ELECTRON–MOLECULE COLLISION CALCULATIONS ON VECTOR AND MPP SYSTEMS

C. Winstead and V. McKoy

Noyes Laboratory of Chemical Physics
California Institute of Technology
Pasadena, California 91125

I. Introduction

The air we breathe consists mainly of electrically neutral molecules of nitrogen and oxygen, in which the positive charge of the molecules' nuclei is exactly balanced by the negative charge of a surrounding "cloud" of molecular electrons. Indeed, because the forces binding electrons to molecules are typically stronger than the forces holding atoms together to form molecules, it is only under rather special, nonequilibrium circumstances that free electrons and gas-phase molecules are found together. Those circumstances often involve contact between a normal molecular gas and an external source of ionizing energy: far-ultraviolet photons from the Sun in the upper atmosphere, for example, or lightning in the lower atmosphere. We have also learned to create such partially-ionized molecular gases, or low-temperature plasmas, under controlled conditions in laboratory and industrial settings, where their unique properties can be exploited to obtain chemical and physical results that would be difficult to achieve in any other way.

The most important technological application of low-temperature plasmas today is in the fabrication of microelectronics. Plasma processing is used at a variety of stages in semiconductor manufacturing; deposition, cleaning, mask stripping, and etching may all involve plasmas. In plasma etching, for example, a radiofrequency field creates a plasma in a gas mixture containing a fluorocarbon gas, such as c-C_4F_8 (octafluorocyclobutane), or a hydrofluorocarbon, such as C_2HF_5 (pentafluoroethane). Collisions between the energetic free electrons and neutral molecules can result in further ionization, producing additional free electrons to sustain the plasma, but they can also break molecules apart to form reactive fragments – ions, atoms, and radicals. A key feature is that these reactive fragments are produced at low gas temperatures, protecting the semiconductor surface from thermal damage: only the electrons, which are extremely light and so can do little damage, acquire high energies. When the molecular fragments collide with the surface, they may etch the substrate by reacting with it to form gaseous products that can be pumped away; under different conditions, they may also react with each other to form a polymeric layer that protects the substrate from etching. Indeed, with careful control of plasma conditions (gas mixture composition, flow rates, applied electric fields, etc.), both processes can be promoted simultaneously. For example, deep, narrow trenches are formed when active etching occurs at the trench bottom while the walls are protected. Achieving such subtle effects, which are essential to the manufacture of modern, high-density microelectronics with their small feature sizes and multiple layers of interconnects, clearly can benefit from the greatest possible understanding of the physical and chemical processes occurring in the plasma and at the surface. In particular, to pursue numerical modeling of a plasma reactor, we would need detailed knowledge of the initial electron-molecule collision step both in order to know which reactive fragments are produced and in what numbers and in order to follow energy transfer between electrons and heavy particles.

Perhaps surprisingly, electron collision information for polyatomic molecules is very limited; indeed, for many of the fluoro- and hydrofluorocarbon molecules used today in plasma etching applications, experimental data, if available at all, are generally lacking for the most important collision processes, including elastic scattering (which changes only the direction of the electron's

motion) and electronic excitation, which is a major mechanism for dissociation of the molecule to neutral fragments. The shortage of experimental information arises in part from the hazardous properties of some of the gases, in part from the intrinsic difficulty of the measurements (particularly measurements of electronic excitation), and in part from the limited number of research groups worldwide that are engaged in such work. Under the circumstances, it is natural to consider a role for computational approaches. After all, Schrödinger's equation gives us, in principle, the means to solve the quantum-mechanical electron-molecule collision problem to any desired accuracy for any molecule we might be interested in. In practice, of course, limitations arise from the sheer scale of the required calculations, and devising methods that can make efficient use of powerful computers becomes crucial. The balance of this paper will describe the computational challenges that arise in such a computational program and how we address those challenges by exploiting both traditional PVP systems and MPP systems.

II. Computational

To understand the computational issues that arise in our studies of electron-molecule collisions, it is necessary to survey briefly the physics and chemistry of the collision process. The most important point is that the collisions of interest in low-temperature plasmas occur at low enough energies that they must be treated as fully quantum-mechanical; that is, the electron's de Broglie wavelength is fairly long compared to the size of the molecule. Moreover, because electrons are fundamentally indistinguishable, and because the electrons bound within the molecule are moving at speeds comparable to that of the projectile, a proper treatment must be based not on a single-particle wavefunction but rather on a many-particle wavefunction describing all $N_e + 1$ electrons (the N_e electrons of the molecule and the colliding electron) on an equal footing. The Schrödinger equation describing this problem is a second-order partial differential equation in $3(N_e+1)$ variables and cannot be solved by brute force for any but the smallest (one- or two-electron) systems. As in conventional bound-state quantum chemistry, therefore, we must look for efficient yet accurate methods of obtaining approximate solutions.

