An MCHF atomic-structure package for large-scale calculations

Charlotte Froese Fischer[a] Georgio Tachiev[a]
Gediminas Gaigalas[b] Michel R. Godefroid[c]

[a]*Department of Electrical Engingeering and Computer Science, Box 1679B, Vanderbilt University, Nashville, TN 37235, USA*

[b]*Institute of Theoretical Physics and Astronomy, A. Goštauto 12, Vilnius 2600, Lithuania*

[c]*Chimie quantique et Photophysique, CP160/09, Université Libre de Bruxelles, B 1050 Brussels, Belgium*

Abstract

An MCHF atomic-structure package is presented based on dynamic memory allocation, sparse matrix methods, and a recently developed angular library. It is meant for large-scale calculations in a basis of orthogonal orbitals for groups of LS terms of arbitrary parity. For Breit-Pauli calculations, all operators – spin-orbit, spin-other orbit, spin-spin, and orbit-orbit – may be included. For transition probabilities the orbitals of the initial and final state need not be orthogonal. A biorthogonal transformation is used for the evaluation of matrix elements in such cases. In addition to transition rates of all types, isotope shifts and hyperfine constants can be computed as well as g_J factors.

Key words: atomic structure, bound states, Breit-Pauli Hamiltonian, configuration interaction, complex atoms, correlation, g_J values, hyperfine constants, isotope shift, LSJ wavefunctions, transition probabilities
PACS: 31.15.Ne, 31.24.-v, 32.10.-f, 32.30.-r

NEW VERSION SUMMARY

Title of program : ATSP2K ; *version number:* 1.00
Catalogue identifier:
Program obtainable from: CPC Program Library, Queen's University of Belfast, N. Ireland
Computers: Pentium III 500 Mhz;
Installations: Vanderbilt University, Nashville, TN 37235, USA
Operating systems under which the present version has been tested: Red Hat 8,

Programming language used in the present version: FORTRAN 90
Memory required to execute with typical data: 256 Mbytes words
Peripherals used: terminal, disk
No. of bits in a word: 32
No. of processors used: 1
Has the code been vectorised or parallelized?: yes
No. of bytes in distributed package, including test data, etc.: 1 705 294 bytes
Distribution format: gzipped compressed tar file
CPC Program Library subprograms used: none
Nature of physical problem
This program determines energy levels and associated wave functions for states of atoms and ions in the MCHF (LS) or Breit-Pauli (LSJ) approximation. Given the wave function, various atomic properties can be computed such as electric (Ek) and magnetic (Mk) multipole radiative transition probabilities ($\mathrm{k}_{\max}$ =10) between LS or LSJ states, isotope shift constants, hyperfine parameters, and g_J factors.
Method of solution
The new version of the program closely follows the design and structure of the previous one [1], except that a simultaneous optimization scheme has been introduced. This program uses the angular methodology of [2] and has been extended to include partially filled f-subshells in wavefunction expansions but assumes all orbitals are orthonormal. The biorthogonal transformation method is used to deal with the non-orthogonality of orbitals between initial and final states of an electromagnetic radiative transition.
Reason(s) for the present version
The previous version of the MCHF atomic structure package [1] was intended for small calculations, ideal for someone not familiar with the code, producing extensive print-out of intermediate results. The codes for the calculation of spin-angular coefficients were often not the most efficient and could only treat configurations with open f-subshells containing at most two electrons or an almost filled shell with one hole. The present version is designed for large-scale computation using algorithms for angular integration that have been shown to be faster, and include the case of arbitrarily filled f-shells. In addition, the MCHF program has been modified to include optimization on an energy functional that is a weighted average of energy functionals for expansions of wavefunctions for different LS terms or parity, thus facilitating Breit-Pauli calculations for complex atomic systems and for computing targets in collision calculations.
Summary of modifications
Programs have been modified to take advantage of the newly developed angular library [2], extended to arbitrarily filled f-shells. New programs have been developed for simultaneous optimization and for the efficient calculation of atomic spectra and transition rates for an iso-electronic sequence. All applications now take advantage of dynamic memory allocation and sparse matrix methods.
Restrictions on the complexity of the problem
All orbitals in a wave function expansion are assumed to be orthonormal. Configuration states are restricted to at most eight (8) subshells in addition to the closed shells common to all configuration states. The maximum size is limited by the available memory and disk space.

Typical running time
Included with the code are scripts for calculating E2 and M1 transitions between levels of $3s^2 3p^2$ for Si and P^+. This calculation has two stages: LS and LSJ. The calculation of the former required 21 minutes for the LS calculation and 36.5 minutes for the Breit-Pauli configuration interaction calculation that determines the mixing of the terms.
Unusual features of the program
The programming style is essentially F77 with extensions for the POINTER data type and associated memory allocation. These have been available on workstations for more than a decade but their implementations are compiler dependent. The present serial code has been installed and tested extensively using both the Portland Group, pgf90, compiler and the IBM SP2, xlf90, compiler. The former is compatible also with the Intel Fortran90 compiler. The MPI codes are included for completeness though testing has not been as extensive.
References

(1) C. Froese Fischer, Comput. Phys. Commun. 74 (2000) 432.

(2) G. Gaigalas, Lithuanian J. Physics **41** (2000) 39.

LONG WRITE-UP

Contents

1 Introduction

The earlier ATSP_MCHF atomic structure package [1] may be used to determine an array of atomic properties as described elsewhere [2]. It was designed primarily with small applications in mind, though with the recent increases in memory size, CPU speed, and disk space, the size of "small cases" has increased. Even so, important differences exist in programs designed for large-scale computation. For one thing, much less information is printed or sent to the screen: otherwise the amount of information becomes overwhelming.

At the same time, large-scale methods need to be more automatic and take into account the fact that the problem can easily grow to where the memory or disk space are limiting factors. In the large-scale methods described here, memory is used reasonably efficiently through the use of dynamic memory allocation. In most workstations, the F77 compilers had extensions for many years that supported the POINTER data type and consequently programs could be designed to use "just enough" memory rather than the previous philosophy of the "maximum allowed memory". At the same time, files which were written in ascii form so as to be readable, needed to be written in binary form in order to retain accuracy, and also save disk space. At the same time, the interaction matrix, whether stored in memory or on disk, was most efficiently stored in a sparse matrix representation. The Davidson algorithm [3] could then be used to find the needed eigenvalues or eigenvectors.

The previous version made use of non-orthogonal orbitals. With only a few configuration states, it was easy to decide which should be non-orthogonal and which orthogonal. With the more automatic, large-scale methods, the management of a limited amount of non-orthogonality became untenable. Thus the current code assumes that all orbitals describing a state are orthogonal. In a Breit-Pauli calculation where relativistic effects are included through the generation of an interaction matrix using the Breit-Pauli Hamiltonian, and optionally also including the mass polarization correction for the finite mass of the nucleus, the orbitals for all terms need to be orthonormal. To achieve this in an "automatic" fashion, the MCHF procedure has been modified, so that an orbital set can be found that simultaneously describes the terms that are important for the mixing, including several eigenvalues of the same LS term. This feature is similar to the "Extended Optimal Level" (EOL) scheme that has been available in GRASP [4] for many years. However, for transitions *between* states, orthogonality is not assumed. For such calculations biorthogonal transformations [5] are determined so that matrix element evaluation can proceed in the usual manner. Thus for E1 transitions, independent optimization of initial and final states is still possible. Indeed, since the initial state is often a much more compact state than the final state, this is highly desirable.

A typical calculation for the wavefunction of a single LS term, for example, proceeds in the following stages:

Generate configuration state functions
$\Downarrow$
Obtain the energy expression for the term
$\Downarrow$
Solve the Multiconfiguration Hartree-Fock (MCHF) problem
$\Downarrow$
Include Breit-Pauli and/or mass-polarization corrections
$\Downarrow$
Evaluate atomic properties

Each stage in such a sequence is considered an application. In order to simplify the dynamic memory allocation, each stage outputs information about the problem and its size to a file so that the next stage has the crucial information for memory allocation. The applications are all designed around a central library. These are classified as angular, radial, common, or special such as the Davidson package [3], and the Lapack and Blas [6] libraries that often are part of a compiler for a specific architecture.

In this paper, we will take a top-down approach to describing the ATSP2K package. We will begin by describing the computational procedure of a Breit-Pauli transition calculation, then individual applications, useful tools, the libraries, and the installation of the codes. We will concentrate here on the changes in algorithms from the codes published earlier and the revised file structures.

2 A Computational Process for a Breit-Pauli Transition Calculation

The ATSP2K application was designed primarily for transition probability calculations that include low-order relativistic effects though, at a more basic level, the process determines one or more wave functions in the LSJ approximation within an expansion model, from which atomic properties can be determined. Energy is a special property since the variational method looks for a stationary solution of an energy functional (or expression). Other properties are transition probablities, hyperfine interactions, isotope shifts, and Zeeman g_J factors.

Let us take a particular problem, namely the calculation of the transition probabilities for the $3s^2\ ^1S_0$ – $3s3p\ ^1P_1^o$ and $^3P_1^o$ transitions in Mg I. Our non-relativistic zero-order approximation will be an expansion over the configuration states $1s^22s^22p^63l3l'$ of the proper parity and LS. The set of associated configuration state functions (CSF) forms a basis for the zero-order wave function and is referred to as the "multi-reference set". In the Breit-Pauli approximation, a ground state wave function that includes the low-order relativistic effects from interactions between the different terms of $3s^2$ and $3p^2$ will be an expansion over $^1S_0$ and $^3P_0$, whereas an excited state calculation for the odd $3s3p$ states that ignores relativistic effects arising from interactions between the different LS terms of $3p3d$ will be an expansion over $^1P_1^o$ and $^3P_1^o$. For each LS, the wave function expansion will consist of single and double excitations from the multi-reference set to the active orbital set, with the restriction that at most one replacement is either $2s$ or $2p$ and $1s$ is inactive. Furthermore, calculations are "systematic" in that the set of orbitals increases in size from $n = 3$ to $4, 5, ..$, where n is the maximum quantum number in the orbital set. Finally, for each parity the $1s$, $2s$, $2p$ orbitals can be obtained from an average energy Hartree-Fock (HF) calculation – $3s^2$ for the even states and $3s3p$ for the odd.

