Solution of a Three-Body Problem in Quantum Mechanics Using Sparse Linear Algebra on Parallel Computers*

Mark Baertschy
JILA
University of Colorado
CO 80309-0440
baertsch@colorado.edu

Xiaoye Li
NERSC
Lawrence Berkeley National Laboratory
Berkeley, CA 94720
xiaoye@nersc.gov

ABSTRACT

A complete description of two outgoing electrons following an ionizing collision between a single electron and an atom or molecule has long stood as one of the unsolved fundamental problems in quantum collision theory. In this paper we describe our use of distributed memory parallel computers to calculate a fully converged wave function describing the electron-impact ionization of hydrogen. Our approach hinges on a transformation of the Schrödinger equation that simplifies the boundary conditions but requires solving very ill-conditioned systems of a few million complex, sparse linear equations. We developed a two-level iterative algorithm that requires repeated solution of sets of a few hundred thousand linear equations. These are solved directly by *LU*-factorization using a specially tuned, distributed memory parallel version of the sparse *LU*-factorization library SuperLU. In smaller cases, where direct solution is technically possible, our iterative algorithm still gives significant savings in time and memory despite lower megaflop rates.

1. INTRODUCTION

This paper describes our use of massively parallel processing (MPP) computers to solve a long-standing, fundamental problem in atomic physics. Our work produced the first calculations of detailed information about two outgoing electrons following an ionizing collision between an electron and a hydrogen atom that agree with experiment over a wide range of energies and angles [21, 4, 5, 13]. We calculate a six-dimensional wave function by solving the time-independent Schrödinger equation using a mathematical transformation to simplify the scattering boundary conditions. This requires solving several large (on the order of 1.2 to 8 million) sets of complex, sparse linear equations that are very ill-conditioned.

For this we developed a specialized, two-level iterative algorithm. As a preconditioning step in iteratively solving the full set of equations we repeatedly solve moderately large sets (between 209,764 and 334,084) of complex, sparse linear equations. These, in turn, are solved iteratively using the direct solution of a simpler set of equations as a preconditioner. To accomplish the inner preconditioning step we use a parallel version of the sparse *LU*-factorization library SuperLU [11, 14] with enhanced capabilities to handle complex data types and to solve multiple independent systems simultaneously on separate groups of processors. Our codes are written in Fortran 90 and C using MPI for communication and have been used on a Cray T3E-900 and an IBM SP.

The scientific breakthrough could not have been achieved without our newly developed algorithms and parallel processing capabilities outlined below:

- A new mathematical transformation for solving the time-independent Schrödinger equation makes the numerical computational task feasible.
- A non-conventional, parallel, two-level iterative algorithm for solving complex linear systems that are very large, sparse and ill-conditioned. In particular, our parallel preconditioner using SuperLU is crucial for convergence.
- Demonstrated high performance in solving systems of equations as large as 8 million taking between 40 and 140 minutes and using up to 96 processors.

The rest of the paper is organized as follows. In Section 2, we discuss the scientific application, the obstacles to performing accurate calculation, and survey the earlier attempts in solving this problem. Section 3 describes our new mathematical formulation for this problem and the resulting sparse linear systems. One major contribution is a simplified fomulation of the scattering boundary conditions that make the computational tasks tractable. Section 4 gives an overview of our parallel solver strategy and compares with some other solution techniques. Section 5 discusses the iterative algorithm and performance for the uncoupled equations, and the preconditioner in particular. Solutions of the uncoupled equations are used as preconditioning to solve the fully coupled equations, the details of which are given

*This work was supported in part by the Director, Office of Science, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC03-76SF00098 and by the U.S. DOE Office of Basic Energy Science, Division of Chemical Sciences.

SC2001 Denver, Colorado USA

in Section 6. Finally, in Section 7 we highlight the scientific results obtained through this computation.

2. SCIENTIFIC APPLICATION

If the collision between a target atom (or molecule) and an electron is of sufficiently high energy then there is some probability that the collision will result in detaching an electron, originally bound to the target. This process is known as electron-impact ionization and is characterized by an initial state with a single electron incident on the target followed by a final state with two electrons outgoing from the ionized target. The two-outgoing electrons make electron-impact ionization much more difficult to treat than other electron-scattering events, such as excitation of the target and elastic scattering, that have only one outgoing electron in the final state. A complete theoretical description of electron-impact ionization requires the solution of a three-body problem in quantum mechanics that is further complicated by the existance of long-range, Coulomb interactions between all three particles in the final state.

Electron-impact ionization is one of the most basic phenomena in low-energy collision physics. It is the fundamental mechanism for ion formation in mass spectroscopy and is responsible for forming and sustaining low-temperature plasmas that are used in applications ranging from fluorescent lighting to the processing of silicon chips. A better understanding of this basic phenomenon will lead to the ability to better understand and model macroscopic phenomena in low-temperature plasmas that are important in the atmospheric sciences, astrophysics, and a variety of industrial applications. Despite its importance, it is only recently that, with the aid of MPP, we have achieved what could be considered a complete description of electron-impact ionization of the simplest atomic target – a ground state hydrogen atom.

Probabilities for collision events are traditionally expressed in units of area and are referred to as cross sections. The ionization cross section, then, gives the probability that an atom will be ionized by collision with an electron at a particular incident energy. A complete theoretical description means calculating differential cross sections that give probability distributions for the final energies and directions of both outgoing electrons. The primary obstacle to doing this is the difficulty in formulating the correct scattering boundary conditions for the two outgoing electrons. Much of the work on the mathematical theory of ionization, beginning in the 1960s [18, 23], has been in developing asymptotic forms of the wave function. So far, no such asymptotic that could be used in an actual calculation have been developed.

The leading approaches to treating electron-atom scattering above the ionization threshold have been attempts to extend close-coupling formalisms, which work well for two-body processes such as discrete excitations of the atom and elastic scattering, to the three-body problem of ionization [10, 8]. The most successful of these, the convergent close-coupling method has produced accurate *total* ionization cross sections, but failed to converge to the correct differential cross sections [9]. Other approaches are based on various approximations that limit their usefulness, when they work at all, to very specific geometries [17, 25]. One method that has recently been shown to be capable of producing correct differential cross sections for ionization [6] involves propagation of the time-dependent Schrödinger equation [19]. However, this method is very computationally intensive and has yet to produce converged results.

Our approach builds on the early, formal theory but obviates the need to specify the exact scattering boundary conditions by using a mathematical transformation of the Schrödinger equation. Although this transformation makes the boundary conditions tractable, its implementation requires solving very large sets of complex, sparse linear equations. Furthermore, the systems are very ill-conditioned. By developing specialized algorithms for solving these systems of equations on distributed memory, parallel supercomputers we have achieved the ability to calculate arbitrarily accurate, time-independent wave functions describing electron-hydrogen scattering above the ionization threshold. From these wave functions we can extract any differential cross section for ionization providing, for the first time, a complete description of electron-impact ionization.