A. Formulation

One common approach to obtaining approximate solutions of physical problems when direct solution is too laborious is to invoke a variational principle for a quantity of interest, using that principle to search for an optimal solution within a "function space" of possible solutions. For example, in conventional computational quantum chemistry, where we are commonly interested in the ground (lowest-energy) state of the molecule, many standard techniques employ the Rayleigh-Ritz variational principle to search a linear function space for the linear combination that minimizes the energy of the system. In the electron-molecule collision problem, however, we have an unbound electron, and the total energy is fixed by the initial conditions (i.e., that long before the collision, we have a free electron of a certain energy and an isolated molecule in a certain energy state). The Rayleigh-Ritz principle therefore cannot be applied. Instead, a computational method for electron-molecule collisions can be based on a variational principle for the scattering amplitude, the quantity whose square modulus gives the probability for a given collision outcome to occur. The scattering amplitude is commonly written $f(\vec{k}_i, \vec{k}_f)$, where the vectors $\vec{k}_i$ and $\vec{k}_f$ are proportional to the initial and final momenta of the electron; thus, any energy lost to the molecule in the collision results in a decrease of the magnitude of $\vec{k}_f$ relative to $\vec{k}_i$. In our work, we use an extension [1] of the Schwinger variational principle, which uses the identity $f = f + f - f$ to represent the scattering

amplitude in a form,

$$f(\vec{k}_i, \vec{k}_f) = \int d\tau \Psi^{(-)*}(\vec{\tau}; \vec{k}_f) V \Phi_i(\vec{\tau}; \vec{k}_i) + \int d\tau \Phi_f^*(\vec{\tau}; \vec{k}_f) V \Psi^{(+)}(\vec{\tau}; \vec{k}_i) - \int d\tau \Psi^{(-)*}(\vec{\tau}; \vec{k}_f) A^{(+)} \Psi^{(+)}(\vec{\tau}; \vec{k}_i),$$

that is stationary with respect to first-order errors in the wavefunctions $\Psi^{(+)}$ and $\Psi^{(-)}$. Here $\vec{\tau}$ represents all $3(N_e + 1)$ electron coordinates and $\int d\tau$ integration over those coordinates; asterisks indicate complex conjugation; Φ_i and Φ_f are interaction-free wavefunctions for the initial and final states, formed from unperturbed states of the molecule and plane waves representing a free electron; V is the interaction potential between the molecule and the projectile electron; and $A^{(+)}$ is a complicated operator containing several terms, the most difficult of which involves a Green's function. The wavefunctions $\Psi^{(+)}$ and $\Psi^{(-)}$ are solutions to the scattering problem satisfying different boundary conditions.

It is not important to understand the above variational expression in detail in order to understand the computational issues; it is only necessary to note that we can extract a system of linear equations from it by assuming a linear trial form for $\Psi^{(+)}$ and $\Psi^{(-)}$:

$$\Psi^{(+)}(\vec{\tau}; \vec{k}_i) = \sum_m x_m(\vec{k}_i) \Xi_m(\vec{\tau}), \quad \Psi^{(-)}(\vec{\tau}; \vec{k}_f) = \sum_m y_m(\vec{k}_f) \Xi_m(\vec{\tau}),$$

where the Ξ_m are known functions and the x_m and y_m unknown coefficients Introducing these assumed forms into the variational expression and setting the variation with x_m and y_m to zero leads to a linear system of the form $\mathbf{A}\mathbf{x} = \mathbf{b}$, where the elements of the matrix $\mathbf{A}$ are given by

$$\mathbf{A}_{mn} = \int d\tau \Xi_m^*(\vec{\tau}) A^{(+)} \Xi_n(\vec{\tau}),$$

the elements of $\mathbf{b}$ by

$$\mathbf{b}_{m,\vec{k}_i} = \int d\tau \Xi_m^*(\vec{\tau}) V \Phi_i(\vec{k}_i),$$

and the elements of $\mathbf{x}$ are the unknown coefficients $x_m(\vec{k}_i)$. The approximation to $f(\vec{k}_i, \vec{k}_f)$ may then obtained as $\mathbf{c}^\dagger \mathbf{x}$, where

$$\mathbf{c}_{m,\vec{k}_f} = \int d\tau \Xi_m(\vec{\tau})^* V \Phi_f(\vec{k}_f)$$

and the dagger indicates complex-conjugate-transpose. Quadratures over the directions of $\vec{k}_i$ and $\vec{k}_f$ are of course used to reduce all matrices to finite dimensions.

