GPU infrastructure code and performance for NIMROD

Jacob King (Tech-X Corporation)

With contributions from
Eric Howell (Tech-X Corporation)
Brian Cornille (Univ. of Wisconsin)

April 3rd 2022
CTTS SciDAC Meeting
Work supported by US DOE

Next gen computing: SciDAC codes need to exploit GPUs

- Perlmutter (NERSC), Frontier (OLCF) and Aurora (ALCF) contain GPUs
  - Three different hardware vendors (Nvidia, AMD, Intel)
  - Three different software eco-systems
- Dust yet to settle on unified programming paradigm
  - CUDA, HIP, SYCL, OpenCL, OpenACC, OpenMP 5, language standards
  - We’ve explored OpenACC for NIMROD – support by nvhpc / gcc
  - May switch to OpenMP 5 or language standards as dust settles

Effort splits NIMROD code into infrastructure and physics repos

- Physics: anything with units
- Infrastructure: FE evaluation & integration, linear algebra, parallel decomp, FFTs, inverse mapping, etc.
- Infrastructure repo now open-source, open-access
  - See https://gitlab.com/NIMRODteam/nimrod-abstract
  - Allows access by ASCR/computing facility partners and compiler teams
  - Useful for hackathons and diagnosing performance and compiler issues
- Using gitlab enables modern software tools
  - Code review
  - Issue tracking
  - Continuous integration testing
  - Code coverage reports

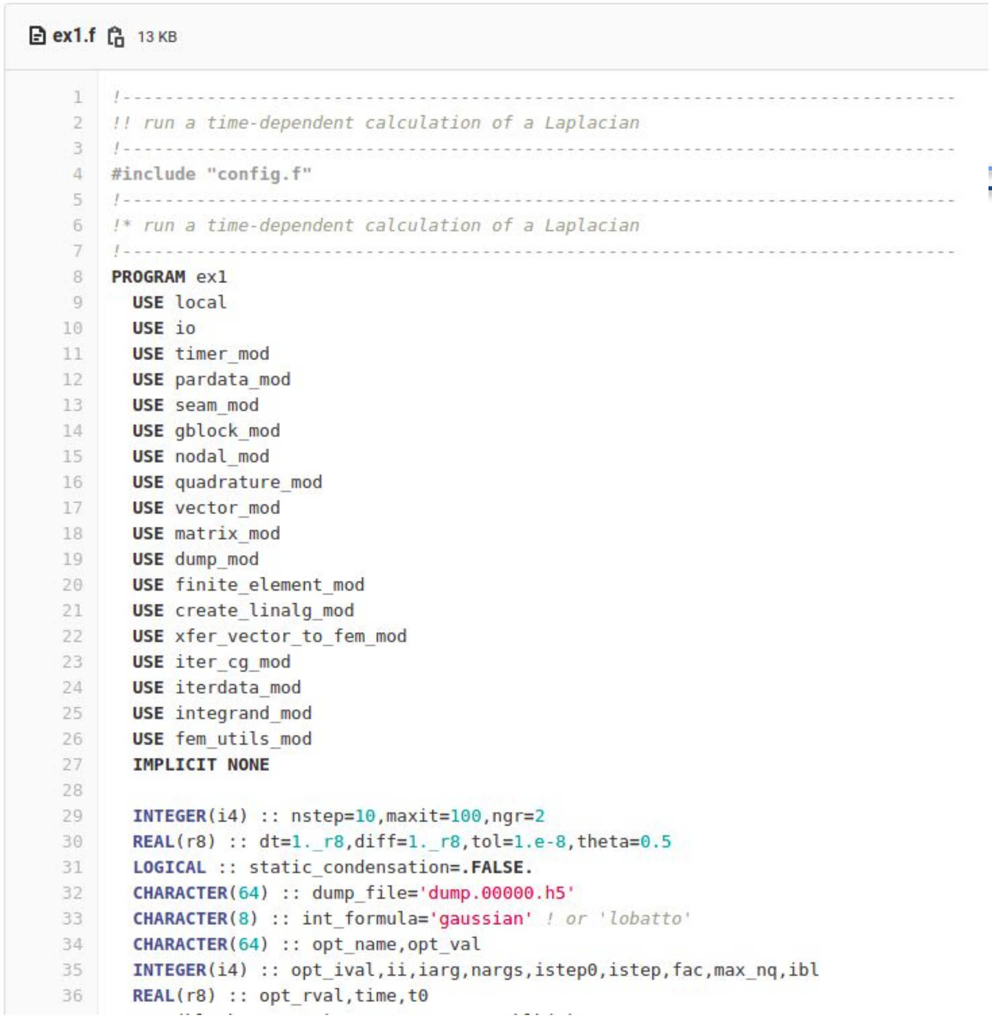
Infrastructure repo close to maturity

- Relies on modern Fortran abstract types to enable flexibility
  - Define interface at abstract (virtual) level
  - Now: working on version with real types
- Investigating API on GPU before expanding concrete base classes
- Done: I/O, seams, MPI with mpi_f08, timing, FEM types, linear algebra, Dirichlet boundary conditions, regularity
- Remaining big tasks: preconditioning, FFTs, surface integrals
- Next: report on progress on GPU by overviewing example application
- Example application: solve time-dependent Laplace equation in a periodic box

Thanks to the full GPU hackathon team!

For the openACC material here, I’d specifically like to thank Eric Howell, Brian Cornille, Torrin Bechtel, Robbie Searles (Nvidia) and Vassillios Mewes (OLCF)

