Theor Chim Acta (1987) 72: 237-251

Theoretica Chimica Acta

Storage management strategies in large-scale quantum dynamics calculations*

David W. Schwenke[1], Kenneth Haug[1], Donald G. Truhlar[1], Roland H. Schweitzer[2], John Z. H. Zhang[3]***, Yan Sun[3] and Donald J. Kouri[3]**

[1] Department of Chemistry and Supercomputer Institute, University of Minnesota, Minneapolis, MN 55455, USA
[2] ETA Systems, Inc., 1450 Energy Park Drive, St. Paul, MN 55108, USA and Minnesota Supercomputer Center, 1200 Washington Avenue South, Minneapolis, MN 55415, USA
[3] Departments of Chemistry and Physics, University of Houston, Houston, TX 77004, USA

(Received June 15; revised and accepted July 15, 1987)

We discuss the computational strategies related to memory and disk storage and to data movement between memory and disk in large-scale quantum dynamics calculations. The discussion includes practical implementations of various strategies for handling large data sets on various supercomputer architectures.

Key words: Quantum mechanical dynamics calculations — Supercomputers — Storage management — Computer memory

1. Introduction

We have recently initiated two research programs involving large-scale dynamics calculations based on time-independent quantum mechanics. The first program [1-8] is directed to inelastic collisions and has been carried out by converting the Schroedinger equation to coupled linear ordinary differential equations which are solved by an invariant-imbedding-type propagation method. (Propagation techniques similar to those employed in this study could also be applied to

* This paper was presented at the International Conference on 'The Impact of Supercomputers on Chemistry', held at the University of London, London, UK, 13-16 April 1987
** *Present address*: NASA Ames Research Center, Moffett Field, CA 94035, USA
*** *Present address*: Department of Chemistry, University of California, Berkeley, CA 94720, USA

reactive scattering [9], but we have not yet done so.) The second research program [10-13] is directed primarily to reactive collisions (although the techniques have also been applied to inelastic scattering [11] and should be quite competitive or even optimal for some inelastic scattering calculations). The second program involves converting the Schroedinger equation to coupled integral equations which are solved by basis-set expansion and linear algebraic techniques. The quantum mechanical theory for inelastic collisions [1, 3, 11] and reactive scattering [9, 11, 13] and the results of applications [2-8, 10-13] are discussed in detail elsewhere. In this contribution to the proceedings of the International Conference on the Impact of Supercomputers on Chemistry, we center our attention on one specific aspect of the calculations, namely the questions of memory and disk usage, or - more broadly stated - the problem of storage management. Results presented at the Conference are described elsewhere [1-8, 10-13].

Section 2 provides a brief review of our algorithms. Section 3 presents the storage management strategies we have used, and Sect. 4 summarizes our conclusions.

2. Algorithms

2.1. Propagation method

The Hamiltonian for vibrationally, rotationally inelastic collisions of two molecules may conveniently be written as

$$H = -\frac{\hbar^2}{2\mu}\frac{1}{R^2}\frac{\partial}{\partial R}\left(R^2\frac{\partial}{\partial R}\right) + H^0 + \mathcal{V}, \tag{1}$$

where μ and R are the reduced mass and radial coordinate for relative translational motion, H^0 consists of centrifugal terms and the internal Hamiltonians of the two colliders, and $\mathcal{V}$ is their interaction potential. We restrict the system to a single Born-Oppenheimer potential surface, and we expand the solution of the Schroedinger equation as

$$\psi^{n_0} = \frac{1}{R}\sum_{n=1}^{N} f_{nn_0}(R, E)X_n(\boldsymbol{x}) \tag{2}$$

where n_0 denotes the initial state, f_{nn_0} is an unknown radial translational wave function, E is the total energy, $\{X_n\}$ is an orthogonal basis set, and $\boldsymbol{x}$ denotes the collection of angular translational coordinates and internal coordinates of the colliders. The different terms in Eq. (2) are called channels. Substituting Eq. (2) into the Schroedinger equation

$$H\psi^{n_0} = E\psi^{n_0}, \tag{3}$$

multiplying by $-2\mu R^2 X_n^*/\hbar^2$, and integrating over $\boldsymbol{x}$ yields

$$\frac{d^2}{dR^2}\boldsymbol{f}(R, E) = \boldsymbol{D}(R, E)\boldsymbol{f}(R, E), \tag{4}$$

where the elements of $\boldsymbol{D}(R, E)$ involve integrals over H^0, $\mathcal{V}$, and the basis functions. The development just given corresponds to the conventional close coupling equations, but equations of the same form, with smaller order N for a given total angular momentum and set of vibrational-rotational states, are obtained in various more approximate formulations involving effective potentials [14, 15] or angular momentum decoupling [16-18].

There are N linearly independent vector solutions of (3) that are regular at the origin, as required for the physical solutions. We obtain a full set of such regular solutions by simultaneously propagating information equivalent to N solutions from small R to large R, where we match to the n_0-dependent boundary conditions to obtain the physical solutions.

