devXlib: intro, status, questions

A. Ferretti (CNR-NANO)

P. Bonfa' (Uni Parma)

motivation

MAX DRIVING THE EXASCALE TRANSITION

- **Performance portability** (support of heterogenous machines) for **Fortran** codes
- deal with **multiple HW and SW stacks,** programming models, missing **standards** (at variance with the MPI-case)
- **wrap/encapsulate** device-specific code
- **do not disrupt** the codes (support the **communities** around the codes)
- rationalise what **device-oriented abstract operations** are exposed to SW developers

example: usage of MPI (QE)

- **wrappers**

```
      SUBROUTINE mp_bcast_i1(msg,source,gid)
        IMPLICIT NONE
        INTEGER :: msg
        INTEGER :: source
        INTEGER, INTENT(IN) :: gid
        INTEGER :: group
        INTEGER :: msglen

#if defined(__MPI)
        msglen = 1
        group = gid
        CALL bcast_integer( msg, msglen, source, group )
#endif
      END SUBROUTINE mp_bcast_i1
```

```
      SUBROUTINE mp_sum_iv(msg,gid)
        IMPLICIT NONE
        INTEGER, INTENT (INOUT) :: msg(:)
        INTEGER, INTENT(IN) :: gid
#if defined(__MPI)
        INTEGER :: msglen
        msglen = size(msg)
        CALL reduce_base_integer( msglen, msg, gid, -1 )
#endif
      END SUBROUTINE mp_sum_iv
```

```
    INTERFACE mp_bcast
      MODULE PROCEDURE mp_bcast_i1, mp_bcast_r1, mp_bcast_c1, &
        mp_bcast_z, mp_bcast_zv, &
        mp_bcast_iv, mp_bcast_rv, mp_bcast_cv, mp_bcast_i, mp_bcast_rm, &
        mp_bcast_cm, mp_bcast_im, mp_bcast_it, mp_bcast_rt, mp_bcast_lv, &
        mp_bcast_ln, mp_bcast_r4d, mp_bcast_r5d, mp_bcast_bt,  mp_bcast_c4d,&
        mp_bcast_c5d
    END INTERFACE
```

- **abstracted operations**
  - **mp_bcast**
  - **mp_sum**
  - mp_barrier
  - mp_get
  - mp_put
  - …
  - also incl other domain specific operations

- **MPI-oriented data types**

```
TYPE bec_type
   REAL(DP),    ALLOCATABLE :: r(:,:)
   COMPLEX(DP),ALLOCATABLE :: k(:,:)
   COMPLEX(DP),ALLOCATABLE :: nc(:,:,:)
   INTEGER :: comm
   INTEGER :: nbnd
   INTEGER :: nproc
   INTEGER :: mype
   INTEGER :: nbnd_loc
   INTEGER :: ibnd_begin
END TYPE bec_type
!
TYPE (bec_type) :: becp  ! <beta|psi>
```

example: usage of MPI (Yambo)

- **wrappers**

```
 subroutine PARALLEL_wait(COMM)
   integer, optional :: COMM
#if defined _MPI
   integer :: local_COMM
   if (npu==1) return
   !
   local_COMM=mpi_comm_world
   if (present(COMM)) then
     local_COMM=COMM
   endif
   if (local_COMM==comm_default_value) return
   call mpi_barrier(local_COMM,i_err)
#endif
 end subroutine
```

```
 interface PP_redux_wait
   module procedure i1share,                                    &
&                   i1share,i2share,i3share,                    &
&                   i8lshare,                                   &
&                   r0share,r1share,r2share,r3share,            &
&                   c0share,c1share,c2share,c3share,c4share,c5share, &
#if ! defined _DOUBLE
&                   d0share,d1share,d2share,d3share,            &
#endif
&                   PARALLEL_wait
 end interface PP_redux_wait
 !
 interface PP_bcast
   module procedure i1bcast,i0bcast,i1bcast,i2bcast,i3bcast,i3bcast,i1bcast,ch0bcast
#if ! defined _DOUBLE
   module procedure                                  z3bcast
#endif
 end interface PP_bcast
```

