 4
Execution times of the parallel Gauss–Jordan inversion algorithm on nCUBE 2, when inverting an $n \times n$ *dense* matrix using p processors

$n = N_x \times N_y$	p			
	8	16	32	64
32×32	15′21″	7′43″	3′54″	1′59″
40×40	58′31″	29′19″	14′44″	7′27″
48×48	–	87′31″	44′3″	22′11″
56×56	–	–	111′11″	55′51″
64×64	–	–	–	125′11″

among the processors. Table 3 shows the measured speedups and Table 4 shows the parallel execution times. The cases in the tables which are labeled with "–" were unable to be computed because of insufficient per-processor memory on nCUBE 2.

5.3. Parallel Monte Carlo on nCUBE 2

Usually, we need to carry out simulations of a particular network for a set of parameters, e.g., potential values $\{V_1, V_2, \ldots, V_J\}$. For each parameter value, multiple simulations, say L, need to be carried out and the system performance (e.g., current value) over these L simulations is averaged. That is, $I(V_j) = (1/L)\sum_{l=0}^{L} I(V_j^{(l)})$, $j = 1, 2, \ldots, J$, where $I(V_j^{(l)})$ is the current value corresponding to the potential value V_j in the lth simulation. Thus totally $m = JL$ simulations are needed.

One way is to carry out these m simulations one by one, each for a specific parameter, that is

```
for j = 1 to J
  for l = 1 to L
```

$$s_0(V_j^{(l)}) \to s_1(V_j^{(l)}) \to s_2(V_j^{(l)}) \to \cdots \to s_\nu(V_j^{(l)})$$

Another way is to carry out the JL simulations in parallel, that is, to make the transitions $s_i \to s_{i+1}$ for all the parameters $\{V_j^{(l)}\}$, $j = 1, 2, \ldots, J$, $l = 1, 2, \ldots, L$.

$$s_0\left(\{V_j^{(l)}\}\right) \to s_1\left(\{V_j^{(l)}\}\right) \to s_2\left(\{V_j^{(l)}\}\right) \to \cdots \to s_\nu\left(\{V_j^{(l)}\}\right)$$

Essentially in the second approach, the two for loops that appeared in the first approach are moved inside the loop on the state transitions, i.e., within the process of $s_i \to s_{i+1}$. In terms of computation time, the two approaches are the same. However, if one considers the communication costs, then the two approaches have different costs. In the first approach there are $m\nu$ communications each of message length, say 1. In the second approach, there are ν communications, each of message length m. The total communication overhead in the first approach is then $T_{comm}^1 = m\nu t_s + m\nu t_w$; and for the second approach, the total communication overhead is $T_{comm}^2 = \nu t_s + m\nu t_w$. For a class of message-passing machines, such as nCUBE 2, usually $t_s \gg t_w$, e.g., for nCUBE 2, $t_s = 180$ μs and $t_w = 3$ μs [23]. Therefore, it is very important to avoid message startup time by coalescing individual messages whenever possible. Hence in the second approach, data structures are organized so that successive communications can be changed into a single communication of concatenated data, thus mitigate the communication startup overhead. Of course, the price we paid for carrying out m simulations concurrently, is that we need to keep the local information on each processor (i.e., Q, V, ΔE, etc.) for all these m simulations, thus increasing the memory requirement.

There are three main steps for each iteration in the Monte Carlo simulation algorithm, and these are summarized as follows.

(1) *Calculation of the energy differential and transition rate for each possible tunnel event.* At each dot, there are at most possible four tunnel event, i.e., a electron charge can tunnel toward the four neighbor dots. By Eq. (25) the energy differentials ΔE's can be calculate locally at each processor, except for those "cross-border" tunnel events, as shown in Fig. 9, where nearest neighbor interprocessor communication is needed to get the voltage values of the dots on the other side of the border. Suppose that the $\sqrt{n} \times \sqrt{n}$

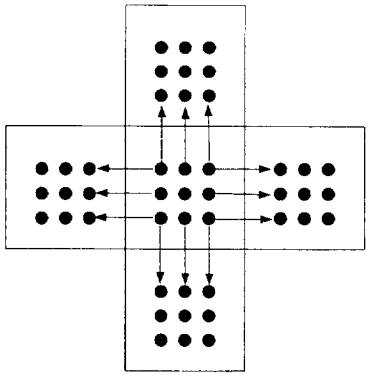


Fig. 9. Cross-border tunnel events for which the calculation of ΔE involves near-neighbor interprocessor communication. Each processor has 9 quantum dots in this example.

nano-array is partitioned into $\sqrt{p} \times \sqrt{p}$ regions, and m simulations are carried out in parallel, then the communication overhead at each processor for this step in each iteration is $(t_s + m\sqrt{n/p}\, t_w)$, since nCUBE 2 is capable of all-port two-way communication. The transition rates Γ's can be calculated locally at each processor using Eq. (6) or (7).