Example overview

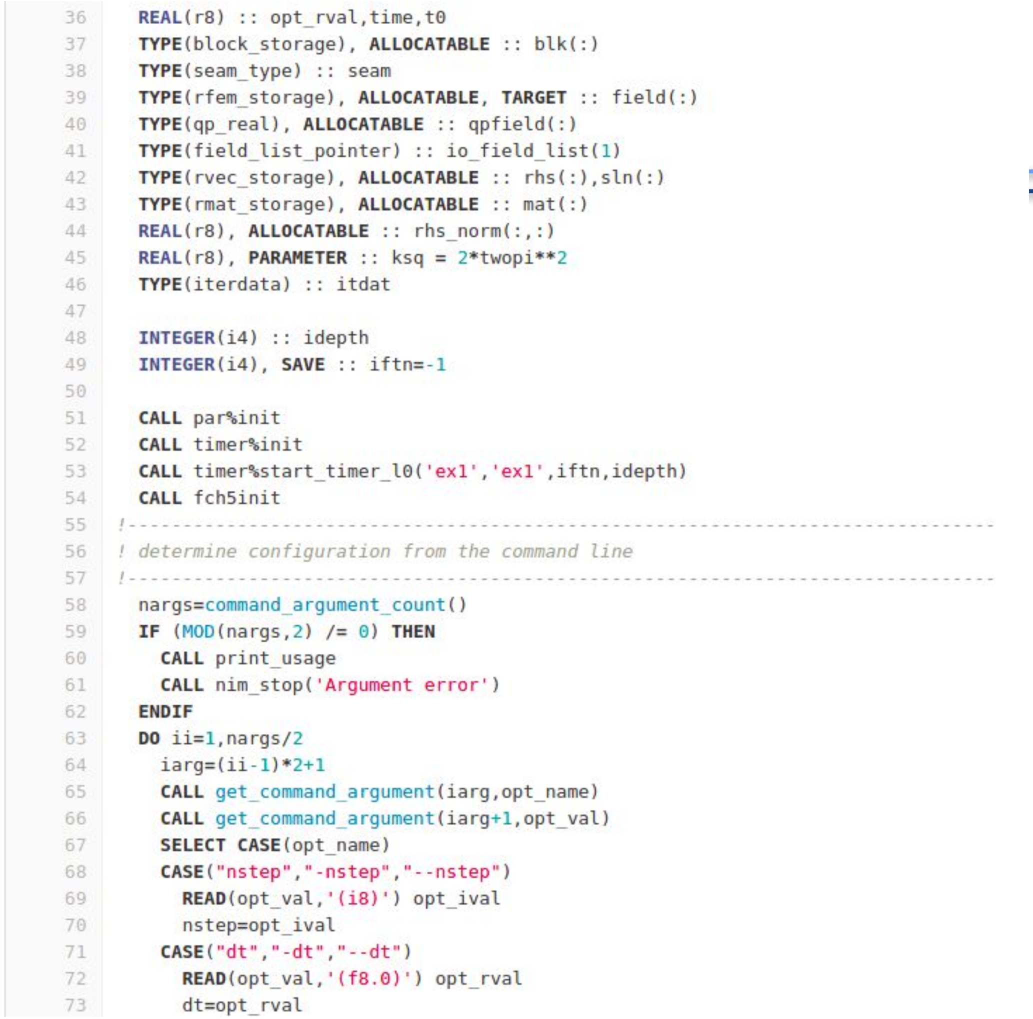


ex1.f 13 KB

```fortran
!-------------------------------------------------------------------------------
!! run a time-dependent calculation of a Laplacian
!-------------------------------------------------------------------------------
#include "config.f"
!-------------------------------------------------------------------------------
!* run a time-dependent calculation of a Laplacian
!-------------------------------------------------------------------------------
PROGRAM ex1
  USE local
  USE io
  USE timer_mod
  USE pardata_mod
  USE seam_mod
  USE gblock_mod
  USE nodal_mod
  USE quadrature_mod
  USE vector_mod
  USE matrix_mod
  USE dump_mod
  USE finite_element_mod
  USE create_linalg_mod
  USE xfer_vector_to_fem_mod
  USE iter_cg_mod
  USE iterdata_mod
  USE integrand_mod
  USE fem_utils_mod
  IMPLICIT NONE

  INTEGER(i4) :: nstep=10,maxit=100,ngr=2
  REAL(r8) :: dt=1._r8,diff=1._r8,tol=1.e-8,theta=0.5
  LOGICAL :: static_condensation=.FALSE.
  CHARACTER(64) :: dump_file='dump.00000.h5'
  CHARACTER(8) :: int_formula='gaussian' ! or 'lobatto'
  CHARACTER(64) :: opt_name,opt_val
  INTEGER(i4) :: opt_ival,ii,iarg,nargs,istep0,istep,fac,max_nq,ibl
  REAL(r8) :: opt_rval,time,t0
```

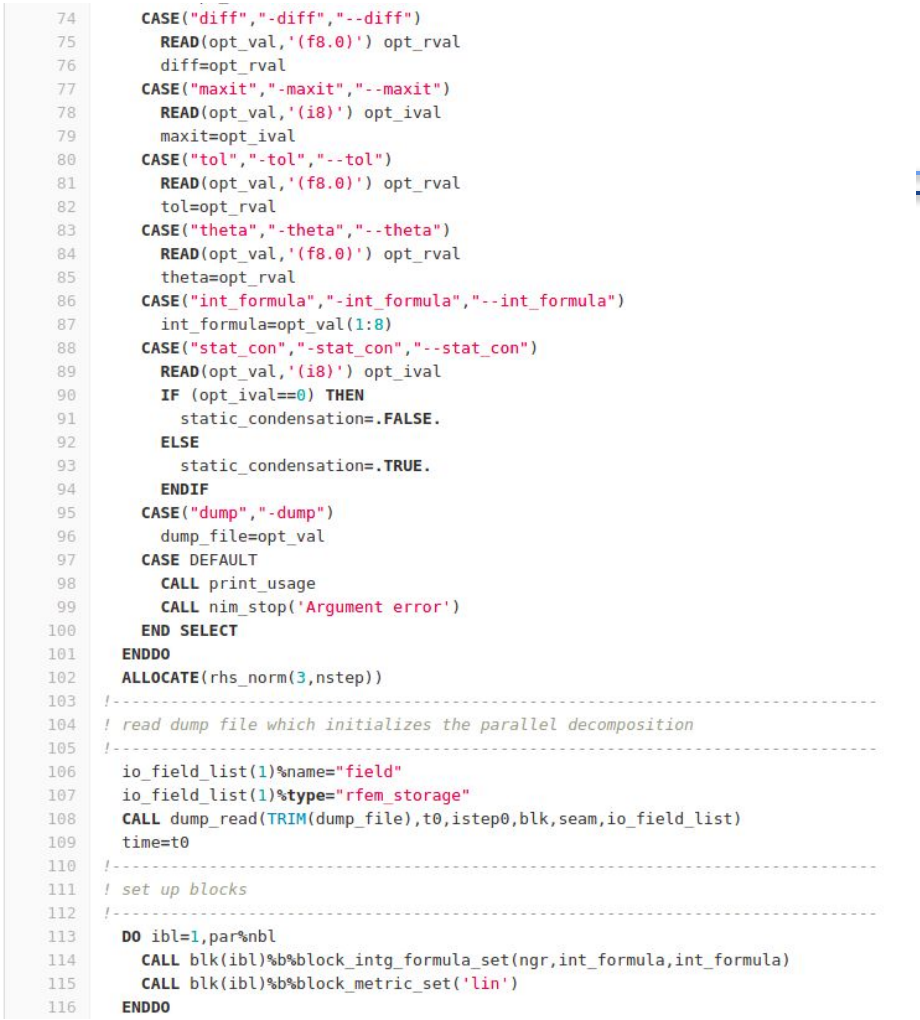
- All module import ABSTRACT base types at the example level

