OpenMP® Offloading Support for VASP Using Cray Compiler

Presenter: Mahdieh Ghazimirsaeed

January 2023

AMD
together we advance_

Contributors

- Leopold Grinberg
- Bill Brantley
- Michael Klemm
- Justin Chang
- Ossian OReily
- Paul Mullowney
- Asitav Mishra

Content

- **Introduction**
- Debugging and profiling OpenMP offloading code in VASP
- OpenMP Offloading Challenges in VASP
  - Concurrent support for different directive-based paradigms
  - Enable/disable offloading in different code paths
  - Interface OMP offloading with ROCM libraries
- Compiler related challenges
  - Pointer aliasing
  - Pointer mismatch in subroutine calls
  - Atomic update
  - Declare target
- Data management
- Summary

VASP (Vienna Ab Initio Simulation Package)

- A computer program for atomic scale materials modelling, e.g., electronic structure calculations and quantum-mechanical molecular dynamics
- Currently used by more than 1400 research groups in academia and industry worldwide
- Software license agreements with the University of Vienna
- ~550K lines of FORTRAN 90 code (some FORTRAN 77)

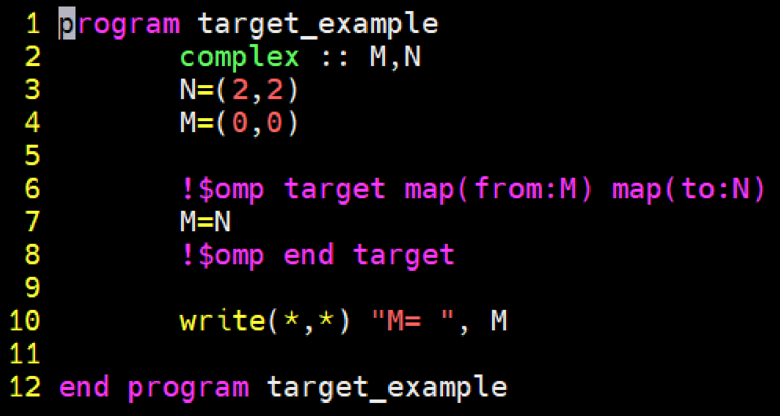
VASP support for directive-based and distributed programming

- Latest version: VASP.6.3.2 released in June 2022
- Supports MPI, OpenMP, and OpenACC
- Support for directive-based programming
  - OpenMP support for execution on the host
  - OpenACC support for execution on GPUs
- Working on adding support for OpenMP offloading to enable VASP execution on GPUs with OpenMP
  - Cray Compiler

Content

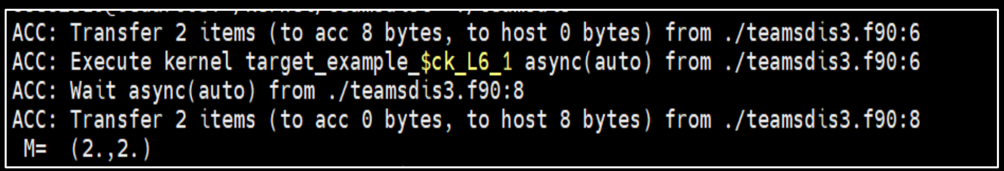
- Introduction
- **Debugging and profiling OpenMP offloading code in VASP**
- OpenMP Offloading Challenges in VASP
    - Concurrent support for different directive-based paradigms
    - Enable/disable offloading in different code paths
    - Interface OMP offloading with ROCM libraries
- Compiler related challenges
    - Pointer aliasing
    - Pointer mismatch in subroutine calls
    - Atomic update
    - Declare target
- Data management
- Summary

Debugging with Cray compiler: CRAY_ACC_DEBUG

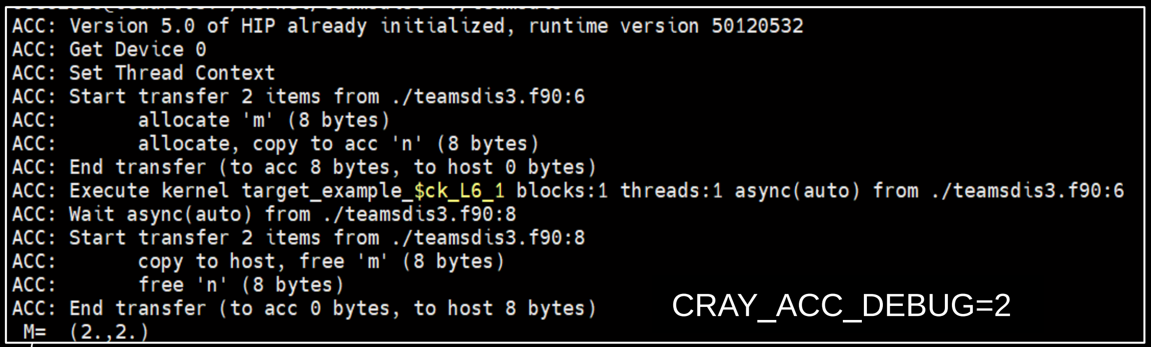


```fortran
program target_example
        complex :: M,N
        N=(2,2)
        M=(0,0)

        !$omp target map(from:M) map(to:N)
        M=N
        !$omp end target

        write(*,*) "M= ", M

end program target_example
```

CRAY_ACC_DEBUG=1

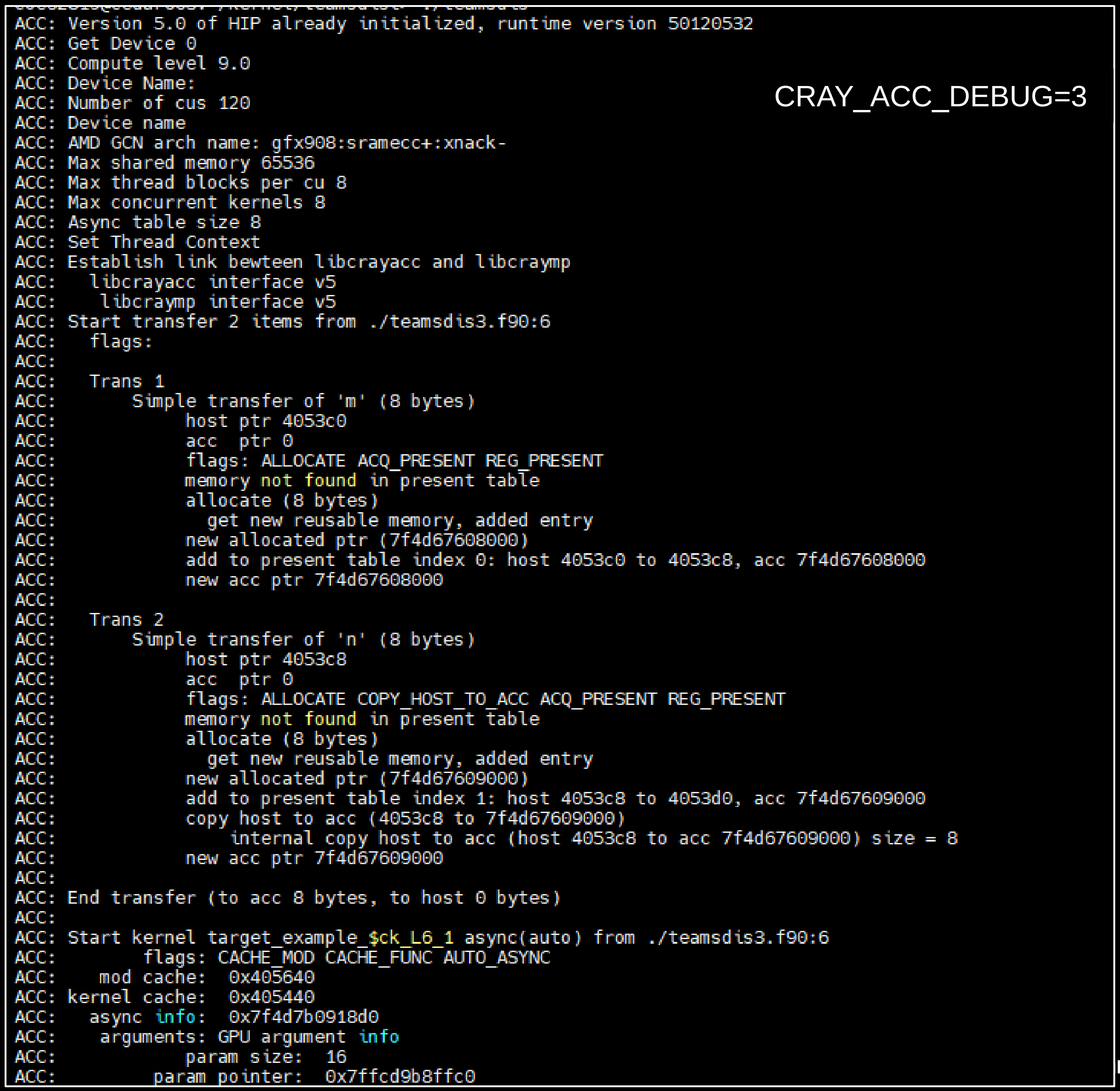


```
ACC: Transfer 2 items (to acc 8 bytes, to host 0 bytes) from ./teamsdis3.f90:6
ACC: Execute kernel target_example_$ck_L6_1 async(auto) from ./teamsdis3.f90:6
ACC: Wait async(auto) from ./teamsdis3.f90:8
ACC: Transfer 2 items (to acc 0 bytes, to host 8 bytes) from ./teamsdis3.f90:8
 M=  (2.,2.)
```



```
ACC: Version 5.0 of HIP already initialized, runtime version 50120532
ACC: Get Device 0
ACC: Set Thread Context
ACC: Start transfer 2 items from ./teamsdis3.f90:6
ACC:       allocate 'm' (8 bytes)
ACC:       allocate, copy to acc 'n' (8 bytes)
ACC: End transfer (to acc 8 bytes, to host 0 bytes)
ACC: Execute kernel target_example_$ck_L6_1 blocks:1 threads:1 async(auto) from ./teamsdis3.f90:6
ACC: Wait async(auto) from ./teamsdis3.f90:8
ACC: Start transfer 2 items from ./teamsdis3.f90:8
ACC:       copy to host, free 'm' (8 bytes)
ACC:       free 'n' (8 bytes)
ACC: End transfer (to acc 0 bytes, to host 8 bytes)
 M=  (2.,2.)
```

CRAY_ACC_DEBUG=2

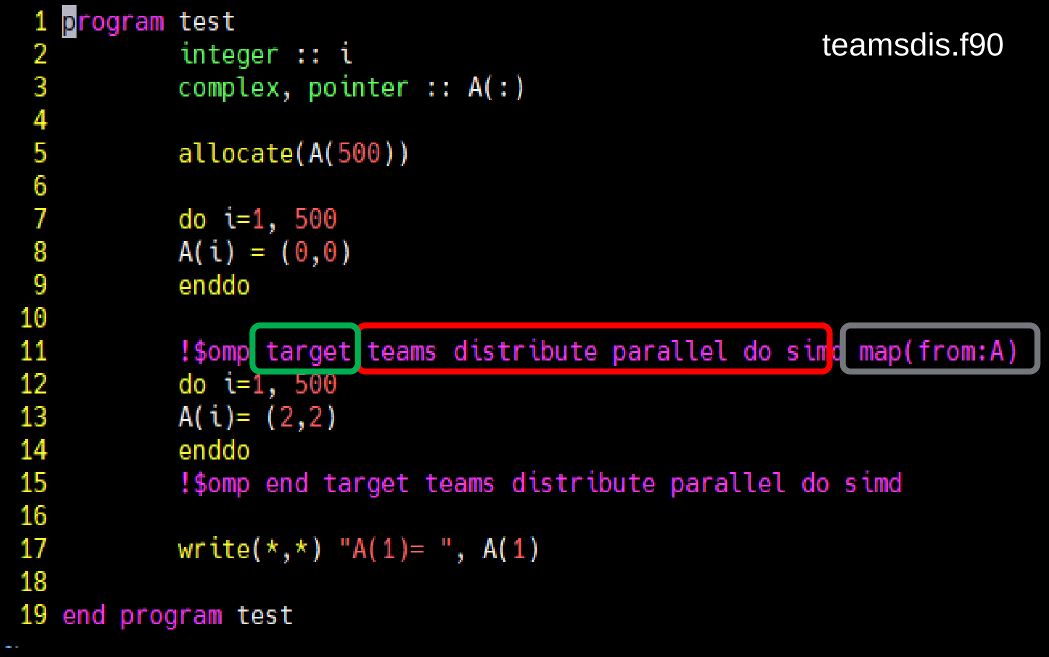


CRAY_ACC_DEBUG=3

```
ACC: Version 5.0 of HIP already initialized, runtime version 50120532
ACC: Get Device 0
ACC: Compute level 9.0
ACC: Device Name:
ACC: Number of cus 120
ACC: Device name
ACC: AMD GCN arch name: gfx908:sramecc+:xnack-
ACC: Max shared memory 65536
ACC: Max thread blocks per cu 8
ACC: Max concurrent kernels 8
ACC: Async table size 8
ACC: Set Thread Context
ACC: Establish link bewteen libcrayacc and libcraymp
ACC:   libcrayacc interface v5
ACC:    libcraymp interface v5
ACC: Start transfer 2 items from ./teamsdis3.f90:6
ACC:   flags:
ACC:
ACC:   Trans 1
ACC:       Simple transfer of 'm' (8 bytes)
ACC:            host ptr 4053c0
ACC:            acc  ptr 0
ACC:            flags: ALLOCATE ACQ_PRESENT REG_PRESENT
ACC:            memory not found in present table
ACC:            allocate (8 bytes)
ACC:              get new reusable memory, added entry
ACC:            new allocated ptr (7f4d67608000)
ACC:            add to present table index 0: host 4053c0 to 4053c8, acc 7f4d67608000
ACC:            new acc ptr 7f4d67608000
ACC:
ACC:   Trans 2
ACC:       Simple transfer of 'n' (8 bytes)
ACC:            host ptr 4053c8
ACC:            acc  ptr 0
ACC:            flags: ALLOCATE COPY_HOST_TO_ACC ACQ_PRESENT REG_PRESENT
ACC:            memory not found in present table
ACC:            allocate (8 bytes)
ACC:              get new reusable memory, added entry
ACC:            new allocated ptr (7f4d67609000)
ACC:            add to present table index 1: host 4053c8 to 4053d0, acc 7f4d67609000
ACC:            copy host to acc (4053c8 to 7f4d67609000)
ACC:                internal copy host to acc (host 4053c8 to acc 7f4d67609000) size = 8
ACC:            new acc ptr 7f4d67609000
ACC:
ACC: End transfer (to acc 8 bytes, to host 0 bytes)
ACC:
ACC: Start kernel target_example_$ck_L6_1 async(auto) from ./teamsdis3.f90:6
ACC:        flags: CACHE_MOD CACHE_FUNC AUTO_ASYNC
ACC:    mod cache:  0x405640
ACC: kernel cache:  0x405440
ACC:   async info:  0x7f4d7b0918d0
ACC:    arguments: GPU argument info
ACC:            param size:  16
ACC:         param pointer:  0x7ffcd9b8ffc0
```