3. MATHEMATICAL FORMALISM

3.1 Differential equation

There are no explicitly time dependent interactions so the system can be described by a wave function Ψ^+ that is a solution to the time-independent Schrödinger equation,

$$\mathrm{H}\Psi^+ = E\Psi^+ \,, \tag{1}$$

where E is the total energy of the system and H is the Hamiltonian describing the interaction of two electrons with each other and with the nucleus. The nucleus is assumed to be infinitely massive and fixed in space so Ψ^+ is a six-dimensional function of the coordinates ($\mathbf{r_1}$ and $\mathbf{r_2}$) for the two electrons relative to the nucleus. As a first step in correctly treating the boundary conditions of Ψ^+ we partition it into two terms,

$$\Psi^+(\mathbf{r_1},\mathbf{r_2}) = \Psi^0_{k_i}(\mathbf{r_1},\mathbf{r_2}) + \Psi^+_{\mathrm{sc}}(\mathbf{r_1},\mathbf{r_2}) \,. \tag{2}$$

The initial state of an electron with momentum $\mathbf{k_i}$ incident on a ground state hydrogen atom is described by $\Psi^0_{k_i}$,

$$\Psi^0_{k_i} = \tfrac{1}{\sqrt{2}}\left[\Phi_{1\mathrm{s}}(\mathbf{r_1})e^{i\mathbf{k_i}\cdot\mathbf{r_2}} + {}_{(-1)^S}\Phi_{1\mathrm{s}}(\mathbf{r_2})e^{i\mathbf{k_i}\cdot\mathbf{r_1}}\right] \,, \tag{3}$$

which is either symmetric (for total spin $S = 0$) or antisymmetric ($S = 1$) with respect to interchange of the electrons' coordinates. The remaning term, Ψ^+_{sc}, is referred to as the scattered wave and contains all of the scattering information in its asymptotic (large distances) limit. Although the asymptotic form of Ψ^+_{sc} still cannot be stated explicitly we do know that at large distances Ψ^+_{sc} is a purely outgoing wave. The scattered wave is calculated by solving the inhomogeneous differential equation

$$(E - \mathrm{H})\Psi^+_{\mathrm{sc}}(\mathbf{r_1},\mathbf{r_2}) = (\mathrm{H} - E)\Psi^0_{k_i}(\mathbf{r_1},\mathbf{r_2}) \,, \tag{4}$$

that comes from rearrangement of the Schrödinger equation, with outgoing wave boundary conditions on Ψ^+_{sc}.

3.2 Angular momentum expansion

The six-dimensional differential equation in Eq. 4 is converted to sets of coupled two-dimensional differential equations by expanding the wave function in terms of coupled spherical harmonics $\mathcal{Y}^{L0}_{l_1,l_2}(\hat{r}_1,\hat{r}_2)$. Like ordinary spherical harmonics, the $\mathcal{Y}^{L0}_{l_1,l_2}$ are orthonormal functions of the angular variables. They are labeled by the total angular momentum quantum number L and the single-electron angular

momentum quantum numbers l_1 and l_2. To calculate $\Psi^+_{\rm sc}$ we then need to evaluate the two-dimensional radial functions $\psi^L_{l_1 l_2}$ in its angular momentum expansion,

$$\Psi^+_{\rm sc}(\mathbf{r_1},\mathbf{r_2}) = \sum_{L,l_1,l_2} \frac{i^L}{r_1 r_2} \psi^L_{l_1 l_2}(r_1,r_2) \mathcal{Y}^{L0}_{l_1,l_2}(\hat{r}_1,\hat{r}_2) \; . \qquad (5)$$

The angular momentum expansion of the right-hand side of Eq. 4 is known analytically.

Substituting the expansions of $\Psi^+_{\rm sc}$ and $\Psi^0_{k_i}$ into Eq. 4 leads to sets of coupled, two-dimensional differential equations

$$\begin{aligned} &(E - {\rm H}_{l_1}(r_1) - {\rm H}_{l_2}(r_2))\, \psi^L_{l_1 l_2}(r_1,r_2) \\ &- \sum_{l_1',l_2'} \langle l_1 l_2 || l_1' l_2' \rangle_L \psi^L_{l_1',l_2'}(r_1,r_2) = \chi^L_{l_1 l_2}(r_1,r_2) \end{aligned} \qquad (6)$$

where the $\chi^L_{l_1 l_2}$ are the radial functions from the expansion of the right-hand side of Eq. 4, the $\langle l_1 l_2 || l_1' l_2' \rangle_L$ are two-dimensional coupling potentials arising from the electron-electron interaction, and the ${\rm H}_l$ are the one-dimensional, Coulomb radial Hamiltonians

$${\rm H}_l(r) \equiv -\frac{1}{2}\frac{d^2}{dr^2} + \frac{l(l+1)}{2r^2} - \frac{1}{r} \; . \qquad (7)$$

Since total angular momentum is a conserved quantity there is a separate set of coupled equations for each value of the quantum number L.

3.3 Simplifying the boundary conditions

The key element in our formalism is the exterior complex scaling transformation that simplifies the scattering boundary conditions for each of the $\psi^L_{l_1 l_2}$. Formally, the $\psi^L_{l_1 l_2}$ are zero along the coordinate axes ($r_1 = 0$ or $r_2 = 0$) but for large distances they are unbounded, oscilatory functions. In the absence of ionization the boundary conditions for large distances can be treated by matching to known asymptotic forms. No such usable asymptotic form is known for ionization.

We avoid having to explicitly specify the asymptotic form for ionization by calculating the $\psi^L_{l_1 l_2}$ on a complex contour [20, 16]. This transformation of the Schrödinger equation, called exterior complex scaling (ECS), was invented by Simon [24] to study molecular resonances in scattering theory. Both radial coordinates are rotated into the upper half of the complex plane beyond some distance R_0. This coordinate mapping,

$$r \rightarrow \begin{cases} r, & r < R_0, \\ R_0 + (r - R_0)e^{i\eta}, & r \geq R_0, \end{cases} \qquad (8)$$

(where $0 < \eta < \pi/2$) defines a box between zero and R_0 in r_1 and r_2 where both coordinates are real. Outside of that box at least one coordinate is complex. The effect of such a coordinate transformation on a purely outgoing wave is to transform it into a exponentially decaying function beyond R_0. An example of a $\psi^L_{l_1 l_2}$ calculated with the ECS transformation is shown in Figure 9. The calculated $\psi^L_{l_1 l_2}$ are identical to the unscaled $\psi^L_{l_1 l_2}$ inside the interior box but decay exponentially for either r_1 or r_2 greater than R_0. Thus, ECS simplifies the scattering boundary conditions so that the transformed $\psi^L_{l_1 l_2}$ satisfy Dirichlet boundary conditions.

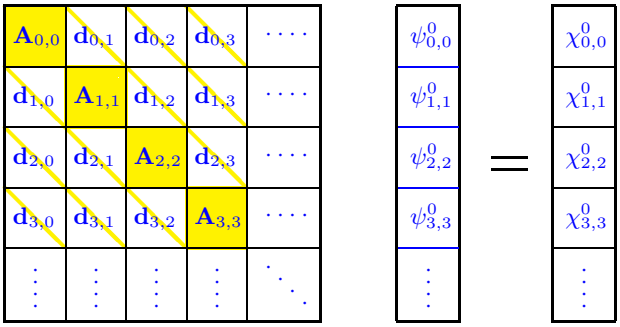


Figure 1: Block-matrix structure of the coupled equations (Equation 6) using $L = 0$ as an example. The $\mathbf{d}_{l_1,l_2}$ are diagonal matrices and the $\mathbf{A}_{l,l}$ are sparse, not diagonal, matrices.

3.4 The matrix problem

We solve the coupled equations in Eq. 6 for the $\psi^L_{l_1 l_2}$ on a two-dimensional radial grid using finite difference to approximate the derivatives. This results in a large linear system with a block-matrix structure illustrated in Figure 1. The dimension of each block is the number of grid points, and the number of blocks is the number of partial wave terms retained in the coupled equations. The matrix is complex non-Hermitian and non-symmetric. The right-hand side vector is formed from the values of each of the $\chi^L_{l_1 l_2}$ stored contiguously. Likewise, the solution vector is partitioned so that individual segments store the values of the corresponding $\psi^L_{l_1 l_2}$. The ordering of the (l_1, l_2) pairs is determined for each value of L by guessing the relative importance of the individual terms in the angular momentum expansion in Eq. 5.

