OPTIMAL METHODS FOR LARGE-SCALE SCIENTIFIC DATABASE AND SPARSE MATRIX APPLICATIONS

P. Rochford, S. C. Park, and J.P. Drasner*
Department of Physics and Astronomy
Louisiana State University, Baton Rouge, LA 70803

and

S.-Q. Zheng
Department of Computer Science
Louisiana State University, Baton Rouge, LA 70803

ABSTRACT

Pursuing a many-nucleon description of collective rotations of nuclei demands that very large sparse matrix representations of hamiltonians be constructed and diagonalized. Two general computational methods are presented for reducing the computer resources required for these processes. Specifically, a space and time optimal numerical database called a Weighted Search Tree that minimizes the number of redundant intermediate calculations required when calculating the matrix elements of the sparse hamiltonian matrices is given, and a matrix multiplication algorithm for a new space optimal optimal representation for sparse matrices is given which is expected to dramatically decrease the cost of diagonalizing large sparse band matrices. Both methods can be implemented in a broad range of large-scale scientific applications.

INTRODUCTION

Because of its remarkable success at describing the various structural features observed for nuclei, the shell model is usually used when attempting to describe the bound states of a nucleus. The basic tenet of the model is that the nucleus is comprised of neutrons and protons that sit within a global mean-field potential which is generated from the average effect of their interactions with each other. The harmonic oscillator is the mean-field usually employed because of the very useful algebraic properties it possesses. Within this mean field potential the nucleons are considered to interact with each other through residual forces. The mean-field plus the sum of these residual interactions yields the many-body hamiltonian for the nuclear system, which is then evaluated within the hilbert space spanned by nucleons occupying states of the oscillator.

A research area which is currently very active within the nuclear physics community is that describing the collective rotational behaviour in nuclei, i.e. motion involving the rotation of the nucleus as a whole. It has been determined through various investigations that the residual interactions which are important for obtaining a good

* Supported in part by the U.S. National Science Foundation, Grant No. 89-22550.

© 1992 American Institute of Physics
Diagonalization of these matrix blocks requires that they be truncated, and it is found that for the lowest energy eigenstates of physical interest that they converge rapidly with increasing shell number, thereby allowing truncation to a finite number of the lower oscillator shells. While the single-particle splitting and pairing interactions in Eq. (1) have off-diagonal matrix elements which connect these blocks, the size of these matrix elements are much smaller than those contained in the blocks defined by $H'$. To a good approximation they can therefore either be ignored, or limited to those connecting only a few of the blocks, which then reduces the problem to constructing and diagonalizing a much smaller matrix. Shown in Table II are some typical dimensions for the $N_0(A_440)$ irrep that yields the lowest energy solution for $H'$ at a given $N$ oscillator shell truncation.

![Table II](image-url)

These matrix representations for the hamiltonian of Eq.(1) are also rather sparse, which is due to the symmetry basis employed being optimal with regard to describing collective rotational motion. For example a typical hamiltonian matrix is found to be only ~10% dense. This significantly reduces the amount of computer resources which are required for constructing and diagonalizing the matrices. However, the very large dimensions of these sparse matrices still result in considerable demands being made upon computer resources. If such calculations are to become feasible, then it is clear that computational techniques must be devised which more rapidly and efficiently solve this large sparse eigenmatrix problem.

In this manuscript, two techniques will be presented which help accomplish this goal. The first is a numerical database structure called a Weighted Search Tree (WST) that dramatically decreases the amount of computer resources required to construct the sparse matrices. The second is a new technique for sparse matrix multiplications which can be implemented to increase the efficiency with which sparse matrix diagonalizations are performed.