A possible computational process for this approximation is the following:

```
1. Perform LS calculations
   For each parity (odd and even)
      Perform a HF calculation*
         For each LS  (1P and 3P for odd, 1S and 3P for even)
            For each n  (3, 4, 5, 6, 7)
               i) Generate the expansion
               ii)Perform an MCHF calculation for orbitals and
                  expansion coefficients*
   For each even LS
      For each odd LS
         Perform transition probability calculations
2. Perform Breit-Pauli calculations
   For each parity (odd and even)
      i)   Concatenate the LS expansions for an LSJ expansion
      ii)  Perform angular integrations
      iii) Form the Breit-Pauli matrix*
      iv)  Compute eigenvalues/eigenvectors*
```

`Perform LSJ transition calculations between odd and even states*`

In "spectrum" calculations where a series of transition probabilities are to be predicted accurately so that energy differences are more important than the actual total energies, it might be more important to have the same common core orbitals ($1s$, $2s$, $2p$) for all levels. The HF program [7] could provide such results given the configuration `3s(1.5)3p(.5)`, a configuration with fractional occupation numbers whose energy expression is the average of the energy expressions for `3s(2)3p(0)` and `3s(1)3p(1)`.

In the above, the steps with the '*' can easily be performed for an iso-electronic sequence as well as a single state since the angular information is independent of the nuclear charge. In spectrum calculations for levels up to a certain degree of excitation [11], the even and odd states may be divided into groups with the orbitals of one group optimized independently from another. For example, the orbitals for excited states can be optimized separately from those of lower states. The needed calculations may then be performed using shell scripts. For such a purpose it is convenient to adopt a "language". Obvious variables are:

variable	*examples*
parity	'o' or 'e'
grp	1,2, ... (separately optimized)
LS	1P, 3P etc.
n	3, 4, 5, ...
Z	12 (Atomic number)
At	Mg (Atom)

From these variables, file names can be generated that denote the dependence of the data and the file type. As in previous versions of ATSP application, file extensions are used to indicate the type:

extension	*data in the file*
`.c`	configuration state function (CSF) expansion
`.w`	radial wave functions (numerical values in binary form)
`.l`	expansion coefficients from a non-relativistic (LS) calculation
`.j`	expansion coefficients from a Breit-Pauli (LSJ) calculation
`.t`	term dependence of a .j file

Using the above variables and file extensions, the MCHF input data for a set of terms (Z-independent) would be the concatenated set of files, `${LS}${parity}${grp}.${n}.c`, and the output as `${LS}${parity}${grp}.${z}_${n}.l` and `${parity}${grp}.${z}_${n}.w` (independent of *LS* but dependent on Z and n). In this notation the “$” indicates that the symbol following is a variable, whose value is to be used. When the name of the variable is more than one character, the name is enclosed in curly brackets. Then the CSF expansion for a Breit-Pauli calculation could be `${parity}${grp}.${n}.c` (independent of *LS* and Z), and the expansion coefficients (Z-dependent) as `${parity}${grp}.${z}_${n}.w`. An example of a shell script that first concatenates the expansions of the even (group 1) 1S and 3P configuration states, performs the angular integrations for the non-relativistic hamiltonian, and performs MCHF calculations for an isoelectronic sequence is given in Figure 1.

3 Applications

Now let us describe the applications in the order in which they may be used in a computation. We shall use small caps for the names of packages (such as ATSP and the typewriter font for names of variables, files, or executables (such as `mchf`) that might be typed either as part of a program or during the execution of a system command. Applications from the previous package will be capitalized along with names of routines.

3.1 Generating configuration state lists: `gencl` *and* `lsgen`

When many levels are to be computed, a consistent model needs to be used that describes all levels satisfactory. In striving for a “best possible” solution, it is easy to attempt a calculation that is overly ambitious. For this reason,we have found it best to begin by generating the expansions and monitoring the size of the expansion.

The new `gencl` program has the same design as GENCL (see Froese Fischer and Liu [12]) but has been extended to 8 open shells, arbitrarily-filled f-shells, and converted to dynamic memory allocation. The original program was written to use external files so as to avoid the dimension problem. This was later changed to using internal files with dimension, but in the present version, arrays are reallocated to larger size, when needed. The extension to f-shells required some changes in notation. The subroutines that changed a lot are the LVAL, SYMB and COUPLD. The rest of the subroutines are either unchanged or the changes are not substantial. The LVAL subroutine is extended with a possibility of converting the symbols O and Q into their corresponding quantum numbers and in SYMB to convert the quantum numbers

Fig. 1. Example of a shell script for an MCHF calculation optimizing simultaneously on even ^1S and ^3P terms of an iso-electronic sequence. CSF expansion files are assumed to exist and that the HF results have been stored as `${parity}${grp}.${z}_2.w`.

```
set -x              ! echo the shell commands
parity=e
grp=1
LS1=1S
LS2=3P

for n in 3  4  5  6  7
do
   (cat \
       ../files_c/${LS1}${parity}${grp}.$n.c \
       ../files_c/${LS2}${parity}${grp}.$n.c \
    > cfg.inp

nonh   # Perform angular integrations for the current expansion

for z in 14 15 16 17 18 19 20 21 22
   do
      (echo $n;
case $n in
   3) ORB=all;   ITER=200;;
   4) ORB==7;    ITER=200;;
   5) ORB==12;   ITER=200;;
   6) ORB==15;   ITER=100;;
   7) ORB==15;   ITER=100;;
esac

IC=0
previous_n=‘expr $n - 1‘;
cp -f ${parity}${grp}.${z}_${previous_n}.w wfn.inp;

mchf << EOF  > mchf_out.$z_$n
Z=${z},${z}
1  # 1S
1  # 3P
${ORB}
1s,2s,2p
y
n
y
n
F,10E-9,10E-9
n
${ITER},${IC}
y
EOF
      mv wfn.out ${parity}${grp}.${z}_${n}.w;
      mv ${LS1}.l ${LS1}${parity}${grp}.${z}_${n}.l;
      mv ${LS2}.l ${LS2}${parity}${grp}.${z}_${n}.l;
      echo "finished" )
    done)
done
```

11 and 12 into corresponding symbols. In the COUPLD subroutine the data blocks containing the term characteristics are enlarged.

Input and output data of the new version are the same as before although the number of subshells has been increased to eight subshells in addition to a common closed core. The latter required a more compact format for the specification of coupling. One needs also to take into account that the classification of terms of the f subshell is more complicated than for s, p, d subshells. For the classification of f-subshell terms the characteristics $(2S{+}1)$ (multiplicity), L (total orbital momentum), and ν (seniority) are not sufficient. Here we use a notation ${}^{(2S+1)}L^{Nr}$ for the classification of an f-subshell. The Nr is single character, which corresponds to the group labels νWU as described by Gaigalas and Froese Fischer [7] and Gaigalas *et al* [13]. The value Nr is found in Table 1 of [13] where all terms for f-subshells are presented. In most cases, Nr appears to be a single digit, but since it is a single character, the single letter `A` is used instead of the number 10.

For other quantum numbers that usually are single digits, say n, the values are encoded on output. When $n > 9$, the value of n in the output file is encoded as `n = CHAR(n+ICHAR('0'))` (see [12]). For example, the values of the principal quantum number n or multiplicity $(2S + 1)$ may exceed 9. On most systems the list of integers, $\{10, 11, 12, 13, 14, 15\}$ map into the list of characters, $\{:, ;, <, =, >, ?\}$. The CSF list that is produced is written to the file `cfg.inp`. For large cases, `gencl` may fail in which case the more recent `lsgen` is recommended.

When f-shells are restricted to two electrons, the earlier LSGEN program developed by Sturesson and Froese Fischer [14] is satisfactory. This program has been extended by Sturreson to arbitrarily filled f-shells, but has not been documented. Like the previous version, the CSF list that is output is left in `clist.out`. For simple cases, `gencl` is easier to use interactively but `lsgen` is more powerful. There is another difference. Unlike `gencl` where ${}^{10}D$ on some systems must be entered as `:D`, `lsgen` allows the user to specify `10D`, although the output file will adhere to the proper convention.

3.2 Angular integrations for LS expansions: `nonh` *and* `nonhz`

The `nonh` program performs the angular integrations that are needed for the evaluations of the interaction matrix of a non-relativistic Hamiltonian. The use of this program has changed very little from the previous version [15] though the computational procedure is totally different. The assumption has now been made that all orbitals within a given LS term are orthonormal, hence there are no overlaps. The new angular libraries [16] are used.

The `cfg.inp` file may now consist of concatenated expansions for several LS and parity terms, each list terminating with an asterisk ("*"), and each retaining header information, for the convenience of the user. It is assumed that there is only one expansion for a given LS and parity.

The role of `nonh` is to produce data needed by `mchf` for deriving the multiconfiguration Hartree-Fock equations, and for generating the interaction matrix. Suppose that we have the wave function expansion

$$\Psi(\gamma LS) = \sum_i^M c_i \, \Phi(\gamma_i, LS), \quad \text{where} \quad \sum_i^M c_i^2 = 1. \tag{1}$$

Then, by definition, the interaction matrix $\mathbf{H} = (H_{ij})$, where (see [2], page 74)

$$H_{ij} = \sum_{ab} w_{ab}^{ij} I(a,b) + \sum_{abcd;k} v_{abcd;k}^{ij} R^k(ab,cd). \tag{2}$$

Here the sum on ab or $abcd$ is a sum over occupied orbitals in either the bra configuration state, $\Phi(\gamma_i LS)$ or ket state $\Phi(\gamma_j LS)$, and $\mathcal{H}$ is the non-relativistic Hamiltonian. The $I(a,b)$ integrals arise from the kinetic energy and the nuclear one-body operators of the Hamiltonian and $R^k(ab,cd)$ are Slater integrals from the two-body Coulomb operator. Then the energy functional can be expressed as a sum of integrals weighted by the components of the eigenvector, and the coefficients in the expression for the interaction between configuration states, namely

$$\mathcal{E}(\gamma LS) = \sum_{ab} w_{ab} I(a,b) + \sum_{abcd;k} v_{abcd;k} R^k(ab,cd) \tag{3}$$

where

$$w_{ab} = \sum_i^M \sum_j^M c_i c_j w_{ab}^{ij} \quad \text{and} \quad v_{abcd;k} = \sum_i^M \sum_j^M c_i c_j v_{abcd;k}^{ij} \tag{4}$$

The program `nonh` first determines the set of all orbitals, and generates a list of all possible integrals, of both types. Symmetry is used to define a canonical form. However, not all possible symmetries of the Slater integrals are used since it is desirable to be able to produce specific mass shift parameters from the same data. This restricts possible symmetries in that interchanging one orbital between the left (bra) and right (ket) is not allowed, since this would change the sign of a non-zero isotope shift contribution. For each LS term and parity in the `cfg.inp` file, the output data consists of a one-dimensional list of coefficients, w_{ab}^{ij} or $v_{abcd;k}^{ij}$ (denoted by `cn(j)` in `nonh`, and `coeff` in `mchf`) and with each such coefficient there is associated an integral. The latter is specified by an integer, `inptr(j)` that indicates the position of the integral within the list of all possible integrals. In this way, there is no need for sorting the data since lists could become too large for in memory sorts as used in [1]. In the generation of a matrix element, we then need to know where the data for a matrix element terminates. This is achieved by specifying the end position of a matrix element without the necessity of specifying the first, since it follows the last coefficient of the previous. This is the role of the array `ico(i)` which indicates the last position of the i'th non-zero matrix elements. Since only non-zero matrix elements are stored, another integer is needed to specify the value of the row index: `jan(i)` (`ih(i)` in `mchf`) is the array containing this information. Because the list of coefficients and associated integrals may become extremely large, the data is collected and written to disk after `LSDIM=30000` have been collected to memory. The last record for a given LS term will always contain less than `LSDIM` elements, possibly zero, if necessary. The value of `LSDIM` can readily be modified by changing four parameter statements.