- **abstracted operations**
  - **PP_bcast**
  - **PP_redux_wait**
  - **PARALLEL_indexes**
  - …

- **MPI-oriented data types**

```
type PAR_matrix
  !
  character(3) :: kind = "XXX" ! "PAR" "SLK" "X
  !
  ! Dimensions
  !
  integer :: I          = 1  ! Block element
  integer :: N          = -1 ! Global dimensio
  integer :: rows(2)    = 0  ! Real
  integer :: cols(2)    = 0  !
  integer :: nrows      = -1 !
  integer :: ncols      = -1 !
  !
  ! BLACS
  !
  integer :: BLCnrows      = -1 ! Dimension of
  integer :: BLCncols      = -1 !
  integer :: BLCrows(2)    = 0  ! Dimension of
  integer :: BLCcols(2)    = 0  !
  integer :: desc(desc_len)
  integer :: lwork      = -1
  integer :: lrwork     = -1
  integer :: liwork     = -1
  integer :: info       = -1
```

motivation

MAX DRIVING THE EXASCALE TRANSITION

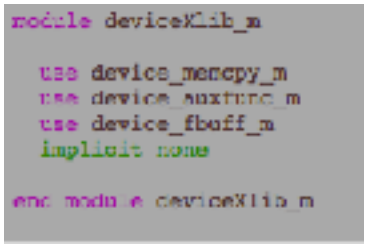
- **Performance portability** (support of heterogenous machines) for **Fortran** codes
- deal with **multiple HW and SW stacks,** programming models, missing **standards** (at variance with the MPI-case)
- **wrap/encapsulate** device-specific code
- **do not disrupt** the codes (support the **communities** around the codes)
- rationalise what **device-oriented abstract operations** are exposed to SW developers
    - Ex: **MPI** introduces new **concepts** (eg communicators) and **operations** (bcast, reductions, distribute/collects) and requires dedicated **data structures**

the same needs to be done for accelerators

overview

- **devXlib** started as a **collection of utilities**, wrappers, interfaces from QE and Yambo to encapsulate/hide **CUDA Fortran** code
- Naturally shared among Fortran codes exploiting CUDA Fortran for NVIDIA GPU portability
- Currently **CUDA Fortran** only, extension to **OPENMP5** foreseen (in principle direct support of CUDA possible)
- At the moment: **proof of concept**, the library is in its infancy, very **flexible** to accommodate diverse **needs** and re-orient the development

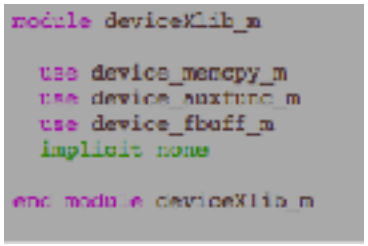
devXlib: status



- Handles:
  - **memcpy** (host-dev, dev-host, dev-dev, host-host), sync and async, data initialization
  - creation and management of device **memory buffers**
  - **selected** rank1 and rank2 operations not on blas/lapack, ex: remapping of a vector, vec Mat vec element-wise mult
  - **LinAlg** wrappers (to come)

- currently made of few modules (see above) containing **multiple interfaces** overloaded over all possible types, kinds, ranks
- routines, interfaces, tests, are **automatically generated** using jinja2 templating via **python** scripts
- fortran modules and fortran headers available
- devXlib is currently used/developers by QE and Yambo; contributors and/or adopters welcome
- https://gitlab.com/max-centre/components/devicexlib

devXlib: status



- Handles:
  - **memcpy** (host-dev, dev-host, dev-dev, host-host), sync and async, data initialization
  - creation and management of device **memory buffers**
  - **selected** rank1 and rank2 operations not on blas/lapack, ex: remapping of a vector, vec Mat vec element-wise mult
  - **LinAlg** wrappers (to come)

