JOURNAL OF COMPUTATIONAL PHYSICS 11, 449–454 (1973)

Note

Construction of the Hamiltonian Matrix in Large Configuration Interaction Calculations*

INTRODUCTION

Large-scale scientific calculation on today's most powerful computers makes balanced use of three types of computational capacity; (a) the speed of the central processing unit (CPU); (b) the capacity of peripheral storage which must carry large data sets, especially temporary data sets generated as intermediate results in the course of computation; and (c) the speed of data transfer between peripheral storage and the high speed storage connected to the CPU (channel speed). Characteristics (b) and (c) define the input–output (IO) capability of the computing system. It must be recognized that use of these three resources goes hand-in-hand and that much scientific computation makes severe demands on (IO) devices and channels. The old image of scientific computation as solely "number crunching" has been made completely invalid by increases in CPU power, which are inevitably associated with increased storage demands that require working with large peripheral data sets. A critical parameter, for example, in evaluating a computational algorithm involving the processing of elements stored in a peripheral data set, is the number of CPU cycles completed in the time that it takes one word to come through a data channel (after transmission through the channel has been initiated).

An important point is that balanced use of CPU power, peripheral storage and channel data rates, with current capabilities, have forced the use of *direct access* data sets to break bottlenecks in scientific computation which did not exist several years ago. Direct access data sets mean lists of data, any element of which can be accessed in a time characteristic of the direct access storage device (such as a magnetic drum or magnetic disk) and independent of the length of the data list. These are to be compared with sequential access data sets, in which an element can be accessed only after passing over all elements between the desired one and the one last accessed. Use of direct access storage makes possible the

* This research was supported in part by the Advanced Research Projects Agency of the Department of Defense under Contract No. DAHCO4 69 C 0080, monitored by U. S. Army Research Office, Box CM, Duke Station, Durham, 27706.

simultaneous processing of more than one long data list by making a *fixed* number of passes over the lists, minimizing the amount of data transmitted through the data channels, whereas sequential access to the same lists can require a number of passes proportional to the length of the lists of data. Minimizing IO may require sorting steps in which the output data list of a previous step is reordered for efficient use in a succeeding step. This would be the case, for example, when elements of an output list from a previous step, needed sequentially in a succeeding step, are scattered through the entire list.

This note will describe a computing algorithm for the construction of the Hamiltonian matrix in the space of a large number of n-electron configurations. This problem arises in performing configuration interaction (CI) calculations, which approximate the electronic states of a system of n-electrons as the solutions of the secular equation

$$(H - E)\, C = 0. \tag{1}$$

Elements of the Hamiltonian matrix H in Eq. (1) are

$$H_{IJ} = \int d\tau_1 \cdots d\tau_n \, \Phi_I^* H \Phi_J , \tag{2}$$

whereas in Eq. (2), Φ_I, Φ_J are n-electron configurations, and are members of an orthonormal set of n-particle functions, which are constructed from an orthonormal set of one particle functions (orbitals), with members ϕ_i determined in an earlier stage of computation. In terms of matrix elements over orbitals, we have that

$$H_{IJ} = \sum_{i,j} C_{IJ,ij}(i \mid j) + \sum_{i,j,k,l} C_{IJ,ijkl}(ij \mid kl). \tag{3}$$

In Eq. (3),

$$(i \mid j) = \int dv \, \phi_i^* \left(-\frac{1}{2} \nabla^2 - \sum_A Z_A / r_A \right) \phi_j \tag{4}$$

is a one-electron matrix element between orbitals of the Hamiltonian operator for one electron moving in the coulombic field of a set of nuclei A, with charge Z_A, where r_A is the distance of the electron from nucleus A. Integrals $(ij \mid kl)$ are given by

$$(ij \mid kl) = \int dv_1 \, dv_2 \, \phi_i^*(1) \, \phi_j(1) \, \phi_k^*(2) \, \phi_l(2)/r_{12} \tag{5}$$

and are two-electron matrix elements of the coulombic repulsion between pairs of electrons. These one- and two-electron matrix elements are available from an earlier stage of computation. The coefficients $C_{IJ,ij}$, $C_{IJ,ijkl}$ are also determined in an earlier computational stage; they depend on the form of configurations

I, J, which are linear combinations of Slater determinants having the symmetry of the electronic state under consideration, each with some specified angular momentum coupling of the electrons.