This information is made available to `mchf` through different files.

(1) `cfg.h`: This file contains information that used to be written as part of the header file of `cfg.inp`. In order to avoid modifying `cfg.inp`, which may become quite large, similar information is now included in `cfg.h`. Some information pertains to the expansions for each term and parity found in the list.
(2) `yint.lst`: This is a file that has global information about all the expansions in `cfg.inp`. This is a binary file.
(3) `c.lst`: The coefficients and integrals needed for the energy expression and the generation of the interaction matrix are stored, by column, sequentially for each term and parity found in `cfg.inp`. This is a binary file.
(4) `ih.0n.lst`: For each term and parity, numbered $n = 1, 2, ..$, a file is created which records the row index of a matrix element. This facilitates the sparse matrix representation. This also is a binary file.

The interaction matrix is, of course, symmetric and so only the lower or upper part needs to be evaluated. When considered as a lower triangular matrix, the data for matrix elements are generated by column. Thus the computational process is:

```
For column = 1 to ncfg
   For row = column to ncfg
       compute the matrix element
```

Only non-zero matrix elements are stored. This particular structure for the information was determined by conditions desired for `mchf`. Unlike the earlier program where the data structures assumed the entire matrix would be in memory, the large-scale version was designed to compute matrix elements sequentially to facilitate sparse matrix representation of the interaction matrix and the use of Davidson's algorithm based on repeated matrix-vector multiplies. In contrast, the earlier version was more integral oriented: for each integral, there was a list which defined the matrix elements in which the integral occurred and the coefficient of that integral in that matrix element. Thus, if an integral occurred 100 times, 100 different matrix elements would be modified. Consequently, the `nonh` program ended sorting the entire list of coefficients. As the list became large, there was the possibility of needing to resort to sorting on disk, something to be avoided, if possible. The present strategy achieves that.

The program `nonhz` allows the user to specify that a certain number of CSFs at the beginning of each *LS* block, are to be considered as part of a zero-order approximation, that for the remaining CSFs, only the interaction between the zero-order set and the CSFs are to be computed as well as the diagonal interaction. This feature may be useful for very large expansions where a first-order wavefunction may be useful in eliminating CSFs with very small expansion coefficients.

3.3 Orbitals and LS expansions: `hf`*,* `mchf`*,* `mchf_C`*, and* `mchf_CH`

A Hartree-Fock calculation determines the radial functions for the orbitals of a single configuration state whereas the multi-configuration Hartree-Fock method determines both orbitals and expansion coefficients using a variational method applied to an energy functional to obtain optimum solutions that represent stationary solutions.

The energy functional used in the optimization process to determine orbitals need not be an expression for a physical state. A well known example, used in the `hf` program, is the calculation of orbitals for the average energy of a configuration. This is an easily derived expression for any configuration that corresponds to a statistically weighted sum of energy expressions over all possible terms. An energy functional can also be a weighted average of average energies. The `hf` program has the ability to interpret `3s(1.5)3p(.5)` as the average of two configurations, `3s(2)` and `3s(1)3p(1)` and is a convenient application for computing common core orbitals for two configurations or many *LS* terms of a configuration as in `3p(3)3d(1)`.

The present version has been extended to arbitrarily filled *f*-shells [7]. Tatewaki *et al* [17] have shown that the non-relativistic Hartree-Fock ionization potentials of singly ionized lanthanides follow observed trends. However, it should be noted that in many configurations with an open f-shell such as $4f^n5d6s$ LS, single configuration Hartree-Fock calculations are not appropriate (even if relativistic effects can be ignored) in that there often are two or more couplings for the same configuration and term, and `mchf` needs to be used to obtain a better energy. Otherwise, the extension to arbitrarily filled f-shells, though much more difficult in angular theory, is similar computationally to other cases.

In the earlier MCHF program [18,2], some non-orthogonal orbitals could be accommodated which tended to increase the number of orbitals but would lead to shorter wave function expansions and hence smaller interaction matrices. With some effort, considerable advantage could be made of non-orthogonal orbitals, but at the cost of more complex procedures. In large-scale calculations, where expansions are generated by the application of rules applied to orbital sets, no convenient method was found for taking advantage of non-orthogonality. Thus in this version of the code, non-orthogonality is not supported.

Once orbitals have been determined by `mchf`, LSJ wave functions can be determined by computing a Breit-Pauli interaction matrix and determining selected eigenvalues and eigenvectors. The codes have always required that the orbitals of the terms that are included in the LSJ representation [2], needed to be orthonormal. The question then arose as to how the orbitals were to be determined. In the study of the mixing of $2s2p\ ^{1,3}P^o$, an (N,N+1) scheme was proposed [2] in which a set of orbitals were obtained for a primary term, and one extra "layer" of orbitals (orbitals with $n = N + 1$ and the same range of l as those for N) was added, keeping all other orbitals fixed. This worked well when mixing of LS terms was limited to two terms ($^1P^o$ and $^3P^o$, in this case), but became impractical when many terms might interact, as happens in the mixing of terms of $2p^43d$, for example.

The GRASP92 code supports the concept of "Extended Optimum Level" (EOL) where the energy functional is a weighted linear average over expansions for different J's and parity [4]. The present `mchf` program has been extended in a similar fashion in that the energy functional may be a weighted linear combination of energy expressions over different LS terms, different eigenstates of the same term, and also different parities. We refer to this as "simultaneous optimization". Suppose $\mathcal{E}(T_i)$ represents an energy functional for term T and eigenvalue i, assuming orbitals and also wave functions are normalized. Then optimization can be performed on the functional

$$\mathcal{E} = \sum_{T_i} w_{T_i}\mathcal{E}(T_i) / \sum_{T_i} w_{T_i}$$

where w_{T_i} is the weight for T_i. This scheme has been used successfully by

Tachiev and Froese Fischer [19,20] for spectrum calculations in the Breit-Pauli approximation, where all levels up to a certain degree of excitation are computed. It was needed in Mg-like calculations to assure that all states (odd or even) included the same amount of correlation in the core [21]. It would also be suitable for computing all targets in a common basis for continuum calculations [22].

The earlier MCHF [18] program also has been modified for sparse matrix representation of the interaction matrix and simultaneous optimization. The numerical methods remain the same but, in order to make efficient use of memory, some properties of the SCF calculation need to be observed. There also are a few minor changes to the input data. The angular integrations pass on information about the set of possible LS terms, but the revised `mchf` program needs to know which (one or more) eigenvalues of an LS term should be used for the energy functional. The input data for a term and its parity is a comma separated list of indicies possibly followed by weights in parentheses. For example, in the calculation for the second eigenstate of an LS term the input might be `1(.3),2`. A weight of 1.0 is assumed for an index without an explicit weight and the lower eigenstate has the reduced weight of 0.3, so as not to be totally ignored in the optimization. The program also now requires the user to state explicitly which of the orbitals to be varied should be “spectroscopic” with the usual number of nodes. Previously, all orbitals in the first CSF were assumed to be spectroscopic and this could have been extended to multiple LS, but it seemed prudent to allow the user full control, particularly since `lsgen` produces a list in a standard order, something the user might not check. Figure 1 includes an example for input data. Defaults are used for many sets of input parameters as in the earlier MCHF (see [2]).

A cycle of the SCF process divides naturally into two phases.

(1) In the orbital update phase, the appropriate view of the energy functional is given by Eq. (3). To compute the potential and/or exchange function for a specified orbital, say a, it is necessary to search the list for integrals involving orbital a. When an integral is found, its contribution needs to be determined and multiplied by w_{ab} or $v_{abcd;k}$. Thus in this phase the values of the integrals themselves are not needed, only the coefficient defining their contribution to the energy. Since these coefficients depend on eigenvectors, they need to be recomputed before the orbital update phase. In the present implementation, it occurs after the diagonalization process.

(2) In the matrix diagonalization phase, the matrix elements as defined by Eq. (2) need to be assembled, in order, as described earlier. Now the value of the integrals are needed but not their total contribution to the energy functional. Thus, before the diagonalization phase, all integrals need to be re-evaluated.

Since these two phases are totally disjoint, the array of contributions to the energy functional, and the array of integral values may share the same memory.

The angular coefficient data and the interaction matrix are the two major data structures which define the memory use for each SCF iteration. As the number of configurations grows the memory capacity may be exceeded thus requiring some or most of the data to be stored on disk. Disk read/write operations are then performed during each SCF and DVDSON iteration. Disk read/write access is considerably slower compared to memory access. In order to accomplish high computational efficiency it is essential to avoid disk I/O and keep all data in memory. Therefore, optimizing `mchf` for large scale calculations requires management of the disk/memory data storage. In general, the angular coefficients, the interaction matrix elements and the associated pointers are stored in one dimensional arrays. The arrays of angular coefficients are considerably larger than the interaction matrix.