**WEIGHTED SEARCH TREE**

When constructing the matrix representation of $H$ in the $Sp(3,R) \supset U(3) \supset SU(3)$ hilbet space basis the hamiltonian is first expressed in terms of $U(3)$ tensor operators $T^{\alpha}_{\sigma}$. Where $\alpha$ denotes the remaining labels in the subgroup chain $SU(3) \supset SO(3) \supset SO(2)$. The hamiltonian matrix elements are then calculated by evaluating those for the tensors using the Wigner-Eckart theorem, which states that the matrix element can be expressed as

$$
\langle N_1(\lambda_1\mu_1)\lambda_1; T^{\alpha}_{\sigma}\rangle \langle N_2(\lambda_2\mu_2)\lambda_2; T^{\alpha}_{\sigma}\rangle = \sum_{\rho_1} \langle \lambda_1\mu_1; \lambda_2\mu_2; \alpha; \lambda_1; \lambda_2; \mu_1; \mu_2; \rho_1 \rangle \times \langle N_1(\lambda_1\mu_1) T^{\alpha}_{\sigma} N_2(\lambda_2\mu_2) \rangle_{\rho_1}
$$

i.e. a sum over the product of a $SU(3)$ coupling coefficient and a reduced matrix element, respectively. The coupling coefficient is determined by a well defined procedure and is interpreted as a geometrical factor associated with the group
Large-Scale Scientific Database

Symmetry chain SU(3) ⊃ SO(3) ⊃ SO(2). The reduced matrix element depends upon the specific physical form of the tensor operator and is interpreted as the quantity associated with the physical properties of the system. While this factorization means the determination of matrix elements can be separated into two parts, these separate quantities are each expensive to compute.

Since in the matrix construction of the Hamiltonian (1) for any given angular momentum J the same coupling coefficients and reduced matrix elements reoccur frequently, it is desirable to eliminate these intermediate results which would otherwise result in wasteful regeneration. The most efficient algorithm for minimizing the CPU time is one which stores the new intermediate quantities as they are computed, while also retrieving previously computed quantities by a binary search. Since computer memory limitations will usually preclude one from saving all the intermediate results which would be generated when constructing the large sparse matrices considered here, the most efficient numerical database is one which is also able to retain the most valuable information when subject to imposed memory space constraints. Such a data structure has been developed by the authors, and is called a Weighted Search Tree (WST).11,12

The WST is a numerical database which allows for the storage and retrieval of dynamically generated lists of length \( n \) in optimal time, i.e., \( O(\log n) \). When implemented, it allows one to search an already generated list for a particular element, and, if that element is not found, to also calculate the required result and add it to the list for possible reuse at a later stage of the calculation. Each list element that is stored must be uniquely identified by a key, which is usually characterized by the input variables used to determine its value, such as the quantum labels in the case of the coupling coefficients and reduced matrix elements considered here. It is the key which is actually searched for in the WST. Once a key is found the corresponding data entries associated with the key, which are also kept in the WST, provide information on the location where the previously calculated results can be found. The key and data elements do not have to be entered in any predefined order, as is maintained by pointers associated with each element in the WST. All keys contained in the numerical database are assigned a weight, or priority, which indicates its relative importance. These priority values provide the criteria by which it is determined whether or not new information is stored once the fixed size constraints of the database have been reached.

Successive calls to the WST package constructs a tree of the type shown in Figure 1, where the left-hand number within the circles is the key value and the right-hand number the priority. While the actual representation of the WST that has been coded as a set of FORTRAN subprograms is different from that shown in Figure 1,12 and is employed because it requires minimal memory space, the concept is essentially the same. The construction logic for the tree involves assigning elements to nodes (represented by the circles) in a linear array and using link information (the pointers and superscript integers) to specify the left and right subtrees of each node. The binary search times for all insert, search, and deletion operations are achieved by insuring the tree is height-balanced. For example, a binary tree is height balanced modulo one if the difference in maximum number of steps from root to apex between left and right subtrees for every node is less than or equal to one. The WST itself is a linear integer array, ID(-10:10). The first eleven elements, ID(-10) to ID(9), are reserved for specifying the structure of the tree and other link and status information. The rest of the array, starting with ID(1), is for the node elements. Each node contains four sets of information: a key that serves to identify the node, its priority, the linkage instructions, and the data. The array for the WST shown in Figure 1 is given in Figure 2 as an example. The columns from left to right indicate the nodes as they occur in the WST linear array, and the LCHILD, RCHILD, and PARENT rows are the linkage instructions. The data in this example is chosen to be the actual key values. The data could specify, as in the current case of calculating matrix elements, the position in an external real array where additional information about the node can be found, such as the values for the coupling coefficients.