Debugging with Cray compiler: –hlist=aimd

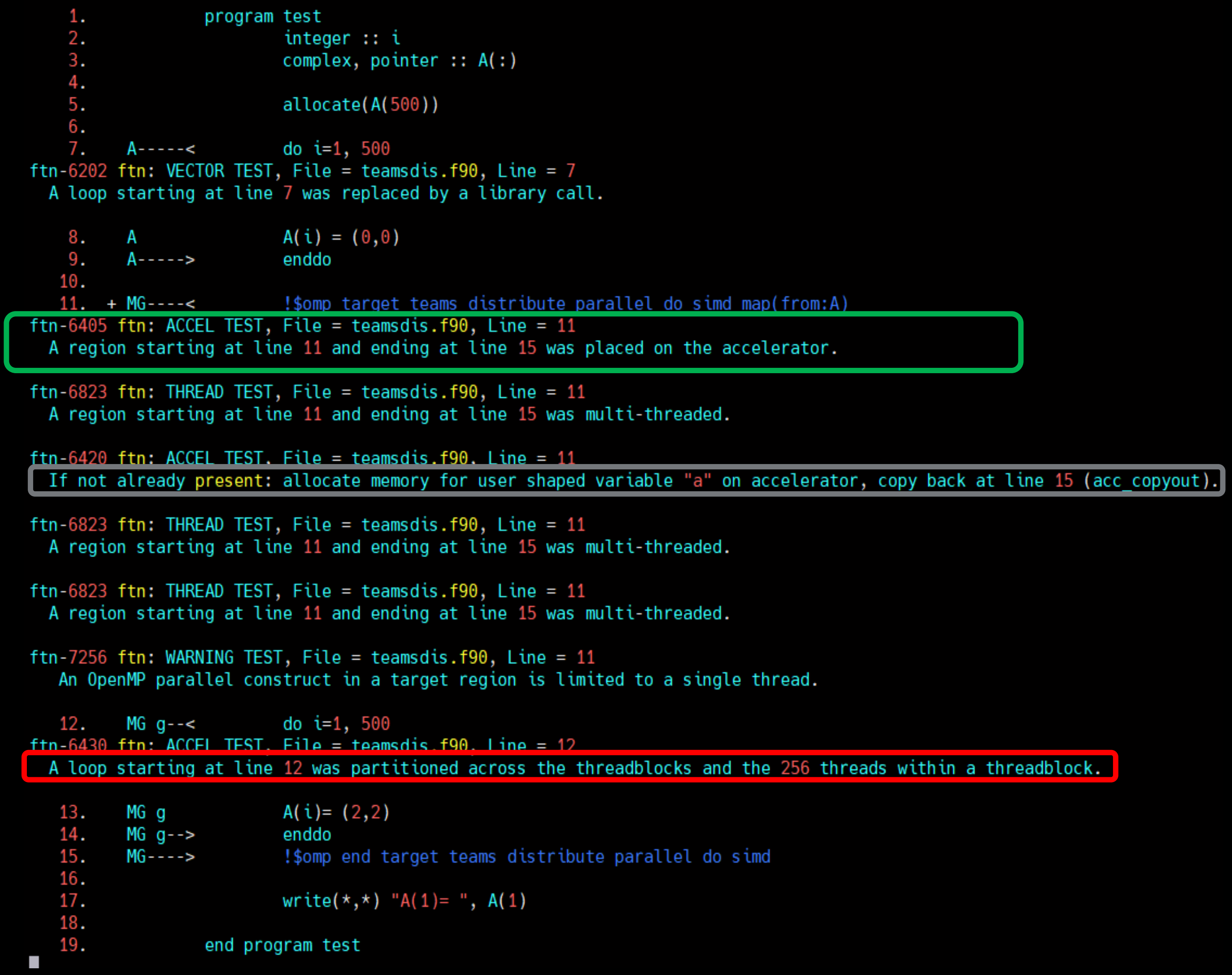
*.lst



teamsdis.f90

```
 1 program test
 2         integer :: i
 3         complex, pointer :: A(:)
 4
 5         allocate(A(500))
 6
 7         do i=1, 500
 8         A(i) = (0,0)
 9         enddo
10
11         !$omp target teams distribute parallel do simd map(from:A)
12         do i=1, 500
13         A(i)= (2,2)
14         enddo
15         !$omp end target teams distribute parallel do simd
16
17         write(*,*) "A(1)= ", A(1)
18
19 end program test
```

$ftn -hnoacc -homp -fopenmp -hlist=aimd -o ./teamsdis ./teamsdis.f90



```
    1.            program test
    2.                    integer :: i
    3.                    complex, pointer :: A(:)
    4.
    5.                    allocate(A(500))
    6.
    7.    A----<          do i=1, 500
ftn-6202 ftn: VECTOR TEST, File = teamsdis.f90, Line = 7
  A loop starting at line 7 was replaced by a library call.

    8.    A               A(i) = (0,0)
    9.    A---->          enddo
   10.
   11.  + MG----<         !$omp target teams distribute parallel do simd map(from:A)
ftn-6405 ftn: ACCEL TEST, File = teamsdis.f90, Line = 11
  A region starting at line 11 and ending at line 15 was placed on the accelerator.

ftn-6823 ftn: THREAD TEST, File = teamsdis.f90, Line = 11
  A region starting at line 11 and ending at line 15 was multi-threaded.

ftn-6420 ftn: ACCEL TEST, File = teamsdis.f90, Line = 11
  If not already present: allocate memory for user shaped variable "a" on accelerator, copy back at line 15 (acc_copyout).

ftn-6823 ftn: THREAD TEST, File = teamsdis.f90, Line = 11
  A region starting at line 11 and ending at line 15 was multi-threaded.

ftn-6823 ftn: THREAD TEST, File = teamsdis.f90, Line = 11
  A region starting at line 11 and ending at line 15 was multi-threaded.

ftn-7256 ftn: WARNING TEST, File = teamsdis.f90, Line = 11
   An OpenMP parallel construct in a target region is limited to a single thread.

   12.    MG g--<         do i=1, 500
ftn-6430 ftn: ACCEL TEST, File = teamsdis.f90, Line = 12
  A loop starting at line 12 was partitioned across the threadblocks and the 256 threads within a threadblock.

   13.    MG g            A(i)= (2,2)
   14.    MG g-->         enddo
   15.    MG---->         !$omp end target teams distribute parallel do simd
   16.
   17.                    write(*,*) "A(1)= ", A(1)
   18.
   19.            end program test
```

AMD
together we advance_

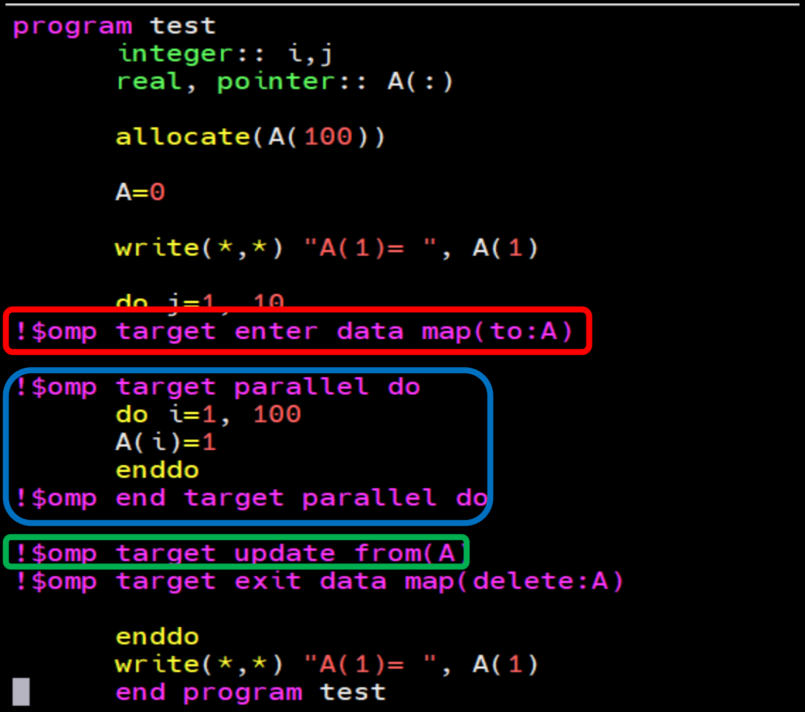
Profiling OpenMP® offloading code on AMD GPUs

- After compiling the code, run it with “rocprof --hip-trace”

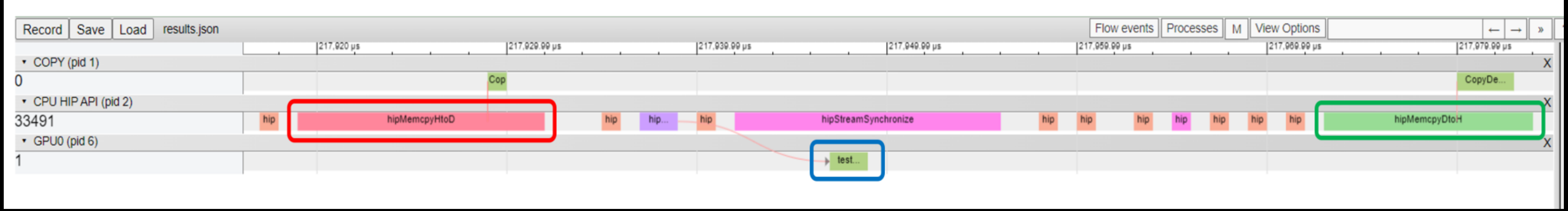
  $ftn -hnoacc -fopenmp -homp -o ./test ./test.f90

  $rocprof --hip-trace ./test

- Open the .json file in chrome://tracing/ or https://ui.perfetto.dev/

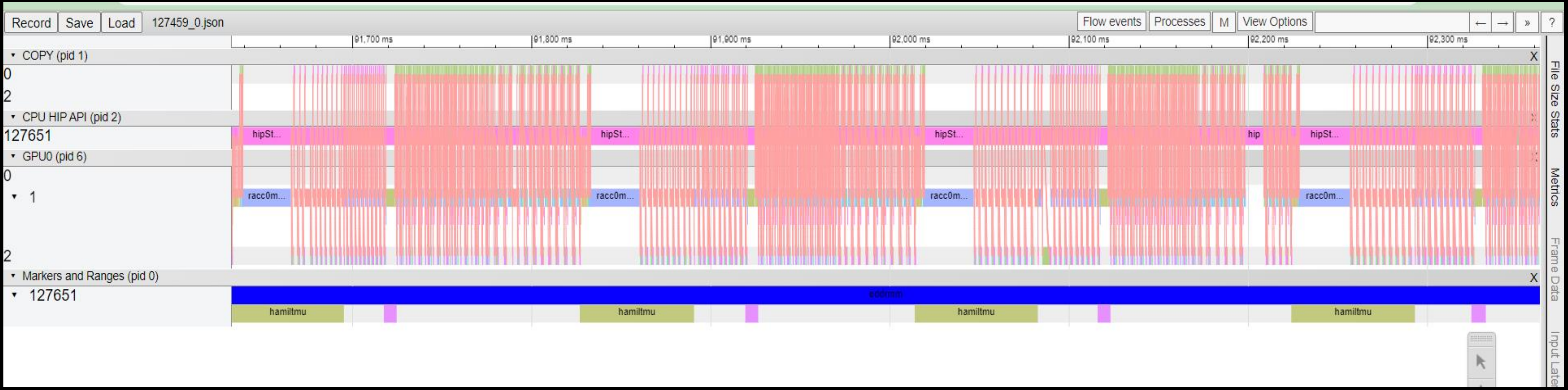


```
program test
      integer:: i,j
      real, pointer:: A(:)

      allocate(A(100))

      A=0

      write(*,*) "A(1)= ", A(1)

      do j=1, 10
!$omp target enter data map(to:A)

!$omp target parallel do
      do i=1, 100
      A(i)=1
      enddo
!$omp end target parallel do

!$omp target update from(A
!$omp target exit data map(delete:A)

      enddo
      write(*,*) "A(1)= ", A(1)
      end program test
```



An example of VASP trace on AMD GPUs

- Use markers to map trace with different sections of the code
  - add roctxRangePushA() and roctxRangePop()
  - Compile with “-lroctx64 –lroctracer64”
  - Run with “rocprof –hip-trace –roctx-trace”

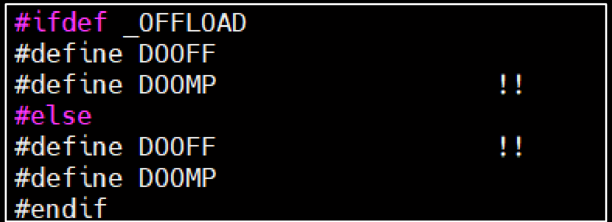


Content

- Introduction
- Debugging and profiling OpenMP offloading code in VASP
- **OpenMP Offloading Challenges in VASP**
  - Concurrent support for different directive-based paradigms
  - Enable/disable offloading in different code paths
  - Interface OMP offloading with ROCM libraries
- Compiler related challenges
  - Pointer aliasing
  - Pointer mismatch in subroutine calls
  - Atomic update
  - Declare target
- Data management
- Summary

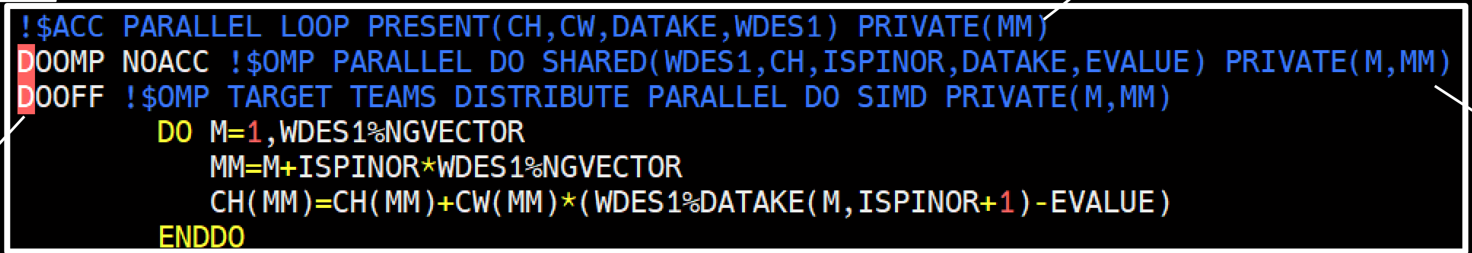
Supporting concurrent directive-based paradigms in VASP