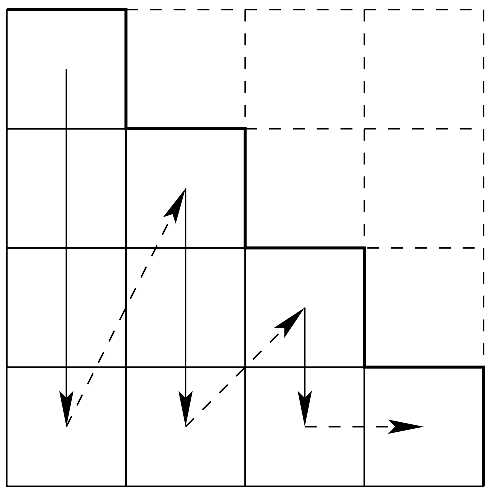
The previous data organization was suitable for computing the entire matrix, in memory. The data structure started with the radial integrals and how they contributed to the interaction matrix. Given a list of integrals, the `nonh` program provided all coefficients and the position in the matrix, where the coefficient times the radial integral made a contribution. Since one radial integral could contribute to many matrix elements, it meant that the matrix elements were not computed in sequential order. Thus the data structure for the interaction matrix was redefined, so that matrix elements were computed in order. Figure 2 shows the order in which the interaction matrix is traversed, the data generated (`hmx`), and the associated pointers (`ico` and `ih`). For each non-zero matrix element, `ico` accounts for the number of total angular coefficients as the matrix is traversed in the direction of the arrows. The row index of each non-zero element is stored in `ih`.

Each coefficient data structure consists of two quantities:

`coeff`	numerical coefficient (double precision)
`inptr`	index pointer to an integral (integer)

Thus the position of each integral needs to be known in advance and this is achieved by generating all possible integrals from the orbital set initially before the start of the SCF iterations.

To maximize the memory utilization, the storage of additional arrays is allocated in a stepwise fashion. Upon memory allocation failure, the remaining arrays are stored on disk. Under this scheme the order in which arrays are allocated becomes important and the arrays used most frequently at each SCF iteration are allocated first.



hmx	− non−zero matrix elements
ico	− index of total coefficients for the current element
ih	− row index of the current element
jptr	− index of total elements for the current column

Fig. 2. Data Generation of the Interaction Matrix

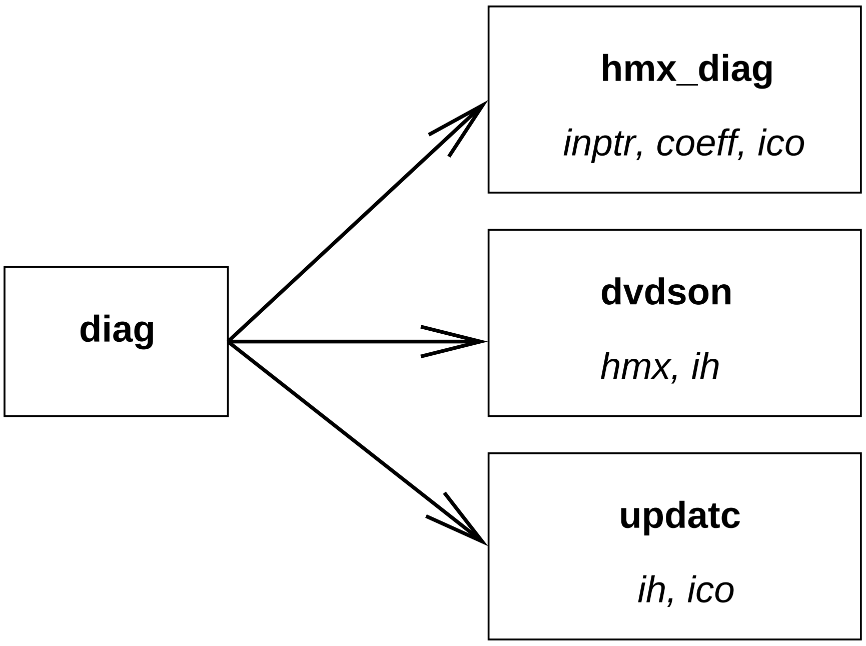


Fig. 3. Storage access requirements in different phases of the routine `diag`.

In the matrix generation phase represented by `diag_hmx`, `inptr`, `coeff` and `ico` are accessed only once. In contrast, the iterative solution of the eigenvalue problem (`dvdson`) requires multiple read access operations on the interaction matrix (only `hmx` and `ih` are used). Therefore, higher priority for storing in memory was given to `hmx` and `ih`. After `dvdson` the memory used in `diag_hmx` and `dvdson` is deallocated and used in `updatc`. Finally, updating the coefficients requires a single access to `ih`, and `ico` suggesting higher priority for `ico` over `coeff`. Table 1 shows the multilevel storage scheduling derived from the frequency of data access. The storage scheduling will depend on the size of the problem and the system capacity and `mchf` is designed to select the best level with respect to computational efficiency.

Table 1
Disk and Memory usage as a function of Level

Memory Use	On Disk	In Memory
Level 1		`hmx, ih, ico, coeff, inptr`
Level 2	`coeff, inptr`	`hmx, ih, ico`
Level 3	`coeff, inptr, ico`	`hmx, ih`
Level 4	`hmx, ih, ico, coef, inptr`	`hmx, ih, ico` (a single column only for each array)

The memory allocation process (`alcsts`), starts with Level 4 and upon success on each level may proceed up to Level 1. At Level 4, if sufficient memory is not available for performing the iterations of Davidson's algorithm with the matrix on disk, the program exits. (This will represent a very large case beyond practical limits for serial computing: in the order of millions of CSFs). On some systems, if the program cannot proceed with allocating memory for the interaction matrix of the largest block, a similar local scheme (in `diag`) is used to allocate memory for each block (the interaction matrix for a particular LS). Blocks can vary significantly in size and in order to improve performance `mchf` is designed to keep smaller blocks in memory if possible.

As shown in levels 1 and 2 of Table 1, `ico` has higher priority for memory allocation compared to `coeff` and `inptr` because `ico` is used in both `diag` and `updatc` routines. However, it is important to note that `updatc` proceeds almost always in memory since all of the memory used in `diag_hmx` and `dvdson` is deallocated and made available.

On some operating systems, this automatic memory management scheme fails and separate programs have been prepared: `mchf_C` assumes the file `c.lst` containing the coefficient information is on disk (Level 3), whereas `mchf_HC` assumes both `c.lst` and `hmx.lst` are on disk (Level 4). Reading the matrix from disk is still faster than relying on swaping memory and virtual memory.

Estimates of the radial wave functions are taken from the file `wfn.inp`, if provided. Otherwise, screened hydrogenic estimates are used. Upon completion (either a maximum of 200 iterations or a change in the weighted average energy of less that 10^{-8} a.u., in default mode) an updated `wfn.out` file is produced. During the course of the calculation intermediate orbital results are written to the `wfn.out` file, for restart purposes, in case of process termination for some reason. In either case, the `wfn.out` may be copied to `wfn.inp` and used to continue SCF iterations. Convergence may occur when the maximum weighted change of all orbitals less than a user defined `scftol` weighted by $\sqrt{Z \times \text{nwf}}$ (where nwf is the number of orbitals), or when the change in the weighted average of energies is less than `cfgtol`. The default values are $1.0{\times}10^{-7}$ and $1.0{\times}10^{-8}$ for `scftol` and `cfgtol`, respectively, though these values may be changed by the user.

The program does not produce a file `cfg.out`. Instead, for each even term, say LSe, a file `LSe.l` is produced with the same format as a `<name>.j` file, but with a line (previously blank) that contains the S parameter for the specific mass shift [2], `Ssms`. If the LS term is odd, the file `LSo.l` is produced.

The `summry` file also contains some additional information. Because of the speed of computers, many one-electron properties that can be derived from the angular data for the energy, can also be computed with negligible CPU time. These include:

(1) The mean radius, the expectation of $\sum_i r_i$.
(2) The mean square radius, the expectation of $\sum_i r_i^2$.
(3) The dipole-dipole operator, the expectation of $(\sum_i r_i)^2$,
(4) The specific mass shift parameter, $S = -\sum_{i<j} \nabla_i \cdot \nabla_j$.

The mean radius gives an indication of the size of the atomic system, whereas the dipole-dipole operator (denoted as `r.r` in the `summry` file) is relevant to long-range interactions [23].

3.4 Breit-Pauli LSJ wave functions: `bpci`, `bp_ang`, `bp_mat`, *and* `bp_eiv`

Once radial functions have been determined that simultaneously represent one or more LS terms, low-order relativistic effects may be included through a configuration interaction calculation using the Breit-Pauli Hamiltonian. Such a calculation may be performed using `bpci`. In effect, it generates potential contributions (independent of J) to the interaction matrix in terms of three files:

(1) `hnr.lst` the non-relativistic, J-independent contributions.
(2) `hzeta.lst` the spin-orbit and spin-other-orbit contributions that need to be multiplied by
$(-1)^{L+S'-J} W(L'S'LS; J1)$ when the matrix is generated.

(3) `hspin.lst` the spin-spin contributions that need to be multiplied by $(-1)^{L+S'-J}W(L'S'LS;J2)$.

In the above, $W(L'S'LS;Jk)$ is a Racah coefficient. Thus this program combines the function of the earlier BREIT [25] and CI [26], without storing the large amounts angular of data that may be generated. At the same time, it is easy to regenerate an interaction matrix either for a new J or to add "term energy corrections" (TEC) where the diagonal matrix elements of a given LS term are all shifted by an amount specified by the user (in cm^{-1}) to improve the accuracy of interactions between terms and the energy spectrum. `bpci_d` is a disk version of this code.

Input files are designated by `<name>.[c,w]` (expansions and radial functions) with the output being `<name>.j` for a relativistic calculation. For each eigenvector in the file, Zeeman factors g_J are reported as well as the $g_J(LS)$ where the latter is the value associated with the LS term of the configuration state with the largest expansion coefficient [24]. The difference between g_J and $g_J(LS)$ is an indication of the importance of the mixing of different LS terms in the wave function expansion. The expansion coefficients are given to eight decimal places. When the "reuse" option is invoked, term energy corrections my be applied to the diagonal matrix elements. For each term, the program asks the user for shifts (in cm^{-1}) which *lower* the energy level by the specified amount. The results are now written to a file called `<name>.new`. Usually, the program is run first without the "reuse" option.

Each state is labelled according to the configuration state of the largest component in the eigenvector. This scheme may not produce unique labels when three or more CSFs interact strongly. In such cases, the user will need to edit the file and determine a suitable, unique sets of labels.

The `tables` program may then be used to find the spectrum from the energies in a `<name>.j` file (this file may be a concatenated file from separate `bpci` runs). By comparing with observation, a shift can be determined. When interactions are not strong, only one "reuse" is often sufficient to bring energy levels into agreement with observation, but for strong mixing, several iterations may be needed. This adjustment process has not been automated.

When Breit-Pauli calculations are performed for an iso-electronic sequence, the time-consuming part of generating the angular data, from which the three files are produced, is repeated each time. For this purpose, the `bpci` program has been divided into three separate applications.

