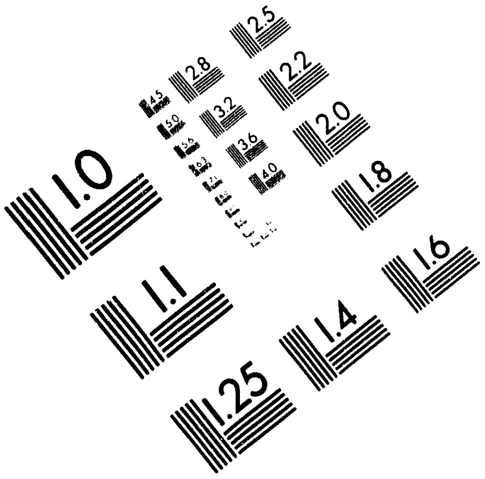
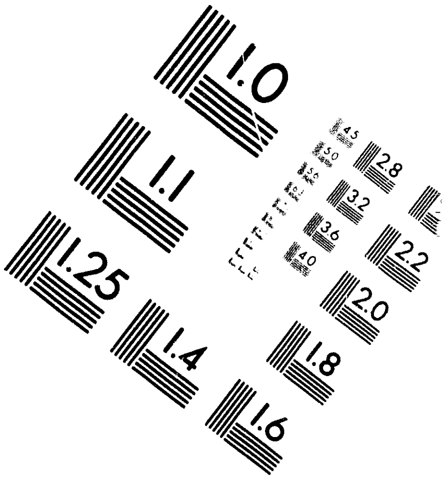
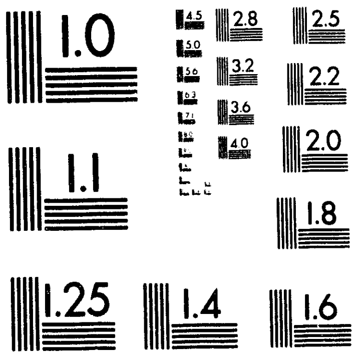
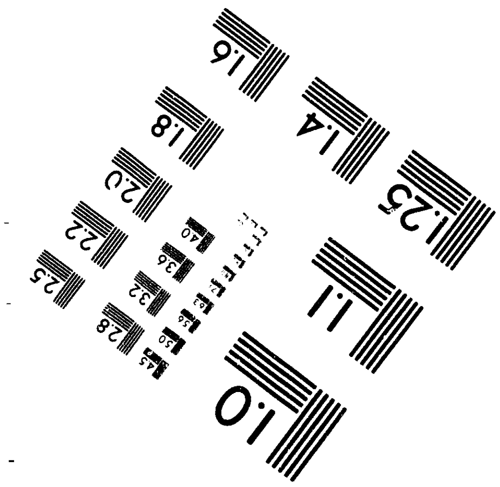
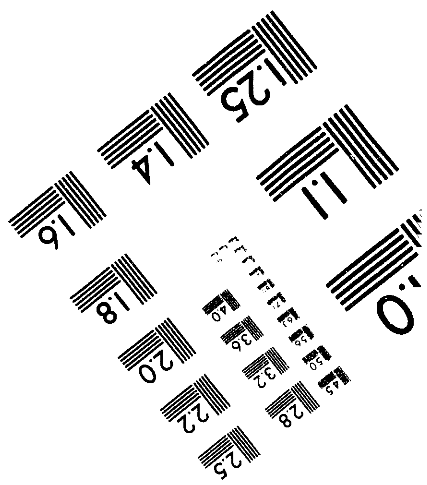



Centimeter
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 mm
1 2 3 4 5
Inches



1 of 1

2.