Example overview



```fortran
  REAL(r8) :: opt_rval,time,t0
  TYPE(block_storage), ALLOCATABLE :: blk(:)
  TYPE(seam_type) :: seam
  TYPE(rfem_storage), ALLOCATABLE, TARGET :: field(:)
  TYPE(qp_real), ALLOCATABLE :: qpfield(:)
  TYPE(field_list_pointer) :: io_field_list(1)
  TYPE(rvec_storage), ALLOCATABLE :: rhs(:),sln(:)
  TYPE(rmat_storage), ALLOCATABLE :: mat(:)
  REAL(r8), ALLOCATABLE :: rhs_norm(:,:)
  REAL(r8), PARAMETER :: ksq = 2*twopi**2
  TYPE(iterdata) :: itdat

  INTEGER(i4) :: idepth
  INTEGER(i4), SAVE :: iftn=-1

  CALL par%init
  CALL timer%init
  CALL timer%start_timer_l0('ex1','ex1',iftn,idepth)
  CALL fch5init
!-------------------------------------------------------------------------------
! determine configuration from the command line
!-------------------------------------------------------------------------------
  nargs=command_argument_count()
  IF (MOD(nargs,2) /= 0) THEN
    CALL print_usage
    CALL nim_stop('Argument error')
  ENDIF
  DO ii=1,nargs/2
    iarg=(ii-1)*2+1
    CALL get_command_argument(iarg,opt_name)
    CALL get_command_argument(iarg+1,opt_val)
    SELECT CASE(opt_name)
    CASE("nstep","-nstep","--nstep")
      READ(opt_val,'(i8)') opt_ival
      nstep=opt_ival
    CASE("dt","-dt","--dt")
      READ(opt_val,'(f8.0)') opt_rval
      dt=opt_rval
```

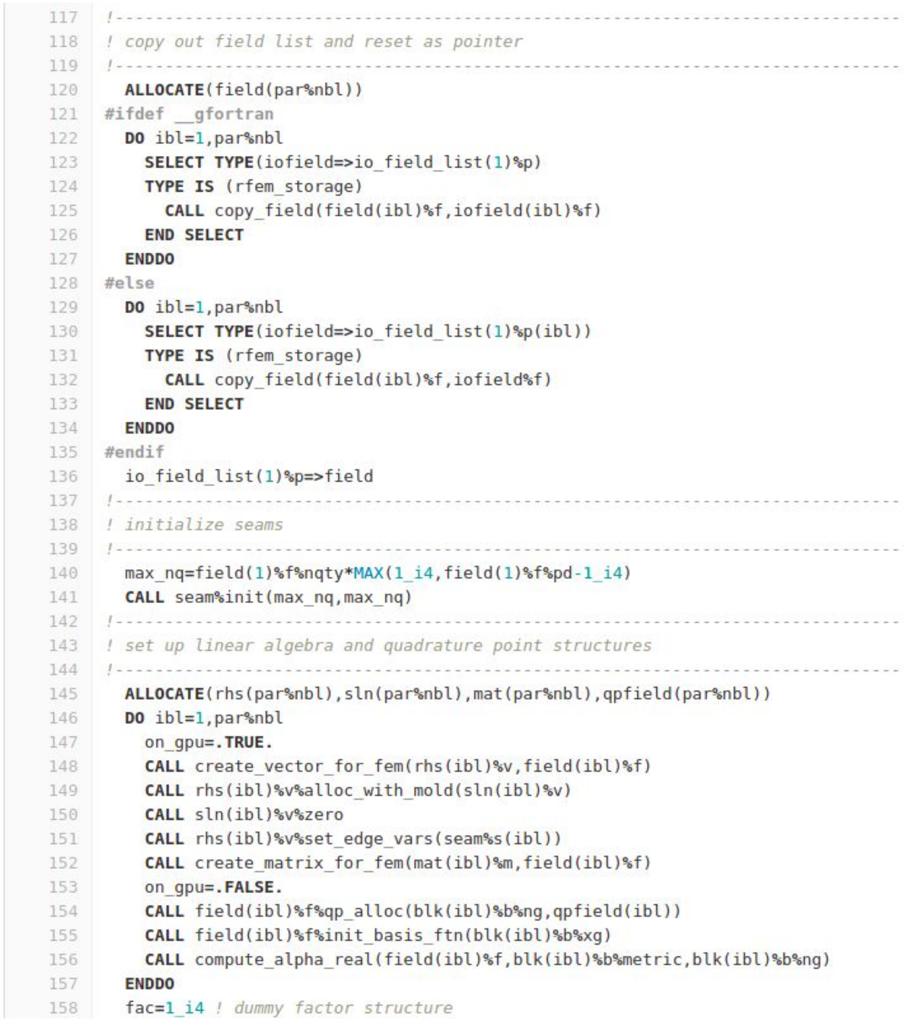
- All data types are only know at the abstract level
- Storage arrays allow different concrete instantiations for different blocks
- Timer and parallel types require initialization of singleton objects
- Example takes command line input (no namelist)



```fortran
    CASE("diff","-diff","--diff")
      READ(opt_val,'(f8.0)') opt_rval
      diff=opt_rval
    CASE("maxit","-maxit","--maxit")
      READ(opt_val,'(i8)') opt_ival
      maxit=opt_ival
    CASE("tol","-tol","--tol")
      READ(opt_val,'(f8.0)') opt_rval
      tol=opt_rval
    CASE("theta","-theta","--theta")
      READ(opt_val,'(f8.0)') opt_rval
      theta=opt_rval
    CASE("int_formula","-int_formula","--int_formula")
      int_formula=opt_val(1:8)
    CASE("stat_con","-stat_con","--stat_con")
      READ(opt_val,'(i8)') opt_ival
      IF (opt_ival==0) THEN
        static_condensation=.FALSE.
      ELSE
        static_condensation=.TRUE.
      ENDIF
    CASE("dump","-dump")
      dump_file=opt_val
    CASE DEFAULT
      CALL print_usage
      CALL nim_stop('Argument error')
    END SELECT
  ENDDO
  ALLOCATE(rhs_norm(3,nstep))
!-----------------------------------------------------------------------
! read dump file which initializes the parallel decomposition
!-----------------------------------------------------------------------
  io_field_list(1)%name="field"
  io_field_list(1)%type="rfem_storage"
  CALL dump_read(TRIM(dump_file),t0,istep0,blk,seam,io_field_list)
  time=t0
!-----------------------------------------------------------------------
! set up blocks
!-----------------------------------------------------------------------
  DO ibl=1,par%nbl
    CALL blk(ibl)%b%block_intg_formula_set(ngr,int_formula,int_formula)
    CALL blk(ibl)%b%block_metric_set('lin')
  ENDDO
```