1. `bp_ang`

This program generates angular data much as the earlier BREIT [25] program. Four types of files are produced:

(1) `cint.lst.s` This file contains global information and plays a similar role as `yint.lst` file in `nonh`. It is a binary file.
(2) `c.lst.s` This file contains coefficient information and pointers to integrals, similar to the `c.lst` file in `nonh`. It is a binary file.
(3) `ico.lst.s` This file contains information defining the end of a matrix element, similar to `ico.lst` in `nonh`. It is a binary file.
(4) `ih.lst.s` This file contains information about the row index of each matrix element and is similar to `ih.lst` in `nonh`. It is a binary file.

This program need be run only once per iso-electronic sequence. It may consume a considerably larger amount of disk space than the interaction matrix itself.

2. `bp_mat`

This program first computes all the possible radial integrals and then combines the angular data with radial integrals to generate three files, `hnr.lst.s, hzeta.lst.s, hspin.lst.` with the same data and format as the similarly named files `hnr.lst, hzeta.lst, hspin.lst` in `bpci`.

3. `bp_eiv`

This program assembles the matrices for a series of J-values, and produces a `<name.j>` file that contains selected eigenvalues/eigenvectors as specified by the user. The reuse option is no longer relevant since the three files defining the matrix have been saved. The program requests TEC shifts on each run. These should be set to zero on the first run from which estimates of shifts can be determined.

It is possible to use `bpci` or `bp_ang+bp_mat+bp_eiv` for non-relativistic calculations either by indicating right from the start that a non-relativistic `name.l` file is to be produced or by eliminating all the relativistic operators from a calculation for a `name.j` file. In such cases it is important that the `name.c` file be an expansion for only a single LS term and parity. The program does not check for multiple LS terms in which case the Davidson method may not find the expected eigenvalue. A better procedure for non-relativistic calculations of a series of LS terms and parity, is to use `nonh` followed by `mchf` where the number of orbitals to be varied is `NONE`.

Figure 4 shows an interactive calculation where the name of the files defining the state are `1Po3Po`.

3.5 Transition calculations: `trans`*,* `biotr`*, and* `bioscl`

For transitions between states of the same parity, it often is the case that the transitions are between levels of the same group of terms or, possibly, between states that have been optimized simultaneously. Under these circumstances, the orbitals for the initial and final state will be orthogonal, and the program `trans` may be used. This program is a combination of MLTPOL [27] and TRANS [28] described in the book [2]. Figure 5 shows an example of the use of `trans` for computing E2 transitions between the levels of $2s2p\ ^{1,3}P^o$ where the data for the levels are in files with the name `1Po3Po`. Though these files are used in the "read only" mode, not all systems allow the same files to be specified for both the initial and the final states in which case copies need to be made with different names. Figure 5 also shows the output file that was produced. Note that velocity values are not reported by `trans` in the present relativistic (Breit-Pauli) case. In Breit-Pauli calculations, it is customary to use the non-relativistic form of the transition operator. For the length form, this will correctly include all low-order effects given a Breit-Pauli wave function, but for the velocity form, some effects are omitted. Corrections to the velocity form are needed which have not been implemented. Under such circumstances, `trans` does not report a velocity value. `biotr` on the other hand, reports both length and velocity forms since, in many cases, the omitted relativistic effects are negligible. Thus the velocity form of E1, E2, E3, .. transitions needs to be interpreted with care. If relativistic effects are significant, it needs to be remembered that some low-order effects have been omitted and that only the length form is reliable.

For transitions between states of opposite parity, it usually is advantageous to represent the initial and final states with different orbital sets that are not constrained to be mutually orthonormal. Then, without any transformations, the matrix elements must be evaluated assuming non-orthogonal orbitals, a task that is not exactly straight forward though it can be achieved by representing configuration states as linear combinations of Slater determinants [29]. This procedure is avoided in the present package for keeping the computational advantages of the Racah-Wigner angular algebra that does not require the explicit expressions of atomic states in Slater determinants for the evaluation of the transition matrix elements. The original Racah-Wigner angular algebra assumes however, just like the Slater-Condon rules, that the bra and the ket of a matrix element are built on a common orthonormal one-electron set. It has been shown [30,5] that this algebra still holds if the two bases, $\{\chi\}$ for the

Fig. 4. Example of the interactive Breit-Pauli calculation for the mixing of $^1P^o$ and $^3P^o$. Answers to prompts are indicated by > at the beginning of a line.

```
#  ........Breit-Pauli Calculation for Iso-electronic Sequence........
> bp_ang
 Enter ATOM, relativistic (Y/N) with mass correction (Y/N)
>1Po3Po,y,y
 Gradient or Slater form? (G/S):
>g
 Indicate the type of calculation
 0 => non-relativistic Hamiltonian only;
 1 => one or more relativistic operators only;
 2 => non-relativistic operators and selected relativistic:
>2
 All relativistic operators ? (Y/N)
>n
Spin-orbit,Spin-other-orbit,Spin-spin,Orbit-Orbit (0/1)
>1,1,1,0
 All Interactions? (Y/N):
>y
#  .....
> bp_mat
 Enter ATOM, relativistic (Y/N) with mass correction (Y/N)
>1Po3Po,y,y
 Gradient or Slater form? (G/S):
>g
 Default Rydberg constant (y/n)
>y
 Finished with the file
#  .....
> bp_eiv
 Enter ATOM, relativistic (Y/N) with mass correction (Y/N)
>1Po3Po,y,y
 Gradient or Slater form? (G/S):
>g
  Enter Maximum and minimum values of 2*J
>4,0
 Enter eigenvalues: one line per term, eigenvalues separated by commas
2*J =  4
>1
2*J =  2
>1,2
2*J =  0
>1
 Default Rydberg constant (y/n)
>y
 :: Allocating memory for Block 2J =  4
 :: IN MEMORY: Block 2J =  4 with  594  matrix elements
 Starting Davidson
    ...
 Finished with the file
-------------------------------------------------------------------------
```

bra and $\{\phi\}$ for the ket, form a biorthonormal system

$$\int d\mathbf{r}\, dm_s\ \hat{\chi}_p^*(\mathbf{r}, m_s)\phi_q(\mathbf{r}, m_s) = \langle \hat{p}|q\rangle = \langle 0|\hat{a}_p a_q^\dagger|0\rangle = \delta_{pq}$$

The program `biotr` may be used for this purpose. Since this important application has not been documented previously, the program structure will be explained. Theory and notations can be found in [5].

The structure of `biotr` is rather simple. To get the adequate representation of initial and final states, with the needed biorthonormality property that allows the evaluation of the transition matrix elements using the original "orthogonal" Racah-Wigner algebra, the core routine BIOTRN

(1) obtains the biorthogonal forms of the initial and final radial functions, then
(2) counter-transforms the configuration interaction coefficients,

according to [5].

The counter-transformation of MCHF or CI eigenvectors, ie. the counter-transformation of the CSF expansions in the transformed one-electron basis resulting from step 1, requires knowledge of the coupling coefficients $A_{ij}^{\mu\nu}$ appearing in the expression of the excitation operator action $a_i^\dagger \hat{a}_j$ on the configuration state functions $\{\Phi_\nu\}$, namely

$$a_i^\dagger \hat{a}_j|\Phi_\nu\rangle = \sum_\mu A_{ij}^{\mu\nu}|\Phi_\mu\rangle.$$

As pointed out in [5], these coefficients are proportional to the angular coefficients of the one-electron integrals L_{n_il,n_jl} mapping the CSF space (see Appendix A of [5]). Therefore, the main program (TRANS) starts to evaluate these coefficients for both the initial and final states through a call to subroutines NONH1 $\rightarrow$ ANGMOM $\rightarrow$ LMATRIX. These coefficients are dumped onto scratch files.

Taking advantage of orthogonality of spherical harmonics, the above two-step treatment can be done separately for each l-value. But the shell-ordering of the basis by increasing angular momentum does not arise naturally from the physical description of an atomic state. Some record-keeping needs to be done first. Therefore, the routine RASIN is called for building up a vector `elras(k*nwd)` containing, for each state (k=1 or 2), the electron labels $\{nl\}$ of the occupied shells, according the following hierarchy

```
do l = 0,lmax
   do n = l+1,nmax
```

```
    end do
```

This is done, after analyzing the two `.c` files in subroutine CFGIN2. The READW2 subroutine extracts the two sets of radial functions from the `.w` files and gets the position of the biorthonormal shells `elras` in the original vector of radial functions `P`. The radial overlap matrix **S** between the two corresponding `.w` files can then be calculated through a call to subroutine BRKT. At this stage, the matrix is block-diagonal thanks to the sorting in l, but each block is not necessarily square, depending on the correlation models used for describing the initial and final states. TRANS calls EIGVEC for reading the eigenvectors (either from the `.l` files in the non-relativistic scheme, or from the `.j` files in Breit-Pauli calculations) after which everything is known for performing the needed transformations in BIOTRN.
In BIOTRN, BIOTRN_MEM is called to compute the sizes of the needed arrays. The two-step transformation described above is integrated in a loop over the l-angular momentum values. Finding the triangular transformation matrix is equivalent to lower and upper (LU) triangular matrix decomposition, as shown in [31]. The special choice of a UL decomposition of the inverse overlap matrices defines biorthonormal bases obtained by upper-triangular orbital transformation matrices. For the current l-value, BIOTRN performs the overlap matrix decomposition into block-triangular factors and finds the new radial functions satisfying

$$\int_0^\infty P^{\chi}_{nl} P^{\phi}_{n'l} = \delta_{nn'}$$

A special treatment is adopted according to Appendix B of [5], when the numbers of orbitals on the left- and right-hand side (in the same l-space) are different.

BIOTRN then proceeds to the second step, performing the eigenvector transformations for initial and final states, respectively. The operator $\hat{s}$, as defined by Eq.(45) of [5], is built in routine PAMTMT. Its exponentiation

$$\sum_{N=0}^{2(2l+1)} \frac{1}{N} \hat{s}^N$$

appearing in eq. (47) of [5], is performed in CITRA that calls TI1TV and TIINI for evaluating the action of the operator on a set of vectors in the diagonal and off-diagonal cases, respectively. In these two routines, GTRAC1 $\rightarrow$ GTRACXVN gets the needed $A^{\mu\nu}_{ij}$ coefficients.

