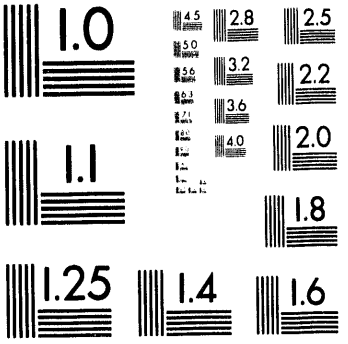

1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

1 of 1

CONF-931115--12

THE QUANTUM STRUCTURE OF MATTER GRAND CHALLENGE PROJECT: LARGE-SCALE 3-D SOLUTIONS IN RELATIVISTIC QUANTUM DYNAMICS

J. C. Wells[1,2], V. E. Oberacker [1,2], A. S. Umar[1,2], C. Bottcher[1], M. R. Strayer[1]
J. Drake[1], and R. Flanery[1]

[1]Oak Ridge National Laboratory
Oak Ridge, TN 37831

[2]Department of Physics & Astronomy
Vanderbilt University
Nashville, TN 37235

to be published in Proceedings of

Supercomputing '93 Conference
Portland, Oregon
November 15-19, 1993

The Quantum Structure of Matter Grand Challenge Project: Large-scale 3-D Solutions in Relativistic Quantum Dynamics

J. C. Wells[1,2], V. E. Oberacker[1,2],
A. S. Umar[1,2], C. Bottcher[1], M. R. Strayer[1]

J. Drake and R. Flanery

[1] Physics Division,
Oak Ridge National Laboratory,
Oak Ridge, TN 37831-6373; and
[2] Department of Physics & Astronomy,
Vanderbilt University, Nashville, TN 37235

Engineering Physics &
Mathematics Division,
Oak Ridge National Laboratory,
Oak Ridge, TN 37831

Abstract

We describe the numerical methods used to solve the time-dependent Dirac equation on a three-dimensional Cartesian lattice. Efficient algorithms are required for computationally intensive studies of nonperturbative relativistic quantum dynamics. Discretization is achieved through the lattice basis-spline collocation method, in which quantum-state vectors and coordinate-space operators are expressed in terms of basis-spline functions on a spatial lattice. All numerical procedures reduce to a series of matrix-vector operations which we perform on the Intel iPSC/860 hypercube, making full use of parallelism. We discuss our solutions to the problems of limited node memory and node-to-node communication overhead inherent in using distributed-memory, multiple-instruction, multiple-data stream parallel computers.

1 Introduction

In this paper, we focus on the time-dependent Dirac equation in three space dimensions and its lattice representation on a distributed-memory hypercube multicomputer. Over the past several years, we have developed a new approach to strong-field relativistic quantum dynamics which combines advanced techniques for solving boundary-value differential equations with supercomputer technology.[1] The Dirac equation is one of the most fundamental equations of nature: it is the relativistic analogue of the Schrödinger equation and describes the quantum dynamics of fermions, i.e. spin-1/2 elementary particles such as leptons and quarks. The lepton family consists of three generations: the well-known electron and its associated neutrino, the muon and muon-neutrino, and the tau-lepton and tau-neutrino. Similarly, the quarks come in three different generations: up/down, charmed/strange, and top/bottom. In the following, we will consider the dynamics of leptons only; however, we believe that our computational methodologies will also have application to the quark sector.

1.1 Physics background

During the collision of highly charged heavy ions at velocities near the speed of light, extremely large time-dependent electromagnetic fields are produced which lead to a variety of effects. These electromagnetic fields are up to 10 billion times stronger than today's strongest laser fields. One of the most interesting effects of these fields is the sparking of the vacuum, i.e. production of matter-antimatter pairs from empty space as the heavy ions pass near each other. In particular, at Oak Ridge National Lab (ORNL), we have studied the vacuum production of lepton pairs, i.e. electrons, muons, and tauons, as well as other exotic particles. Collisions such as these are currently performed at experimental facilities around the world such as Brookhaven National Laboratory on Long Island, and CERN, the European Center for Nuclear Research, in Geneva, Switzerland. During the last nine years, much study has been devoted to the problem of vacuum production of lepton-pairs in anticipation of new experimental opportunities at the Relativistic Heavy-Ion Collider (RHIC), currently under construction at Brookhaven. This facility will provide colliding beams of ions as heavy as gold with all of the ion's atomic electrons removed, fully exposing the

large charge of the atomic nucleus.

Lepton-pair production from nuclear processes has been widely discussed as a possible signal for the formation of a quark-gluon plasma phase of matter, which is thought to have existed in the initial phases of the Big Bang. The recreation of this phase of matter in the laboratory is the primary goal of the RHIC project. Electromagnetic electron-pair production from the vacuum by highly stripped heavy ions in relativistic motion is the dominant background process for these signatures. Furthermore, an accurate description of electromagnetic electron-pair production is important for both the design of experimental detectors for RHIC and the performance of the colliding-beam accelerator. In particular, electron and muon pair production, with subsequent atomic capture of the negatively charged lepton, changes the charge state of a participant heavy ion, leading to a decrease in the beam lifetime of the collider [3]. In addition to these practical matters, lepton-pair production is of interest because it allows physicists to test quantum electrodynamics – the fundamental theory of the interaction of light with the subatomic world – in a new energy regime.

Traditionally, processes such as lepton-pair production have been studied using perturbation theory, which is extremely successful in predicting phenomena associated with weak fields. However, this approach fails to describe the physics of the most intense collisions. In order to understand these nonperturbative effects for the important electron-capture problem, we explicitly solve the time-dependent Dirac equation coupled to the strong, time-dependent external fields produced by the heavy ions.

