```
    /* diagonalize S */

    dtime(&Stime);

    if (parallel_mode==0){
      EigenBand_lapack(S,ko,n,1);
    }
    else if (SCF_iter==1 || all_knum!=1){
      Eigen_PHH(MPI_CommWD2[myworld2],S,ko,n,n,1);
    }

    dtime(&Etime);
    time2 += Etime - Stime;

    if (SCF_iter==1 || all_knum!=1){

      if (3<=level_stdout){
        printf(" myid0=%2d spin=%2d kloop %2d  k1 k2 k3 %10.6f %10.6f %10.6f¥n",

myid0,spin,kloop,T_KGrids1[kloop],T_KGrids2[kloop],T_KGrids3[kloop]);
        for (i1=1; i1<=n; i1++){
          printf("  Eigenvalues of OLP  %2d  %15.12f¥n",i1,ko[i1]);
        }
      }

      /* minus eigenvalues to 1.0e-14 */

      for (l=1; l<=n; l++){
        if (ko[l]<0.0) ko[l] = 1.0e-14;
        koS[l] = ko[l];
      }

      /* calculate S*1/sqrt(ko) */

      for (l=1; l<=n; l++) ko[l] = 1.0/sqrt(ko[l]);

      /* S * 1.0/sqrt(ko[l]) */

#pragma omp parallel shared(BLAS_S,ko,S,n) private(OMPID,Nthrds,Nprocs,i1,j1)
      {

        /* get info. on OpenMP */

        OMPID = omp_get_thread_num();
        Nthrds = omp_get_num_threads();
        Nprocs = omp_get_num_procs();

        for (i1=1+OMPID; i1<=n; i1+=Nthrds){
          for (j1=1; j1<=n; j1++){

            S[i1][j1].r = S[i1][j1].r*ko[j1];
            S[i1][j1].i = S[i1][j1].i*ko[j1];

            BLAS_S[(j1-1)*n+i1-1] = S[i1][j1];
          }
        }
      } /* #pragma omp parallel */
    }


    /*****************************************************
     1.0/sqrt(ko[l]) * U^t * H * U * 1.0/sqrt(ko[l])
    *****************************************************/

    dtime(&Stime);

    /* first transposition of S */

    /*
    for (i1=1; i1<=n; i1++){
      for (j1=i1+1; j1<=n; j1++){
        Ctmp1 = S[i1][j1];
```