After BIOTRN has completed all transformations, control is transferred back to the main (TRANS) program, a call to BRKT is performed to confirm the success of the radial orbital transformation resulting in biorthonormality. Once these transformations have been completed the calculation of transition amplitudes and probabilities can proceed as in the orthogonal scheme but remembering that the bra $\{\chi\}$ and ket $\{\phi\}$ one-electron functions refer to different radial spaces. RADINT evaluates the needed radial transition integrals in the biorthonormal basis. CALCUL then proceeds to the final calculation of the length (and velocity) line strengths using the transformed eigenvectors for each pair. The oscillator strengths and transition probabilities are evaluated in PROBAB, using the formula of reference [28]. Several types of electric and magnetic multipole transitions (E1, M1, E2, M2, . . .) between all states in the “Initial” and all states of the “Final” groups are computed with transitions reordered so that the reported data always is for a transition from a lower energy to a higher energy. The output file is identified by the names of the two cases, say `name1` and `name2`. Either non-relativistic or relativistic calculations may be performed: the former requires that `.l` files exist for both sets and produces a file `name1.name2.ls` whereas for a relativistic calculation `.j` files need to exist and the file produced is `name1.name2.lsj`.

As shown in [5], the transformations performed in BIOTRN assume the expansions are “closed under de-excitation”. From a practical point of view, this means, that if a CSF is present in the expansion, a corresponding CSF obtained by replacing an nl orbital by an ml orbital, where $m < n$, should also be in the expansion.

For the calculation of transitions in iso-electronic sequences where angular integrations need only be performed once, the calculation of angular data is performed by `biotr_ang` and the transition calculation by `biotr_tr`. These are the only applications that use the Fortran90 libraries.

The `biotr` program computes both length and velocity forms for all E1 and E2 transitions and is the preferred program.

3.6 Isotope Shifts and Hyperfine Effects: `isotop[e` *and* `hfs`

The programs `isotope` and `hfs` are similar to the earlier versions ISO [32] and HFS [33] but converted to the present method of angular computation. The former computes the effect of the nuclear motion and the nuclear volume on the energy level. Shifts between masses are also computed. The hyperfine program computes the magnetic dipole and electric quadrupole constants from MCHF wave functions. These programs are also described in [2].

Fig. 5. Example of the calculation of forbidden transitions between the states of the $2s2p\ {}^{1,3}P^o$ named 1Po3Po using the trans program.

```
#  ........An E2 transition between two odd parity states........
  Name of Initial State
1Po3Po
  Name of Final State
1Po3Po
  intermediate printing (y or n) ?
n
  transitions only for E(initial) < E(final) (y or n) ?
y
 Default Rydberg constant (y/n)
y
  Relativistic calculation ? (y/n)
y
  Type of transition ? (E1, E2, M1, M2, .. or *)
E2
  Use existing file for angular data ? (y/n)
n
#
#  ........Display of  the output file of trans for this transition
>cat 1Po3Po.1Po3Po.lsj
  Transition between files:
  1Po3Po
  1Po3Po


   4  -50.89325980  1s(2).2s_2S.2p_3P
   2  -50.59531993  1s(2).2s_2S.2p_1P
   65387.68 CM-1      1529.34 ANGS(VAC)      1529.34 ANGS(AIR)
 E2  S =  5.05274D-06   GF =  2.37176D-13   AKI =  2.25468D-04


   2  -50.89389850  1s(2).2s_2S.2p_3P
   4  -50.89325980  1s(2).2s_2S.2p_3P
     140.17 CM-1    713409.99 ANGS(VAC)    713336.24 ANGS(AIR)
 E2  S =  1.76493D+00   GF =  8.16145D-16   AKI =  2.13925D-12


   2  -50.89389850  1s(2).2s_2S.2p_3P
   2  -50.59531993  1s(2).2s_2S.2p_1P
   65527.85 CM-1      1526.07 ANGS(VAC)      1526.07 ANGS(AIR)
 E2  S =  8.83934D-06   GF =  4.17594D-13   AKI =  3.98682D-04


   0  -50.89417639  1s(2).2s_2S.2p_3P
   4  -50.89325980  1s(2).2s_2S.2p_3P
     201.16 CM-1    497119.62 ANGS(VAC)    497068.23 ANGS(AIR)
 E2  S =  7.84321D-01   GF =  1.07193D-15   AKI =  5.78653D-12
```

With computer speeds increasing and memory expanding rapidly, the need for parallel computing is reduced, but angular integrations that often are the computational bottleneck exhibit near perfect parallelism when tasks are distributed by columns. No communication is required except at the initialization or finalization stage [34]. In a distributed memory environment the message passing interface (MPI) is now often used. In such an environment, if each processor writes to a different disk, parallel I/O can also be achieved. With the use of 64 or 128 processors, many hours of CPU time are reduced to minutes. The table below shows which serial codes have an mpi version.

Serial	Parallel
`nonh`	`nonh_mpi`
`mchf`	`mchf_mpi`
`bp_ang, bp_mat, bp_eiv`	`bp_ang_mpi, bp_mat_mpi, bp_eiv_mpi`
`biotr_ang, biotr_tr`	`biotr_ang_mpi, biotr_tr_mpi`

Atomic structure calculations have underlying loops that are related to the number of configuration states. For example, the calculation of the matrix elements of an interaction matrix has the `DO` loop structure

```
DO column = 1, ncfg
    Do row = 1, ncfg
       ...
    End do
End do
```

Such a calculation can easily be parallelized by changing the outer loop to

```
DO column = 1+myid, ncfg, nprocs
```

where `myid` is the rank of the MPI process and `nprocs` is the number of processors. The starting process has rank of zero and others range from `1, ... , nprocs-1`. Then the starting column for a given process, is `1+myid`, the next are `1+myid+nprocs`, `1+myid+2nprocs`, etc. In such an implementation tasks are assigned by columns. Even though columns are not of uniform length, excellent performance has been found for angular integrations and configuration interaction calculations [8,9]. Each processor reads/writes from its own files with names identified by an extension that represents the rank, performs a computation, and performs global operations such as summing all results or communicates results to other processors. On clusters, the potentially huge files of angular data or the interaction matrix in a configuration interaction calculation preferrably should be stored on local disks in order to avoid network transfers. But on other systems these files may need to be stored on large, shared disks. With many clusters having nodes with multiple processors, it may be that two processors are on the same node but the present MPI version considers them as independent processors.

The MPI programs have been used primarily on an IBM SP2 or SP3 and to a lesser extent on a Red Hat LINUX cluster.

4 Utilities: `comp`, `condens`, `lsreduce`, `LS_trends`, `tables`, `T_dependence`, `relabel`, `select`, `w_format` and `w_unformat`

In addition to the applications, there are a number of utilities (short programs) that can be useful.

(1) `comp`: Given the `name` of a set of files, and a tolerance, this program displays the composition of each state in the file by listing the CSFs and their expansion coefficients in order of decreasing magnitude provided the magnitude is greater than the tolerance. The `name.c` option usually may not apply (seen `condens` below). This composition information may be of importance in determining the multi-reference set or the extent of correlation. It is also essential for resolving naming conflicts.

(2) `condens`: Given the `name` of a case, the source for the expansion coefficients (usually .l or .j) and whether results are to be sorted, the program produces a file `cfg.out` that includes only those CSFs for which $\sqrt{\sum_i c_i^2} \geq T$, where T is the cut-off tolerance. Since `mchf` no longer produces a `cfg.out` with the expansion coefficients included, the `name.c` option cannot usually be applied. The coefficients are included in the condensed output file, `cfg.out`. To restore this file to the present .c format (without expansion coefficients), move the file to `clist.inp` and run `lsgen` selecting the “r” mode (for restore). Note that condensing may destroy the “closed under de-excitation” requirement of `biotr` but when the tolerance is sufficiently small, no problems have been encountered. In a systematic method, where maximum principal quantum number is increased by unity from one iteration to the next, it is unlikely that a CSF with a high n will remain and the equivalent CSF with a lower n be deleted.

(3) `lsreduce`: When expansions are generated, there is no attempt made to assure that all couplings of the configuration states are important. From a practical point of view we may partition the wave function into the zero-order approximation defined by the set of CSFs called the “multireference set” and then require that all the remaining CSFs interact with at least

one member of this multireference set. The program `lsreduce` performs this task provided the multireference set is given in `lsgen` format in the file `mrlist`. `lsreduce` will take a `cfg.inp` file and produce a `cfg.out` file in which every member interacts with at least one CSF in the `mrlist` file. The latter should include all CSFs that are important in the wave function expansion as determined from `comp`. This process can greatly reduce the size of an expansion.

(4) `LS_trends`: This program reads all the `.ls` files in a directory of the form `name1.name2.${z}_${n}.ls` and produces a file for each Z, lists the energy of the initial and final state, length and velocity forms of the line strength and gf value, and the error, $|gf(L)-gf(V)|/\max(gf(L), gf(V))$ as a function of n. These are accuracy indicators that may be useful in monitoring a calculation: the total energies should decrease with n, and the length and velocity should be converging as n increases as shown in the following example:

```
-------------------------------------------------------------------------------------
 Z  n       EL            EU         S(L)       S(V)       gf(L)     gf(V)   Error
-------------------------------------------------------------------------------------

3s(2).3p(4)1S0_1S 3s(2).3p(3)2D3_2D.3d_1P
 18 5 -525.3315067 -524.8330645 4.216e-04 3.300e-04 1.401e-04 1.097e-04 0.217
 18 6 -525.3337060 -524.8374088 2.815e-04 1.144e-03 9.315e-05 3.785e-04 0.754
 18 7 -525.3343408 -524.8385308 2.171e-04 9.919e-04 7.175e-05 3.279e-04 0.781

3s(2).3p(4)1S0_1S 3s(2).3p(3)2P1_2P.4s_1P
 18 5 -525.3315067 -524.5203280 3.216e-01 3.398e-01 1.739e-01 1.837e-01 0.054
 18 6 -525.3337060 -524.5247768 3.189e-01 3.246e-01 1.720e-01 1.750e-01 0.017
 18 7 -525.3343408 -524.5260028 3.235e-01 3.249e-01 1.743e-01 1.751e-01 0.005

3s(2).3p(4)1S0_1S         3s.3p(5)_1P
 18 5 -525.3315067 -524.4720362 3.255e-01 3.541e-01 1.865e-01 2.029e-01 0.081
 18 6 -525.3337060 -524.4768068 3.249e-01 3.508e-01 1.856e-01 2.004e-01 0.074
 18 7 -525.3343408 -524.4782244 3.228e-01 3.464e-01 1.843e-01 1.977e-01 0.068
-------------------------------------------------------------------------------------
```

Shown here is transition information for three transitions from $3s^23p^4\ {}^1S$ in Ar III (Z=18) for $n = 5, 6, 7$ expansions. Total energies are decreasing for both states, the length form of the line strength is converging, but the discrepancy between length and velocity remains large when the line strength is nearly zero. Thus for some transitions, the results are excellent but for others, they at best indicate the order of magnitude of the line strength.