The propagation algorithm we use is the R matrix propagation method [19-21], which is an invariant-imbedding-type [22-25] algorithm. It involves dividing the R coordinate into a large number of sectors (typically 300-600) and recursively assembling the global R_4 matrix. The steps performed for sector i are to construct the D matrix at the sector midpoint, determine the eigenvectors $\boldsymbol{T}(i)$ and eigenvalues $\boldsymbol{\lambda}^2(i)$ of $\boldsymbol{D}$ (which is real and symmetric and of order N, the number of channels), build the sector overlap matrix $\tau(i-1, i)$ by

$$\boldsymbol{\tau}(i-1, i) = \boldsymbol{T}^T(i-1)\boldsymbol{T}(i) \tag{5}$$

and form its inverse, and propagate the global R_4 matrix by

$$\boldsymbol{R}_4(i) = \boldsymbol{r}_4(i) - \boldsymbol{r}_3(i)[\boldsymbol{R}_4(i-1) + \boldsymbol{r}_1(i)]^{-1}\boldsymbol{r}_2(i), \tag{6}$$

where the $\boldsymbol{r}_\alpha(i)$ matrices are functions of $\boldsymbol{\tau}(i-1, i)$ and diagonal sector propagators that depend on $\boldsymbol{\lambda}(i)$. The final quantities of physical interest can be easily constructed from $\boldsymbol{R}_4$ for the final sector.

For the energy-independent potential used in the present study, the total energy occurs in (4) only as a multiple of the unit matrix in $\boldsymbol{D}$, thus the eigenvectors $\boldsymbol{T}$ and the sector overlap matrices $\boldsymbol{\tau}(i-1, i)$ are independent of energy and can be re-used for calculations with the same potential and channel set but a different energy. Therefore for second and subsequent energies, only the steps associated with Eq. (6) have to be performed.

As we have coded the R matrix propagation algorithm, it involves 3 matrix multiplications, 2 linear equation solutions, and 1 matrix diagonalization per sector for the first energy, and 2 matrix multiplications and 1 linear equation solution per sector for subsequent energies. All of these operations are on full matrices of order N. Our strategies for solving Eq. (4) have been developed for N on the order of 10^3. For many problems of interest, even higher values of N will be required for convergence.

Further details may be found elsewhere [1, 3, 19, 20].

2.2. Algebraic method

The second approach to molecular collision dynamics that will be discussed in the present paper is to convert the Schroedinger equation to coupled Lippmann-Schwinger integral equations for the amplitude density and to solve these by expanding the amplitude density in a square integrable ($\mathscr{L}^2$) basis set [11]. This method is applicable to both reactive scattering, for which several arrangements of the particles are considered, and nonreactive inelastic collisions. As in Sect. 2.1, we restrict the system to a single Born-Oppenheimer potential surface. We will give the equations for reactive scattering and consider the single-arrangement inelastic problem as a special case. When the amplitude denisty is expanded in a basis and the coefficients of the basis functions are obtained by the method of moments [11, 26], the resulting algebraic Lippmann-Schwinger equation can be written in matrix form as

$$\boldsymbol{a} = \boldsymbol{b} + \boldsymbol{C}\boldsymbol{a}, \tag{7}$$

where the $\boldsymbol{b}$ vector and $\boldsymbol{C}$ matrix have elements given by multidimensional integrals over readily obtained functions, and the elements of the $\boldsymbol{a}$ vector are the basis set expansion coefficients to be solved for. In particular, for reactive scattering, we write the reactive amplitude density for each arrangement α of the particles (e.g. $\alpha = 1$ for A+BC, 2 for B+AC, and 3 for C+AB) as

$$\zeta_\alpha^{\alpha_0 n_0} = \sum_{n=\bar{n}_\alpha}^{\bar{\bar{n}}_\alpha} \sum_{m=1}^{M_\alpha} a_{\alpha n m}^{\alpha_0 n_0} \Phi_{\alpha n m}(R_\alpha, \boldsymbol{x}_\alpha), \tag{8}$$

where

$$\Phi_{\alpha n m} = \frac{1}{R_\alpha} \lambda_m^{\alpha n}(R_\alpha) X_n^\alpha(\boldsymbol{x}_\alpha), \tag{9}$$

α_0 and n_0 denote initial arrangement and channel, $\bar{n}_\alpha = (1, \bar{\bar{n}}_1 + 1, \dots)$, $\bar{\bar{n}}_\alpha = (N_1, N_1 + N_2, \dots)$, R_α and $\boldsymbol{x}_\alpha$ are radial translational and angular-translational-internal coordinates for arrangement α, $\{X_n^\alpha\}_{n=\bar{n}_\alpha}^{\bar{\bar{n}}_\alpha}$ is an arrangement-dependent generalization of the basis of Eq. (2), and $\{\lambda_m^{\alpha n}\}_{m=1}^{M_\alpha}$ is a nonorthogonal basis of $\mathscr{L}^2$ radial translational basis functions. Then we define β as a collection of the three indices

$$\beta = (\alpha, n, m). \tag{10}$$

We will consider the case in which all three arrangements α have the same number of basis functions so that N_α and M_α are independent of α, and we define the product

$$N = 3 N_\alpha M_\alpha, \tag{11}$$

which is the order of a vector with index β or a matrix with indices β and β'. With these definitions

$$b_{\beta'}^{\alpha n} = \int dR_\alpha \, e_{\beta'}^{\alpha n}(R_\alpha) \, {}^{(r)}f_n^\alpha(E, R_\alpha) \tag{12}$$

and

$$C_{\beta'\beta} = (1 - \delta_{\alpha'\alpha}) Z_{\beta'\beta} - \frac{2\mu}{\hbar^2} \int dR_\alpha \, e_{\beta'}^{\alpha n}(R_\alpha) \times \int dR'_\alpha \, {}^{(r)}f_n^\alpha(E, R_\alpha^<) \, {}^{(i)}f_n^\alpha(E, R_\alpha^>) \lambda_m^{\alpha n}(R'_\alpha), \tag{13}$$

where

$$R_\alpha^< = \min(R_\alpha, R'_\alpha) \tag{14}$$

$$R_\alpha^> = \max(R_\alpha, R'_\alpha). \tag{15}$$

In these equations the functions ${}^{(r)}f_n^\alpha$ and ${}^{(i)}f_n^\alpha$ are energy-dependent solutions to zero-order problems as obtained in earlier steps, and $e_\beta^{\alpha n}$ and $Z_{\beta'\beta}$ are an energy-independent coupling vector (i.e. a vector of integrals over the coupling potential) and an energy-independent overlap matrix, respectively, also obtained in earlier steps. The integrals in Eqs. (12) and (13) are evaluated by numerical quadrature, and the total order of this quadrature in arrangement α will be called N_α^{QR} and will be assumed for discussion purposes to be independent of α. We will assume in the discussion that follows that $\lambda_m^{\alpha n}$ is independent of n.

