Reverse Communication Interface in ELSI

Yingzhou Li, Jianfeng Lu, and ELSI Team
Duke University
Eigenvalue Problem in KS DFT

\[ H\psi = \lambda S\psi \]

- Here \( H \) is the Hamiltonian matrix, \( S \) is the overlapping matrix, \( \lambda \) is an eigenvalue and \( \psi \) is the corresponding eigenvector.
- In Kohn-Sham DFT, smallest \( k \) eigenpairs are needed.
- In practice, the size of the Hamiltonian matrix is large and \( k \) is proportional to the Hamiltonian size.
- Solving the eigenvalue problem is required in each self-consistency iteration.
ELectronic Structure Infrastructure: ELSI

- ELSI provides an stable interface layer between KS-DFT codes and various solvers (ELPA, libOMM, PEXSI, SIPs…)
- Matrix format conversion is performed automatically by ELSI
- ELSI and redistributed solvers supports five major compilers (Intel, GNU, IBM, PGI, Cray)
- Reverse Communication Interface for Iterative Eigensolvers.
  - Currently, we have Davidson, OMM and PPCG implemented.
Davidson Method

- $\Psi$
- Solve Rayleigh Ritz problem $(\Psi^* \mathbb{H} \Psi, \Psi^* \Psi)$ for smallest eigenpairs $\Lambda, Q$
- $R = \mathbb{H} \Psi Q - \Psi Q \Lambda$
- If $\|R\| < tol$, converged
- Approximately solve $(H - \Lambda_i I)V_i = R_i$ for all $V_i$

$\Psi = [\Psi ~ V]$
Orbital Minimization Method

- Solve an unconstrained minimization problem
  \[
  \min_{\Psi} \text{tr}( (2I - \Psi^*\Psi)\Psi^*H\Psi )
  \]
- with conjugate gradient method and exact line search.

- \( \Psi_m \)
- \( G_m = \mathcal{P} \nabla f_{obj}(\Psi_m) \)
- \( D_m = G_m + \beta_m G_{m-1} \)
- \( \Psi_{m+1} = \Psi_m + \gamma_m D_m \)
Projected Preconditioned CG

[Vecharynski, et al 2015]

• Solve a constrained minimization problem

\[
\min_{\Psi \Psi = I} \text{tr}(\Psi^* H \Psi)
\]

• in the subspace of wavefunctions, gradient and conjugate gradient direction.

• \(\Psi_m\)

• \(G_m = P2(I - \Psi_m \Psi_m^*)H\Psi_m\)

• Minimize the objective with \(\Psi_{m+1} = \Psi_m C_{\Psi} + G_m C_G + D_m C_D\) and \(C_?\) is diagonal

• \(D_{m+1} = G_m C_G + D_m C_D\)

Solve Rayleigh Ritz problem for \(\Psi_{m+1}\) every few iterations
Reverse Communication Interface

**Initialization**

- The instruction is accomplished

**Do as the instruction**

- Converged

**Post-calculation**

- An instruction

**RCI**

**Initialized**
Reverse Communication Interface

Zone of Matrices & MPI
- Initialization
- Post-calculation

Zone of Scalars & Sequential
- Do as the instruction

RCI

ELSI-RCI’s responsibility
Instruction Data Structure

type, public :: rci_instr

character :: jobz, uplo, side ! job char; upper or lower; left or right
character :: trH, trS, trP, trA, trB ! Operation for H, S, P, A, B

integer :: m, n ! size of the output matrix
integer :: k ! size of the intermedia multiplication
integer :: lda, ldb, ldc ! leading size for matrix A, B and C

integer :: rAoff, cAoff ! row and column offset of A
integer :: rBoff, cBoff ! row and column offset of B

integer :: Aidx, Bidx, Cidx ! indices for matrix A, B and C

real(r8) :: alpha, beta ! coefficients

end type
Initialization

- Allocate memory (RCI)

- Initialize the initial wave functions
  - All tests of iterative methods currently use random initial guess
Common Instructions

- RCI_NULL
  - Null

- RCI_CONVERGE
  - Converged Flag

- RCI_STOP
  - Stop Flag

- RCI_ALLOCATE
  - Allocate a matrix

- RCI_DEALLOCATE
  - Deallocate a matrix

- RCI_H_MULTI
  - $B = op(H) \ast A$

- RCI_S_MULTI
  - $B = op(S) \ast A$

- RCI_P_MULTI
  - $B = op(P) \ast A$
Common Instructions

- **RCI_COPY**
  - $B = op(A)$

- **RCI_SUBCOPY**
  - $B \left(r_B + (1:m), c_B + (1:n)\right) = A\left(r_A + (1:m), c_A + (1:n)\right)$

- **RCI_SUBCOL**
  - $B = A(:, res)$

- **RCI_SUBROW**
  - $B = A(res, :)$

- **RCI_SCALE**
  - $A = \alpha \ast A$

- **RCI_COLSSCALE**
  - $A = A \ast \text{diag}(res)$

- **RCI_COL_NORM**
  - res = \sqrt{\text{diag}(A' \ast A)}

- **RCI_TRACE**
  - res = tr(A)