(2) *Calculation of the transition probability of each tunnel event.* The local tunneling rate of each event i is calculated as $S_k(i) = \sum_{j=1}^{i} \Gamma_k(j)$, where $\Gamma_k(j)$ is the transition rate of event j at processor k. The global transition probability of this event is defined as

$$P_k(i) = \frac{\sum_{l=0}^{k-1} S_l(\mu_l) + S_k(i)}{\sum_{l=0}^{p-1} S_l(\mu_l)} = \frac{\sum_{l=0}^{k} S_l(\mu_l) + S_k(i) - S_k(\mu_k)}{\sum_{l=0}^{p-1} S_l(\mu_l)}, \tag{29}$$

where μ_l is the total number of tunnel events at processor l. Thus each processor need to get $\sum_{l=0}^{k} S_l(\mu_l)$ and $\sum_{l=0}^{p-1} S_l(\mu_l)$. For simplicity, let $p_l = S_l(\mu_l)$, then each processor is to get $s_k = \sum_{l=0}^{k} p_l$ and $s = \sum_{l=0}^{p-1} p_l$. s_k can be calculated by using parallel prefix addition, and s can be calculated by using parallel summation. Both can be carried out in log p, where each step corresponds to a one-to-one permutation. Fig. 10 shows the process of parallel prefix addition and parallel summation for $p = 8$. Let the binary representation of l, $0 \leqslant l \leqslant p-1$ be $l_{d-1} l_{d-2} \cdots l_1 l_0$, where $d = \log p$. The pseudo code for calculating s_k and s is as follows:

$s_l = p_l$, $s = p_l$
for $k = 0$ to $d - 1$
  $t = s$
  send (s) to $l_{d-1} \cdots l_k \cdots l_0$
  receive (s) from $l_{d-1} \cdots l_k \cdots l_0$
  if $(l_{d-1} \cdots l_k \cdots l_0 > l_{d-1} \cdots l_k \cdots l_0)$ $s_l = s_l + s$
  $s = s + t$

Because m simulations proceed in parallel, the communication overhead for the parallel prefix sum and parallel summation is $(t_s + t_w m) \log p$.

(3) *Determination of a tunnel event and updating of the local charge and voltage information at each dot.* After calculating the transition probabilities of all the events,

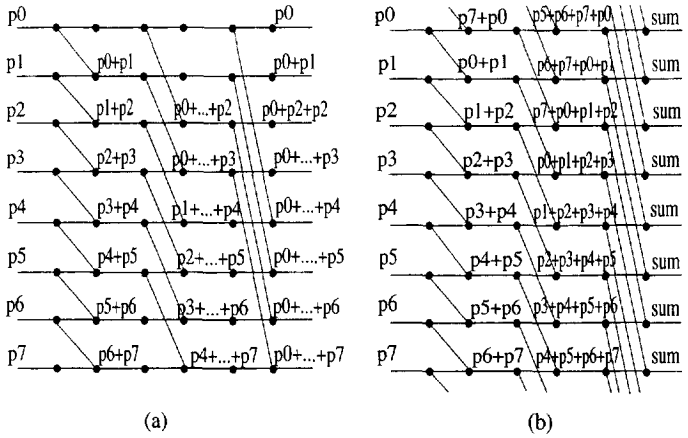


Fig. 10. Recursive doubling on a 3-cube. (a) Parallel prefix sum. (b) Parallel summation.

for each simulation, a same random number r is generated in all the processors. One of the processor, say processor k, will find this r as corresponding to an event i in its region, i.e., $P_k(i-1) \leqslant r \leqslant P_k(i)$. This tunnel event information needs to be broadcasted to all the other processors by processor k. Since there are m concurrent simulations, we can use the parallel summation to broadcast all the m events efficiently in the following manner. The processor k stores the event i information into variables `src` and `dest`, meaning event i corresponds to an electron tunneling from dot `src` to dot `dest`, where both `src` and `dest` are the global indexes of the dots. All the other processors set their variables `src` and `dest` for that particular simulation to 0. Since there are m simulations, `src` and `dest` are vectors of length m. After choosing events for all these m simulations, the global summation is used to get global sums of `src` and `dest` at each processor. In this way all the processors get all the m events information. The communication overhead of this step is therefore $(t_s + t_w m) \log p$. Based on the events information `src` and `dest`, all the processors then update the dot charge and voltage values in the region for all the m simulations, using Eq. (22).