B. Implementation

Having derived a linear system to solve, the remaining task is to evaluate the elements of $\mathbf{A}$, $\mathbf{b}$, and $\mathbf{c}$. Evaluating these elements is, in fact, the principal computational challenge. When we write the $(N_e + 1)$-electron functions Ξ_m and $\Phi_{i,f}$ as (properly antisymmetrized and spin-adapted) products of one-electron functions, we obtain vast numbers of integrals involving those one-electron functions and the operators occurring in the variational expression. The greatest difficulty, as we noted above, comes from the Green's-function term included in the $A^{(+)}$ operator. By choosing a convenient (spectral) representation of the Green's function and introducing a quadrature, we can reduce the problem to evaluating one- and two-electron integrals arising, respectively, from

electron-nucleus and electron-electron Coulomb interactions. The more numerous two-electron integrals have the form

$$\int d^3r_1 \int d^3r_2 \alpha(\vec{r}_1)\beta(\vec{r}_1)|\vec{r}_1 - \vec{r}_2|^{-1}\gamma(\vec{r}_2)\exp(i\vec{k}' \cdot \vec{r}_2),$$

where α, β, and γ are the one-electron basis functions used in the represntation of Ξ_m and of $\Phi_{i,f}$ and where the vector $\vec{k}'$ occurring in the plane wave is the quadrature variable in the spectral representation of the Green's function. Because we are using a Schwinger-type functional indifferent to the long-range behavior of the trial functions, we are free to choose square-integrable one-electron functions α, β, γ, and we therefore choose them to be Cartesian Gaussians of the type commonly used in conventional, bound-state quantum chemistry. With this choice, all of the integrals required to form $\mathbf{A}$, $\mathbf{b}$, and $\mathbf{c}$ may be evaluated analytically to any desired accuracy.

It is clear from the expression given above that the number of two-electron integrals will grow as the cube of the number of Gaussians, and therefore roughly as N_e^3, the cube the number of electrons in the molecule. The number of such integrals is also proportional to the number N_k of quadrature points in $\vec{k}'$ needed to evaluate the Green's function, which may be in the tens of thousands. Even for moderate-sized fluorocarbons, 10^{12} such integrals might be required. Each integral evaluation requires on the order of 100 floating-point operations, carried out within an intricate subprogram. Although pre-estimation and symmetry can be used to reduce the number of integrals that must actually be evaluated, clearly, a significant computational task exists for large molecules.

Moreover, evaluating the "raw" integrals over Gaussians and plane waves is only a first step; those integrals must then be transformed to the desired matrix elements of $\mathbf{A}$, $\mathbf{b}$, and $\mathbf{c}$. Details of this transformation are described elsewhere [2]; the important point here is that the work involved scales as $N_e^4 N_k$. Although the transformation step can be formulated as matrix-matrix linear algebra and therefore has both a much smaller prefactor and much higher efficiency than the integral evaluation step, the extra factor of N_e in the scaling relation makes the transformation step the computational bottleneck for sufficiently large molecules.

The computational method we have described above was originally implemented as a uniprocessor program for conventional computers and for vector supercomputers such as the Cray X-MP and Y-MP. The rapid scaling with problem size, however, severely limits what is feasible with a uniprocessor approach. Moreover, although certain computationally intensive portions of the program—in particular, the integral transformation—perform well on vector processors, others, including the generation of the raw integrals, are poorly vectorizable. These considerations point away from PVPs and toward MPPs. On the other hand, there are large and complex portions of our code that make only minor contributions to the overall operation count, and others that are primarily bound by disk usage rather than compute time. In both cases, devoting scarce human resources to parallelizing those sections of code would be a poor investment, if not counterproductive. Accordingly, our current implementation of the SMC program is partitioned into uniprocessor and massively parallel components that execute independently. The partition point is chosen so that the intermediate datasets that must be transferred from the uniprocessor to the parallel portion is of moderate size.