Diagonal blocks, $\mathbf{A}_{l_1,l_2}$, are matrix representations of the two-dimensional operator,

$$\mathbf{A}_{l_1,l_2} \equiv E - {\rm H}_{l_1}(r_1) - {\rm H}_{l_2}(r_2) - \langle l_1 l_2 || l_1, l_2 \rangle_L \; . \qquad (9)$$

Each $\mathbf{A}_{l_1,l_2}$ has the sparsity structure of a two-dimensional, sixth-order (7-point formulas for each second derivative), finite difference Laplacian. The exact structure of the diagonal blocks is shown on the right-hand side of Figure 2. The $\mathbf{A}_{l_1,l_2}$ are complex, non-Hermitian because of the ECS transformation and they are non-symmetric because of the high-order finite difference formulas. The off-diagonal blocks are diagonal matrices representing the coupling potentials $\langle l_1 l_2 || l_1' l_2' \rangle_L$.

In order to obtain an accurate description of ionization we calculate the $\psi^L_{l_1 l_2}$ out to distances of at least $R_0 = 80a_0$ for higher energies and $R_0 = 140a_0$ for lower energies. One $a_0 = 5.29 \times 10^{-11}$ meters is the radius of a hydrogen atom in its ground state. The primary grid spacings range from $0.2a_0$ to $0.3a_0$. However, at small distances the grid spacing is $0.05a_0$ because of the singularity in the Coulomb potential. The grids typically extend beyond R_0 about $25a_0$. In this region the $\psi^L_{l_1 l_2}$ are exponentially decaying functions and larger grid spacings may be used. The number of grid points (in one dimension) used in our calculations ranges between 458 and 578 so the dimension of the individual blocks in Figure 1 ranges between 209,764 and 334,084. By using 7-point finite difference formulas we can calculate the $\psi^L_{l_1 l_2}$ very accurately on grids composed of sub-regions with uni-

form grid spacing. This is particularly important when using the ECS transformation given in Eq. 8 which requires that the finite difference formulas be generalized so that the grid "spacings" are complex beyond R_0. The number of blocks in the matrix equation illustrated in Figure 1 is determined by the number of (l_1, l_2) pairs kept in the angular momentum expansion (Eq. 5) for a particular value of L. Typically, the number of blocks ranges between 6 (for $L = 0$) and 24 (for higher L). Thus, for a single set of coupled equations the size of the system of complex, linear equations that we solve can be as large as 8 million.

4. OVERVIEW OF THE PARALLEL ALGORITHMS

Because of the size of the matrix (dimension up to 8 million), we need to use an iterative algorithm for the linear systems. We devloped a two-level, iterative algorithm for solving the sets of coupled differential eqution. Here we give an overview of the algorithm and our parallelization strategy. The detailed algorithms and performance appear in Sections 5 and 6. Since there is no coupling between $\psi^L_{l_1 l_2}$ with different values of the quantum number L, there is an independent set of coupled equations for each L. Rather than having an "embarrassingly parallel" component of our algorithm we solve the coupled equations for each L independently. Thus, our two-level, parallel algorithm is designed to solve a single set of coupled equations for some value of L.

The first level of our algorithm is based on the block-matrix representation of the coupled equations illustrated in Figure 1. We solve the coupled equations iteratively using solution to the *uncoupled* equations as a block-Jacobi preconditioner. Each diagonal block in Figure 1 is the matrix for one of the *uncoupled* equations (i.e. with $\langle l_1 l_2 || l_1' l_2' \rangle_L = 0$ for $(l_1, l_2) \neq (l_1', l_2')$).

We also use an iterative algorithm for solving the uncoupled equations which are themselves large linear systems of equations. This inner iteration level accounts for the bulk of the computational work. For the preconditioning step in the inner iteration we use a direct solver to solve the equations that have the same dimension but are more sparse than the original matrix. For our application, the key advantage of using the iterative algorithm for solving each uncoupled equation, compared with using a direct solver, is that much less memory is required for storing the LU-factors of the preconditioner than storing the factors of the original matrix. Because of the memory savings we can use a smaller number of processors per (l_1, l_2) pair in the coupled equations.

Sparse direct solvers are much harder to parallelize, a task much too involved for an application programmer to spend time on. In recent years, some new algorithms and software packages have emerged which exploit new architectural features, such as memory hierarchy and parallelism. Examples of publically available, parallel unsymmetric solvers include MUMPS [1] (multifrontal algorithm), SPOOLES [3] (left-looking algorithm), SuperLU [14] (right-looking algorithm), and WSMP [12] (multifrontal algorithm). WSMP is tuned particularly for the IBM SP architecture, however, it only has support for shared memory parallelism. MUMPS does not have support for complex matrices. In a separate work, we compared MUMPS and SuperLU only for the real matrices on the Cray T3E, up to 512 processors. SuperLU often uses less memory and scales better, and MUMPS is usually faster on smaller number of processors. See [2] for detailed comparison results. For our application, SuperLU seems to be the only choice because of support for both complex matrices and distributed memory machines.

In solving these subsystems using SuperLU, we first reorder the equations and variables using a minimum degree algorithm [15], applied on the graph of $A^T + A$, to reduce the fill-ins in the LU-factors. In the initial stage of the development, we also experimented with nested dissection ordering applied on $A^T + A$, but the fill reduction is not better than using minimum degree ordering for our 2D meshes. So we did not pursue that any further.

The block-matrix structure (see Figure 1) provides a natural, "coarse" level of parallelism. We divide the total number of processors into processor subgroups of equal size. Each subgroup is assigned to a particular (l_1, l_2) pair. Therefore, the number of processor subgroups scales directly with the number of terms (for a particular L) that are kept in the angular momentum expansion given in Eq. 5. The bulk of the computations, such as solving individual uncoupled equations, are then local to individual subgroups. We typically use four processors for each subgroup. In the case of the Cray T3E-900, on which these codes were initially developed, this was the minimum number of nodes required for solving a single uncoupled equations because of memory limitations. On newer machines such as NERSC's current IBM SP it is possible to solve the same uncoupled equations with fewer processors, but we still find that using four processors per subgroup strikes a good balance between absolute time and efficiency.

5. UNCOUPLED EQUATIONS AS PRECONDITIONER

5.1 Solving the uncoupled equations

Our iterative algorithm for solving the coupled equations in Eq. 6 requires solution to uncoupled equations, defined by setting $\langle l_1 l_2 || l_1' l_2' \rangle_L = 0$ for $(l_1, l_2) \neq (l_1', l_2')$, that have the form

$$\mathbf{A}_{l_1,l_2} \mathbf{x}^L_{l_1,l_2} = \mathbf{b}^L_{l_1,l_2} \ , \tag{10}$$

where $\mathbf{A}_{l_1,l_2}$ is shown in Eq. 9. Even for a single uncoupled equation the dimension of the linear system can be very large, up to 334,084 in production runs and more than 2 million for testing purposes. Each uncoupled equation is solved by the processor group assigned to that (l_1, l_2) pair. Since this is the most computationally intensive step in our algorithm most of the time in solving the coupled equations is spent in computations that are local to processor groups.

The matrix structure of the uncoupled equations (for a very small example) is pictured in the right-hand side of Figure 2. The structure of the corresponding LU-factors is shown in the right-hand side of Figure 3. Direct LU-factorization of the high-order, finite difference matrix will quickly exhaust available time and memory resources as we scale up the size of the system. Ultimately, we need to solve many of these systems simultaneously. Thus we need a parallel iterative solver to solve the uncoupled equations with a modest number of processors.