Suppose m simulations are carried out and ν iterations are required for each simulation. Then the total serial computation load is

$$T(n) = m\nu n t_1, \tag{30}$$

where t_1 is the computation cost at each dot in each iteration in one simulation, including the time for calculating the energy differentials, the transition rates and updating the local potentials. The total communication overhead at each processor is $\nu[t_s + m\sqrt{n/p}\, t_w + 2(t_s + t_w m) \log p]$. Therefore the parallel execution time is

$$T_p(n, p) = \frac{m\nu n t_1}{p} + \nu\left[\left(t_s + t_w m\sqrt{\frac{n}{p}}\right) + 2(t_s + t_w m) \log p\right]. \tag{31}$$

The speedup is then

$$S(n, p) = \frac{T}{T_p} = \frac{n t_1 p}{n t_1 + t_s[(2 \log p + 1)p/m] + t_w\left(\sqrt{np} + 2p \log p\right)}. \tag{32}$$

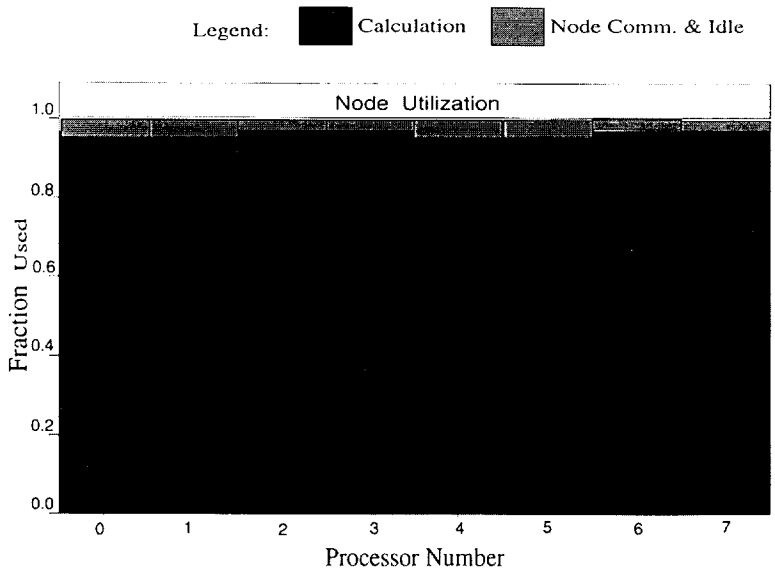


Fig. 11. Node utilizations profile of the parallel Monte Carlo algorithm, $N = 32 \times 32$, $p = 8$.

Fig. 11 depicts the run-time node utilizations of the parallel MC algorithm on nCUBE 2. The slight computation load differences among the processors is caused by the fact that in some processors there are fewer possible tunnel events than in the other processors, e.g., in Fig. 2, for the dots on the upper border of the nano-array, there are no upward tunnel events. Table 5 shows the measured speedups, and Table 6 shows the actual parallel execution times.

5.4. Scalability analysis

The scalability of an algorithm on a parallel architecture is a measure of its capability to effectively use an increasing number of processors. *Isoefficiency* is one of the metrics for characterizing the scalability of different parallel algorithms and architectures [23], which is defined as the rate of change of problem size as a function of number of

Table 5
Speedups of the parallel Monte-Carlo simulation algorithm on nCUBE 2 (100 simulations, 5000 iterations for each simulation)

$N = N_x \times N_y$	p			
	8	16	32	64
32×32	7.84	15.51	30.07	59.05
40×40	7.90	15.62	30.09	60.11
48×48	–	15.66	30.14	60.84
56×56	–	–	31.17	61.39
64×64	–	–	–	61.42

Table 6
Execution times of the parallel Monte-Carlo simulation algorithm on nCUBE 2 (100 simulations, 5000 iterations for each simulation)

$N = N_x \times N_y$	p			
	8	16	32	64
32×32	143′2″	74′24″	40′27″	22′38″
40×40	213′42″	111′51″	61′08″	33′16″
48×48	–	151′6″	79′36″	44′59″
56×56	–	–	110′22″	59′21″
64×64	–	–	–	80′37″

processors to maintain a fixed parallel efficiency. An algorithm that requires a smaller change in problem size to obtain fixed efficiency is considered more scalable. The efficiency E of a parallel algorithm is defined to be S/p. By substituting the expressions for speedups for the different algorithms discussed in the previous sections, we can derive the following statements:

- For the parallel Gauss–Seidel algorithm on the hypercube, since there is no communication overhead, efficiency is determined by the numerical property of the data. Ideally, increasing p will not affect the parallel efficiency.
- For the parallel Monte Carlo algorithm on the hypercube, fixed efficiency can be maintained if n is increased as $O(p \log p)$.

Therefore our parallel MIMD formulation on the hypercube is highly scalable. This is also demonstrated by the experiment results, where near-linear speedups are obtained for all the cases.

6. SIMD implementation on MasPar MP-1 parallel computer

6.1. Architecture of MasPar MP-1

The MasPar MP-1 [27] is a massively parallel computer with 16,384 PEs arranged in a 128×128 array. Each PE has a 4-bit ALU and 16 Kbytes of local memory. The MP-1 supports two different networks for interprocessor communication, the *xnet* and the *global* router. Communicating with xnet is fast but a data transfer can take place only between PEs that lie on the same horizontal, vertical or diagonal row as shown in Fig. 12. Communications which use the global router are slower, but the router can support any arbitrary PE-to-PE communication.

