Theor Chim Acta (1991) 79: 199–213

Theoretica Chimica Acta

Benchmark studies of the BCRLM reactive scattering code: Implications for accurate quantum calculations

Edward F. Hayes[1], **Zareh Darakjian**[1], **and Robert B. Walker**[2]

[1] Department of Chemistry, Rice University, Houston, TX 77251, USA

[2] Theoretical Division (T-12, MS B268), Los Alamos National Laboratory, Los Alamos, NM 87545, USA

Received September 1, 1990/Accepted November 13, 1990

Summary. The Bending Corrected Rotating Linear Model (BCRLM), developed by Hayes and Walker, is a simple approximation to the true multidimensional scattering problem for reactions of the type: $A+BC \rightarrow AB+C$. While the BCRLM method is simpler than methods designed to obtain accurate three-dimensional quantum scattering results, this turns out to be a major advantage in terms of our benchmarking studies. The computer code used to obtain BCRLM scattering results is written for the most part in standard FORTRAN and has been ported to several scalar, vector, and parallel architecture computers including the IBM 3090-600J, the Cray XMP and YMP, the Ardent Titan, IBM RISC System/6000, Convex C-1 and the MIPS 2000. Benchmark results will be reported for each of these machines with an emphasis on comparing the scalar, vector, and parallel performance for the standard code with minimum modifications. Detailed analysis of the mapping of the BCRLM approach onto both shared and distributed memory parallel architecture machines indicates the importance of introducing several key changes in the basic strategy and algorithms used to calculate scattering results. This analysis of the BCRLM approach provides some insights into optimal strategies for mapping three-dimensional quantum scattering methods, such as the Parker–Pack method, onto shared or distributed memory parallel computers.

Key words: Chemical reactions – Quantum dynamics – Parallel computing

1. Introduction

In the last several years we have witnessed significant progress in the development of new methods for obtaining accurate state–to–state reaction cross sections for three atom systems [1–11]. While this progress has enabled the study of several real chemical reactions of interest, these computational studies required significant computational resources. As a result, there remains a high level of interest in developing more powerful computational approaches to obtaining quantum scattering information. The most important component in this search for more effective computational methods will continue to be the discovery of better scattering methods and algorithms. However, the rendering of these methods into efficient computer codes will require special attention, if we are to benefit from the enhanced capability of future generations of computer hardware.

The purpose of this paper is to provide a progress report on our benchmark studies of the BCRLM reactive scattering code for several scalar, vector, and parallel architecture machines, and to discuss the implications of these studies for mapping three-dimensional quantum scattering methods, such as the Parker-Pack method (1) onto shared or distributed memory parallel computers.

The Bending Corrected Rotating Linear Model (BCRLM), developed by Hayes and Walker [12], is a simple approximation to the true multidimensional scattering problem for reactions of the type: $A + BC \rightarrow AB + C$. The BCRLM method is simpler than methods designed to obtain accurate three dimensional quantum scattering results. Yet the BCRLM method includes a number of the computational challenges presented by these accurate methods [1-11]. The basic approach we have adapted is to gain as much insight as possible from a thorough analysis and benchmarking of the BCRLM code and to use this experience to guide more effective implementations of three-dimensional methods such as the Parker-Pack method. Since the computation times for a series of BCRLM calculations are modest (i.e., minutes on a Cray YMP), we have been able to execute numerous tests on a number of machines including the IBM 3090-600J, Cray XMP and YMP, Ardent Titan, IBM RISC System/6000, Convex C-1, and MIPS 2000. We have found that having a series of test problems that can be executed without requiring a significant allocation of computer resources is a real advantage at this stage in our work since it enhances our ability to test various compilers, to evaluate the relative scalar, vector, and parallel performance of a machine and to obtain detailed performance statistics.

2. BCRLM method

2.1. Coupled equations

The coupled-channel equations for the BCRLM method have been developed using both natural collision coordinates and hyperspherical coordinates [12]. In this study we have used the code based on natural collision coordinates because it is a more mature code, but the performance of either code is expected to be about the same for the test calculations reported in this paper. Since the details of the derivation of these equations have been reported elsewhere, we report only the final results from Ref. [12b].

In the natural collison coordinates, u and v, the basic equations in matrix form are:

$$\frac{d^2}{du^2} g^J_{nm}(u) = \sum_{k=0}^{N} D^J_{nk} g^J_{km}(u) \tag{1}$$

where the coupling matrix elements D^J_{nm} for each sector, (i), are:

$$\frac{\hbar^2}{2\mu} D^{(i)}_{nm} = \frac{3}{4\sigma^2}\delta_{nm} + \langle G^{(i)}_n|\eta^2\left[V(u^i; v) - V^0(v; i) + \frac{\hbar^2}{2\mu\varrho^2}(J(J+1)+1)\right]|G^{(i)}_m\rangle$$
$$+ [\tfrac{1}{2}(\varepsilon^i_n + \varepsilon^i_m) - E]\langle G^{(i)}_n|\eta^2|G^{(i)}_m\rangle \tag{2}$$

Here u is the collision coordinate, v is a vibrational coordinate perpendicular to u, and J is the total angular momentum. The potential term $V^0(v; i)$ is the potential at the center of sector (i) and $V(u^i; v)$ is the full potential in sector (i) including the bending correction terms. We have suppressed the labels for J and the bending quantum numbers, λ. The functions $G_n^{(i)}$ are obtained by solving the reference vibrational Hamiltonian defined at the center of each sector:

$$\left[\frac{\hbar^2}{2\mu}\frac{d^2}{dv^2} + V^0(v; i) - \varepsilon_n^i\right] G_n^{(i)} = 0 \tag{3}$$

