Theor Chim Acta (1991) 79: 175–182

Theoretica Chimica Acta

Massively parallel high-energy time-dependent wave-packet calculations

David Chasman[1], Robert J. Silbey[1], and Michael Eisenberg[2]

[1]Department of Chemistry and [2]Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 01239, USA

Received September 1, 1990/Accepted November 13, 1990

Summary. The exact solution of the time-dependent Schrödinger equation is obtained using a parallel implementation of the standard grid techniques. Most of the operations involved in this calculation may be executed concurrently for each of the grid points. For the few operations which may not be executed concurrently, we have implemented parallel algorithms. In our two-dimensional implementation on the Connection Machine, we have obtained optimal speed-up – that is, by using N processors we achieve a speed-up which is proportional to N. In addition to the discussion of our 2-dimensional implementation, we shall discuss our proposed 3-dimensional implementation of these grid techniques.

Key words: Wave packets – Quantum dynamics – Parallel computing

1. Introduction

The use of grid techniques for the solution of the time-dependent Schrödinger equation (TDSE) in 2 dimensions has become commonplace in the physical chemistry community. The usual alternative approach is the eigenvalue/eigenvector approach which requires a converged basis set calculation followed by the expansion of the initial wave-packet in the eigenstates of the Hamiltonian. Since the eigenvalue/eigenvector calculation scales as N^3 (N is the basis size) it is often prohibitive when a large basis set is necessary [1] as is the case when the problem being considered has a high density of states. It is in this case that grid techniques are preferred. Examples include vibrational dynamics on dissociative electronic surfaces [2] and high energy dynamics on bound surfaces [3]. Although grid techniques allow us to avoid the difficulty of a large matrix diagonalization, they are difficult to apply to large amplitude 3-dimensional problems because of their prohibitive running times on conventional computers. Fortunately, wave-packet propagation, using grid techniques, seems to have been custom tailored to the architecture of the Connection Machine (CM). In this paper, we present an optimal implementation of wave-packet propagation schemes – that is, by using N processors, our speed-up is proportional to N.

Although the switching devices in serial computers may become faster and faster, the speed of serial computers will ultimately be limited by the speed of switching devices. However, given the same switching devices, a massively parallel machine will always outperform the serial computer by a factor proportional to the number of processors – provided that efficient parallel algorithms are used. As we tackle larger and larger problems we have no choice but to implement our current algorithms in parallel and to discover new parallel algorithms.

2. The Connection Machine – a number cruncher's view

The Connection Machine (CM) may be viewed as an array of 8–64 K processors which concurrently execute commands which are issued by a front end computer. Each of these processors has 64, 128, or 256 K bits of memory and executes a normal sort of instruction set, albeit more slowly than high speed serial computers. In addition to the normal instruction set, there is facility for 12-dimensional inter-processor communication. There are 2 configurations for inter-processor communication – *news* (North East West South) and *send*. The *news* configuration is optimized for nearest-neighbor communication and the *send* configuration is optimized for processors which are 1, 2, 4, 8, 16 . . . 2^N distant from each other on the computational grid. What the current generation of Connection Machines may lack in processor speed is compensated for by its parallelism and inter-processor communication facilities.

3. Computational procedure

The standard wave-packet propagation techniques can be most easily viewed as two completely separate operations:

1. Evaluation of the Hamiltonian
   (a) Second-order differencing
   (b) Fourier transform
2. Expansion of the propagator
   (a) Second-order differencing
   (b) Chebeychev expansion

In any implementation of these techniques, the user selects one item from each section of this Chinese menu – repeating choices (1) and (2) until the solution at the desired time is obtained. Although any combination of selections may be used – those who savor precision usually select 1b and 2b – as this gives the most exact results [4].

Here, we review the algorithms which we use, compare the number of machine cycles required on a serial and parallel machines, and examine their suitability for parallelization. The most important consideration for the parallelization of any operation is whether it may be executed independently for each point on the grid. Any operation which may be executed independently for each grid point is trivially parallelizeable. Operations which cannot be executed independently for each grid point are not intrinsically parallelizeable and a non-trivial parallel algorithm must be used to obtain optimal speed-up. An

algorithm is optimal if the speed-up with N processors is directly proportional to N. In this disucssion, the calculations take place on an $N_g = N_a \times N_a$ grid. There are a total of $N_p - P \times P$ available processors.

3.1. *Evaluation of the Hamiltonian*

The Hamiltonian:

$$\hat{H}\psi(\boldsymbol{x}, t) = \left[\frac{-\hbar^2}{2m}\nabla^2 + V\right]\psi(\boldsymbol{x}, t) \tag{1}$$

is the sum of the kinetic and the potential energies. The potential energy term is evaluated by simple multiplication, that is:

$$V\psi(\boldsymbol{x}, t) = V(\boldsymbol{x}) \cdot \psi(\boldsymbol{x}, t) \tag{2}$$

This requires a single multiplication at each grid point. This operation may of course be executed independently for each grid point.