Unfortunately, the ECS transformation, which is necessary for simplifying the scattering boundary conditions, causes

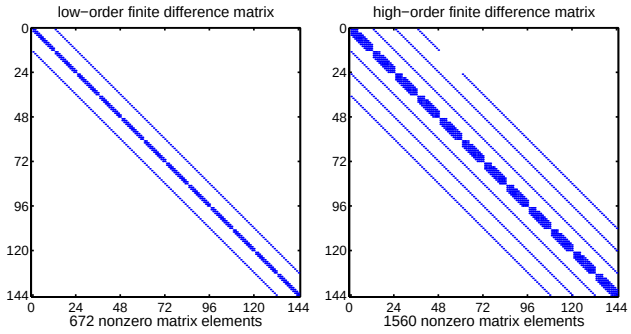


Figure 2: Sparsity structure of the finite difference matrix of the two-dimensional Hamiltonian. On the left is the low-order matrix which uses 3-point formulas for the second derivates. On the right is the high-order matrix which uses 7-point formulas. These examples are very small (144 total grid points extending only to $2a_0$) so that the basic structure can be seen.

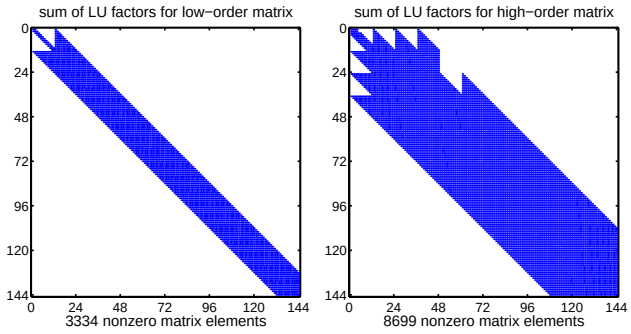
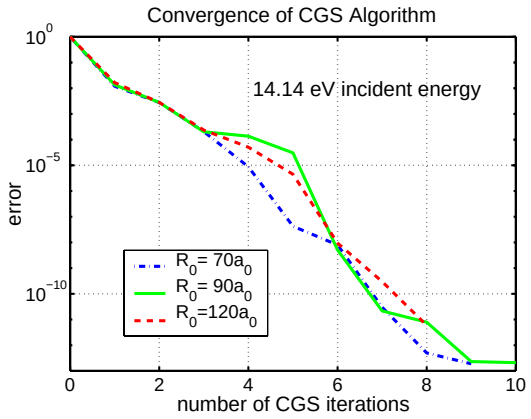


Figure 3: Sparsity structure of the LU factors of the matrices in Figure 2. The factors U and L are upper and lower-triangular matrices, respectively. The sparsity of the sum $L+U$ is shown here.

Preconditioned Conjugate Gradient Squared Algorithm

Start with initial guess $\mathtt{x}^{(0)}$
Compute $\mathtt{r}^{(0)} = \mathtt{b} - \mathbf{A}\mathtt{x}^{(0)}$
for $i = 1$ to max_iterations
  $\rho_{i-1} = \mathtt{b}^T\mathtt{r}^{(i-1)}$
  if $\rho_{i-1} = 0$ method fails
  if $i = 1$ **then**
    $\mathtt{u}^{(1)} = \mathtt{r}^{(0)}$
    $\mathtt{p}^{(1)} = \mathtt{r}^{(0)}$
  else
    $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    $\mathtt{u}^{(i)} = \mathtt{r}^{(i-1)} + \beta_{i-1}\mathtt{q}^{(i-1)}$
    $\mathtt{p}^{(i)} = \mathtt{u}^{(i)} + \beta_{i-1}\left(\mathtt{q}^{(i-1)} + \beta_{i-1}\mathtt{p}^{(i-1)}\right)$
  endif
  solve $\mathbf{M}\hat{\mathtt{p}} = \mathtt{p}^{(i)}$
  $\hat{\mathtt{v}} = \mathbf{A}\hat{\mathtt{p}}$
  $\alpha_i = \rho_{i-1}/\mathtt{b}^T\hat{\mathtt{v}}$
  $\mathtt{q}^{(i)} = \mathtt{u}^{(i)} - \alpha_i\hat{\mathtt{v}}$
  solve $\mathbf{M}\hat{\mathtt{u}} = \mathtt{u}^{(i)} + \mathtt{q}^{(i)}$
  $\mathtt{x}^{(i)} = \mathtt{x}^{(i-1)} + \alpha_i\hat{\mathtt{u}}$
  $\mathtt{r}^{(i)} = \mathtt{b} - \mathbf{A}\mathtt{x}^{(i)}$
  error $= \|\mathtt{r}^{(i)}\|$
  if error $<$ tolerance **exit**
end

Figure 4: The preconditioned Conjugate Gradient Squared algorithm based on the one given in [7, pp.26]. Matrix M is the preconditioner. We define the arbitrary vector $\tilde{\mathtt{r}}$ in [7] to be the driving term $\mathtt{b}$. Also the full residual $\mathtt{r}^{(i)}$ is computed in each iteration rather than updating the previous residual.



Iterative Algorithm

factorization time:	42.4	seconds
single solve time:	1.57	seconds
solves required:	14	
iteration time:	24.0	seconds
total time:	66.9	seconds

Direct Solution

factorization time:	897	seconds
solve time:	2.93	seconds
total time:	937	seconds

Figure 5: Convergence of the CGS algorithm for a single "uncoupled" equation for three grids which are real out to different values of R_0 is shown on the left. The time required for an $R_0 = 60a_0$ calculation on a single 200Mhz Power3 CPU is shown on the top right. In this case, the total number of grid points is 88,804. The preconditioner is applied twice in each CGS iteration. The time required to solve directly is shown on the bottom right.

each $\mathbf{A}_{l_1,l_2}$ to be very ill-conditioned. We tested various iterative algorithms on small, one-dimensional problems to see which algorithms are compatible with ECS. Every algorithm that we tried failed to converge for the test problems without preconditioning. Furthermore, using various standard preconditioners failed to cause any of these algorithms to converge. We obtained convergence only when we solved linear equations with the lowest order (i.e. three-point formula) finite difference matrix as a preconditioning step in iteratively solving the linear equation for high order (seven-point formula) finite difference. Using low-order finite difference as a preconditioner for solving the high-order matrix equation caused a few of the Krylov subspace methods (CGS, Bi-CGStab, and GMRES) to converge. All had about the same stability and convergence rate. We chose to use the CGS algorithm [7], outlined in Figure 4, because it requires the least amount of memory.

Convergence of this iterative algorithm is shown in Figure 5. The preconditioning step is accomplished by using SuperLU to directly solve the low-order matrix equation. Also given is a comparison between the time required for the iterative algorithm and for using SuperLU to directly solve the high-order matrix equation. As can be seen in Figure 2, the low-order finite difference matrix is much sparser. Therefore, an LU-factorization algorithm that takes advantage of the structure of the matrix can solve the low-order equations much more quickly than the high order equations. Why this is so is illustrated by the sparsity patterns of the corresponding LU factors, shown in Figure 3. The timings listed in Figure 5 includes a breakdown of the time spent in different parts of the algorithm. SuperLU computes the LU-factors for the low-order matrix in much less time than it takes for the high-order matrix. Even though many triangular solutions using the LU-factors are required, the total time of the iterative algorithm is about 7% of the time needed for the direct solution.

5.2 SuperLU as preconditioner for uncoupled equations

A distinct advantage of the direct method is its *robustness*, in the sense that it involves a fixed number of floating point operations independent of the conditioning. Sparse Gaussian elimination is much harder to parallelize than iterative methods, mainly because of the *fill-ins* in the LU factors. If we use classical partial pivoting, those fill-ins are generated on the fly as factorization proceeds, which requires dynamically adaptive data structures to represent the matrix. This incurs prohibitive cost on parallel machines because of many fine-grained messages. Our novel static pivoting strategy overcomes this difficulty and maintains numerical stability [14]. Another challenge to parallelizing this algorithm is the existence of many task dependencies among different elimination steps. We have to exploit as much as possible the parallelism across multiple steps while preserving these dependencies. We spent much time improving the parallel factorization and triangular solve algorithms.