- Switch between different directive-based paradigms without letting them impact on each other
- Take advantage of source preprocessing
  - Pros: switch between different directive-based paradigms
  - Cons: makes the code messy



```
#ifdef _OFFLOAD
#define DOOFF
#define DOOMP                  !!
#else
#define DOOFF                  !!
#define DOOMP
#endif
```

Used when VASP is compiled with OpenACC



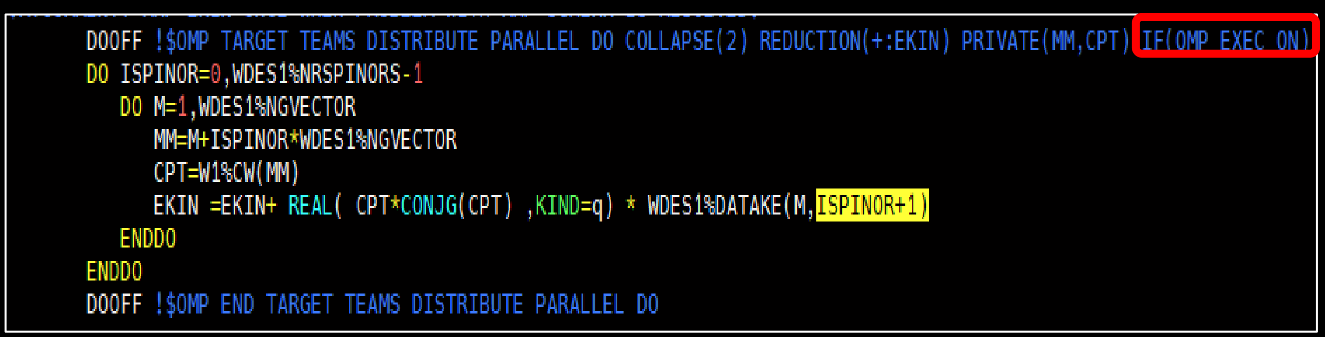
```
!$ACC PARALLEL LOOP PRESENT(CH,CW,DATAKE,WDES1) PRIVATE(MM)
DOOMP NOACC !$OMP PARALLEL DO SHARED(WDES1,CH,ISPINOR,DATAKE,EVALUE) PRIVATE(M,MM)
DOOFF !$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD PRIVATE(M,MM)
        DO M=1,WDES1%NGVECTOR
           MM=M+ISPINOR*WDES1%NGVECTOR
           CH(MM)=CH(MM)+CW(MM)*(WDES1%DATAKE(M,ISPINOR+1)-EVALUE)
        ENDDO
```

Used when OpenMP offloading is enabled

Used when OpenMP (host) is enabled and OpenMP offloading/OpenACC is disabled

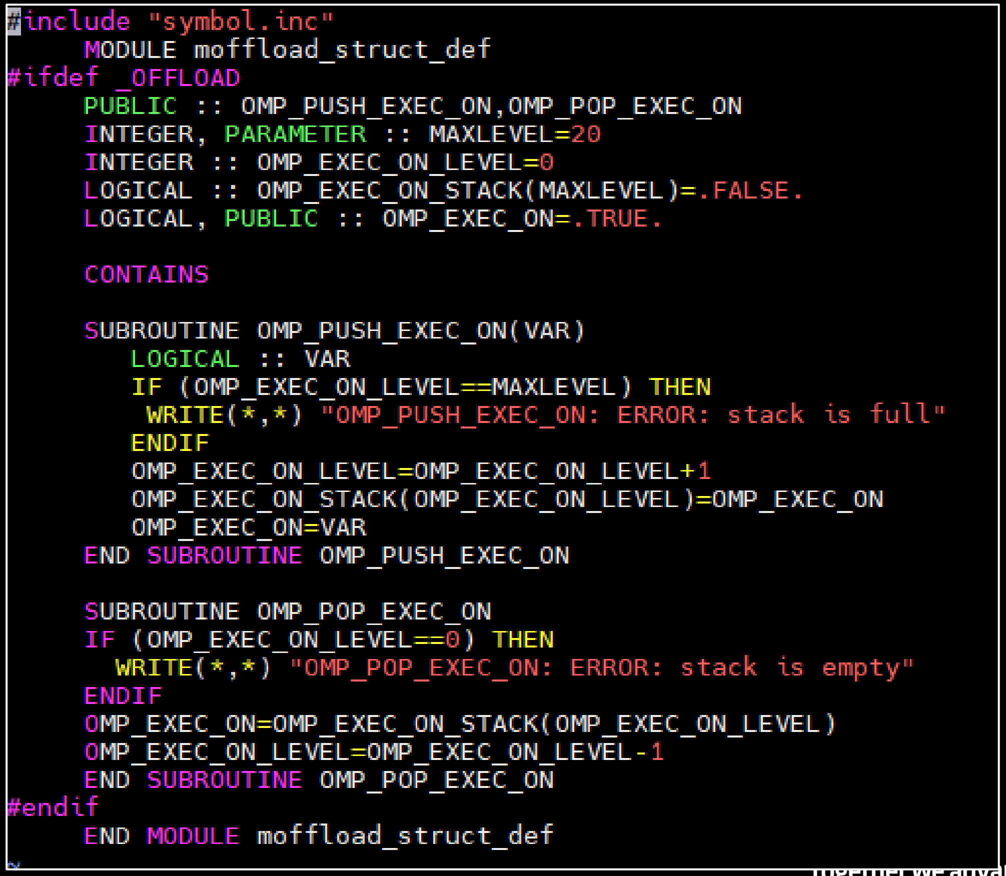
Enable/disable offloading in different code paths

- Many of the VASP subroutines are called from different code paths
  - How can we enable offloading for a subroutine in one path and disable offloading for others
    - It would be useful for code development and debugging



```
      DOOFF !$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO COLLAPSE(2) REDUCTION(+:EKIN) PRIVATE(MM,CPT) IF(OMP_EXEC_ON)
      DO ISPINOR=0,WDES1%NRSPINORS-1
         DO M=1,WDES1%NGVECTOR
            MM=M+ISPINOR*WDES1%NGVECTOR
            CPT=W1%CW(MM)
            EKIN =EKIN+ REAL( CPT*CONJG(CPT) ,KIND=q) * WDES1%DATAKE(M,ISPINOR+1)
         ENDDO
      ENDDO
      DOOFF !$OMP END TARGET TEAMS DISTRIBUTE PARALLEL DO
```

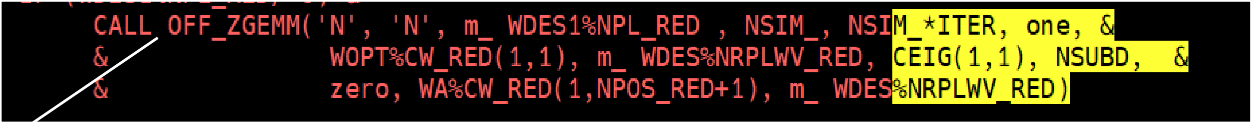
We can call OMP_PUSH_EXEC_ON(.TRUE.) or OMP_PUSH_EXEC_ON(.FALSE.) to enable or disable offloading in different code paths



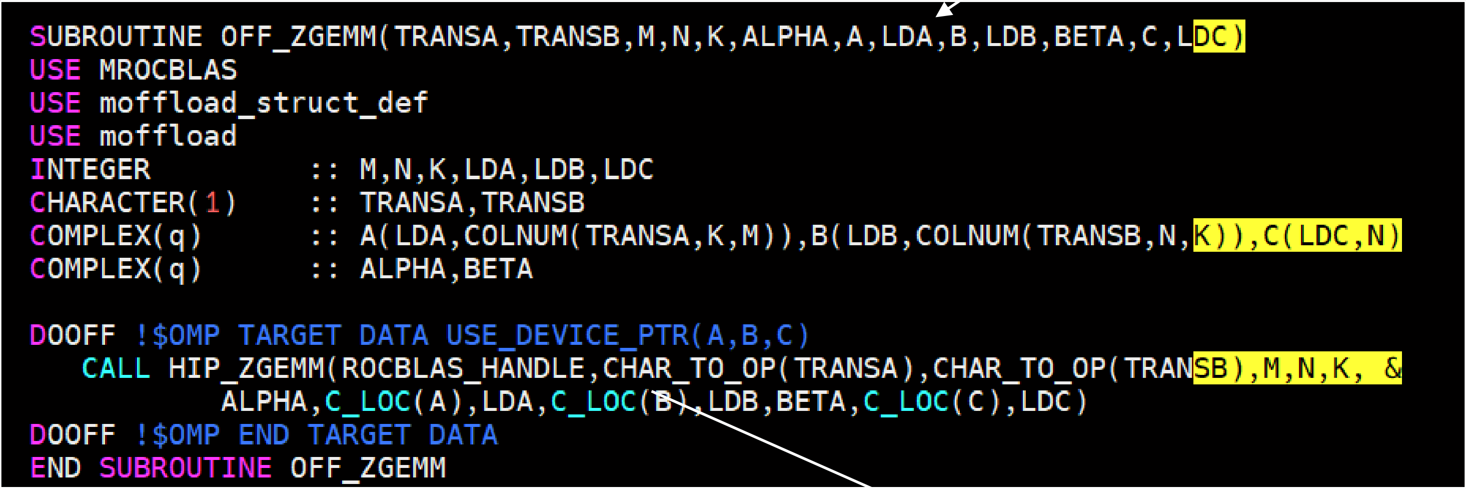
```
#include "symbol.inc"
      MODULE moffload_struct_def
#ifdef _OFFLOAD
      PUBLIC :: OMP_PUSH_EXEC_ON,OMP_POP_EXEC_ON
      INTEGER, PARAMETER :: MAXLEVEL=20
      INTEGER :: OMP_EXEC_ON_LEVEL=0
      LOGICAL :: OMP_EXEC_ON_STACK(MAXLEVEL)=.FALSE.
      LOGICAL, PUBLIC :: OMP_EXEC_ON=.TRUE.

      CONTAINS

      SUBROUTINE OMP_PUSH_EXEC_ON(VAR)
         LOGICAL :: VAR
         IF (OMP_EXEC_ON_LEVEL==MAXLEVEL) THEN
          WRITE(*,*) "OMP_PUSH_EXEC_ON: ERROR: stack is full"
         ENDIF
         OMP_EXEC_ON_LEVEL=OMP_EXEC_ON_LEVEL+1
         OMP_EXEC_ON_STACK(OMP_EXEC_ON_LEVEL)=OMP_EXEC_ON
         OMP_EXEC_ON=VAR
      END SUBROUTINE OMP_PUSH_EXEC_ON

      SUBROUTINE OMP_POP_EXEC_ON
      IF (OMP_EXEC_ON_LEVEL==0) THEN
        WRITE(*,*) "OMP_POP_EXEC_ON: ERROR: stack is empty"
      ENDIF
      OMP_EXEC_ON=OMP_EXEC_ON_STACK(OMP_EXEC_ON_LEVEL)
      OMP_EXEC_ON_LEVEL=OMP_EXEC_ON_LEVEL-1
      END SUBROUTINE OMP_POP_EXEC_ON
#endif
      END MODULE moffload_struct_def
```

Interface OMP offloading with ROCM libraries

- VASP uses FFT, BLAS, and LAPACK extensively
- Developed a wrapper to interface OMP target regions with ROCM libraries
  - rocFFT
  - rocBLAS
  - rocSolver

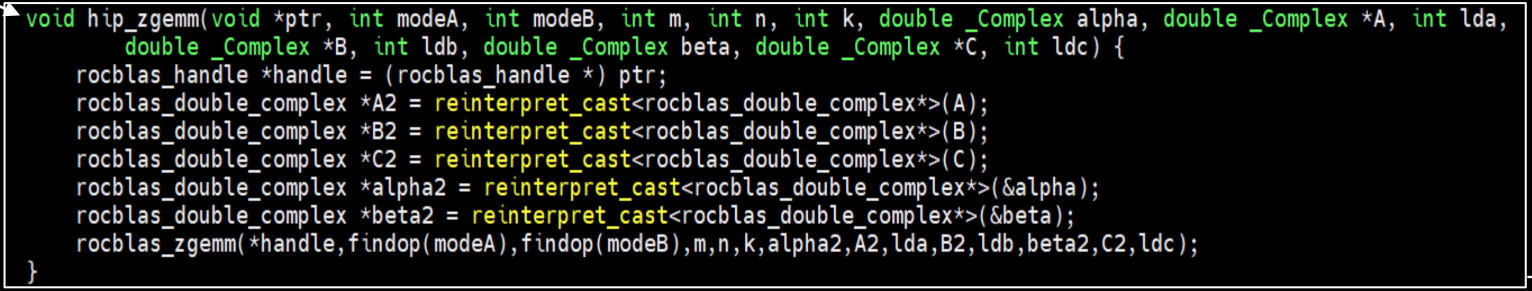


```
      CALL OFF_ZGEMM('N', 'N', m_ WDES1%NPL_RED , NSIM_, NSIM_*ITER, one, &
      &                 WOPT%CW_RED(1,1), m_ WDES%NRPLWV_RED, CEIG(1,1), NSUBD,  &
      &                 zero, WA%CW_RED(1,NPOS_RED+1), m_ WDES%NRPLWV_RED)
```



```
SUBROUTINE OFF_ZGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
USE MROCBLAS
USE moffload_struct_def
USE moffload
INTEGER          :: M,N,K,LDA,LDB,LDC
CHARACTER(1)     :: TRANSA,TRANSB
COMPLEX(q)       :: A(LDA,COLNUM(TRANSA,K,M)),B(LDB,COLNUM(TRANSB,N,K)),C(LDC,N)
COMPLEX(q)       :: ALPHA,BETA

DOOFF !$OMP TARGET DATA USE_DEVICE_PTR(A,B,C)
   CALL HIP_ZGEMM(ROCBLAS_HANDLE,CHAR_TO_OP(TRANSA),CHAR_TO_OP(TRANSB),M,N,K, &
           ALPHA,C_LOC(A),LDA,C_LOC(B),LDB,BETA,C_LOC(C),LDC)
DOOFF !$OMP END TARGET DATA
END SUBROUTINE OFF_ZGEMM
```

WOPT%CW_RED(A), CEIG(B), and WA%CW_RED(C) are mapped to device with “omp target enter data map” directive