The energy and angular momentum dependence of matrix elements defined in Eq. (2) can be written as follows:

$$D_{nm}^{(i)}(E) = D_{nm}^{(i)}(E_{\text{ref}}, J = 0) + (E_{\text{ref}} - E)A_{nm}^{(i)} + J(J+1)B_{nm}^{(i)} \tag{4}$$

where the energy and angular momentum independent matrices, $A_{nm}^{(i)}$ and $B_{nm}^{(i)}$ are given by:

$$A_{nm}^{(i)} = \frac{\hbar^2}{2\mu}\langle G_n^{(i)}|\eta^2|G_m^{(i)}\rangle \tag{5}$$

and

$$B_{nm}^{(i)} = \frac{\hbar^2}{2\mu}\langle G_n^{(i)}|\eta^2\varrho^{-2}|G_m^{(i)}\rangle \tag{6}$$

To apply the appropriate boundary conditions for reactive scattering we also need the coupled equations in Jacobi coordinates, R and r:

$$\frac{d^2}{dR^2} f_{nm}(R) = \sum_{k=0}^{N} C_{nk} f_{km}(R) \tag{7}$$

where the coupling matrix elements C_{nm} are:

$$\frac{\hbar^2}{2\mu} C_{nm} = (\varepsilon_n - E)\delta_{nm} + \langle F_n | V(R, r) - V^0(r) + \frac{\hbar^2}{2\mu\varrho^2}(J(J+1)+1)|F_m\rangle \tag{8}$$

Here R is the distance from the separated atom to the center of mass of the associated diatomic molecule and r is the internuclear separation of the diatomic molecule. The potential term $V(R, r)$ is the full potential in the asymptotic atom–diatom region and $V^0(r)$ is the asymptotic diatomic potential. The functions F_n are obtained by solving the reference vibrational Hamiltonian:

$$\left[\frac{\hbar^2}{2\mu}\frac{d^2}{dr^2} + V^0(r) - \varepsilon_n\right] F_n = 0 \tag{9}$$

The potential, $V^0(r)$, will in general depend on the particular arrangement channel (i.e., reactants or products) but we have suppressed the channel index in Eq. (9).

2.2. *Vibrational basis representations*

Equations (3) and (9) are solved by expanding the eigenfunctions in a set of harmonic oscillator functions, $\phi_n(i)$, that have been selected for each sector. While the matrix elements in Eqs. (2) and (8) may be solved efficiently in this representation, it is convenient to transform to the representation that diagonalizes these equations, since this permits one to contract the basis set thus reducing the order of the matrix to be propagated and, importantly, replacing a symmetric matrix with a vector containing the eigenvalues.

At this point we can define **Stage 1** of the BCRLM code. It consists of the following steps:

— Generate the Primitive Basis Set, $\phi(i)$.
— Evaluate $\boldsymbol{D}^{(i)}(E_{\text{ref}})$, $\boldsymbol{A}^{(i)}$ and $\boldsymbol{B}^{(i)}$.
— Diagonalize $\boldsymbol{D}^{(i)}(E_{\text{ref}})$ and transform $\boldsymbol{A}^{(i)}$ and $\boldsymbol{B}^{(i)}$ to the diagonal representation.

Each of these steps must be performed for all of the sectors in the natural collision coordinate representation and for the two asymptotic Jacobi arrangements corresponding to reactants and products. Since each of these operations can be carried out independently of all the others, this **Stage 1** operation is ideally suited for parallel computation.

2.3. Matching

As the wavefunction is propagated from one sector to the next with the advancing natural collision coordinate, one must make certain that the wavefunction and its derivative are continuous across the sector boundary. The key to meeting this requirement is the sector–to–sector overlap matrix, $[\sigma(i, i+1)]_{nm}$, which is given by:

$$[\sigma(i, i+1)]_{nm} = \langle G_n(v; i) | G_m(v; i+1) \rangle \tag{10}$$

With this overlap matrix we can easily enforce the sector matching by requiring that

$$\boldsymbol{g}_R(i-1) = \sigma(i, i+1) \boldsymbol{g}_L(i) \tag{11}$$

and

$$\frac{d}{du} \boldsymbol{g}_R(i-1) = \sigma(i, i+1) \frac{d}{du} \boldsymbol{g}_L(i) \tag{12}$$

where $\boldsymbol{g}_R(i-1)$ is the value of the matrix of solutions to Eq. (1) at the right side of sector $(i-1)$, and $\boldsymbol{g}_L(i)$ is the value of the matrix at the left hand side of sector (i).

In matching to the Jacobi coordinate representation for the reactant and product channels the matching conditions are a bit more involved. At these matching boundaries we need the following two matrices:

$$S^{(1)}_{nm} = \langle G_n(r; N) | \eta^{-1/2} | F_m(r; N+1) \rangle \tag{13}$$

and

$$S^{(2)}_{nm} = \langle G_n(r; N) | \eta^{1/2} | F_m(r; N+1) \rangle \tag{14}$$

The asymptotic sector–to–sector continuity is preserved as follows:

$$\boldsymbol{g}_R(N) = \boldsymbol{S}^{(1)} \boldsymbol{f}_L(N+1) \tag{15}$$

and

$$\frac{d}{du} \boldsymbol{g}_R(N) = \boldsymbol{S}^{(2)} \frac{d}{du} \boldsymbol{f}_L(N+1) \tag{16}$$

where N is the last of the natural collision coordinate sectors.

Now we can define the **Stage 2** in the BCRLM code in terms of the following steps:

— Determine the sector–to–sector overlap matrices.
— Transform the overlap matrices to the local eigenfunction representation.

