MAX DRIVING THE EXASCALE TRANSITION

GPU accelerated QE

Pietro Bonfà

QE Developers' Meeting

Trieste, 7 Jan. 2019

CINECA

GPU accelerated QE, past and present

Porting effort carried out within MaX CoE and supported by NVIDIA.

2018

2017

2016

2015

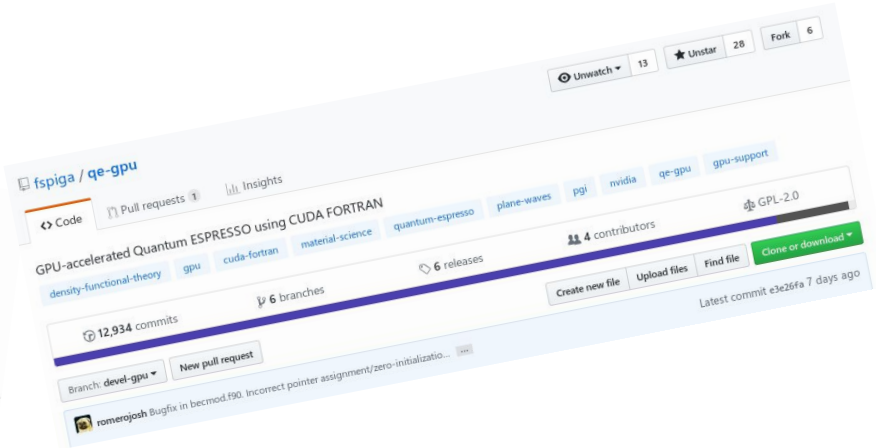
2014

2013

2012

Independent CUDA Fortran based port of QE 6.1 (pw.x) developed by F. Spiga and NVIDIA. Provides best performance, most used features implemented.

CUDA C based plugin for QE 5.x (pw.x) developed by F. Spiga and I. Girotto.



Goals

- Align/add GPU accelerated version based on the CUDA Fortran approach of v6.1.
- Maintain it aligned with CPU version.
- Add unit testing.
- Identify code refactoring steps needed for sustainable GPU port.

Current QE GPU port(s)

- Libraries
  - UtilXlib: entirely ported, needs minor changes to QEF/q-e codes.
  - LAXlib: serial diagonalization ported. Still using custom eigensolver from NVIDIA.
  - FFTXlib: WIP
  - KS_Solver: Davidson and CG ported.
- Codes
  - Pwscf v6.3
  - Phonon (prototype from v 6.1, internally developed by NVIDIA)
- Developers
  - Fabio, Carlo and I @ CINECA
  - Thorsten Kurth, Brandon Cook (others) @ NERSC
  - A. Chandra, I. Girotto @ ICTP
  - J. Romero, M. Maric, E. Philipps, M. Fatica @ NVIDIA
  - Other contributors

UtilXlib

Done:

- Whole set of CUDA Fortran interfaces.
- Unit testing system for GPU accelerated code.
- Optional switch to CUDA-Aware MPI.

To do:

- Requires initialization and finalization call to allocate and deallocate buffers.

(when: this afternoon.)

LAXlib

Done:

- CUDA Fortran interfaces.
- Small unit testing system for the GPU accelerated code.
- Optional switch to global buffer for better performance.

To do:

- Still based on unmaintained but very efficient custom eigensolver.
- May require initialization and finalization call to allocate and deallocate buffers.
- Parallel eigensolver.

FFTXlib

Done:

- 1D+2D and 1D+1D+1D CPU versions with runtime selection.
- 1D+2D (fast) and 1D+1D+1D (slow) GPU versions with runtime selection.
- Batched FFT for GPU subroutines.

Partially done:

- Unit testing suite.
- Batched FFT for CPU subroutines (in collaboration with Intel).

To do:

- Finalize development plan.
- Finalize CPU batched FFT.

Pwscf

GPU version	Total Energy (K points)	Forces	Stress	Collinear Magnetism	Non-collinear magnetism	Gamma trick	US PP	PAW PP	DFT+U	All other functionalities
v5.4	A	W	W	B (?)	U	A	A	?	W (?)	W (?)
v6.1	A	A	A	A	U	W (*)	A	A (*)	U (?)	U (?)
v6.3	A	W	W	A	A	A	A	A (*)	W	W

Accelerated, Working, Unavailable, Broken

* Acceleration obtained through other parts of the code.