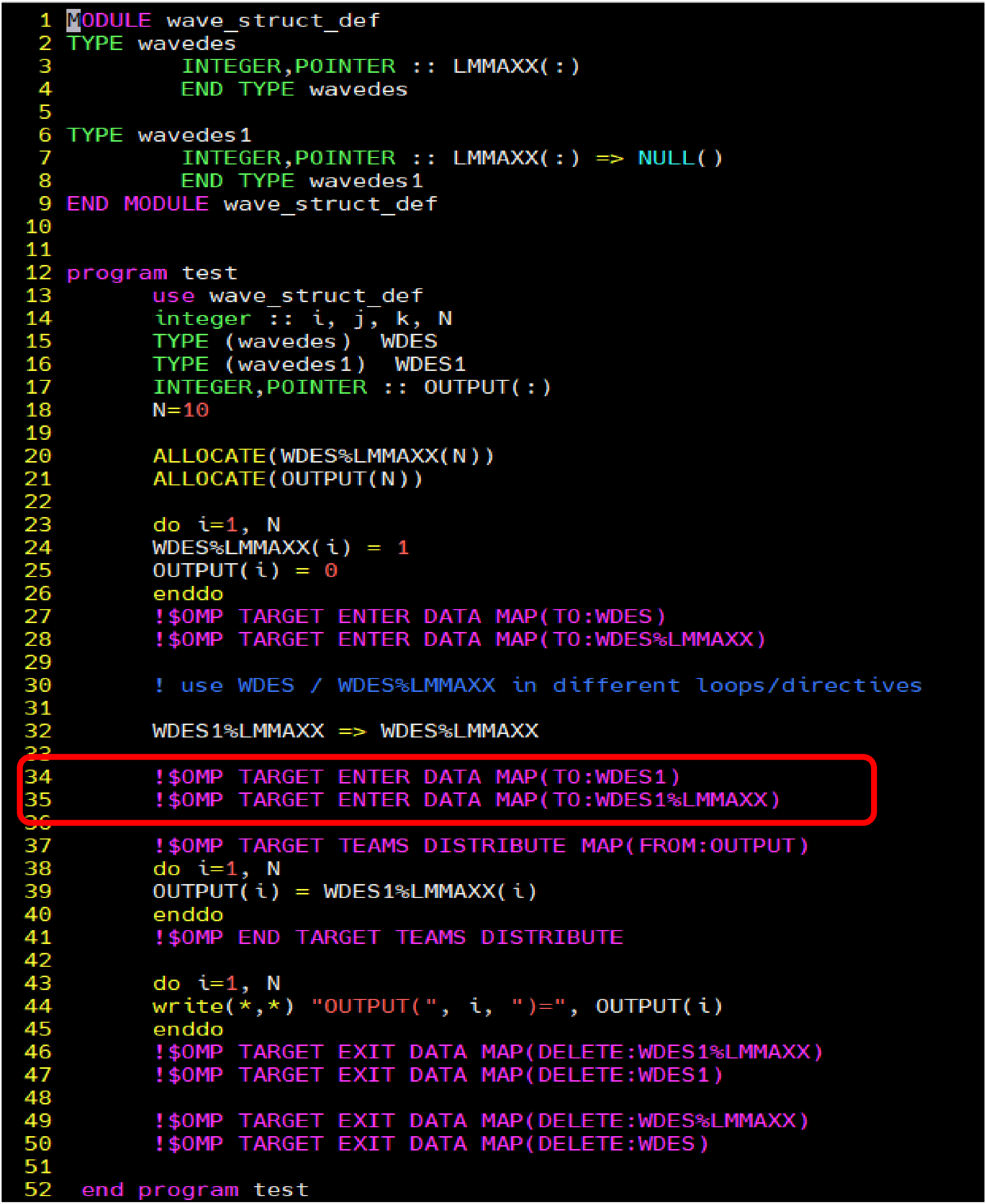


```
void hip_zgemm(void *ptr, int modeA, int modeB, int m, int n, int k, double _Complex alpha, double _Complex *A, int lda,
        double _Complex *B, int ldb, double _Complex beta, double _Complex *C, int ldc) {
    rocblas_handle *handle = (rocblas_handle *) ptr;
    rocblas_double_complex *A2 = reinterpret_cast<rocblas_double_complex*>(A);
    rocblas_double_complex *B2 = reinterpret_cast<rocblas_double_complex*>(B);
    rocblas_double_complex *C2 = reinterpret_cast<rocblas_double_complex*>(C);
    rocblas_double_complex *alpha2 = reinterpret_cast<rocblas_double_complex*>(&alpha);
    rocblas_double_complex *beta2 = reinterpret_cast<rocblas_double_complex*>(&beta);
    rocblas_zgemm(*handle,findop(modeA),findop(modeB),m,n,k,alpha2,A2,lda,B2,ldb,beta2,C2,ldc);
}
```

Content

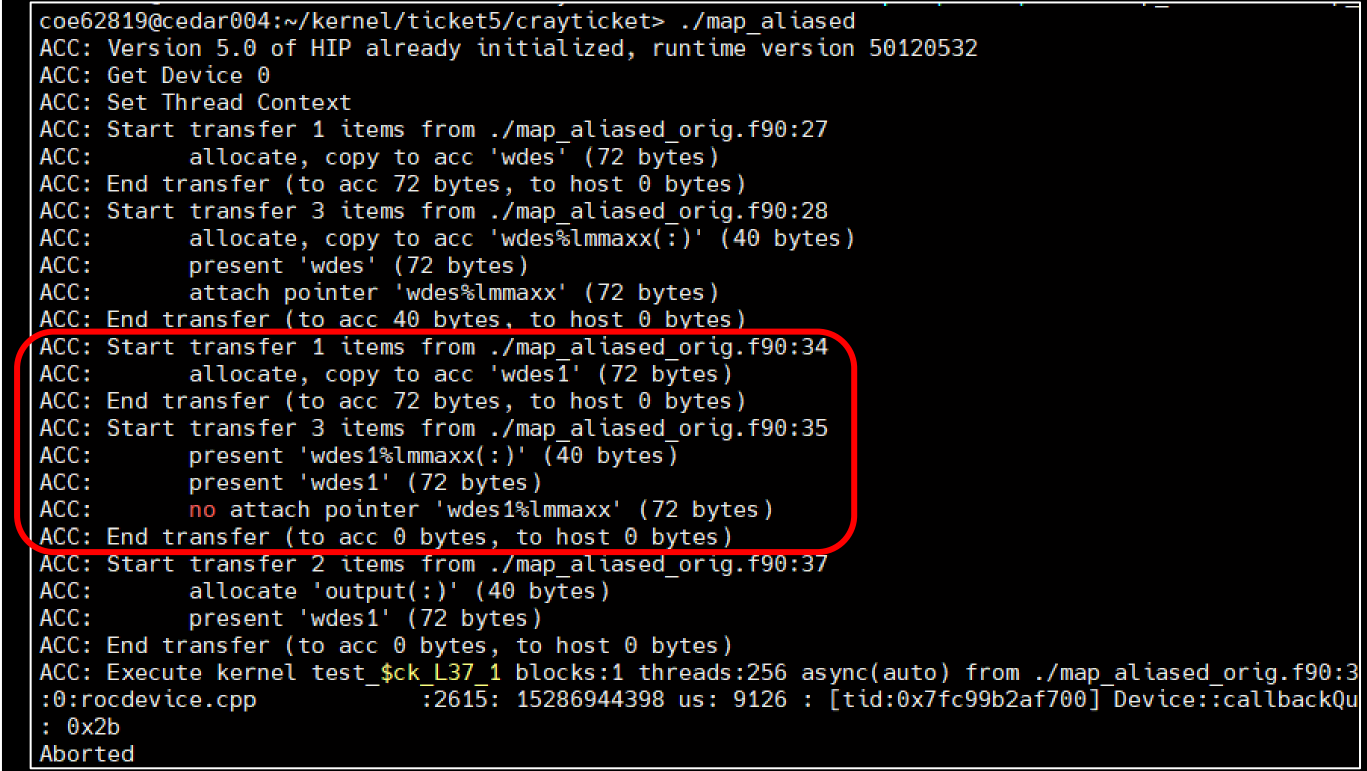
- Introduction
- Debugging and profiling OpenMP offloading code in VASP
- OpenMP Offloading Challenges in VASP
  - Concurrent support for different directive-based paradigms
  - Enable/disable offloading in different code paths
  - Interface OMP offloading with ROCM libraries
- **Compiler related challenges**
  - Pointer aliasing
  - Pointer mismatch in subroutine calls
  - Atomic update
  - Declare target
- Data management
- Summary

Pointer aliasing



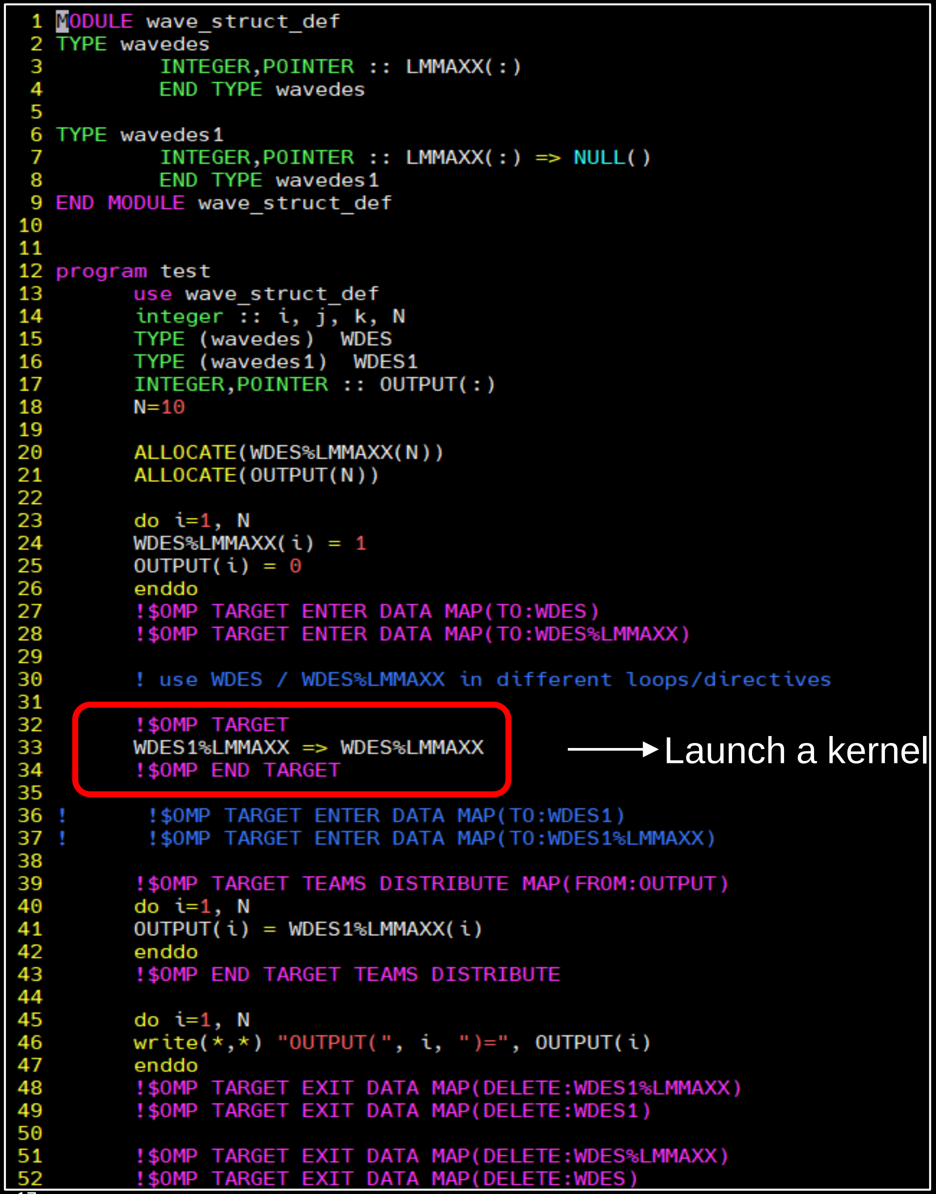
```
 1 MODULE wave_struct_def
 2 TYPE wavedes
 3         INTEGER,POINTER :: LMMAXX(:)
 4         END TYPE wavedes
 5
 6 TYPE wavedes1
 7         INTEGER,POINTER :: LMMAXX(:) => NULL()
 8         END TYPE wavedes1
 9 END MODULE wave_struct_def
10
11
12 program test
13       use wave_struct_def
14       integer :: i, j, k, N
15       TYPE (wavedes)  WDES
16       TYPE (wavedes1)  WDES1
17       INTEGER,POINTER :: OUTPUT(:)
18       N=10
19
20       ALLOCATE(WDES%LMMAXX(N))
21       ALLOCATE(OUTPUT(N))
22
23       do i=1, N
24       WDES%LMMAXX(i) = 1
25       OUTPUT(i) = 0
26       enddo
27       !$OMP TARGET ENTER DATA MAP(TO:WDES)
28       !$OMP TARGET ENTER DATA MAP(TO:WDES%LMMAXX)
29
30       ! use WDES / WDES%LMMAXX in different loops/directives
31
32       WDES1%LMMAXX => WDES%LMMAXX
33
34       !$OMP TARGET ENTER DATA MAP(TO:WDES1)
35       !$OMP TARGET ENTER DATA MAP(TO:WDES1%LMMAXX)
36
37       !$OMP TARGET TEAMS DISTRIBUTE MAP(FROM:OUTPUT)
38       do i=1, N
39       OUTPUT(i) = WDES1%LMMAXX(i)
40       enddo
41       !$OMP END TARGET TEAMS DISTRIBUTE
42
43       do i=1, N
44       write(*,*) "OUTPUT(", i, ")=", OUTPUT(i)
45       enddo
46       !$OMP TARGET EXIT DATA MAP(DELETE:WDES1%LMMAXX)
47       !$OMP TARGET EXIT DATA MAP(DELETE:WDES1)
48
49       !$OMP TARGET EXIT DATA MAP(DELETE:WDES%LMMAXX)
50       !$OMP TARGET EXIT DATA MAP(DELETE:WDES)
51
52  end program test
```

- Pointer aliasing occurs a lot in VASP
  - It can be challenging for the compilers to deal with pointer aliasing on device
- Set CRAY_ACC_DEBUG=3 as environment variable to get the log
- This issue is resolved in CCE15



