Theor Chim Acta (1991) 79: 225–239

Theoretica Chimica Acta

Quantum chemical reaction dynamics on a highly parallel supercomputer

Yi-Shuen Mark Wu*, Steven A. Cuccaro, Paul G. Hipes***, and Aron Kuppermann**

Arthur Amos Noyes Laboratory of Chemical Physics, Division of Chemistry and Chemical Engineering****, California Institute of Technology, Pasadena, CA 91125, USA

Received October 9, 1990/Accepted November 13, 1990

Summary. In this paper we describe the solution of the quantum mechanical equation for the scattering of an atom by a diatomic molecule on a high-performance distributed-memory parallel supercomputer, using the method of symmetrized hyperspherical coordinates and local hyperspherical surface functions. We first cast the problem in a format whose inherent parallelism can be exploited effectively. We next discuss the practical implementation of the parallel programs that were used to solve the problem. The benchmark results and timing obtained from the Caltech/JPL Mark IIIfp hypercube are competitive with the CRAY X-MP, CRAY 2 and CRAY Y-MP supercomputers. These results demonstrate that such highly parallel architectures permit quantum scattering calculations with high efficiency in parallel fashion and should allow us to study larger, more complicated chemical systems. Future extensions to this approach are discussed.

Key words: Reactive scattering – Hyperspherical coordinates – Parallel supercomputer

1. Introduction

Chemistry has long been one of the primary application areas for computers in scientific research. Quantum mechanical reactive scattering calculations, in particular, have consumed vast quantities of computer time on machines of all sizes. Accurate solutions have proved to be difficult and computationally expensive to obtain [1–4]. Such calculations would allow an interplay between theory and experiment which is vital to advance our understanding of the details of chemical reactions at the molecular level. Perhaps more importantly, the existence of accurate benchmark calculations permits the testing of approximate theories which in turn provides physical insights into the chemistry.

---

* Work performed in partial fulfillment of the requirements for the Ph.D. degree in Chemistry at the California Institute of Technology.

** *Current address*: IDA/SRC, 17100 Science Drive, Bowie, MD 20715, USA

*** *Current address*: 2338 Redwood Road, Scotch Plains, NJ 07076, USA

**** Contribution number 8209

The first calculations of accurate quantum mechanical cross sections were reported in 1975 by Schatz and Kuppermann [5, 6] and Elkowitz and Wyatt [7] for the simplest chemical reaction $H + H_2 \rightarrow H_2 + H$. After this, there was a lapse of over 10 years before these results were extended to higher energies and other systems. The problem is not only the inherent limitations in the theoretical methods but also the lack of sufficiently powerful computers [1–9]. Recently, a variety of efficient methodologies have been developed for carrying out calculations of reaction cross sections. With the current access to the CRAY-type supercomputers, there has been a remarkable surge in the number of publications in this field [10–21]. In particular, the use of symmetrized hyperspherical coordinates (SHC) [22, 23] and local hyperspherical surface functions (LHSF) [10, 17, 18] is a very promising approach [12, 24, 25]. However, even the fastest available supercomputers are not sufficiently fast to allow the study of chemical reactions involving more than three atoms. Mathematical modelling and understanding the chemistry involved have progressed to a point that only the lack of sufficient computing power is delaying detailed insight into the nature of many chemical reactions.

The initial route to supercomputing led by the CRAY machines is based on the construction of computers with very fast cycle times. Although this approach has produced very powerful machines, it is generally believed [26–28] that the key to future high performance computation to satisfy our need for both large numbers of CPU cycles and large amounts of fast memory is concurrent processing, or the use of several computers tied together through a very high speed network to solve a single problem. The algorithms used and the codes developed on sequential machines must be adapted to parallel computing. Hence, these new parallel algorithms, coupled with the capabilities of parallel supercomputers, permit theoretical studies of a wide variety of chemical reactions.

The considerations above provide motivation for investigating the use of highly parallel computers as a possible way to reduce the computational time for such calculations [29]. We chose the Caltech/JPL Mark IIIfp 64 processor hypercube [26–28], a distributed memory message passing parallel computer, as our test machine. The essential property a calculation must have to be efficiently done on a highly parallel computer is that it be decomposable in such a way that in performing it almost all processors should be computing efficiently almost all of the time, and that the communication time between the processors should represent a small fraction of the computation time. In this paper, we show how quantum mechanical reactive scattering calculations can be structured so as to fulfill these criteria. The performance of this implementation is also examined.

We divide this paper into four additional sections. In Sect. 2 we provide an overview of the methodology and computational requirements for calculating LHSF and for using Johnson's logarithmic derivative method [30, 31], modified to include the improvements suggested by Manolopoulos [32], for integrating the resulting coupled channel reactive scattering equations. In Sect. 3 the parallel algorithm is presented. In Sect. 4 benchmark results of scattering calculations for the $H + H_2$ system total angular momentum $J = 0, 1, 2$ partial waves on the LSTH [33, 34] potential energy surface are presented. We emphasize that even though the results we report were obtained for three identical particles, the implementation itself is applicable to general three body system in a parallel fashion. Ongoing and future extensions to this approach are also discussed. The last section contains some concluding remarks.

2. Quantum chemical dynamics

The goal of bimolecular quantum chemical dynamics is to calculate from first principles the reaction cross sections for an atom (or molecule) scattered by another molecule. Most chemical reactions take place as a result of interactions among three or four atoms. The only type of chemical reaction we are likely to be able to solve rigorously in the forseeable future is a three atom reaction of the type $A + BC \rightarrow AB + C$ or its four atom counterpart. Given the potential energy surface that governs an electronically adiabatic reaction, we use the nuclear motion Schrödinger equation to describe the collision of an atom and a diatomic molecule and the ensuing chemical reaction process.

The Schrödinger equation is a linear, second-order partial differential equation with $3N$ independent variables where N is the number of atoms in the system. One fruitful approach to solve this equation is based on hyperspherical coordinates [10, 11, 17, 18]. The detailed formulation of this approach is discussed elsewhere [10, 11, 17] and we will present a very brief review of the theory, listing the equations necessary to facilitate the explanation of the parallel algorithms.

