Data Structures for variational atomic structure software

Charlotte Fischer

May 5, 2019

1 Introduction

A translated version of GRASP2K has been developed that is capable of large MPI runs with CSF expansions in the millions. Unlike the earlier version, no error messages have been encountered generated by the RCI code but were not replicated on other systems, which is encouraging. In particular,

1. Modules replaced COMMON. The latter had many restrictions such as double precision variables should precede single precision which was impossible to adhere to through maintenance. In many instances, compiler would pad common but correctness was not guaranteed. This was a concern. Over time, many new COMMONS were created as software was modified. If a new argument was needed, it was easier to pass the variable through a new COMMON rather than modify the calling sequence of a procedure. In the new version, some COMMONS were merged into a single MODULE, but a similar structure was maintained.

2. In the new version, the data-type of all variables are declared along with an IMPLICIT NONE statement.

3. DO loops and logical structure were translated to FORTRAN 95 standards.

4. But most important, the Cray Pointers were replaced by standard pointers and allocatable arrays. Since Cray Pointers were not a standard feature of FORTRAN 77, this change was implemented manually. In order to be able to quickly detect error a memory-management system was devised so that users would be informed about the location where an error occurred. I many respects, this was the most important change in that the result was a program where array-bound checking can now be used as a debugging tool.

The above modifications make it much easier to maintain the software and make modifications as needed for research development.

However, the translated program is not a MODERN Fortran program in that the algorithms that are implemented are based on algorithms that involve vectors, matrices and lists or arrays. The structure of MODULES is far more powerful than that of COMMON and allows calculations to be described in a much simpler way. The organization of a program is more data oriented, where the data is part of the module as well as the procedures that manage and process the data. Such an approach is sometimes referred to as being "object oriented".

In this report, some objects for atomic structure calculations are proposed, and procedures outlined for developing modern codes. Information can be organized in many different ways. In Object Oriented programming information about objects should be grouped together something that can be done through the use of data structures. The examples introduced in the next section are conceptual and have not yet been tested.

2 Vectors and Arrays of Mathematics

Often mathematics deals entirely in terms of vectors and two-dimensional arrays of numerical values. Vectors are stored in adjacent memory locations with the memory required being related to the data-type of arithmetic numbers. The BLAS routines have been optimized for various operations:

1. Level 1 – operations linear in time often dealing with stride (nearest neighbors have stride = 1). Typical operation is *axpy* or $y \leftarrow \alpha x + y$.
2. Level 2 – operations quadratic in time involving matrix arrays. Typical operation $y \leftarrow \alpha Ax + \beta y$.
3. Level 3 – operations cubic in time involving matrix- matrix operations such as $C \leftarrow \alpha AB + \beta C$ where A,B,C are matrices. A general matrix-matrix multiplication routine is *gemm*. For optimum performance in a cache environment this process is optimized by blocks. A highly tuned implementation based on optimization ideas is part of the GotoBLAS and OpenBLAS.

In FORTRAN, matrices are stored contiguously by column. For large, symmetric, real matrices, only the upper values (including the diagonal) are stored contiguously by column. When the matrix is sparse only the non-zero values of the upper triangular matrix are stored contiguously along with an array that indicates the row associated with the non-zero value. Thus, a sparse matrix structure is beneficial only if zero values exceed the number of non-zero values.

Note that the upper triangular matrix stored by columns is the same as the lower triangular matrix stored by rows for a real, symmetric matrix.

Frequently used routines related to linear algebra are part of a Lapack Library with the ScaLapack library useful for MPI programs.

3 Atomic Physics data types

In atomic physics, an orbital has many properties. It is helpful to group this information together. This can be done through the definition of a data type, say `orb`.

```
TYPE  orb
   Character(LEN=3) :: el            ! symbol for the orbital such as '1s'
   Integer(kind=1)  :: n, l, jj      ! values of n,l, 2j and parity
END TYPE  Orb
 ...
 Type(orb), Dimension(1:nwf) :: my_orbs
```

defines a list of orbitals with 4 components where $my\_orbs(1)\%el$ is the symbol for the first orbital, $my\_orbs(1)\%n$ the n-value, etc. Note that packing of quantum numbers is avoided by representing values in single byte integers and the value of $jj = 2 * j$ is stored is stored. The integer(kind=1) stores integers with values up to 127. The parity of an orbital, if necessary, could be the "sign of $2j$". For relativistic calculations κ might also be a property.