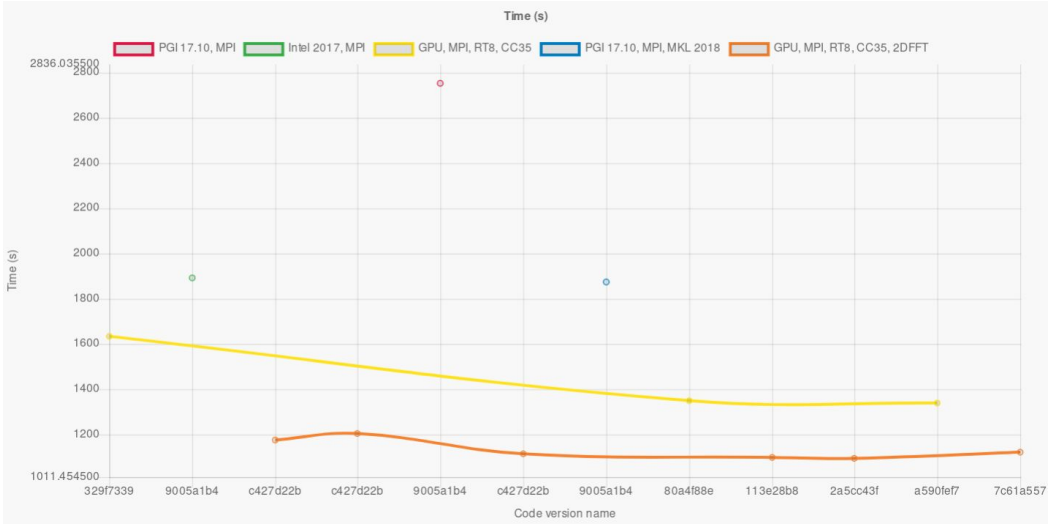
Code status

- Aligned with QEF/q-e (monthly)
- Pure Fortran (some features from 2003 standard)
- Git diff --stat

  `248 files changed, 46591 insertions(+), 219 deletions(-)`
- Most parts of the GPU code can be compiled also on the CPU.
- Accelerated parts can be selectively activated (possibly at runtime).
- Almost only `!$cuf kernel` directives.
- Limited use of CUDA Fortran interfaces (only in libraries).

Testing

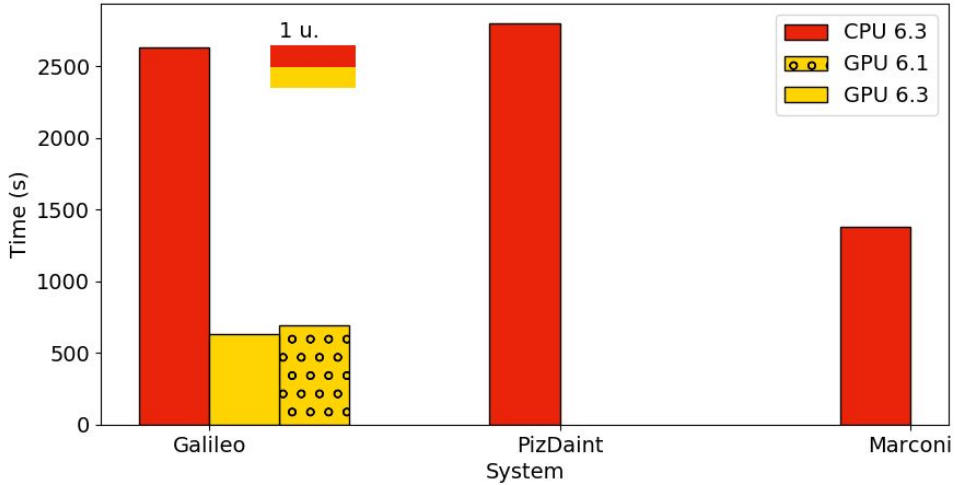
- Passes all 186 tests.
- Tested on K80 & V100 with gitlab runner based CI system.
- “Continuous Benchmarking” under development with AiiDA system.