Both of these steps must be carried out for each of the sector intersections, N, using Eqs. (13) and (14) for the final asymptotic entrance and exit channel matching and Eq. (10) for other sector–to–sector boundaries. There are three points that should be noted here. First of all each of these operations can be carried out independently of the others. Thus **Stage 2** is also ideally suited for parallel computation. Moreover, if a distributed memory machine is to be used for both **Stage 1** and **Stage 2**, the interprocessor communication between stages would be relatively low since the data required by any Stage 2 processor are generated by only two **Stage 1** processors, and, importantly, in most instances these processors can be arranged to be nearest neighbors.

3. Solving the coupled equations

3.1. R-matrix method

The method selected to solve the coupled–channel equations is the R–matrix propagation method of Light and Walker [13]. The particular notation used here follows from Ref. [12b].

As noted in the previous section the coupling matrices $\boldsymbol{D}(i)$, Eq. (4) are calculated at the center of each sector and are assumed to be constant within the sector. Since the $\boldsymbol{D}(i)$ are real symmetric matrices, they may be diagonalized in each sector by a real orthogonal matrix $\boldsymbol{U}(i)$:

$$\boldsymbol{U}^T(i)\boldsymbol{D}(i)\boldsymbol{U}(i) = \lambda^2(i) \tag{17}$$

where $\boldsymbol{U}^T(i)$ is the transpose of $\boldsymbol{U}(i)$. In each sector the propagation functions $f_{nm}(i)$ must be transformed to this locally diagonal representation giving the new propagation functions:

$$\tilde{f}(R;i) = \boldsymbol{U}(i)f(R;i) \tag{18}$$

The global R matrix between the initial sector, 0, and sector i is:

$$\begin{bmatrix} \tilde{f}(R_0^-;0) \\ f(R_i^+;i) \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_1(i) & \boldsymbol{R}_2(i) \\ \boldsymbol{R}_3(i) & \boldsymbol{R}_4(i) \end{bmatrix} \begin{bmatrix} -\tilde{f}'(R_0^-;0) \\ f'(R_i^+;i) \end{bmatrix} \tag{19}$$

The sector R–matrix relating the values of the locally uncoupled functions to the derivatives within sector j is:

$$\begin{bmatrix} \tilde{f}(R_j^-;j) \\ \tilde{f}(R_j^+;j) \end{bmatrix} = \begin{bmatrix} r_i(j) & r_2(j) \\ r_3(j) & r_4(j) \end{bmatrix} \begin{bmatrix} -\tilde{f}'(R_j^-;j) \\ f'(R_j^+;j) \end{bmatrix} \tag{20}$$

where for open channels ($\lambda^2 \leqslant 0$) we have:

$$\begin{aligned} [r_1(i)]_{nm} &= [r_4(i)]_{nm} = \delta_{nm}[-|\lambda_n(i)|^{-1} \cot \Delta R_i |\lambda_n(i)|] \\ [r_2(i)]_{nm} &= [r_3(i)]_{nm} = \delta_{nm}[-|\lambda_n(i)|^{-1} \csc \Delta R_i |\lambda_n(i)|] \end{aligned} \tag{21}$$

For closed channels ($\lambda^2 \geqslant 0$) the sector R–matrix is:

$$\begin{aligned}[r_1(i)]_{nm} &= [r_4(i)]_{nm} = \delta_{nm}[|\lambda_n(i)|^{-1} \coth \Delta R_i |\lambda_n(i)|] \\ [r_2(i)]_{nm} &= [r_3(i)]_{nm} = \delta_{nm}[|\lambda_n(i)|^{-1} \operatorname{csch} \Delta R_i |\lambda_n(i)|]\end{aligned} \tag{22}$$

In Eqs. (19) and (20) ΔR_i is the width of sector i.

To propagate from sector (i) to sector $(i+1)$ we need to transform the sector–to–sector overlap matrix, $[\sigma(i, i+1)]_{nm}$, to the locally uncoupled representation of sector $(i+1)$:

$$\boldsymbol{T}(i, i+1) = \boldsymbol{U}^T(i)\sigma(i, i+1)\boldsymbol{U}(i+1) \tag{23}$$

The working R-matrix recursion relations are [12b] as follows:

$$\boldsymbol{R}_1(i+1) = \boldsymbol{R}_1(i) - \boldsymbol{R}_2(i)\boldsymbol{T}(i, i+1)\boldsymbol{Z}(i+1)\boldsymbol{T}^T(i, i+1)\boldsymbol{R}_3(i) \tag{24}$$

$$\boldsymbol{R}_2(i+1) = \boldsymbol{R}_3^T(i+1) = \boldsymbol{R}_2(i)\boldsymbol{T}(i, i+1)\boldsymbol{Z}(i+1)\boldsymbol{r}_2(i+1) \tag{25}$$

$$\boldsymbol{R}_4(i+1) = \boldsymbol{r}_4(i+1) - \boldsymbol{r}_3(i+1)\boldsymbol{Z}(i+1)\boldsymbol{r}_2(i+1) \tag{26}$$

$$\boldsymbol{Z}(i+1) = [\boldsymbol{r}_1(i+1) + \boldsymbol{T}^T(i, i+1)\boldsymbol{R}_4(i)\boldsymbol{T}(i, i+1)]^{-1} \tag{27}$$

In the BCRLM code, we propagate all four blocks of the R–matrix outward first towards the entrance channel and then towards the exit channel. The two R matrices are then combined and the scattering boundary conditions are enforced. The propagation of the coupled equations defines **Stage 3** of the calculations. It is relatively straightforward to see that given the setup information from **Stage 1** and **Stage 2** all of the energy calculations involve the same steps for each sector, namely:

— Gather needed matrices from earlier stages.
— Generation of the coupling matrix elements $\boldsymbol{D}(i)$.
— Diagonalization of $\boldsymbol{D}(i)$.
— Transformation of the matching matrix, $\sigma(i, i+1)$.
— R–matrix propagation.
— Temporary Storage for the Asymptotic R–matrix elements with E and J labels.