For a triatomic system, we label the three atoms A_α, A_β and A_γ. Let (λ, ν, κ) be any cyclic permutation of the indices (α, β, γ). After removing the motion of the center of mass, we define as the λ coordinates the mass-scaled [35] internuclear vector $\boldsymbol{r}_\lambda$ from A_ν to A_κ, and the mass-scaled position vector $\boldsymbol{R}_\lambda$ of A_λ with respect to the center of mass of $A_\nu A_\kappa$ diatom. The symmetrized hyperspherical coordinates [22] are the hyperradius $\varrho = (R_\lambda^2 + r_\lambda^2)^{1/2}$, and a set of 5 angles ω_λ, γ_λ, θ_λ, ϕ_λ and ψ_λ, denoted collectively as ζ_λ. The first two of these are in the range 0 to π and are respectively 2 arctan r_λ/R_λ and the angle between $\boldsymbol{R}_\lambda$ and $\boldsymbol{r}_\lambda$. The angles θ_λ, ϕ_λ are the polar angles of $\boldsymbol{R}_\lambda$ in a space-fixed frame and ψ_λ is the tumbling angle of the $\boldsymbol{R}_\lambda$, $\boldsymbol{r}_\lambda$ half-plane around its edge $\boldsymbol{R}_\lambda$. The hamiltonian $\hat{H}_\lambda$ is the sum of a radial kinetic energy operator in ϱ, and the surface hamiltonian $\hat{h}_\lambda$, which contains all differential operators in ζ_λ and the electronically adiabatic potential energy function $V(\varrho, \omega_\lambda, \gamma_\lambda)$.

$$\hat{H}_\lambda = -\frac{\hbar^2}{2\mu}\left(\frac{\partial^2}{\partial\varrho^2} + \frac{5}{\varrho}\frac{\partial}{\partial\varrho}\right) + \hat{h}_\lambda \tag{1}$$

where

$$\hat{h}_\lambda = \frac{\hat{\Lambda}^2}{2\mu\varrho^2} + V(\varrho, \omega_\lambda, \gamma_\lambda) \tag{2}$$

and

$$\hat{\Lambda}^2 = -4\hbar^2\left(\frac{\partial^2}{\partial\omega_{\lambda_\lambda^2}} + 2\cot\omega_\lambda\frac{\partial}{\partial\omega_\lambda}\right) + \frac{\hat{j}_\lambda^2}{\sin^2\frac{\omega_\lambda}{2}} + \frac{\hat{l}_\lambda^2}{\cos^2\frac{\omega_\lambda}{2}} \tag{3}$$

$\hat{j}_\lambda$ is the angular momentum operator corresponding to $\boldsymbol{r}_\lambda$, $\hat{l}_\lambda$ is that corresponding to $\boldsymbol{R}_\lambda$ and $\mu = [m_\alpha m_\beta m_\gamma/(m_\alpha + m_\beta + m_\gamma)]^{1/2}$ is the reduced mass appropriate for the mass-scaled coordinates. $\hat{h}_\lambda$ depends on ϱ parametrically and is therefore the "frozen" hyperradius part of $\hat{H}_\lambda$.

The scattering wave function $\Psi^{JM\Pi\Gamma}$ is labelled by the total angular momentum J, its projection M on the laboratory-fixed Z axis, the inversion parity Π with respect to the center of mass of the system and the irreducible representation Γ of the permutation group of the system (P_3 for $H + H_2$) to which the

electronuclear wave function, excluding the nuclear spin part [36, 37], belongs. It can be expanded in terms of the LHSF $\Phi^{JM\Pi\Gamma}$, defined below, and calculated at the values $\bar{\varrho}_q$ of ϱ:

$$\Psi_i^{JM\Pi\Gamma}(\varrho, \zeta_\lambda) = \sum_n b_{ni}^{J\Pi\Gamma}(\varrho; \bar{\varrho}_q)\Phi_n^{JM\Pi\Gamma}(\zeta_\lambda; \bar{\varrho}_q) \tag{4}$$

The equation that defines the LHSF $\Phi^{JM\Pi\Gamma}$ with associated eigenvalues $\varepsilon_n^{J\Pi\Gamma}$ is

$$\hat{h}_\lambda \Phi_n^{JM\Pi\Gamma}(\zeta_\lambda; \bar{\varrho}_q) = \varepsilon_n^{J\Pi\Gamma}(\bar{\varrho}_q)\Phi_n^{JM\Pi\Gamma}(\zeta_\lambda; \bar{\varrho}_q) \tag{5}$$

The domain of the surface function equation is closed and the spectrum is real and discrete. The index i is introduced to permit consideration of a set of many linearly independent solutions of the Schrödinger equation corresponding to distinct initial conditions which are needed to obtain the appropriate scattering matrices.

The LHSF $\Phi_n^{JM\Pi\Gamma}(\zeta_\lambda; \bar{\varrho}_q)$ and associated energies $\varepsilon_n^{J\Pi\Gamma}(\bar{\varrho}_q)$ are obtained by using a Rayleigh–Ritz variational approach [17]. The key to the success of the variational approach is finding a set of functions which are numerically inexpensive to calculate and also embody some of the structure of the true surface function. One effective set of functions consists of products of Wigner rotation matrices $D^J_{M\Omega}(\phi_\lambda, \theta_\lambda, \psi_\lambda)$, associated Legendre functions of γ_λ and functions of ω_λ which depend parametrically on $\bar{\varrho}_q$ and are obtained from the numerical solution of one-dimensional eigenvalue-eigenfunction differential equations in ω_λ involving a potential related to $V(\bar{\varrho}, \omega_\lambda, \gamma_\lambda)$.

The variational method leads to an eigenvalue problem with coefficient and overlap matrices $h^{J\Pi\Gamma}(\bar{\varrho}_q)$ and $s^{J\Pi\Gamma}(\bar{\varrho}_q)$ and whose elements are 5-dimensional integrals involving the variational basis functions.