3.2. *Evaluation of the kinetic energy operator*

The kinetic energy operator may be evaluated by using either a finite difference scheme or the fourier transform method. The latter of these two techniques is preferred for two reasons. First, the commutation relations of quantum-mechanics are preserved in the grid representation. Second, this technique is exact for band limited functions with a finite number of components.

3.2.1. Evaluation of kinetic energy operator using finite difference scheme. The expression for the kinetic energy operator[1] using the standard finite difference scheme is:

$$f''(x_i) = \frac{\dfrac{\psi_i - \psi_{i-1}}{l} - \dfrac{\psi_{i+1} - \psi_i}{l}}{l} = \frac{2\psi_i - \psi_{i+1} - \psi_{i-1}}{l^2} \tag{3}$$

where l is the distance between grid points. This requires $6N_g$ operations – which of course depend on the neighboring grid points – this sort of communication is appropriate to the *news* CM configuration.

3.2.2. Evaluation of kinetic energy operator using the Fourier method. The preferred technique for evaluating the kinetic energy term is the Fourier transform technique. We recall that:

$$FT(f^{(n)}(x)) = (-i\boldsymbol{k})^n FT(f(x)) \tag{4}$$

where $FT(f(x))$ indicates the Fourier transform of $f(x)$. Thus, in order to evaluate the kinetic energy operator, we simply take the Fourier transform, multiply by $-\boldsymbol{k}^2$, and take the inverse Fourier transform. That is:

$$\nabla^2\psi(x) = FT^{-1}(-\boldsymbol{k}^2 FT(\psi(x)) \tag{5}$$

In order to speed this process, a Fast Fourier Transform (FFT) is used.

---

[1] This is for the 1-dimensional case – the extension to higher dimensions is clear

The Fourier transform:

$$\hat{z}_j = \sum_{n=0}^{N-1} \omega_N^{jn} z_n \qquad \text{where } \omega_n = e^{i2\pi/N} \tag{6}$$

may broken into two parts, and rearranged into an odd and even series [5]:

$$\hat{z}_{j'} = \begin{cases} \sum_{n=0}^{N/2-1} \omega_{N/2}^{j'}(z_n + z_{(n+N/2)}) & j - \text{even} \\ \sum_{n=0}^{N/2-1} \omega_{N/2}^{j'} \omega_N^{2j'+1}(z_n - z_{(n+N/2)}) & j - \text{odd} \end{cases} \tag{7}$$

which we recognize as two separate Fourier transforms – that of $(z_n + z_{(n+N/2)})$ and that of $\omega_N^{2j'+1}(z_n - z_{(n+N/2)})$ of size $N/2$. At the first step of the FFT, the indices are split by the value of their lowest order bit. At each successive step of the transform, we again split the indices depending on the next lowest order bit. This process has the effect of obtaining the output in "bit-reverse"[2] order relative to the input. The data-flow for this algorithm is diagrammed in Fig. 1 which corresponds to the familiar *butterfly*. When cells of the *butterfly* are re-used on successive levels – this network has the topology of a *Boolean n-cube* ($n = \log_2 N$). This is the topology of the CM [6].

It is clear that if we are doing an N point FFT and N is a power of 2 there will be $\log_2 N$ steps each of which involves 2 complex additions and $N/2$ complex multiplications. This gives us a total operation count of $5N \log_2 N$. Because the FFT may be balanced evenly between all N processors, the FFT requires only $5 \log_2 N$ cycles. Thus, the implementation of the FFT for powers of 2 is optimal. On the CM, all of the x-rows of a given axis may be transformed simultaneously. Thus, a 2-D FFT requires only $2(5 \log_2 N)$ cycles on the CM. For a more

[2] That is, the bits of the index in the output are the reverse of the bit ordering in the input. For example if we are computing FFT of an 8 point array the first (001 binary) element of the input will correspond to the fourth (100 binary) element of the output

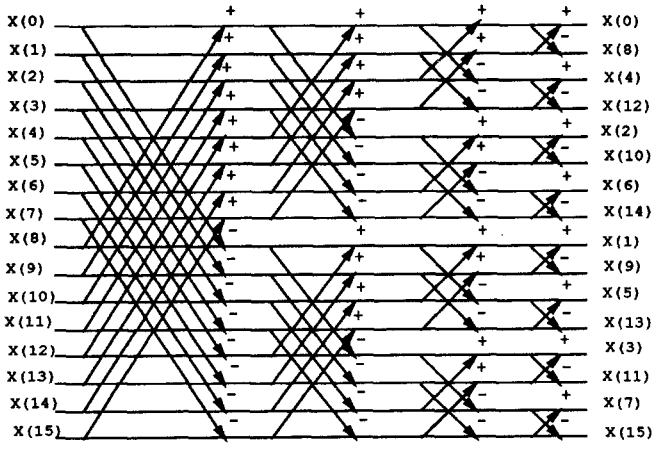

Fig. 1. Data flow for parallel FFT

detailed description of the implementation of the FFT on the CM, we refer the reader to Refs. [5] and [7].