Thus as we look at the feasibility of using various parallel architecture machines to perform numerous energy calculations there are a number of critical factors that must be assessed. In a *shared memory* environment, one can achieve high levels of parallelization by just assigning different energy and total angular momentum values to different processors, and, importantly the temporary storage for large numbers of matrices is straightforward. However, in a *distributed memory* environment it may be more efficient to assign one or more sectors to a particular processor and run the various energy and angular momentum calculations through the processors in a pipeline mode. This approach cuts down on the amount of interprocessor communication needed to gather the necessary matrices from the earlier stages, and, if the number of calculations needed is large relative to the number of processors, the percentage of time that there will be idle processors will be acceptable. Moreover, since the matrix elements that need to be moved to temporary storage all come from a single processor at the end of the pipeline, it is straightforward to pass the asymptotic R–matrix elements from that processor to external storage. In the near future we plan to test these assertions concerning operations in a *distributed memory* environment on the 32 node INTEL iPSC/860 machine at Rice University.

3.2. Boundary conditions

For large values of R the coupled-channel equations (Eq. (8)) are decoupled and we have (ignoring an extra $1/\varrho^2$ term):

$$\frac{\hbar^2}{2\mu} D_{nm} = \left[\varepsilon_n - E + \frac{\hbar^2}{2\mu R^2}(J(J+1)) \right] \delta_{nm} \tag{28}$$

As a result the scattering wavefunctions, $f_{nm}(R; J)$ have the asymptotic form:

$$f_{nm}(R; J) \to -ik_n R[h_J^{(2)}(k_n R)\delta_{nm} + h_J^{(1)}(k_n R)(k_m/k_n)^{1/2} S_{nm}^J] \tag{29}$$

where k_n is the channel wavenumber, $\hbar^2 k_n^2 = 2\mu(E - \varepsilon_n)$, S_{nm}^J is an element of the S-matrix, and the functions $h_J^{(i)}$ are spherical Hankel functions of the first and second kind. To obtain the S-matrix from the final R matrix we need Eq. (29) and its derivative in the form:

$$\boldsymbol{f}(R; J) = \boldsymbol{IN}(R; J) - \boldsymbol{OT}(R; J)\boldsymbol{k}^{-1/2}\boldsymbol{S}^J\boldsymbol{k}^{1/2} \tag{30}$$

$$\boldsymbol{f}'(R; J) = \boldsymbol{IN}'(R; J) - \boldsymbol{OT}'(R; J)\boldsymbol{k}^{-1/2}\boldsymbol{S}^J\boldsymbol{k}^{1/2} \tag{31}$$

where the $\boldsymbol{f}(R; J)$ and $\boldsymbol{f}'(R; J)$ matrices are determined by propagation into the entrance and exit channel asymptotic regions. The $\boldsymbol{IN}(R; J)$ and $\boldsymbol{OT}(R; J)$ are diagonal matrices of the spherical Hankel functions:

$$[OT(R; J)]_{nm} = ik_n R h_J^{(1)}(k_n R) \tag{32}$$

and

$$[IN(R; J)]_{nm} = [OT(R; J)]_{nm}^* \tag{33}$$

Letting $R^\infty(J)$ be the final calculated R-matrix for a particular value of the total angular momentum, J, following Ref. [12b] we may write down the expression for the S matrix in terms of $R^\infty(J)$:

$$\boldsymbol{S}^J = \boldsymbol{k}^{1/2}[\boldsymbol{OT}(J) - \boldsymbol{R}^\infty(J)\boldsymbol{OT}'(J)][\boldsymbol{OT}(J) - \boldsymbol{R}^\infty(J)\boldsymbol{OT}'(J)]^*\boldsymbol{k}^{-1/2} \tag{34}$$

The integral cross section may be calculated from the S-matrix elements as follows:

$$\sigma_{nm}(E) = \pi k_m^{-2} \sum_{J=0}^{\infty} (2J+1)|\delta_{nm} - S_{nm}^J|^2 \tag{35}$$

In the last stage in the BCRLM code, **Stage 4**, we need to carry out the following operations for each energy:

— Gather the $\boldsymbol{R}^\infty(J)$ matrices for each energy.
— Determine Hankel functions and derivatives.
— Generate $\boldsymbol{S}^J$ matrices.
— Calculate the integral cross section.
— Output of the final results.

For a typical scattering study we are interested in determining the integral cross section for numerous energy and angular momentum values. Here again one can achieve high levels of parallelization by assigning different energy and angular momentum values to different processors. The number of operations needed to gather the necessary matrices, $\boldsymbol{R}^\infty(J)$, will depend both on the strategy selected for **Stage 3** and whether memory is shared or distributed. For a *shared memory* environment this step is relatively simple – at most requiring a simple sort if all the matrices (J and E) are not available at once in core. However, in a *distributed memory* environment the gathering of the needed matrices and the communication to the appropriate processor is more involved, but it is still a relatively simple sorting process.

4. Results and discussion

4.1. Computer systems used

To date we have obtained benchmark results using seven different computing systems. A summary of the characteristics of these systems is presented in Table 1. For each system studied we have generated timings on a standard problem, the reaction $F + H_2 \rightarrow H + HF$. The Muckerman 5 potential energy surface [14] is used to represent the FH_2 interaction potential and the number of coupled channels is systematically varied from 5 to 30. For the comparison of scalar, vector, the parallel performance we have taken a relatively simple test case that consists of four energy calculations (i.e., $E = 1.75$, 1.65, 1.66, and 1.67 eV) for $J = 0$.