Example overview

- Mesh and block decomposition set by initialization
- “dump_read” also initializes parallel block decomposition
- Blocks govern FEM integration and setting quadrature rules and weights mirrors non-abstract code
- This structure allows for natural separate of blocks (e.g. CK type bound to a CK block with CK integration rules)

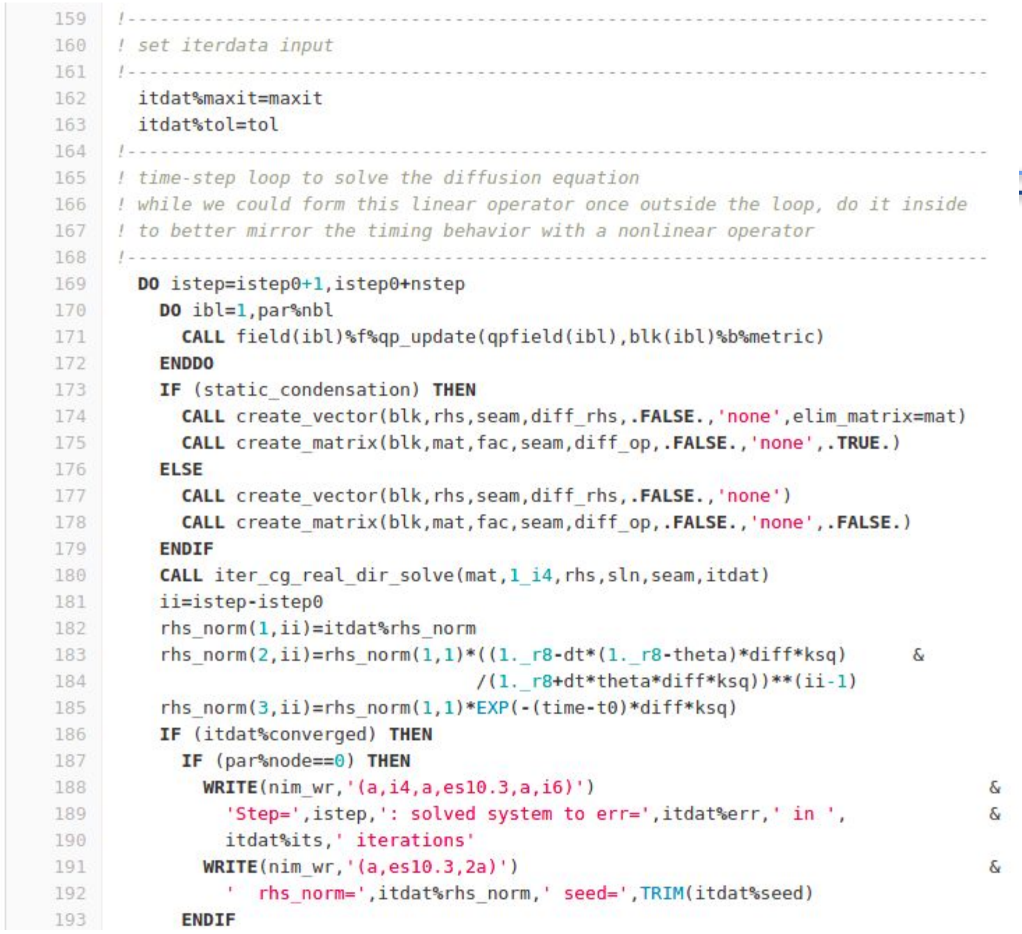


```fortran
!---------------------------------------------------------------------------
! copy out field list and reset as pointer
!---------------------------------------------------------------------------
  ALLOCATE(field(par%nbl))
#ifdef __gfortran
  DO ibl=1,par%nbl
    SELECT TYPE(iofield=>io_field_list(1)%p)
    TYPE IS (rfem_storage)
      CALL copy_field(field(ibl)%f,iofield(ibl)%f)
    END SELECT
  ENDDO
#else
  DO ibl=1,par%nbl
    SELECT TYPE(iofield=>io_field_list(1)%p(ibl))
    TYPE IS (rfem_storage)
      CALL copy_field(field(ibl)%f,iofield%f)
    END SELECT
  ENDDO
#endif
  io_field_list(1)%p=>field
!---------------------------------------------------------------------------
! initialize seams
!---------------------------------------------------------------------------
  max_nq=field(1)%f%nqty*MAX(1_i4,field(1)%f%pd-1_i4)
  CALL seam%init(max_nq,max_nq)
!---------------------------------------------------------------------------
! set up linear algebra and quadrature point structures
!---------------------------------------------------------------------------
  ALLOCATE(rhs(par%nbl),sln(par%nbl),mat(par%nbl),qpfield(par%nbl))
  DO ibl=1,par%nbl
    on_gpu=.TRUE.
    CALL create_vector_for_fem(rhs(ibl)%v,field(ibl)%f)
    CALL rhs(ibl)%v%alloc_with_mold(sln(ibl)%v)
    CALL sln(ibl)%v%zero
    CALL rhs(ibl)%v%set_edge_vars(seam%s(ibl))
    CALL create_matrix_for_fem(mat(ibl)%m,field(ibl)%f)
    on_gpu=.FALSE.
    CALL field(ibl)%f%qp_alloc(blk(ibl)%b%ng,qpfield(ibl))
    CALL field(ibl)%f%init_basis_ftn(blk(ibl)%b%xg)
    CALL compute_alpha_real(field(ibl)%f,blk(ibl)%b%metric,blk(ibl)%b%ng)
  ENDDO
  fac=1_i4 ! dummy factor structure
```