The coefficients $b_{ni}^{J\Pi\Gamma}(\varrho; \bar{\varrho}_q)$ defined by Eq. (4) satisfy a coupled set of second order differential equations involving an interaction matrix $\mathscr{I}^{J\Pi\Gamma}(\varrho; \bar{\varrho}_q)$ whose elements are defined by:

$$[\mathscr{I}^{J\Pi\Gamma}(\varrho; \bar{\varrho}_q)]_n^{n'} = \langle \Phi_n^{JM\Pi\Gamma}(\zeta_\lambda; \bar{\varrho}_q) | V(\varrho, \omega_\lambda, \gamma_\lambda) - (\bar{\varrho}_q/\varrho)^2 V(\bar{\varrho}_q, \omega_\lambda, \gamma_\lambda) | \Phi_{n'}^{JM\Pi\Gamma}(\zeta_\lambda; \bar{\varrho}_q) \rangle \tag{6}$$

The configuration space ϱ, ζ_λ is divided in a set of Q hyperspherical shells $\varrho_q \leqslant \varrho \leqslant \varrho_{q+1}$ $(q = 1, 2, \ldots, Q)$ within each of which we choose a value $\bar{\varrho}_q$ used in Eq. (4).

When changing from the LHSF set at $\bar{\varrho}_q$ to the one at $\bar{\varrho}_{q+1}$ neither $\Psi_i^{JM\Pi\Gamma}$ nor its derivative with respect to ϱ should change. This imposes continuity conditions on the $b_{ni}^{J\Pi\Gamma}$ and their ϱ-derivatives at $\varrho = \varrho_{q+1}$, involving the overlap matrix $\mathscr{O}^{J\Pi\Gamma}(\bar{\varrho}_{q+1}, \bar{\varrho}_q)$ between the LHSF evaluated at $\bar{\varrho}_q$ and $\bar{\varrho}_{q+1}$:

$$[\mathscr{O}^{J\Pi\Gamma}(\bar{\varrho}_{q+1}, \bar{\varrho}_q)]_n^{n'} = \langle \Phi_n^{JM\Pi\Gamma}(\zeta_\lambda; \bar{\varrho}_{q+1}) | \Phi_{n'}^{JM\Pi\Gamma}(\zeta_\lambda; \bar{\varrho}_q) \rangle \tag{7}$$

The 5-dimensional integrals required to evaluate the elements of $h^{J\Pi\Gamma}$, $s^{J\Pi\Gamma}$, $\mathscr{I}^{J\Pi\Gamma}$ and $\mathscr{O}^{J\Pi\Gamma}$ are performed analytically over the three Euler angles ϕ_λ, θ_λ and ψ_λ and by two-dimensional numerical quadratures over γ_λ and ω_λ. These numerical quadratures are the most expensive part of the entire LHSF computation and account for over 90% of the total time needed to calculate the $\Phi_n^{JM\Pi\Gamma}$ and the matrices $\mathscr{I}^{J\Pi\Gamma}$ and $\mathscr{O}^{J\Pi\Gamma}$.

The system of second-order ordinary differential equations in the $b_{ni}^{J\Pi\Gamma}$ is integrated as an initial value problem from small values of ϱ to large values using Manolopoulos' logarithmic derivative propagator [32]. Matrix inversions account for more than 90% of the time used by this propagator on sequential

machines. All aspects of the physics can be extracted from the solutions at large ϱ by a constant ϱ projection [10, 11, 38].

3. Parallel algorithm

One must first have an idea of what can be gained by parallel processing. Vast speedup can only be achieved for problems that can be grouped into concurrent cooperative subtasks. This, of course, involves an understanding of the level of parallelism that a problem manifests. Even with such understanding and an adequate mapping onto a system of cooperative processors, there still remains the critical issue of how to best implement processor coordination. Further, it is vitally important that the local data in processors be correct in a global sense, i.e., data modifications must be distributed across private memory boundaries. Since quantum reactive scattering calculations are well suited to multiprocessor systems, parallelism is achieved by data decomposition rather than by functional decomposition, i.e., by making all nodes execute the same code but with different data. In building a parallel implementation on the hypercube architecture, our main guidelines have been simplicity and utilization of as much of the original sequential code as possible.

The computer used for this work is a 64-processor Mark IIIfp hypercube. It consists of an ensemble of individual processing elements called nodes. The design of the Mark IIIfp hypercube permits as few as one and as many as 256 nodes in the ensemble. It is a leading design for MIMD-type (multiple instruction stream multiple data stream) distributed memory parallel architectures based on message passing [26–28]. Each node consists of two independent Motorola 68020 microprocessors, one for computation and one for I/O, and four megabytes of dynamic local memory with an access speed of 400 nanoseconds. The computation microprocessor has a Motorola 68882 floating-point arithmetic coprocessor, two serial ports, one printer port and 128 kilobytes of static private memory. The I/O microprocessor has 64 kilobytes of static private memory, one serial port and hardware to support the node to node communication within the hypercube. An additional daughter board with a pipe-lined 32-bit floating point unit based on the Weitek XL series of chips is attached to each node which further contains 128 kilobytes of code cache, 128 kilobytes of static memory and has a nominal peak speed of 16 Mflops. The Crystalline Operating System (CrOS)-channel-addressed synchronous communication provides the library routines to handle communications between nodes [28, 39, 40]. Program development is done on a Motorola 68020-based Counterpoint workstation that runs on UNIX. It acts as an access controller mechanism to the peripherals for the entire hypercube. This allows the native compilers and linkers of the control processor to be used to construct executable code to run on the nodes of the hypercube. The parallel programs consist of two complementary parts with one running on the control processor and one running on each hypercube node. It is written in C programming language except for the time-consuming two-dimensional quadratures and matrix inversions, which are optimized in Weitek XL assembly language.

The hypercube is configured as a two-dimensional array of processors. The mapping is done using binary Gray codes [28, 41] which gives the Cartesian coordinates in processor space and communication channel tags for a processor's nearest neighbors. With a distributed-memory machine like the hypercube, the elements of a large matrix of data must be distributed across the memory of all

the processors. This makes it possible to fully utilize the large memory available and if done properly facilitates the load-balancing task of keeping most of the processors busy doing useful arithmetic most of the time. The parallelization of scientific codes is frequently based on a large grain size decomposition of the task. To port a sequential code to a hypercube, a method of distributing the global matrix among the processors is the first choice that must be made and it is closely related to the parallel algorithm chosen.

We mapped the matrices into processor space by local decomposition. Let N_r and N_c be the number of processors in the rows and columns of the hypercube configuration, respectively. Element $A(i, j)$ of an $M \times M$ matrix is placed in processor row $P_r = int((i \times N_r)/M)$ and column $P_c = int((j \times N_c)/M)$, where *int* x means the integer part of x. This data decomposition has been found easy to maintain and has provided satisfactory load balancing; it has the further advantage that it does not require matrices of special dimensions.