Another application of the lattice Dirac equation is a study of the fission dynamics of actinide nuclei. In this case, muons are captured by actinides and form excited muonic atoms. By nonradiative transitions (inverse internal conversion), sufficient atomic excitation energy is transferred to the nucleus to give a high probability that the nucleus will fission. Through the dynamics of a muon in the presence of the fissioning nucleus one expects to gain a deeper understanding of the energy-dissipation mechanism in large-amplitude nuclear collective motion.[4]

1.2 Dirac equation

In discussing the solution of the Dirac equation, we use natural units, i.e. $\hbar = c = m = 1$. This implies that energies are measured in units of the lepton rest mass, mc^2, and that our length and time units are the lepton Compton wavelength $\lambda_c = \hbar/m_c$ and Compton time $\tau_c = \lambda_c/c$, respectively. We solve the time-dependent Dirac equation in a reference frame in which one nuclei, henceforth referred to as the target, is at rest. The target nucleus and the lepton interact via the static Coulomb field, $A^0_{\rm T}$. The only time-dependent interaction, $(\vec{A}_{\rm P}(t), A^0_{\rm P}(t))$, arises from the classical motion of the projectile. Thus, it is natural to split the Dirac Hamiltonian into static and time-dependent parts. Accordingly, we write the Dirac equation for a lepton described by a spinor $\phi(\vec{r}, t)$ coupled to an external, time-dependent electromagnetic field as

$$[H_{\rm S} + H_{\rm P}(t)]\phi(\vec{r}, t) = i\frac{\partial}{\partial t}\phi(\vec{r}, t), \qquad (1)$$

where the static Hamiltonian, $H_{\rm S}$, which describes a stationary lepton in the presence of the strong, external Coulomb field of the target nucleus, is given by

$$H_{\rm S} = -i\vec{\alpha}\cdot\nabla + \beta - eA^0_{\rm T}\ , \qquad (2)$$

and the time-dependent interaction of the lepton with the projectile is

$$H_{\rm P}(t) = e\vec{\alpha}\cdot\vec{A}_{\rm P}(t) - eA^0_{\rm P}(t) \qquad (3)$$

where α_x, α_y, α_z, and β are the 4×4 Dirac spin matrices. The stationary states of the system, i.e. the eigenstates of the static Hamiltonian $H_{\rm S}$ in Eq. (2), are defined

$$H_{\rm S}\chi_i(\vec{r}) = E_i\chi_i(\vec{r})\ , \qquad (4)$$

which are also proper ingoing and outgoing states for asymptotic times $|t| \to \infty$, where the interaction $H_{\rm P}(t)$ is zero.

2 Numerical implementation

We solve the time-dependent Dirac equation using a lattice approach to obtain a discrete representation of all Dirac spinors and coordinate-space operators on a three-dimensional Cartesian mesh. We implement our lattice solution using the basis-spline collocation method, which is discussed in detail in Refs. [1, 5], and briefly summarized in Section 2.1. We limit this discussion to the special case of cubic lattices with uniform spacing in all three directions and periodic boundary conditions. However, the basis-spline collocation method is equally well suited for nonuniform lattice spacings and fixed-boundary conditions [1]. Both the applications previously introduced require the solution of the lowest-energy static bound state and subsequent evolution of this state in time. We describe the algorithms used for these tasks in Sections 2.2 and 2.3, respectively.

2.1 Lattice basis-spline collocation

Splines of order M are functions $S^M(x)$ of a single, real variable belonging to the class $C^{(M-2)}$ with continuous $(M-2)$th derivatives. These functions are piecewise continuous, as they are constructed from continuous polynomials of $(M-1)$th order joined at points in an ordered set $\{x'_i\}$ called knots. Basis-splines are the subset of the spline functions with minimal support in that they are zero outside the range of $M+1$ consecutive knots $x'_i, \ldots, x'_{i+M}$, and non-negative otherwise. We label these functions with the index of their first knot from the left as $B_i^M(x)$.

Consider a region of space with boundaries at $x_{\min}$ and $x_{\max}$ containing $N+1$ knots, including the knots on the boundaries. For a set of Mth-order basis-splines to be complete, M of the functions must be nonzero on each knot interval $[x'_i, x'_{i+1}]$ within the physical region. For this to occur, $M-1$ basis-splines must extend outside each boundary. Therefore, to construct a complete set of basis-splines requires $N+M+1$ functions naturally numbered as $B_1^M(x), \ldots B_{N+M-1}^M(x)$.

For purposes of illustration, we seek to approximate a continuous function $f(x)$, defined in the interval $[x'_{\min}, x'_{\max}]$, which is the solution of the differential equation

$$\mathcal{O}f(x) = 0 \,, \tag{5}$$

subject to periodic boundary conditions, where $\mathcal{O}$ is a coordinate-space differential operator. We introduce an approximate solution f^a in terms of the complete set of basis-spline functions $\{B_i^M(x)\}$ which is required to satisfy periodic boundary conditions exactly, i.e. $f^a(x_0) = f^a(x_0 + L)$. We do this in two steps: first we form a closed space by requiring $x_{\min} = x_{\max}$, and then we wrap the last $M-1$ splines in the basis set, which extend beyond the upper physical boundary, so that they enter the space from the lower boundary [5]. With this construction, $M-1$ basis-spline functions are redundantly labeled [5], resulting in the following expansion for f^a

$$f^a(x) = \sum_{i=1}^{N} B_i^M(x) c^i \,, \tag{6}$$

so that $R(x) \equiv \mathcal{O}f^a(x)$, where the quantities $\{c^i\}$ denote the expansion coefficients, and $R(x)$ denotes the residual in the interior of the region.

To obtain a set of equations for the expansion coefficients $\{c^i\}$, we apply the collocation method in which inner products of the residual weighted with Dirac-delta functions are required to be zero, where the set $\{x_\alpha\}$ contains the collocation points. Using Eqs. (6), the trivial integrals of the weighted residual are evaluated to obtain

$$R(x_\alpha) = \sum_{i=1}^{N} [\mathcal{O}B]_{\alpha i} c^i = 0 \,, \tag{7}$$

where $[\mathcal{O}B_i^M(x)]$ is the function resulting from the operation of $\mathcal{O}$ on the basis-spline function $B_i^M(x)$, and $[\mathcal{O}B]_{\alpha i} \equiv [\mathcal{O}B_i^M(x_\alpha)]$. In obtaining Eq. (7), we approximate the differential equation Eq. (5) for the function $f(x)$ by a set of linear equations for the expansion coefficients. This is done in a manner so that the coefficients require the residual $R(x)$ to be zero at the collocation points with the basis-splines providing an accurate interpolation to other values of x.