We chose the MasPar MP-1 for simulating larger nanoelectronic networks, i.e., in the range from 100×100 to 1000×1000 quantum dots, because our parallel algorithm can efficiently utilize tens of thousands of PEs. In addition, our implementation benefits from fast scan and reduce operation, which are characteristic of the target architecture. Lastly, xnet communication on the MP-1 is efficient for our application since most of the communication is restricted to near-neighbor PEs.

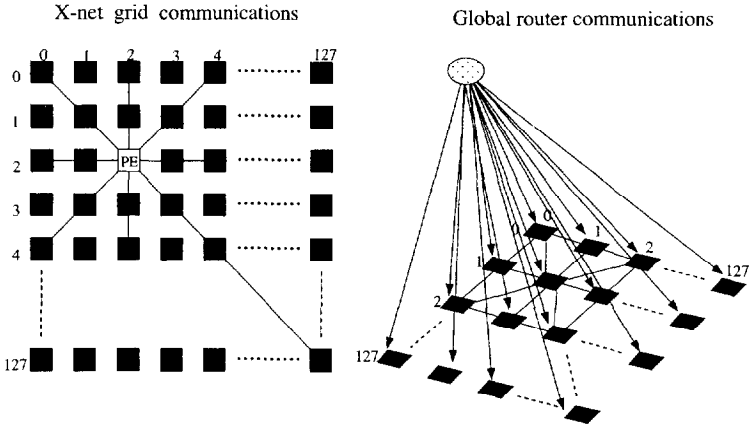


Fig. 12. Interprocessor communication on the MasPar MP-1.

6.2. Mapping the Gauss–Seidel algorithm onto MasPar MP-1

Suppose that we want to simulate a network of $n = N \times N$ quantum dots using p processors, then each processor has n/p dots. And for each dot l, there is a synaptic weight vector $\mathbf{w}_l$. We assume that the synaptic weight between dot l and any dot outside a $(2r+1)\times(2r+1)$ window centered at dot l is zero. Therefore $\mathbf{w}_l$ is of length $(2r+1)^2$.

Fig. 13 illustrates the SIMD calculation of $\mathbf{w}_l$'s on MP-1. At a particular time step, all the processors are calculating the weight value w_{lj} between a dot l in its share region of

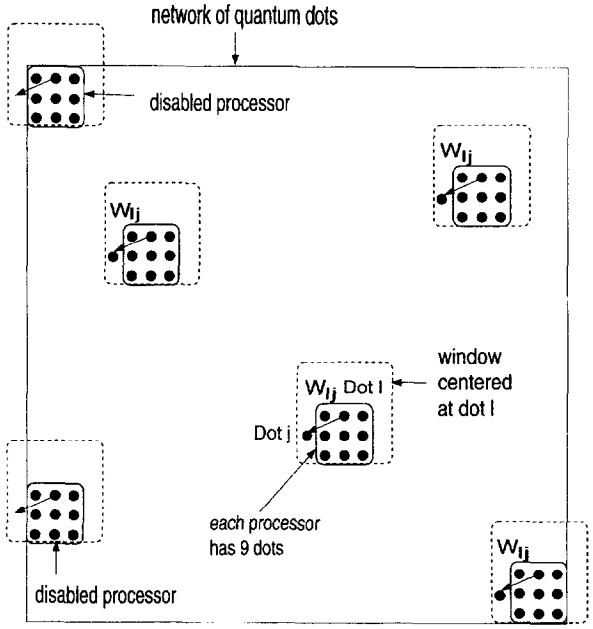


Fig. 13. SIMD implementation of Gauss–Seidel algorithm on MasPar MP-1.

Table 7
Parallel execution times of the SIMD Gauss-Seidel algorithm on MP-1 ($\varepsilon = 10^{-6}$, $r = 3$)

Problem size $n = N_x \times N_y$	Execution time T_p (seconds)
128×128	0.77
256×256	2.93
384×384	6.48
512×512	11.15
640×640	17.87
768×768	25.47
896×896	34.62
1024×1024	42.74

the network, and a dot j inside the window centered at dot l. If the dot j is outside the range of the network (i.e., dot j does not exist), then the corresponding processor is disabled at this time step. No interprocessor data exchange is needed during the Gauss–Seidel iteration, as discussed in Section 4.2.2. However, at the end of each iteration, each processor tests if all the $\mathbf{w}_l$'s in its region have met the convergence condition. The control unit (CU) then collects this information by using the global OR router. If convergence conditions are met at all the processors, then the computation of W is over; otherwise the iteration continues on those processors which still have unconverged $\mathbf{w}_l$'s, and the rest of the processors are disabled.