An alternative form of the b vector and C matrix may be obtained by inserting the identity in Eqs. (12) and (13) in terms of the translational basis set $\{\lambda_m^{\alpha n}\}_{m=1}^{M_\alpha}$, i.e.

$$1 = \sum_{m=1}^{M_\alpha} \sum_{m'=1}^{M_\alpha} |\lambda_m^{\alpha n}\rangle \Delta_{mm'}^{\alpha n} \langle \lambda_{m'}^{\alpha n}|, \tag{16}$$

where $\Delta_{mm'}^{\alpha n}$ is the inverse of the translational overlap matrix, i.e.

$$\mathbf{\Delta}^{\alpha n} = (\mathcal{O}^{\alpha n})^{-1} \tag{17}$$

and

$$\mathcal{O}_{m'm}^{\alpha n} = \int dR_\alpha\, \lambda_{m'}^{\alpha n}(R_\alpha) \lambda_m^{\alpha n}(R_\alpha). \tag{18}$$

This is exact for a complete basis, but we use finite M_α which yields the approximations:

$$b_{\beta'}^{\alpha n} = \sum_{m''=1}^{M_\alpha} E_{\beta'\beta''} \int dR_\alpha\, \lambda_{m''}^{\alpha n}(R_\alpha) {}^{(r)}f_n^\alpha(E, R_\alpha) \tag{19}$$

and

$$C_{\beta'\beta} = (1 - \delta_{\alpha'\alpha}) Z_{\beta'\beta} - \frac{2\mu}{\hbar^2} \sum_{m''=1}^{M_\alpha} E_{\beta'\beta''} \int dR_\alpha\, \lambda_{m''}^{\alpha n}(R_\alpha)$$
$$\times \int dR'_\alpha {}^{(r)}f_n^\alpha(E, R_\alpha^<) {}^{(i)}f_n^\alpha(E, R_\alpha^>) \lambda_m^{\alpha n}(R'_\alpha), \tag{20}$$

where we have defined $E_{\beta'\beta}$ as a coupling matrix with elements

$$E_{\beta'\beta} = \int dR_\alpha\, e_{\beta'}^{\alpha n}(R_\alpha) \sum_{m''=1}^{M_\alpha} \Delta_{m''m}^{\alpha n} \lambda_{m''}^{\alpha n}(R_\alpha). \tag{21}$$

Note that for computational purposes it may be useful to rewrite (21) as

$$E_{\beta'\beta} = \sum_{m''}^{M_\alpha} \Delta_{m''m}^{\alpha n} \hat{C}_{\beta'\beta''}, \tag{22}$$

where we have first defined the integral

$$\hat{C}_{\beta'\beta} = \int dR_\alpha\, e_{\beta'}^{\alpha n}(R_\alpha)\lambda_m^{\alpha n}(R_\alpha). \tag{23}$$

Further details may be found elsewhere [11]. The method just discussed is the method of moments for the amplitude density, but essentially all the discussion will apply equally well to two closely related methods - the Schwinger variational method and Newton's variational principle for the amplitude density [26, 27] with or without angular momentum decoupling approximations.

3. Storage management strategies

One condition which must be satisfied before a calculation can be carried out is that sufficient memory be available for the algorithm used. In the past, this was not usually a problem for scattering calculations. This was because, even though the available physical memory was small, the CPU power was not usually great enough to tackle problems exhausting this memory. The availability of class VI supercomputers, however, brought a new facet to the problem. These machines (the Cray-1 and X-MP machines and the Cyber 205) have the CPU power to treat much larger problems than before. At the same time, the amount of memory is not increased by an equivalent factor. For example, in 1980 the Chemistry Department at the University of Minnesota obtained a Digital Equipment Corporation VAX 11/780 equipped with scalar floating point accelerator and 4 metabytes (equivalent to half a megaword of 64-bit word memory) of physical memory. In comparison, the University Computer Center had a Cray Research Inc. Cray-1b with 1 megaword of physical memory (a word is 64 bits on the Cray-1b). Our estimates of CPU times for HF+HF scattering calculations show the Cray-1b to be up to about a factor of 400 times faster than the VAX, yet it has only twice the physical memory. Although 8-million word Cyber 205's appeared in 1985, and the introduction of the class-VII Cray-2 machine, with 268 million 64-bit words, in 1986, ushered in a new era of large-memory machines, most supercomputing is still done with small memory machines, e.g. the Cray X-MP/48 has only two million words (equivalent to 16 megabytes) per processor. (An IBM Personal System/2 workstation may also be equipped with 16 megabytes per processor, and a Macintosh II may be equipped with 8.)