The solution of the linear system (Eq. (7)) for the expansion coefficients c^i provides, upon substitution into Eq. (6), the solution at all values of x within the boundaries. However, the essence of the lattice method is to eliminate the expansion coefficients c^i from the calculation in favor of a representation of the functions only on the collocation points. To implement this transformation, we create a linear system of equations by evaluating Eq (6) at the collocation points x_α, which we invert to isolate the expansion coefficients, i.e.

$$c^i = \sum_{\alpha=1}^{N} B^{i\alpha} f_\alpha^a \,, \tag{8}$$

where $B^{i\alpha} \equiv [B^{-1}]_{i\alpha}$. Using Eq. (8) to eliminate the expansion coefficients from the linear system in Eq. (7), one obtains

$$\sum_{i=1}^{N} [\mathcal{O}B]_{\alpha i} c^i = \sum_{\beta=1}^{N} O_\alpha^\beta f_\beta^a = 0 \,, \tag{9}$$

where we define the collocation-space or lattice representation of the operator $\mathcal{O}$ as

$$O_\alpha^\beta \equiv \sum_{i=1}^{N} [\mathcal{O}B]_{\alpha i} B^{i\beta} \,. \tag{10}$$

The most important applications of Eq. (10) are to local functions of the coordinates, and to spatial differential operators. Local operators such as potentials simply become diagonal matrices of their values at the collocation points, i.e. $V(x) \to V_\alpha$. Also, for example, the lattice representation of the first-derivative operator in the basis-spline collocation method is

$$D_\alpha^\beta \equiv \sum_{i=1}^{N} B'_{\alpha i} B^{i\beta} \,, \tag{11}$$

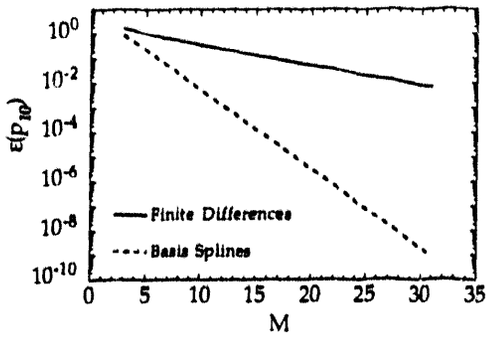


Figure 1: Errors in basis-spline and finite-difference representations of a single linear-momentum eigenvalue as a function of the order of the representation in a periodic space. Discrete values are shown as lines to guide the reader's eye.

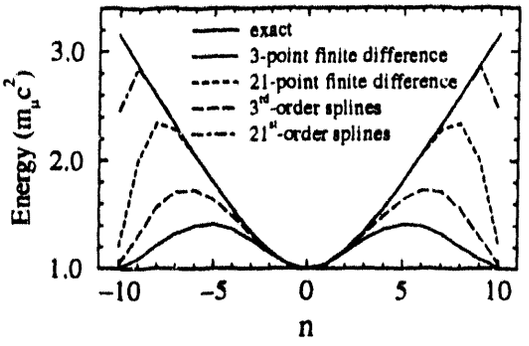


Figure 2: Depicted are the positive branches of the energy spectra for basis-spline and finite-difference representations of the free Dirac equation in one dimension.

where $B'_{\alpha i} \equiv \frac{dB_i^M(x)}{dx}|_{x=x_\alpha}$. In Fig. (1), we demonstrate the accuracy of the basis-spline collocation method, as compared to traditional finite-difference schemes, by computing the error in lattice representations of a single eigenvalue of the linear-momentum operator $\hat{p} \equiv -i(d/dx)$. Figure 2 demonstrates that the use of high-order spline representations of the momentum operator avoids the notorious energy-spectrum doubling problem for lattice representations of the Dirac equation [5].

We generalize the above procedure to functions in three-dimensional space by expanding in terms of products of basis-spline functions, i.e. $B_i^M(x)B_j^M(y)B_k^M(z)$. For example, using Eq. (10), consider the collocation-lattice representation of the gradient operator in Cartesian coordinates

$$\vec{D}^{\mu\nu\xi}_{\alpha\beta\gamma} = \hat{e}_1 D^\mu_\alpha \delta^\nu_\beta \delta^\xi_\gamma + \hat{e}_2 D^\nu_\beta \delta^\mu_\alpha \delta^\xi_\gamma + \hat{e}_3 D^\xi_\gamma \delta^\mu_\alpha \delta^\nu_\beta \ , \quad (12)$$

where and $\hat{e}_j$ is a unit vector in the j^{th} coordinate direction. In matrix notation, we denote Eq. (12) as

$$\vec{\mathbf{D}} = \hat{e}_1 \mathbf{D}_1 + \hat{e}_2 \mathbf{D}_2 + \hat{e}_3 \mathbf{D}_3 \ , \quad (13)$$

with the obvious definitions of the matrices $\mathbf{D}_1$, $\mathbf{D}_2$, and $\mathbf{D}_3$. Using Eq. (13), the lattice representation of the static Hamiltonian, Eq. (4), may be written

$$\mathbf{H_S} = -i\vec{\alpha} \cdot \vec{\mathbf{D}} + \beta - e\mathbf{A}^0_{\mathrm{T}} \ . \quad (14)$$

In summary, the collocation points define the lattice on which the calculations are performed; neither the splines nor the knots appear explicitly again once the lattice representation of the operators has been obtained at the beginning of the calculation. We have reduced the partial differential equation (Eq. (4)) to a set of matrix equations which may be solved using iterative techniques. As a consequence of eliminating the expansion coefficients from the theory, $\mathbf{H_S}$ has a blocked sparse representation which is self-adjoint for periodic boundary conditions and uniform meshes.