The above defines orbitals for angular calculations but for the energy and other applications, orbitals also have radial functions associated with them. So we might need a "radial" type by "extending" the orbital definition. Thus the type "radial" inherits the properties of "orb".

```
TYPE,  EXTENDS(orb) :: radial
   integer(kind=4)                          :: npt
   REAL(kind=8), DIMENSION(:), ALLOCATABLE :: P, Q
END TYPE radial
 ...
 TYPE(radial), DIMENSION(1:nwf), ALLOCATABLE :: my_radial
```

Here I have defined the information associated with radial function as an array of radial data types. I suspect that it may be more efficient to have a data structure of arrays where the information in arrays are stored in contiguous locations. Here we need parameterized data types which is supported by gfortran 7.2 but not gfortran 4.8 .

```
MODULE rad_def
    IMPLICIT NONE
    TYPE :: orbs(nwf)
       INTEGER, LEN :: nwf
       CHARACTER(LEN=3), DIMENSION(1:nwf) :: el
       INTEGER(kind=1), DIMENSION(1:nwf) :: n,l, jj
    END TYPE orbs
    TYPE, EXTENDS(orbs) :: rads(nwf,nnnp)
        INTEGER(kind=4), LEN :: nwf, nnnp
        INTEGER(kind=4), DIMENSION(1:nwf) :: npt
        REAL(kind=8), DIMENSION(nnnp, nwf) :: P, Q
    END TYPE rads
END MODULE rad_def

Program  Main
    IMPLICIT NONE
    USE rad_def
    type(rads) :: radials
    Integer    :: nwf=10, nnnp=350

    allocate radials(nwf, nnnp)
    radials%npt(1)= 50.
End Program
```

The above defines sets of nwf orbitals. In the allocate for the arrays, the nnnw is used as a dimension for the radial function but the actual extend of a given orbital is npt. Because the grid is logarithmic, the variation in the number of grid points is unexpectedly small. In an $n = 7$ calculation for Pr+3, the grid points ranged from 297 for the $1s$ orbital to a maximum of 320 for a $6d$ correlation orbital. So the convenience of having all orbitals the same length comes at a relatively small cost but it still is useful to have the precise length because the calculation of matrix elements for any operator can stop when the orbital with the shortest range becomes zero.

When dimensions are known, memory can be allocated as needed, but in some instances, such as the generation of angular data, there is no simple way of predicting the memory required. In such instance, linked lists are an easy solution and have been used by C++ programmers for decades.

Figure 1 shows one of many versions of singly and doubly linked lists. A feature of such a structure is that the data portion may be any data, including user defined data types. In this figure, the singly linked list can only be accessed through a pointer to the "head" of the list, navigated from the head to the end of the list. Frequently, there is a second pointer to the last item in the list that is called the "tail". A structure with both "head" and "tail" can add new items to the list (at the end) but navigating from head to tail. A doubly linked list includes pointers for both the "next" item in the list and the "previous" item. Linked lists are convenient way of sorting data since new items can be inserted without moving any of the items, merely be redefining pointers.

As an example from atomic physics, let us consider the list of angular data for the non-zero matrix elements of an interaction matrix. Angular data consists of a list of