In the initial, uniprocessor phase, input data describing the molecule that is to be studied are read in, and all computations that are independent of the collision energy, and therefore of the magnitudes of $\vec{k}_i$ and $\vec{k}_f$, are carried out. This phase includes computation and transformation of two-electron integrals involving four Gaussians (rather than, as above, three Gaussians and plane wave), which are needed in the construction of matrix elements of the the $A^{(+)}$ operator. This step, which also occurs in ordinary, bound-state quantum chemistry programs, has N_e^5 scaling, but it is a

minor task compared to the $N_e^4 N_k$ transformation of the three-Gaussian, one-plane-wave integrals. Indeed, although it is important to have a fast processor available to carry out the transformation, it is perhaps more important to have a large and efficient file system available on which to store the raw and transformed integrals, whose number scales as N_e^4. Because the integral transformation is well suited to vector processors, which also typically offer large and fast scratch file systems, we have found traditional PVPs such as the Cray SV1 at JPL to be logical choices for carrying out this phase of the calculation. In recent years, we have also made increasing use of x86-based Linux systems in this phase as the capabilities of such systems have grown, though the 2 Gbyte file-size limit common to 32-bit Unix file systems was, until recently, a limitation.

The parallel phase of the calculation reads data files containing the problem description and the intermediate results (contributions to the $A^{(+)}$ matrix elements) produced by the initial, uniprocessor phase, and it then carries out the work needed to complete the construction of the elements of the **A**, **b**, and **c** matrices. For this phase, we adopted a loosely synchronous, single-program, multiple-data programming model. That is, every processor in a multiprocessor system executes a copy of the same program, with each processor performing a different subset of the total work and with synchronization across processors imposed only at intervals, usually when data must be exchange among them. Parallelism is explicit: data decomposition, synchronization, and communication are all handled with application code or explicit calls to library routines. Although this programming model was adopted long ago because it was suited to the hardware and software then available to us, it has proven remarkably resilient and adaptable. The resulting code has been ported with relative ease to a variety of MPP systems, including the nCUBE-2, Intel Delta and Paragon, IBM SP2, Cray T3D and T3E, and SGI Origin 2000; it has also adapted easily to Linux-based clusters. On most of these systems, scalability has been excellent, with large problems running efficiently on hundreds of processors. Thus, although explicit message passing and data decomposition require a higher initial investment, we believe that investment has paid off in terms of portability, efficiency, and, hence, program longevity.

The partitioning of work in the parallel phase of the calculation aims to achieve good load balance and minimal communication and synchronization overhead. To those ends, the three-Gaussian, one-plane-wave two-electron integrals are organized into distributed, two-dimensional arrays indexed in one dimension by an (α, β) pair index and in the other dimension by the direction of $\vec{k}'$, with the γ and $|\vec{k}'|$ indices being processed sequentially in order to limit memory requirements. The global integral arrays are distributed as equally as possible over a logical two-dimensional processor mesh. The time to evaluate a given integral can vary strongly depending on the Gaussian and plane-wave parameters; however, a simple but effective statistical load-balancing strategy is to distribute the array indices cyclically. Because the number of integrals per processor is large and there is no interprocessor synchronization during this step, variations in integral evaluation time tend to average out.

When a batch of integrals is completed, a "transformation matrix" is constructed that incorporates the rules for computing contributions to the final $(N+1)$-particle matrix elements from the raw integrals. The distribution of the transformation matrix is complementary to that of the integral array. This step is, again, perfectly parallel. When both integrals and transformation matrix are in hand, a distributed matrix multiplication routine is invoked, and it is only there that interprocessor communication is needed. Because the global arrays are large, the computation to communication ratio is favorable, and parallel overhead is low. Moreover, on most systems, an optimized library routine can be used to carry out the subblock multiplications, giving high absolute performance. We have further found that having a regular communication pattern well localized within the program increases the ease of porting and optimization.

So far we have focused on those portions of our work that employ our special-purpose electron–

molecule collision code. However, there is also a "zeroth" phase, in which we employ standard, third-party electronic-structure programs to characterize the molecule of interest and to obtain descriptions of its low-lying electronic states which will then be used as input to collision calculation. These programs, although highly developed and reliable as uniprocessor applications, often scale poorly, if they support parallelism at all, and many of the methods they implement are disk-intensive. PVP systems such as the Cray SV1 have thus remained an important resource for preparing an electron–molecule collision calculation, as well as for carrying out the sequential phase of that calculation.