**Diagonalization of S matrix**

- eigenvector of **S** = **U** → S
- eigenvalue of **S** = **s** → ko

$\mathbf{s}^{-1/2} = ko^{-1/2} \rightarrow ko$

**Consruction of X matirix**

**X** = **U** * $\mathbf{s}^{-1/2}$ = S * ko → S

S → BLAS_S

```
        Ctmp2 = S[j1][i1];
        S[i1][j1] = Ctmp2;
        S[j1][i1] = Ctmp1;
      }
    }
    */

    /*********************************
      for parallel in the second world
    *********************************/

    if (all_knum==1){

      /* H * U * 1.0/sqrt(ko[l]) */
      /* C is distributed by row in each processor */

      /*
      for (j1=is1[myid2]; j1<=ie1[myid2]; j1++){
        for (i1=1; i1<=n; i1++){

          sum  = 0.0;
          sumi = 0.0;

          for (l=1; l<=n; l++){
            sum  += H[i1][l].r*S[j1][l].r - H[i1][l].i*S[j1][l].i;
            sumi += H[i1][l].r*S[j1][l].i + H[i1][l].i*S[j1][l].r;
          }

          C[j1][i1].r = sum;
          C[j1][i1].i = sumi;

        }
      }
      */

      /* note for BLAS, A[M*K] * B[K*N] = C[M*N] */

#pragma omp parallel shared(myid2,ie1,is1,BLAS_S,BLAS_H,BLAS_C,n)
 private(OMPID,Nthrds,Nprocs,Ctmp1,Ctmp2,BM,BN,BK,ns,ne)
      {

        /* get info. on OpenMP */

        OMPID = omp_get_thread_num();
        Nthrds = omp_get_num_threads();
        Nprocs = omp_get_num_procs();

        ns = is1[myid2] + OMPID*(ie1[myid2]-is1[myid2]+1)/Nthrds;
        ne = is1[myid2] + (OMPID+1)*(ie1[myid2]-is1[myid2]+1)/Nthrds - 1;

        BM = n;
        BN = ne - ns + 1;
        BK = n;

        Ctmp1.r = 1.0;
        Ctmp1.i = 0.0;
        Ctmp2.r = 0.0;
        Ctmp2.i = 0.0;

        F77_NAME(zgemm,ZGEMM)("N","N", &BM,&BN,&BK,
                              &Ctmp1,
                              BLAS_H, &BM,
                              &BLAS_S[(ns-1)*n], &BK,
                              &Ctmp2,
                              &BLAS_C[(ns-1)*n], &BM);

      } /* #pragma omp parallel */


      /*
      BM = n;
      BN = ie1[myid2] - is1[myid2] + 1;
```

```
145         BK = n;
146
147         Ctmp1.r = 1.0;
148         Ctmp1.i = 0.0;
149         Ctmp2.r = 0.0;
150         Ctmp2.i = 0.0;
151
152         F77_NAME(zgemm,ZGEMM)("N","N", &BM,&BN,&BK,
153                               &Ctmp1,
154                               BLAS_H, &BM,
155                               &BLAS_S[(is1[myid2]-1)*n], &BK,
156                               &Ctmp2,
157                               &BLAS_C[(is1[myid2]-1)*n], &BM);
158         */
159
160
161         /* 1.0/sqrt(ko[l]) * U^+ H * U * 1.0/sqrt(ko[l]) */
162         /* H is distributed by row in each processor */
163
164         /*
165         for (j1=is1[myid2]; j1<=ie1[myid2]; j1++){
166           for (i1=1; i1<=n; i1++){
167
168             sum  = 0.0;
169             sumi = 0.0;
170
171             for (l=1; l<=n; l++){
172               sum  +=  S[i1][l].r*C[j1][l].r + S[i1][l].i*C[j1][l].i;
173               sumi +=  S[i1][l].r*C[j1][l].i - S[i1][l].i*C[j1][l].r;
174             }
175
176             H[j1][i1].r = sum;
177             H[j1][i1].i = sumi;
178
179           }
180         }
181         */
182
183         /* note for BLAS, A[M*K] * B[K*N] = C[M*N] */
184
185     #pragma omp parallel shared(H,myid2,ie1,is1,BLAS_S,BLAS_H,BLAS_C,n)
186      private(OMPID,Nthrds,Nprocs,Ctmp1,Ctmp2,BM,BN,BK,ns,ne,i1,j1)
187         {
188
189           /* get info. on OpenMP */
190
191           OMPID = omp_get_thread_num();
192           Nthrds = omp_get_num_threads();
193           Nprocs = omp_get_num_procs();
194
195           ns = is1[myid2] + OMPID*(ie1[myid2]-is1[myid2]+1)/Nthrds;
196           ne = is1[myid2] + (OMPID+1)*(ie1[myid2]-is1[myid2]+1)/Nthrds - 1;
197
198           BM = n;
199           BN = ne - ns + 1;
200           BK = n;
201
202           Ctmp1.r = 1.0;
203           Ctmp1.i = 0.0;
204           Ctmp2.r = 0.0;
205           Ctmp2.i = 0.0;
206
207           F77_NAME(zgemm,ZGEMM)("C","N", &BM,&BN,&BK,
208                                 &Ctmp1,
209                                 BLAS_H, &BM,
210                                 &BLAS_S[(ns-1)*n], &BK,
211                                 &Ctmp2,
212                                 &BLAS_C[(ns-1)*n], &BM);
213
214         } /* #pragma omp parallel */
215
216
```

```
217         /*
218         BM = n;
219         BN = ie1[myid2] - is1[myid2] + 1;