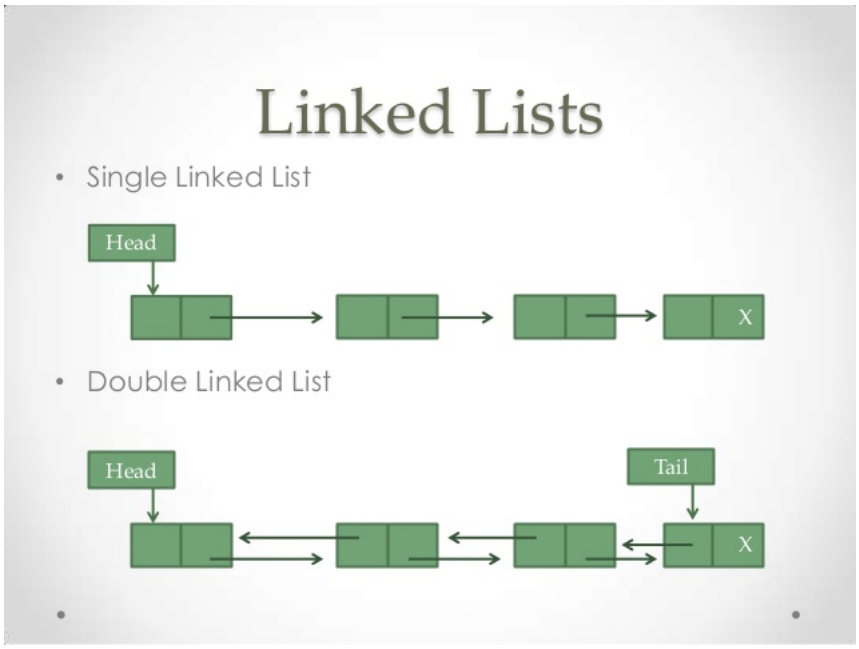

Figure 1:

coefficients (COEFF) and an index to a radial integral (or a pointer). The long list is divided into records of length say 20,000 (this could be a parameter). This record together with an integer N designating the actual number of entries, and a pointer that points to the next record defines a data type. The linked list consists of a pointer to the "head" of the list and another pointer to the "tail" of the list. The list may grow as long as needed without data being moved in memory. This is particularly useful when items include a lot of information or several long lines as for CSFs.

```
MODULE ang_def
   ! The size of a record is 20,000 data items.
   IMPLICIT  NONE
   TYPE ang_rec
      INTEGER :: N  ! The number of items in the rec
      REAL(kind=8),     dimension(1:20000) :: coeff
      INTEGER(kind=8),  dimension(1:20000) :: intgrl
      TYPE(ang_rec), POINTER :: next
   END TYPE ang_rec
END MODULE ang_def


MODULE ang_linked_list
   USE ang_def
   IMPLICIT NONE
CONTAINS
   SUBROUTINE init(head, tail)
      ! Dummy arguments
      TYPE(ang_rec), POINTER :: head, tail
      NULLIFY(head, tail)
```

```
   END SUBROUTINE init

   SUBROUTINE add(new, head, tail)
      !Add a new rec to the list
      !Dummy arguments
      TYPE(ang_rec), POINTER :: new, head, tail

      ! Check if the list is empty
      IF (ASSOCIATED(head)) THEN
         ! list is NOT empty‘
         tail%next =>new    ! (no need to allocate?)
         NULLIFY(new%next)
         tail => new
      ELSE
         ! list is empty setup the first rec
         head => new
         tail => new
         NULLIFY(tail%next)
      END IF
   END SUBROUTINE add

   SUBROUTINE write(head) ! output the contents of the list
      ! Dummy argument
      TYPE(ang_rec), POINTER, INTENT(in):: head
      ! Local variable
      TYPE(ang_rec), POINTER :: ptr
      INTEGER :: j
      !Check if list is empty
      IF (.NOT. ASSOCIATED(head)) THEN
         Print *, 'List is empty'
      ELSE
         ! Set local pointer to head
         ptr => head
         DO
            Do j = 1,ptr%N
              print *, ptr%coeff(j), ptr%intgrl(j)
            End Do
            ptr => ptr%next
            If (.NOT. Associated(ptr)) EXIT
         END DO
      END IF
   END SUBROUTINE
END MODULE ang_linked_list


Program generate_ang_data
      USE ang_linked_list
      IMPLICIT NONE

      ! generate the linked list
      TYPE(ang_rec), POINTER :: ang_head, ang_tail
      TYPE(ang_rec), POINTER :: my_data
```

```
      INTEGER :: i, j, nsize=20000, ierr, iend=10

      CALL init(ang_head, ang_tail)
      i=1       ! counter for angular data
      Do
         allocate(my_data, STAT=ierr)
         If (ierr .ne. 0) then
            print *, 'Something wrong with alloc'
            stop
         end if

         my_data% N = 0
         Do j = 1, nsize
            my_data % coeff(j) = i*1.d0
            my_data % intgrl(j) = i
            my_data % N = my_data % N + 1
            i=i+1
            if ( i .gt. iend) exit
         END DO
         print *, 'Calling add when i=',i
         call add(my_data, ang_head, ang_tail)
         if ( i .gt. iend) exit
      END DO

      print *, 'Finished generating data'
      call write(ang_head)
END program generate_ang_data
```