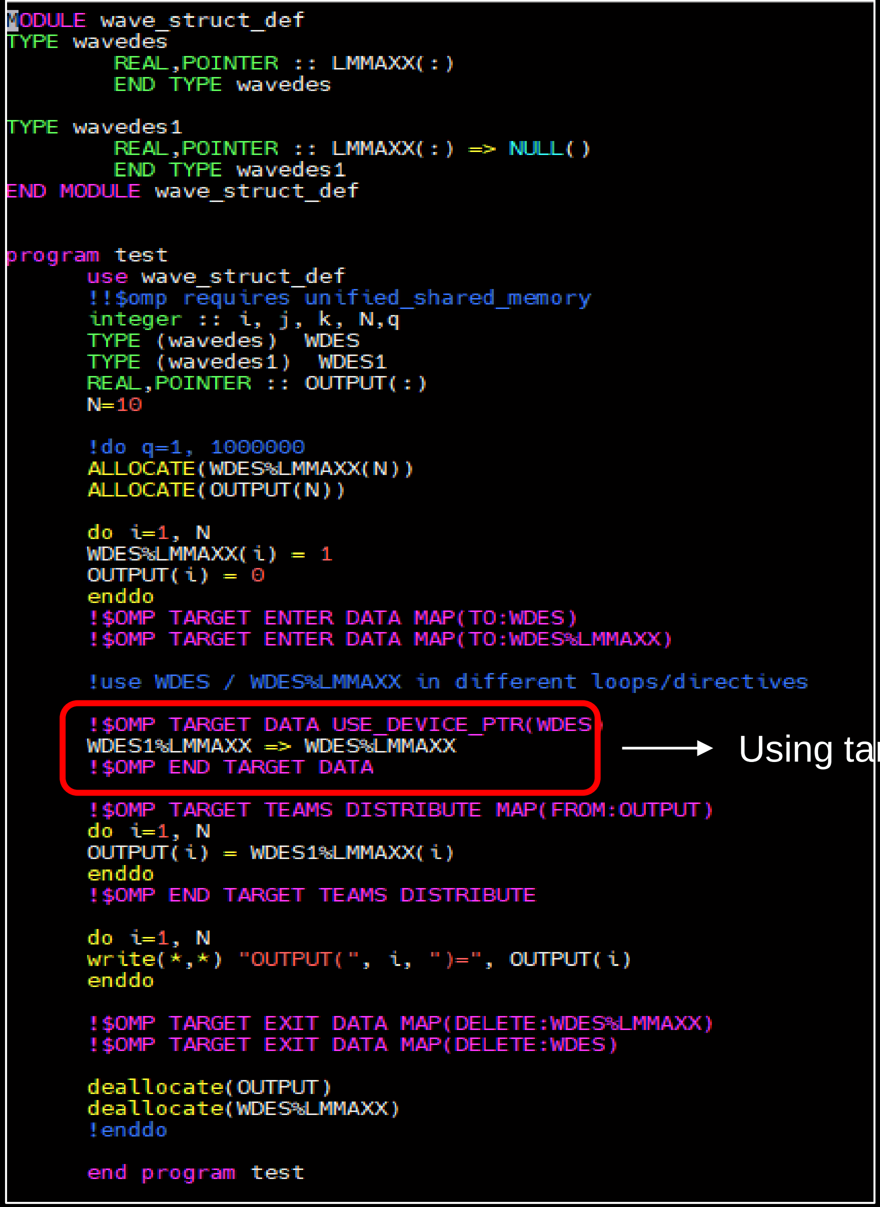
```
coe62819@cedar004:~/kernel/ticket5/crayticket> ./map_aliased
ACC: Version 5.0 of HIP already initialized, runtime version 50120532
ACC: Get Device 0
ACC: Set Thread Context
ACC: Start transfer 1 items from ./map_aliased_orig.f90:27
ACC:        allocate, copy to acc 'wdes' (72 bytes)
ACC: End transfer (to acc 72 bytes, to host 0 bytes)
ACC: Start transfer 3 items from ./map_aliased_orig.f90:28
ACC:        allocate, copy to acc 'wdes%lmmaxx(:)' (40 bytes)
ACC:        present 'wdes' (72 bytes)
ACC:        attach pointer 'wdes%lmmaxx' (72 bytes)
ACC: End transfer (to acc 40 bytes, to host 0 bytes)
ACC: Start transfer 1 items from ./map_aliased_orig.f90:34
ACC:        allocate, copy to acc 'wdes1' (72 bytes)
ACC: End transfer (to acc 72 bytes, to host 0 bytes)
ACC: Start transfer 3 items from ./map_aliased_orig.f90:35
ACC:        present 'wdes1%lmmaxx(:)' (40 bytes)
ACC:        present 'wdes1' (72 bytes)
ACC:        no attach pointer 'wdes1%lmmaxx' (72 bytes)
ACC: End transfer (to acc 0 bytes, to host 0 bytes)
ACC: Start transfer 2 items from ./map_aliased_orig.f90:37
ACC:        allocate 'output(:)' (40 bytes)
ACC:        present 'wdes1' (72 bytes)
ACC: End transfer (to acc 0 bytes, to host 0 bytes)
ACC: Execute kernel test_$ck_L37_1 blocks:1 threads:256 async(auto) from ./map_aliased_orig.f90:3
:0:rocdevice.cpp            :2615: 15286944398 us: 9126 : [tid:0x7fc99b2af700] Device::callbackQu
: 0x2b
Aborted
```

Pointer aliasing (alternative methods)



```
MODULE wave_struct_def
TYPE wavedes
        INTEGER,POINTER :: LMMAXX(:)
        END TYPE wavedes

TYPE wavedes1
        INTEGER,POINTER :: LMMAXX(:) => NULL()
        END TYPE wavedes1
END MODULE wave_struct_def


program test
      use wave_struct_def
      integer :: i, j, k, N
      TYPE (wavedes)  WDES
      TYPE (wavedes1)  WDES1
      INTEGER,POINTER :: OUTPUT(:)
      N=10

      ALLOCATE(WDES%LMMAXX(N))
      ALLOCATE(OUTPUT(N))

      do i=1, N
      WDES%LMMAXX(i) = 1
      OUTPUT(i) = 0
      enddo
      !$OMP TARGET ENTER DATA MAP(TO:WDES)
      !$OMP TARGET ENTER DATA MAP(TO:WDES%LMMAXX)

      ! use WDES / WDES%LMMAXX in different loops/directives

      !$OMP TARGET
      WDES1%LMMAXX => WDES%LMMAXX
      !$OMP END TARGET

!      !$OMP TARGET ENTER DATA MAP(TO:WDES1)
!      !$OMP TARGET ENTER DATA MAP(TO:WDES1%LMMAXX)

      !$OMP TARGET TEAMS DISTRIBUTE MAP(FROM:OUTPUT)
      do i=1, N
      OUTPUT(i) = WDES1%LMMAXX(i)
      enddo
      !$OMP END TARGET TEAMS DISTRIBUTE

      do i=1, N
      write(*,*) "OUTPUT(", i, ")=", OUTPUT(i)
      enddo
      !$OMP TARGET EXIT DATA MAP(DELETE:WDES1%LMMAXX)
      !$OMP TARGET EXIT DATA MAP(DELETE:WDES1)

      !$OMP TARGET EXIT DATA MAP(DELETE:WDES%LMMAXX)
      !$OMP TARGET EXIT DATA MAP(DELETE:WDES)
```

→ Launch a kernel

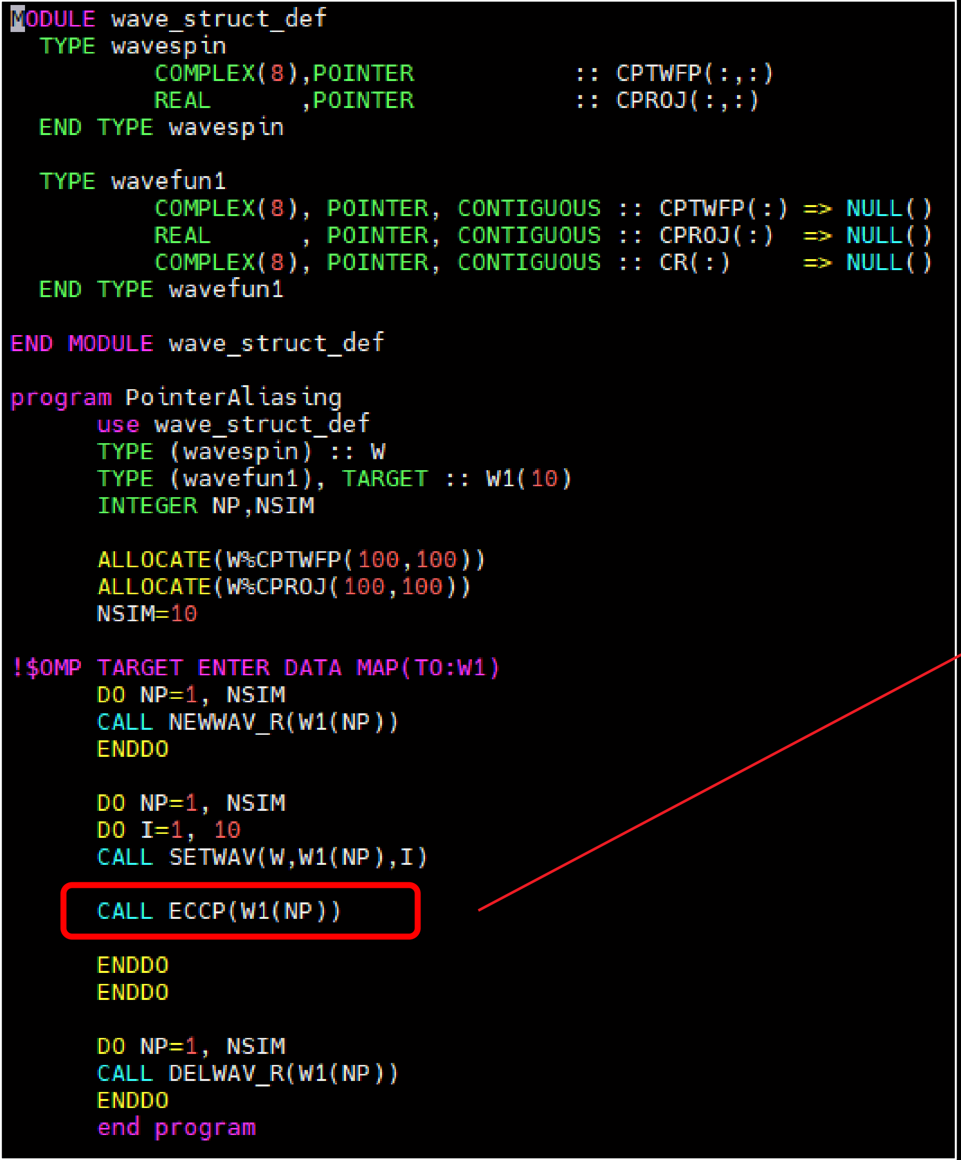


```
MODULE wave_struct_def
TYPE wavedes
        REAL,POINTER :: LMMAXX(:)
        END TYPE wavedes

TYPE wavedes1
        REAL,POINTER :: LMMAXX(:) => NULL()
        END TYPE wavedes1
END MODULE wave_struct_def


program test
      use wave_struct_def
      !!$omp requires unified_shared_memory
      integer :: i, j, k, N,q
      TYPE (wavedes)  WDES
      TYPE (wavedes1)  WDES1
      REAL,POINTER :: OUTPUT(:)
      N=10

      !do q=1, 1000000
      ALLOCATE(WDES%LMMAXX(N))
      ALLOCATE(OUTPUT(N))

      do i=1, N
      WDES%LMMAXX(i) = 1
      OUTPUT(i) = 0
      enddo
      !$OMP TARGET ENTER DATA MAP(TO:WDES)
      !$OMP TARGET ENTER DATA MAP(TO:WDES%LMMAXX)

      !use WDES / WDES%LMMAXX in different loops/directives

      !$OMP TARGET DATA USE_DEVICE_PTR(WDES)
      WDES1%LMMAXX => WDES%LMMAXX
      !$OMP END TARGET DATA

      !$OMP TARGET TEAMS DISTRIBUTE MAP(FROM:OUTPUT)
      do i=1, N
      OUTPUT(i) = WDES1%LMMAXX(i)
      enddo
      !$OMP END TARGET TEAMS DISTRIBUTE

      do i=1, N
      write(*,*) "OUTPUT(", i, ")=", OUTPUT(i)
      enddo

      !$OMP TARGET EXIT DATA MAP(DELETE:WDES%LMMAXX)
      !$OMP TARGET EXIT DATA MAP(DELETE:WDES)

      deallocate(OUTPUT)
      deallocate(WDES%LMMAXX)
      !enddo

      end program test
```

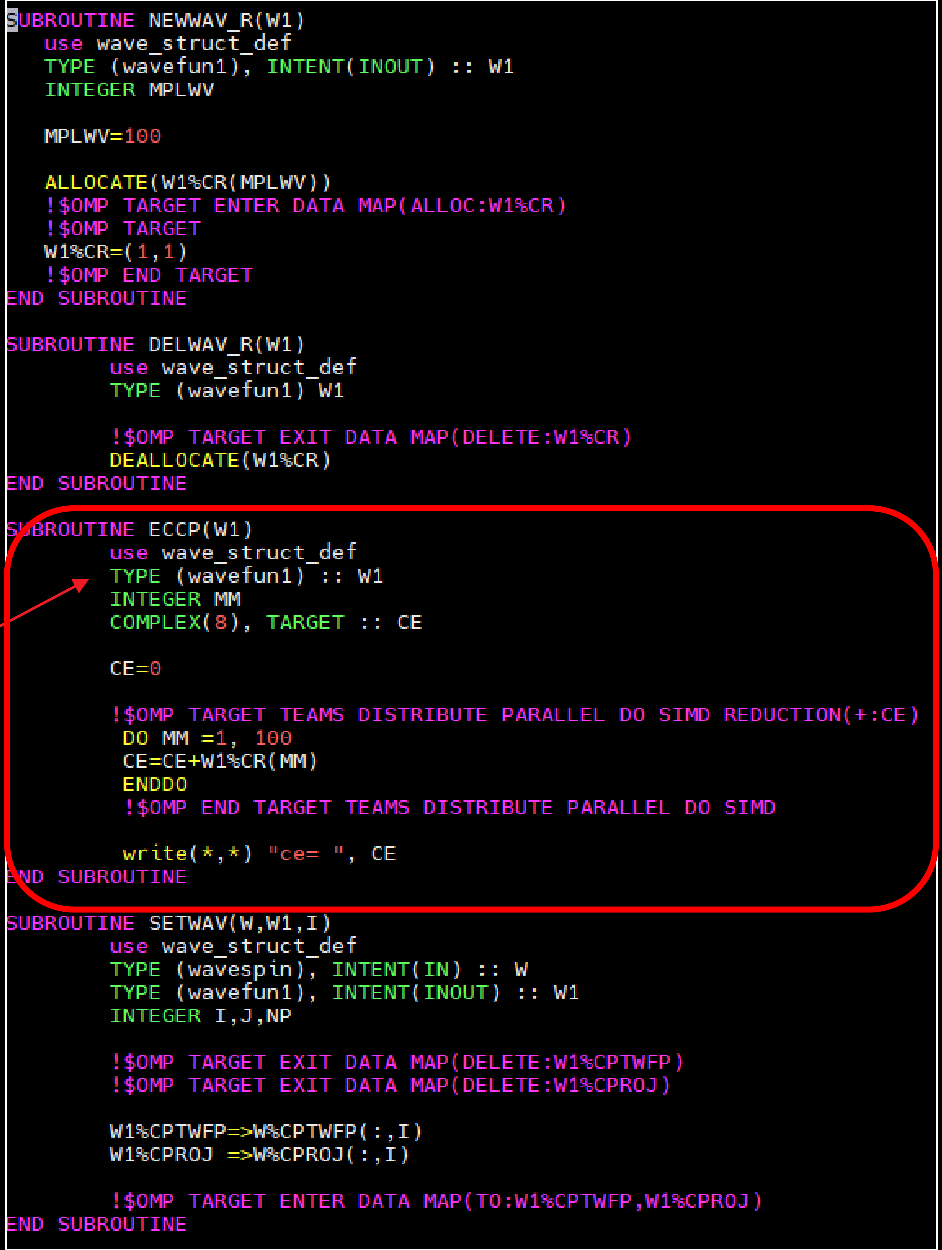
→ Using target data construct

AMD
together we advance_

Pointer mismatch in subroutine calls



```fortran
MODULE wave_struct_def
  TYPE wavespin
          COMPLEX(8),POINTER             :: CPTWFP(:,:)
          REAL      ,POINTER             :: CPROJ(:,:)
  END TYPE wavespin

  TYPE wavefun1
          COMPLEX(8), POINTER, CONTIGUOUS :: CPTWFP(:) => NULL()
          REAL      , POINTER, CONTIGUOUS :: CPROJ(:)  => NULL()
          COMPLEX(8), POINTER, CONTIGUOUS :: CR(:)     => NULL()
  END TYPE wavefun1

END MODULE wave_struct_def

program PointerAliasing
      use wave_struct_def
      TYPE (wavespin) :: W
      TYPE (wavefun1), TARGET :: W1(10)
      INTEGER NP,NSIM

      ALLOCATE(W%CPTWFP(100,100))
      ALLOCATE(W%CPROJ(100,100))
      NSIM=10

!$OMP TARGET ENTER DATA MAP(TO:W1)
      DO NP=1, NSIM
      CALL NEWWAV_R(W1(NP))
      ENDDO

      DO NP=1, NSIM
      DO I=1, 10
      CALL SETWAV(W,W1(NP),I)

      CALL ECCP(W1(NP))

      ENDDO
      ENDDO

      DO NP=1, NSIM
      CALL DELWAV_R(W1(NP))
      ENDDO
      end program
```