2.2 Lowest-energy bound state

The complete eigensolution of $\mathbf{H_S}$, providing its full spectrum of stationary states, currently approaches the state-of-the-art in computational capabilities due to the size of $\mathbf{H_S}$, which is equivalent to a rank $8N^3$ real matrix. We believe convergent calculations will be achieved for $N \approx 100$, based on the length and momentum scales involved, and experience with one-dimensional calculations. For this reason, we compute the lowest energy bound state ($1s$) needed as the initial state for our time-dependent problems by a partial eigensolution of $\mathbf{H_S}$.

Standard methods for partial eigensolution of large matrices, which are designed to converge to the lowest energy eigenstate of the spectrum, are not directly applicable for computing the $1s$ state of $\mathbf{H_S}$ because its spectrum extends to negative energies. The analytic operator H_S has positive and negative continua, $E > mc^2$ and $E < mc^2$, as well as bound states $|E| < mc^2$; the spectrum of $\mathbf{H_S}$ has the same branches though all the eigenvalues are discrete. The $1s$ state has been computed using a damped relaxation method [2]. This algorithm is constructed to remove the high-frequency components from the residual, and does not depend on the spectrum of $\mathbf{H_S}$ being bounded from below.

For larger lattice sizes discussed in this paper, we have developed a more efficient iterative Lanczos algorithm to compute the initial state [2]. The Lanczos algorithm proves attractive for our purposes as the memory requirements are relatively small and the method approximates extremal eigenvalues in the spectrum very well. Since convergence is most rapid for extremal

eigenvalues, we solve for the lowest energy eigenstate of $\mathbf{H_S}^2$, which has a positive-definite spectrum. By solving for the ground state of $\mathbf{H_S}^2$, we obtain the lowest-energy bound state of $\mathbf{H_S}$.

2.3 Time evolution

The formal solution of the time-dependent Dirac equation (Eq. (1)) is $\phi_j(t) = \hat{U}(t,t_0)\phi_j(t_0)$, where the unitary time propagator $\hat{U}(t,t_0)$ is given in the Schrödinger picture by the time-ordered exponential

$$\hat{U}(t,t_0) = \mathrm{Texp}\left(-i\int_{t_0}^{t} dt' \left[H_S + H_P(t')\right]\right) . \quad (15)$$

We discretize time in the sense that the electromagnetic interactions are taken as constant in successive small intervals of possibly varying size Δt_ℓ, i.e. $t_{\ell+1} = t_\ell + \Delta t_{\ell+1}$, $\ell = 0, 1, \ldots, L$, and express the evolution operator in successive factors $\hat{U}(t,t_0) = \hat{U}(t,t_{L-1}), \ldots, \hat{U}(t_1,t_0)$.

A number of different methods have been used to approximate the time-evolution operator

$$\hat{U}(t_{\ell+1},t_\ell) = \exp\left(-i\left[H_S + H_P(t_{\ell+1})\right]\Delta t_{\ell+1}\right) , \quad (16)$$

particularly in studies of the time-dependent Hartree-Fock method applied to atomic and nuclear collisions. The choice of a method usually depends on the dimensionality and structure of the Hamiltonian matrix. Several methods which work well in one- and two-dimensional problems are impractical for unrestricted three-dimensional problems because they require the inversion of part or all of the Hamiltonian matrix. In our three-dimensional solution of the Dirac equation, the exponential operator, Eq. (16), is implemented as a finite-number of terms of its Taylor series expansion.

In conclusion, all of the numerical procedures discussed for implementing our lattice methods reduce to a series of matrix-vector operations which can be executed with high efficiency on vector or parallel supercomputers without explicitly storing the matrix in memory.

3 Hypercube implementation

The iPSC/860 at ORNL is a distributed-memory, multiple-instruction, multiple-data-stream multicomputer containing 128 processors with 8 MBytes of memory per processor connected via a hypercube topology. The details of our implementation of the lattice representation of the Dirac equation on this computer are discussed in detail in Ref. [2] and briefly discribed here. As with many parallel implementations, we face the problems of limited memory per node and the optimization of the algorithm to minimize the communication between nodes. We discuss these issues for our implementation of the Dirac equation in Sections 3.1 and 3.2. Section 3.3 contains a brief analysis of the floating-point performance of the i860 processor.

3.1 Decomposition of the lattice

As discussed in Section 3, Dirac spinors are represented on a three-dimensional Cartesian lattice in our numerical solution of the Dirac equation

$$\phi(x,y,z) \longrightarrow \begin{pmatrix} \phi^{(1)}_{\alpha,\beta,\gamma} \\ \phi^{(2)}_{\alpha,\beta,\gamma} \\ \phi^{(3)}_{\alpha,\beta,\gamma} \\ \phi^{(4)}_{\alpha,\beta,\gamma} \end{pmatrix} , \quad (17)$$

where x_α, y_β, and z_γ denote the collocation lattice points in the x, y, and z directions, respectively. Indices in parentheses denote the Dirac spinor component. In the following, we denote the number of lattice points in the three Cartesian directions by N_x, N_y, and N_z.

We choose to parallelize the time-dependent Dirac equation by data decomposition. In practice, we partition the y and z dimensions of the lattice into subblocks while maintaining the full x dimension on each node. These subblocks are distributed onto the processors using a two-dimensional Gray-lattice binary identification scheme.[2]

To maximize the occurrence of nearest-neighbor communication, the number of lattice points in the y and z directions are chosen to be powers of two. If the number of allocated nodes, p, is an exact square, we allocate $p_z = \sqrt{p}$ and $p_y = \sqrt{p}$ nodes in y and z directions, respectively. This results in a square Gray lattice. For intermediate powers of two, the partition is performed by $p_z = \sqrt{2p}$, $p_y = \sqrt{p/2}$, thus resulting in a rectangular Gray lattice. We determine the number of lattice points kept on each node by $m_y = N_y/p_y$, and $m_z = N_z/p_z$. Thus, all local arrays have a spatial dimension of $N_x m_y m_z$ on each node.