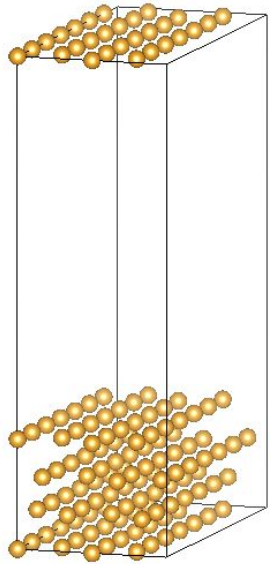
Performance

Hard to define a fair metric.

Node to node comparison: 2x to 3x speedup.



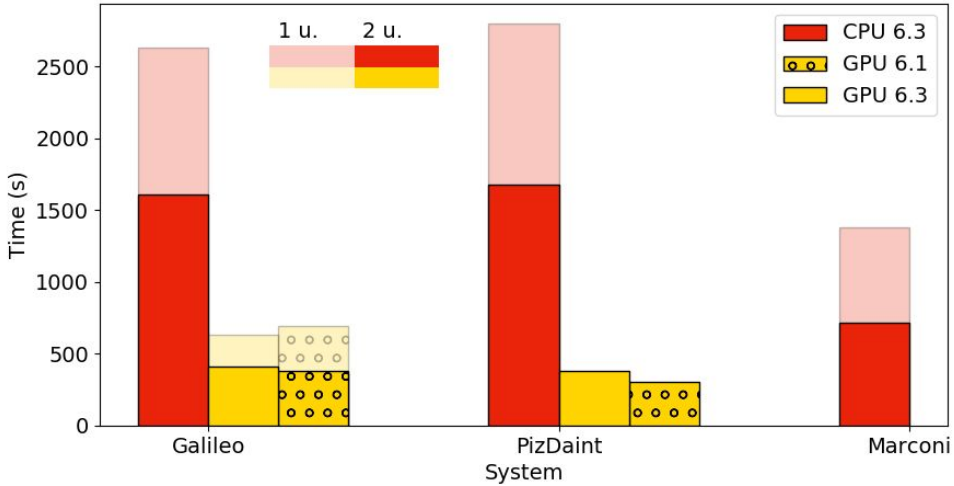
AuSurf, 2 k points



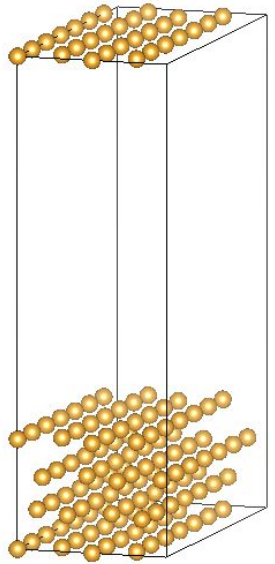
Performance

Hard to define a fair metric.

Node to node comparison: 2x to 3x speedup.



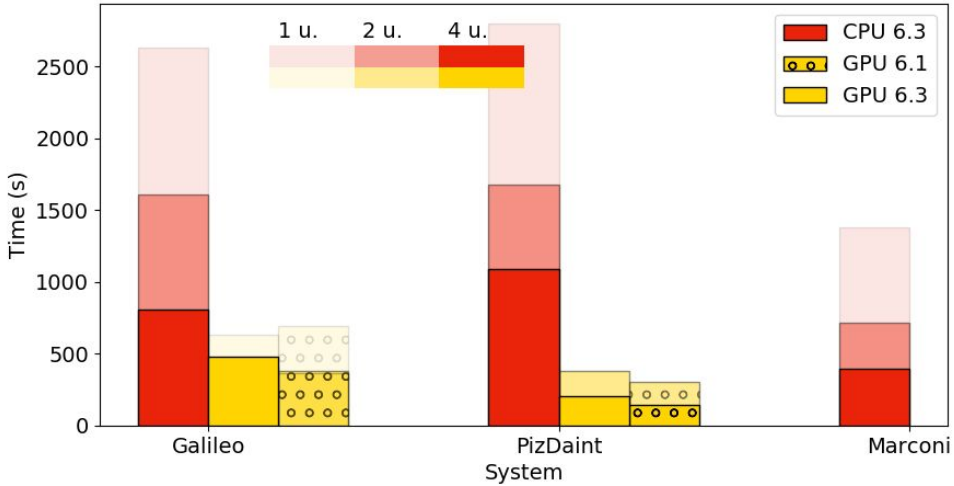
AuSurf, 2 k points



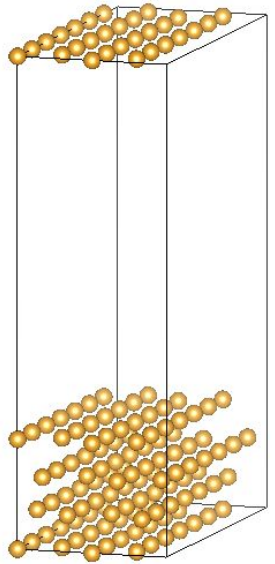
Performance

Hard to define a fair metric.