Suppose that the total number of iterations before all the $\mathbf{w}_l$ vectors converge is K, then the algorithmic complexity of our implementation is $O(K(2r+1)^2N^2/p) = O(K(2r+1)^2 n/p)$. The only communication overhead is the global OR operation after each iteration, but his overhead is negligible compared with the cost of computation, since the global OR router on MP-1 is very fast. Table 7 shows the parallel execution time of the Gauss–Seidel algorithm on MP-1, where the tolerance for convergence $\varepsilon = 10^{-6}$, and window size $r = 3$, Fig. 14 is the corresponding plot.

6.3. Mapping the Monte Carlo simulation algorithm onto MasPar MP-1

Unlike in the MIMD implementation on nCUBE 2, the m simulations are carried out one by one for our SIMD implementation on MP-1. This is efficient because the communication startup cost on MP-1 is negligible, while the per-processor memory is very limited.

While the three computation steps for each iteration in the SIMD Monte Carlo algorithm are the same as discussed in Section 5.3, the communication mechanism for each step on the MP-1 mesh is different from that on nCUBE 2.

1. The calculation of the energy differentials involves near-neighbor interprocessor communication on the mesh, which can be done using `xnet`. This consists of four steps of near-neighbor data transfer along the four directions (east, west, north and south), and the data size for each transfer is $\sqrt{n/p}$.
2. The calculation of the transition probability of all the tunnel events involves parallel prefix and parallel summation on the mesh. This is done by using the optimized routines `reduceAdd()` and `scanAdd()` from MasPar's program-

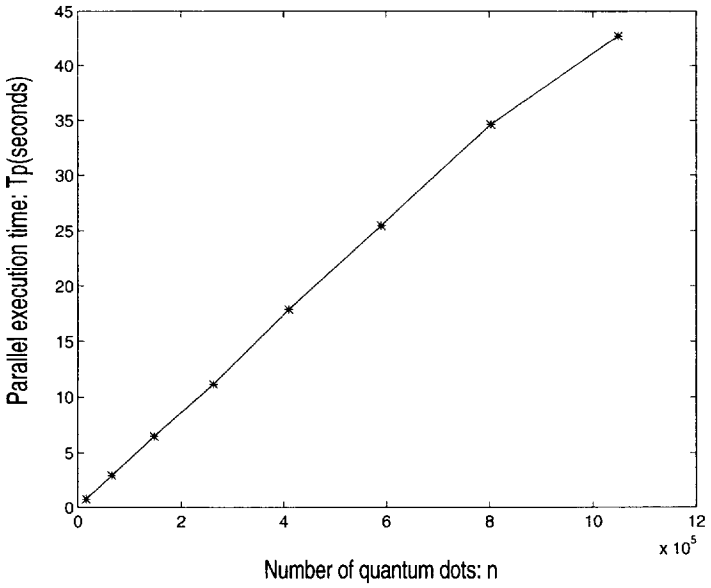


Fig. 14. Parallel execution times of the SIMD Gauss–Seidel algorithm on MP-1.

ming library. The architectural complexity of summing the partial results is worse than the algorithmic complexity because the MP-1's `reduceAdd()` and `scanAdd()` require $O(\log\ p)$ addition steps but the communication cost is $O(p)$. However, the experimental results indicate that communication costs are dominated by the computation costs, so that the measured complexity approximates algorithmic complexity.

3. The information of the selected event is collected by the CU using the global OR tree and broadcast to all the processors along with the next instruction. The global OR tree is the fastest way on MP-1 to broadcast data from one processor to all the other processors.

Table 8
Parallel execution times of the SIMD Monte Carlo algorithm on MasPar MP-1 for one simulation ($\varepsilon = 10^{-6}$, $r = 3$, $K = 10{,}000$)

Problem size $n = N_x \times N_y$	Execution time T_p (seconds)
128×128	65.6
256×256	127.4
384×384	220.4
512×512	342.5
640×640	492.4
768×768	681.4
896×896	894.7
1024×1024	1146.7

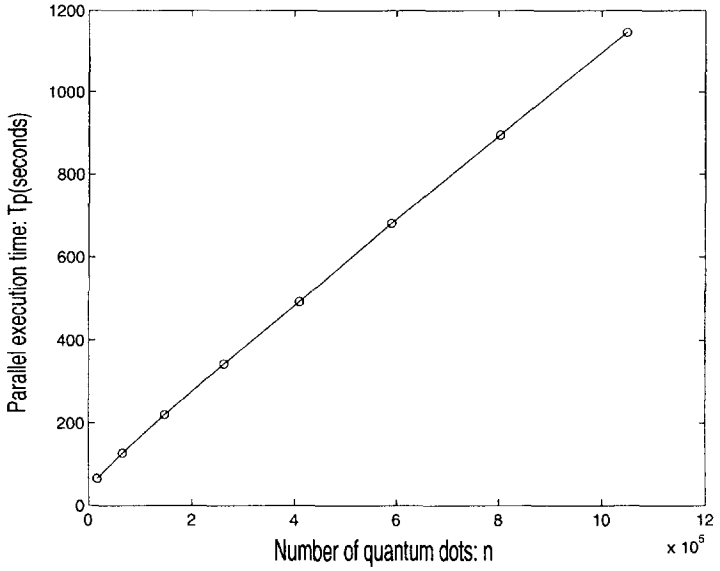


Fig. 15. Parallel execution times of the SIMD Monte Carlo algorithm on MasPar MP-1.