The lattice subblock maintained on a particular node is established using the row and column indices of that node as obtained from the Gray lattice. For y_β and z_γ, the indices β and γ are offset by

$$\begin{aligned} 1 + m_y(i_c - 1) \leq\ & \beta \ \leq m_y + m_y(i_c - 1) \\ 1 + m_z(i_r - 1) \leq\ & \gamma \ \leq m_z + m_z(i_r - 1) \end{aligned} , \quad (18)$$

where $1 \le i_c \le p_y$ and $1 \le i_r \le p_z$ denote the column and row location of the node in the Gray lattice.

3.2 Ring algorithm

All of our iterative algorithms for the solution of the Dirac equation make use of the operation of the Dirac Hamiltonian matrix multiplying a Dirac spinor, $\phi' = (\mathbf{H_S} + \mathbf{H_P}(t))\phi$. Furthermore, most of the computational effort needed is required in computing this generalized matrix-vector product. In our lattice representation, the action of the Hamiltonian on a spinor is given schematically in Eq. (19). Using Cartesian coordinates, this product naturally decomposes into four parts, one for each coordinate direction (x, y, z), and a diagonal part. This separability makes it easy to define this product implicitly in a storage-efficient way. The explicit Hamiltonian matrix is never created in memory, reducing our memory requirements from order N^6 to order N^3.

The Dirac Hamiltonian matrix contains local potential terms, which are diagonal matrices, and nonlocal derivative terms, which are dense matrices. Performing matrix-vector multiplications with the nonlocal summations in the y and z dimensions requires node-to-node communication as these dimensions of the lattice are distributed across the processors. These terms which require communication are shown in brackets in Eq. (19),

$$
\begin{aligned}
&\left(\phi^{(s)}\right)'_{\alpha,\beta,\gamma} = \\
&- i \sum_{\alpha'=1}^{n_x} D_{\alpha}^{\alpha'} \sum_{s'=1}^{4} \alpha_x^{(ss')} \phi_{\alpha',\beta,\gamma}^{(s')} \\
&- \sum_{i_c=1}^{p_y-1} \left[i \sum_{\beta'=1+m_y(i_c-1)}^{m_y+m_y(i_c-1)} D_{\beta}^{\beta'} \sum_{s'=1}^{4} \alpha_y^{(ss')} \phi_{\alpha,\beta',\gamma}^{(s')} \right] \\
&- \sum_{i_r=1}^{p_z-1} \left[i \sum_{\gamma'=1+m_z(i_r-1)}^{m_z+m_z(i_r-1)} D_{\gamma}^{\gamma'} \sum_{s'=1}^{4} \alpha_z^{(ss')} \phi_{\alpha,\beta,\gamma'}^{(s')} \right] \\
&+ eA_x(i,j,k) \sum_{s'=1}^{4} \alpha_x^{(ss')} \phi_{\alpha,\beta,\gamma}^{(s')} \\
&+ eA_y(i,j,k) \sum_{s'=1}^{4} \alpha_y^{(ss')} \phi_{\alpha,\beta,\gamma}^{(s')} \\
&+ eA_z(i,j,k) \sum_{s'=1}^{4} \alpha_z^{(ss')} \phi_{\alpha,\beta,\gamma}^{(s')} \\
&+ \sum_{s'=1}^{4} \left([\beta]^{(ss')} - eA^0(i,j,k)\delta_{ss'} \right) \phi_{\alpha,\beta,\gamma}^{(s')} . \qquad (19)
\end{aligned}
$$

In the execution of the y and z nonlocal sums in Eq. (19), we use a ring algorithm, in which each subblock of the Dirac spinor visits each node once to perform the nonlocal matrix-vector operations economically [6]. This is achieved by having loops over the number of y and z nodes performed on each node as shown in lines 3 and 4 of Eq. (19). All the derivative matrices are stored in full on each node.

3.3 Computational kernel

The inner loops of Eq. (19) may be written as daxpy operation, i.e. $\mathbf{y} = a\mathbf{x} + \mathbf{y}$, where $\mathbf{x}$ and $\mathbf{y}$ are vectors and a is a scalar. To optimize the utilization of the high-performance features of the i860 processor, such as dual-instruction, pipeline, and quad-load modes, we have written an implementation of the daxpy in assembler language [2]. Figure 3 shows the performance results for the daxpy on the i860 for our assembler language routine. The vector length is measured in 64-bit words. The execution rate shown is obtained using timing tests that make 10^5 successive calls of the basic routine, using a stride of 1, and using the same argument list for each call. We see that the real performance of the daxpy saturates at about 25 Mflops. Because of memory constraints on the iPSC/860 hypercube, we currently realize modest vector lengths of 8 to 64 words in our solution of the Dirac equation. The performance of the daxpy over this range varies significantly due to pipelining.

3.4 Scaling model

In discussing the performance of our application, we will consider only the matrix-vector product discribed in Eq. (19), as this operation consumes more than 95% of the CPU time needed in solving the time-dependent Dirac equation. We will develop a simple

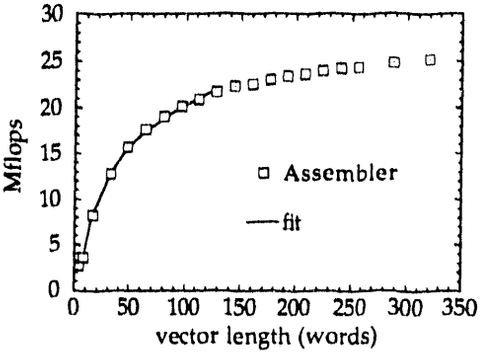


Figure 3: Execution rates as a function of vector length for the daxpy operation on the Intel i860.

scaling model of Eq. (19) for the time needed for useful calculation and internode communication in terms of the number of lattice points, the number of nodes, and the particular performance characteristics of the iPSC/860.