The IBM 3090 system used here is operated by the Cornell National Supercomputer Facility, CNSF. The Cray XMP and YMP machines are located at Los Alamos National Laboratory, and execute under the CTSS operating system. The Ardent Titan is owned by the T-12 Group at Los Alamos. The IBM RISC Systems/6000 is operated by the Bonner Physics Laboratory at Rice University. The Convex-1 is run by the Center for Research on Parallel Computing at Rice University. All calculations were carried out between June 1, 1990 and August 30, 1990.

4.2. Code modifications

The BCRLM code is based on the original RXN1D scattering code [15] with several modifications. The code has been modified in its physics content in two significant ways: (1) to solve the BCRLM equations, rather than those limited to collinear reaction events, and (2) to compute potential matrix elements over the harmonic oscillator primitive basis functions by an efficient Gauss–Hermite quadrature. Other cosmetic modifications to the code have been made to improve its portability between a variety of different computing environments,

Table 1. Characteristics of computers used

Machine	Number of Processors	Operating System	FORTRAN Compiler	Memory (MBytes)
IBM 3090-600J	6	VM/XA	Fortvs/pfpcomp	512
Cray-XMP	4	CTSS	CFT 1.16	512
Cray-YMP	8	CTSS	CFT 1.16	1024
MIPS 2000	1	RISC/OS 6.0	F77 2.0	32
Convex-1	1	Convexos 8.0	F77 6.0	64
Ardent Titan	2	UNIX	NFC	64
RISC System/6000-530	1	AIX 3.0	xlf	32

particularly in the handling of input and output functions. None of these modifications significantly affects how long it takes to do scattering calculations. Several obvious modifications were made to improve the computational performance of the code, and to improve its vectorizability. These modifications focus on the portions of the code that perform the standard matrix algebra functions of matrix addition, multiplication, inversion, and diagonalization. Wherever possible, inner DO loops in the original code were replaced with calls to the equivalent routines available in the Basic Linear Algebra Subroutines (BLAS routines). Wherever possible, imbedded loop structures were rearranged to improve the vectorizability or typical trip counts of the inner loop. The BLAS routine that performs most of the matrix work is SAXPY (or DAXPY in double precision versions).

To render the code in a form appropriate for parallel execution it was necessry to rewrite major sections of the code to conform to the four stages outlined in the previous sections. The original code, which was designed for a serial environment, mixed the code for the four stages in such a way that it was not possible to unscramble them using normal compiler directives. In fact, early attempts to generate a code suitable for execution on the IBM 3090-600J without major code modifications produced code that actually ran slower in terms of both wall time and total CPU time. Thus, the code used to obtain timings for the parallel operation on the IBM 3090-600J, while it carries out the same operations and uses the same algorithms as for the serial and vector timings, has been fully segmented into the four stages that we presented above for parallel computing in a *shared memory* environment. At Stage 3 and Stage 4 each of the energies was assigned to a different processor.

4.3. *Scalar and vector comparisons*

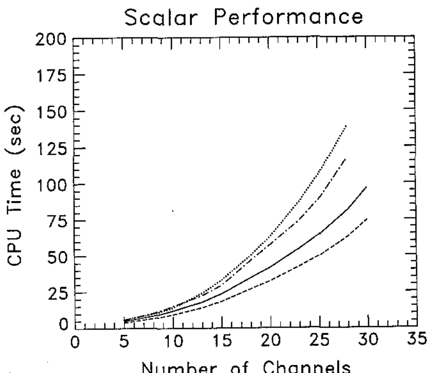
While this study involves only 7 different machines, there are actually 11 different cases that we have investigated corresponding to different levels of scalar, vector, and parallel operations. The details of each of these cases are summarized in Table 2.

Table 2. Cases studied

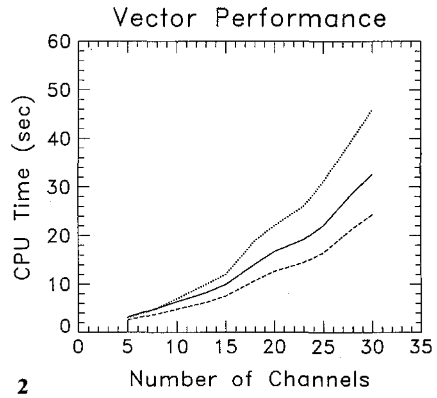
Case Label	Machine	Processors Used	Mode
IBM-S	IBM 3090-600J	1	Scalar
IBM-V	IBM 3090-600J	1	Vector
IBM-P	IBM 3090-600J	4	Parallel
Cray-XS	Cray-XMP	1	Scalar
Cray-XV	Cray-XMP	1	Vector
Cray-YS	Cray-YMP	1	Scalar
Cray-YV	Cray-YMP	1	Vector
MIPS	MIPS 2000	1	Scalar
Conv	Convex-1	1	Vector
Ardnt	Ardent Titan	1	Vector
RISC	RISC System/600	1	Vector[a]

[a] While not a vector architecture, the RISC System/6000 machine performs best on codes that vectorize well

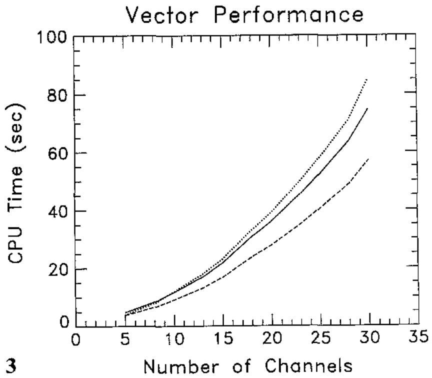
In Fig. 1, we compare the execution times for the three scalar cases. Then in Fig. 2 the vector cases are compared as a function of the number of coupled equations being solved for the IBM 3090 600J, and the Cray XMP and YMP. For each case in Fig. 2 we have used the optimized assembly language equivalents to the BLAS (Basic Linear Algebra Subroutines). In Fig. 3, for comparison purposes we present timings obtained using FORTRAN versions of the BLAS. There are significant improvements in performance due to the use of the machine language versions. In Fig. 4, we compare vectorized runs on the