When physical memory is insufficient for the problem at hand, we must use mass storage, usually magnetic disk or tape. These media often, but not always, have enough storage to handle the problems for which one can afford the CPU time; however, they have the drawback of being much slower than physical memory. The Solid-state Storage Device (SSD) "secondary memory", available only on the Cray X-MP, is very large and much faster than conventional disk, but it is still slower than primary memory and has a large startup time before the fast aggregate transfer rate can be achieved. Thus it is possible for calculations using mass storage to spend more time waiting for data transfer to and from mass storage than time executing in the CPU. In quantal scattering calculations, most of the CPU time is consumed in performing matrix operations which, for large enough N, where N is the order of the matrix, scale as N^3. If the matrices involved are on mass storage, the time to retrieve them will scale with the order, as N^2, and for small or intermediate N, this time can dominate compared to the matrix operations. However, as N increases, eventually the N^3 term will dominate, and more time will be spent in the CPU than waiting for data transfer. This, of course, assumes that the data required for the matrix operation will fit into available physical memory. Otherwise an "out-of-core" algorithm must be employed, and this would bring a new dimension into the problem. Furthermore although some operations, such as matrix multiply and linear equations solution, can be partitioned efficiently so that only a partition of the matrix needs to be in memory at one time; others, such as eigenanalysis, are not as readily partitioned. Thus for larger problems, even if memory overflow is accommodated on mass storage, it is still important to have an efficient strategy for storing matrix elements in memory.

For our calculations on small-memory ($\leq$8 million 64-bit words) computers, we implemented the memory overflow to mass storage in two ways. On the Cray-1b, we explicitly move data via FORTRAN READ/WRITE statements (the Cray routines BUFFERIN and BUFFEROUT can also be used, and in fact are probably preferable, but they lack portability). Because of the 1-megaword memory limitation, our code for this machine was written to hold only two matrices in memory at any one time. This is sufficient for the matrix operations involved in the R-matrix propagation algorithm. In particular the eigenanalysis, which is carried out using the EISPACK routine RS for real symmetric matrices, and the real linear equation solution both require only 1 matrix in memory, and matrix multiply requires 2 matrices. The matrix multiply is performed as a sequence of matrix-times-vector-gives-vector steps, with the result matrix overwriting the second matrix. On the 1 megaword Cray-1b, this code could handle up to about $N = 640$ (640 coupled channels).

Our computational strategy on the Cyber 205 took advantage of the virtual memory aspect of the operating system. Here, as far as coding is concerned, the memory of the machine appears to be much larger than the physical memory available. At run time, the operating system automatically transfers data to and from memory as it is needed. There are several trade-offs between using explicit FORTRAN data transfer and using the virtual memory. On the one hand, the virtual memory is convenient, it automatically performs less data transfer if more memory is available or if a smaller problem is treated, and it uses efficient operating system input/output procedures. On the other hand, if explicit and intelligent data transfer statements can be used to minimize the total amount of data movement or organize it efficiently, they may decrease the amount of paging, make the code more transportable, and allow for sophisticated options like memory-resident files (codes with explicit data transfer are also readily adapted to utilize the SSD).

An important consideration in running the R matrix propagation code is that the amount of memory required in the course of a calculation can fluctuate greatly. Consider the various steps outlined in Sect. 2.1. The first is the construction of the D matrix, and the most time consuming part of this is computing matrix elements V_{ij} of the interaction potential. In general we can write

$$V_{nn'}(R) = \sum_{m=1}^{M} B^{m}_{\beta_n \beta_{n'}} C^{m}_{\gamma_n \gamma_{n'}}(R), \tag{24}$$

where the $B^m_{\beta\beta'}$ are independent of R, the $C^m_{\gamma\gamma'}$ depend on R, and β and γ denote subsets of the channel quantum numbers. For example in HF+HF collisions, $B^m_{\beta\beta'}$ is a 6-dimensional angular integral which depends on rotational quantum numbers and the angular momentum coupling and is expressible in terms of $3-j$, $6-j$, and $9-j$ symbols, and $C^m_{\gamma\gamma'}$ is a 2-dimensional vibrational integral of component m of the interaction potential function. We have performed calculations which require M up to 825. Even though the B^m matrix is sparse (about 5% nonzero for this case) and we store only the nonzero elements, a considerable amount of memory or disk is required to save these elements, in particular about $40N^2$ words. The remaining steps (diagonalization of $\boldsymbol{D}$, building the sector overlaps, and propagation) in the R-matrix propagation algorithm for a single energy run have much more modest memory requirements, and in particular our code requires $7N^2$ words (we neglect all requirements proportional to only N in the present discussion). If the operating system assigns priorities to jobs by their memory requirements, as it does in our batch environment, it can be advantageous (e.g. to minimize wall clock time) to advise the system of the maximum memory requirements of the various steps of the calculation.

In our calculations we found that the virtual memory of the Cyber 205 is indeed useful for extending the size of the system which can be treated. Virtual memory on the Cyber 205 is divided into units called pages. There are two different page sizes, small pages which are 16 blocks of 512 64-bit words per block, and large pages which are 128 blocks. The size of the small page is selected by the site and may be 1, 4, or 16 blocks. We grouped our data on large pages and the Cyber 205 virtual memory manager, the pager, automatically moved pages in and out of memory as needed. However, if we did not advise the system of the changing memory requirements of different steps of our job, we found that the turn-around time for a large job (a job requiring ~115 large pages) was much longer than the CPU time required. This occurred because a large job is run at a low priority so our jobs were forced to wait for large amounts of memory during busy production periods.

Jobs are scheduled for the CPU by priority, provided that the job with the highest priority still waiting to be scheduled will fit into the available memory. For a typical one of our jobs ($N=824$), the memory requirements would fluctuate, starting at about a small fraction of the machine memory (~23 large pages), climbing rapidly (to ~115 large pages) for some sections, and then backing down (to ~23 large pages) again. The result of this behavior is that the working set, which is used to estimate the job's memory requirement for CPU scheduling, *grows* at about the same rate as the actual memory requirement. However, the system uses a scaling factor when *reducing* the working set size, so that once the working set is large it takes several evaluation periods to reduce it, even though the actual memory requirement is much less than the current working set.