- currently made of few modules (see above) containing **multiple interfaces** overloaded over all possible types, kinds, ranks
- routines, interfaces, tests, are **automatically generated** using jinja2 templating via **python** scripts
- fortran modules and fortran headers available
- devXlib is currently used/developers by QE and Yambo; contributors and/or adopters welcome
- https://gitlab.com/max-centre/components/devicexlib

device_memcpy

- **dev_memcpy** (host-dev, dev-host, dev-dev, host-host)
- **dev_memcpy_async**
- **dev_stream_sync** (stream synchronization)
- **dev_memset** (initialization)

```
!
subroutine dp_dev_memcpy_r1d(array_out, array_in, &
                                        range1, lbound1 )
    implicit none
    !
    integer, parameter :: PRSN = selected_real_kind(14,200)
    real(PRSN), intent(inout) :: array_out(:)
    real(PRSN), intent(in)    :: array_in(:)
    integer, optional, intent(in) ::  range1(2)
    integer, optional, intent(in) ::  lbound1
#if defined(__CUDA)
    attributes(device) :: array_out, array_in
#endif
    !
    integer :: i1, dls, dle
    integer :: lbound1_, range1_(2)
    !
    lbound1_=1
    if (present(lbound1)) lbound1_=lbound1
    range1_=(/1,size(array_out, 1)/)
    if (present(range1)) range1_=range1
    !
    ! the lower bound of the assumed shape array passed to the subroutine is 1
    ! lbound and range instead refer to the indexing in the parent caller.
    dls = range1_(1) - lbound1_ +1
    dle = range1_(2) - lbound1_ +1
    !
    !
    !$acc kernels do(1)
    do i1 = dls, dle
        array_out(i1 ) = array_in(i1 )
    enddo
    !
end subroutine dp_dev_memcpy_r1d
```

device_fbuff

```
!> The main **fbuf** class.
  type :: tb_dev_t
     logical :: verbose = .false.
     !
   contains
     procedure :: init                    !
     final :: clean

     procedure :: reinit
     generic, public :: lock_buffer => &
                        lock_buffer_iv, &
                        lock_buffer_im, &
                        lock_buffer_it, &
                        lock_buffer_if, &
                        lock_buffer_rv, &
                        lock_buffer_rm, &
                        lock_buffer_rt, &
                        lock_buffer_rf, &
                        lock_buffer_cv, &
                        lock_buffer_cm, &
                        lock_buffer_ct, &
                        lock_buffer_cf
     !
     procedure, private :: lock_buffer_iv
     procedure, private :: lock_buffer_im
     procedure, private :: lock_buffer_it
     procedure, private :: lock_buffer_if
     procedure, private :: lock_buffer_rv
     procedure, private :: lock_buffer_rm
     procedure, private :: lock_buffer_rt
     procedure, private :: lock_buffer_rf
     procedure, private :: lock_buffer_cv
     procedure, private :: lock_buffer_cm
     procedure, private :: lock_buffer_ct
     procedure, private :: lock_buffer_cf
     !
```

```
     generic, public :: release_buffer => &
                        release_buffer_iv, &
                        release_buffer_im, &
                        release_buffer_it, &
                        release_buffer_if, &
                        release_buffer_rv, &
                        release_buffer_rm, &
                        release_buffer_rt, &
                        release_buffer_rf, &
                        release_buffer_cv, &
                        release_buffer_cm, &
                        release_buffer_ct, &
                        release_buffer_cf
     !
     procedure, private :: release_buffer_iv
     procedure, private :: release_buffer_im
     procedure, private :: release_buffer_it
     procedure, private :: release_buffer_if
     procedure, private :: release_buffer_rv
     procedure, private :: release_buffer_rm
     procedure, private :: release_buffer_rt
     procedure, private :: release_buffer_rf
     procedure, private :: release_buffer_cv
     procedure, private :: release_buffer_cm
     procedure, private :: release_buffer_ct
     procedure, private :: release_buffer_cf
     !
     generic, public :: prepare_buffer => &
                        prepare_buffer_iv, &
                        prepare_buffer_im, &
                        prepare_buffer_it, &
                        prepare_buffer_if, &
                        prepare_buffer_rv, &
                        prepare_buffer_rm, &
                        prepare_buffer_rt, &
                        prepare_buffer_rf, &
                        prepare_buffer_cv, &
                        prepare_buffer_cm, &
                        prepare_buffer_ct, &
                        prepare_buffer_cf
     !
```