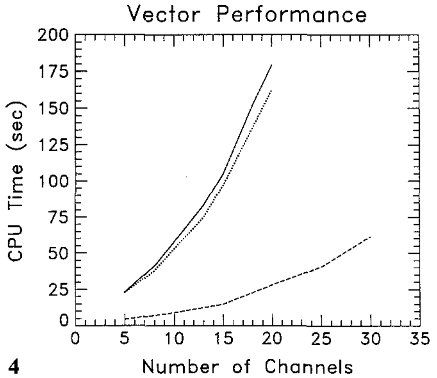
1



2



3



4

Fig. 1. Scalar performance (CPU time) vs. number of coupled equations, *N*. See Table 2 for definition of case labels. Legend: (*dots*) *IBM-S*, (*solid line*) Cray-XS, (*dashed line*) Cray-YS, (*dash-dot line*) MIPS

Fig. 2. Vector performance (CPU time) vs. number of coupled equations, *N*, obtained using optimized assembly language versions of the BLAS. Legend: (*dots*) IBM-V, (*solid line*) Cray-SV, (*dashed line*) Cray-YV

Fig. 3. Vector performance (CPU time) vs. number of coupled equations, *N*, using vectorized FORTRAN versions of the BLAS. See Table 2 for definition of case labels. Legend: (*dots*) IBM-V, (*solid line*) Cray-XV, (*dashed line*) Gray-YV

Fig. 4. Vector performance (CPU time) vs. number of coupled equations, *N*, obtained using vectorized FORTRAN versions of the BLAS. Legend: (*dots*) Ardnt, (*solid line*) Conv, (*dashed line*) RISC

Table 3. CPU times in seconds as a function of N

Case Label	N					
	5	10	15	20	25	30
IBM-S	6.0	14.0	33.0	63.0	106.0	173.0
IBM-V	2.0	7.0	13.0	22.0	31.0	46.0
Cray-XS	4.9	12.0	23.6	41.6	64.3	96.9
Cray-XV	3.2	6.4	9.9	16.7	22.0	32.6
Cray-YS	3.9	9.4	18.4	32.3	49.7	74.7
Cray-YV	2.6	4.8	7.6	12.6	16.5	24.3
MIPS	5.7	15.0	29.4	57.5	88.7	148.2
Conv	23.2	57.7	105.5	179.5	235.5	262.4
Ardnt	22.9	53.0	94.8	158.6	214.2	363.3
RISC	4.4	9.1	15.0	28.1	40.6	61.5

Table 4. Scalar to vector speedups as a function of N

Case Label	N					
	5	10	15	20	25	30
IBM-3090	2.0	2.0	2.5	2.9	3.4	4.4
Cray-XMP	1.3	1.9	2.4	2.5	2.9	3.0
Cray-YMP	1.5	2.0	2.4	2.6	3.0	3.1
	Fractional vectorization					
IBM-3090	0.60	0.61	0.69	0.75	0.79	0.86

Ardent Titan, and the Convex-1, with the RISC System/6000. In each of these cases we have used FORTRAN versions of the BLAS. The performance of the RISC System without the special machine language BLAS is within about a factor of two of the vectorized IBM 3090-600J, and the Cray XMP and YMP performance with the machine language BLAS.

In Table 3 we present the performance of the various cases as a function of N.

In Table 4 we report the scalar to vector speedups for selected values of the total number of coupled channels obtained for the Cray XMP, Cray YMP, and IBM 3090. Also included in this table are the simple Amdahl's law analysis of the fractions of the BCRLM code that are vectorizable, f_v as determined by analyzing the time spent in the vector unit on the IBM 3090. For N equal to 30, the percent vectorization is 86%.

4.4. Parallel results for the IBM 3090-600J

We have modified the BCRLM code extensively so that it can be run in parallel on a shared memory parallel architecture machine. We have carried out parallel test runs on the IBM 3090-600J at the Cornell National Supercomputer Facility for each of the four stages indicated in the previous section. These preliminary test runs indicate that there is very little additional overhead associated with the parallel execution – probably less than 1%. As a result we expect to be able to achieve speedups that are nearly linear with the total number of processors. We are not reporting detailed timing runs at this time because we are continuing to upgrade the new code and to make certain that its scalar, vector, and parallel performance has been optimized to the same extent. We expect to have these results in the near future.

4.5. Implications of 3D reactions scattering calculations

In this section we draw on our experience [16, 17] with the Parker–Pack Adiabatically-adjusting, Principal axis Hyperspherical (APH) coordinates approach to obtaining accurate 3D scattering results [1]. While the Parker–Pack method involves many additional operations not encountered in the BCRLM approach, the basic idea of dividing up the computations into four separate stages carries over to this 3D approach without major modifications.

To demonstrate this we need to review some of the basic equations from Ref. [1]. In the APH theory the total scattering wavefunction is expanded in a basis of sector adiabatic surface functions. These surface functions are bound state eigenfunctions of the surface Hamiltonian:

$$H(\theta, \chi; \varrho_\xi)\Phi_t(\theta, \chi; \varrho_\xi) = \mathscr{E}_t(\varrho_\xi)\Phi_t(\theta, \chi; \varrho_\xi) \tag{36}$$

where θ and χ are the two APH hyperangles. Equations (36) also depends parametrically on ϱ_ξ, the center of a sector, where the range of the APH hyperradius is divided into n sectors, $\xi = 1, \ldots, n$. When the total scattering wavefunction is substituted into the full Schrödinger equation for $J = 0$, a set of n coupled channel (CC) equations is obtained, where N is the number of surface functions ($t = 1, \ldots, N$) in the CC expansion. These exact CC equations are propagated from ϱ_1 to ϱ_n using the log-derivative method [18], then the boundary conditions are applied as usual [1b].