Conf-9408116--1

PARALLEL IMPLEMENTATION OF THE DIRAC EQUATION IN THREE CARTESIAN DIMENSIONS

J. C. Wells, V. E. Oberacker, M. R. Strayer, and A. S. Umar

to be published in

Proceedings of

6th Joint EPS-APS International Conference on Physics Computing

Lugano, Switzerland
August 22-26, 1994

[illegible]

PARALLEL IMPLEMENTATION OF THE DIRAC EQUATION IN THREE CARTESIAN DIMENSIONS

J. C. WELLS[1], V. E. OBERACKER[1,2], M. R. STRAYER[1], and A. S. UMAR[1,2]

[1] *Physics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831-6373*

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

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 vacuum-pair production in relativistic heavy-ion collisions. Discretization is achieved through the lattice-collocation method. 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]. Much of this work has been motivated by the opportunity to study nonperturbative electromagnetic electron-pair production from relativistic heavy-ion collisions. Electromagnetic production from the vacuum of single- and multiple-lepton pairs is estimated to be a major contribution to the physical background in the search for leptonic signals from hard hadronic processes occurring in nonperipheral heavy-ion collisions [3] and, therefore, must be understood in detail. In addition, the process of electron-positron production with electron capture by one of the participant ions is a principal beam loss mechanism for highly charged relativistic ions in a storage ring, and thus, plays a central role in the design and the operation of these machines [2].

A semiclassical approximation is appropriate for the pair-production problem using the classical, imperturbable nature of the electromagnetic field generated by the heavy ions, and neglecting lepton-lepton interactions. In this formalism, strong-field quantum electrodynamics is reduced to numerically solving the Dirac equation coupled to the sharply pulsed fields generated by the heavy ions. We solve the time-dependent Dirac equation ($\hbar = c = m = 1$) 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_{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

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

where the static Hamiltonian, $H_{\rm S}$, is given by

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

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

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

where α_x, α_y, α_z, and β are the 4×4 Dirac spin matrices.

2 Numerical solution

The solution of the Dirac equation coupled to such an external field is a difficult numerical task. At extreme relativistic velocities, the retarded electromagnetic interaction breaks all symmetries of three-dimensional space. In addition, the relativistic wavefunctions associated with the K-shell of a heavy ion are highly localized in space and present a rather simple picture which is easy to interpret. For these reasons, methods used in this work are designed for the lattice solution of the Dirac equation in unrestricted configuration space using three-dimensional Cartesian coordinates. In addition, multiple length scales enter the electron-position pair-production problem with heavy ions; the radius of the heavy ion, i.e. $\mathrm{R_{ion}} \approx$ 10 fm, the Compton wavelength of the lepton, i.e. $\lambda_e \approx$ 400 fm, and the radius of the heavy ions K-shell, i.e. $\mathrm{R_K} \approx 10\lambda_e$. The numerical methods implemented must overcome the problems posed by the unbounded, non-positive-definite energy spectrum of the Dirac Hamiltonian. In addition, naive lattice implementations of the Dirac equation suffer from an anomaly of the Dirac-energy dispersion relation known as *fermion doubling.*

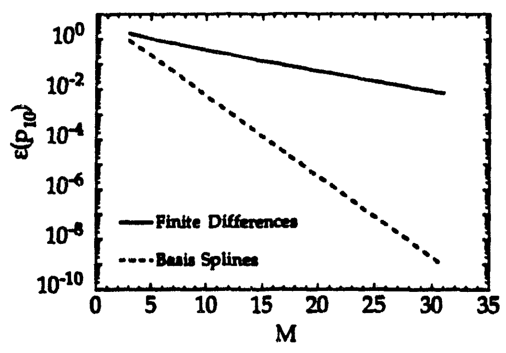


Figure 1: Errors in basis-spline collocation and finite-difference representations of a single linear-momentum eigenvalue as a function of the order of the representation in a periodic space.

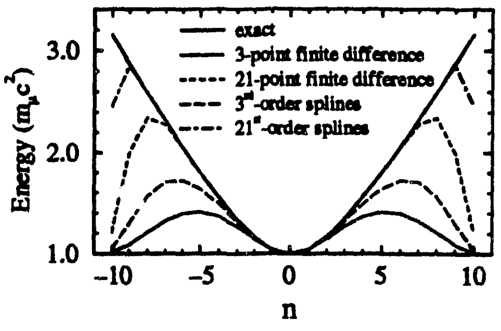


Figure 2: Fermion doubling is demonstrated by plotting the positive branches of the energy spectra for a few lattice representations of the free Dirac equation in one dimension.