4 Data Structures for GRASP objects

4.1 Orbitals

Types orb and radial have been introduced in the previous section according to their quantum mechanical definition. But in calculations orbitals are also of different types -- closed shells, occupied, valence correlation, pair-correlation functions (PCF) as discussed in the report on wave function expansion. Further study is needed on how to define these different types but they are most likely indices. For example, the closed shells could be the first 10 orbitals of a variable n_core =10.

4.2 Radial integrals

Given the set of orbitals, a list of radial integrals can be generated for different types of Hamiltonians.

In ATSP the types were I, F, G, R. In GRASP only the RCI program used the list of integrals so that integrals need be evaluated only once and could be used many times. The size of each list could be determined before memory allocation was needed. Associated with the list of integrals is an array that may be an array of values (as need for determining the interaction matrix) or the coefficient of the integral in an energy expression when determining the potential and exchange functions for computing the radial functions. It is suggested that, for RMCDHF the classification be I, F, RK , where $k = 0, \ldots, kmax$. In other words, all the $F^k(a, b)$ integrals are in the same classification but the Slater

integrals $R^k(ab;cd)$ (excluding the F^k integrals) are classified according to k.

```
MODULE intgrl_def
    INTEGER :: kmax
    TYPE int_arg
      UNSIGNED(kind=1) :: a,b,c,d
    END Type int_arg
    TYPE int_list
      INTEGER :: n
      TYPE(int_arg), ALLOCATABLE, DIMENSION(1:n) :: arg   ! or should this be a packed value?
      REAL*8, ALLOCATABLE :: coeff(1:n)
      LOGICAL ALLOCATABLE :: used(1:n)
    END TYPE int_list

    TYPE(intgrl) :: T, F, Rk(0:kmax)
    CONTAINS
       Subroutine genint -- generates the list of integrals -- a,b,c,d values
                            determines size (n-values)
                            allocates
                            or computes values
                            or computes coefficients of the integral that depend
                               on expansion coefficients
       (could be three different routines, but 1 and 2 can be combined)
       END SUBROUTINE genint
END MODULE intgrl_def
```

Notice here that we have used the UNSIGNED(kind=1) data type for a,b,c,d which is a 1 byte integer with values $0-255$. There is a question as to whether ABCD should be packed or not. In searching the list we have some relationships: $a \le b; a \le c; a \le d; b \le d$. With this order it is possible to use a binary search on packed values. So there are times when it is desirable to search a packed UNSIGNED(kind=4) integer and others when the individual kind=1 values are needed. A packed value would be $ABCD = (((a* 2**8 + b) * 2**8 + c) * 2**8 + d$ each multiply being equivalent to a binary shift of 8 places. Maybe

```
ABCD = a
ABCD=LSHIFT(ABCD,8)+b
ABCD=LSHIFT(ABCD,8)+c
ABSD=LSHIFT(ABCD,8)+d
```

would be faster because the compiler would translate this into inline shift operations rather than multiply.