device_fbuff

```
subroutine lock_space(this, d, cloc, info)
  use iso_c_binding
  implicit none
  class(tb_dev_t), intent(inout)     :: this     !< The class.
  integer(kind=LLI), intent(in)      :: d
#if defined(__CUDA)
  type(c_devptr), intent(inout)  :: cloc
#else
  type(c_ptr), intent(inout)         :: cloc
#endif
  integer, intent(out)               :: info
  !
```

```
  ! Find the smallest usable buffer
  good_one_idx = 0
  temp => Head
  NULLIFY(good)
  DO WHILE (ASSOCIATED(temp))
     sz = SIZE(TemP%space, kind=LLI)
     IF ( ( sz >= d ) .and. (TemP%locked .eqv. .false.) ) THEN
        IF ( good_one_idx >= 1 ) THEN
           IF ( sz - d < r) THEN
              good => temp
              r = SIZE(TemP%space, kind=LLI) - d
              good_one_idx = i
           END IF
        ELSE
           good_one_idx = i
           good => temp
           r = sz - d
        END IF
        info = 0
     END IF
     !
     tsz = tsz + sz
     i = i - 1
     !
     TemP => TemP%Next
  END DO
```

```
  ! Allocate a new buffer
  IF ( good_one_idx == 0 ) THEN
     ALLOCATE (good)
#if defined(__CUDA)
     ALLOCATE (good%space(d), stat=info)
#else
     ALLOCATE (good%space(d), stat=info)
#endif
     good%Next   => Head
     Head        => good
     tsz         = tsz + d
     if (this%verbose) write (*, '(1X,"[tb_dev] Created new buffer")')
  ELSE
     if (this%verbose) write (*, '(1X,"[tb_dev] Locked buffer", I4)') good_one_idx
  END IF
```

device_auxfunc

- **dev_conjg** (cmplx conjg on dev array)
- **dev_vec_upd_remap** (vector remapping)
- **dev_vec_upd_v_remap_v** (vector * remapped vector)
- **dev_mat_upd_dMd** (vector * matrix * vector el-wise)

```
subroutine dp_dev_vec_upd_remap_r1d(ndim, vout, v1, map1, scal)
    implicit none
    !
    integer, parameter :: PRCSN = selected_real_kind(14,200)
    integer,       intent(in)    :: ndim
    real(PRCSN), intent(inout) :: vout(:)
    real(PRCSN), intent(in)    :: v1(:)
    integer,       intent(in)    :: map1(:)
    real(PRCSN), optional, intent(in)    :: scal
#if defined(__CUDA)
    attributes(device) :: vout, v1, map1
#endif
    integer :: i
    !
    if (present(scal)) then
      !$cuf kernel do(1)
      do i = 1, ndim
          vout(i) = v1(map1(i))*scal
      enddo
    else
      !$cuf kernel do(1)
      do i = 1, ndim
          vout(i) = v1(map1(i))
      enddo
    endif
end subroutine dp_dev_vec_upd_remap_r1d
```

```
subroutine dp_dev_vec_upd_v_remap_v_x_r1d(ndim, vout, v1,op1, map1, v2,op2, scal)
    implicit none
    !
    integer, parameter :: PRCSN = selected_real_kind(14,200)
    integer,       intent(in)    :: ndim
    complex(PRCSN), intent(inout) :: vout(:)
    complex(PRCSN), intent(in)    :: v1(:)
    integer,       intent(in)    :: map1(:)
    complex(PRCSN), intent(in)    :: v2(:)
    character(1), intent(in)    :: op1, op2
    complex(PRCSN), optional, intent(in)    :: scal
#if defined(__CUDA)
    attributes(device) :: vout, v1, v2, map1
#endif
    integer :: i
    !
    if (op1=='N'.and.op2=='N') then
       if (present(scal)) then
         !$cuf kernel do(1)
         do i = 1, ndim
             vout(i) = v1(map1(i))*v2(i)*scal
         enddo
       else
         !$cuf kernel do(1)
         do i = 1, ndim
             vout(i) = v1(map1(i))*v2(i)
         enddo
       endif
    elseif (op1=='C'.and.op2=='N') then
       if (present(scal)) then
         !$cuf kernel do(1)
         do i = 1, ndim
             vout(i) = conjg(v1(map1(i)))*v2(i)*scal
         enddo
```