220         BK = n;
221
222         Ctmp1.r = 1.0;
223         Ctmp1.i = 0.0;
224         Ctmp2.r = 0.0;
225         Ctmp2.i = 0.0;
226
227         F77_NAME(zgemm,ZGEMM)("N","N", &BM,&BN,&BK,
228                               &Ctmp1,
229                               BLAS_H, &BM,
230                               &BLAS_S[(is1[myid2]-1)*n], &BK,
231                               &Ctmp2,
232                               &BLAS_C[(is1[myid2]-1)*n], &BM);
233         */
234
235
236         /* 1.0/sqrt(ko[l]) * U^+ H * U * 1.0/sqrt(ko[l]) */
237         /* H is distributed by row in each processor */
238
239         /*
240         for (j1=is1[myid2]; j1<=ie1[myid2]; j1++){
241           for (i1=1; i1<=n; i1++){
242
243             sum  = 0.0;
244             sumi = 0.0;
245
246             for (l=1; l<=n; l++){
247               sum  +=  S[i1][l].r*C[j1][l].r + S[i1][l].i*C[j1][l].i;
248               sumi +=  S[i1][l].r*C[j1][l].i - S[i1][l].i*C[j1][l].r;
249             }
250
251             H[j1][i1].r = sum;
252             H[j1][i1].i = sumi;
253
254           }
255         }
256         */
257
258         /* note for BLAS, A[M*K] * B[K*N] = C[M*N] */
259
260     #pragma omp parallel shared(H,myid2,ie1,is1,BLAS_S,BLAS_H,BLAS_C,n)
261      private(OMPID,Nthrds,Nprocs,Ctmp1,Ctmp2,BM,BN,BK,ns,ne,i1,j1)
262         {
263
264           /* get info. on OpenMP */
265
266           OMPID = omp_get_thread_num();
267           Nthrds = omp_get_num_threads();
268           Nprocs = omp_get_num_procs();
269
270           ns = is1[myid2] + OMPID*(ie1[myid2]-is1[myid2]+1)/Nthrds;
271           ne = is1[myid2] + (OMPID+1)*(ie1[myid2]-is1[myid2]+1)/Nthrds - 1;
272
273           BM = n;
274           BN = ne - ns + 1;
275           BK = n;
276
277           Ctmp1.r = 1.0;
278           Ctmp1.i = 0.0;
279           Ctmp2.r = 0.0;
280           Ctmp2.i = 0.0;
281
282           F77_NAME(zgemm,ZGEMM)("C","N", &BM,&BN,&BK,
283                                 &Ctmp1,
284                                 BLAS_S, &BM,
285                                 &BLAS_C[(ns-1)*n], &BK,
286                                 &Ctmp2,
287                                 &BLAS_H[(ns-1)*n], &BM);
288
```

```
289           for (j1=ns; j1<=ne; j1++){
290             for (i1=1; i1<=n; i1++){
291               H[j1][i1] = BLAS_H[(j1-1)*n+i1-1];
292             }
293           }
294
295         } /* #pragma omp parallel */
296
297         /* broadcast H */
298
299         BroadCast_ComplexMatrix(MPI_CommWD2[myworld2],H,n,is1,ie1,myid2,numprocs2,
300                                 stat_send,request_send,request_recv);
301       }
302
303       else{
304
305         /* H * U * 1.0/sqrt(ko[l]) */
306
307         /*
308         for (j1=1; j1<=n; j1++){
309           for (i1=1; i1<=n; i1++){
310
311             sum  = 0.0;
312             sumi = 0.0;
313
314             for (l=1; l<=n; l++){
315               sum  += H[i1][l].r*S[j1][l].r - H[i1][l].i*S[j1][l].i;
316               sumi += H[i1][l].r*S[j1][l].i + H[i1][l].i*S[j1][l].r;
317             }
318
319             C[j1][i1].r = sum;
320             C[j1][i1].i = sumi;
321
322           }
323         }
324         */
325
326         /* note for BLAS, A[M*K] * B[K*N] = C[M*N] */
327
328   #pragma omp parallel shared(BLAS_S,BLAS_H,BLAS_C,n)
329    private(OMPID,Nthrds,Nprocs,Ctmp1,Ctmp2,BM,BN,BK)
330         {
331
332           /* get info. on OpenMP */
333
334           OMPID = omp_get_thread_num();
335           Nthrds = omp_get_num_threads();
336           Nprocs = omp_get_num_procs();
337
338           BM = n;
339           BN = (OMPID+1)*n/Nthrds - (OMPID*n/Nthrds+1) + 1;
340           BK = n;
341
342           Ctmp1.r = 1.0;
343           Ctmp1.i = 0.0;
344           Ctmp2.r = 0.0;
345           Ctmp2.i = 0.0;
346
347           F77_NAME(zgemm,ZGEMM)("N","N", &BM,&BN,&BK,
348                                 &Ctmp1,
349                                 BLAS_H, &BM,
350                                 &BLAS_S[(OMPID*n/Nthrds)*n], &BK,
351                                 &Ctmp2,
352                                 &BLAS_C[(OMPID*n/Nthrds)*n], &BM);
353
354         } /* #pragma omp parallel */
355
356         /* 1.0/sqrt(ko[l]) * U^+ H * U * 1.0/sqrt(ko[l]) */
357
358         /*
359         for (j1=1; j1<=n; j1++){
360           for (i1=1; i1<=n; i1++){
```