The parallel code implemented on the hypercube consists of five major steps. Step one constructs, for each value of $\bar{\varrho}_q$, a primitive basis set composed of the product of Wigner rotation matrices, associated Legendre functions, and the numerical one-dimensional functions in ω_λ mentioned in Sect. 2 and obtained by solving the corresponding one-dimensional eigenvalue-eigenvector differential equation using a finite difference method. This requires that a subset of the eigenvalues and eigenvectors of a tridiagonal matrix be found.

A bisection method [42] which accomplishes the eigenvalue computation using the TRIDIB routine from EISPACK [43] was ported to the Mark IIIfp. This implementation of the bisection method allows computation of any number of consecutive eigenvalues specified by their indices. Eigenvectors are obtained using the EISPACK inverse iteration routine TINVIT with modified Gram–Schmidt orthogonalization. Each processor solves independent tridiagonal eigenproblems since the number of eigenvalues desired from each tridiagonal system is small but there are a large number of distinct tridiagonal systems. To achieve load balancing, we distributed subsets of the primitive functions among the processors in such a way that no processor computes greater than one eigenvalue and eigenvector more than any other. These large grain tasks are most easily implemented on MIMD machines; SIMD (single instruction stream multiple data stream) machines would require more extensive modifications and would be less efficient because of the sequential nature of currently known effective eigenvalue iteration procedures. The one-dimensional functions obtained are then broadcast to all the other nodes.

In step two a large number of two-dimensional quadratures involving the primitive basis functions which are needed for the variational procedure are evaluated. These quadratures are highly parallel procedures requiring no communication overhead once each processor has the necessary subset of functions. Each processor calculates a subset of integrals independently.

Step three assembles these integrals into the real symmetric dense matrices $s^{JII\Gamma}(\bar{\varrho}_q)$ and $h^{JII\Gamma}(\bar{\varrho}_q)$ which are distributed over processor space. The entire spectrum of eigenvalues and eigenvectors for the associated variational problem is sought. With the parallel implementation of the Householder method [44], this generalized eigensystem is tridiagonalized and the resulting single tridiagonal matrix is solved in each processor completely with the QR algorithm [45]. The QR implementation is purely sequential since each processor obtains the entire solution to the eigensystem. However, only different subsets of the solution are kept in different processors for the evaluation of the interaction and overlap

matrices in step four. This part of the algorithm is not time-consuming and the straightforward sequential approach was chosen. It has the further effect that the resulting solutions are fully distributed, so no communication is required.

Step four evaluates the two-dimensional quadratures needed for the interaction $\mathscr{I}^{JM\Gamma}(\varrho;\bar{\varrho}_q)$ and overlap $\mathscr{O}^{JM\Gamma}(\bar{\varrho}_{q+1};\bar{\varrho}_q)$ matrices. The same type of algorithms are used as were used in step two. By far, the most expensive part of the sequential version of the surface function calculation is the calculation of the large number of two-dimensional numerical integrals required by steps 2 and 4. These steps are however highly parallel and well suited for the hypercube.

Step five uses Manolopoulos' [32] algorithm to integrate the coupled linear ordinary differential equations. The parallel implementation of this algorithm is discussed elsewhere [30]. The algorithm is dominated by parallel Gauss–Jordan matrix inversion and is I/O intensive, requiring the input of one interaction matrix per integration step. To reduce the I/O overhead a second source of parallelism is exploited. The entire interaction matrix (at all ϱ) and overlap matrix (at all $\bar{\varrho}_q$) data sets are loaded across the processors and many collision energies are calculated simultaneously. This strategy works because the same set of data is used for each collision energy and because enough main memory is available. Calculation of scattering matrices from the final logarithmic derivative matrices is not computationally intensive, and is done sequentially.

The program steps were all run on the Weitek coprocessor which only supports 32-bit arithmetic. Experimentation has shown that this precision is sufficient for the work reported below. The 64-bit arithmetic hardware needed for larger calculations was installed after the present calculations were completed.

4. Results and discussion

Accuracy. Calculations were performed for the $H + H_2$ system on the LSTH surface [33, 34] for partial waves with total angular momentum $J = 0, 1, 2$ and energies up to 1.6 eV. Flux is conserved to better than 1% for $J = 0$, 2.3% for $J = 1$ and 3.6% for $J = 2$ for all open channels over the entire energy range considered.

To illustrate the accuracy of the 32-bit arithmetic calculations, the scattering results from the Mark IIIfp with 64 processors are compared with the results obtained using a CRAY X-MP/48 and a CRAY 2. The differences of the transition probability do not exceed 0.004 in absolute value over the energy range investigated.

Timing and parallel efficiency. In Tables 1 and 2 we present the timing data on the 64 processor Mark IIIfp, a CRAY X-MP/48 and a CRAY 2, for both the surface function code (including calculation of the overlap $\mathscr{O}^{JM\Gamma}$ and interaction $\mathscr{I}^{JM\Gamma}$ matrices) and the logarithmic derivative propagation code. For the surface function code, the speeds on the first two machines is about the same. The CRAY 2 is 1.43 times faster than the Mark IIIfp and 1.51 times faster than the CRAY X-MP/48 for this code. The reason is that this program is dominated by matrix-vector multiplications which are done in optimized assembly code in all 3 machines. For this particular operation the CRAY 2 is 2.03 times faster than the CRAY X-MP/48 whereas for more memory-intensive operations the CRAY 2 is slower than the CRAY X-MP/48 [46]. A slightly larger primitive basis set is

Table 1. Performance of the surface function code[a]

J	Mark IIIfp[b] 64 processors		CRAY X-MP/48		CRAY 2	
	Time (h)	Speed (Mflops)	Time (h)	Speed (Mflops)	Time (h)	Speed (Mflops)
0	0.71[c]	100[d]	0.74[e]	96[f]	0.49[g]	145[h]
1	2.88[i]	112[d]	3.04[j]	106[f]	2.01[k]	160[h]
2	5.60[l]	124[d]	5.94[m]	117[n]	3.96[o]	176[k]