Example overview

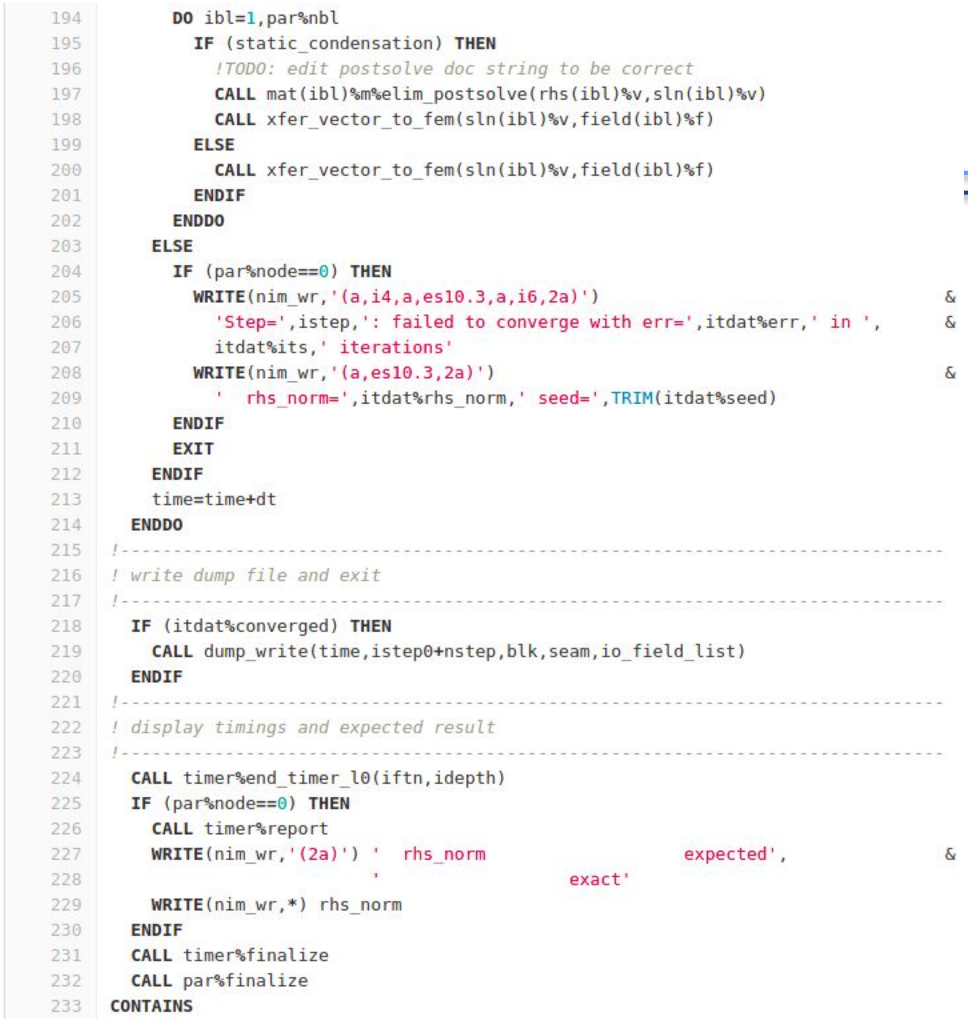
- Minor penalty for not binding fields to blocks: one extra copy after read
- #ifdefs based on compiler demonstrate difficulty with modern Fortran
- Vectors and matrices are initialized from FE field
- Each field (or combination of fields) needs to know how to map to appropriate linear algebra structures
  - See create_*_for_fem
- Basis functions are precomputed based on FE field

Example overview



```fortran
!-------------------------------------------------------------------------------
! set iterdata input
!-------------------------------------------------------------------------------
  itdat%maxit=maxit
  itdat%tol=tol
!-------------------------------------------------------------------------------
! time-step loop to solve the diffusion equation
! while we could form this linear operator once outside the loop, do it inside
! to better mirror the timing behavior with a nonlinear operator
!-------------------------------------------------------------------------------
  DO istep=istep0+1,istep0+nstep
    DO ibl=1,par%nbl
      CALL field(ibl)%f%qp_update(qpfield(ibl),blk(ibl)%b%metric)
    ENDDO
    IF (static_condensation) THEN
      CALL create_vector(blk,rhs,seam,diff_rhs,.FALSE.,'none',elim_matrix=mat)
      CALL create_matrix(blk,mat,fac,seam,diff_op,.FALSE.,'none',.TRUE.)
    ELSE
      CALL create_vector(blk,rhs,seam,diff_rhs,.FALSE.,'none')
      CALL create_matrix(blk,mat,fac,seam,diff_op,.FALSE.,'none',.FALSE.)
    ENDIF
    CALL iter_cg_real_dir_solve(mat,1_i4,rhs,sln,seam,itdat)
    ii=istep-istep0
    rhs_norm(1,ii)=itdat%rhs_norm
    rhs_norm(2,ii)=rhs_norm(1,1)*((1._r8-dt*(1._r8-theta)*diff*ksq)       &
                                 /(1._r8+dt*theta*diff*ksq))**(ii-1)
    rhs_norm(3,ii)=rhs_norm(1,1)*EXP(-(time-t0)*diff*ksq)
    IF (itdat%converged) THEN
      IF (par%node==0) THEN
        WRITE(nim_wr,'(a,i4,a,es10.3,a,i6)')                                    &
          'Step=',istep,': solved system to err=',itdat%err,' in ',             &
          itdat%its,' iterations'
        WRITE(nim_wr,'(a,es10.3,2a)')                                           &
          '  rhs_norm=',itdat%rhs_norm,' seed=',TRIM(itdat%seed)
      ENDIF
```

- Time step loop:
  - update quadrature storage
  - Create matrix/RHS
  - Call iterative solver
  - Use RHS norm (inf_norm in this case) to test solution
  - Check convergence