5.2.1 *matrix distribution and parallel algorithms*

The matrix partitioning is based on the notion of an *unsymmetric supernode*, which consists of consecutive columns of L with the diagonal block being full, and the same nonzero structure elsewhere. This supernode partition is used as the block partition in *both* row and column dimensions. Figure 6 illustrates such a block partition. The P processes are also arranged as a 2D grid of dimension $P_r \times P_c = P$. We use 2D block-cyclic layout, meaning block (I, J) (of L or U) is mapped onto the process at coordinate $((I-1) \bmod P_r, (J-1) \bmod P_c)$ of the process grid. In this 2D mapping, each block column of L is spread across every processor in a single column of the process grid. For example in Figure 6, the second block column of L resides on the column processes $\{1, 4\}$. Process 1 only owns two nonzero blocks, which are not contiguous in the global matrix. The advantages of this 2D mapping over a 1D mapping are reduced communication, enhanced load balance and scalability. The user can set the shape of the process grid, such as 2×3 or 3×2. The more square the grid, the better the performance expected. This rule of thumb was used in our computations to define the grid shapes.

The parallel sparse factorization algorithm is right-looking and loosely synchronous. At the K-th step, it factors the K-th block column of L and the K-th block row of U. Then, using the outer-product of these two factored block column and row, it performs the updates to the trailing submatrix. All these operations are performed in parallel. The actual implementation uses a pipelined organization and the non-

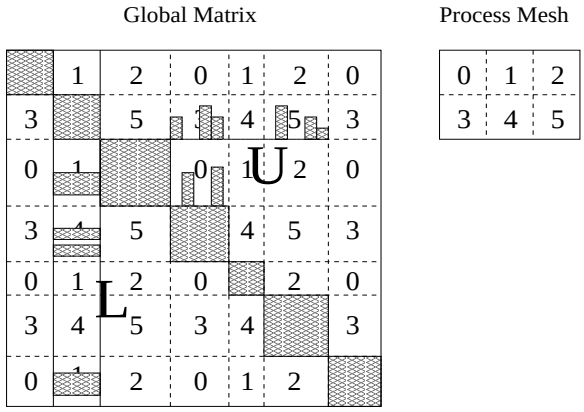


Figure 6: SuperLU 2D block-cyclic mapping of matrix to processors.

blocking send and receive (`MPI_Isend/MPI_Irecv`) so that independent tasks across multiple loop iterations are scheduled simultaneously, thus exploiting better parallelism and overlaping computation and communications.

The triangular solve algorithm is fully asynchronous and is based on a sequential variant called "inner product" formulation. The execution of the program is completely *message-driven*. Each process is in a self-scheduling loop, performing appropriate local computation depending on the type of message received. This approach enables large overlap between communication and computation and helps overcome the much higher communication to computation ratio in this phase.

5.2.2 SuperLU performance and scalability

To illustrate the performance and scalability of SuperLU, we report the results obtained with the high-order systems in Table 1. The grid sizes were chosen so that with increasing number of processors, the number of factorization operations per processor is kept roughly constant. Table 1 lists the grid sizes, the number of operations, the timings and the megaflop rate per processor on the Cray T3E-900 (DEC EV-5 processors, 256 Mbytes memory per processor, 450 MHz clock rate) at NERSC.

For the *LU* factorization, the number of operations is almost constant per processor ($\approx 8 \times 10^9$). The parallel time increases slowly, and megaflop rate per processor decreases slowly. The parallel efficiency drops slowly but still maintains at 50% level even with 64 processors. So the factorization phase scales quite well. For the triangular solution, the number of operations increases at a lower rate than the factorization. But the megaflop rate per processor decreases more rapidly, meaning the algorithm is less scalable. This is because in this phase, there is a higher ratio of communication over computation. On the other hand, the triangular solution time is always less than 4% of the factorization time.

5.3 Performance data for uncoupled equations

The complex-scaled, 2D Hamiltonian matrices in our application exhibit different SuperLU performance characteristics than many other matrices, such as the 3D problems. Here are our observations:

- The matrices are very sparse—about 5 nonzeros per row in a low-order matrix or 13 nonzeros per row in a high-order matrix, independent of the grid size. Furthermore, the matrices remain sparse during LU factorization. The fill-in growth rate of $L + U$ over the original A is between 10 and 20 for low-order matrices with dimension up to 2 million. Whereas for many 3D problems, the growth rate can be more than an order of magnitude higher.

- The matrix structure is very irregular in that the dense blocks identified in L and U are very small. Therefore, more integer indices are required to represent the sparsity structure, resulting in more indirect addressing in the computations.

These properties lead to lower memory usage and possibly faster runtimes, but also lower megaflop rates. Table 2 gives the detailed matrix statistics and our solver performance for several largest uncoupled equations (dimension up to 2 million). For each grid size, we compare the CGS solution times with SuperLU solving either low-order equations as preconditioner or high-order equations directly. The average block size is smaller for low-order matrix, and the fraction of integer indices (hence the amount of indirect addressing) is higher. That is why the megaflop rate is much lower for the low-order matrices. This is particularly true for the triangular solves, because there is less computation but more communication. Although SuperLU gives a much better megaflop rate for high-order matrices in both factorization and triangular solution, the total memory requirement is about an order of magnitude larger. Sometimes direct solution of high-order matrix can be faster (see the case $R_0 = 180$) because the triangular solution is much more efficient and scalable for the high-order matrix.

For such sparse systems, the metric for high performance cannot be a mere megaflop rate, because there are many unavoidable integer operations and indexed loads/stores that do not use the floating point unit. What is important is the time for solution and the memory usage. Table 3 illustrates this point. Here, we compare three solvers for problems of increasing size on a single processor. The LAPACK banded solver delivers the highest megaflop rate, but is the slowest and most demanding in memory. It uses simple and efficient data structures at the expense of storing and operating on many zero entries in the matrix. Using SuperLU to directly solve the matrix equation gives slightly reduced megaflop rates, but takes much less time. The iterative solver strikes a good balance between numerical efficiency and the use of computer resources, therefore it is the fastest and demands the least amount of memory, even though it gives the lowest megaflop rate. Just as important, it increases the size of the problem that we can solve with a fixed amount of memory.

Performance of our parallel CGS algorithm using SuperLU on the low-order matrix as preconditioning is listed in Table 4. The time is broken down into Factor time from SuperLU, Iteration time, and Total time. The total time is the sum of the first two plus some set-up time. For these matrices, it takes 7 to 8 CGS iterations to converge. Each iteration requires two triangular solutions from SuperLU, which accounts for a large fraction of the iteration time. It is clear that SuperLU factorization scales quite well, and it constitutes a large fraction of the total time on smaller numbers of processors (up to about 8). For more processors, the CGS iteration time starts to surpass the factorization time, because the triangular solution algorithm does not scale as

Table 1: SuperLU performance scaling with the high-order systems on the CRAY T3E-900.

Nprocs	1	2	4	8	16	32	64
Grid size	$R_0 = 21$	$R_0 = 29$	$R_0 = 39$	$R_0 = 50$	$R_0 = 65$	$R_0 = 80$	$R_0 = 100$
Matrix order	20,164	30,276	45,796	66,564	101,124	142,884	209,764
Nonzeros in A (10^6)	0.3	0.4	0.6	0.9	1.3	1.8	2.7
Nonzeros in $L+U$ (10^6)	6.9	11.5	19.3	31.1	51.6	80.3	128.1
LU Factorization							
Flops (10^9)	8.7	17.3	35.1	69.3	134.7	257.6	498.3
Time (seconds)	28.4	31.9	34.0	35.8	39.9	43.5	52.5
Mflops	307.7	541.6	1031.5	1937.2	3373.5	5921.3	9490.4
Mflops per proc	307.7	270.8	257.9	242.2	210.8	185.1	148.3
Triangular Solution							
Flops (10^6)	55.9	92.9	156.0	252.0	417.0	648.7	1034.7
Time (seconds)	0.8	1.0	1.0	1.2	1.2	1.4	1.5
Mflops per proc	67.8	46.8	39.0	26.4	21.9	14.1	10.4

Table 2: CGS solution of uncoupled equations on 64 processors of the IBM SP at NERSC. SuperLU solves either the low-order systems as preconditioner or the high-order systems directly. "Average block" is the average number of columns in a dense block, see Figure 6. "%Index" is the percentage of integer indices used in the compact sparse storage over the number of nonzeros in L and U.