Our strategy for reducing the turn-around time on a busy system was twofold. First, we were able to use subroutine calls to a system program Q5ADVISE that allow the programmer to tell the operating system when a page should be moved from memory to disk. We used this instruction to advise the system that certain arrays would not be needed for the up coming step in the calculation. Secondly, the scaling factor used by the system routine PAGER to reduce the job's working set was altered so pages that were advised out would result in a reduced working set in fewer evaluation periods. With a smaller working set, a low-priority job has a greater chance of being scheduled for the CPU. This strategy reduced the time needed to turn the job around by a factor of about three (without, however, significantly affecting the CPU time needed to complete the calculation). It should be noted that if the job was alone in the system these changes would not be needed since they only affect the job's ability to compete in a priority environment for memory. Decisions about working set evaluations and virtual memory management are complex. The change that was made to the system pager, though minor, is necessary to realize any benefits from this strategy, and it will affect all of the jobs running in the system.

To see more concretely how the memory resource demand changes and how the operating system can benefit from user intervention, we must consider in more detail the operation mode of the Cyber 205 computer. The system paging algorithm is based on a first-in, last-out criterion, so after the evaluation of the V matrix for a given sector via Eq. (24), physical memory will contain the V matrix and the last used parts of the B^m matrix. The B^m matrix elements will not be required again until the other steps, Eqs. (5)-(6), are complete, however the system pager will keep them in memory until they are evicted by other memory demands. The next step, the formation of D and its diagonalization, requires an additional N^2 words of memory beyond that for V. Also note that the diagonalization of $\boldsymbol{D}$ is an expensive step ($\sim 10\ N^3$ floating point operations), so it is very inefficient to let the operating system occupy a significant amount of physical memory which is not being used during this step. It is better to page out the B^m matrix elements immediately after they have been used, and this is what we did with the Q5ADVISE subroutine calls. Another place where memory management is helpful is the storage of the eigenvectors $\boldsymbol{T}(i)$ of the D matrix. After they are calculated in sector i, they are needed immediately in Eq. (5), but not again until Eq. (5) is evaluated for sector $i+1$. Thus it is advantageous to page $\boldsymbol{T}(i)$ out immediately after Eq. (5) is calculated. For large-scale calculations on a busy system, these considerations can be critical for good turnaround.

Another possibility for decreasing the memory requirements is to rearrange the order of computational steps. An example of this possibility is provided by the approach we implemented for multiple-energy runs in the R matrix propagation algorithm. As mentioned at the end of Sect. 2.1, the matrices $\boldsymbol{\tau}(i-1, i)$ and their inverses can be re-used for second and subsequent energies. In an early version of our code, we performed a complete calculation at one energy and saved these matrices and their inverses on disk files for each of the 300 to 600 sectors. This storage becomes prohibitive for large-scale calculations, so in our Cyber 205 production code we save $\boldsymbol{R}_4$ for each energy instead of the sector overlap matrices. Thus we determine the R_4 matrix for all energies at sector i before we calculate any quantities for sector $i+1$. Since the number of energies we calculate is typically 3 to 7, it is much less than twice the number of sectors; thus this approach greatly reduces the storage requirements. Our code requires $(6+N_E)N^2$ words of storage for the diagonalization, sector overlap, and propagation steps for a multi-energy calculation consisting of N_E energies.

Alternatively, one can simply recalculate some of the quantities when they are needed a second time rather than saving them. For example, in the R matrix propagation algorithm, for large enough N, the operation count per sector is $64N^3/3$ for first energies and only $20N^3/3$ for second or subsequent energies; thus there is a considerable cost for not re-using the sector transformation matrices. For example, one can perform 3.2 second energy runs for the same cost as the first energy run. In the $\mathscr{L}^2$ method employing Eqs. (12) and (13), the only quantities which can be re-used for subsequent energies are the overlap matrix and coupling vector. Depending on the problem, it may be possible to code the evaluation of the coupling vector in such an efficient manner that it is not prohibitive to recalculate it whenever necessary. This should usually be true for atom-plus-diatom collisions. Other quantities which might be considered as candidates for recalculation are the coefficients $B^m_{\beta\beta'}$ in Eq. (24). We have found that in HF+HF collisions, the calculation of these coefficients can be very time consuming, so that recalculation is not a viable alternative. The main reason for this is that our current algorithm for calculating vector coupling coefficients is not vectorizable. These comments point up clearly the fact that vectorization and memory strategies are far from independent.

Many of the above considerations also apply to the new generation of large-real-memory supercomputers such as the Cray-2, which is currently configured with up to 268 megawords of memory. In the case of the Cray-2 and currently employed time-independent quantal scattering algorithms, the memory is probably sufficient for most scattering calculations which would be affordable in terms of CPU time with current budgets. However, there are instances where saving data on disk instead of memory could be useful. As usual, the primary motivation is economic. Depending on the installation and the charging algorithm, memory usage may cost more than mass storage and data transfers. Another aspect is rolling out of jobs. By rolling out, we refer to the process by which the operating system copies a calculation's physical memory to mass storage in order to use the memory for something else. This may occur frequently in a priority-based batch system or infrequently at ends of availability periods or both. Most installations do not have enough mass storage, so a limit is placed on the memory size of jobs which can be rolled out. By using mass storage where possible to decrease memory usage, a nonrollable calculation could be made rollable. This could mean faster turn-around time, because there is a limit on the number of nonrollable jobs (they must all fit in memory simultaneously), as well as better restart ability in case of system failures or failure to complete by the end of a computer availability period. Eventually it could be the determining factor on whether or not a calculation is doable.