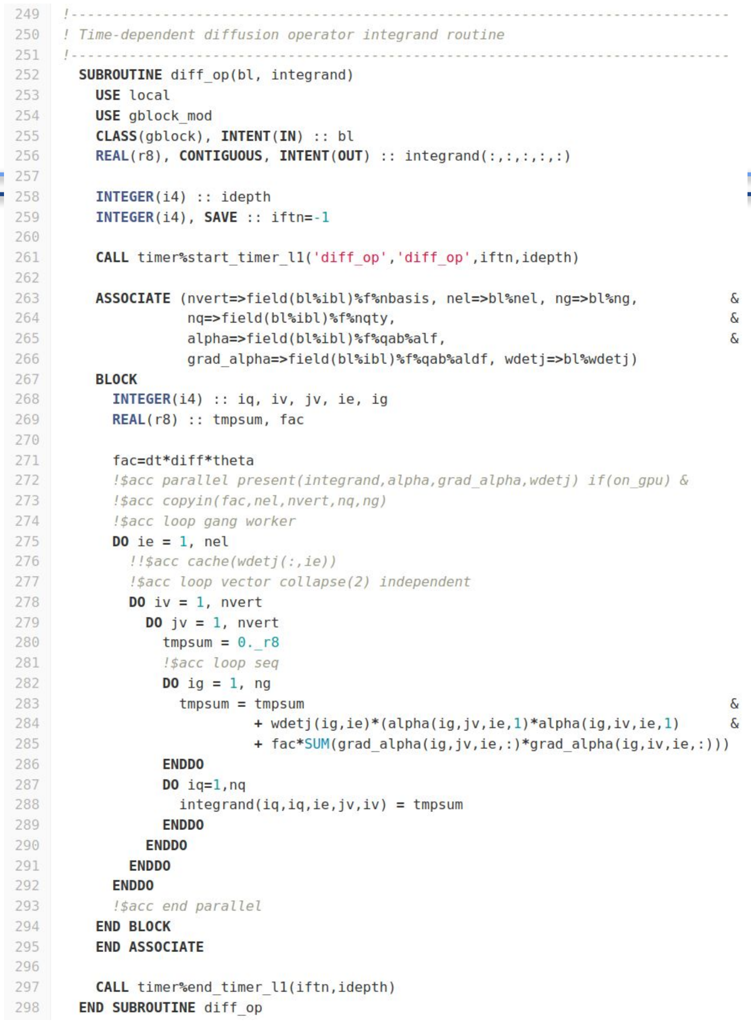


```fortran
      DO ibl=1,par%nbl
        IF (static_condensation) THEN
          !TODO: edit postsolve doc string to be correct
          CALL mat(ibl)%m%elim_postsolve(rhs(ibl)%v,sln(ibl)%v)
          CALL xfer_vector_to_fem(sln(ibl)%v,field(ibl)%f)
        ELSE
          CALL xfer_vector_to_fem(sln(ibl)%v,field(ibl)%f)
        ENDIF
      ENDDO
    ELSE
      IF (par%node==0) THEN
        WRITE(nim_wr,'(a,i4,a,es10.3,a,i6,2a)')                                  &
          'Step=',istep,': failed to converge with err=',itdat%err,' in ',      &
          itdat%its,' iterations'
        WRITE(nim_wr,'(a,es10.3,2a)')                                            &
          '   rhs_norm=',itdat%rhs_norm,' seed=',TRIM(itdat%seed)
      ENDIF
      EXIT
    ENDIF
    time=time+dt
  ENDDO
!-------------------------------------------------------------------------------
! write dump file and exit
!-------------------------------------------------------------------------------
  IF (itdat%converged) THEN
    CALL dump_write(time,istep0+nstep,blk,seam,io_field_list)
  ENDIF
!-------------------------------------------------------------------------------
! display timings and expected result
!-------------------------------------------------------------------------------
  CALL timer%end_timer_l0(iftn,idepth)
  IF (par%node==0) THEN
    CALL timer%report
    WRITE(nim_wr,'(2a)') '  rhs_norm                  expected',                &
                         '                 exact'
    WRITE(nim_wr,*) rhs_norm
  ENDIF
  CALL timer%finalize
  CALL par%finalize
CONTAINS
```

Example overview

- After time step loop, we test solution

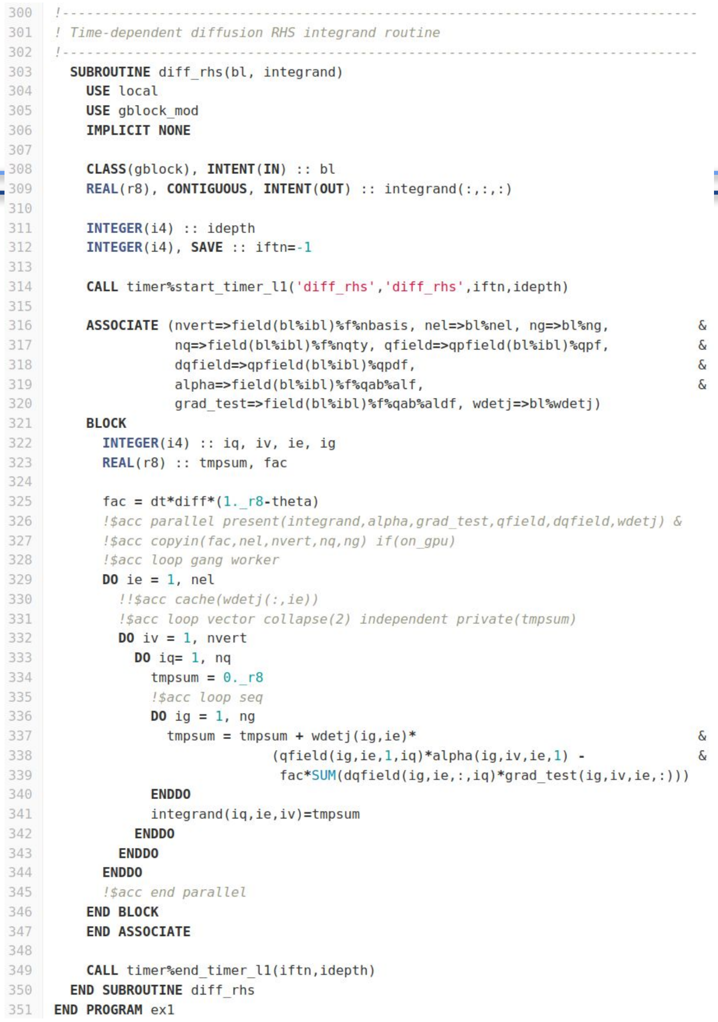
Wait! Where’s the GPU stuff?!?



```fortran
!-----------------------------------------------------------------------
! Time-dependent diffusion operator integrand routine
!-----------------------------------------------------------------------
  SUBROUTINE diff_op(bl, integrand)
    USE local
    USE gblock_mod
    CLASS(gblock), INTENT(IN) :: bl
    REAL(r8), CONTIGUOUS, INTENT(OUT) :: integrand(:,:,:,:,:)

    INTEGER(i4) :: idepth
    INTEGER(i4), SAVE :: iftn=-1

    CALL timer%start_timer_l1('diff_op','diff_op',iftn,idepth)

    ASSOCIATE (nvert=>field(bl%ibl)%f%nbasis, nel=>bl%nel, ng=>bl%ng,          &
               nq=>field(bl%ibl)%f%nqty,                                       &
               alpha=>field(bl%ibl)%f%qab%alf,                                 &
               grad_alpha=>field(bl%ibl)%f%qab%aldf, wdetj=>bl%wdetj)
    BLOCK
      INTEGER(i4) :: iq, iv, jv, ie, ig
      REAL(r8) :: tmpsum, fac

      fac=dt*diff*theta
      !$acc parallel present(integrand,alpha,grad_alpha,wdetj) if(on_gpu) &
      !$acc copyin(fac,nel,nvert,nq,ng)
      !$acc loop gang worker
      DO ie = 1, nel
        !!$acc cache(wdetj(:,ie))
        !$acc loop vector collapse(2) independent
        DO iv = 1, nvert
          DO jv = 1, nvert
            tmpsum = 0._r8
            !$acc loop seq
            DO ig = 1, ng
              tmpsum = tmpsum                                                  &
                       + wdetj(ig,ie)*(alpha(ig,jv,ie,1)*alpha(ig,iv,ie,1)     &
                       + fac*SUM(grad_alpha(ig,jv,ie,:)*grad_alpha(ig,iv,ie,:)))
            ENDDO
            DO iq=1,nq
              integrand(iq,iq,ie,jv,iv) = tmpsum
            ENDDO
          ENDDO
        ENDDO
      ENDDO
      !$acc end parallel
    END BLOCK
    END ASSOCIATE

    CALL timer%end_timer_l1(iftn,idepth)
  END SUBROUTINE diff_op
```