4.3 Wave function expansions

The basic object in a wave function expansion is a configuration and all the couplings for a given total J and parity. Here it is important to consider a large case, namely $4d^8 4f^4$ where the $J = 0$ is about six times shorter (with regard to the number of CSFs).
ut

```
Core subshells:
  1s   2s   2p-  2p   3s   3p-  3p
Peel subshells:
```

```
  4d-  4d   4f-  4f
CSF(s):
  4d-( 2)  4d ( 6)  4f ( 4)
        0                 0
                           0+
        2           2;    2
                           0+
        2           4;    2
                           0+
  4d-( 2)  4d ( 6)  4f-( 1)  4f ( 3)
        0              5/2      5/2
                          5/2      0+
        2              5/2      7/2
                          7/2      0+
        2              5/2      3/2
                          3/2      0+
        2              5/2      5/2
                          5/2      0+
        2              5/2      9/2
                          9/2      0+
  4d-( 2)  4d ( 6)  4f-( 2)  4f ( 2)
        0                 0        0
                                     0+
        0                 2        2
                             2      0+
        0                 4        4
                             4      0+
        2                 0        2
                                     0+
        2                 2        0
                             0      0+
        2                 2        2
                             2      0+
        2                 2        4
                             4      0+
        2                 4        2
                             2      0+
        2                 4        4
                             4      0+
        2                 4        6
                             6      0+
  4d-( 2)  4d ( 6)  4f-( 3)  4f ( 1)
        2              5/2      7/2
                          7/2      0+
        2              3/2      7/2
                          7/2      0+
        2              9/2      7/2
                          7/2      0+
  4d-( 2)  4d ( 6)  4f-( 4)
        0                 0
                           0+
        2                 2
                           0+
  4d-( 3)  4d ( 5)  4f ( 4)
     3/2      5/2   2;    2
                    2      0+
     3/2      5/2   2;    4
                    4      0+
     3/2      5/2   4;    2
                    2      0+
     3/2      5/2   4;    4
                    4      0+
  4d-( 3)  4d ( 5)  4f-( 1)  4f ( 3)
     3/2      5/2      5/2      7/2
                    1      7/2      0+
                    2      7/2      0+
                    3      7/2      0+
                    4      7/2      0+
                    1      3/2      0+
                    2      3/2      0+
                    3      3/2      0+
                    4      3/2      0+
     3/2      5/2      5/2      5/2
                    1      5/2      0+
                    2      5/2      0+
                    3      5/2      0+
                    4      5/2      0+
```

```
4d-( 3)  4d ( 5)  4f-( 1)  4f ( 3)
    3/2      5/2      5/2      9/2
                  2      9/2      0+
                  3      9/2      0+
                  4      9/2      0+
    3/2      5/2      5/2     11/2
                  3     11/2      0+
    3/2      5/2      5/2     11/2
                  4     11/2      0+
4d-( 3)  4d ( 5)  4f-( 2)  4f ( 2)
    3/2      5/2        0        2
                  2                0+
    3/2      5/2        0        4
                  4                0+
    3/2      5/2        2        0
                  2        0      0+
    3/2      5/2        2        2
                  1        2      0+
                  2        2      0+
                  3        2      0+
                  4        2      0+
    3/2      5/2        2        4
                  2        4      0+
                  3        4      0+
                  4        4      0+
    3/2      5/2        2        6
                  4        6      0+
    3/2      5/2        4        0
                  4        0      0+
    3/2      5/2        4        2
                  2        2      0+
                  3        2      0+
                  4        2      0+
    3/2      5/2        4        4
                  1        4      0+
                  2        4      0+
                  3        4      0+
                  4        4      0+
    3/2      5/2        4        6
                  2        6      0+
                  3        6      0+
                  4        6      0+
4d-( 3)  4d ( 5)  4f-( 3)  4f ( 1)
    3/2      5/2      5/2      7/2
                  1      7/2      0+
                  2      7/2      0+
                  3      7/2      0+
                  4      7/2      0+
    3/2      5/2      3/2      7/2
                  2      7/2      0+
                  3      7/2      0+
                  4      7/2      0+
    3/2      5/2      9/2      7/2
                  1      7/2      0+
                  2      7/2      0+
                  3      7/2      0+
                  4      7/2      0+
4d-( 3)  4d ( 5)  4f-( 4)
    3/2      5/2        2
                  2      0+
    3/2      5/2        4
                  4      0+
                0        0
                         0+
4d-( 4)  4d ( 4)  4f ( 4)
                2   2;   2
                         0+
                2   4;   2
                         0+
                4   2;   4
                         0+
                4   4;   4
                         0+
4d-( 4)  4d ( 4)  4f-( 1)  4f ( 3)
                0      5/2      5/2
                          5/2      0+
                2      5/2      7/2
```

```
                        7/2      0+
               2      5/2      3/2
                        3/2      0+
               2      5/2      5/2
                        5/2      0+
               2      5/2      9/2
                        9/2      0+
               4      5/2      7/2
                        7/2      0+
               4      5/2      3/2
                        3/2      0+
               4      5/2      5/2
                        5/2      0+
               4      5/2      9/2
                        9/2      0+
               4      5/2     11/2
                       11/2      0+
4d-( 4)  4d ( 4)  4f-( 2)  4f ( 2)
               0        0        0
                                  0+
               0        2        2
                           2      0+
               0        4        4
                           4      0+
               2        0        2
                                  0+
               2        2        0
                           0      0+
               2        2        2
                           2      0+
               2        2        4
                           4      0+
               2        4        2
                           2      0+
               2        4        4
                           4      0+
               2        4        6
                           6      0+
               4        0        4
                                  0+
               4        2        2
                           2      0+
               4        2        4
                           4      0+
               4        2        6
                           6      0+
               4        4        0
                           0      0+
               4        4        2
                           2      0+
               4        4        4
                           4      0+
               4        4        6
                           6      0+
4d-( 4)  4d ( 4)  4f-( 3)  4f ( 1)
               2      5/2      7/2
                        7/2      0+
               2      3/2      7/2
                        7/2      0+
               2      9/2      7/2
                        7/2      0+
               4      5/2      7/2
                        7/2      0+
               4      3/2      7/2
                        7/2      0+
               4      9/2      7/2
                        7/2      0+
4d-( 4)  4d ( 4)  4f-( 4)
               0        0
                         0+
               2        2
                         0+
               4        4
                         0+
```