The deletion of a node in the WST does not occur until the numerical database is full. Any new key which is generated is only inserted in place of the existing key if that particular data item has a higher priority value than the lowest priority node in the database. When this is the case the lowest priority node is deleted, the tree is rebalanced, and a node with the new element is then inserted. The actual deletion of the node is accomplished by exploiting the fact that the WST is a linear array, and rearranging the nodes in what is referred to as a heap structure13 such that the lowest priority one starts at ID(1). This is seen to be the case for the example in Figure 2, as the leftmost column is the one with the lowest priority.

Figure 1. A weighted search tree.

| PRIORITY | 1 2 3 4 5 6 7 8 9 10 11 |
| KEY     | 11 8 9 6 12 2 3 4 7 5 10 |
| LCHILD  | 0 10 0 0 0 12 6 0 4 7 3 |
| RCHILD  | 5 11 0 0 0 0 8 0 0 9 1 |
| PARENT  | 11 0 11 9 1 7 10 7 10 2 1 |

Figure 2. Linear array representation of above weighted search tree.

It is important to point out that the priority assigned to data items in the WST can incorporate the frequency of use of an item as well as its intrinsic value. The reason is that while the generation times of two data items may be quite different, this difference may be offset by use frequency. For example, if result A takes ten times longer to
node and rebalance the WST so that a node containing the new key can be added. The insert option is then called as indicated above for the case when the WST was not full.

This application of the WST as given up to now works well in the situation where there is one computed quantity associated with each node in the WST, such as one reduced matrix element (Eq. (4)) in an external real array. However, this is not the most efficient method for using the WST, as frequently a subset of the generated list of intermediate results can be identified by a single key. For example, the reduced matrix elements occurring in the matrix element calculation of Eq. (3) may have $p_1, \ldots, p_{\text{max}}$ and can be identified by the $N(\Lambda \mu)$ irreps labels. The more efficient approach in such a case is to assign only a single node in the WST for this subset, and store the starting and ending positions (or the starting position and number of elements) where this information is contained in the external array (henceforth called the buffer) as its data items, thereby saving memory space and reducing the number of nodes to be searched. The WST can easily accommodate this as it has the flexibility to contain any number of data items per node as specified by the user.

While this straightforward extension is fine when constructing the WST, problems occur once either the WST or buffer reach their storage limitations and a deletion operation is required. The reason is bookkeeping information must be retained that indicates where and how much free space is available in the buffer. It must not only keep account of space which is free at several points throughout the buffer, as more than one node may have to be deleted from the WST before enough buffer space is available to accommodate the higher priority data, but also contain information as to how to retrieve it when the intermediate results are required for reuse later on. For example, a new set of reduced matrix elements having $p_{\text{max}}=5$ may be generated having a higher priority than two sets with $p_{\text{max}}=3$ and $p_{\text{max}}=4$ that occupy different and nonsequential locations in the buffer. While some of the elements of this new set may occupy space vacated by one set, it is the remainder that must occupy the other. By this set of five reduced matrix elements will be located in at least two different parts of the buffer, extra information must exist indicating which locations they occupy. The problem of keeping track of this information is exactly what is involved in a file directory management system. Incorporated within the WST routines is therefore a file directory system which is in the form of an array containing pointer information for the location of the data in the buffer.