Node to node comparison: 2x to 3x speedup.



AuSurf, 2 k points



Not yet accelerated

Priority, higher to lower

- (merge) EXX
- Exchange and correlation (funct.f90)
- Forces
- Stress
- PAW subroutines (good OpenMP implementation)

Moving forward

Desiderata:

- Get rid of the current data synchronization mechanism.
- Templating mechanism / code duplication.
- A (global?) scratch space library.
- More testing - add (unit?) tests for uncovered paths.
- More verification.
- Link to standard CUDA Toolkit libraries.

Data Synchronization

```
!----------------------------------------------------------------------------
SUBROUTINE g2_kin_gpu ( ik )
  !----------------------------------------------------------------------------
  !
  ! ... Calculation of kinetic energy - includes the case of the modified
  ! ... kinetic energy functional for variable-cell calculations
  !
  USE kinds,                ONLY : DP
  USE cell_base,            ONLY : tpiba2
  USE gvecw,                ONLY : ecfixed, qcutz, q2sigma
  USE klist,                ONLY : xk, ngk, igk_k_d
  USE wvfct_gpum,           ONLY : g2kin_d, using_g2kin_d
  USE gvect_gpum,           ONLY : g_d, using_g_d
```

```
  ! ... kinetic energy functional for variable-cell calculations
  !
  USE kinds,                ONLY : DP
  USE cell_base,            ONLY : tpiba2
  USE gvecw,                ONLY : ecfixed, qcutz, q2sigma
  USE klist,                ONLY : xk, ngk, igk_k
  USE gvect,                ONLY : g
  USE wvfct,                ONLY : g2kin
  USE wvfct_gpum,           ONLY : using_g2kin
  !
  IMPLICIT NONE
  !
  INTEGER, INTENT (IN) :: ik
  !
  ! ... local variables
  !
  INTEGER :: ig, npw
  REAL(DP), EXTERNAL :: qe_erf
  !
  !
  CALL using_g2kin(1)
  npw = ngk(ik)
  g2kin(1:npw) = ( ( xk(1,ik) + g(1,igk_k(1:npw,ik)) )**2 + &
                   ( xk(2,ik) + g(2,igk_k(1:npw,ik)) )**2 + &
                   ( xk(3,ik) + g(3,igk_k(1:npw,ik)) )**2 ) * tpiba2
  !
  IF ( qcutz > 0.D0 ) THEN
     !
     DO ig = 1, npw
        !
        g2kin(ig) = g2kin(ig) + qcutz * &
             ( 1.D0 + qe_erf( ( g2kin(ig) - ecfixed ) / q2sigma ) )
        !
     END DO
     !
  END IF
  !
  RETURN
  !
END SUBROUTINE g2_kin
```

Manual control of memory management:

- Allocations are *duplicated.*
- Access to each of them is recorded.
- Update is done only if the requested version is found to be out-of-date.