To execute Eq. (19) once, the total predicted time per node needed to perform floating-point operations ($T_{\rm calc}$) is the number of floating-point operations required, multiplied by the time $t_{\rm flop}(N)$ required to perform a single 64-bit floating-point operation within a vector of length N words. Assuming that the lattice has an equal number of points in the three coordinate directions, $N_x = N_y = N_z = N$, the estimated calculation time for Eq. (19) is

$$T_{\rm calc} = (48N + 448)\,\frac{N^3}{p}\,t_{\rm flop}(N)\ . \qquad (20)$$

The dependence of $t_{\rm flop}(N)$ on N is caused by the pipelined floating-point units of the i860 processor. From the performance of the assembler-coded daxpy operation shown in Fig. 3, we determine that $t_{\rm flop}(N)$ varies with N as the inverse of a logarithmic function

$$t_{\rm flop}(N) \approx \frac{1}{(15.1\log(N) - 9.9)\times 10^6}\ \text{seconds} \qquad (21)$$

over N ranging from 8 to 128.

Empirically, the communication time for a one-hop node-to-node message is a linear function of the size of the message [2]. In performing the nonlocal summations in Eq. (19), we are required to pass $p_y + p_z$ messages of length $8N^3/p$ 64-bit words. Passing these subblocks of the Dirac spinor around the two-dimensional Gray lattice ideally consumes the time

$$T_{\rm pass} = (p_y + p_z)\left(8\frac{N^3}{p}t_{\rm comm} + t_{\rm start}\right)\ , \qquad (22)$$

where $t_{\rm comm}$ is the typical time needed to actually transmit a single 64-bit word of data between two nodes, and $t_{\rm start}$ is the startup time for a single communication request. Typical times for the iPSC/860 are $t_{\rm comm} = 3.2\times 10^{-6}$sec, and $t_{\rm start} = 1.36\times 10^{-4}$sec [2].

Other overheads associated with communication add to the total communication time and are difficult to quantify. For example, since the nonlocal operations of Eq. (19) are dominated by communication, as we shall see in Section 5, a node must occasionally pause from performing useful computation until it receives the next subblock of the Dirac spinor. This waiting leads to additional delays caused by loss of synchronization between the nodes during message passing. We denote these overheads in useful computation as $T_{\rm ohead}$, and include this in our overall estimate of the communication time needed to perform Eq. (19)

$$T_{\rm comm} = (p_y + p_z)\left[8\frac{N^3}{p}t_{\rm comm} + t_{\rm start}\right] + T_{\rm ohead}\ . \qquad (23)$$

We will adjust $T_{\rm ohead}$ to fit the measured communication time.

Table 1: Presented are execution times in seconds for 2004 iterations of Eq. (19) for $p = 1$ and $p = N$ processors using various lattice sizes. Extrapolated values are denoted by an asterisk. Speedup and parallel efficiency are computed using these values.

N	$T(p=N)$	$T(p=1)$	$S(p=N)$	$\epsilon(p=N)$
8	48.4	90.5	1.9	0.24
12	-	257.4	-	-
16	239.5	718.5	3.0	0.19
20	-	1580.2	-	-
32	1269.0	6375.9*	5.0	0.16
64	6421.0	55816.8*	8.7	0.14

4 Timing results

Table 1 presents the time in seconds consumed by 2004 iterations of Eq. (19) on the iPSC/860 for lattice sizes of N^3, where $N = 8$, 16, 32, and 64 using one and $p = N$ nodes, respectively, and the corresponding values of the speedup and parallel efficiency. The total execution time for Eq. (19) on one i860 processor, which is necessary for computing the speedup and the efficiency, cannot be measured directly for our application for $N > 20$ because of memory constraints. However, we obtain estimates for Eq. (19) for lattice sizes $N = 32$ and 64 by extrapolation using a power law fit to the measured dependence on problem sizes for $N = 8$, 12, 16, and 20.

To distinguish accurately the time spent in communication from the time spent performing useful calculations in a realistic parallel algorithm such as Eq. (19) is a difficult task. Since we are only interested in understanding, in general terms, the balance between communication and computation in Eq. (19), we will use a simple, indirect approach to obtain communication and calculation times. Our method is based on the fact that node-to-node communication occurs only in the nonlocal summations in the y and z dimensions, namely, lines 3 and 4 of Eq. (19). Also, if $N_x = N_y = N_z = N$, each of the three nonlocal

Table 2: Execution times in seconds for 2004 operations with Eq. (19) on a lattice with N^3 points using an Intel iPSC/860 hypercube with $p = N$ nodes.

N	T_{xprd}	T_{yprd}	T_{zprd}	T_{calc}	T_{comm}	f_c
8	6.1	10.9	23.9	25.6	22.6	0.88
16	22.3	102.9	96.1	84.2	154.5	1.84
32	97.7	422.0	702.4	349.6	929.0	2.66
64	479.7	3006	2963	1659	5010	3.02

operations performs the same amount of useful work. We give these x, y, and z dimensional summations the names xprd, yprd, zprd, respectively, and the full operation of Eq. (19) is named hdprd. We attribute the difference in time needed to execute xprd and yprd, or xprd and zprd to node-to-node communication, and, thus, estimate this time as

$$T_{\text{comm}} \equiv T_{\text{yprd}} + T_{\text{zprd}} - 2T_{\text{xprd}} , \qquad (24)$$

where T_{xprd} is the time spent in xprd, and so on. To obtain our estimate of the time T_{hdprd} spent performing useful calculations in Eq. (19), we simply subtract T_{comm} from the total time needed to compute Eq. (19),

$$T_{\text{calc}} = T_{\text{hdprd}} - T_{\text{comm}} . \qquad (25)$$

Execution times for xprd, yprd, and zprd are presented in Table 2 for 2004 iterations of Eq. (19). Calculation and communication times are determined using Eqs. (24) and (25) and are also presented in Table 2 along with the fractional communication overhead, $f_c = T_{\text{comm}}/T_{\text{calc}}$. In order to perform the larger size calculations presented here, we are constrained by memory to increase the number of nodes used as the lattice size increases. We choose to increase the number of nodes in such a way that $p = N$.