Thus, in the computational problem being addressed here, we are combining a coefficient list containing $C_{IJ,ij}$ and $C_{IJ,ijkl}$ and an integral list containing $(i \mid j)$ and $(ij \mid kl)$ according to Eq. (3). This process produces the list of matrix elements H_{IJ}. To put the magnitude of the problem into perspective, we are concerned with H matrices of dimension of the order of thousands (up to 10^4) having several million nonzero elements (the only ones stored) computed from integral lists and coefficient lists, each containing millions of elements. With high speed storage availability enough to store of the order of 30 000 elements in total, it is clear that all three lists under discussion are kept on peripheral storage.

ALGORITHM

A discussion of the computing algorithm will be simplified by changing the notation of the introduction, where the physical problem was described.

Thus, the coefficient list will be denoted $C_{P,Q}$ where the double index IJ, labeling a Hamiltonian matrix element H_{IJ}, has been contracted to the single index P. Since, in general, many matrix elements H_{IJ} are identically zero, only the nonzero elements of H_{IJ} will be stored. $P = 1, 2, 3,...$ corresponds to $(IJ)_1, (IJ)_2, (IJ)_3, ...$ where the index pairs $(IJ)_P$ correspond to nonzero elements of H_{IJ} in the desired order for storing H_{IJ}.

The integral list, available from a previous step and denoted X_Q, is ordered in such a way that the position Q in the list can be determined from the orbital indices labeling the integral.

Equation (3), the computational procedure to be carried out by this algorithm, can now be rewritten

$$H_P = \sum_Q C_{P,Q} X_Q . \tag{6}$$

In carrying out the summation, Eq. (6), we will put as big a block of consecutive members of the integral list X_Q into fast memory as possible, with substantially smaller storage requirements for other data. Suppose that N_{XC} is the size of this block, then the integral list will be divided into $[(N_X - 1)/N_{XC}] + 1$ blocks, all but the last block containing N_{XC} elements. Each block of integrals, serially numbered by index n, is processed completely by evaluating the contributions $H_{P,n}$ to H_P according to

$$H_{P,n} = \sum_{Q_n} C_{P,Q_n} X_{Q_n} , \tag{7}$$

where Q_n runs over indices Q in block n. In a later step, we sum contributions from the different blocks by

$$H_P = \sum_n H_{P,n} . \tag{8}$$

The summation of Eq. (7) followed by that of Eq. (8) is obviously equivalent to the desired result, Eq. (6). Remembering that the coefficient list is ordered to increasing P, it is clear that to complete the summation of Eq. (7) we need coefficients distributed through the entire length of the coefficient list. Since this must be done as many times as we have blocks, n, we would have to read the coefficient list a number of times proportional to the length of the integral list. This can be reduced to a single read if the coefficient list is first reordered so that all coefficients with Q values Q_n, belonging to the integral block n, are collected together; i.e., we reorder to C_{P,Q_n} ordering to increasing n, and for a given n retaining the original ordering of increasing P.

Thus, the implementation of Eq. (6) will be done as a three-step process:

(1) Reorder C_{PQ} to C_{P,Q_n};
(2) Carry out $H_{P,n} = \sum_{Q_n} C_{P,Q_n} X_{Q_n}$, Eq. (7);
(3) Carry out $H_P = \sum_n H_{P,n}$, Eq. (8).

These three steps will now be described in detail.

Step 1: *Reorder $C_{P,Q}$ to C_{P,Q_n}*

Elements of the coefficient list $C_{P,Q}$ are used sequentially, and can therefore be stored on magnetic tape.

Suppose the number of blocks into which the integral list is divided in carrying out Eq. (7) is N. Then, in this first step available fast storage is divided into N buffers, the nth of which will receive coefficients C_{P,Q_n} required for processing the nth block of integrals. $C_{P,Q}$ is now read sequentially, each element dropped into the appropriate buffer, and any time a buffer is filled it is written out as a record on direct access storage. Each record written out will contain the record number of the last record written out which was associated with the same n value; i.e., the records for a given n are backwards chained. An index of the last record written out for each n value will be retained in fast storage. Records belonging to a given n value will be randomly distributed through the direct access data set; since it is direct access, they can be efficiently retrieved. This is a key point, because in the next step, each $C_{P,Q}$ will be read only once; without the availability of direct access each element would be read N times, a cost which can be prohibitive.

The next step can be done using C_{P,Q_n} from the direct access storage. However, since the same CI calculation may be carried out many times, for different nuclear geometries for example, it is usually best to transfer the C_{P,Q_n} to tape, ordering

the records to increasing n. Thus, in the next step these can be used sequentially and the original list $C_{P,Q}$ is no longer needed. There is another advantage in writing the C_{P,Q_n} onto tape, and that is that it greatly decreases the requirement for direct access storage in a run. If a run involves Step 1 (reordering coefficients) and Step 3 [Eq. (8)], direct access storage requirements are dictated by the size of C_{P,Q_n}. If only Step 3 is involved, direct access requirements are for the much shorter list $H_{P,n}$. We will therefore assume the C_{P,Q_n} written onto tape.