With this directory system, the user stores as the data for the WST node the starting point and number of locations occupied in the index array, which in turn contains the pointers for the locations in the buffer where the intermediate results are stored. This integer array INDEX(2, *) is dimensioned the same size as the buffer array plus three extra positions, where the first three elements INDEX(2) to INDEX(0) contain the information on the available free space. On a successful search for a given key, the information in the buffer is retrieved by using the values of INDEX(0) as the locations in the buffer array. When inserting and deleting information the logic is almost identical to that described above for the WST. The exceptions are that an additional check must now be made to determine whether there is enough storage in the buffer for the new results. This information is kept in INDEX(3,0) along with pointers that give the starting point location of the free space and number of positions available. The deletion option automatically updates INDEX for the new free space available in the buffer.

Using this system, the specific coupling coefficients and reduced matrix elements needed in the construction of the sparse matrices of interest here are calculated only once, while unneeded values are never generated. Moreover, under a fixed memory size constraint, those values which are the most expensive to generate are retained in favor of the least expensive. This means the required matrix representations can be generated such that the cpu time is minimized, as the non-zero matrix elements are
SPARSE MATRIX MULTIPLICATIONS

Once the sparse matrix representations of the Hamiltonian given by Eq. (1) have been constructed, one is confronted with the task of diagonalizing them in as efficient a manner as possible in order to obtain the energy eigenvalues and eigenvector wavefunctions. The best approach is clearly to exploit their sparse nature and employ a method where the memory storage required for the matrices is minimal and the calculations involving the zero matrix elements are avoided. All standard diagonalization algorithms for real symmetric matrices of a general form, be they either of modest or large size, reduce the full matrix to tridiagonal form as an intermediate step. When all eigenvalues and eigenvectors are desired the reduction from full to tridiagonal form is accomplished by a series of similarity transformations using reflector matrices, which is very costly when dealing with very large matrices. When only a few of the extremal eigenvalues and their eigenvectors are desired, as in the present case, this reduction is achieved at much less cost by using the Lanczos method which constructs a guess of the tridiagonal matrix given a trial starting vector. Once the tridiagonal form is achieved the QR and AR algorithms with quotient shifts and plane Jacobi rotations are employed to complete the diagonalization process. In both cases, matrix-matrix and matrix-vector multiplications are performed that involve the original matrix and its descendants that arise during execution of the algorithm. If the original matrix is sparse, then in both cases it is reasonable to expect that the successive operations will involve matrices which are sparse. A means of performing these calculations more rapidly is to therefore devise and implement a more efficient algorithm for sparse matrix-matrix and matrix-vector multiplications.

Such an algorithm has been developed by the authors and tested for matrices of a general band form, since many Hamiltonians of physical interest such as that in Eq. (1) possess matrix representations of this form. While there are specialized algorithms for the diagonalization of dense band matrices, these are only found to be faster than the standard algorithms for those dense matrices where the bandwidth L (maximum number of non-zero elements in a row or column of the band) is less than one third the dimension of the matrix L ≤ D/3). For larger bandwidths the standard approach is found to be necessary.

In the results presented below it will become apparent that the sparse matrix multiplication algorithm introduced here outperforms the standard approaches even in the situation of a fully dense matrix, thereby underscoring the practicality of this algorithm when diagonalizing matrices of any bandwidth.

First, it is obvious that a two-dimensional array representation for sparse matrices not only wastes space but also prevents matrix operations from being performed in optimal time because of the additional time that are performed on the null elements. A standard representation for a sparse matrix which enables it to be stored as a linear list, is one where each non-zero element is represented by a 3-tuple of the form (i, j, v), where i and j are the row and column numbers of the element, respectively, and v is the value of the element. For example, a sparse matrix such as that shown in Figure 4 would be stored in memory in row-major order as shown in Figure 5. That is, the nonzero elements are in increasing order of their row numbers, and all non-zero elements with the same row number are in increasing order of their columns numbers.