**Introducing X (1)**

**H * X =** H * S → C

```
361
362             sum  = 0.0;
363             sumi = 0.0;
364
365             for (l=1; l<=n; l++){
366               sum  += S[i1][l].r*C[j1][l].r + S[i1][l].i*C[j1][l].i;
367               sumi += S[i1][l].r*C[j1][l].i - S[i1][l].i*C[j1][l].r;
368             }
369
370             H[j1][i1].r = sum;
371             H[j1][i1].i = sumi;
372           }
373         }
374         */
375
376         /* note for BLAS, A[M*K] * B[K*N] = C[M*N] */
377
378   #pragma omp parallel shared(H,BLAS_S,BLAS_H,BLAS_C,n)
379    private(OMPID,Nthrds,Nprocs,Ctmp1,Ctmp2,BM,BN,BK,i1,j1)
380         {
381
382           /* get info. on OpenMP */
383
384           OMPID = omp_get_thread_num();
385           Nthrds = omp_get_num_threads();
386           Nprocs = omp_get_num_procs();
387
388           BM = n;
389           BN = (OMPID+1)*n/Nthrds - (OMPID*n/Nthrds+1) + 1;
390           BK = n;
391
392           Ctmp1.r = 1.0;
393           Ctmp1.i = 0.0;
394           Ctmp2.r = 0.0;
395           Ctmp2.i = 0.0;
396
397           F77_NAME(zgemm,ZGEMM)("C","N", &BM,&BN,&BK,
398                                 &Ctmp1,
399                                 BLAS_S, &BM,
400                                 &BLAS_C[(OMPID*n/Nthrds)*n], &BK,
401                                 &Ctmp2,
402                                 &BLAS_H[(OMPID*n/Nthrds)*n], &BM);
403
404           for (j1=(OMPID*n/Nthrds+1); j1<=(OMPID+1)*n/Nthrds; j1++){
405             for (i1=1; i1<=n; i1++){
406               H[j1][i1] = BLAS_H[(j1-1)*n+i1-1];
407             }
408           }
409
410         } /* #pragma omp parallel */
411
412       } /* else */
413
414       /* H to C (transposition) */
415
416       for (i1=1; i1<=n; i1++){
417         for (j1=1; j1<=n; j1++){
418           C[j1][i1] = H[i1][j1];
419         }
420       }
421
422       /* penalty for ill-conditioning states */
423
424       EV_cut0 = Threshold_OLP_Eigen;
425
426       for (i1=1; i1<=n; i1++){
427
428         if (koS[i1]<EV_cut0){
429           C[i1][i1].r += pow((koS[i1]/EV_cut0),-2.0) - 1.0;
430         }
431
432         /* cutoff the interaction between the ill-conditioned state */
```

**Introducing X (2)**

$\mathbf{X}^{\dagger} * (\mathbf{H} * \mathbf{X}) = \mathrm{BLAS\_S}^{\dagger} * \mathrm{BLAS\_C} \rightarrow \mathrm{H}$

$\mathrm{H} \rightarrow \mathrm{C}$

```
  if (1.0e+3<C[i1][i1].r){
    for (j1=1; j1<=n; j1++){
      C[i1][j1] = Complex(0.0,0.0);
      C[j1][i1] = Complex(0.0,0.0);
    }
    C[i1][i1].r = 1.0e+4;
  }
}

dtime(&Etime);
time3 += Etime - Stime;

/* diagonalize H' */

dtime(&Stime);

if (parallel_mode==0){
  EigenBand_lapack(C,ko,n,all_knum);
}
else{
  /*  The output C matrix is distributed by column. */
  Eigen_PHH(MPI_CommWD2[myworld2],C,ko,n,MaxN,0);
}

dtime(&Etime);
time4 += Etime - Stime;

for (l=1; l<=MaxN; l++){
  EIGEN[spin][kloop][l] = ko[l];
}

if (3<=level_stdout && 0<=kloop){
  printf(" myid0=%2d spin=%2d kloop %i, k1 k2 k3 %10.6f %10.6f %10.6f¥n",
         myid0,spin,kloop,T_KGrids1[kloop],T_KGrids2[kloop],T_KGrids3[kloop]);
  for (i1=1; i1<=n; i1++){
    if (SpinP_switch==0)
      printf("  Eigenvalues of Kohn-Sham %2d %15.12f %15.12f¥n",
             i1,EIGEN[0][kloop][i1],EIGEN[0][kloop][i1]);
    else
      printf("  Eigenvalues of Kohn-Sham %2d %15.12f %15.12f¥n",
             i1,EIGEN[0][kloop][i1],EIGEN[1][kloop][i1]);
  }
}

/* calculation of wave functions */

dtime(&Stime);

if (all_knum==1){

  /*  The H matrix is distributed by row */

  /*
  for (i1=1; i1<=n; i1++){
    for (j1=is2[myid2]; j1<=ie2[myid2]; j1++){
      H[j1][i1] = C[i1][j1];
    }
  }
  */

  for (i1=1; i1<=n; i1++){
    for (j1=is2[myid2]; j1<=ie2[myid2]; j1++){
      BLAS_H[(j1-1)*n+i1-1] = C[i1][j1];
    }
  }

  /* the second transposition of S */

  /*
  for (i1=1; i1<=n; i1++){
    for (j1=i1+1; j1<=n; j1++){
```

