Notes on improvements in RCI

Per Jönsson
The Compas Group

February 18, 2014

1 Memory allocation and overflow

For large cases the scalar version of `rci` fails to allocate memory for the Davidson diagonalization. The reason is an overflow in integer variables.

The fix would be to redefine all relevant integer variables to `integer*8`. In the first round we look for the following variables:

```
nelmnt
nelmnt_a
nelmnttmp
nelmntt
nstore
```

There may be more variables and this needs careful attention.

Libraries

In `lib92` we have `nelmnt` in:

```
spicmv2.f:      COMMON/HMAT/PNTEMT,PIENDC,PNIROW,NELMNT
spicmv.f:      COMMON/HMAT/PNTEMT,PIENDC,PNIROW,NELMNT
```

In `mpi` we have `nelmnt` in:

```
spicmvmpi.f:      COMMON/HMAT/PNTEMT,PIENDC,PNIROW,NELMNT
```

Applications, rci

In `ric` we have `nelmnt`, `nelmnt_a`, `nelmnttmp`, `nelmntt`, `nstore` in:

```
dnicmv.f:      COMMON/HMAT/PNTEMT,PIENDC,PNIROW,NELMNT

genmat2.f:      SUBROUTINE genmat2 (irestart, nelmnt_a, elsto)
genmat2.f:*   The mpi version (genmat2mpi) also gets nelmnt_a and elsto
genmat2.f:     &  NCOREtmp, NVPItmp, NKEItmp, NVINTItmp, NELMNTtmp, NCFtmp
genmat2.f:      NELMNT_a = NELMNTtmp
genmat2.f:      DENSTY = DBLE (NELMNTtmp) / DBLE ((NCFtmp*(NCFtmp+1))/2)
genmat2.f:      WRITE (24,312) NELMNTtmp  % FORMAT STATEMENT NEEDS ATTENTION

genmat.f:      COMMON/HMAT/PNTEMT,PIENDC,PNIROW,NELMNT
genmat.f:     &  NCOREtmp, NVPItmp, NKEItmp, NVINTItmp, NELMNTtmp, NCFtmp
genmat.f:      nelmnt = 0     ! Counting continues in setham
genmat.f:! nelmnt, eav, elsto obtained (to be further modified in setham)
genmat.f:               nelmnt = nelmnt + nelc
genmat.f:         CALL setham (myid, nprocs, jblock, elsto, icstrt, nelmnt
genmat.f:         NELMNTtmp = NELMNT

hmout.f:      COMMON/HMAT/PNTEMT,PIENDC,PNIROW,NELMNT

maneig.f:      COMMON/HMAT/PNTEMT,PIENDC,PNIROW,NELMNT
maneig.f:!            IF (NELMNT .LT. NBRKEV) THEN
maneig.f:               NSTORE = NELMNT+NELMNT/2+(NCF+1)/2
maneig.f:         print *, ’nelmnt = ’, nelmnt
maneig.f:                  CALL ALLOC (PNTEMT,NELMNT,8)
maneig.f:                  CALL ALLOC (PNIROW,NELMNT,4)

matrix.f:     :      /HMAT/PNTEMT,PIENDC,PNIROW,NELMNT
matrix.f:*           ...eav, nelmnt are also obtained from genmat
matrix.f:            CALL genmat2 (irestart, nelmnt_a, elsto)
matrix.f:         CALL genmat2 (irestart, nelmnt_a, elsto)

setham_gg.f:      SUBROUTINE SETHAM (myid, nprocs, jblock, ELSTO,ICSTRT, nelmntt
setham_gg.f:     :      /HMAT/PNTEMT,PIENDC,PNIROW,NELMNT
setham_gg.f:     &  NCOREtmp, NVPItmp, NKEItmp, NVINTItmp, NELMNTtmp, ncftmp
setham_gg.f:      nelmnt = nelmntt
setham_gg.f:         NELMNT = NELMNT + NELC
setham_gg.f:      NELMNTtmp = NELMNT
```

We need to pay careful attention to all this and also see if there are integers defined that in turn require other changes.

Applications, rscf

In `rscf` we have `nelmnt` in:

```
hmout.f:      COMMON/HMAT/PNTEMT,PIENDC,PNIROW,NELMNT

maneig.f:     :      /HMAT/PNTEMT,PIENDC,PNIROW,NELMNT

matrix.f:      COMMON/hmat/pntemt,piendc,pnirow,nelmnt
matrix.f:      READ (30) nelmnt
matrix.f:      CALL alloc (pnirow, nelmnt, 4)
matrix.f:      CALL alloc (pntemt, nelmnt, 8)
matrix.f:      DO i = 1, nelmnt
matrix.f:     &           (irow(i), i = 1, nelmnt)

setcof.f:      COMMON/HMAT/PNTEMT,PIENDC,PNIROW,NELMNT
setcof.f:         READ (30) NELMNT
setcof.f:         CALL alloc (PNIROW, NELMNT, 4)
setcof.f:     &             (IROW(I), I = 1, NELMNT)
setcof.f:            READ (30) NELMNT
setcof.f:            CALL alloc (PNIROW, NELMNT, 4)
setcof.f:     &                (IROW(I), I = 1, NELMNT)

setham.f:      COMMON/HMAT/PNTEMT,PIENDC,PNIROW,NELMNT
setham.f:            IF (LOC .GT. NELMNT) THEN
setham.f:               PRINT *, ’  LOC = ’, LOC, ’  NELMNT = ’, NELMNT
setham.f:            IF (LOC .GT. NELMNT) THEN
setham.f:               PRINT *, ’  LOC = ’, LOC, ’  NELMNT = ’, NELMNT
```

2 Implemented changes

The changes have been implemented at Monster in `/home/per/programs/grasp2k_light_2014-02-12`

3 Breit integrals

The computation of Breit integrals have been identified as a bottleneck for scalar, and even more so for parallel, `rci` calculations. The computation of Breit integrals are inside the double loop over CSFs in `setham` by means of calls to `brint1`, ..., `brint6`. For every interaction between two CSFs there is a call to `brint1` that loops through the full integral list. If the integral is not available then:

1. space is allocated to keep the integral
2. the integral is computed
3. the integral is stored

When the integral list is long this takes forever.

We intend to recode this so that all the Breit integrals are computed in advance as is done for the Rk integrals. This is a two-step procedure implemented in `genintrk`:

1. loop through the orbital set and count the maximum number of integrals
2. allocate space for all the integrals
3. loop through the orbital set, compute and store the integrals