device_auxfunc

MAX DRIVING THE EXASCALE TRANSITION

```
subroutine dp_dev_mat_upd_dMd_r2d(ndim1, ndim2, mat, v1,op1, v2,op2, scal)
    !
    ! performs: mat(i,j) = scal * op1(v1(i)) * mat(i,j) * op2(v2(j))
    ! op = 'N', 'R', 'C',       'RC'
    !       x   1/x  conjg(x)   conjg(1/x)
    implicit none
    !
    integer, parameter :: PRCSN = selected_real_kind(14,200)
    integer,      intent(in)    :: ndim1,ndim2
    real(PRCSN), intent(inout) :: mat(:,:)
    real(PRCSN), intent(in)    :: v1(:)
    real(PRCSN), intent(in)    :: v2(:)
    character(1), intent(in)    :: op1, op2
    real(PRCSN), optional, intent(in)  :: scal
#if defined(_CUDA)
    attributes(device) :: mat, v1, v2
#endif
    integer :: i,j
    !
    if (op1=="N".and.op2=='N') then
       if (present(scal)) then
#ifndef _CUDA
         !$omp parallel do default(shared), private(i,j), collapse(2)
#else
         !$cuf kernel do(2)
#endif
         do j = 1, ndim2
         do i = 1, ndim1
             mat(i,j) = scal * v1(i) * mat(i,j) * v2(j)
         enddo
         enddo
```

dev_defs.h

```
#ifdef __STDC__
#  define CAT(a,b) a##b
#else
#  define PASTE(a) a
#  define CAT(a,b) PASTE(a)b
#endif

#ifdef _CUDA
#  define DEV_SUBNAME(x)        CAT(x,_gpu)
#  define DEV_SUBNAME_ALT(x)    CAT(x,_gpu)
#  define DEV_VARNAME(x)        CAT(x,_d)
#  define DEV_ATTRIBUTE         , device
#else
#  define DEV_SUBNAME(x)        x
#  define DEV_SUBNAME_ALT(x)    CAT(x,_cpu)
#  define DEV_VARNAME(x)        x
#  define DEV_ATTRIBUTE
#endif
```

- precompiled-based defs
- effective to handle variable/routine renaming
- source decoration