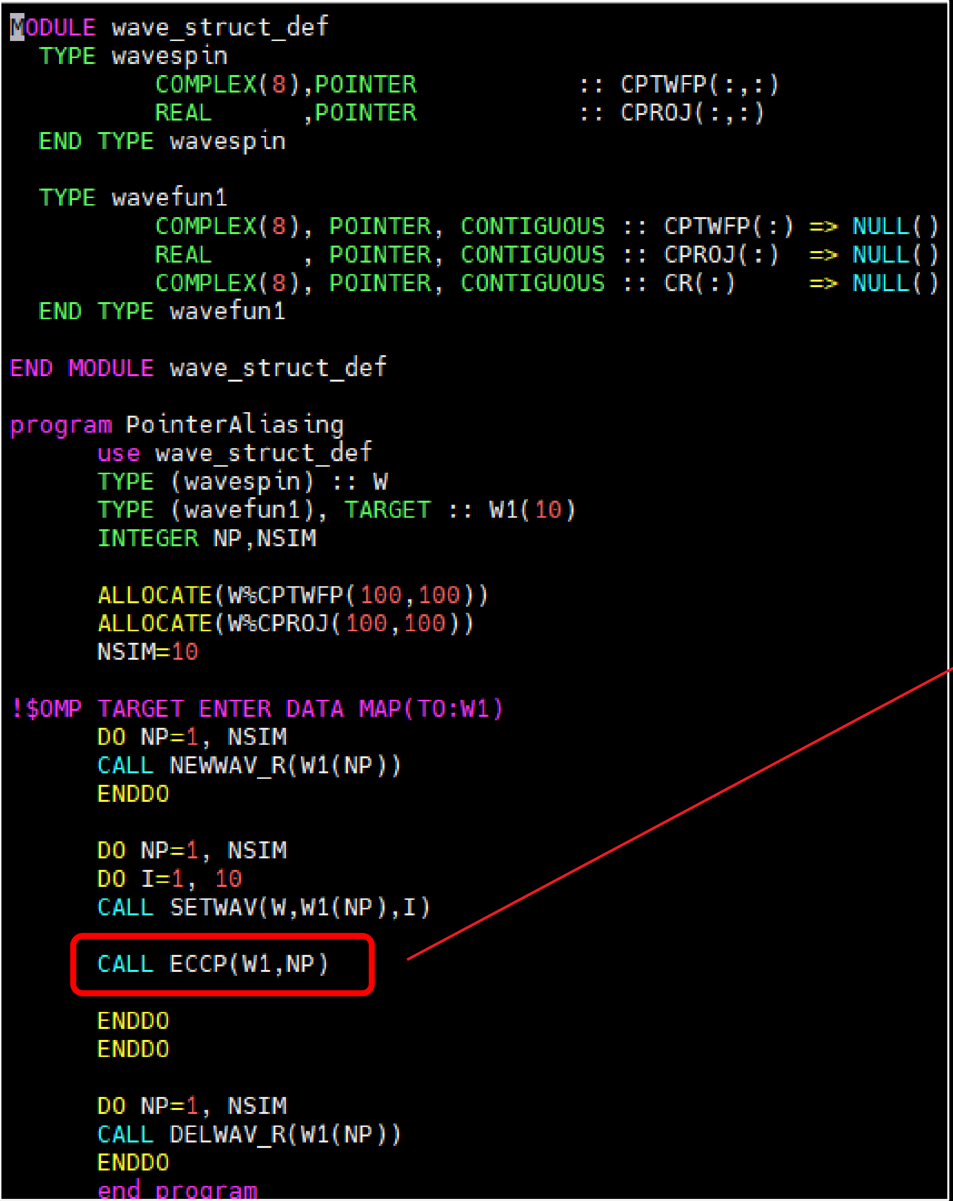


```fortran
SUBROUTINE NEWWAV_R(W1)
   use wave_struct_def
   TYPE (wavefun1), INTENT(INOUT) :: W1
   INTEGER MPLWV

   MPLWV=100

   ALLOCATE(W1%CR(MPLWV))
   !$OMP TARGET ENTER DATA MAP(ALLOC:W1%CR)
   !$OMP TARGET
   W1%CR=(1,1)
   !$OMP END TARGET
END SUBROUTINE

SUBROUTINE DELWAV_R(W1)
        use wave_struct_def
        TYPE (wavefun1) W1

        !$OMP TARGET EXIT DATA MAP(DELETE:W1%CR)
        DEALLOCATE(W1%CR)
END SUBROUTINE

SUBROUTINE ECCP(W1)
        use wave_struct_def
        TYPE (wavefun1) :: W1
        INTEGER MM
        COMPLEX(8), TARGET :: CE

        CE=0

        !$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD REDUCTION(+:CE)
         DO MM =1, 100
         CE=CE+W1%CR(MM)
         ENDDO
         !$OMP END TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD

         write(*,*) "ce= ", CE
END SUBROUTINE
SUBROUTINE SETWAV(W,W1,I)
        use wave_struct_def
        TYPE (wavespin), INTENT(IN) :: W
        TYPE (wavefun1), INTENT(INOUT) :: W1
        INTEGER I,J,NP

        !$OMP TARGET EXIT DATA MAP(DELETE:W1%CPTWFP)
        !$OMP TARGET EXIT DATA MAP(DELETE:W1%CPROJ)

        W1%CPTWFP=>W%CPTWFP(:,I)
        W1%CPROJ =>W%CPROJ(:,I)

        !$OMP TARGET ENTER DATA MAP(TO:W1%CPTWFP,W1%CPROJ)
END SUBROUTINE
```

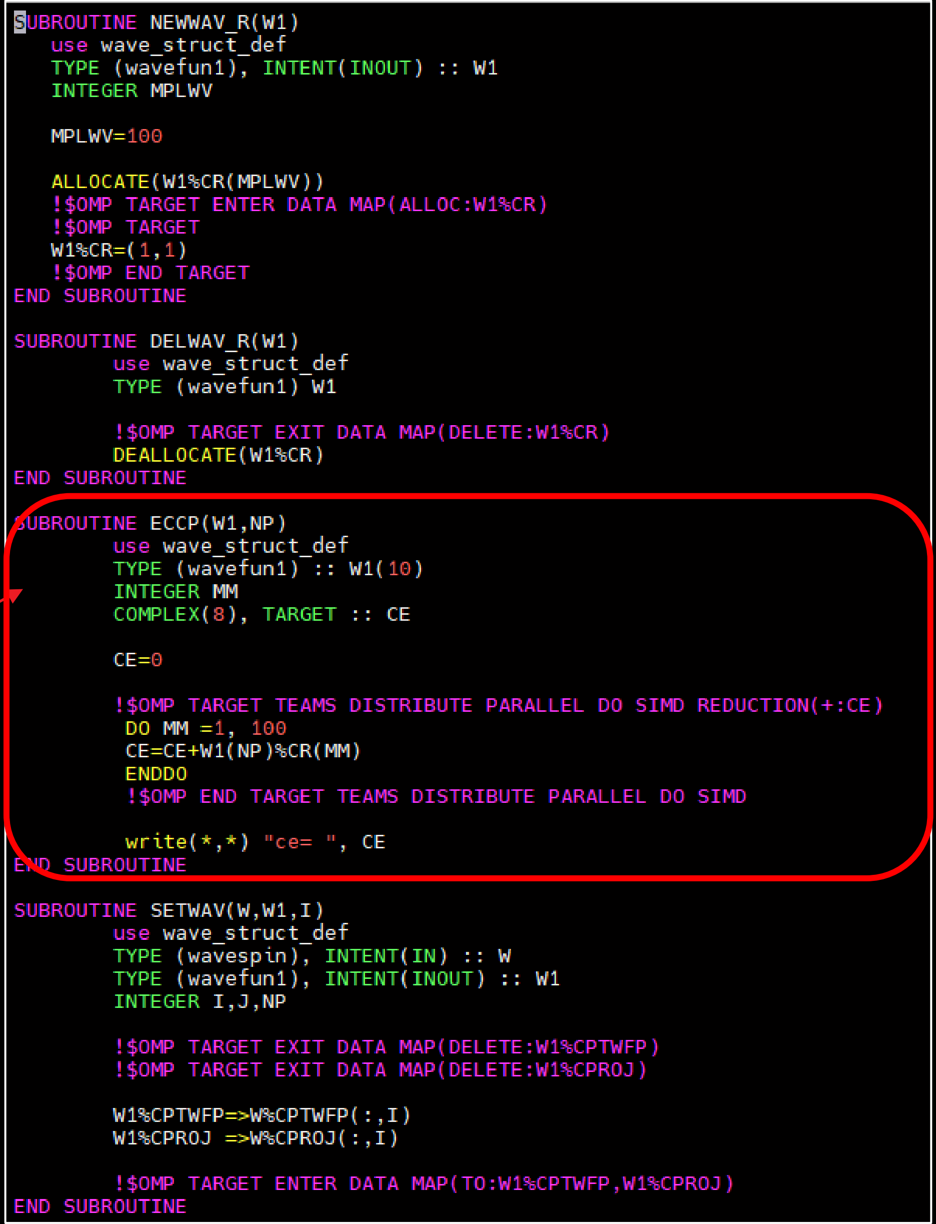
```
$./aliasing
 ce=  (100.,100.)
 ce=  (100.,100.)
 ce=  (100.,100.)
 ce=  (100.,100.)
 ce=  (100.,100.)
 ce=  (100.,100.)
 ce=  (100.,100.)
 ce=  (100.,100.)
 ce=  (100.,100.)
 ce=  (100.,100.)
:0:rocdevice.cpp           :2660: 1637590862517 us:
86531: [tid:0x7fbe82217700]
Device::callbackQueue aborting with error :
HSA_STATUS_ERROR_MEMORY_APERTURE_V
IOLATION: The agent attempted to access memory
beyond the largest legal address. code: 0x29
Aborted
```

AMD
together we advance_

Pointer mismatch in subroutine calls (alternative method)

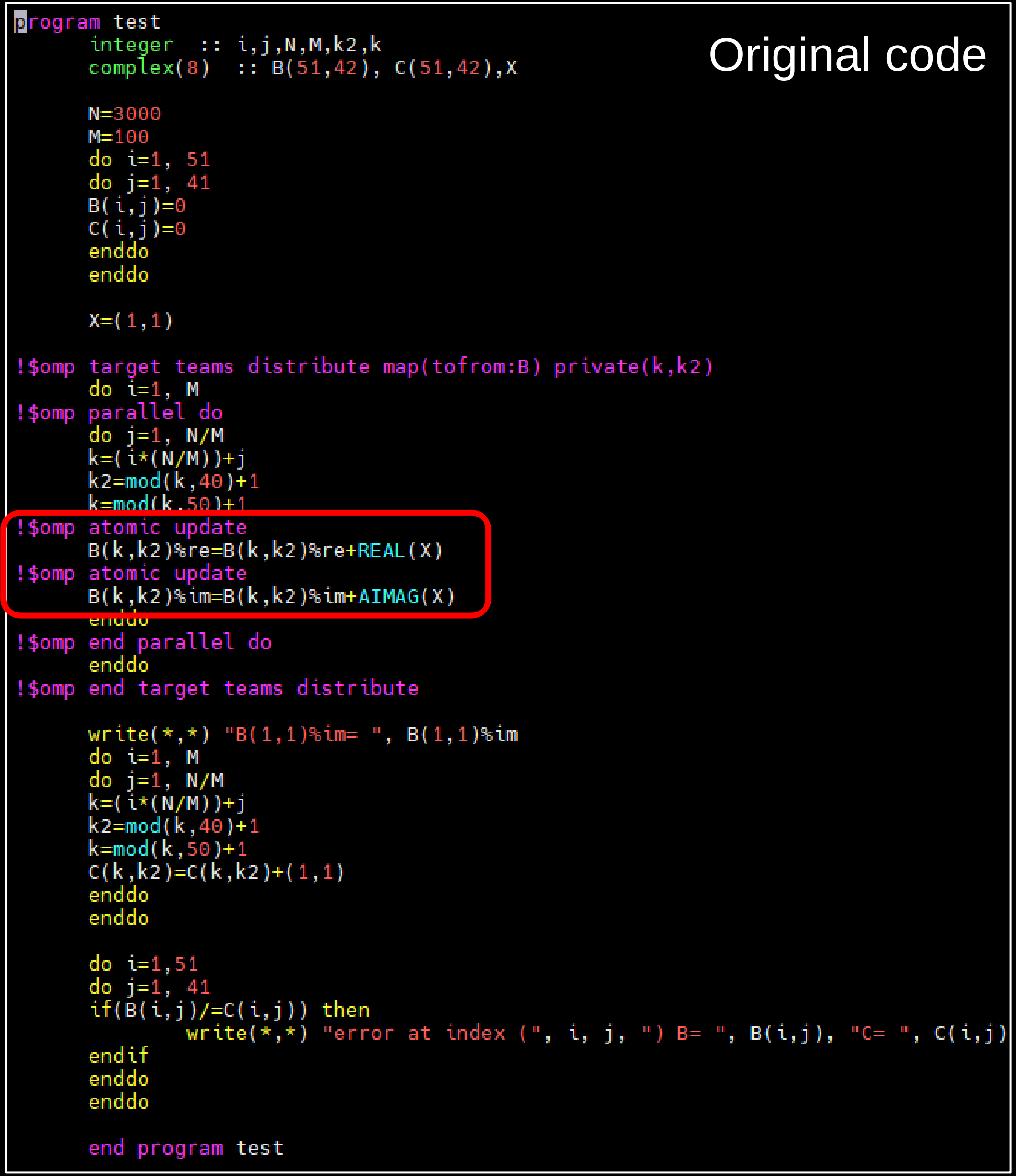


```
MODULE wave_struct_def
  TYPE wavespin
          COMPLEX(8),POINTER           :: CPTWFP(:,:)
          REAL      ,POINTER           :: CPROJ(:,:)
  END TYPE wavespin

  TYPE wavefun1
          COMPLEX(8), POINTER, CONTIGUOUS :: CPTWFP(:) => NULL()
          REAL      , POINTER, CONTIGUOUS :: CPROJ(:)  => NULL()
          COMPLEX(8), POINTER, CONTIGUOUS :: CR(:)     => NULL()
  END TYPE wavefun1

END MODULE wave_struct_def

program PointerAliasing
      use wave_struct_def
      TYPE (wavespin) :: W
      TYPE (wavefun1), TARGET :: W1(10)
      INTEGER NP,NSIM

      ALLOCATE(W%CPTWFP(100,100))
      ALLOCATE(W%CPROJ(100,100))
      NSIM=10

!$OMP TARGET ENTER DATA MAP(TO:W1)
      DO NP=1, NSIM
      CALL NEWWAV_R(W1(NP))
      ENDDO

      DO NP=1, NSIM
      DO I=1, 10
      CALL SETWAV(W,W1(NP),I)

      CALL ECCP(W1,NP)

      ENDDO
      ENDDO

      DO NP=1, NSIM
      CALL DELWAV_R(W1(NP))
      ENDDO
      end program
```