We observe good agreement between our model for the calculation time and the measured result. The overhead T_{ohead} is fit by a power law so that Eq. (23) reproduces the measured communication time. The predicted fractional communication overhead obtained using Eqs. (20) and (23), and the measured values of this quantity listed in Table 2 are compared in Fig. 4. Notice that the predicted and measured values for this quantity agree well throughout the range of problem sizes, $8 \leq N \leq 64$, and that the communication overhead increases rapidly up to $N = 64$. This initial increase in overhead with problem size at first seems counterintuitive, but is explained by pipelining. Increasing floating-point performance with problem size causes the fractional communication overhead to initially increase.

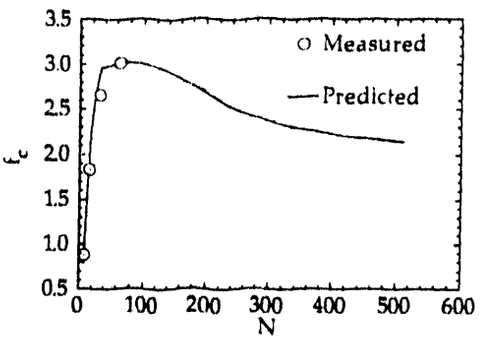


Figure 4: Plotted is the fractional communication overhead f_c as a function of the lattice size N obtained from the predictions in Eqs. (20) and (23) and from the measurements given in Table 2.

The large communication overheads in Table 2 and the small efficiencies in Table 1 indicate a poor balance between computation and communication for current problem sizes. There are two main reasons for our program being communication-bound. The first results from the slow speed of node-to-node communication relative to the speed for performing floating-point operations on the iPSC/860. The ratio of the time to communicate one node-to-node message of length 64 bits to the time to perform one double-precision floating-point operation, Eq. (21), in large, i.e.

$$\frac{t_{\text{comm}}}{t_{\text{flop}}} = 48.3 \log N - 31.7 . \qquad (26)$$

Another reason for the low efficiency of our application is its large memory requirement resulting is large messages being passed from node to node. The number of these messages passed increases roughly as $2\sqrt{p}$ with the number of processors used.

In Fig. 5, we compare the performance of our solution of the time-dependent Dirac equation on the iPSC/860 with its performance on two other computers to which we have access: a Cray-2 supercomputer and an IBM RS/6000 320H workstation. In computing the floating-point performance of the i860 for the purposes of this comparison, we use the overall time T_{hdprd} for Eq. (19) without factoring the communication. In this case, floating-point performance can be considered proportional to CPU time. We optimize our implementation of Eq. (19) on the Cray-2 machine using the cf77 Fortran compiler with default vectorization, loop unrolling, and no autotasking. For the IBM workstation, we use the IBM AIX XL Fortran Compiler/6000 version 2.2 with full optimization. We see that for $N = p = 64$, the iPSC/860 performs better than the Cray-2 by a factor of 2.2, with the trend for larger problem sizes clearly in favor of the hypercube.

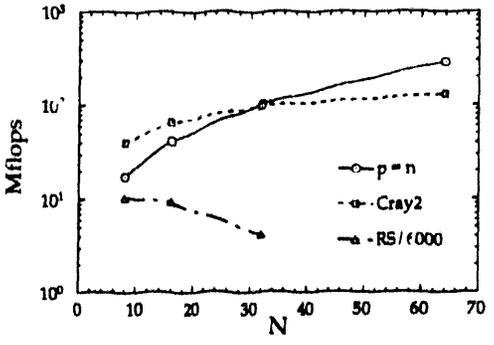


Figure 5: A comparison of the performance of implementations of Eq. (19) on the Intel iPSC/860 with $p = N$ processors, on a Cray-2, and on an IBM RS/6000 320H.

5 Numerical results

We present preliminary results for muon-pair production with capture into the ground state in collisions of $^{197}Au + {}^{197}Au$ at energies of 2 GeV per nucleon in a collider frame of reference. In Fig. 6, we show the time-evolution of the muon-position probability density plotted as a negative logarithm for a grazing impact parameter. In the lower part of Fig. 6, we show the scalar-component $(A^0(\vec{r},t))$ of the interaction of the muon with the time-dependent electromagnetic field. The contribution to this interaction from the target is the relatively small bump in the center of the lattice. The contribution from the projectile is the large, negative spike moving across the lattice.

When the projectile is very far away from the target, the initial density of the ground state is spherical. As the projectile passes the target, this spherical density deforms, expands, and develops both positive and negative energy continuum (free) components. This time-evolved spinor is required to compute the probability for muon-pair production with capture. We have reported preliminary results for these calculations in Ref. 3. Larger numerical boxes with more lattice points are needed for convergent calculations.

In Fig. 7, we present results for prompt fission of ^{238}U induced by the $E2 : (3d \rightarrow 1s, 9.6MeV)$ nonradiative muonic atom transition. In the upper part of Fig. 7 we show the time-development of the muon position probability density during fission. The lower part of the figure displays the Coulomb interaction energy between the muon and the fission fragments. Initially, the muon is bound to a deformed ^{238}U nucleus. We can see that the muonic wave function tends to follow the two Coulomb wells of the fission fragments in motion. The deeper well on the right is generated by the heavy fission fragment. For a fragment mass ratio of $A_H/A_L = 1.40$, we observe that the muon sticks predominately to the heavy fragment; the muon attachment probability to the light fragment is represented by the small bump on the right. Preliminary results for the muon attachment probability to the light fission fragment, P_L, as a function of the dissipated nuclear energy have been published in a recent Letter journal [4]. We are currently in the process of performing a quantitative comparison between the theory and all the available experimental data.

Acknowledgements

This research was sponsored in part by the U.S. Department of Energy under contract No. DE-AC05-84OR21400 managed by Martin Marietta Energy Systems, Inc., and under contract No. DE-FG05-87ER40376 with Vanderbilt University. The numerical calculations were carried out on the Intel iPSC/860 hypercube multicomputer at the Oak Ridge National Laboratory, and the CRAY-2 supercomputers at the National Energy Research Supercomputer Center at Lawrence Livermore National Laboratory, and the National Center for Supercomputing Applications in Illinois.