Lattice-collocation methods have been developed as state-of-the-art approachs to solving partial-differential equations [1, 4]. Using this approach, one forsakes a continuous representation of the quantum-state vector and coordinate-space operators in favor of a representation only on a discrete set of spatial lattice points, e.g. $\chi(\vec{r}) \rightarrow \chi_{\alpha,\beta,\gamma}$. Therefore, Dirac spinors become discrete vectors of $4N_xN_yN_z$ complex numbers, and the Dirac Hamiltonian becomes a matrix of the same rank. Local operators like the electromagnetic interaction become diagonal matrices with the values along the diagonal being simply the value of the interaction at a given lattice point. Nonlocal operators, like coordinate-space derivatives, are represented by full matrices, whereas, operators are banded matrices in a finite-difference approach. The memory requirement of large three-dimensional problems is dominated by storage for the wavefunction, not the Hamiltonian matrix, because of the separability of the kinetic-energy operator in Cartesian coordinates. Therefore, high-order methods which reduce the total number of lattice points required, even at the expense of dense lattice operators, have an advantage.

2.1 *Lowest-energy bound state*

The complete eigensolution of $\mathbf{H}_\mathrm{S}$, providing its full spectrum of stationary states, currently approaches the state-of-the-art in computational capabilities due to the size of $\mathbf{H}_\mathrm{S}$, 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-dir ensional 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 $\mathbf{H}_\mathrm{S}$. 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 $\mathbf{H}_\mathrm{S}$ because its spectrum extends to negative energies. We have developed an 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 $\mathbf{H}_\mathrm{S}{}^2$, which has a positive-definite spectrum. By solving for the ground state of $\mathbf{H}_\mathrm{S}{}^2$, we obtain the lowest-energy bound state of $\mathbf{H}_\mathrm{S}$.

2.2 *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

$$\hat{U}(t,t_0) = \mathrm{Texp}\left(-i\int_{t_0}^{t} dt'\,[H_\mathrm{S} + H_\mathrm{P}(t')]\right) . \tag{4}$$

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

$$\hat{U}(t_{\ell+1},t_\ell) = \exp\left(-i\,[H_\mathrm{S} + H_\mathrm{P}(t_{\ell+1})]\,\Delta t_{\ell+1}\right) , \tag{5}$$

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. (5), is implemented as a finite-number of terms of its Taylor series expansion.

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 described 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 choose to parallelize the lattice Dirac equation by data decomposition. In practice, we partition the y and z dimensions of the lattice, of size N_y and N_z, respectively, 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.

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. 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 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. In the execution of the y and z nonlocal sums, 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 [5]. This is achieved by having loops over the number of y and z nodes performed on each node. All the derivative matrices are stored in full on each node.

3.1 *Performance model*

The inner loops of the matrix-vector product 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.

In discussing the performance of our application, we will consider only the Hamiltonian-vector product, as this operation consumes more than 95% of the CPU time needed in solving the time-dependent Dirac equation. To execute this operation once, the total predicted time per node needed to perform floating-point operations (T_{calc}) is the number of floating-point operations required, multiplied by the time $t_{\text{flop}}(N)$ 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 the matrix-vector product is

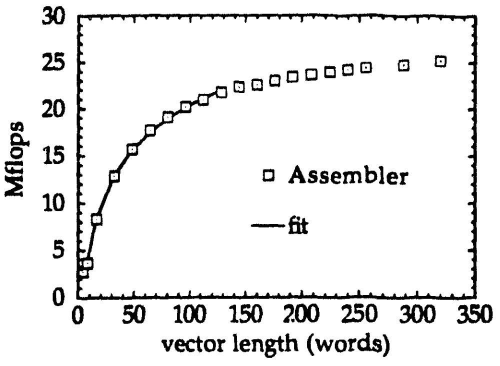


Figure 3: Execution rates as a function of vector length for the **daxpy** operation on the Intel i860.