Logically, the above is a list of configurations and their coupling.

It is desirable to think of the above expansion as a list of configurations. This has the potential of dramatically reducing the time for generating angular data in that the search for differences in the subshells for an interaction matrix element or for a transition operator would be done only once rather than many times. The dependence on specific quanum numbers could then quickly applied. It is essential that the definition of a CFG (configuration) include indices for various quantities. A possible data structure could be the following.

```
MODULE  CFG_def
   IMPLICIT NONE
   TYPE CFG (n, m)
   INTEGER, LEN :: n, m
   INTEGER(kind=1),DIMENSION(n) :: iel, qa
   INTEGER(kind=1),DIMENSION(n,m) :: qn, jcup
   END TYPE cfg

   TYPE(cfg, DIMENSION(:,:)), DIMENSION(ncfg) :: CSF_list
MODULE CFG_def
```

In the above "n" is the number of subshells outside the closed core and "m" the number of CSFs for a specific configuration. Here iel is the index for a specific "el" label, qa, the occupation of the subshell, qn the $2 * J$ quantum number (negative if odd) for the subshell, and jcup, the coupling of the subshells from left to right.

A more ambitious procedure would take full advantage of the shorter format.

GRASP92 stored one list of configurations with all possible couplings which made specifying eigenvalues in an RCI calculation was not straight forward.. So, for historical reasons, GRASP reads all blocks even ehen a more efficient procedure is to loop a calculation over the various blocks. An example, is RANGULAR and RCI. RMCDHF is a bit different but, only needs to store the occupation numbers -- coupling informaton is not needed.

Viewing the CSFs for a given block as a list of configurations, has important consequences for performing angular integrations. Processing of orbital sets and occupation number would be done only once for a configuration even though there may be many couplings.

The previous CFG_def, n was the number of shells, and m the number of couplings.

```
MODULE  CFG_def
   IMPLICIT NONE
   TYPE CFG (n, m1, m2)
   INTEGER, LEN :: n, m1, m2
   INTEGER(kind=1),DIMENSION(n) :: iel, qa
   INTEGER(kind=1),DIMENSION(n,m1) :: qn
   INTEGER(kind=1),DIMENSION(n,m2) :: jcup
   INTEGER(kind=1),DIMENSION(m1)   :: qn-set
   END TYPE cfg

   TYPE(cfg, DIMENSION(:,:)), DIMENSION(ncfg) :: CSF_list
MODULE CFG_def
```

In this format, we store: iel - index of electrons in the subshells qa - number of equivalent electrons in each shell qn - number of sets of quantum numbers for each subshell jcup - coupling information (left to right) and final JP iend - last jcup entry for a given qn

With the qn information is the last jcup entry for each qn. If for example, the first entry has 2 cases and the second one 8, the iendjcup entries are (2, 10).