References

[1] A. S. Umar, J.-S. Wu, M. R. Strayer, and C. Bottcher, *J. Comp. Phys.* 93 (1991) 426.

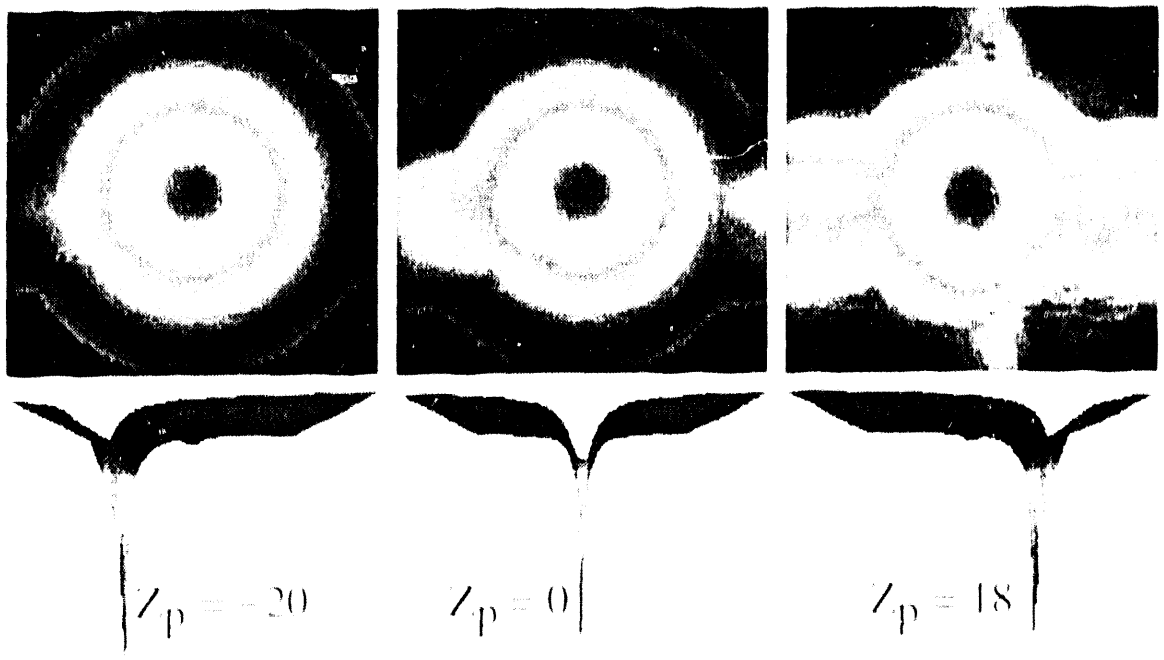
[2] J. C. Wells, V. E. Oberacker, A. S. Umar, C. Bottcher, M. R. Strayer, J.-S. Wu, J. Drake, and R. Flanery, *Int. J. Mod. Phys. C* 4 (1993) 459.

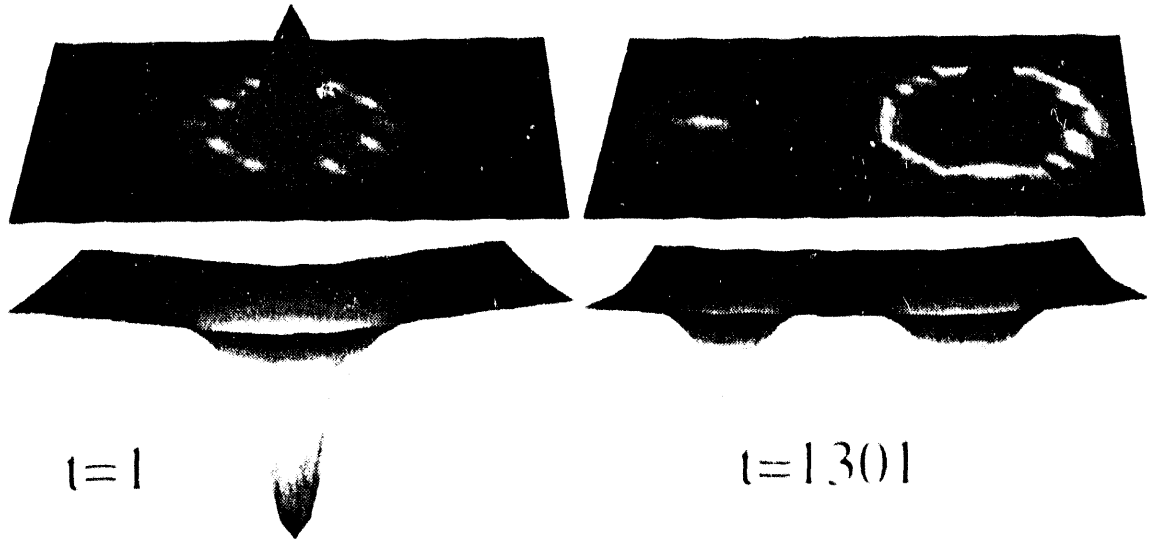
[3] J. C. Wells, V. E. Oberacker, A. S. Umar, C. Bottcher, M. R. Strayer, J.-S. Wu, and G. Plunien, *Phys. Rev. A* 45 (1992) 6296.

[4] V. E. Oberacker, A. S. Umar, J. C. Wells, C. Bottcher, and M. R. Strayer, Phys. Lett. B293, (1992) 270.

[5] J. C. Wells, V. E. Oberacker, M. R. Strayer, and A. S. Umar, submitted to *Int. J. Mod. Phys. C* (1993).

[6] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors, Vol. I* (Prentice-Hall, Englewood Cliffs, N.J., 1988), p. 261.


$Z_p = -20$
$Z_p = 0$
$Z_p = 18$


t=1
t=1301

DATE

FILMED

11/19/93

END