C. Performance

Because the bulk of our computation executes in parallel, the scalability and absolute performance of that phase of the calculation is most critical. In describing our application's performance, we will therefore concentrate on the parallel portion. Nonetheless, because relative PVP and workstation performance may be of interest, we also examine performance for the initial, uniprocessor phase of the calculation.

As a sample problem for the uniprocessor phase, consider, from work currently in progress, a calculation of elastic and inelastic scattering by sulfur hexafluoride, SF_6, a fairly large molecule by electron-collision standards though by no means the largest we have studied. This calculation employed 211 (contracted) Gaussian functions in the one-electron basis set and 1229 terms in the $(N+1)$-particle variational basis set. Because of disk requirements, this job is too large to run to completion on any of our in-house Linux machines. On the SV1, it executes 1.91×10^{12} floating-point operations in 2.63×10^4 CPU seconds, for an overall 72.7 MFLOP. A more detailed examination shows that 88% of the operations occur in the 4-index transformation of the 2-electron integrals, and that the throughput in that routine, inclusive of disk overhead, is 175 MFLOP. Most of the elapsed time, therefore, is consumed in less efficient stages of the program, in particular the initial evaluation of the 2-electron integrals and the disk-intensive and logically complicated process of assembling contributions to the $(N+1)$-particle matrix elements from the transformed 2-electron integrals.

Examination of other uniprocessor runs confirms that the above performance is typical. Obviously it is unspectacular; however, we emphasize that absolute MFLOPs are *not* the most critical issue. What is important is that the uniprocessor phase perform reasonably, eliminating the need to parallelize a lengthy and complicated program that is, in the end, a minor component of the overall computation. With the aid of systems such as the SV1 that are able to supply not only good CPU performance but large memory and disk to a single process, we have so far been able to achieve this.

Absolute computational speed *is* important in the parallel phase of our calculation, where the operation count is high and, as we have seen above, increases rapidly with problem size. High performance in the parallel phase is virtually guaranteed, however, because most of the work in large problems occurs within the distributed matrix multiplication. By using optimized library routines for the matrix subblock multiplication, we typically obtain large fractions of peak performance in this step.

A second important measure is of parallel performance is scalability, i.e., the degree to which program performance remains proportional to the number of processors employed. Fig. 1 shows a plot of performance for a fixed-size test problem (taken from a study of C_2F_6, hexafluoroethane) on a variety of MPP systems as well as on Linux workstations and clusters. The slope of the diagonal line in Fig. 1 represents perfect speedup; as may be seen, scalability on the T3E is excellent up to at least 256 processors, although absolute performance is lower than on more recent systems with higher clock speeds. The scaling on the T3E reflects its low-latency, high-bandwidth interprocessor communication, which is especially important in this moderate-sized test case because, on large

numbers of processors, the computation-to-communication is becoming low. As mentioned earlier, the critical communication steps are localized in the distributed matrix multiplication. On the T3E, we obtained enhanced performance by using shared-memory library calls to perform just those communication steps, while the remainder of the communication throughout the program was handled by MPI. The flat interconnect of the T3E is also responsible for the speedup remaining linear out to high numbers of processors; on some NUMA (non-uniform memory access) and cluster systems, we see a fall-off when communication begins to flow over a slower layer of the interconnect.