An important consideration for $\mathscr{L}^2$ methods, such as the method reviewed in Sect. 2.2, is the size of the resulting vectors and matrices, and different approaches can significantly alter their sizes and therefore determine which systems can be studied. We will discuss this from the point of view of real memory computers such as the Cray-2, where we must be concerned with time-integrated memory usage and rolling strategy (rather than working sets and paging as on a virtual-memory architecture). We will consider as examples how the storage requirements change for using Eqs. (12) and (13) as compared to Eqs. (19) and (20), for inelastic as compared to reactive scattering, and for the use of localized vs delocalized translational basis functions. Since the coupling vector that appears in Eqs. (12) and (13) is independent of energy, an efficient strategy is to store it for re-use if the study involves several energies, as is usually the case. With this in mind, we will count the storage requirements for several cases.

The storage requirements for the various arrays occurring in the $\mathscr{L}^2$ method are summarized in Table 1. The nonreactive column refers to the case where only one arrangement is included, A + BC refers to reactive scattering with no two atoms identical, and A + B_2 refers to reactive scattering with B identical to C and array elements identical by symmetry stored only once (as discussed below).

First consider the nonreactive scattering case using Eqs. (12) and (13). For nonreactive scattering, the Z matrix is multiplied by zero and thus is not required, so we need arrays 1-5, 7, and 9 in Table 1. (Although this is not shown in Sect. 2.2, the factor $\Delta^{\alpha n}_{m'm}$ appearing in the definition of the E matrix is also used in forming the e vector, and therefore it is required when using Eqs. (12) and (13).) For nonreactive scattering the coupling vector has a particularly convenient form,

Table 1. Array sizes for the $\mathscr{L}^2$ method

Array		Size[a]		
No.	Name	Nonreactive	A+BC[b]	A+B_2[b]
1	${}^{(r)}f(E, R_\alpha)$	$N_\alpha N_\alpha^{QR}$	$3N_\alpha N_\alpha^{QR}$	$3N_\alpha N_\alpha^{QR}$
2	${}^{(i)}f(E, R_\alpha)$	$N_\alpha N_\alpha^{QR}$	$3N_\alpha N_\alpha^{QR}$	$3N_\alpha N_\alpha^{QR}$
3	$\Delta^{\alpha n}_{mm'}$	$M_\alpha^2 N_\alpha$	$3M_\alpha^2 N_\alpha$	$3M_\alpha^2 N_\alpha$
4	$\lambda^{\alpha n}_{m'}(R_\alpha)$	$M_\alpha N_\alpha N_\alpha^{QR}$	$3M_\alpha N_\alpha N_\alpha^{QR}$	$3M_\alpha N_\alpha N_\alpha^{QR}$
5	$b^{\alpha n}_{\beta'}$	$M_\alpha N_\alpha^2$	$9M_\alpha N_\alpha^2$	$9M_\alpha N_\alpha^2$
6	$Z_{\beta\beta'}$	$\cdots$	$6M_\alpha^2 N_\alpha^2$	$3M_\alpha^2 N_\alpha^2$
7	$C_{\beta\beta'}$	$M_\alpha^2 N_\alpha^2$	$9M_\alpha^2 N_\alpha^2$	$9M_\alpha^2 N_\alpha^2$
8	$\hat{C}_{\beta\beta'}$	$M_\alpha^2 N_\alpha^2$	$9M_\alpha^2 N_\alpha^2$	$9M_\alpha^2 N_\alpha^2$
9	$e^{\alpha n}_{\beta'}(R_\alpha)$	$M_\alpha N_\alpha^2 N_\alpha^{QR}$	$9M_\alpha N_\alpha^2 N_\alpha^{QR}$	$5M_\alpha^2 N_\alpha^2 N_\alpha^{QR}$
10	$E_{\beta\beta'}$	$M_\alpha^2 N_\alpha^2$	$9M_\alpha^2 N_\alpha^2$	$5M_\alpha^2 N_\alpha^2$

[a] Assuming $\lambda^{\alpha n}_{m'}(R_\alpha)$ independent of n
[b] Also assuming N_α, M_α, and N_α^{QR} independent of α

i.e.,

$$e_{\beta'}^{\alpha n}(R_\alpha) = \sum_{m''} \Delta_{m'm''}^{\alpha n'} \lambda_{m''}^{\alpha n'}(R_\alpha) \tilde{e}_{nn'}(R_\alpha), \tag{25}$$

where

$$\tilde{e}_{nn'}(R_\alpha) = \int dx_\alpha \, X_n^\alpha(x_\alpha) V_\alpha^C(R_\alpha, x_\alpha) X_n^\alpha(x_\alpha) \tag{26}$$

and V_α^C is the coupling potential. Substituting (26) into (12) yields

$$b_{\beta'}^{\alpha n} = \sum_{m''} \Delta_{mm''}^{\alpha n'} \int dR_\alpha \, \lambda_{m''}^{\alpha n'}(R_\alpha) P_{\beta'}^{\alpha n}(R_\alpha), \tag{27}$$

where

$$P_{\beta'}^{\alpha n}(R_\alpha) = \tilde{e}_{nn'}(R_\alpha) f_n^\alpha(E, R_\alpha). \tag{28}$$

If we use localized basis functions, e.g., distributed gaussians [28], the integrals in (27) can be done using a localized quadrature centered at the peak of $\lambda_{m''}^{\alpha n}(R_\alpha)$, such that N_α^{QR} can be given a small value, say N_α^{QL}, and each $\tilde{e}_{nn'}(R_\alpha)$ needs to be stored at only N_α^{QL} values of R_α. To make the example even more concrete, consider the case where $N_\alpha \cong M_\alpha \cong N_\alpha^{QL} \cong x$, where x might be about 30–50. Then the storage requirement for arrays 1 and 2 is x^2 each, that for 3, 4, and 5 is x^3 each, and that for arrays 7 and 9 is x^4 each and clearly dominates. Neither x^4 array is symmetric.