$$T_{\text{calc}} = (48N + 448)\,\frac{N^3}{p}\,t_{\text{flop}}(N)\ . \tag{6}$$

The dependence of $t_{\text{flop}}(N)$ 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 $t_{\text{flop}}(N)$ varies with N as the inverse of a logarithmic function

$$t_{\text{flop}}(N) \approx \frac{1}{(15.1\log(N) - 9.9)\times 10^6}\ \text{seconds} \tag{7}$$

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 the matrix-vector product, 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

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

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

Other overheads associated with communication add to the total communication time and are difficult to quantify. For example, since the nonlocal operations 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

Table 1: Presented are execution times in seconds for 2004 iterations of the matrix-vector product 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

between the nodes during message passing. We denote these overheads in useful computation as T_{ohead}, and include this in our overall estimate of the communication time needed to perform a matrix-vector product

$$T_{\text{comm}} = (p_y + p_z)\left[8\frac{N^3}{p}t_{\text{comm}} + t_{\text{start}}\right] + T_{\text{ohead}}. \tag{9}$$

4 Results

Table 1 presents the time in seconds consumed by 2004 iterations of the Hamiltonian-vector product 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.

Since we are only interested in understanding, in general terms, the balance between communication and computation in the matrix-vector product, 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. Also, if $N_x = N_y = N_z = N$, each of the three nonlocal 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 the matrix-vector product 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

$$T_{\text{comm}} \equiv T_{\text{yprd}} + T_{\text{zprd}} - 2T_{\text{xprd}} , \tag{10}$$

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, we simply subtract T_{comm} from the total time needed to compute a single matrix-vector product,

$$T_{\text{calc}} = T_{\text{hdprd}} - T_{\text{comm}} . \tag{11}$$

Execution times for **xprd**, **yprd**, and **zprd** are presented in Table 2 for 2004 iterations of the matrix-vector product. Calculation and communication times are determined using Eqs. (10) and (11) and are also presented in Table 2 along with the fractional communication overhead, $f_c = T_{\text{comm}}/T_{\text{calc}}$.

We observe good agreement between our model for the calculation time and the measured result. The predicted fractional communication overhead obtained using

Table 2: Execution times in seconds for 2004 operations of the matrix-vector product on a lattice with N^3 points using an Intel iPSC/860 hypercube with $p = N$ nodes.

N	$T_{\rm xprd}$	$T_{\rm yprd}$	$T_{\rm zprd}$	$T_{\rm calc}$	$T_{\rm comm}$	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

Eqs. (6) and (9), 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.

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. (7), in large, i.e.

$$\frac{t_{\rm comm}}{t_{\rm flop}} = 48.3 \log N - 31.7 \,. \qquad (12)$$

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.

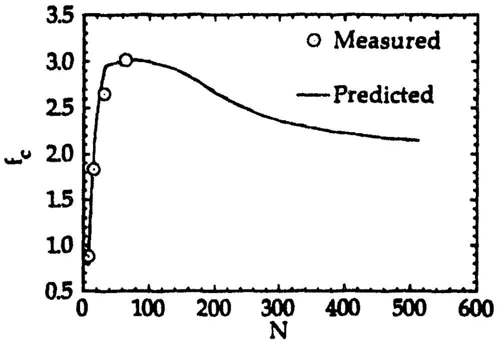


Figure 4: Plotted is the fractional communication overhead f_c as a function of the lattice size N obtained from the predictions in Eqs. (6) and (9) and from the measurements given in Table 2.

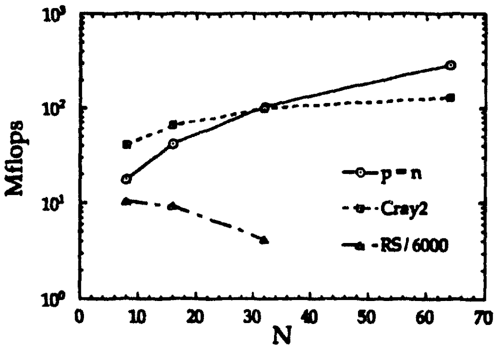


Figure 5: A comparison of the performance of implementations of the Hamiltonian-vector product on the Intel iPSC/860 with $p = N$ processors, on a Cray-2, and on an IBM RS/6000 320H.