This first stage of the 3D calculations is by far the most time consuming and demanding of memory. For instance for our $J = 0$ study of the reaction $He + H_2^+ \rightarrow HeH^+ + H$ on the Ardent Titan, the first stage required about 90 hours, the second stage about 10 hours, and the combined third and fourth stages about 8 minutes per energy after the first energy which requires about 16 minutes. However, the total time involved in these **Stage 3** and **Stage 4** calculations was nearly 48 hours, since nearly 350 energy calculations were needed to obtain the desired energy resolution over the energy range studied. Clearly we need significant improvements in computation speed.

Shared memory environment. For a large shared memory parallel computer, typically with a modest number of processors, the Parker–Pack approach lends itself to high levels of parallel activity in very much the same way that we have outlined for the BCRLM code. Solutions to Eq. (36) may be obtained by dividing up the total number of sectors into groups of equal numbers (approximately) of adjacent sectors so that each processor may be assigned about the same amount of work. Following this initial assignment, the surface function calculations can be carried out in parallel without generating any significant additional computational overhead. This approach has a possible advantage in that different techniques may be used for solving Eq. (36) in different regions of ϱ space. For the Parker–Pack approach there is some additional computational overhead compared to the BCRLM code that is related to the larger sizes of the matrices involved in a 3D calculation. Since the number of surface function matrix elements is much larger than that encountered in the BCRLM code, one cannot keep all of these matrix elements for all of the sectors in main memory at the same time. However, by assigning a group of adjacent sectors to the same processor one can interleave the first two stages and then generate the sector–to–sector overlap matrices before it is necessary to move one of the adjacent surface functions to temporary storage. The only exceptions to this are for adjacent sectors that are assigned to different processors. The sector surface function matrices needed to calculate these particular sector–to–sector overlap matrices can be kept in main memory or retrieved from temporary files. The decision on whether to keep or retrieve these matrix elements will depend on the amount of main memory available and the size of the problem being solved.

At the end of these first two stages we have the matrix elements needed for the propagation stage, **Stage 3**. Since these matrix elements cannot all be held in main memory at one time, for efficient propagation of the coupled equations they must be arranged in temporary file space so that they can be read sequentially from the lowest to the highest ϱ values as needed. If care is taken during the first two stages, this can easily be accomplished by creating two separate files for each group of sectors (i.e., one for the surface function matrices and another for the sector-to-sector overlap matrices). Then within each range of sectors all the matrices can be appropriately ordered. The propagation code then only needs to know the sector ranges assigned to each file.

By assigning different energies to each processor for **Stage 3** one can achieve high levels of parallel activity. An issue that comes up here is whether it is desirable to synchronize the timing among the processors so that each processor is working on the same sector during the same time period. If this is done, the number of file reads will be reduced by the number of processors used. The problem is that some processors will need to wait while others are finishing up work for the active sector. Since each of the processors is basically going through the same basic steps, the amount of time lost waiting will be a small percentage of the total compute time per sector.

For Stage 4 the assembly of the asymptotic matrices needed to calculate the S matrix can be carried out in parallel fashion as in the case of the BCRLM code. Each processor just continues on from the **Stage 3** calculations with the values of the asymptotic R–matrix elements and calculates the S–matrix for the assigned energy.

Distributed memory environment. Here, we are most encouraged by the work of Kuppermann et al. [19]. Their important work, about which we expect to learn more at this Conference on Parallel Computing for Chemical Reactivity, has shown the wave of the future for distributed memory parallel computing. Using the Caltech/JPL Mark IIIfp 64 processor hypercube they have demonstrated that such a distributed memory parallel architecture machine can be competitive with single processor computation speeds on the Cray XMP, Cray II, and Cray YMP. To achieve this improved performance they adopted a different strategy from that presented here for the BCRLM code. The change is necessary because the 3D hyperspherical treatment as implemented by Kuppermann's group, and by Parker and Pack requires many very large matrices to generate the surface functions for a particular sector. For example, the version of the Parker–Pack code that uses the Discrete Variable Representation (DVR) approach [20, 21, 22] and a sequential diagonalization-truncation procedure [23] requires about 50 Mbytes of memory for each sector. When the amount of distributed memory associated with each of the processors is less than this, it is necessary to distribute the operations for a sector over several processors. For example, on the INTEL iPSC/860 at Rice University, which has 8 Mbytes per node, one would need to allocate seven or eight processors to each sector in order to have enough random access memory to write an efficient code. However, since the BCRLM code fits easily within 8 Mbytes, it is not necessary to allocate more than one processor to a sector during **Stage 1** processing.

The calculation of the sector–to–sector overlap matrices, **Stage 2**, also requires considerable random access memory for efficient processing. As a result, several processors are also needed for this stage. Moreover, since this stage requires matrix elements from adjacent sectors it will be more efficient to interleave the first two stages so that the sector–to–sector overlap matrices are calculated before one of the sector surface functions is moved to temporary storage (e.g., the concurrent file system on the INTEL iPSC/860).

For Stage 3 we note that the Kuppermann group [19] has obtained good results for this propagation stage by clustering processors in groups of eight and then assigning different energies to each cluster. The final stage of calculating the S matrix from the asymptotic values generated in **Stage 3** is straightforward for parallel execution using the Kuppermann cluster approach [19], and is not very time consuming.