[a] This code calculates the surface functions at the 51 values of $\bar{\varrho}$ from 2.0 bohr to 12.0 bohr in steps of 0.2 bohr, the corresponding overlap matrices between consecutive values of $\bar{\varrho}$ and the propagation matrices in ϱ steps of 0.1 bohr. The number of primitives used for each J and described in the remaining footnotes permits us to generate enough LHSF to achieve the accuracy described in the text

[b] 64 single precision processors

[c] For $80A_1$, $80A_2$ and $160E$ primitives. This basis is larger than the one described in [e] below and is needed to generate the same number of linearly independent surface functions as in [e]. The reason for this difference is the 32-bit arithmetic of the Mark IIIfp compared to the 64-bit arithmetic of the CRAY X-MP/48

[d] Estimated on the basis of the absolute measured speed on the CRAY X-MP/48 and the measured relative speeds of the Mark IIIfp with respect to the CRAY X-MP/48

[e] For $76A_1$, $76A_2$ and $152E$ primitives

[f] Measured using the hardware-performance monitor of the PERFMON and PERFPRT subroutines

[g] This time, for the same primitives as described in [e] was estimated on the basis of the relative speeds of the CRAY 2 and CRAY X-MP/48 measured for a set of 5 values of $\bar{\varrho}$. It is smaller than the time in [e] for the reason given in [h]

[h] Estimated on the basis of the relative speed of the CRAY 2 with respect to the CRAY X-MP/48 described in [g]. The reason this speed is 2/3 of the corresponding CRAY X-MP/48 speed is that the dominant parts of the calculation are optimized assembly code matrix-vector multiplications for which the CRAY 2 is 50% faster than the CRAY X-MP/48. Otherwise, the CRAY 2 is slightly slower than CRAY X-MP/48. See text

[i] For $72A_1$, $80A_2$ and $152E$ primitives of even parity and $152A_1$, $160A_2$ and $312E$ primitives of odd parity. These numbers of primitives are larger than the ones given in [j] for the reason given in [c]

[j] For $64A_1$, $76A_2$ and $140E$ primitives of even parity and $140A_1$, $152A_2$ and $292E$ primitives of odd parity

[k] Estimated on the basis of the relative speeds of the CRAY X-MP/48 and CRAY 2 and the measured CRAY X-MP/48 times or speeds

[l] For $216A_1$, $232A_2$ and $448E$ primitives of even parity and $136A_1$, $152A_2$ and $288E$ primitives of odd parity. These numbers are larger than those in [o] for the reason given in [c]

[m] This time is estimated as in [k], since the calculation cannot be done on the CRAY X-MP/48 because of insufficient memory

[n] Estimated to be the same as in [f], since the calculation cannot be done on the CRAY X-MP/48 for the reason given in [m]

[o] For $204A_1$, $216A_2$ and $420E$ primitives of even parity and $128A_1$, $140A_2$ and $268E$ primitives of odd parity

required on the Mark IIIfp in order to obtain surface function energies of an accuracy equivalent to that obtained with the CRAY machines. This is due to the lower accuracy of the 32-bit arithmetic of the former with respect to the 64-bit arithmetic of the latter.

The absolute times presented in Tables 1 and 2 are apt to decrease as the codes are improved and the numerical parameters are further tuned. As a result, they are not well suited for a comparison of the relative effectiveness of different

Table 2. Performance of the logarithmic derivative code[a]

	Mark IIIfp[b]		CRAY X-MP/48	CRAY 2
	64 processor global configuration[c]	8 clusters of 8 processors[d]		
Total time (h)	4.8[e]	3.4[f,g]	1.5	2.9[h]
Time for 1 energy (min)	2.2[i]	1.6[i]	0.7	1.3
Efficiency	0.52	0.81	—	—
Speed[j] (Mflops)	34.4[k]	48.5[k]	110	55.4

[a] Based on a calculation using 245 surface functions and 131 energies, and a logarithmic derivative integration step of 0.01 bohr

[b] 64 single precision processors

[c] The calculation for each energy was distributed among all 64 processors

[d] The hypercube was configured into 8 clusters of 8 processors each. Each cluster did full calculations for 16 energies, for a total of 128 energies. The times reported were multiplied by 131/128 for normalization purposes. All 8 clusters operated simultaneously

[e] This includes 1.9 hours of I/O time

[f] This includes 1.6 hours of I/O time. This time is shorter than that in [e] because of a different and more efficient broadcast of the data between the host and the 8 clusters

[g] Each cluster did full calculations for 16 energies for a total of 128 energies. The total time reported was obtained by subtracting the I/O time from the measured time, multiplying the result by 131/128 for normalization to 131 energies and adding the I/O time

[h] Estimated on the basis of the CRAY X-MP/48 times and the ratio of the speeds of the CRAY 2 and CRAY X-MP/48 for the logarithmic derivative code

[i] This includes the pro-rated I/O contribution

[j] All speeds include I/O contribution

[k] Estimated on the basis of the measured CRAY X-MP/48 speed for the logarithmic derivative code and the relative speeds of the Mark IIIfp and CRAY X-MP/48 for this code

reactive scattering methodologies [10–21]. The relevant information in those tables is, instead, the relative times among different machines as given by the corresponding speeds. These are indicative of the relative effectiveness of these machines for performing the reactive scattering calculations described in this paper.

The efficiency (ε) of the parallel LHSF code was determined using the definition $\varepsilon = T_1/(N \times T_N)$ where T_1 and T_N are respectively the execution times using a single processor and N processors. The single processor times are obtained from runs performed after removing the overhead of the parallel code, i.e., after removing the communication calls and some logical statements. Perfect efficiency ($\varepsilon = 1.0$) implies that the N-processor hypercube is N times faster than a single processor. In Fig. 1 efficiencies for the surface function code (including the calculation of the overlap and interaction matrices) as a function of the size of the primitive basis set are plotted for 2, 4, 8, 16, 32 and 64 processor configurations of the hypercube. The global dimensions of the matrices used are chosen to be integer multiples of the number of processor rows and columns in order to insure load balancing among the processors. Because of the limited size of a single processor memory, the efficiency determination is limited to 32 primitives. As shown in Fig. 1, the efficiencies increase monotonically and approach unity asymptotically as the size of the calculation increases. Converged results require large enough primitive basis sets so that the efficiency of the surface function code is estimated to be about 0.95 or greater.