The `LS_trends` program does not show the convergence of the transition energy. The program `FLS_trends` [10] reads the files produced by `LS_trends` and displays also the transition energy in cm^{-1}, for easy comparison with observation. When different groups have overlapping terms, the output from `LS_trends` may contain multiple lines. `FLS_trends` selects the calculation in which the total energy is lowest for both the initial and final states, for a given n.

(5) `tables`: This is a program written in C++ that takes a `name.lsj` file, usually a concatenated file of all the `.lsj` transition files for a given atom or ion, and finds the energy level structure of the levels and the multiplet transition arrays. The tables posted at the website `http://atoms.vuse.vanderbilt.edu` are examples of tables produced by the `tables` program. When an energy level is present in the `.lsj` file with **two** different energies, the higher level is considered to be unphysical. It and all data associated with this level are removed from the table. The user may also specify certain levels as "unphysical" in which case they will be removed. Finally, the program computes the lifetimes of the levels from the transition data provided in the file.

(6) `T_dependence`: Given a `.c` and a corresponding `.j` file, this program displays the term dependence of each state included in the file [10]. This gives an indication of LS term mixing in the wave function of a state. The Breit-Pauli programs all assign labels to states according to the largest expansion coefficient. When this process produces the same label for two different states a careful analysis is needed. The LS value should be that term with the largest composition and within that LS, the largest expansion coefficient identifies the label. For more on labeling, see [11].

(7) `relabel` This utility reads radial functions in turn from `wfn.inp`. For each function, the user may enter either a blank, "d", or a new 3-character label for which the respective action is to write the radial function unchanged into `wfn.out`, skip (or delete) the radial function, write the radial function changing the displayed label to the entered label.

(8) `select` This routine selects those CSFs from a designated list that contain orbitals specified by the user. This may be useful when expansions that have been condensed are extended with new CSFs to be included to first-order. The selected CSFs may be appended to the condensed list.

(9) `w_format and w_unformat`: Binary file formats are not always compatible in going from one system environment to another. `w_format` takes a `wfn.inp` file (in .w format) and produces a formatted `wfn.fmt` file. The program `w_unformat` reads `wfn.fmt` and produces the binary file (in .w format) `wfn.out`. Some loss of accuracy can be expected in the process but the radial functions are tabulated to 11-digits of accuracy.

5 Libraries

All applications are developed around libraries that can be broadly classified as angular, radial, common, or special libraries.

An angular library of routines has been developed based on second quantization in coupled tensorial form, angular momentum theory in three spaces (orbital, spin and quasispin), and graphical techniques of angular momentum [35–37,13]. These angular routines extend the possible configuration states to include partially filled f-shells. A complete description of the library is presented in [16]. Unlike the earlier code, the present version allows for as many as eight (8) shells outside the set of common closed shells.

The radial library (MCHF_LIB_RAD) is similar to the library described in [38] except that pointer variables point to arrays whose size is determined by the number of orbitals in the application and only one library is needed in all cases. To illustrate the change, consider the declarations in the subroutine ZK in the previous version:

```
      PARAMETER(NOD=220,NWD=30)
      COMMON /PARAM/H,H1,H3,CH,EH,RHO,Z,TOL,NO,ND,NWF,MASS,NCFG,IB,IC,ID
     :   ,D0,D1,D2,D3,D4,D5,D6,D8,D10,D12,D16,D30,FINE,NSCF,NCLOSD,RMASS
      COMMON /RADIAL/R(NOD),RR(NOD),R2(NOD),P(NOD,NWD),YK(NOD),
     :   YR(NOD),X(NOD),AZ(NWD),L(NWD),MAX(NWD),N(NWD)
```

Note the parameter NWD=30. Because in transition applications the number of radial functions could double, in theory, a second library was needed for the case where NWD=60. In the dynamic version the equivalent declarations became

```
      SUBROUTINE ZK(I,J,K)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER(NOD=220)
      COMMON /PARAM/H,H1,H3,CH,EH,RHO,Z,TOL,NO,ND,NWF,MASS,NCFG,IB,IC,ID
     :   ,D0,D1,D2,D3,D4,D5,D6,D8,D10,D12,D16,D30,FINE,NSCF,NCLOSD,RMASS
      COMMON /RADIAL/R(NOD),RR(NOD),R2(NOD),YK(NOD),YR(NOD),X(NOD)
      POINTER(IQP,P(NOD,1)),(IQN,N(1)),(IQL,L(1)),(IQAZ,AZ(1)),
     :       (IQMAX,MAX(1))
      COMMON/NEL/IQP,IQN,IQL,IQAZ,IQMAX,IQ(7)
      DIMENSION F(NOD),G(NOD)
```

The COMMON /PARAM/ is unchanged since it contains a series of single variables, but COMMON /RADIAL/ has been divided. One dimensional arrays of dimension 220, are not of sufficient size to warrant making dynamic. In this case, it was decided to allocate all arrays associated with the variable NWF, the number of one-electron wave functions or orbitals. The new COMMON /NEL/ now only contains the pointer values. On many architectures, the pointer data type is like an integer (32 bits) but on others, like the DEC Alpha and newer 64 bit PCs, it is 64-bits. For this reason, all pointers in common, whether used or not, need to be delared explicitly as type pointer. Note also that two local arrays have been added, namely F(NOD),G(NOD). These arrays were used to improve the vectorization of the routine, used in an exchange calculation, since in an

MCHF calculation a great deal of time is spent computing exchange functions.

The common library (MCHF_LIB_COM) is essentially unchanged [38] except that three platform dependent routines have been added:

`alloc`	allocates memory for a number of data elements each of length nbytes.
`dalloc`	deallocates the memory associated with an array
`realloc`	reallocates the size of the array

During the last decade, a number of variations were in common usage, but these have now been reduced to essentially two, which we refer to as `LINUX` or `IBMSP`. At issue is how `call malloc` is implemented, whether in the Fortran style of call-by-reference or the C style of call-by-value supported by the xlf90 compiler under AIX.

There also are three libraries of routines of a more general nature.

(1) `libdvdson`: This is a library of routines for the Davidson method for finding selected eigenvalues and eigenvectors of a symmetric matrix [3].
(2) `libnet`: This is a library of routines used by the atomic structure package that are part of the Blas and Lapack Libraries [6]. These routines should be replaced by platform dependent versions often available with the compiler.
(3) `libmpi`: This library contains some system dependent routines for developing a parallel version of the application using the message passing interface, MPI [39]. This is not a general purpose library and may need modification to adhere to cluster design and policy.

6 Installation

The tar file for the application contains a directory with several subdirectories.

(1) `src`: This directory contains fixed format code (essentially F77 with some Fortran90 statements, but no modules). There are separate subdirectories for libraries, application, and utilities. Some of the utilities (`LS_trends, tables`) are written in C++.
(2) `src90`: This directory contains a Fortran90 `.f90` version of the libraries designed around the use of modules. Initial versions of these routines were obtained by using the VAST/77to90 translator [40] with subsequent modifications. Only two applications, `biotr_ang, biotr_tr`, have been developed using this library. Both serial and parallel versions are available.
(3) `examples`: This directory contains executable script files for determining the E2 and M1 forbidden transitions between all levels of $3s^2 3p^2$ (^3P, ^1D,

and ^1S) for Z=14 and 15 for expansions over orbital sets n=4, 5, and 6. A subdirectory contains output results for comparison.

A simple `make` command from either the `src` or `scr90` directory, will compile applications in the respective directories. This process relies on environment variables. The following table lists the variables, gives a definition, and follows this by an example when the Portland Group Fortran 90 compiler for a Linux system is used.

Variable	Definition
`FC`	Name of the compiler
`FC_MPI`	Name of the compiler for MPI code
`FC_FLAGS`	Flags to be used with the `FC` compiler.
`FC_MPIFLAGS`	Compiler flags for the MPI compiler.
`FC_LD`	Loader flags for the Fortran compiler.
`FC_MPILD`	Loader flags for the MPI compiler.
`FC_MALLOC`	Version to be used for memory allocation
`CPP`	Name of C++ compiler
`CPP_FLAGS`	Flags to be used with the C++ compiler
`CPP_LD`	Flags to be used with C++ loader
`LAPACK_DIR`	Directory containing the LAPACK and BLAS libraries
`LAPACK_LIB`	Libraries to be searched in `LAPACK_DIR`
`MPI_TMP`	Location of distributed (temporary) files in MPI runs
`ATSP`	Directory where a `bin` and `lib` subdirectories should be created to save the executable applications and the compiled libraries, respectively.

Two versions of `FC_MALLOC` are supported:

- IBMSP This version has been used for the IBM AIX operating system.
- LINUX This is the default version for 32-bit architectures of LINUX and many UNIX operating systems. This environment variable is used also whenever difference between the LINUX and IBMSP environments are encountered and not only for memory allocation.

The following commands, when included in the `.cshrc` file, for example, and when using the Portland Group `pgf90` compiler, will provide the needed environment for compilation.

```
setenv ATSP /home/${USER}/atsp2K
set path = (${ATSP}/$path )

# Compiler variables
setenv FC "pgf90"                        # Fortran compiler
setenv FC_MPI "mpif90" # MPI compiler
setenv FC_FLAGS "-O2 -byteswapio" # Serial code compiler flags
setenv FC_MPIFLAGS "-O2 -byteswapio" # Parallel code compiler flags
setenv FC_LD "-Bstatic" # Serial F90 linker flags
setenv FC_MPILD "-Bstatic" # Parallel F90 linker flags
setenv FC_MALLOC LINUX # memory allocation routine
setenv CPP "g++" # C++ compiler
setenv CPP_FLAGS "-O3" # C++ compiler flags
setenv CPP_LD "-static" # C++ linker flags

# mpi locaton of distributed files
setenv MPI_TMP "/tmp/$USER"

#Lapack libraries
setenv LAPACK_DIR /usr/pgi/linux86/5.0/lib
setenv LAPACK_LIB "-llapack -lblas"

#When Lapack libraries are not available use:
setenv LAPACK_DIR ${ATSP}/lib
setenv LAPACK_LIB "-lnet"
```

Once the environment variables have been set, the `make` command may be issued from `src` or `src90` directory. The makefile checks whether the `bin` and `lib` directories need to be created, creates them if necessary, and then makes the libraries, the applications, and the utilities, in turn respectively. If problems are encountered, it is advisable to make each of the libraries in turn by entering the source directory of the library and issuing the `make` command, confirming that a file in the expected directory has been created. Then a similar procedure can be applied to each application. If the source of the problem has been identified, it is possible to issue the ”make clean” command from the `src` directory and start over again. Attempts will be made to compile the MPI versions, but any failures here do not affect the serial applications. Some compilers are not able to compile `lsgen.f` because of the many nested Do-loops. In some instances, optimization should be reduced and compiler options used to increase the depth of nesting. For Linux systems, the free `g77` compiler can be used.