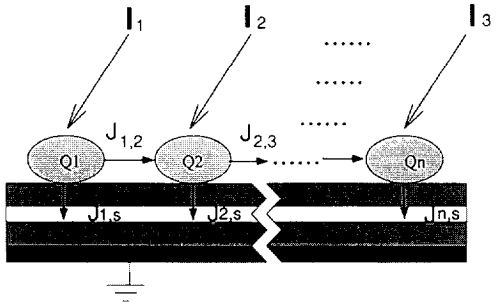
Suppose that totally K iterations are needed for one simulation, then the computation complexity of the parallel Monte Carlo algorithm is $O(KN^2/p) = O(Kn/p)$. Table 8 shows the parallel execution times on MasPar MP-1 for one simulation, where the number of iterations $K = 10{,}000$. Fig. 15 is the corresponding plot. From the plot we can see that the measured complexity is close to the computation complexity because the communication cost is dominated by the computation cost.

To determine how well suited our algorithm is to the architecture of the MP-1, we timed the various portion of our algorithm to see how much time they spent computing versus communicating. Our timings indicate that less than 5% of the execution time is spent in communication overhead, so the bandwidth of the xnet and the global router are not limiting factors. In fact, we spend roughly the same amount of time assigning variables and accessing arrays in local PE memory as communicating, in part because every group of 16 PEs in the MP-1 has to share a single data bus.

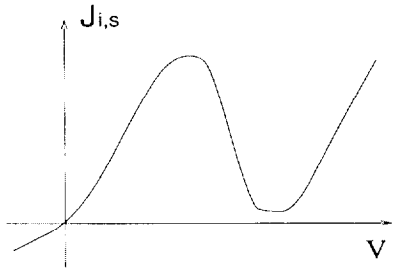
7. Current-driven networks

In this section, we consider a current-driven network of nanoscale metallic dots, as shown in Fig. 16(a) [6]. The systems we shall study will consist of arrays of dots, which have been deposited on a layer of material with non-ohmic electrical characteristics, which in turn has been grown on a conductive substrate. There are no electrodes at the boundary of the array. Instead, the interior nodes of the array can be directly contacted and supplied with charge and energy. We will consider networks with the steady flow of electrons.

In this network, single-electron effects and resonant tunneling are allowed to coexist. The dynamics of the tunnel events between two neighboring dots is the same as



(a) Current-driven network



(b) Substrate transport function

Fig. 16. (a) An array of current-driven quantum dots. (b) A non-monotone substrate nonlinearity of the kind shown here, is the minimal condition for the realization of collective effects.

described in the previous subsection. However, we use a nonlinear function to describe transport between each dot and the substrate. It has been shown that non-monotonic nonlinearities are necessary to obtain nontrivial collective activity in these networks [6]. Recently, it has been shown that novel electric field quantization effects can prevail in slim semiconductor superlattices when single-electron effects coexist with non-monotonic nonlinear tunnel rates between wells [22]. In the following we will use, nonlinearities of the kind shown in Fig. 16(b) to account for the discrete tunneling of electrons into the substrate.

The single-electron tunnel rate through the junction is

$$\Gamma = \frac{J(-\Delta E/q)}{q[1-\exp(\Delta E/kT)]}, \tag{33}$$

where $J(V)$ is the current-voltage characteristics of the junction, e.g., as shown in Fig. 16(b). A strategy for analyzing this network might consist in writing a Kirchoff current balance condition for each dot i,

$$\frac{d}{dt}q_i = -(dq_i/dt)_{sub} - \sum_{j\in n,n}(dq_i/dt)_j + Idi, \tag{34}$$

where the differential terms on the right-hand side denote the discrete transfer of electrons to the substrate, and to the nearest neighbors. This equation is only symbolic, and can not be treated as a proper differential equation since the dot charges belong to the discrete set,

$$q_i(t) = \int^t I_i(\tau)\,d\tau + nq, \quad n \in \{\pm 1, \pm 2, \ldots\}, \tag{35}$$

where we have made explicit the fact that the continuous delivery of charge by a current source is punctuated by discrete charge transfer through tunnel junctions. Single electron tunneling is a stochastic process which can be described in terms of a master equation, written for the probability $\rho(Q, t)$ that the array is in the charge state Q at the time t [13,12],

$$\left[\frac{\partial}{\partial t} + \sum_{i=1}^{N} I_i \frac{\partial}{\partial q_i}\right] \rho(Q, t) = \sum_m \Gamma_m(Q - q_m)\,\rho(Q - q_m, t) - \rho(Q, t) \sum_n \Gamma_n(Q, t), \tag{36}$$

where the two summations on the right-hand side take stock of the single particle tunnel events which will either bring, or take away the system from the charge state Q. This master equation [3] will require a numerical approach due to the lack of a regular method to solve systems of this kind which can have a very large state space even for a very small number of dots. The best procedure then is to develop a Monte Carlo technique which mimics in detail the physics of single-electronics.