The IO activity of Step 1 is one sequential read ($C_{P,Q}$ in), one direct access write with no random access (emptying buffers), one direct access read with random access (reading C_{P,Q_n} back in order of n), and one sequential write (C_{P,Q_n} out). We reiterate that Step 1 is done only once for a series of CI calculations using the same configurations.

Also, remember that the $C_{P,Q}$ were originally ordered to increasing P. The process just described will result, for a given n, in the record with highest P values being accessed first, but in the record the ordering to increasing P is retained.

Step 2: *Evaluate* $H_{P,n}$, *Eq.* (7)

Available fast storage will be largely taken up by the block X_{Q_n} of the integral list, around which this algorithm has been designed. Additional fast storage must be assigned for one record of C_{P,Q_n}, and a single $H_{P,n}$ buffer. Records of C_{P,Q_n} will be read in, one at a time, for the current n value. A buffer for $H_{P,n}$ will be filled from the end first to decreasing P, processing the C_{P,Q_n} from the end of a record to the beginning. $H_{P,n}$ buffers written into direct access are backward chained as in the first step. Analysis of the order will show that first $H_{P,n}$ available on reading will contain the lowest P values stored to increasing order.

The IO activity of Step 2 is two sequential reads (C_{P,Q_n} in, X_{Q_n} in), and one direct access write no random access ($H_{P,n}$ out).

Step 3: *Evaluate* H_P, *Eq.* (8)

In this step, available fast memory is mostly assigned to the largest block of the Hamiltonian matrix that can be accommodated. Additional fast memory must be assigned to one $H_{P,n}$ record. This block will receive elements from $H_{P_{\min}}$ to $H_{P_{\max}}$. One record at a time is read from the direct access $H_{P,n}$. For the first block of the H matrix we read the records for all n from the first available up to the one containing $H_{P_{\max},n}$ for this block. For the second block of the H matrix, the record which contained $H_{P_{\max},n}$ for the first block is the one containing $H_{P_{\min},n}$ for the second block so that reading of $H_{P,n}$ is resumed by rereading the last record needed for the first block. After each block of H is completed, it is written out into a sequential access data set. The process continues until the H matrix is completed.

The IO activity in this step is one direct access read with random access ($H_{P,n}$ in) and one sequential write (H_P out).

Extended Algorithm

For very large calculations, it may be the case that the direct access storage availability is not enough to accommodate the C_{P,Q_n} during the sorting stage of Step 1. A trivial modification of the algorithm will handle this case. In Step 1, run through as many $C_{P,Q}$ as can be accommodated in the available direct access storage, and write $C^{(1)}_{P,Q_n}$. Now continue processing $C_{P,Q}$ until direct access is refilled, and write $C^{(2)}_{P,Q_n}$, etc. Repeat steps 2 and 3 for each $C^{(m)}_{P,Q_n}$ where m indexes the number of writes in Step 1.

Discussion

The importance of this algorithm is that it minimizes the amount of IO in evaluating the Hamiltonian matrix. The number of words transmitted through the data channels is *independent* of the amount of fast storage available. The only dependence on fast storage capacity is the integral list block lengths, and buffer lengths associated with C_{P,Q_n}. Once the fast storage is large enough to accommodate buffer lengths to produce records that optimally conform to the characteristics of the direct access decive, there is no significant added efficiency from increased fast storage. Thus, fast storage requirements are modest.

These algorithms have been used extensively in the ALCHEMY computer programs[1] with up to 5000 configurations. We anticipate that calculations of this size, and larger, will be routine, particularly as efficient diagonalization procedures for the lowest roots of the Hamiltonian matrix are developed.

Acknowledgment

The author is indebted to A. D. McLean for his critical reading of the manuscript.

Received: May 1, 1972

M. Yoshimine

IBM San Jose Research Laboratory,
San Jose, California 95114

[1] An algorithm for determining these matrix elements, using similar sorting techniques has been devised by the author. A description may be found in A. D. McLean, "Potential Energy Surfaces from *ab initio* Computation: Current and Projected Capabilities of the ALCHEMY Computer Program," Proceedings of the Conference on Potential Energy Surfaces in Chemistry, Publication RA18, IBM Research Library, San Jose, CA, 1971.