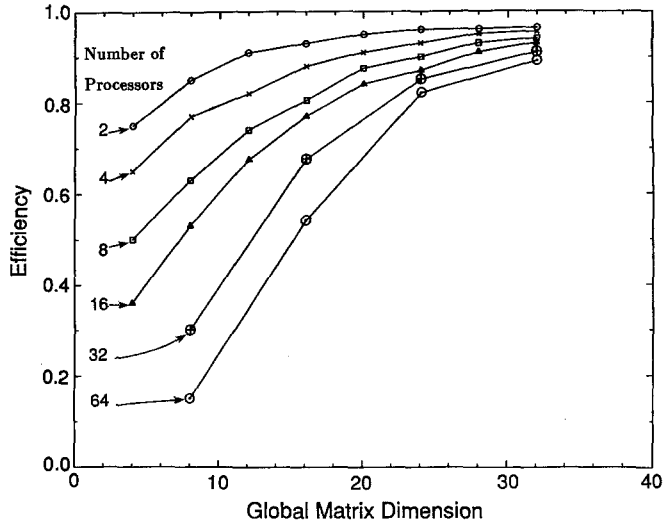

Fig. 1. Efficiency of the surface function code (including the calculation of the overlap and interaction matrices) as a function of the global matrix dimension (i.e., the size of the primitive basis set) for 2, 4, 8, 16, 32, and 64 processors. The solid curves are straight line segments connecting the data points for a fixed number of processors and are provided as an aid to examine the trends

The data for the logarithmic derivative code given in Table 2 for a 245 channel (i.e., LHSF) example show that the Mark IIIfp has a speed about 62% to that of the CRAY 2 but only about 31% of that of the CRAY X-MP/48. This code is dominated by matrix inversions, which are done with optimized assembly code in all three machines. The reason for the slowness of the hypercube with respect to the CRAYs is that the efficiency of the parallel logarithmic derivative code is 0.52 and by slow I/O to external storage devices. This relatively low value is due to the fact that matrix inversions require a significant amount of inter-processor communication. Figure 2 displays efficiencies of the logarithmic derivative code as a function of the number of channels propagated for different processor configurations, as done previously for the Mark III [30, 47] hypercubes. The data can be fit well by an operations count formula developed previously for the matrix inversion part of the code

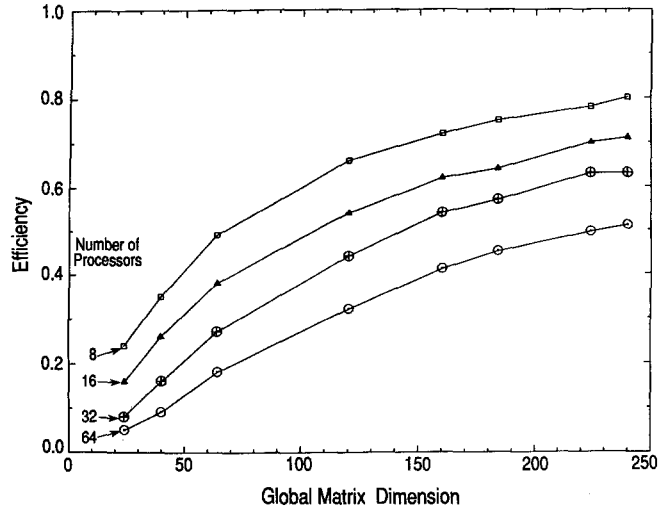

Fig. 2. Efficiency of logarithmic derivative code as a function of the global matrix dimension (i.e., the number of channels or LHSF) for 8, 16, 32, and 64 processors. The solid curves are straight line segments connecting the data points for a fixed number of processors and are provided as an aid to examine the trends

[48]; this formula can be used to extrapolate the data to larger numbers of processors or larger numbers of channels. It can be seen that for an 8 processor configuration, the code runs with an efficiency of 0.81. This observation suggested that we divide the Mark IIIfp into 8 clusters of 8 processors each and perform calculations for different energies in different clusters. The corresponding timing information is also given in Table 2. As can be seen from the last row of this table, the speed of the logarithmic derivative code using this configuration of the 64 processor Mark IIIfp is 48.5 Mflops, which is about 44% of that of the CRAY X-MP/48 and 88% of that of the CRAY 2. As the number of channels increases, the number of processors per cluster may be made larger in order to increase the amount of memory available in each cluster. The corresponding efficiency should continue to be adequate due to the larger matrix dimensions involved.

Ongoing and future work. From the previous discussions it appears that our application is well adapted to the hypercube architecture. However, our systems are experimental and continually evolving in terms of both hardware and software. In the near future, the number of processors of the Mark IIIfp will be increased to 128 and the I/O system will be replaced by high performance CIO (concurrent I/O) hardware. The new Weitek coprocessors, installed since the present calculations were done perform 64 bit floating point arithmetic at about the same nominal peak speed as the 32 bit boards. From the data in the present paper it is possible to predict with good reliability the performance of this upgraded version of the Mark IIIfp. Speed measurements on the CRAY Y-MP/864 of the San Diego Supercomputer Center show that it is 2 times faster than the CRAY X-MP/48 for the surface function code and 1.7 times faster for the logarithmic derivative code. In Table 3, we summarize the available or predicted speed information for the present codes for the current 64 processor and near future 128 processor Mark IIIfp as well as the CRAY X-MP/48, CRAY 2 and CRAY Y-MP/864 supercomputers. It can be seen that Mark IIIfp machines are competitive with all of the currently available CRAYs (operating as single processor machines).