- **RCI_DOT**
  - res = tr(A * B) = A(:,)' * B(:,)

Locking Technique Requires These Ops
Common Instructions

- **RCI_GEMM**
  - \( C = \alpha \times \text{op}(A) \times \text{op}(B) + \beta \times C \)

- **RCI_AXPY**
  - \( B = \alpha \times A + B \)

- **RCI_HEGV**
  - \( A \times C = B \times C \times \text{diag}(\text{res}) \)

- **RCI_POTRF**
  - \( A = U' \times U \) or \( A = L \times L' \)

- **RCI_TRSM**
  - Solve \( \text{op}(A) \times X = \alpha \times B \) or \( X \times \text{op}(A) = \alpha \times B \) for \( X \)
ijob = RCI_INIT_IJOB
do
    call rci_solve(r_h, ijob, iS, task, resvec)
    select case (task)
    case (RCI_NULL)
    case (RCI_STOP)
        exit
    case (RCI_CONVERGE)
        exit
    case (RCI_H_MULTI)
        call dgemm(iS%TrH, 'N', n, n_state, n, 1.0, H, n, Work(iS%Aidx)%Mat, lda, 0.0, Work(iS%Bidx)%Mat, ldb)
    case (RCI_S_MULTI)
        call dgemm(iS%TrS, 'N', n, n_state, n, 1.0, S, n, Work(iS%Aidx)%Mat, lda, 0.0, Work(iS%Bidx)%Mat, ldb)
    case (RCI_P_MULTI) ! No preconditioner
        Work(iS%Bidx)%Mat = Work(iS%Aidx)%Mat
    case (RCI_GEMM)
        call dgemm(iS%trA, iS%trB, iS%m, iS%n, iS%k, iS%alpha, Work(iS%Aidx)%Mat, iS%lda, Work(iS%Bidx)%Mat, iS%ldb, iS%beta, Work(iS%Cidx)%Mat, iS%ldc)
    case (RCI_AXPY)
        call daxpy(iS%m*iS%n, iS%alpha, Work(iS%Aidx)%Mat, 1, Work(iS%Bidx)%Mat, 1)
    case (RCI_COPY)
        Work(iS%Bidx)%Mat = Work(iS%Aidx)%Mat
    .......
end select
end do
Do as the instruction:
call rci_omm(r_h, ijob, iS, task, resvec)
call rci_davidson(r_h, ijob, iS, task, resvec)
call rci_ppcg(r_h, ijob, iS, task, resvec)
Post-calculation

- Obtain the converged wave functions and the energy
- Obtain eigenvalues or the density matrix if requested
- Deallocate matrices for the iterative method
**ELSI-RCI**

**Target Users**
- Discretization methods such that the Hamiltonian matrix can only be applied as an operator.
- Discretization methods such that $n_{basis}/n_{state}$ is relatively large.
- Eigenvalue problem beyond DFT, e.g., BSE eigenvalue problems.

**Benefits**
- Knowledge of (P)BLAS and (SCA)LAPACK is sufficient to use many different iterative eigensolvers in ELSI_RCI.
- Coding the driver for ELSI_RCI from an existing iterative eigensolver is relatively easy.
- One driver runs many iterative eigensolvers.
Numerical Results

- Silicon $2 \times 2 \times 2$ and $4 \times 4 \times 4$
- Planewave discretization with $E_{\text{cut}} = 20$ Hatree
- ONCV pseudo potential
- Hamiltonian operator from a converged SCF calculation
- Random initial wave functions

- Kerker preconditioner is used
- Convergence criteria $10^{-7}$

<table>
<thead>
<tr>
<th></th>
<th>$N_{\text{basis}}$</th>
<th>$N_{\text{pw}}^{\text{basis}}$</th>
<th>$N_{\text{state}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_{\text{basis}}$</td>
<td>74088</td>
<td>4553</td>
<td>16</td>
</tr>
<tr>
<td>$N_{\text{pw}}^{\text{basis}}$</td>
<td>571787</td>
<td>37073</td>
<td>128</td>
</tr>
<tr>
<td>$N_{\text{state}}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Numerical Results

Si8

Si64

[Graphs showing numerical results for Si8 and Si64 with different methods: Davidson, OMM, PPCG]
Conclusion & Future Work

• ELSI-RCI is a stand alone code without any dependency on other packages except I/O

• When any of $H$, $S$ and $P$ is not explicitly available, ELSI-RCI would be the choice in ELSI

• One time implementation of RCI driver benefits Davidson method, OMM, PPCG and more eigensolvers in the future

• Implement other iterative eigensolvers

• Reduce the memory usage in the absence of overlapping matrix
ELSI

- ELSI and ELSI-RCI are available on the ELSI Gitlab as a project at

  www.elsi-interchange.org