Here each column contains information on a nonzero element of the matrix A. For columns with k > 0 the three rows contain the row and column index i and j of the nonzero element matrix Ajk, and its value, respectively. The 0 column contains the global information of the matrix A, where the first two rows contain the number of rows and columns, respectively, and the third row the number I of non-zero elements in the matrix A.

While faster matrix operations are achieved with this representation, it is easy to see that this data structure is not the most space efficient for a sparse matrix that contains adjacent non-zero elements in its rows, as most of the row and column numbers in a segment are redundant. Consider as an example the calculation of Cy via C=AxB. If one of A or B is a zero element and the other is a non-zero element, say Ay=0 and Bx=0, then Bx is checked even though the multiplication operation is not performed. Clearly, it would be preferable to add additional information to the data structure for the matrix that allows for skipping those elements which do not contribute to the final result of the matrix operation. This is the idea behind the data structure designed here.

A maximal nonzero segment (or simply segment) in row i of an m x n matrix A is defined here as a closed interval [i, k] such that all elements Ajf ≠ 0 for j ≤ i, i ≤ k, Ajh = 0 if j > i and Ajh,k+1 = 0 if k ≤ m. This representation for A consists of two arrays, EA(1:SA), and SA(0:SA, 1:3), where SA is the number of nonzero elements in A and SA is the number of segments in A in row-major order. Clearly, each segment corresponds to a subarray of EA and all segments of A are in row major order in EA. SA contains SA+1 entries, where each entry is a 3-tuple (SA(r,1),SA(r, 2),SA(r, 3)) that contains information of a segment for r > 0. SA(0), and SA(0, 2) are the number of rows and columns of A, respectively, whereas SA(0, 3) contains the number of segments of A. If the corresponding segment [i, k] is in row i, then SA(r, 1) = i, SA(r, 2) = k, and SA(r, 3) = k. Since non-zero elements and segments of A are arranged in EA in a unique order, the indices of non-zero elements to be used can be calculated by a linear scan of SA. Shown in Figure 6 is an example of this representation for the matrix A of Figure 4.
Large-Scale Scientific Database

<table>
<thead>
<tr>
<th>EA:</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>8</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 6.** New Representation of a sparse matrix.

It can be shown that this representation of the sparse matrix occupies less memory space when \( t_{EA} > 1.5 t_{SA} \), i.e., when each segment contains more than 1.5 elements on average.\(^{17}\) This condition is one which is frequently satisfied in most scientific applications, and is found to be true for the hamiltonian matrices of interest to the authors.

Consider now the explicit multiplication of a sparse \( m \times n \) matrix \( A \) with a sparse \( n \times p \) matrix \( B \) to obtain a \( m \times p \) matrix \( C \). This operation involves multiplying the rows of \( A \) by the columns of \( B \), or equivalently, by the rows of the transpose of \( B \) (\( B^T \)). In terms of the new matrix representation introduced here, matrix multiplication therefore requires first transforming the representation of \( B \) into its transpose. This can be accomplished in a standard manner by implementing the bucket sort algorithm.\(^{18}\) Once this has been carried out, the problem is reduced to evaluating the dot products of the vectors corresponding to the rows of \( A \) with those of the rows of \( B^T \), which in turn involves evaluating the overlaps of the segments contained in each of the rows. Performing the latter is relatively straightforward as there are only nine different kinds of segment overlap conditions, and these can be sorted into three groups as shown in Figure 7 below.