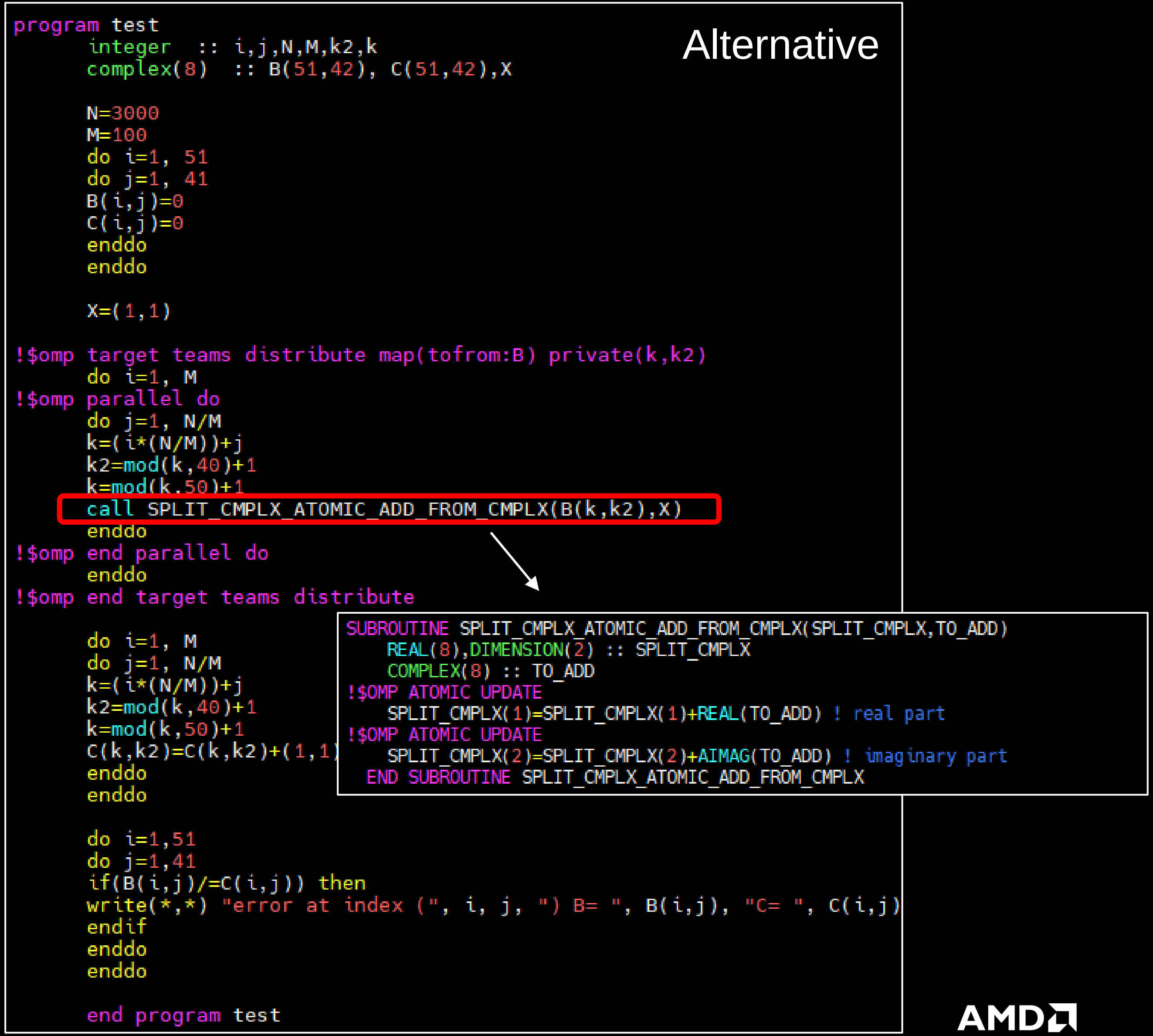
```
SUBROUTINE NEWWAV_R(W1)
   use wave_struct_def
   TYPE (wavefun1), INTENT(INOUT) :: W1
   INTEGER MPLWV

   MPLWV=100

   ALLOCATE(W1%CR(MPLWV))
   !$OMP TARGET ENTER DATA MAP(ALLOC:W1%CR)
   !$OMP TARGET
   W1%CR=(1,1)
   !$OMP END TARGET
END SUBROUTINE

SUBROUTINE DELWAV_R(W1)
        use wave_struct_def
        TYPE (wavefun1) W1

        !$OMP TARGET EXIT DATA MAP(DELETE:W1%CR)
        DEALLOCATE(W1%CR)
END SUBROUTINE

SUBROUTINE ECCP(W1,NP)
        use wave_struct_def
        TYPE (wavefun1) :: W1(10)
        INTEGER MM
        COMPLEX(8), TARGET :: CE

        CE=0

        !$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD REDUCTION(+:CE)
         DO MM =1, 100
         CE=CE+W1(NP)%CR(MM)
         ENDDO
         !$OMP END TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD

         write(*,*) "ce= ", CE
END SUBROUTINE

SUBROUTINE SETWAV(W,W1,I)
        use wave_struct_def
        TYPE (wavespin), INTENT(IN) :: W
        TYPE (wavefun1), INTENT(INOUT) :: W1
        INTEGER I,J,NP

        !$OMP TARGET EXIT DATA MAP(DELETE:W1%CPTWFP)
        !$OMP TARGET EXIT DATA MAP(DELETE:W1%CPROJ)

        W1%CPTWFP=>W%CPTWFP(:,I)
        W1%CPROJ =>W%CPROJ(:,I)

        !$OMP TARGET ENTER DATA MAP(TO:W1%CPTWFP,W1%CPROJ)
END SUBROUTINE
```

Atomic update for complex(8)



Original code

```
program test
      integer  :: i,j,N,M,k2,k
      complex(8)  :: B(51,42), C(51,42),X

      N=3000
      M=100
      do i=1, 51
      do j=1, 41
      B(i,j)=0
      C(i,j)=0
      enddo
      enddo

      X=(1,1)

!$omp target teams distribute map(tofrom:B) private(k,k2)
      do i=1, M
!$omp parallel do
      do j=1, N/M
      k=(i*(N/M))+j
      k2=mod(k,40)+1
      k=mod(k,50)+1
!$omp atomic update
      B(k,k2)%re=B(k,k2)%re+REAL(X)
!$omp atomic update
      B(k,k2)%im=B(k,k2)%im+AIMAG(X)
      enddo
!$omp end parallel do
      enddo
!$omp end target teams distribute

      write(*,*) "B(1,1)%im= ", B(1,1)%im
      do i=1, M
      do j=1, N/M
      k=(i*(N/M))+j
      k2=mod(k,40)+1
      k=mod(k,50)+1
      C(k,k2)=C(k,k2)+(1,1)
      enddo
      enddo

      do i=1,51
      do j=1, 41
      if(B(i,j)/=C(i,j)) then
             write(*,*) "error at index (", i, j, ") B= ", B(i,j), "C= ", C(i,j)
      endif
      enddo
      enddo

      end program test
```