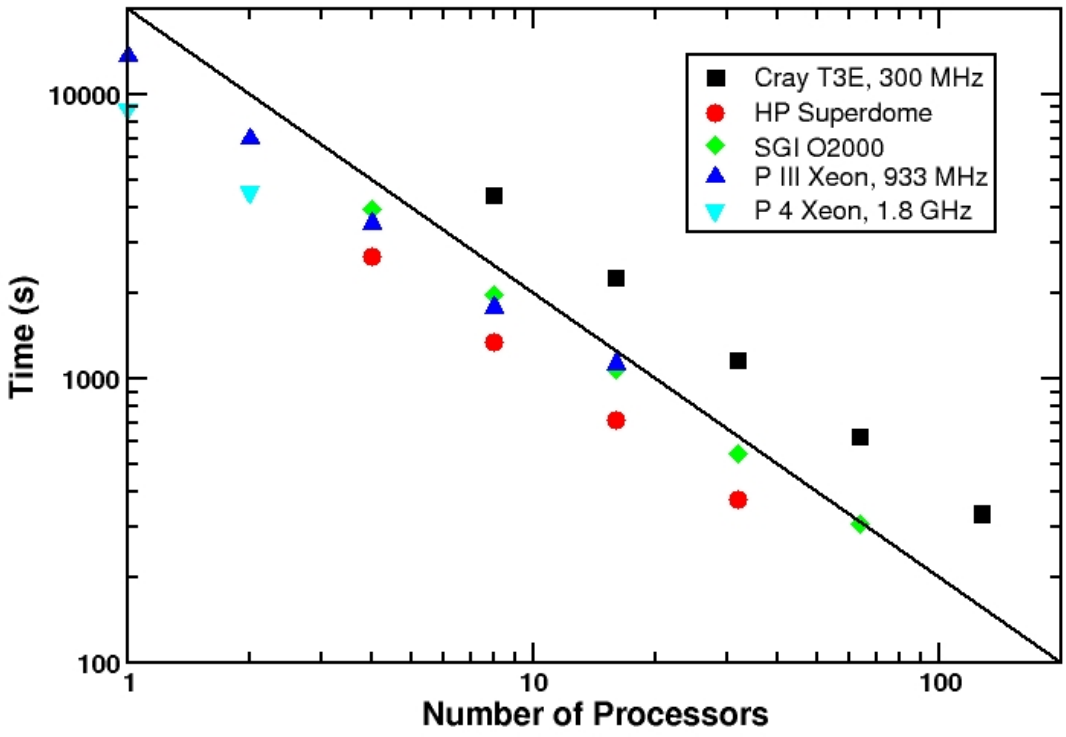

Fig. 1 Scaling on various MPP systems.

III. Illustrative Results

To illustrate the use of the electron-scattering program we have been describing, we look in this section at some recent results [3,4] for the fluorocarbon C_2F_4, tetrafluoroethene, which has attracted recent interest as a plasma etchant [5]. For this modest-sized fluorocarbon, it proved possible to do an unusually complete study. We computed not only elastic electron collision data but also cross sections (probabilities) for excitation of C_2F_4 by electron impact to ten electronic states. Our work, moreover, was done in collaboration with experimental scientists, some of whom measured electron-impact ionization cross sections for C_2F_4, while others collected data on the propagation of low-energy electron swarms in C_2F_4 under the influence of an electric field, data against which the calculated and measured cross sections could be tested. A third important component of the collaboration, therefore, was an expert in the construction and refinement of

cross section *sets*, who would evaluate the cross sections against the swarm data, supply estimates of unknown cross sections (such as those for vibrational excitation), and, if necessary, adjust the cross sections to produce a self-consistent set that was also consistent with the swarm data.