**Algorithm** *MATRIX_MULTIPLY* (EA, SA, EB, SB, EC, SC)

1. Transpose \( B \) to obtain \( EB^T \) and \( SB^T \);
2. Let \( a_1, a_2, \ldots, a_g \) be the row numbers of non-zero rows of \( A \) such that \( a_i \leq a_{i+1} \);
3. Let \( b_1, b_2, \ldots, b_h \) be the row numbers of non-zero rows of \( B^T \) such that \( b_j \leq b_{j+1} \);
4. For \( i = 1 \) to \( g \) do
   a. Scan through segments \( \{a_i, k_{a_i}^A\} \) in row \( a_i \);
   b. For \( j = 1 \) to \( h \) do
      i. Scan through segments \( \{b_j, k_{b_j}^B\} \) in row \( b_j \);
      ii. If \( \{a_i, k_{a_i}^A\} \cap \{b_j, k_{b_j}^B\} \neq \emptyset \) then compute the dot-product of sub-vectors of \( A \) and \( B \) defined by segment overlap;
   c. else
      i. null case;
   endfor
5. Store the result of \( C_{a_i b_j} \) in EC and update SC;
6. endfor

**end of MATRIX_MULTIPLY**

It is clear from the above algorithm that fewer comparisons will be required when performing matrix multiplications with this representation than the standard one of Figure 5. Moreover, this algorithm will require much less time than that employing the representation of Figure 4, as a much more efficient comparison operation is performed in the former instead of a multiplication or addition of two numbers as for the latter.

To test the efficiency of this algorithm, it was coded in FORTRAN and executed for the matrix multiplication of two identical matrices \( A \) and \( B \). Initially \( A \) and \( B \) are \( D \times D \) band matrices, where \( D = 300 \) and the band width \( L = 31 \) (31 nonzero elements in a row or column), with each row containing one segment. Test data was then generated from this initial distribution iteratively by breaking a segment into two segments (i.e. replacing a non-zero element with 0 in a segment of at least three non-zero elements) for both \( A \) and \( B \). The performance of the algorithm is shown in Figure 8, along with the same results obtained for the same data when using the standard multiplication algorithm and the conventional sparse matrix one.\(^{13}\) It is important to note that the distribution of segments in the test data for each of the cases of different numbers of segments was closely related to a normal distribution, and was therefore representative of the situation frequently found in actual scientific applications. From the figure it is evident that when each row contains only one segment, that the new algorithm is 7-8 times faster than the conventional one\(^{13}\) and 40 times faster than the standard algorithm. This performance only becomes poorer than the conventional one once there are more than 12 segments in each row (except the first 15 rows and the last 15 rows), that is 4 segments with one nonzero element and 8 segments with two elements. For many sparse matrices that occur in physical problems, it is therefore highly likely that this new matrix representation will out perform the conventional method with regard to execution speed in addition to requiring less space utilization.
SUMMARY

Pursuing a many-nucleon description of collective rotations of nuclei leads to the common problem of very large matrices which must be constructed and diagonalized. By exploiting group symmetries, the dimensions of the Hilbert space can be reduced to a level where the resulting matrices are of a size that they can be done with existing computers. Furthermore, these matrices have a sparsity on the order of 10-20%. The challenge for improving the practicality of such a description of collective nuclear rotations is to devise efficient methods by which these very large sparse matrices, which are typically of the order of 5000×5000 and greater, can be constructed and diagonalized. In response to this challenge, a numerical database called a Weighted Search Tree was developed by the authors to minimize the redundant intermediate calculations that occur in constructing the matrix representation of the many-body Hamiltonian when limited by fixed memory space constraints. This numerical database can be implemented in any scientific application to minimize the use of computer resources, and has been found to yield gains of at least a factor of two in typical applications. With regard to the diagonalization of these large sparse matrices, a new storage representation has been developed, along with a matrix multiplication algorithm, which holds the potential to significantly diminish the cost of storing and reducing large sparse band matrices to tridiagonal form (an important intermediate step in all diagonalization processes). With both this matrix multiplication algorithm and the Weighted Search Tree, the cost of performing the quantum physics computations required for the many-body description of collective nuclear rotations that are of interest to the authors have become much more feasible, and thereby allows the scope of applications that can be undertaken to be much broader.

REFERENCES