3.3. *Integration in time*

3.3.1. Second-order differencing in time. This formula is most easily obtained as follows. First, we write the time derivative using the second-order difference formula:

$$\frac{\partial\psi}{\partial t} = \frac{\psi(t+\Delta t) - \psi(t-\Delta t)}{2\Delta t} \tag{8}$$

By rearranging this equation and substituting $-i\hat{\boldsymbol{H}}\psi/\hbar$ for $\partial\psi/\partial t$ we obtain:

$$\psi(t+\Delta t) \approx \psi(t-\Delta t) - \frac{2i\Delta t\hat{\boldsymbol{H}}\psi}{\hbar} \tag{9}$$

Advancing the time using this second-order differencing scheme requires $2N_g$ operations once $\hat{\boldsymbol{H}}\psi$ has been evaluated. These operations may be executed independently for each grid point.

3.3.2. Expansion of propagator in the Chebeychev polynomials. The integration in time is done by expanding the propagator $e^{-i\hat{\boldsymbol{H}}t}$ in the Chebeychev polynomials:

$$e^{-i\hat{\boldsymbol{H}}t} = \sum_{n=0}^{N_c} a_n\left[\frac{\Delta Et}{2\hbar}\right]\phi_n\left[\frac{-i\hat{\boldsymbol{H}}t}{R}\right] \tag{10}$$

where R is a constant which is determined from the potential of the Hamiltonian as well as the spacing of the grid points (and therefore the maximum wave-vector) being used for the integration and:

$$a_n[\alpha] = \int_{-i}^{i} dx \frac{e^{i\alpha x}\Phi_n(x)}{[1-x^2]^{1/2}} = 2J_n(\alpha) \tag{11}$$

The Chebeychev polynomials $\phi_n(\hat{\boldsymbol{X}})$ are related by the recurrence relation:

$$\phi_n(\hat{\boldsymbol{X}}) = 2\hat{\boldsymbol{X}}\phi_{n-1}(\hat{\boldsymbol{X}}) + \phi_{n-2}(\hat{\boldsymbol{X}})$$

Thus, in order to evaluate Eq. (10), we need only retain the value of the two previous Chebeychev polynomials at each grid point. This means that memory demand does not increase as a function of the number of terms in the Chebeychev expansion. This thrifty use of memory is important because the processors in the current generation of massively parallel computers have limited memory. The Chebeychev expansion of the propagator requires $2N_gN_c$ operations in addition to N_c Hamiltonian operations to evaluate the terms of Eq. (10) using Eq. (12). This technique of integration in time is very precise in comparison with other schemes[3] because the Chebeychev polynomials are exponentially convergent on any finite interval [4].

[3] For example, finite differencing in time.

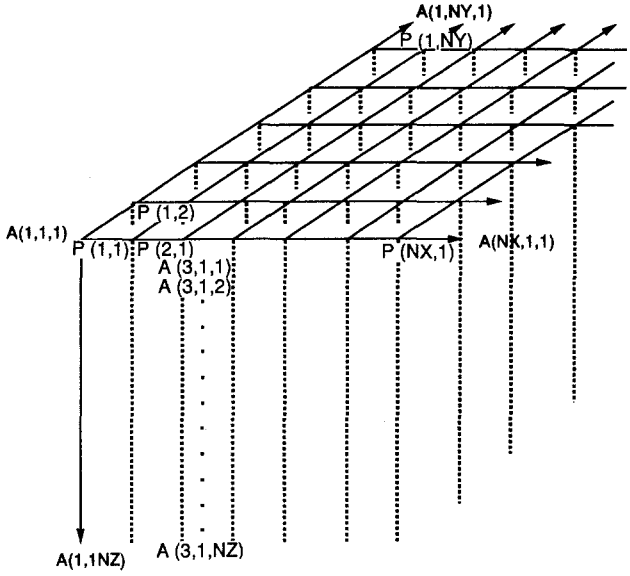

Fig. 2. CM layout for 3-D implementation. $P(i, j)$ Processor grid; $A(i, j, k)$ 3-D data array; NX, NY, $N2$ axis-dimensions. Off processor axis ———, on processor axis ·····