	Matrix Properties						Solver Performance		
Grid size	Order (10^6)	Nonzeros (10^6)	Fill-in ratio	Average block	%Index	Memory (MB)	Factor sec. (Mflops)	Tri. Solve sec (Mflops)	CGS time
$R_0 = 180$									
low-ord	0.6	3.0	15	4	12%	844	53.5 (1000)	82.8 (5)	2307.7
high-ord	0.6	7.9	55	8	4%	7149	230.8 (10822)	21.5 (163)	415.0
$R_0 = 240$									
low-ord	1.0	5.2	17	8	8%	1599	81.1 (1431)	60.2 (13)	2116.1
high-ord	1.0	13.4	64	14	3%	14096	1209.3 (5570)	25.6 (271)	2387.6
$R_0 = 340$									
low-ord	2.0	10.0	19	8	8%	3295	188.3 (1792)	371.8 (4)	6361.9
high-ord	2.0	26.1	**out of memory**						

Table 3: Comparison of times in seconds (and Mflops) to solve a single uncoupled equation using the iterative algorithm, direct solution with SuperLU, and direct solution with a banded solver in LAPACK, on a single 200 MHz Power3 CPU.

Grid size	10	20	40	60	80	100
Matrix Order	9,604	19,044	47,524	88,804	142,884	209,764
Banded Solver:						
factor	11.7 (568)	44.1 (592)	**out of**			
solve	0.29 (78)	0.90 (70)	**memory**			
total	12.0	45.0				
SuperLU (direct):						
factor	5.83 (433)	17.0 (460)	76.8 (505)	220 (503)	504 (509)	
solve	0.34 (93.8)	0.48 (106)	1.40 (115)	2.93 (118)	**did not**	
total	7.63	21.2	91.5	281.5	**finish**	
CGS + SuperLU:						
factor	0.70 (115)	1.55 (153)	4.86 (212)	10.5 (258)	19.6 (288)	33.2 (324)
iteration	2.45 (22)	5.03 (24)	13.0 (26)	24.0 (29)	30.6 (30)	60.5 (30)
total	3.96	8.96	28.4	66.9	138	254

Table 4: The parallel runtimes in seconds of the preconditioned CGS algorithm to solve a single uncoupled equation of various sizes on the IBM SP at NERSC, with 200 MHz Power3 CPUs.

Grid size	Matrix dimension	Time	#Processors						
			1	2	4	6	8	12	16
$R_0 = 60$	88,804	Factor	42.4	27.1	17.3	14.5	13.3	11.7	10.7
		Iter.	24.0	21.7	19.9	19.3	14.3	18.4	23.8
		Total	66.9	49.2	37.7	34.2	27.9	30.5	34.8
$R_0 = 100$	209,764	Factor	192.4	112.9	65.9	51.1	45.1	37.0	31.6
		Iter.	60.5	51.3	47.2	48.5	35.5	46.6	53.4
		Total	253.9	165.2	114.2	100.5	81.7	84.6	86.1
$R_0 = 140$	381,924	Factor	852.6	327.3	184.8	142.9	111.8	86.7	73.4
		Iter.	155.2	108.1	100.5	104.5	76.0	96.2	130.3
		Total	1037.6	437.3	287.1	249.2	189.5	184.7	205.5
$R_0 = 180$	605,284	Factor			411.4	294.6	257.4	192.1	147.4
		Iter.			160.8	161.8	128.2	156.9	203.6
		Total			575.1	459.3	388.6	351.7	353.9

well.

6. ITERATIVE SOLUTION TO COUPLED EQUATIONS

6.1 Parallel implementation

We solve the coupled equations (Eq. 6) with an iterative algorithm using solution to uncoupled equations (Eq. 10) as a preconditioner. Since a preconditioned CGS algorithm worked well in solving the uncoupled equations, we also use CGS for solving the coupled equations. We have found that convergence is reliable in all cases of interest with convergence being faster at higher energies where the diagonal-blocks (see Figure 1) are more dominant.

Our basic parallelization strategy is to partition the total number of processors into small subgroups of equal numbers (usually four or six) of CPUs. Each subgroup is then assigned to a particular (l_1, l_2) pair. In doing this we exploit the symmetry relation between (l_1, l_2) and (l_2, l_1),

$$\psi^L_{l_2 l_1}(r_2, r_1) = {}^{(-1)^S}\psi^L_{l_1 l_2}(r_1, r_2) \ , \qquad (11)$$

by explicitly storing only the $\psi^L_{l_1 l_2}$ with $l_1 \leq l_2$. In the iterative algorithm we include the $l_1 > l_2$ terms implicitly. Therefore, the number of processor subgroups is equal to the number of (l_1, l_2) terms kept in the coupled equations with $l_1 \leq l_2$. However, the number of (l_1, l_2) blocks in the matrix equation that we are solving may actually be as high as twice that number, depending upon the value of L.

The preconditioning steps are by far the most time consuming operations in the algorithm. Since these are just solutions to the uncoupled equations the preconditioning steps consist of calculations that are entirely local to each processor subgroup. The preconditioner is applied twice for each CGS iteration and is accomplished by each processor subgroup implementing the algorithm described in Section 5. The second most time-consuming operations in the algorithm are matrix-vector multiplies between the diagonal subblocks $\mathbf{A}_{l_1,l_2}$ and the vectors for their corresponding $\psi^L_{l_1 l_2}$. Again, these operations are entirely local to a processor subgroup. The only operations that require communication between different subgroups are vector operations such as vector-vector adds, diagonal matrix-vector multiplies, and scalar-vector multiplies.

6.2 Performance

Typically, we solve 20 separate sets of coupled equations for each collision energy. We must calculate separate wave functions for the two spin symmetries ($S = 0$ and $S = 1$) each of these requires solving the coupled equations for total angular momentum quantum numbers ranging from $L = 0$ to $L = 9$. The size of each calculation depends on the size of the two-dimensional grid used and the number of (l_1, l_2) pairs. As an example, a calculation for 20 eV collision energy used a grid that extends to $130a_0$ and has 209,764 two-dimensional grid points. We can use a smaller grid for higher energies while a large grid is required for lower energies. For a particular energy, the same two-dimensional grid is used for every $\psi^L_{l_1 l_2}$.

We include a minimum of six (l_1, l_2) pairs in each coupled equation. For $L = 0$ and $L = 1$ this is usually sufficient. With increasing L we need to include more terms in the coupled equations. For $L = 6$ we typicaly include 16 (l_1, l_2) pairs. Beyond $L = 6$ the relative importance of individual $\psi^L_{l_1 l_2}$ to the angular momentum expansion diminishes and we can slightly reduce the number of (l_1, l_2) pairs per coupled equation. Higher energies require more (l_1, l_2) pairs per coupled equation, as many as 24 in our calculations.