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 $T_{\rm hdprd}$ without factoring the communication. 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.

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. 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. The muon density was computed for a grazing impact parameter collision of $^{197}Au +^{197} Au$ at energies of $2GeV$ per nucleon in the collider frame of reference.

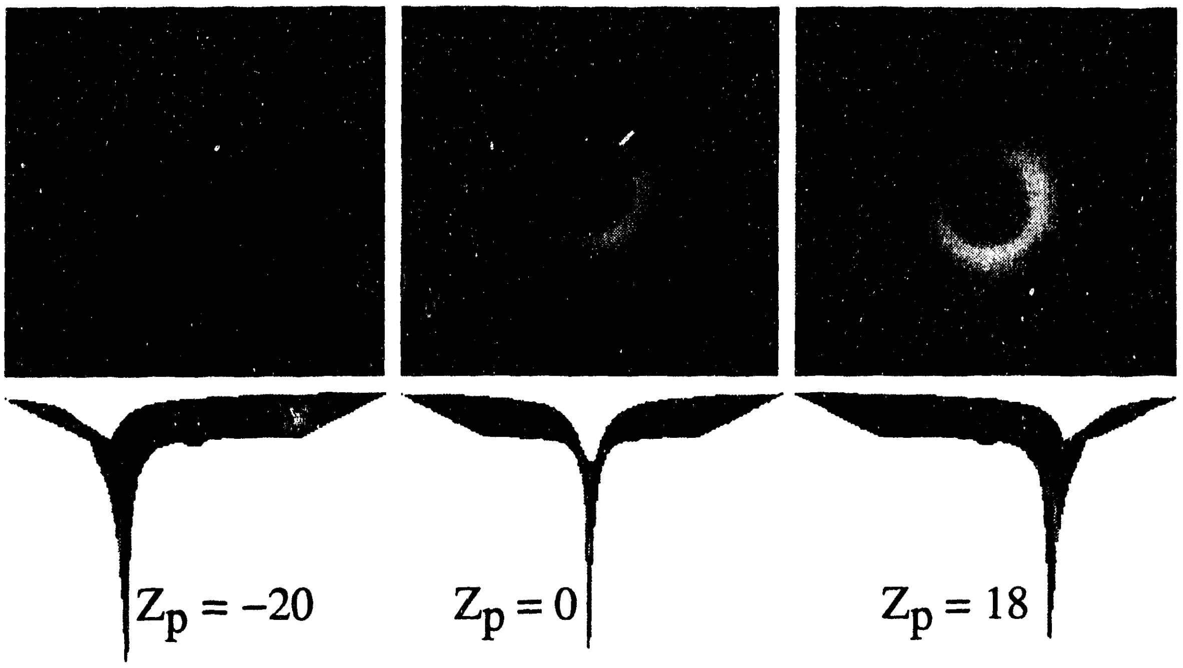


Figure 6: Plotted is the negative logarithm of the muon-position probability density (*top*) and the scalar-component $A^0(\vec{r},t)$ of the muon's interaction with the time-dependent electromagnetic field (*bottom*) for projectile positions of $z_p = -20, 0$, and $18\lambda_c$.

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.

References

[1] A. S. Umar, et al., *J. Comp. Phys.* **93** (1991) 426.

[2] J. C. Wells, et al., *Int. J. Mod. Phys. C* **4** (1993) 459.

[3] J. C. Wells, et al., *Phys. Rev. A* **45** (1992) 6296.

[4] J. C. Wells, et al., submitted to *Int. J. Mod. Phys. C* (1993).

[5] G. Fox, et al., *Solving Problems on Concurrent Processors, Vol. I* (Prentice-Hall, Englewood Cliffs, N.J., 1988), p. 261.