3.4. 3-D *implementation considerations*

Although there is little conceptual difference between the 2-D and the 3-D implementation of these techniques, there is a physical constraint of the current generation of CMs which is relevant. That is, there are at most $2^{16} = 65536$ processors in each machine. This means that if we wish to work on a $128 \times 128 \times 128 = 2^{21}$ grid – it is impossible to allocate one processor to each grid point. In our 3-D implementation, which is in progress, one dimension of the grid is on processor and the other two dimensions are formed by the processor grid – as illustrated in Fig. 2. In this figure, the X and Y axes form processor grid P_{ij} and the array elements A_{ijk} reside on the processor P_{ij} $\forall k$. Because there is not an array of processors along the serial Z-axis – it might seem that the kinetic energy calculation will scale as n (finite differences) or $N \log_2 N$ (FFT). However, since the calculations along the Z-axis are executed concurrently for all values of X and Y, we are again able to evenly distribute the Z-axis operations among all of the processors. In addition, all Z-axis operations are done on a part of the grid which is local to a single processor. This yields an additional speedup by avoiding inter-processor communication. Thus, not only are the 3-D implementations of these algorithms still optimal but our "lack" of processors is somewhat compensated for by this on-processor/off-processor speed trade-off.

4. Conclusions

In summary, almost every computational step of these grid algorithms may be executed concurrently for each grid site. Those steps which cannot be executed concurrently are implemented using optimal parallel algorithms. The speedup for massively parallel over serial is proportional to the number of processors for any

Table 1. Table of computational requirements: 2D implementation

Estimate of machine cycles used in the various propagation schemes – 2D			
Hamiltonian ↓	Time →	Second-Order Difference	Chebeychev expansion
Fourier transform	Serial	$5N_g + 20N_a^2 \log_2 N_a$	$5N_cN_g + 20N_cN_a^2 \log_2 N_a$
	Parallel	$5\frac{N_g}{N_p} + 20 \log_2 N_a$	$5N_c\frac{N_g}{N_P} + 20N_c \log_2 N_a$
Finite difference	Serial	$8N_g$	$8N_cN_g$
	Parallel	$8\frac{N_g}{N_p}$	$8N_c\frac{N_g}{N_p}$

N_a	Number of grid points per axis
$N_g\ (=N_a \times N_a)$	Number of total points on grid
P	Number of processors per axis
$N_p\ (=P \times P)$	Total number of processors
N_c	Number of terms in Chebeychev expansion

Table 2. Table of computational requirements: Projected 3D implementation

Estimate of machine cycles used in the various propagation schemes – 3D			
Hamiltonian ↓	Time →	Second-Order Difference	Chebeychev expansion
Fourier transform	Serial	$5N_g + 30N_a^3 \log_2 N_a$	$5N_cN_g + 30N_cN_a^3 \log_2 N_a$
	Parallel	$5\frac{N_g}{N_p} + 30N_a \log_2 N_a$	$5N_c\frac{N_g}{N_p} + 30N_cN_a \log_2 N_a$
Finite difference	Serial	$8N_g$	$8N_cN_g$
	Parallel	$8\frac{N_g}{N_p}$	$8N_c\frac{N_g}{N_p}$

N_a	Number of grid points per axis
$N_g\ (=N_a \times N_a)$	Number of total points on grid
P	Number of processors per axis
$N_p\ (=P \times P)$	Total number of processors
N_c	Number of terms in Chebeychev expansion

combination of techniques reviewed in this section. The comparison is summarized in Tables 1 and 2 for the 2-D and the 3-D case, respectively. To reiterate, although the solution of the 2-D TDSE has become routine, this work lays the groundwork for making the solution of the 3-D TDSE routine by using the massively parallel architectures as they emerge.

Acknowledgements. The authors wish to thank the NSF for partial support and Thinking Machines Corporation for computer time and expertise. In particular, we wish to thank Adam Greenberg, J. P. Massar, Alan Mainwaring, and Doug MacDonald of Thinking Machines for helpful discussions.

References

1. Kosloff R (1988) J Phys Chem 92(8):2087
2. Chasman D, Tannor DJ, Imre D (1988) J Chem Phys 89(11):6667
3. Chasman D, Silbey J, Eisenberg M (1990) Chem Phys Lett 175(6):633
4. Gerber RB, Kosloff R, Berman M (1986) Molecular scattering from surfaces. Computer Physics Reports 5(2):59
5. Johnsson SL, Krawitz L, Frye R, MacDonald D (1989) Cooley-Tukey FFT on the connection machine. Technical Report NA89-4 Thinking Machines Corporation
6. Hillis WD (1985) The connection machine. MIT Press, Cambridge, Massachusets, ACM Distinguished Dissertation.
7. Conte SD, deBoor C (1980) Elementary numerical analysis. McGraw-Hill, New York