Data Synchronization

```
14  ! ... kinetic energy functional for variable-cell calculations
15  !
16  USE kinds,                ONLY : DP
17  USE cell_base,            ONLY : tpiba2
18  USE klist,                ONLY : xk, ngk, igk_k
19  USE gvect,                ONLY : g
20  USE gvecw,                ONLY : ecfixed, qcutz, q2sigma
21  USE wvfct,                ONLY : g2kin
22  USE wvfct_gpum,           ONLY : using_g2kin
23  !
24  IMPLICIT NONE
25  !
26  INTEGER, INTENT (IN) :: ik
27  !
28  ! ... local variables
29  !
30  INTEGER :: ig, npw
31  REAL(DP), EXTERNAL :: qe_erf
32  !
33  !
34  CALL using_g2kin(1)
35  npw = ngk(ik)
36  g2kin(1:npw) = ( ( xk(1,ik) + g(1,igk_k(1:npw,ik)) )**2 + &
37                   ( xk(2,ik) + g(2,igk_k(1:npw,ik)) )**2 + &
38                   ( xk(3,ik) + g(3,igk_k(1:npw,ik)) )**2 ) * tpiba2
39  !
```

```
!-----------------------------------------------------------------------
SUBROUTINE g2_kin_gpu ( ik )
  !-----------------------------------------------------------------------
  !
  ! ... Calculation of kinetic energy - includes the case of the modified
  ! ... kinetic energy functional for variable-cell calculations
  !
  USE kinds,                ONLY : DP
  USE cell_base,            ONLY : tpiba2
  USE gvecw,                ONLY : ecfixed, qcutz, q2sigma
  USE klist,                ONLY : xk, ngk, igk_k_d
  USE wvfct_gpum,           ONLY : g2kin_d, using_g2kin_d
  USE gvect_gpum,           ONLY : g_d, using_g_d
```

Code duplication

```
359       !$omp parallel do collapse(3)
360       DO n = 1, notcnv
361          DO ipol = 1, npol
362             DO m = 1, numblock
363                psi((m-1)*blocksize+1:MIN(npw, m*blocksize),ipol,nbase+n) = &
364                   psi((m-1)*blocksize+1:MIN(npw, m*blocksize),ipol,nbase+n) / SQRT( ew(n) )
365             END DO
366          END DO
367       END DO
368       !$omp end parallel do
```

```
402    !$cuf kernel do(3) <<<*,*>>>
403          DO i = 1,notcnv
404             DO j=1,npol
405                DO k=1,npw
406                   psi_d(k,j,nbase+i) = psi_d(k,j,nbase+i)/SQRT( ew_d(i) )
407                END DO
408             END DO
409          END DO
```

Arch. specific optimization.

In v6.1 GPU this is done through *preprocessor directives and macros*.

Code duplication

```
!-----------------------------------------------------------------------
subroutine xc_spin (rho, zeta, ex, ec, vxup, vxdw, vcup, vcdw)
  !-----------------------------------------------------------------------
  !     lsd exchange and correlation functionals - Hartree a.u.
  !
  !     exchange  :  Slater (alpha=2/3)
  !     correlation: Ceperley & Alder (Perdew-Zunger parameters)
  !                  Perdew & Wang
  !
  !     input : rho = rhoup(r)+rhodw(r)
  !             zeta=(rhoup(r)-rhodw(r))/rho
  !
  implicit none

  real(DP) :: rho, zeta, ex, ec, vxup, vxdw, vcup, vcdw
  real(DP) :: ec__, vcup__, vcdw__
  !
  real(DP), parameter :: small= 1.E-10_DP, third = 1.0_DP/3.0_DP, &
       pi34= 0.6203504908994_DP ! pi34=(3/4pi)^(1/3)
  real(DP) :: rs
  !
  if (rho <= small) then
     ec = 0.0_DP
     vcup = 0.0_DP
     vcdw = 0.0_DP
     ex = 0.0_DP
     vxup = 0.0_DP
     vxdw = 0.0_DP
     return
  else
     rs = pi34 / rho**third
  endif
  !..exchange
  IF (iexch == 1) THEN      ! 'sla'
     call slater_spin (rho, zeta, ex, vxup, vxdw)
```

```
!-----------------------------------------------------------------------
attributes(device) subroutine xc_spin_dev (iexch, icorr, rho, zeta, ex, ec
  !-----------------------------------------------------------------------
  !     lsd exchange and correlation functionals - Hartree a.u.
  !
  !     exchange  :  Slater (alpha=2/3)
  !     correlation: Ceperley & Alder (Perdew-Zunger parameters)
  !                  Perdew & Wang
  !
  !     input : iexch, icorr
  !             rho = rhoup(r)+rhodw(r)
  !             zeta=(rhoup(r)-rhodw(r))/rho
  !
  implicit none

  integer, value :: iexch, icorr
  real(DP), value :: rho, zeta
  real(DP), device, intent(out)  :: ex, ec, vxup, vxdw, vcup, vcdw
  real(DP) :: ec__, vcup__, vcdw__
  !
  real(DP), parameter :: small= 1.E-10_DP, third = 1.0_DP/3.0_DP, &
       pi34= 0.6203504908994_DP ! pi34=(3/4pi)^(1/3)
  real(DP) :: rs
  !
  if (rho <= small) then
     ec = 0.0_DP
     vcup = 0.0_DP
     vcdw = 0.0_DP
     ex = 0.0_DP
     vxup = 0.0_DP
     vxdw = 0.0_DP
     return
  else
     rs = pi34 / rho**third
  endif
```