Monte Carlo simulation technique. Next we will outline a Monte Carlo simulation technique [4,13] for the simulation of a current biased network of dots, shown in Fig. 16(a). The state of the system of dots is fully described at time t by the vector of dot charges $Q(t)$. Over a very small time interval δt, each dot is now, assumed to be delivered with an increment of charge $\delta q_i = I_i \delta t$ by a current source I_i. The new charge state then is $Q(t + \delta t) = Q(t) + \delta t I$, where $I = [I_1, \ldots, I_N]$. Next, the entire system is swept, and a vector of numbers $\{\Delta E^m\}$ corresponding to the energy dissipated as a result of tunnel events (indexed by the superscript m), between each dot and its nearest neighbors, as well as between each dot and the substrate are accumulated. From the vector of dissipated energies, a vector of tunnel rates $\{\Gamma^m\}$ is then generated. A further vector of cumulative tunnel rates is then generated by replacing each element of the vector $\{\Gamma^m\}$ with the sum of all previous rates

$$S^m = \sum_{k=1}^{m} \Gamma^k. \tag{37}$$

The probability that any one of these tunnel events should proceed over a very small time interval δt is then determined by calculating,

$$P(t + \delta t) = e^{-\delta t S^l}, \tag{38}$$

where the total tunnel rate S^l is the last element in the vector of cumulative rates $\{S^m\}$. A random number r_π, distributed uniformly on the unit interval is now drawn, and a

decision to carry out a tunnel is then made if the condition $P < r_\pi$ is met. This prescription conforms to the intuitive idea that the larger the total tunnel rate, the greater the frequency with which the events are carried out. If a decision to carry out a tunnel event is made, then a particular event needs to be selected from among all the events which were considered. This selection is made with probability proportional to the individual tunnel rates. This is numerically implemented by picking again a random number r_ϕ distributed uniformly on the unit interval, and then selecting the tunnel event with the lowest index i, which meets the condition $(S^i/S^l) > r_\phi$. The above procedure is then repeated until convergence, or until adequate numerical evidence has been accumulated.

From the above descriptions we can see that the Monte Carlo technique for simulating the current-driven networks is very similar to that for the voltage-driven networks. Therefore the adaptation of the parallel simulation algorithms developed in this paper to the current-driven networks is straightforward.

8. Concluding remarks

Continuing advances in the miniaturization of electron devices have made possible the fabrication of nanoelectronic devices with feature sizes in the 1–100 nm range. Besides providing a framework for understanding the physics of nanoscale structures, realistic computer modeling constitutes a valuable tool for designing quantum devices that enables the development of a nanoelectronic technology along with the associated advances in fabrication technologies. In this paper, we have presented massively parallel algorithms for simulating large-scale nanoelectronic networks based on the single-electron tunneling effect, which is arguably the quantum effect of greatest significance to nanoelectronic technology. We have carried out a MIMD implementation on a 64-processor nCUBE 2, and a SIMD implementation on a 16,384-processor MasPar MP-1. Our theoretical performance analysis and experimental timing results indicate that the parallel algorithms are highly efficient and scalable. This work demonstrates the usefulness of massively parallel high performance computers for nanoelectronic research. Moreover, we are able to simulate large-scale nanoelectronic structures within a reasonable time period, which would be impractical on conventional work stations.

References

[1] D.V. Averin, A.N. Korotkov and K.K. Likharev, Theory of single electron charging of quantum wells and dots, *Phys. Rev. B* 44 (1991) 6199–6211.

[2] D.V. Averin and K.K. Likharev, Coulomb blockade of single-electron tunneling and coherent oscillations in small tunnel junctions, *J. Low Temp. Phys.* 62 (3) (1986) 345–373.

[3] D.V. Averin and K.K. Likharev, Single electronics: A correlated transfer of single electrons and cooper pairs in systems of small tunnel junctions, in: B.L. Altshuler, P.A. Lee and R.A. Webb, eds., *Mesoscopic Phenomena in Solids* (Elsevier, Amsterdam, 1991) 173–271.

[4] N.S. Bakhvalov, G.S. Kazachca, K.K. Likharev and S.I. Serdyukova, Single-electron solitons in two-dimensional tunnel structures, *Sov. Phys. JETP* 68 (3) (1989) 581–587.