**Diagonalization of $X^{\dagger} * H * X$ (= H')**

- eigenvector of **H'** = **C'** → C
- eigenvalue of **H'** = **ε** → ko

ko → EIGEN

C → BLAS_H

```
505           Ctmp1 = S[i1][j1];
506           Ctmp2 = S[j1][i1];
507           S[i1][j1] = Ctmp2;
508           S[j1][i1] = Ctmp1;
509         }
510       }
511       */
512
513       /* C is distributed by row in each processor */
514
515       /*
516       for (j1=is2[myid2]; j1<=ie2[myid2]; j1++){
517         for (i1=1; i1<=n; i1++){
518
519           sum  = 0.0;
520           sumi = 0.0;
521           for (l=1; l<=n; l++){
522             sum  += S[i1][l].r*H[j1][l].r - S[i1][l].i*H[j1][l].i;
523             sumi += S[i1][l].r*H[j1][l].i + S[i1][l].i*H[j1][l].r;
524           }
525
526           C[j1][i1].r = sum;
527           C[j1][i1].i = sumi;
528         }
529       }
530       */
531
532       /* note for BLAS, A[M*K] * B[K*N] = C[M*N] */
533
534   #pragma omp parallel shared(C,myid2,ie2,is2,BLAS_S,BLAS_H,BLAS_C,n)
535    private(OMPID,Nthrds,Nprocs,Ctmp1,Ctmp2,BM,BN,BK,ns,ne,i1,j1)
536       {
537
538         /* get info. on OpenMP */
539
540         OMPID = omp_get_thread_num();
541         Nthrds = omp_get_num_threads();
542         Nprocs = omp_get_num_procs();
543
544         ns = is2[myid2] + OMPID*(ie2[myid2]-is2[myid2]+1)/Nthrds;
545         ne = is2[myid2] + (OMPID+1)*(ie2[myid2]-is2[myid2]+1)/Nthrds - 1;
546
547         BM = n;
548         BN = ne - ns + 1;
549         BK = n;
550
551         Ctmp1.r = 1.0;
552         Ctmp1.i = 0.0;
553         Ctmp2.r = 0.0;
554         Ctmp2.i = 0.0;
555
556         F77_NAME(zgemm,ZGEMM)("N","N", &BM,&BN,&BK,
557                               &Ctmp1,
558                               BLAS_S, &BM,
559                               &BLAS_H[(ns-1)*n], &BK,
560                               &Ctmp2,
561                               &BLAS_C[(ns-1)*n], &BM);
562
563         for (j1=ns; j1<=ne; j1++){
564           for (i1=1; i1<=n; i1++){
565             C[j1][i1] = BLAS_C[(j1-1)*n+i1-1];
566           }
567         }
568
569       } /* #pragma omp parallel */
570
571       /* broadcast C:
572        C is distributed by row in each processor
573       */
574
575       dtime(&Stime0);
576
```

**C** = **X** * **C'** = BLAS_S * BLAS_H → BLAS_C

BLAS_C → C

```
577         BroadCast_ComplexMatrix(MPI_CommWD2[myworld2],C,n,is2,ie2,myid2,numprocs2,
578                                 stat_send,request_send,request_recv);
579
580         /* C to H (transposition)
581            H consists of column vectors
582         */
583
584         for (i1=1; i1<=MaxN; i1++){
585           for (j1=1; j1<=n; j1++){
586             H[j1][i1] = C[i1][j1];
587           }
588         }
589
590         dtime(&Etime0);
591         time51 += Etime0 - Stime0;
592
593       } /* if (all_knum==1) */
594
595       dtime(&Etime);
596       time5 += Etime - Stime;
597
598     } /* kloop0 */
```