Table 3. Overall speed of reactive scattering codes on several machines

	Mark IIIfp		CRAY X-MP/48	CRAY 2	CRAY Y-MP 864
	64 processor	128 processors			
Surface function code for $J = 2$ (Mflops)	124	240[a]	117[b]	176[b]	232[b]
Logarithmic derivative code[c] (Mflops)	48.5[d]	127[a,d,e]	110[b]	55.4[b]	187[b]
Total main memory of computer (64 bit Mwords)	32	64	8	256	64

[a] Estimated on the basis of the 64 processor performance

[b] For single processor operation

[c] For 245 channels. As the number of channels increases, the Mark IIIfp speed increases by a factor not exceeding 1.25, but the speed of the CRAY machines remains approximately constant

[d] Hypercube configured in clusters of 8 processors

[e] This speed assumes four-fold increase in the I/O data rate, compared to the 64 processor machine, due to concurrent I/O hardware

Today's supercomputers perform billions of arithmetic operations per second; by the mid 1990s, speed should be at least hundreds of times greater. In Table 4, we indicate the characteristics of some hypothetical future parallel machine characteristics including the upgraded Mark IIIfp hypercube just mentioned. Quantum chemical reaction dynamics computations to date have involved three atoms and strain the power of current supercomputers. The number N of coupled equations (or channels) which must be solved depends on the number of molecular ro-vibration states that are accessible, which in turn determines the order of the matrices that are manipulated. The matrix operations require a number of floating point operations of the order of N^3, and as a result the computational load increases as the cube of the number of channels. The memory requirements, on the other hand, increase with N^2. For class A type machines (see Table 4), of the order of 100 hours of CPU time on a single CPU of a CRAY Y-MP and 48 Mwords of memory are needed to calculate cross sections for the simple $H + H_2 \rightarrow H_2 + H$ reaction at 100 energies. Chemical reactions which are not thermoneutral and involve heavier atoms such as $O + H_2$ or $F + HD$ will require about two orders of magnitude more computing time and up to 2 Gigawords of memory, depending on the number of channels involved. Class B type machines will be needed for the study of such reaction. For three atom reactions on two electronically adiabatic surfaces or four atoms reactions like $H + H_2O$ or $H + COH$, class C Tflop machines will be needed. Such machines should be available in the next 5 to 10 years. They should permit the *ab inito* study of hundreds of bimolecular chemical reactions of importance for the understanding of combustion, plasmas, atmospheric chemistry and other complex systems of basic and technological interest. The experience gained in the use of class A and B machines should improve the design and facilitate the use of class C computers.

From Tables 1 to 3, we can find that the design details of different supercomputers make some better-suited for certain computations than others. For example, the surface function code is more efficient on the current Mark IIIfp 128 node hypercube while the logarithmic derivative code will run better on CRAY-type machines. Distributing large computations among several supercomputers will provide the opportunity both to bring to bear greater computing power than is available in any single machine and to use the most suitable

Table 4. Hypothetical future parallel supercomputer characteristics

	Class A Mark IIIfp	Class B (1991–1995)[a]	Class C (1996–2000)[a]
Sustained speed/node (Mflops)	2	20	200
Memory/node (Mwords)	0.5	4	32
Inter-node communication bandwidth (Mbyte/s)	1	100	1000
Number of nodes	128	1024	8192
Total sustained speed	256 Mflops	20 Gflops	1.6 Tflops
Total memory	64 Mword	4 Gword	262 Gword
Total I/O rate	128 Mbyte/s	10 Gbyte/s	1 Tbyte/s

[a] Time frame within which this machine class is expected to become available

machine for each step of the task. Currently, a high performance network is being developed to support host interfaces that operate at 800 million bits per second (Mbps) and that will connect multiple supercomputers at the Los Alamos National Laboratory, the California Institute of Technology, the Jet Propulsion Laboratory and the San Diego Supercomputer Center. With such a distributed heterogeneous computer, it should be possible for example to run a single program on the eight processors of the SDSC CRAY Y-MP/864 and the 128 processors of the Caltech Mark IIIfp hypercube at the same time with a total available memory of 128 Mwords. Quantum scattering calculations on larger, more complicated chemical systems would also then become feasible with heterogeneous computers of this type.

5. Conclusion

We have developed and implemented a strategy for performing quantum mechanical reactive scattering calculations on the Mark IIIfp hypercube parallel supercomputer. The results obtained for the $H + H_2$ system $J = 0$, 1, 2 partial waves agree well with those from a CRAY X-MP/48 and a CRAY 2. The high degree of parallelism of the most time-consuming step of the surface function calculation (the evaluation of two-dimensional numerical quadratures) leads to a high efficiency for that calculation. As a result, the speed of the 64 processor Mark IIIfp for the surface function calculation is about the same as that of the CRAY X-MP/48 and about 0.7 of that of the CRAY 2. When configuring the Mark IIIfp into 8 clusters of 8 processors each, the logarithmic derivative code is about 56% slower than the CRAY X-MP/48 and 12% slower than the CRAY 2. The speed of the 128 processor Mark IIIfp soon to become available should exceed, both for the surface function calculation and the logarithmic derivative calculation, that of the CRAY X-MP/48 and of the CRAY 2; however, although still comparable to the CRAY Y-MP/864 for the surface function code, it will be 32% slower for the logarithmic derivative code (the CRAYs operating as single processor machines). These results demonstrate the feasibility of performing reactive scattering calculations with high efficiency in parallel fashion. As the processors continue to become more powerful and their number continue to increase and with the help of high speed networks of the type currently being developed, such parallel calculations in systems of greater complexity will become practical in the not too distant future.

Acknowledgements. The work described in this paper was supported in part by DOE grant DE-AS03-83ER, by Air Force Astronautics Laboratory contract F04611-86-K-0067, and by funds from the corporation for National Research Initiatives. The calculations were performed on the 64 processor Mark IIIfp Caltech/JPL hypercube, the CRAY X-MP/48 at JPL, the CRAY X-MP/48 and CRAY Y-MP/864 at the NSF San Diego Supercomputing Center and the CRAY 2 at the Air Force Weapons Laboratory and we thank those institutions for their help. One of the authors (SAC) thanks NSF for a graduate fellowship. We also thank Dr. B. Lepetit and Prof. Geoffrey Fox for useful discussions.