Example overview

- Physics application is not completely exempt from being GPU aware, openACC statements required in integrand routines
- Parallel clause create GPU kernel
- present/copyin clauses manage data movement from host to device
- Structure of acc loops follow pre-set pattern
- With full time step loop on device, minimal copyin statements will be needed (more on this later)



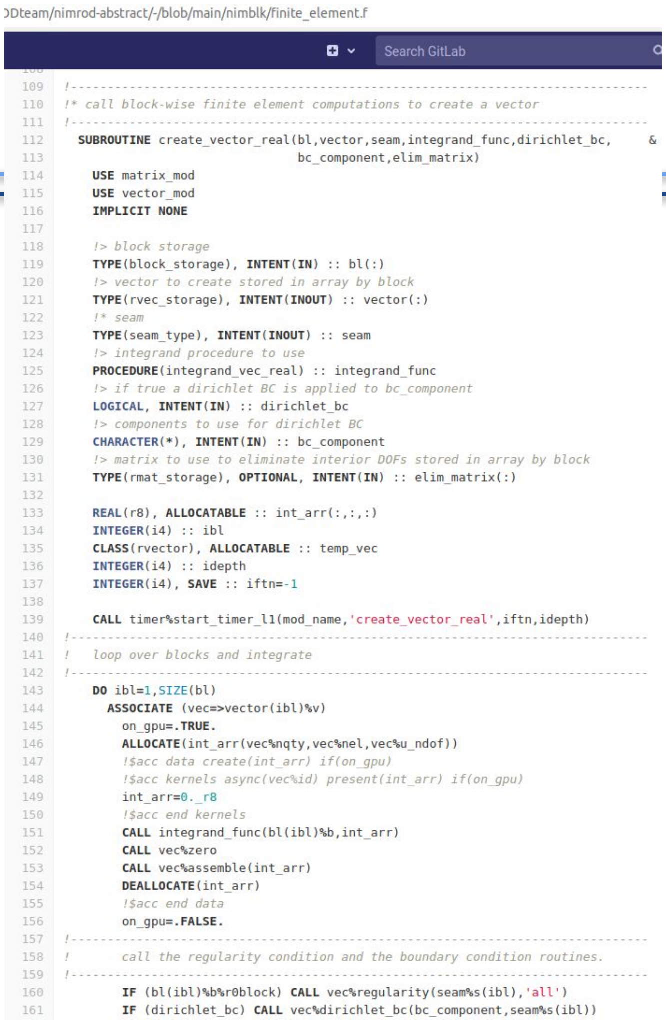
```
!--------------------------------------------------------------------------------
! Time-dependent diffusion RHS integrand routine
!--------------------------------------------------------------------------------
  SUBROUTINE diff_rhs(bl, integrand)
    USE local
    USE gblock_mod
    IMPLICIT NONE

    CLASS(gblock), INTENT(IN) :: bl
    REAL(r8), CONTIGUOUS, INTENT(OUT) :: integrand(:,:,:)

    INTEGER(i4) :: idepth
    INTEGER(i4), SAVE :: iftn=-1

    CALL timer%start_timer_l1('diff_rhs','diff_rhs',iftn,idepth)

    ASSOCIATE (nvert=>field(bl%ibl)%f%nbasis, nel=>bl%nel, ng=>bl%ng,           &
               nq=>field(bl%ibl)%f%nqty, qfield=>qpfield(bl%ibl)%qpf,           &
               dqfield=>qpfield(bl%ibl)%qpdf,                                   &
               alpha=>field(bl%ibl)%f%qab%alf,                                  &
               grad_test=>field(bl%ibl)%f%qab%aldf, wdetj=>bl%wdetj)
    BLOCK
      INTEGER(i4) :: iq, iv, ie, ig
      REAL(r8) :: tmpsum, fac

      fac = dt*diff*(1._r8-theta)
      !$acc parallel present(integrand,alpha,grad_test,qfield,dqfield,wdetj) &
      !$acc copyin(fac,nel,nvert,nq,ng) if(on_gpu)
      !$acc loop gang worker
      DO ie = 1, nel
        !!$acc cache(wdetj(:,ie))
        !$acc loop vector collapse(2) independent private(tmpsum)
        DO iv = 1, nvert
          DO iq= 1, nq
            tmpsum = 0._r8
            !$acc loop seq
            DO ig = 1, ng
              tmpsum = tmpsum + wdetj(ig,ie)*                                   &
                           (qfield(ig,ie,1,iq)*alpha(ig,iv,ie,1) -              &
                            fac*SUM(dqfield(ig,ie,:,iq)*grad_test(ig,iv,ie,:)))
            ENDDO
            integrand(iq,ie,iv)=tmpsum
          ENDDO
        ENDDO
      ENDDO
      !$acc end parallel
    END BLOCK
    END ASSOCIATE

    CALL timer%end_timer_l1(iftn,idepth)
  END SUBROUTINE diff_rhs
END PROGRAM ex1
```

Example overview

- For completeness, here's the RHS routine
- It is pretty similar, but quadrature point data from field is used

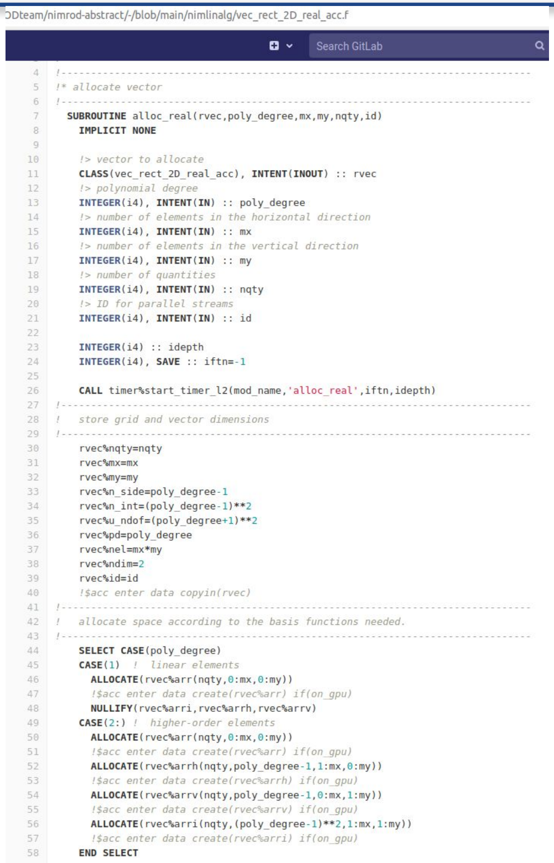
- Majority of OpenACC code in infrastructure repo.
- Management of data locality is key (next slide)