Now consider the case where we use Eqs. (19) and (20). Then we must store array 10 instead of 9. To take advantage of the localized translational functions in forming array 10, which is the E matrix, the summation over the overlap inverse and the integral that appears in Eq. (21) must be done in separate steps given by Eqs. (22) and (23) because the factor $\sum_{m''}^{M_\alpha} \Delta_{m''m}^{\alpha n} \lambda_{m''}^{\alpha n}(R_\alpha)$ of the integrand of (21) is delocalized even if $\lambda_m^{\alpha n}$ is localized. This separation requires the use of a temporary matrix $\hat{C}$ to carry out the matrix multiply but since $\hat{C}$ has the same dimensions as C, we can use C itself as the temporary matrix. For the case just considered, both $e_{\beta'}^{\alpha n}(R_\alpha)$ and $E_{\beta\beta'}$ require x^4 storage locations. If, however, we used a variational method [26, 27] for which $M_\alpha \cong 0.3x$, then Eqs. (12) and (13) require $\sim 0.4x^4$ storage locations, whereas Eqs. (19) and (20) would require only $\sim 0.2x^4$. Notice however that the relative storage requirements for the two approaches do not depend on the number of channels since all the big arrays scale as N_α^2.

We now consider how these requirements would change if we used delocalized translational basis functions. In this case N_α^{QR} would have to be much larger, for examle 400–1000; we will use the symbol N_α^{QD} to denote a typical value of N_α^{QR} for a delocalized basis. Let $N_\alpha^{QD} \cong 10x$ for discussion purposes. Then the method based on Eqs. (12) and (13) requires about $11x^4$ storage locations whereas that based on Eqs. (19) and (20) still requires only $2x^4$ storage locations. This is the primary motivation for our insertion of the translational basis, Eq. (16). An important consideration though is that, if the insertion step is more slowly convergent than the original problem with respect to increasing M_α, as appears to be the case [26], then the second approach may require larger basis sets - and hence more storage in the long run.

Now consider the A+BC case in Table 1, which applies to three arrangement channels, with N_α, M_α, and N_α^{QR} again independent of α. Notice that the E and C matrices require $9N_\alpha^2M_\alpha^2$, but the Z matrix requires only $6N_\alpha^2M_\alpha^2$, because as seen in Eq. (13), it contributes to only six of the nine possible (α', α) blocks. The table shows that for $M_\alpha \gg 1$, $N_\alpha \gg 1$, the storage requirement is dominated by the C and Z matrices together with either the e vector or the E matrix, and we will therefore concentrate on these terms in the rest of the discussion of storage.

First we continue our comparison of the storage requirements of the two methods of generating the b vector and C matrix. As for nonreactive scattering, the direct formulation of Eqs. (12) and (13) differs from the insertion formulation of Eqs. (19) and (20) in that $E_{\beta'\beta}$ replaces $e_{\beta'}^{\alpha n}(R_\alpha)$. However to save time by taking advantage of the localized translational functions in forming the E matrix, we must use Eqs. (22) and (23) just as for the nonreactive case discussed above. This introduces the temporary $\hat{C}$ matrix which we store in the C matrix and this results in a requirement of storing the Z matrix by itself, instead of storing it in the C matrix as could otherwise be done. Therefore in practice the direct formulation differs from the insertion formulation for localized translational functions in that the E and Z matrices of total size $15M_\alpha^2N_\alpha^2$ replace the e vector of size $5M_\alpha N_\alpha^2 N_2^{QR}$ while for delocalized translational functions the E matrix of size $9M_\alpha^2N_\alpha^2$ replaces the e vector. As discussed above the storage tradeoff depends on the relative sizes of N_α^{QR} and M_α. For reactive scattering though, the e vector is always delocalized so $N_\alpha^{QR} = N_\alpha^{QD} \gg M_\alpha$. Therefore the E matrix formulation of Eqs. (19) and (20) may require an order of magnitude less storage than the e vector method for a given M_α. This, when combined with the extra factor of 9 in Table 1 effectively *precluded* the use of the e vector formulation for our large-scale reactive scattering problems.

Of course, in addition to the convergence problem mentioned above, the space-saving nature of the E matrix formulation is paid for with the extra time required to form the E matrix by performing the calculation in Eq. (21).

For a chemical reaction of the form $A+B_2$, where B_2 is homonuclear, symmetry can be used to cut the storage space needed as compared to that for the general A+BC chemical system as is shown in the last column of Table 1. The symmetry relations between the two arrangement channels of the form B+AB are manifested in the a, b, and e vectors and the E, C, and Z matrices. Because of this symmetry only three (α, α') blocks of the Z matrix need to be stored as compared to six in the general case and only five (α, α') blocks need be stored for the E matrix instead of all nine as in the general case. Similar savings are possible for the e vector, e.g. if all M_α are equal, we need store only two thirds of this vector. Due to the structure of the algebraic Lippmann–Schwinger equation [Eq (7)], though, the symmetry of the C matrix cannot be used to save space since the quantity needed is $(\mathbf{1}-\mathbf{C})^{-1}$. In summary then the A+BC system with localized translational functions requires a C matrix of size $9N_\alpha^2 M_\alpha^2$, a Z matrix of size $6N_\alpha^2 M_\alpha^2$, and an E matrix of size $9N_\alpha^2 M_\alpha^2$ while the A+B_2 system requires a C matrix of size $9N_\alpha^2 M_\alpha^2$, a Z matrix of size $3N_\alpha^2 M_\alpha^2$ and an E matrix of size $5N_\alpha^2 M_\alpha^2$.