4.4 Interaction Matrix

Since the interaction matrix, Hmat, is symmetric, only the upper triangle arrays need to be stored. And in the sparse mode, only the non-zero values along with a row index. If the row index is represented as an INTEGER*4 the maximimum value would be $2^{31}-1 = 2,147,483,647$, sufficient for most needs. For fast vector operations, Hmat should probably be in adjacent memory locations. If the matrix size is N, and number of non-zero matrix elements, NZE $\leq N(N+1)/2$, a value that needs to be stored as INTEGER*8.

In GRASP2K, the entire matrix is stored as one long array which, for $N = 1,000,000$ could require $8N(N+1)/2$ bytes of total memory. When distributed over, say 64 processors, each processor would require about 31.25 GB of memory. More experience is needed but it would appear that a datastructure in which the length of the j'th column vector $V_j \leq j$ might have better performance. Memory for each vector would be allocated separately although the SEQUENCE option might be useful. Below is a data structure for a column vector

```
MODULE Hmat_def
   TYPE col(n)
       INTEGER, LEN :: n
       INTEGER, DIMENSION(1:n), ALLOCATABLE :: row_ind
       REAL*8, DIMENSION(1:n), ALLOCATABLE  :: H_val

   END TYPE col

   Type(col), Dimension(1:ncfg)  :: H_mat
END MODULE Hmat_def
```

It is possible that pointers need to be used. This definition could be checked with the full upper triangular matrix. Note that all the Lapack routines need a single array. In this notation $H_mat(j)$

Closely related to this matrix of values is the matrix of expressions. Every numerical value is associated with an expression for a matrix element. So, what is needed is a data type for a matrix element which consists of coefficients and the index of a radial integral. Since the lengths will vary and is unknown, each element could be treated as an ang_linked_list. The size of the record might be something like 20. Or there may be a better solution.

```
TYPE, EXTENDS(col) :: expr(ns)
    INTEGER, LEN   :: ns
    Type(ang_linked_list), DIMENSION(1:ns), ALLOCATABLE ;; H_expr
END TYPE expr
```

These various data structures need to be checked and tested.

5 Tasks to be performed

5.1 Generating expansions of CSFs that interact

There are two routines for generating SD expansions from configurations -- jjgen and gencsl. It is not clear to me what algorithms are used. The RCSFgenerate interface is the better. If jjgen is used as the basis for our algorithm, then we need to determine

1) How to add CSFs to the list (rather than merge)

2) How to efficiently eliminate the CSFs that do not interact

2) How to write the CSFs in the new format

A fundamental question is whether the expansion should be stored externally in the new format or whether the rscf.inp file should consist of the *recipe* for generating. For example, instead of having the information the full list, we could have something like the current script, which I call the "recipe".

```
    # 1.  The MR list
Z='Pr+3'
jb='0'
je='12'

rcsfgenerate <<EOF1
*
4
4d(10,*)4f(2,*)5s(2,*)5p(6,*)
4d(10,*)5s(2,*)5p(6,*)5f(2,*)
4d(10,*)4f(2,*)5s(2,*)5p(4,*)5d(2,*)

 5s, 5p, 5d, 5f
 ${jb}, ${je}
0
n
EOF1

mv rcsf.out ${Z}.MR.c
cp ${Z}.MR.c rcsfmr.inp

rcsfgenerate <<EOF1
*
4
4d(10,*)4f(2,*)5s(2,*)5p(6,*)
4d(10,*)5s(2,*)5p(6,*)5f(2,*)
4d(10,*)4f(2,*)5s(2,i)5p(4,i)5d(2,i)
4d(10,i)4f(2,*)5s(2,*)5p(4,*)5d(2,*)

 5s, 5p, 5d, 5f, 5g
 ${jb}, ${je}
2
n
EOF1
```

This information (in a slightly revised format) could define the name.c file in the external view. It would mean that after reading the file, the list is generated and stored as the internal view. Every application would need to regenerate the list, so it could only be done if lists could be generated rapidly. This procedure would be appropriate if lists can be generated rapidly.