In Figure 7 we present examples of the convergence of the iterative algorithm for solving the coupled equations. The behavior that we typically see is that the error in the computed residual increases during the first few iterations, but then decreases fairly steadily. The rate of convergence is dependent upon a number of factors: the number of (l_1, l_2) pairs, the total energy, and the quantum number L. For sufficient numbers of (l_1, l_2) pairs the convergence rate depends sublinearly on the number of included pairs. Whenever we list the number of (l_1, l_2) pairs for a given coupled equation we are only counting those for which $l_1 \leq l_2$ so the actual number of terms in the coupled equation could be higher by as much as a factor of two, depending upon the value of L.

The energy being considered also affects the rate of convergence. For higher energies, the coupling between different $\psi^L_{l_1 l_2}$ is relatively weak and the iterative algorithm converges more rapidly than at lower energies. The two energies used as examples in Figure 7 represent the extremes of this dependence. In the case of 54.4 eV collision energy, the highest that we have considered, the coupled equations always converge within a dozen CGS iterations. The lowest collision

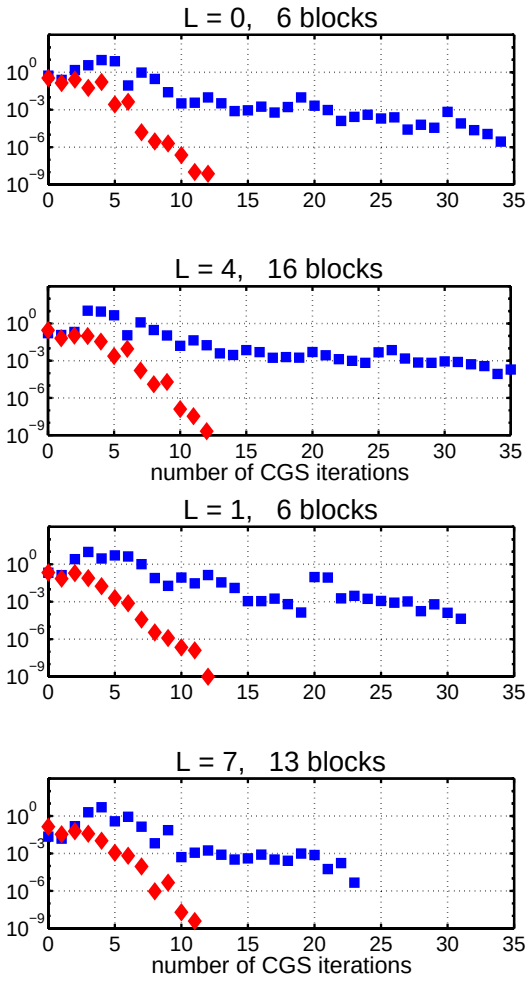


Figure 7: Convergence of the CGS algorithm for the coupled equations with singlet spin symmetry for various total angular momenta L. Error of the calculated scattered wave is plotted for collision energies of 15.6 eV (squares) and 54.4 eV (diamonds). For each L the number of blocks included, counting only those with $l_1 \leq l_2$, is listed.

energy we have considered is 15.6 eV, just 2 eV above the ionization threshold. At this energy the convergence rate is much slower and more dependent on L than at the higher energy. We have found that the present algorithm sometimes fails to converge for collisions energies within about 1 eV above the ionization threshold.

In our production runs we use the smallest number of processors per (l_1, l_2) pair, usually four, that will solve an uncoupled equation in a reasonable amount of time. This is a prudent management of resources because the solve step using the LU factors computed by SuperLU does not scale well. In our two-level iterative algorithm the SuperLU solve step is executed hundreds of times. From Figure 7 we see the number of outer CGS iterations ranges from 12 to 35. Since each CGS iteration requires two preconditioning steps, 24 to 70 solutions of each uncoupled equation are required to solve one coupled equation. Each uncoupled equation solution typically require 8 CGS iterations for a total of 16 SuperLU

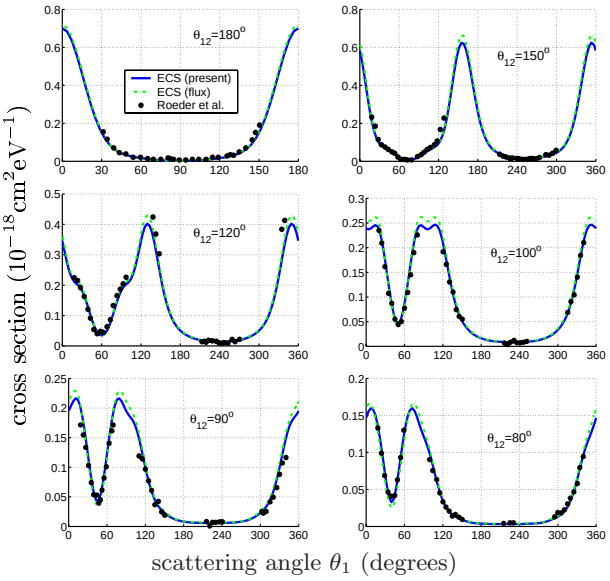


Figure 8: Equal-energy sharing, coplanar TDCS for 25 eV incident energy with θ_{12} fixed. Internormalized measurements [22], originally reported in arbitrary units, were multiplied by 0.16 to fit calculated cross section. Solid and dashed curves represent two different methods of calculating the TDCS [5,4] .

solves. Thus, between 384 and 1120 SuperLU solves for each (l_1, l_2) pair are used to converge a set of coupled equations.

The total number of processors used scales directly with the size of the system and is four times the number of (l_1, l_2) blocks, ranging from 24 to 96. Using the example of a 209,764 point grid we are solving for between 1.2 million and 5 million double precision, complex numbers for each coupled equation. The actual dimension of the linear system being solved iteratively may be as high as twice that number. Production runs generally take 2 to 3 minutes per iteration on an IBM SP with 200 MHz Power 3 CPUs. Taking into account the set-up time for computing the LU factors and the extremely variable number of iterations required the total time for solving a coupled equation ranges from 30 to 500 minutes on the SP. The longer runs are generally accomplished in multiple stages. Our codes are written so that the solution vectors are periodically saved during the iterative algorithm. If convergence has not been reached within pre-determined limits, or if a system crash occurs before completion, the iterative algorithm is restarted using the last solution that was saved.

7. SCIENTIFIC RESULTS

The real significance of our work lies in the scientific accomplishment. Our results represent the first truly complete solution of the electron-impact ionization problem. This would not have been possible without modern, scalable numerical algorithms, such as SuperLU, and massively parallel computers, such as the Cray T3E and the IBM SP. With these tools we were able to implement the exterior complex scaling formalism on a scale large enough to produce accurate and detailed differential cross sections for electron-