Finally we note that when the algorithm is rearranged to cut the storage requirements, the possibilities for vectorization change. Since vectorization can change the CPU time by an order of magnitude or more, this is a very significant consideration. However it is beyond the scope of the present paper.

4. Conclusions and future work

1. The virtual-memory Cyber 205 can be used efficiently for quantal scattering calculations by the R matrix propagation method. Turn-around time can be enhanced by use of the Q5ADVISE subroutine call.

2. We have also performed large-scale quantal scattering calculations on large-memory class-VII supercomputers by use of $\mathscr{L}^2$ methods. The N^3 step in these equations can be carried out by general vectorized linear algebra routines, but further work is required to vectorize the N^2 steps and to find the best compromise of memory storage, disk storage, and number of floating point operations for a given computing environment and charging algorithm.

3. Further increases in the number of channels that can be treated efficiently is possible using time-dependent quantum mechanics [29–33]. In such methods, one solves a true initial value problem rather than a boundary value problem. As a result one can solve for a single initial condition and there need be no steps scaling as the cube of the number of channels. This cuts down the computation time, increasing the size of the problem that can be tackled and hence also increasing the storage demands. In addition results can be obtained over a broad range of energies from a single calculation by using wave packets with a broad energy spectrum. It will be very challenging to understand which approach, time-dependent or $\mathscr{L}^2$ time-independent with optimized basis functions, can provide the most efficient compromise of processor time and storage demand for each category of large-scale dynamics problem on the new large-memory supercomputers.

Acknowledgments. The authors are grateful to Rob Peglar for assistance with the Cyber 205 jobs. This work was supported in part by the National Science Foundation, the Minnesota Supercomputer Institute, and the Control Data Corporation.

References

1. Schwenke DW, Truhlar DG (1985) In: Numrich RW (ed) Supercomputer applications. Plenum Press, New York, p 295
2. Schwenke DW, Truhlar DG (1985) In: Cray Research Inc Science and Engineering Symposium. Minneapolis

3. Schwenke DW, Truhlar DG (1986) Supercomputer simulations in chemistry. Springer, Berlin Heidelberg New York Tokyo, p 165
4. Schwenke DW, Truhlar DG, Coltrin ME (1986) First Symposium on Computational Chemistry on Cray Supercomputers, Minneapolis
5. Schwenke DW, Truhlar DG (1987) Theor Chim Acta 71:1
6. Schwenke DW, Truhlar DG (1987) J Comput Chem, in press
7. Schwenke DW, Truhlar DG, Coltrin ME (1987) J Chem Phys, in press
8. Schwenke DW, Truhlar DG (1987) Theor Chim Acta, 72:1-12
9. Schwenke DW, Truhlar DG, Kouri DJ (1987) J Chem Phys 86:1646
10. Haug K, Schwenke DW, Shima Y, Truhlar DG, Zhang JZH, Kouri DJ (1986) J Phys Chem 90:6757
11. Zhang JZH, Kouri DJ, Haug K, Schwenke DW, Shima Y, Truhlar DG (1987) J Chem Phys, in press
12. Haug K, Schwenke DW, Truhlar DG, Zhang Y, Zhang JZH, Kouri DJ (1987) J Chem Phys, 87:1892
13. Zhang JZH, Zhang Y, Kouri DJ, Haug K, Schwenke DW, Truhlar DG (1987) Faraday Discuss Chem Soc, 84, in press
14. Rabitz H (1972) J Chem Phys 57:1718
15. Zarur G, Rabitz H (1973) J Chem Phys 59:943
16. Pack RT (1974) J Chem Phys 60:633
17. McGuire P, Kouri DJ (1974) J Chem Phys 60:2488
18. Rabitz H (1975) J Chem Phys 63:5208
19. Light JC, Walker RB (1976) J Chem Phys 65:4272
20. Mullaney NA (1979) PhD thesis. University of Minnesota, Minneapolis
21. Truhlar DG, Harvey NM, Onda K, Brandt MA (1979) In: Thomas L (ed) Algorithms and computer codes for atomic and molecular scattering theory, vol 1. National Resource for Computation in Chemistry, Lawrence Berkeley Laboratory, Berkeley, p 220
22. Bellman R, Kalaba R (1956) Proc Natl Acad Sci 42:629
23. Bellman R, Kalaba R, Wing GM (1960) J Math Phys 1:280
24. Bellman R, Kalaba R, Wing GM (1960) Proc Natl Acad Sci 46:1646
25. Secrest D (1979) In: Bernstein RB (ed) Atom-molecule collision theory. Plenum Press, New York, p 265
26. Staszewska G, Truhlar DG (1987) J Chem Phys 86:2793
27. Schwenke DW, Haug K, Zhao M, Truhlar DG, Sun Y, Zhang JZH, Kouri DJ, unpublished
28. Hamilton IP, Light JC (1986) J Chem Phys 84:306
29. Mowrey RC, Kouri DJ (1986) J Chem Phys 84:6466
30. Kouri DJ, Mowrey RC (1987) J Chem Phys 86:2087
31. Mowrey RC, Bowen HF, Kouri DJ (1987) J Chem Phys 86:2441
32. Sun Y, Mowrey RC, Kouri DJ (1987) J Chem Phys, 87:339
33. Mowrey RC, Sun Y, Kouri DJ, Truhlar DG: work in progress