It is important to logoff and log on again in order to get new executables onto a search path before testing examples. As a check on the installation, shell scripts in the examples directory may be executed. These scripts, however, should also be viewed as examples of replies to prompts that can be issued interactively by the user.

Further comment on the MPI codes is in order. The MPI versions were designed for a 4 node Linux cluster but have also been used on the IBM SP2 and SP3. The MPI design implements two types of files: permanent files, which typically are stored in the starting directory, and distributed data files which often are temporary in the sense that they are not needed after all calculations have been completed successfully. These are typically large files that are distributed in this implementation. In order to accommodate the various ways of distributing these files, each MPI application starts with a call to `mpi_work_dir(startdir, permdir, tmpdir)` to determine the names of these three types of directories. This routine is found in `libmpi`, one of the special libraries. `startdir` is the directory from which the process was started, called the *current working directory* or `cwd` and is determined by a system call. The `permdir` is the directory in which the permanent (non-distributed) files of a calculation are to be found and written. The `tmpdir` directory (or directories) is the directory where distributed input or output files are located for a given process. With the `FC_MALLOC` environment variable set to `IBMSP`,

`permdir=startdir` and `tmpdir=startdir/tmp_mpi`.

Thus the distributed files are a subdirectory of the starting directory. Such calculations are typically started from a large scratch disk. This routine invoked when `FC_MALLOC` is `LINUX`, also has the permanent directory as the starting directory, but the directory for the temporary files is determined by the environment variable `MPI_TMP`. This could be a large scratch disk but then reading and writing of large distributed files would be done over the network. Such traffic can avoided if the distributed files are on the local disk of each process in which case the read/write operations require no special hardware for execution in parallel. On most clusters each node/processor has its own `/tmp/$USER` directory in which case the `MPI_TMP` variable should be set as shown in our example of environment variables. In order to identify the MPI process that generated a file, the file name is extended with a suffix suffix `*.xxxx`, where `xxxx` is a 4 digit number equal to the rank of the processor. When a sequence of applications is executed, it is important that a process of given rank, has the correct data files on its local disk. For ths distributed method it is recommended that MPI programs be initiated using the `-ch_p4` option. This ensures that during each step the processor rank does not change and the appropriate files are on the required local disk.

Additional information in the form of manuals describing the use of code and details of program design can be found at `http://atoms.vuse.vanderbilt.edu` under the link *Programs and Methods* and the sublink *Using atsp2K*. In some sense, this online document is a historical record of how the code was adapted to a variety of computing environments – Sun, Dec Alpha, Cray T3E, Linux, IBMSP – over the last decade. Under the section on testing the installation, sample output is provided. A wealth of information is provided also on the computational process. For each individual application, the purpose of the code is stated, the program structure displayed, the input data needed, and the output data produced. The present code described here is the final version of this evolving project, simplified to just two types of platforms.

7 E2 and M1 transitions between levels of $3s^23p^2$ for $Z = 14$ and 15

In the examples directory included with the code are scripts for a complete calculation of E2 and M1 transitions between levels of $3s^23p^2$. The scripts are set up for only the first two members of the iso-electronic sequence but show that it is easy to extend the calculation to higher members. The `sh_example_ls` is the script that performs LS calculations, both of wave functions and transitions. The former are stored in the directory `e1` and the latter in `tr`. `FLS_Si_Z_14` contains the trends for LS allowed ^1D – ^1S E2 transitions as a function of the expansion and shows that the length has stablized at $n = 5$ whereas the velocity form is still increasing with length and velocity differing by 14% at $n = 6$.

Relativistic effects are included by running the `sh_example_lsj` script. The two steps are separated partly because it is advisable to review the LS results prior to performing the LSJ, and partly because a manual step is needed for our script. In the directory `e1`, the last set of expansions in the file `cfg.inp` should be copied to `CI.c` and editted. Whereas the former is viewed as three different CSF expansions for three different LS terms, the Breit-Pauli calculation views the CSFs as *one* expansion. In the `cfg.inp` each expansion starts with a header (possibly blank) and a list of common closed shells and terminates with an asterisk `*`. By searching for `*` and deleting this asterisk and the following header and list of closed shells but leaving the final asterisk, the expansion becomes one list for all terms. The Breit-Pauli script uses the `bp_ang, bp_mat, bp_eiv` sets of code which is appropriate for an iso-electronic sequence, but uses more disk space. The results for LSJ transitions are found in the directories `z14` and `z15`.

On an Intel Xeon 2.4 GHz processor and 512KB of cache, the LS calculation required 314 seconds whereas the LSJ calculations required 620 seconds.

Output is included in the examples directory.

Acknowledgements

The development of this version of ATSP2K was performed over many years at Vanderbilt with funding support from the Division of Chemical Science, Geosciences, and Bioscience, Office of Basic Energy Sciences, Office of Science, U.S. Department of Energy. The participation of G. Gaigalas was funded by a co-operative research project from the National Science Foundation under grant No. PHY-9501830. M. Godefroid thanks the Communauté française of Belgium (Action de Recherche Concertée) and the Belgian National Fund for Scientific Research (FRFC/IISN Convention) and for financial support.

References

[1] C. Froese Fischer, Comput. Phys. Commun. 128 (2000) 635.

[2] C. Froese Fischer, T. Brage and P. Jönsson, Computational Atomic Structure. An MCHF Approach (Institute of Physics Publishing, Bristol/Philadelphia, 1997)

[3] A. Stathopoulos and C. Froese Fischer, Comput. Phys. Commun. 79 (1994) 268.

[4] F.A. Parpia, C. Froese Fischer, and I.P. Grant, Comput. Phys. Commun. 94 (1996) 249.

[5] J. Olsen, M.R. Godefroid, P. Jönsson, P. Å Malmqvist, and C. Froese Fischer, Phys. Rev. E, 52 (1995) 4499.

[6] Basic Linear Algebra Subroutines (BLAS) and Linear Algebra Package (LAPACK) (`http://www.netlib.org/LAPACK`).

[7] G. Gaigalas and C. Froese Fischer, Comput. Phys. Commun. 98 (1996) 255.

[8] C. Froese Fischer, M. Tong, M. Bentley, Z. Shen, and C. Ravimohan, J. Supercomputing **8**, 117-134 (1994).

[9] A. Stathopoulus. A. Ynnerman, and C. Froese Fischer, Int'l J. Supercomputers & High Perf. Comput. **10**, 41 (1996).

[10] A. Irimia (unpublished).

[11] C. Froese Fischer and G. Tachiev, Atom. Data Nucl. Data Tables, 87 (2004) 1.

[12] C. Froese Fischer and B. Liu, Comput. Phys. Commun. 64 (1991) 406.

[13] G. Gaigalas, Z. Rudzikas and C. Froese Fischer, Atomic Data and Nuclear Data Tables 70 (1998) 1.

[14] L. Sturesson and C. Froese Fischer, Comput. Phys. Commun. 74 (1993) 432.

[15] A. Hibbert and C. Froese Fischer, Comput. Phys. Commun. 64 (1991) 417.

[16] G. Gaigalas, Lithuanian J. Phys. 42 (2002) 73-86; (physics/0405072 in `http://www.arxiv.org`).

[17] H. Tatewaki, M. Sekiya, F. Sasaki, O. Matsuoki and T. Koga, Phys. Rev. A51 (1996) 197.

[18] C. Froese Fischer, Comput. Phys. Commun. 64 (1991) 431.

[19] G. Tachiev and C. Froese Fischer, J. Phys. B. 32 (1999) 5805.

[20] G. Tachiev and C. Froese Fischer, J. Phys. B. 33 (2000) 2419.

[21] C. Froese Fischer, G. Tachiev, and A. Irimia, Atom. Data Nucl. Data Tables (submitted).

[22] J. Xi and C. Froese Fischer, Phys. Rev. A 59 (1999) 307.

[23] P. Karchenko, J.F. Babb, A. Dalgarno, Phys. Rev. A 55 (1997) 3566.

[24] P. Jönsson and S. Gustafsson, Computer Phys. Commun. **144**, 188 (2002).

[25] A. Hibbert, R. Glass and C. Froese Fischer, Comput. Phys. Commun. 64 (1991) 455.

[26] C. Froese Fischer, Comput. Phys. Commun. 64 (1991) 473.

[27] C. Froese Fischer, M.R. Godefroid and A. Hibbert, Comput. Phys. Commun. 64 (1991) 486.

[28] C. Froese Fischer and M.R. Godefroid, Comput. Phys. Commun. 64 (1991) 501.

[29] O. Zatsarinny, Comput. Phys. Commun. 124 (2000) 247.

[30] M. Moshinsky and T.H. Seligman, Ann. Phys. (N.Y.) 66 (1971) 311

[31] P. Å Malmqvist, Int. J. Quantum Chemistry 30 (1986)479

[32] C. Froese Fischer, L. Smentek-Mielczarek, N. Vaeck, and G. Miecznik, Comput. Phys. Commun. 74 (1993) 415.

[33] P. Jönsson, C-G Wahlström, and C. Froese Fischer, Comput. Phys. Commun. 74 (1993) 399.

[34] A. Stathopoulos, A. B. Ynnerman, C. Froese Fischer, Int'l J. Supercomput. Appl. High Perf. Computing 10 (1996) 41.

[35] G. Gaigalas and Z. Rudzikas, J. Phys. B: At. Mol. Phys. 29 (1996) 3303.

[36] G. Gaigalas, Z. Rudzikas and C. Froese Fischer, J. Phys. B: At. Mol. Phys. 30 (1997) 3747.

[37] G. Gaigalas, A. Bernotas, Z. Rudzikas and C. Froese Fischer, Physica Scripta 57 (1998) 207.

[38] C. Froese Fischer, Comput. Phys. Commun. 64 (1991) 399.

[39] Message Passing Interface (`http://www-unix.mcs.anl.gov/mpi/`)

[40] VAST Fortran77 to Fortran90 Automatic Translater, (`http://www.crescentbaysoftware.com/end_user.html`).