References

1. Truhlar DG, Wyatt RE (1976) Ann Rev Phys Chem 27:1
2. Schatz GC (1986) in: Clary DC (ed) Theory of chemical reaction dynamics. Proc NATO workshop, Orsay, France, p 1

3. Garrett BC, Truhlar DG (1984) Ann Rev Phys Chem 35:159
4. Bowman JM (1985) Adv Chem Phys 61:115
5. Schatz GC, Kuppermann A (1975) J Chem Phys 62:2502
6. Schatz GC, Kuppermann A (1976) J Chem Phys 65:4642; 65:4668
7. Elkowitz AB, Wyatt RE (1975) J Chem Phys 62:2504, 63:702
8. Walker RB, Stechel EB, Light JC (1978) J Chem Phys 69:2922
9. Bernstein RB (ed) (1979) Atom-molecule collision theory. Plenum, New York
10. Kuppermann A, Hipes PG (1986) J Chem Phys 84:5962
11. Hipes PG, Kuppermann A (1987) Chem Phys Lett 133:1
12. Parker GA, Pack RT, Archer BJ, Walker RB (1987) Chem Phys Lett 137:564; Pack RT, Parker GA (1987) J Chem Phys 87:3888
13. Zhang JZH, Miller WH (1987) Chem Phys Lett 140:329; (1988) 153:465; (1989) 159:130
14. Schatz GC (1988) Chem Phys Lett 150:92
15. Haug K, Schwenke DW, Shima Y, Truhlar DG, Zhang J, Kouri DJ (1986) J Phys Chem 90:6757; Schwenke DW, Haug K, Truhlar DG, Sun Y, Zhang JZH, Kouri DJ (1987) J Phys Chem 91:6080; Zhang JZH, Kouri DJ, Haug K, Schwenke DW, Shima Y, Truhlar DG (1988) J Chem Phys 88:2492
16. Mladenovic M, Zhao M, Truhlar DG, Schwenke DW, Sun Y, Kouri DJ (1988) Chem Phys Lett 146:358; Yu CH, Kouri DJ, Zhao M, Truhlar DG (1989) Chem Phys Lett 157:491
17. Cuccaro SA, Hipes PG, Kuppermann A (1989) Chem Phys Lett 154:155
18. Cuccaro SA, Hipes PG, Kuppermann A (1989) Chem Phys Lett 157:440
19. Webster F, Light JC (1989) J Chem Phys 90:300
20. Linderberg J, Padkjaer S, Öhrn Y, Vessal B (1989) J Chem Phys 90:6254
21. Manolopoulos DE, Wyatt RE (1989) Chem Phys Lett 159:123
22. Kuppermann A (1975) Chem Phys Lett 32:374
23. Ling RT, Kuppermann A (1975) in: Rusley JS, Geballe R (eds) Electronic and atomic collisions. Abstracts of papers of the 9th Int Conf Physics of Electronic and Atomic Collisions, Seattle, Washington, 24–30 July 1975, Vol 1, Univ Washington Press, Seattle, pp 353, 354
24. Launay JM, Dourneuf ML (1989) Chem Phys Lett 163:178
25. Launay JM, Dourneuf ML (1990) Chem Phys Lett 169:473
26. Seitz CL, Matisoo J (1984) Phys Today 37(5):38; Seitz CL (1985) Comm of the ACM 28(1):22
27. Fox GC, Otto SW (1984) Phys Today 37(5):50
28. Fox GC, Johnson MA, Cyzenga GA, Otto SW, Salmon JK, Walker DW (1988) Solving problems in concurrent processors. Prentice Hall, New Jersey
29. Wu YM, Cuccaro SA, Hipes PG, Kuppermann A (1990) Chem Phys Lett 168:429
30. Hipes PG, Mattson T, Wu YM, Kuppermann A (1988) Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications, Pasadena. ACM, New York, pp 1051–1061
31. Johnson BR (1973) J Compl Phys 13:445; (1977) J Chem Phys 67:4086; (1979) NRCC Workshop, Lawrence Berkeley Laboratory, Report No LBL 9501
32. Manolopoulos DE (1986) J Chem Phys 85:6425
33. Liu B (1973) J Chem Phys 58:1925; Siegbahn P, Liu B (1973) J Chem Phys 58:1925; Siegbahn P, Liu B (1978) J Chem Phys 68:2457
34. Truhlar DG, Horowitz CJ (1978) J Chem Phys 68:2468; (1979) 71:1514 (E)
35. Delves LM (1959) Nucl Phys 9:391; (1960) 20:275
36. Lepetit B, Peng Z, Kuppermann A (1990) Chem Phys Lett 166:572
37. Lepetit B, Kuppermann A (1990) Chem Phys Lett 166:581
38. Hood DM, Kuppermann A (1986) in: Clary DC (ed) Theory of chemical reaction dynamics. Reidel, Dordrecht, pp 193–214
39. Fox GC (ed) (1985) Caltech JPL concurrent computation project Annual report 1983–1984
40. Fox GC, Lyzenga G, Rogstad D, Otto S (1985) The Caltech concurrent computation program – Project description, Proc. 1985 ASME Intl Computers in Engineering Conference
41. Gilbert EN (1958) Bell System Technical Journal 37:815; Salmon J (1984) Caltech Concurrent Computation Project Report C^3P-51

42. Ipsen ICF, Jessup ER (1987) Proc Second Conf Hypercube Multiprocessors, Knoxville, Tennessee; Ipsen ICF, Jessup ER (1987) Yale internal report: YALEU/DCS/RB-548; Fox GC (1984) Caltech Concurrent Computation Project Report C^3P-95
43. Smith BT (1976) Matrix Eigensystem Routine-EISPACK Guide, 2nd ed., Vol 6 of Lecture Notes in Computer Science, Springer-Verlag, New York
44. Fox GC (1984) Caltech Concurrent Computation Project Report C^3P-98; Patterson J (1986) Caltech Concurrent Computation Project Report C^3P-56.58
45. Wilkinson JH, Reinsch C (1971) Linear algebra, vol II of Handbook for Automatic Computation, Springer-Verlag, New York, pp 227–240
46. Pfeiffer W, Alagar A, Kamrath A, Leary RH, Rogers J (1988) Benchmarking and optimization of scientific codes on the CRAY X-MP, CRAY 2, and SCS-40 vector computers. San Diego Supercomputer Center Report GA-A19478
47. Messina P, Baillie CF, Felten EW, Hipes PG, Walker DW, Williams RD, Pfeiffer W, Alagar A, Kamrath A, Leary RH, Rogers J (1988) Benchmarking advanced architecture computers. Caltech Concurrent Computation Project Report C^3P-712
48. Hipes PG, Kuppermann A (1988) Gauss–Jordan inversion with pivoting on the Caltech Mark II Hypercube, in: Fox GC (ed) Proc Third Conf Hypercube Multiprocessors, Pasadena, CA, 19–20 January, Vol II – Applications. California Institute of Technology, Mail Stop 206-49, Pasadena, CA, pp 1621–1634