Code duplication

```
real(DP) :: rho, zeta, ex, ec, vxup, vxdw, vcup, vcdw
real(DP) :: ec__, vcup__, vcdw__
!
real(DP), parameter :: small= 1.E-10_DP, third = 1.0_DP/3.0_DP, &
     pi34= 0.6203504908994_DP ! pi34=(3/4pi)^(1/3)
real(DP) :: rs
!
if (rho <= small) then
   ec = 0.0_DP
   vcup = 0.0_DP
   vcdw = 0.0_DP
   ex = 0.0_DP
   vxup = 0.0_DP
   vxdw = 0.0_DP
   return
else
   rs = pi34 / rho**third
endif
!..exchange
IF (iexch == 1) THEN      ! 'sla'
   call slater_spin (rho, zeta, ex, vxup, vxdw)
ELSEIF (iexch == 2) THEN  ! 'sl1'
   call slater1_spin (rho, zeta, ex, vxup, vxdw)
ELSEIF (iexch == 3) THEN  ! 'rxc'
   call slater_rxc_spin ( rho, zeta, ex, vxup, vxdw )
ELSEIF ((iexch == 4).or.(iexch==5)) THEN  ! 'oep','hf'
   IF (exx_started) then
      ex   = 0.0_DP
      vxup = 0.0_DP
      vxdw = 0.0_DP
   else
      call slater_spin (rho, zeta, ex, vxup, vxdw)
   endif
ELSEIF (iexch == 6) THEN  ! 'pb0x'
```

```
real(DP), device, intent(out)  :: ex, ec, vxup, vxdw, vcup, vcdw
real(DP) :: ec__, vcup__, vcdw__
!
real(DP), parameter :: small= 1.E-10_DP, third = 1.0_DP/3.0_DP, &
     pi34= 0.6203504908994_DP ! pi34=(3/4pi)^(1/3)
real(DP) :: rs
!
if (rho <= small) then
   ec = 0.0_DP
   vcup = 0.0_DP
   vcdw = 0.0_DP
   ex = 0.0_DP
   vxup = 0.0_DP
   vxdw = 0.0_DP
   return
else
   rs = pi34 / rho**third
endif
!..exchange
IF (iexch == 1) THEN      ! 'sla'
   call slater_spin_dev (rho, zeta, ex, vxup, vxdw)
!ELSEIF (iexch == 2) THEN  ! 'sl1'
!   call slater1_spin (rho, zeta, ex, vxup, vxdw)
!ELSEIF (iexch == 3) THEN  ! 'rxc'
!   call slater_rxc_spin ( rho, zeta, ex, vxup, vxdw )
!ELSEIF ((iexch == 4).or.(iexch==5)) THEN  ! 'oep','hf'
!   IF (exx_started) then
!      ex   = 0.0_DP
!      vxup = 0.0_DP
!      vxdw = 0.0_DP
!   else
!      call slater_spin (rho, zeta, ex, vxup, vxdw)
!   endif
!ELSEIF (iexch == 6) THEN  ! 'pb0x'
```

Code duplication

```
131   CALL hinit0()
132   !
133   CALL potinit()
134   !
135   IF ( use_gpu ) THEN
136     !
137     CALL newd_gpu()
138     !
139     CALL wfcinit_gpu()
140     !
141   ELSE
142     !
143     CALL newd()
144     !
145     CALL wfcinit()
146     !
147   END IF
148   !
149   IF(use_wannier) CALL wannier_init()
```

Code duplication

```
131   CALL hinit0()
132   !
133   CALL potinit()
134   !
135   IF ( use_gpu ) THEN
136     !
137     CALL newd_gpu()
138     !
139     CALL wfcinit_gpu()
140     !
141   ELSE
142     !
143     CALL newd()
144     !
145     CALL wfcinit()
146     !
147   END IF
148   !
149   IF(use_wannier) CALL wannier_init()
```