Direct calculation of time delays. For chemical reactions that exhibit quantum resonances (long-lived complexes) there is considerable interest in being able to calculate the resonance lifetime. Smith [24] has shown that the scattering time delay, Δt_{uv}, may be calculated from the state–to–state S–matrix elements, S_{uv}, as follows:

$$\Delta t_{uv} = Re[-i\hbar(S_{uv})^{-1}\, dS_{uv}/dE] \tag{37}$$

where u and v are the initial and final set of quantum numbers for the state–to–state process. While the scattering approaches discussed above provide the S–matrix elements directly at each energy, the energy derivatives must be determined separately. The traditional way to calculate the energy derivatives of the S–matrix is to calculate the S–matrix at many closely spaced energies and then find the energy derivatives by numerical differentiation.

Recently it has been shown [17] that the Parker–Pack method may be extended to include the direct calculation of the energy derivatives of the S–matrix. The additional code to generate these energy derivatives at the same time as the S–matrix is being calculated has now been added to the Parker–Pack code [25]. Test results on HeH_2^+ indicate that the method is both accurate and efficient. Furthermore, the additional steps associated with this direct method fit well within the context of the parallel architecture approaches discussed above both for shared and distributed memory machines. For systems with significant resonance structure this direct method offers a straightforward way to reduce the number of energy calculations required by as much as a factor of ten.

The future. The prospect for future 3D quantum scattering studies is encouraging both for shared memory parallel computers such as the IBM 3090-600J and the Cray YMP 8/64, as well as for distributed memory parallel computers. It will be interesting to see if distributed memory parallel architecture machines such as the INTEL iPSC/860, with 8 Megabytes of memory at each of 128 nodes will outperform shared memory machines operating in parallel. It will also be interesting to see if we are able to develop approaches that require less memory per processor and what the trade-offs will be between memory requirements and additional computing requirements. For 3D reactive scattering, memory may turn out to be dearer than computing power.

Acknowledgements. This work was performed in part under the auspices of the U.S. Department of Energy. One of us (E.F.H.) was supported by a grant from the Robert A. Welch Foundation. The authors would like to thank Russ Pack, Joel Kress, and Greg Parker for many insightful discussions on 3D quantum reaction dynamics. We also wish to thank Danny Sorensen for several insights into how to make effective use of distributed memory parallel computers. We also thank Phil Pendergast for helpful suggestions and comments on the content of this paper.

References

1. (a) Parker GA, Pack RT, Archer BJ, Walker RB (1987) Chem Phys Lett 137:564; (b) Pack RT, Parker GA (1987) J Chem Phys 87:3888
2. (a) Zhang JZH, Miller WH (1987) Chem. Phys. Lett. 140:329; (b) Zhang JZH, Chu SI, Miller WH (1988) J Chem Phys 88:4549
3. (a) Schwenke DW, Haug K, Truhlar DG, Sun Y, Zhang JZH, Kouri DJ (1987) J Phys Chem 91:6080; (b) Schwenke DW, Haug K, Zhao M, Truhlar DG, Sun Y, Zhang JZH, Kouri DJ (1988) J Phys Chem 92:3202
4. Manolopoulos DE, Wyatt RE (1988) Chem Phys Lett 152:23
5. (a) Haug K, Schwenke DW, Truhlar DG, Zhang JZH, Kouri DJ (1986) J Phys Chem 90:6757; (b) Zhang JZH, KOuri, DJ, Haug K, Schwenke DW, Shima Y, Truhlar DG (1988) J Chem Phys 88:2492
6. Baer M, Shima Y (1987) Phys Rev A 35:5252; Baer M (1987) J Phys Chem 91:5846; Neuhauser D, Baer M (1988) J Chem Phys 88:2856; Baer M (1989) J Chem Phys 90:3043
7. Webster F, Light JC (1989) J Chem Phys 90:265; (1989) J Chem Phys 90:300
8. Kuppermann A, Hipes PG (1986) J Chem Phys 84:5692
9. Linderberg J, Vessal B (1987) Int J Quant Chem 31:65; Linderberg J, Padkjaer SB, Öhrn Y, Vessal B (1989) J Chem Phys 90:6254
10. Schatz GC (1988) Chem Phys Lett 150:92
11. Launay JM, Pepetit B (1988) Chem Phys Lett 144:346; Lepetit B, Launay JM (1988) Chem Phys Lett 151:287
12. (a) Walker RB, Hayes EF (1983) J Phys Chem 87:1255; (1984) 88:1194 (b) Walker RB, Hayes EF (1986) in: Clary DC (ed) The theory of chemical reaction dynamics. Reidel, New York, p 105
13. Light JC, Walker RB (1976) J Chem Phys 65:4272
14. Muckerman JT (1971) J Chem Phys 54:1155; (1972) 56:2997; (1972) 57:3388
15. Walker RB, QCPE Program No 352
16. Kress JD, Walker RB, Hayes EF (1990) J Chem Phys 93:8085
17. Darakjian Z, Hayes EF "Extension of the Pack–Parker quantum reactive scattering method to include direct calculation of time delays," accepted for publication in J Chem Phys
18. Johnson BR (1977) J Chem Phys 67:4086; (1978) 69:4678
19. Wu YM, Cuccaro SA, Hipes PG, Kuppermann A (1990) Chem Phys Lett 168:429
20. Light JC, Hamilton IP, Lill JV (1985) J Chem Phys 82:1400
21. Bačić Z, Kress JD, Parker GA, Pack RT (1990) J Chem Phys 92:2344
22. Whitenell RM, Light JC (1988) J Chem Phys 89:3674
23. Bačić Z, Light JC (1986) J Chem Phys 85:4594; (1989) Ann Rev Phys Chem 40:469
24. Smith TS (1960) Phys Rev 118:349
25. Parker GA, Butcher E, Hayes EF, Darakjian Z (1990) unpublished results