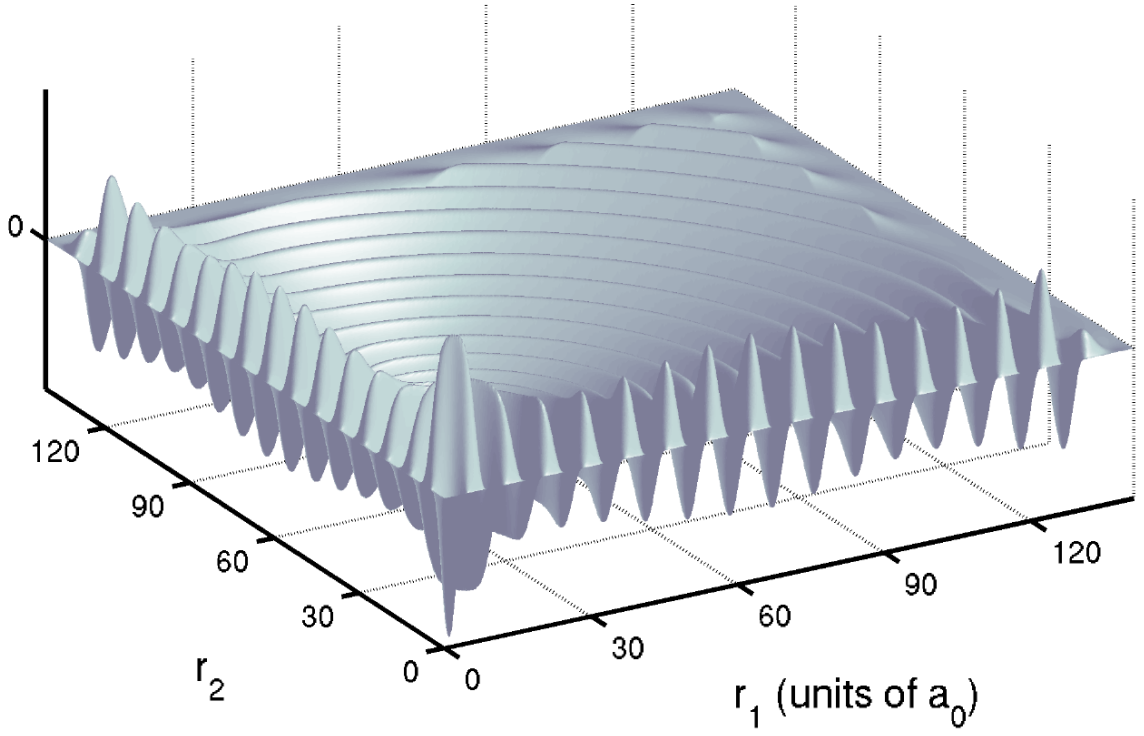


Figure 9: Real part of an example radial function. In this example $l_1 = l_2$ so the radial function is symmetric about $r_1 = r_2$.

impact ioniztion.

A comparison between some of our calculated differential cross sections and the corresponding experimental data is shown in Figure 8. More complete sets of results have been published elsewhere [21, 4, 13, 5]. This experimental data is available only in arbitrary units so the entire set of data must be multiplied by a single normalization factor which was chosen based on comparison with our calculations. Even so, agreement in shape between our calculations and the experiment is a remarkable testimony to the accuracy of our calculated results. However, our goal was not just to reproduce the experimental data. The experiments are very difficult and only a very limited set of data exists [22], most of which is available only in arbitrary units. Ultimately, a complete description of electron-impact ionization must come from a theoretical treatment.

The computationally intensive part of this endeavor was calculating the wave function using the algorithm described in the preceding parts of this paper. An example of one of the two-dimensional radial functions used to construct the wave function is shown in Figure 9. Although an intermediate step in the process of calculating the cross sections, the calculated wave functions, themselves, represent a significant scientific acheivement. Never before has a fully converged wave function that describes two outgoing electrons been calculated out to such large distances. These wave functions will assist scientists in better understanding the complex dynamics of electron-impact ionization as they seek to develop theoretical and numerical methods appropriate for more complex systems.

8. ACKNOWLEDGMENTS

The exterior complex scaling formalism was developed in collaboration with C. William McCurdy and Thomas N. Rescigno.

These calculations were performed on the Cray T3E-900 and the IBM SP computers at the National Energy Research Scientific Computing Center and on the IBM "Blue-Pacific" machine at the Lawrence Livermore National Laboratory.

9. REFERENCES

[1] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[2] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and X. S. Li. Analysis and comparison of two general sparse solvers for distributed memory computers. *ACM Transactions on Mathematical Software*, (to appear), 2001. Also Lawrence Berkeley National Laboratory report LBNL-45992.

[3] C. Ashcraft and R. Grimes. SPOOLES: An object-oriented sparse matrix library. In *Proceedings of the Ninth SIAM Conference on Parallel Processing*, 1999.

[4] M. Baertschy, T. N. Rescigno, W. A. Isaacs, X. Li, and C. W. McCurdy. Electron-impact ionization of atomic hydrogen. *Phys. Rev. A*, 63:022712, 2001.

[5] M. Baertschy, T. N. Rescigno, and C. W. McCurdy. Accurate amplitudes for electron-impact ionization. *Phys. Rev. A*, 64:022709, 2001.

[6] M. Baertschy, T. N. Rescigno, C. W. McCurdy, J. Colgan, and M. S. Pindzola. Ejected-energy differential cross sections for the near threshold electron-impact ionization of hydrogen. *Phys. Rev. A*, 63:050701R, 2001.

[7] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, V. Pozo, C. Romine, and H. van der Vorst. *Templates for the solution of linear systems: Building blocks for the iterative methods*. SIAM, Philadelphia, 1994.

[8] K. Bartschat, E. T. Hudson, M. P. Scott, P. G. Burke, and V. M. Burke. Convergent r matrix with pseudostates calculation for e-he collisions. *Phys. Rev. A*, 54:R998, 1996.

[9] I. Bray. Close-coupling theory of ionization: Successes and failures. *Phys. Rev. Lett.*, 78:4721, 1997.

[10] I. Bray. Electron-impact ionization of atomic hydrogen from near threshold to high energies. *J. Phys. B*, 33, 2000.

[11] J. W. Demmel, J. R. Gilbert, and X. S. Li. SuperLU Users' Guide. Technical Report LBNL-44289, Lawrence Berkeley National Laboratory, September 1999. Software is available at http://www.nersc.gov/∼xiaoye/SuperLU.

[12] A. Gupta. WSMP: Watson Sparse Matrix Package. Technical report, IBM research division, T.J. Watson Research Center, Yorktown Heights, 2000. http://www.cs.umn.edu/∼agupta/wsmp.html.

[13] W. A. Isaacs, M. Baertschy, C. W. McCurdy, and T. N. Rescigno. Doubly differential cross sections for the electron impact ionization of hydrogen. *Phys. Rev. A*, 63:030704R, 2001.

[14] X. S. Li and J. W. Demmel. Making sparse gaussian elimination scalable by static pivoting. In *Proceedings of the ACM/IEEE SC 98 Conference*, Los Alamitos, CA, November 1998. IEEE.

[15] J. W. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Trans. Math. Software*, 11:141–153, 1985.

[16] C. W. McCurdy, T. N. Rescigno, and D. A. Byrum. Approach to electron-impact ionization that avoids the three-body coulomb asymptotic form. *Phys. Rev. A*, 56:1958, 1997.

[17] C. Pan and A. F. Starace. Angular distributions for near-threshold (*e,2e*) processes for h, he, and other rare-gas targets. *Phys. Rev. A.*, 45:4588, 1992.

[18] R. K. Peterkop. *Opt. Spectrosc.*, 13:87, 1962.

[19] M. S. Pindzola and F. J. Robicheaux. Time-dependent close-coupling calculations for the electron impact ionizatiuon of helium. *Phys. Rev. A*, 61:052707–1, 2000.

[20] T. N. Rescigno, M. Baertschy, D. Byrum, and C. W. McCurdy. Making complex scaling work for long-range potentials. *Phys. Rev. A*, 55:4253, 1997.

[21] T. N. Rescigno, M. Baertschy, W. A. Isaacs, and C. W. McCurdy. Collisional breakup in a quantum system of three charged particles. *Science*, 286:2474, December 24, 1999.

[22] J. Röder, J. Rasch, K. Jung, C. T. Whelan, H. Ehrhardt, R. Allan, and H. Walters. Coulomb three-body effects in low-energy impact ionization of H(1*s*). *Phys. Rev. A*, 53:225, 1996.

[23] M. R. H. Rudge and M. J. Seaton. Ionization of atomic hydrgoen by electron impact. *Proc. Roy. Soc. Ser. A*, 283:262, 1965.

[24] B. Simon. The definition of molecular resonance curves by the method of exterior complex scaling. *Phys. Letts. A*, 71:211, 1979.

[25] C. T. Whelan, H. R. J. Walters, J. Hansen, and R. M. Dreizler. High energy electron impact ionization of h(1s) in coplanar asymmetric geomerty. *Austr. J. Phys.*, 44:39, 1991.