Code duplication

```
DO ik = 1, nks
   !
   ! ... Hpsi initialization: k-point index, spin, kinetic energy
   !
   current_k = ik
   IF ( lsda ) current_spin = isk(ik)
   call g2_kin (ik)
   !
   ! ... More Hpsi initialization: nonlocal pseudopotential projectors |
   !
   IF ( nkb > 0 ) CALL using_vkb(1)
   IF ( nkb > 0 ) CALL init_us_2( ngk(ik), igk_k(1,ik), xk(1,ik), vkb )
   !
   ! ... Needed for LDA+U
   !
   IF ( nks > 1 .AND. lda_plus_u .AND. (U_projection .NE. 'pseudo') ) &
      CALL get_buffer( wfcU, nwordwfcU, iunhub, ik )
   !
   ! ... calculate starting wavefunctions (calls Hpsi)
   !
   CALL init_wfc ( ik )
   !
   ! ... write  starting wavefunctions to file
   !
   IF ( nks > 1 .OR. (io_level > 1) .OR. lelfield ) CALL using_evc(0)
   IF ( nks > 1 .OR. (io_level > 1) .OR. lelfield ) &
       CALL save_buffer ( evc, nwordwfc, iunwfc, ik )
   !
END DO
!
CALL stop_clock( 'wfcinit' )
RETURN
!
END SUBROUTINE wfcinit
```

```
DO ik = 1, nks
   !
   ! ... Hpsi initialization: k-point index, spin, kinetic energy
   !
   current_k = ik
   IF ( lsda ) current_spin = isk(ik)
   call g2_kin_gpu (ik)
   !
   ! ... More Hpsi initialization: nonlocal pseudopotential projectors
   !
   IF ( nkb > 0 ) CALL using_vkb_d(1)
   IF ( nkb > 0 ) CALL init_us_2_gpu( ngk(ik), igk_k_d(1,ik), xk(1,ik),
   !
   ! ... Needed for LDA+U
   !
   IF ( nks > 1 .AND. lda_plus_u .AND. (U_projection .NE. 'pseudo') ) &
      CALL get_buffer( wfcU, nwordwfcU, iunhub, ik )
   !
   ! ... calculate starting wavefunctions (calls Hpsi)
   !
   CALL init_wfc_gpu ( ik )
   !
   ! ... write  starting wavefunctions to file
   !
   IF ( nks > 1 .OR. (io_level > 1) .OR. lelfield ) CALL using_evc(0)
   IF ( nks > 1 .OR. (io_level > 1) .OR. lelfield ) &
       CALL save_buffer ( evc, nwordwfc, iunwfc, ik )
   !
END DO
!
CALL stop_clock( 'wfcinit' )
RETURN
!
END SUBROUTINE wfcinit_gpu
```

Scratch space

- GPU memory is small compared to DRAM.
- Allocations and deallocations on the GPU are time consuming operations.
- Many subroutines of the libraries and the codes require scratch space.

Current status:

- UtilXlib and LAXlib use *their own* scratch arrays.
- LAXlib (optionally), KS_Solver and PW share the same buffers.

Testing

A number of paths in the code is not tested. Flawed GPU subroutines were found to pass all tests. Mostly (but not only!) due to the testing suite not covering all parallelization schemes.

Should we continue to work with the current infrastructure (testing suite)?

- Execution time
- A lot of overlap in tested code with feature testing
- Numerical noise

Verification

Test the GPU code with input data bigger than the structure considered in tests.

- Set of functionalities?
- Continue to use current AiiDA platform?
- Release alpha version?

https://bonfus.gitlab.io/qe-gpu-verification/

Accelerated libraries

- Eigensolver from cusolver (to be included in future CUDA Toolkit)
- ELPA as accelerated parallel eigensolver.
- Accelerated LAPACK library?

Programming paradigms

- Experiment with dropping dependency on PGI compilers
- OpenMP 5

Accelerated libraries

- Eigensolver from cusolver (to be included in future CUDA Toolkit)
- ELPA as accelerated parallel eigensolver.
- Accelerated LAPACK library?

Programming paradigms

- Experiment with dropping dependency on PGI compilers
- OpenMP 5

Thank you!