Device data management


ODteam/nimrod-abstract/-/blob/main/nimblk/finite_element.f

```fortran
!-------------------------------------------------------------------------------
!* call block-wise finite element computations to create a vector
!-------------------------------------------------------------------------------
  SUBROUTINE create_vector_real(bl,vector,seam,integrand_func,dirichlet_bc,     &
                                bc_component,elim_matrix)
    USE matrix_mod
    USE vector_mod
    IMPLICIT NONE

    !> block storage
    TYPE(block_storage), INTENT(IN) :: bl(:)
    !> vector to create stored in array by block
    TYPE(rvec_storage), INTENT(INOUT) :: vector(:)
    !* seam
    TYPE(seam_type), INTENT(INOUT) :: seam
    !> integrand procedure to use
    PROCEDURE(integrand_vec_real) :: integrand_func
    !> if true a dirichlet BC is applied to bc_component
    LOGICAL, INTENT(IN) :: dirichlet_bc
    !> components to use for dirichlet BC
    CHARACTER(*), INTENT(IN) :: bc_component
    !> matrix to use to eliminate interior DOFs stored in array by block
    TYPE(rmat_storage), OPTIONAL, INTENT(IN) :: elim_matrix(:)

    REAL(r8), ALLOCATABLE :: int_arr(:,:,:)
    INTEGER(i4) :: ibl
    CLASS(rvector), ALLOCATABLE :: temp_vec
    INTEGER(i4) :: idepth
    INTEGER(i4), SAVE :: iftn=-1

    CALL timer%start_timer_l1(mod_name,'create_vector_real',iftn,idepth)
!-------------------------------------------------------------------------------
!   loop over blocks and integrate
!-------------------------------------------------------------------------------
    DO ibl=1,SIZE(bl)
      ASSOCIATE (vec=>vector(ibl)%v)
        on_gpu=.TRUE.
        ALLOCATE(int_arr(vec%nqty,vec%nel,vec%u_ndof))
        !$acc data create(int_arr) if(on_gpu)
        !$acc kernels async(vec%id) present(int_arr) if(on_gpu)
        int_arr=0._r8
        !$acc end kernels
        CALL integrand_func(bl(ibl)%b,int_arr)
        CALL vec%zero
        CALL vec%assemble(int_arr)
        DEALLOCATE(int_arr)
        !$acc end data
        on_gpu=.FALSE.
!-------------------------------------------------------------------------------
!       call the regularity condition and the boundary condition routines.
!-------------------------------------------------------------------------------
        IF (bl(ibl)%b%r0block) CALL vec%regularity(seam%s(ibl),'all')
        IF (dirichlet_bc) CALL vec%dirichlet_bc(bc_component,seam%s(ibl))
```

- Unstructured data blocks:
  - persistent data on GPU
  - “!$acc enter data create”
  - “!$acc exit data delete”
- Structured data blocks:
  - GPU data in local scope
  - “!$acc data create”
  - “!$acc end data”


ODteam/nimrod-abstract/-/blob/main/nimlinalg/vec_rect_2D_real_acc.f

```fortran
!-------------------------------------------------------------------------------
!* allocate vector
!-------------------------------------------------------------------------------
  SUBROUTINE alloc_real(rvec,poly_degree,mx,my,nqty,id)
    IMPLICIT NONE

    !> vector to allocate
    CLASS(vec_rect_2D_real_acc), INTENT(INOUT) :: rvec
    !> polynomial degree
    INTEGER(i4), INTENT(IN) :: poly_degree
    !> number of elements in the horizontal direction
    INTEGER(i4), INTENT(IN) :: mx
    !> number of elements in the vertical direction
    INTEGER(i4), INTENT(IN) :: my
    !> number of quantities
    INTEGER(i4), INTENT(IN) :: nqty
    !> ID for parallel streams
    INTEGER(i4), INTENT(IN) :: id

    INTEGER(i4) :: idepth
    INTEGER(i4), SAVE :: iftn=-1

    CALL timer%start_timer_l2(mod_name,'alloc_real',iftn,idepth)
!-------------------------------------------------------------------------------
!   store grid and vector dimensions
!-------------------------------------------------------------------------------
    rvec%nqty=nqty
    rvec%mx=mx
    rvec%my=my
    rvec%n_side=poly_degree-1
    rvec%n_int=(poly_degree-1)**2
    rvec%u_ndof=(poly_degree+1)**2
    rvec%pd=poly_degree
    rvec%nel=mx*my
    rvec%ndim=2
    rvec%id=id
    !$acc enter data copyin(rvec)
!-------------------------------------------------------------------------------
!   allocate space according to the basis functions needed.
!-------------------------------------------------------------------------------
    SELECT CASE(poly_degree)
    CASE(1)  !  linear elements
      ALLOCATE(rvec%arr(nqty,0:mx,0:my))
      !$acc enter data create(rvec%arr) if(on_gpu)
      NULLIFY(rvec%arri,rvec%arrh,rvec%arrv)
    CASE(2:) !  higher-order elements
      ALLOCATE(rvec%arr(nqty,0:mx,0:my))
      !$acc enter data create(rvec%arr) if(on_gpu)
      ALLOCATE(rvec%arrh(nqty,poly_degree-1,1:mx,0:my))
      !$acc enter data create(rvec%arrh) if(on_gpu)
      ALLOCATE(rvec%arrv(nqty,poly_degree-1,0:mx,1:my))
      !$acc enter data create(rvec%arrv) if(on_gpu)
      ALLOCATE(rvec%arri(nqty,(poly_degree-1)**2,1:mx,1:my))
      !$acc enter data create(rvec%arri) if(on_gpu)
    END SELECT
```

GPU port progress

- Integrand, vector assemble kernels on GPU
- Left to port: matvec, qp_update, matrix assemble
- Testing on Ascent at ORNL
- Kernels on 1x V100 GPU comparable or 2x faster than 32 POWER9 cores
  - CPU performance measured with timer
  - GPU performance measured with Nsight Compute
  - 64x64 FE mesh with bi-quintic basis functions
- But there’s 6x GPU/node but only 42 POWER9 cores
- Overall application performance degraded on GPU
  - Until full time-step loop on GPU additional host-device memory transfers
  - After time-step loop on GPU plan optimization