examples: Yambo HF

```
!
subroutine XCo_Hartree_Fock(E,k,Xk,q,mode)
 !
 ! Hartree-Fock
 !
```

```
 use wrapper,        ONLY:Vstar_dot_VV
 use global_XC,      ONLY:WF_exx_screening
 use pseudo,         ONLY:becp, pp_is_uspp
 use qe_pseudo_m,    ONLY:becscl
 use devicexlib_m,   ONLY:dev_memcpy
 !
#include<dev_defs.h>
#include<memory.h>
 !
```

```
    do jb=Sx_lower_band,Sx_upper_band
      !
      if (.not.PAR_IND_G_b%element_1D(jb)) cycle
      !
      isc%os(1)=jb
      iscp%os=isc%os
      !
      call DEV_SUBNAME(scatter_Bamp)(isc)
      !
      ! Normal case, the density matrix is diagonal
      !
      if (isc%is(1)/=iscp%is(1)) then
        call DEV_SUBNAME(scatter_Bamp)(iscp)
      else
        ! iscp%rhotw = isc%rhotw
        call dev_memcpy(DEV_VARNAME(iscp%rhotw),DEV_VARNAME(isc%rhotw))
      endif
      !
      DP_Sx_1=Vstar_dot_VV(isc%ngrho,DEV_VARNAME(iscp%rhotw),&
&                                    DEV_VARNAME(isc%rhotw),DEV_VARNAME(isc%qamp)(:,1))
      DP_Sx=DP_Sx + DP_Sx_1 * ( -4._SP/spin_occ*pi*E%f(jb,isc%os(2),isc%os(4)) )
      !
      if (master_thread.and.is_ibz==1.and.n_lt_steps>0) call live_timing(steps=1)
      !
    enddo
```

examples: Yambo X_redux

```
call X_redux_build_kernel(KERNEL,Xo,Xo_rows,Xo_cols)

!
! X(g,gp)=Sum_gpp[KERNEL]^-1_(g,gpp)*Xo(gpp,gp)
!
if (X_use_lin_sys) then
  !
  ! Linear System
  !
  call dev_memcpy(DEV_VARNAME(BUFFER%blc)(:,:,BUFFER%I),DEV_VARNAME(Xo%blc)(:,:,1))
  call LINEAR_ALGEBRA_driver(LIN_SYS,M_slk=KERNEL,B_slk=BUFFER)
  !
else
  !
  ! Matrix Inversion + Matmul
  !
  call LINEAR_ALGEBRA_driver(INV,M_slk=KERNEL)
  call LINEAR_ALGEBRA_driver(MAT_MUL,M_slk=KERNEL,B_slk=Xo,C_slk=BUFFER)
  !
endif
```

examples: Yambo X_redux

```
  !
  ! Update X by multiplying left and right for v_coul^1/2 (diagonal in reciprocal space)
  !
  if (have_cuda) then
     !
!     call dev_mem_cpy_dMd(Xo_rows(2)-Xo_rows(1)+1,Xo_cols(2)-Xo_cols(1)+1,BUFFER%blc_d(:,:,iw_par),&
!&                         bare_qpg_d(Xo_rows(1):,iq),'R',bare_qpg_d(Xo_cols(1):,iq),'R',scal=cmplx(4._SP*pi,kind=SP))
     !
     BUFFER_blc_d => BUFFER%blc_d
     !
     !$acc kernels do(2)
     do ig2=Xo_cols(1),Xo_cols(2)
       do ig1=Xo_rows(1),Xo_rows(2)
         BUFFER_blc_d(ig1,ig2,iw_par)=BUFFER_blc_d(ig1,ig2,iw_par)*4._SP*pi/bare_qpg_d(ig1,iq)/bare_qpg_d(ig2,iq)
       enddo
     enddo
     !
     call dev_acxfer_async(acxfer_default)
     call dev_memcpy_async(BUFFER%blc(:,:,BUFFER%T),BUFFER%blc_d(:,:,BUFFER%T), stream_d2h)
     !
  else
     !
     !$omp parallel do default(shared), private(ig1,ig2), collapse(2)
     do ig2=Xo%cols(1),Xo%cols(2)
       do ig1=Xo%rows(1),Xo%rows(2)
         BUFFER%blc(ig1,ig2,iw_par)=BUFFER%blc(ig1,ig2,iw_par)*4._SP*pi/bare_qpg(iq,ig1)/bare_qpg(iq,ig2)
       enddo
     enddo
     !$omp end parallel do
     !
  endif
```