5.2 Angular integrations

There are two questions that need to be investigated further.

1) Instead of evaluating matrix elements (interactions between pairs of CSFs) one by one, can faster, more efficient algorithms be developed that evaluate all the matrix elements between two configurations at essentially the same time. There should be gains in efficiency in finding the interacting orbitals that need be done only once, but maybe there are other speed-ups.

2) Studies are needed on how to deal with different cores of filled subshells. We would still like to deal with contributions to the total energies from core shells by formula.

5.3 Finding many eigenvalues and eigenvectors of large matrices

Our computational method is based on

1) The MR set of CSFs with which other CSFs should interact. A CSF interacts if it interacts with at least one member of the MR set.

2) The zero-order space of CSFs with large expansion coefficients or possibly CSFs that define the VV-space. All others define the first-order space of the wave function.

Let us assume the two sets of expansion coefficients are represented by the column vectors $c^{(0)}$ and $c^{(1)}$, respectively. This also partitions the interaction matrix H into blocks so that eq:HM-CI becomes

$$\begin{pmatrix} H^{(00)} & H^{(01)} \\ H^{(10)} & H^{(11)} \end{pmatrix} \begin{pmatrix} c^{(0)} \\ c^{(1)} \end{pmatrix} = E \begin{pmatrix} c^{(0)} \\ c^{(1)} \end{pmatrix}, \tag{1}$$

where $H^{(00)}$ is the interaction matrix between large components, $H^{(11)}$ for interactions between small components of the wave function, and $H^{(01)} = H^{(10)}$ represents the interactions between CSFs of the large and those of the small block. This equation can be rewritten as a pair of equations, namely

$$\begin{array}{rcrl} (H^{(00)} - EI)c^{(0)} & + & H^{(01)}c^{(1)} & = 0, \\ H^{(10)}c^{(0)} & + & (H^{(11)} - EI)c^{(1)} & = 0. \end{array} \tag{2}$$

Solving for $c^{(1)}$ in the second equation and substituting into the first, we get an eigenvalue problem for $c^{(0)}$

$$\left(H^{(00)} - H^{(01)} \left(H^{(11)} - EI \right)^{-1} H^{(10)} - EI \right) c^{(0)} = 0. \tag{3}$$

This is known as a *method of deflation* in numerical analysis since it reduces an eigenvalue for a matrix of size $N \times N$ to an eigenvalue problem of size $m \times m$, where m is the expansion size of $c^{(0)}$. Of course, once E and $c^{(0)}$ have been determined, the small components can be generated from the expression

$$c^{(1)} = -(H^{(11)} - EI)^{-1} H^{(10)} c^{(0)} \tag{4}$$

and a full wave function is defined. Note that the eigenvalue problem is now non-linear and needs to be linear.

The Davidson method used by GRASP needs to be reviewed. Per Andersson (Swedish Defense Research Board) recommended the use of FEAST which is an elaborate eigensolver. A simpler approach would be to used the eigenvectors from a linearized Eq. 3 as initial estimates and use Davidson for finding the required orthonormal eigenvectors. With this method, it also seems natural that $H^{(00)}$ and $H^{(01)}$ be expressed in terms of dense matrices which could speed up the Matrix-Vector multiplications at the heart of the Davidson method.

5.4 HCI – The Human Computer Interaction

Our codes so far have largely been interactive in that the user provides data as requested by the software. But Fortran 90 and higher support command line arguments that can over-ride default values that the system can provide as done in the program DBSR_HF. In such a system, a calculation depends on a set of values that function like a "dashboard" or information in a file like a <name>.c file. The namelist feature for data input in an order independent manner makes the command-line feature user friendly.