[5] N.S. Bakhvalov, G.S. Kazachca, K.K. Likharev and S.I. Serdyukova, Statics and dynamics of single-electron solitons in two-dimensional arrays of ultrasmall tunnel junctions, *Physica B.* 173 (1991) 319–328.
[6] P. Balasingam and V.P. Roychowdhury, Nanoelectronic functional devices, Tech. Rept. TR-EE 94-24, School of Electrical Engineering, Purdue University, 1994.
[7] S. Bandyopadhyay, B. Das and A.E. Miller, Supercomputing with spin polarized single electrons in a quantum coupled architecture, *Nanotechnology* (1994), to appear.
[8] S. Bandyopadhyay, V.P. Roychowdhury and X. Wang, Computing with quantum dots: Novel architectures for nanoelectronics, *Physics of Low Dimensional Structures* 8/9 (1995) 29–81.
[9] D.S. Bethune et al., Atoms in carbon cages: The structure and properties of endohedral fullerenes, *Nature* 366 (1993) 123–128.
[10] R.G. Osifchin et al., Synthesis of a quantum dot superlattice using molecularly linked metal clusters, *Superlattice and Microstructures* 18 (4) (1995) 283–289.
[11] L.J. Geerligs, Charge quantisation effects in small tunnel junctions, in: J.H. Davies and A.R. Long, eds., *Physics of Nanostructures* (Scottish Universities Summer School in Physics Publications, Edinburgh, 1992) 171–204.
[12] U. Geigenmüller and G. Schön, Single electron effects and Bloch oscillations in normal and superconducting tunnel junctions, *Physica B* 152 (1988) 186–202.
[13] U. Geigenmüller and G. Schön, Single-electron effects in arrays of normal tunnel junctions, *Europhys. Lett.* 10 (1989) 765–770.
[14] K.K. Gullapalli, D.R. Miller and D.P. Neikirk, Simulation of quantum transport in memory-switching double-barrier quantum-well diodes, *Phys. Rev. B* 49 (2) (1994) 2622–2628.
[15] K. Hess, J.P. Leburton and U. Ravaioli, *Computational Electronics* (Kluwer, Boston, MA, 1991).
[16] P.G. Hipes and A. Kuppermann, Gauss–Jordan inversion with pivoting on the Caltech Mark II hypercube, in: *Proc. 3rd Conf. on Hypercube Concurrent Computers and Applications* (ACM, 1988) 1621–1634.
[17] S.L. Johnsson and C-T Ho, Optimum broadcasting and personalized communication in hypercubes, *IEEE Trans. Comput.* 38 (9) (1989) 1249–1268.
[18] M.A. Kastner, The single-electron transistor, *Rev. Mod. Phys.* 64 (3) (1992) 849–858.
[19] M.A. Kastner, Artifical atoms, *Physics Today* (January 1993) 24–31.
[20] D.P. Kern, Nanoelectronic devices: Opportunities and challenges, in: *Proc. IEEE Internat. Electron Devices Meeting*, Washington, DC (IEEE, 1993), p. 7.
[21] H. Körner and G. Mahler, Optically driven quantum networks: Applications in molecular electronics, *Phys. Rev. B* 48 (4) (1993) 2335–2346.
[22] A.N. Korotkov, D.V. Averin and K.K. Likharev, Single-electron quantization of electric field domains in slim semiconductor superlattices, *Appl. Phys. Lett.* 62 (1993) 3282–3284.
[23] V. Kumar, S. Shekhar and M.B. Amin, A scalable parallel formulation of the backpropagation algorithm for hypercube and related architectures, *IEEE Trans. Parallel Distributed Systems* 5 (10) (1994) 1073–1089.
[24] S. Lloyd, A potentially realizable quantum computer, *Science* 261 (1993) 1569–1571.
[25] J.H. Luscombe, Current issues in nanoelectronic modelling, *Nanotechnology* 4 (1993) 1–20.
[26] J.H. Luscombe and W.R. Frensley, Models for nanoelectronic devices, *Nanotechnology* 1 (1990) 131–140.
[27] MasPar Corporation, *MasPar System Overview* (1990).
[28] nCUBE Corporation, *nCUBE 2 Processor Manual* (1990).
[29] J.M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems* (Plenum Press, New York, 1988).
[30] M.A. Reed, Quantum dots, *Scientific American* (1993) 118–123.
[31] V.P. Roychowdhury, X. Wang and S. Bandyopadhyay, Artificial quantum solids that compute: Quantum-mechanical logic gates and neuromorphic networks, in: *Proc. 8th Internat. Conf. on Superlattices*, Cincinnati, OH, 1995.
[32] C. Schonenberger and H. van Houten, Single-electron tunneling observed at room temperature observed by scanning-tunneling microscopy, in: *Proc. Internat. Conf. on Solid State Devices and Materials* (1992), p. 726.
[33] A.D. Stone, Theory of coherent quantum transport, in: J.H. Davies and A.R. Long, eds., *Physics of Nanostructures* (Scottish Universities Summer School in Physics Publications, Edinburgh, 1992) 65–100.