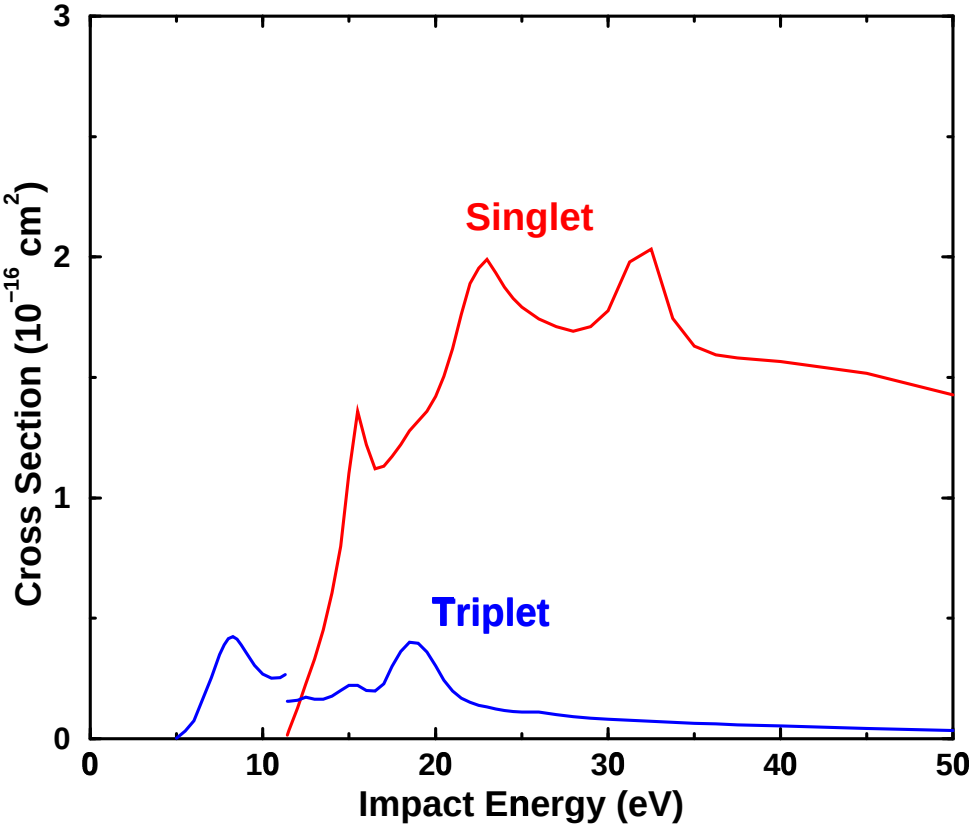

Fig. 2 Computed cross sections for two important electron collision processes in C_2F_4, excitation of the $1\ ^3B_{1u}$ ("Triplet") and $1\ ^1B_{1u}$ ("Singlet") states. See Ref. 3 for details.

Fig. 2 shows some of the cross sections we obtained from our calculations [3]. The cross sections are shown as a function of the kinetic energy of the colliding electron (measured in electron-volts, a convenient unit); dependence on the collision angles has been removed by averaging over all angles of incidence (appropriate because gases contain vast numbers of randomly oriented molecules) and integrating over all angles of departure. Although one can often gain much physical and chemical insight from detailed analysis of the individual cross sections, for present purposes it is more interesting to see how useful those cross sections are in describing the actual behavior of electrons in C_2F_4. Thus in Fig. 3 we compare properties of an electron swarm in C_2F_4 that were measured by our collaborators with the same properties computed directly from the cross sections [4]. As may be seen, the agreement is generally excellent, giving confidence that the cross section set is correct. Moreover, it was not found necessary to adjust the calculated cross sections in order to achieve agreement with the swarm measurements, giving us some confidence that they provide a generally correct description of the interaction between low-energy electrons and C_2F_4. Validated

cross section sets of this type, which can form a sound basis for plasma simulations, are extremely rare; prior to our work, no such set was available for C_2F_4.

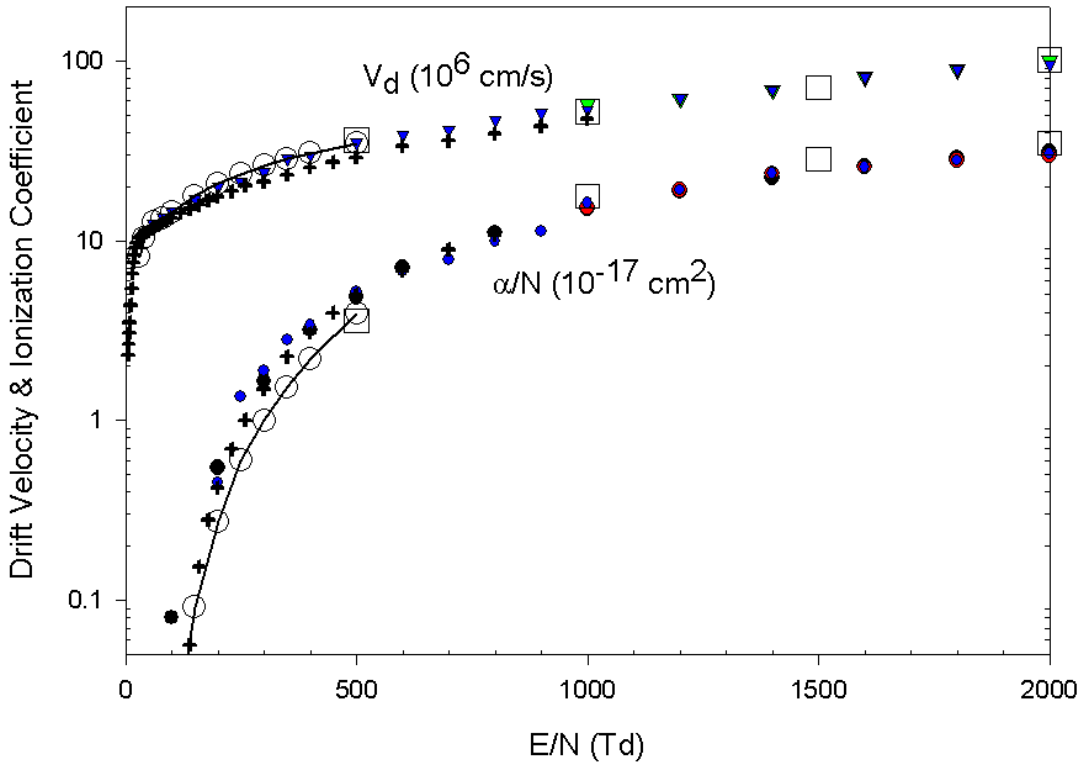

Fig. 3 Comparison of predicted (open symbols) and measured (filled symbols) swarm parameters for C_2F_4, from Ref. 4. The predictions are based in large part on calculated cross sections.

IV. Conclusions

Although the great bulk of the computer cycles required in our electron–molecule collision studies are now provided by MPPs and clusters, a significant fraction of our code (by line count, more than half) continues to rely on uniprocessing. As we have discussed, this is in part because the return on (human) investment is not sufficient to justify parallelizing everything, but it also reflects, in part, some of the traditional strengths of large uniprocessor systems and, conversely, some of the traditional weaknesses of highly distributed systems. For example, portions of the calculation whose computational requirements scale moderately with problem size, but whose disk usage grows rapidly, are well suited to PVP systems that offer both good single-CPU performance and a large, fast file system, whereas they are poorly suited to many MPP or cluster systems. Whether, in the future, such calculations will be carried out on vector supercomputers or on "commodity" workstations based on open-source software and inexpensive hardware is, to us, an open question, particularly as 64-bit workstations begin to compete with a new generation of vector systems.

On the other hand, we have seen that for key portions of our overall calculation, the use of highly parallel computers, whether tightly integrated MPPs or large clusters of commodity workstations, is critical. These are tasks where the amount of computation scales rapidly with one

or more problem dimensions, so much so that these tasks are by far the dominant ones in terms of CPU time, and where the work involved is highly amenable to a data-parallel approach in which different processors simultaneously tackle different portions of the total problem. Because it is common for scientific computing applications to have such rapidly-scaling steps and therefore, for large enough problems, to be suited to highly parallel architectures, a permanent role for large-scale parallelism seems, to us, to be assured.

Indeed, after more than fifteen years of applying vector and parallel computers to the electron–molecule collision problem, during which exponential progress in accordance with Moore's Law has produced astonishing improvement in single-processor performance, our research remains resource-constrained: The definition of a large problem has certainly changed, but not the fact that there are interesting problems too large to tackle on the machines available to us today. We thus look forward with interest and anticipation to the advent of systems such as the Cray X1, which promise greatly increased performance as well as a melding of PVP and MPP features.

V. Acknowledgments

We acknowledge use of the resources of the JPL Supercomputing Project and thank the Project staff for their continued assistance. The work described here was supported in part by the Department of Energy, Office of Basic Energy Sciences, and by Sematech, Inc. We thank our collaborators who have contributed to various aspects of the work described here, including Márcio Bettega, W. Lowell Morgan, and Márcio Varella.

VI. References

[1] K. Takatsuka and V. McKoy, Phys. Rev. A **24**, 2473 (1981); **30**, 1734 (1984).

[2] C. Winstead and V. McKoy, Adv. At., Mol., Opt. Phys. **36**, 183 (1996); C. Winstead, C.-H. Lee, and V. McKoy, in *Industrial Strength Parallel Computing: Programming Massively Parallel Processing Systems*, A. Koniges, editor (Morgan-Kaufmann, San Francisco, 2000), p. 247.

[3] C. Winstead and V. McKoy, J. Chem. Phys. **116**, 1380 (2002).

[4] K. Yoshida, S. Goto, H. Tagashira, C. Winstead, V. McKoy, and W. L. Morgan, J. Appl. Phys. **91**, 2637 (2002).

[5] S. Samukawa, T. Mukai, and K. Tsuda, J. Vac. Sci. Technol. A **17**, 2551 (1999); S. Samukawa and T. Mukai, Thin Sol. Films **374**, 235 (2000).