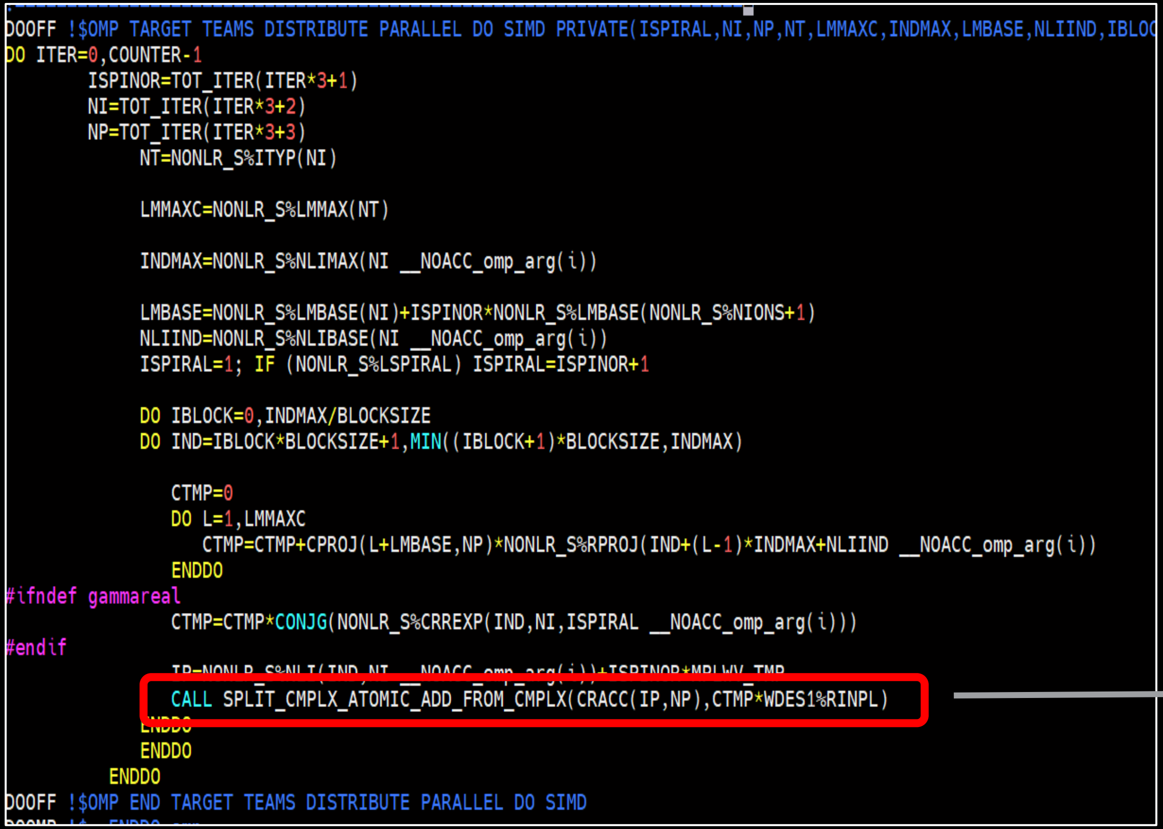


Alternative

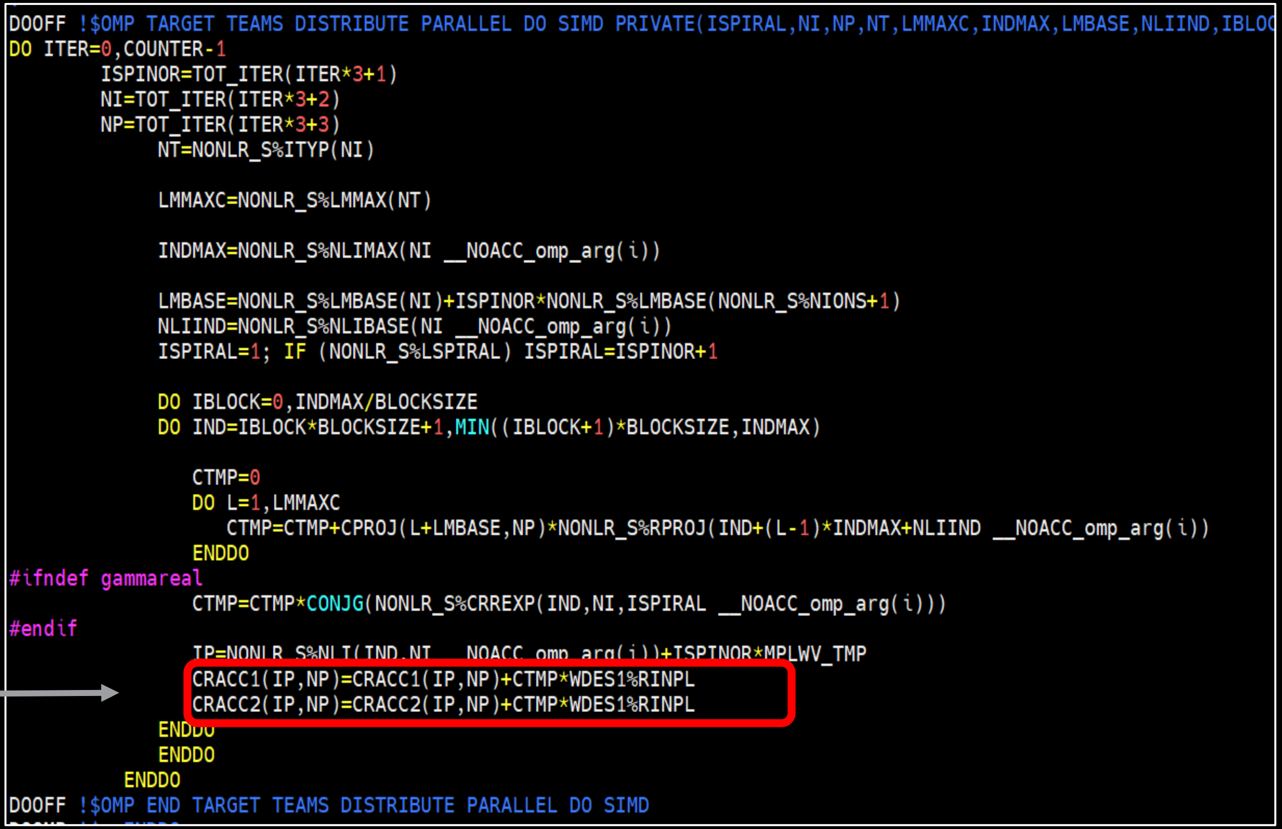
```
program test
      integer  :: i,j,N,M,k2,k
      complex(8)  :: B(51,42), C(51,42),X

      N=3000
      M=100
      do i=1, 51
      do j=1, 41
      B(i,j)=0
      C(i,j)=0
      enddo
      enddo

      X=(1,1)

!$omp target teams distribute map(tofrom:B) private(k,k2)
      do i=1, M
!$omp parallel do
      do j=1, N/M
      k=(i*(N/M))+j
      k2=mod(k,40)+1
      k=mod(k,50)+1
      call SPLIT_CMPLX_ATOMIC_ADD_FROM_CMPLX(B(k,k2),X)
      enddo
!$omp end parallel do
      enddo
!$omp end target teams distribute

      do i=1, M
      do j=1, N/M
      k=(i*(N/M))+j
      k2=mod(k,40)+1
      k=mod(k,50)+1
      C(k,k2)=C(k,k2)+(1,1)
      enddo
      enddo

      do i=1,51
      do j=1,41
      if(B(i,j)/=C(i,j)) then
      write(*,*) "error at index (", i, j, ") B= ", B(i,j), "C= ", C(i,j)
      endif
      enddo
      enddo

      end program test
```

```
SUBROUTINE SPLIT_CMPLX_ATOMIC_ADD_FROM_CMPLX(SPLIT_CMPLX,TO_ADD)
    REAL(8),DIMENSION(2) :: SPLIT_CMPLX
    COMPLEX(8) :: TO_ADD
!$OMP ATOMIC UPDATE
    SPLIT_CMPLX(1)=SPLIT_CMPLX(1)+REAL(TO_ADD) ! real part
!$OMP ATOMIC UPDATE
    SPLIT_CMPLX(2)=SPLIT_CMPLX(2)+AIMAG(TO_ADD) ! imaginary part
  END SUBROUTINE SPLIT_CMPLX_ATOMIC_ADD_FROM_CMPLX
```

The overhead of subroutine call assuming there is no need for atomic update



```
DOOFF !$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD PRIVATE(ISPIRAL,NI,NP,NT,LMMAXC,INDMAX,LMBASE,NLIIND,IBLOC
DO ITER=0,COUNTER-1
        ISPINOR=TOT_ITER(ITER*3+1)
        NI=TOT_ITER(ITER*3+2)
        NP=TOT_ITER(ITER*3+3)
             NT=NONLR_S%ITYP(NI)

             LMMAXC=NONLR_S%LMMAX(NT)

             INDMAX=NONLR_S%NLIMAX(NI __NOACC_omp_arg(i))

             LMBASE=NONLR_S%LMBASE(NI)+ISPINOR*NONLR_S%LMBASE(NONLR_S%NIONS+1)
             NLIIND=NONLR_S%NLIBASE(NI __NOACC_omp_arg(i))
             ISPIRAL=1; IF (NONLR_S%LSPIRAL) ISPIRAL=ISPINOR+1

             DO IBLOCK=0,INDMAX/BLOCKSIZE
             DO IND=IBLOCK*BLOCKSIZE+1,MIN((IBLOCK+1)*BLOCKSIZE,INDMAX)

                CTMP=0
                DO L=1,LMMAXC
                   CTMP=CTMP+CPROJ(L+LMBASE,NP)*NONLR_S%RPROJ(IND+(L-1)*INDMAX+NLIIND __NOACC_omp_arg(i))
                ENDDO
#ifndef gammareal
                CTMP=CTMP*CONJG(NONLR_S%CRREXP(IND,NI,ISPIRAL __NOACC_omp_arg(i)))

#endif
                IP=NONLR_S%NLI(IND,NI __NOACC_omp_arg(i))+ISPINOR*MPLWV_TMP
                CALL SPLIT_CMPLX_ATOMIC_ADD_FROM_CMPLX(CRACC(IP,NP),CTMP*WDES1%RINPL)
             ENDDO
             ENDDO
          ENDDO
DOOFF !$OMP END TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD
```

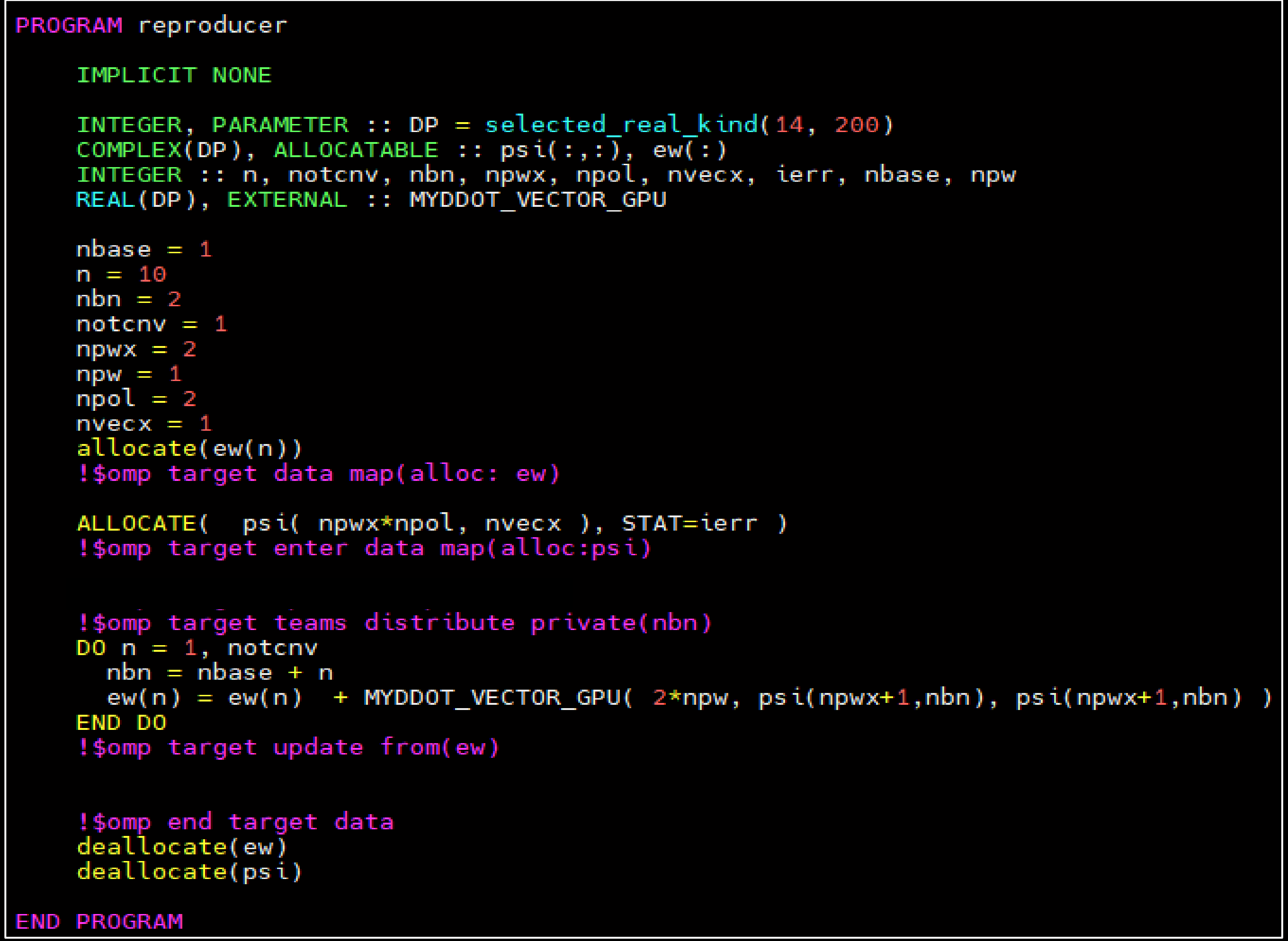


```
DOOFF !$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD PRIVATE(ISPIRAL,NI,NP,NT,LMMAXC,INDMAX,LMBASE,NLIIND,IBLOC
DO ITER=0,COUNTER-1
        ISPINOR=TOT_ITER(ITER*3+1)
        NI=TOT_ITER(ITER*3+2)
        NP=TOT_ITER(ITER*3+3)
             NT=NONLR_S%ITYP(NI)

             LMMAXC=NONLR_S%LMMAX(NT)

             INDMAX=NONLR_S%NLIMAX(NI __NOACC_omp_arg(i))

             LMBASE=NONLR_S%LMBASE(NI)+ISPINOR*NONLR_S%LMBASE(NONLR_S%NIONS+1)
             NLIIND=NONLR_S%NLIBASE(NI __NOACC_omp_arg(i))
             ISPIRAL=1; IF (NONLR_S%LSPIRAL) ISPIRAL=ISPINOR+1

             DO IBLOCK=0,INDMAX/BLOCKSIZE
             DO IND=IBLOCK*BLOCKSIZE+1,MIN((IBLOCK+1)*BLOCKSIZE,INDMAX)

                CTMP=0
                DO L=1,LMMAXC
                   CTMP=CTMP+CPROJ(L+LMBASE,NP)*NONLR_S%RPROJ(IND+(L-1)*INDMAX+NLIIND __NOACC_omp_arg(i))
                ENDDO
#ifndef gammareal
                CTMP=CTMP*CONJG(NONLR_S%CRREXP(IND,NI,ISPIRAL __NOACC_omp_arg(i)))
#endif
                IP=NONLR_S%NLI(IND,NI __NOACC_omp_arg(i))+ISPINOR*MPLWV_TMP
                CRACC1(IP,NP)=CRACC1(IP,NP)+CTMP*WDES1%RINPL
                CRACC2(IP,NP)=CRACC2(IP,NP)+CTMP*WDES1%RINPL
             ENDDO
             ENDDO
          ENDDO
DOOFF !$OMP END TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD
```

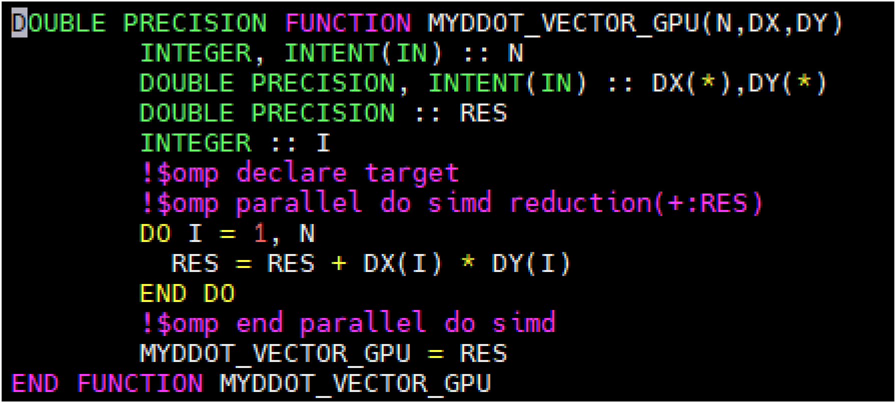
Kernel time= 80 ms

Kernel time= 22 ms

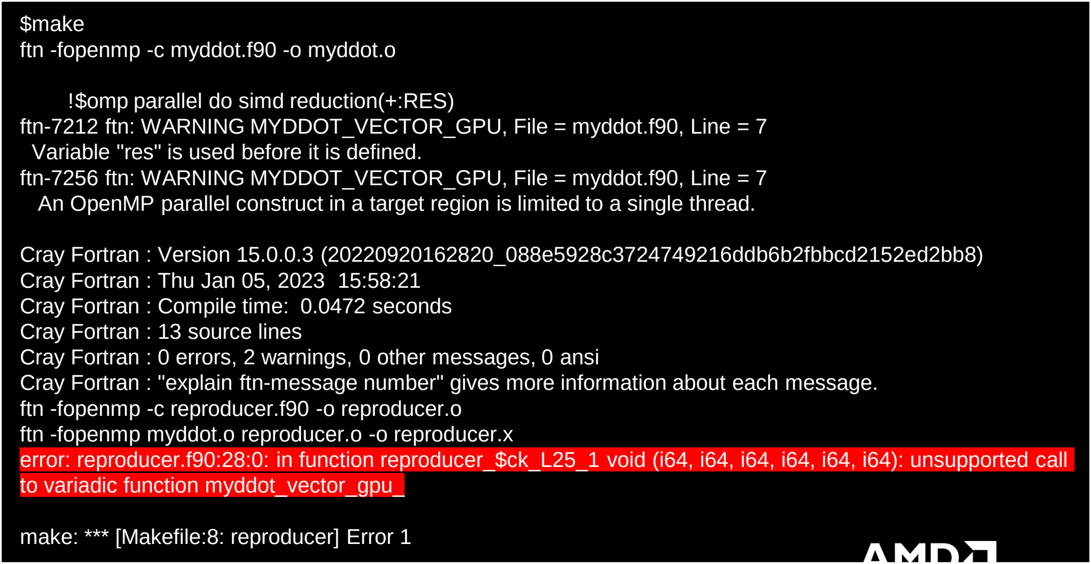
Declare target



```fortran
PROGRAM reproducer

    IMPLICIT NONE

    INTEGER, PARAMETER :: DP = selected_real_kind(14, 200)
    COMPLEX(DP), ALLOCATABLE :: psi(:,:), ew(:)
    INTEGER :: n, notcnv, nbn, npwx, npol, nvecx, ierr, nbase, npw
    REAL(DP), EXTERNAL :: MYDDOT_VECTOR_GPU

    nbase = 1
    n = 10
    nbn = 2
    notcnv = 1
    npwx = 2
    npw = 1
    npol = 2
    nvecx = 1
    allocate(ew(n))
    !$omp target data map(alloc: ew)

    ALLOCATE(  psi( npwx*npol, nvecx ), STAT=ierr )
    !$omp target enter data map(alloc:psi)


    !$omp target teams distribute private(nbn)
    DO n = 1, notcnv
      nbn = nbase + n
      ew(n) = ew(n)  + MYDDOT_VECTOR_GPU( 2*npw, psi(npwx+1,nbn), psi(npwx+1,nbn) )
    END DO
    !$omp target update from(ew)


    !$omp end target data
    deallocate(ew)
    deallocate(psi)

END PROGRAM
```

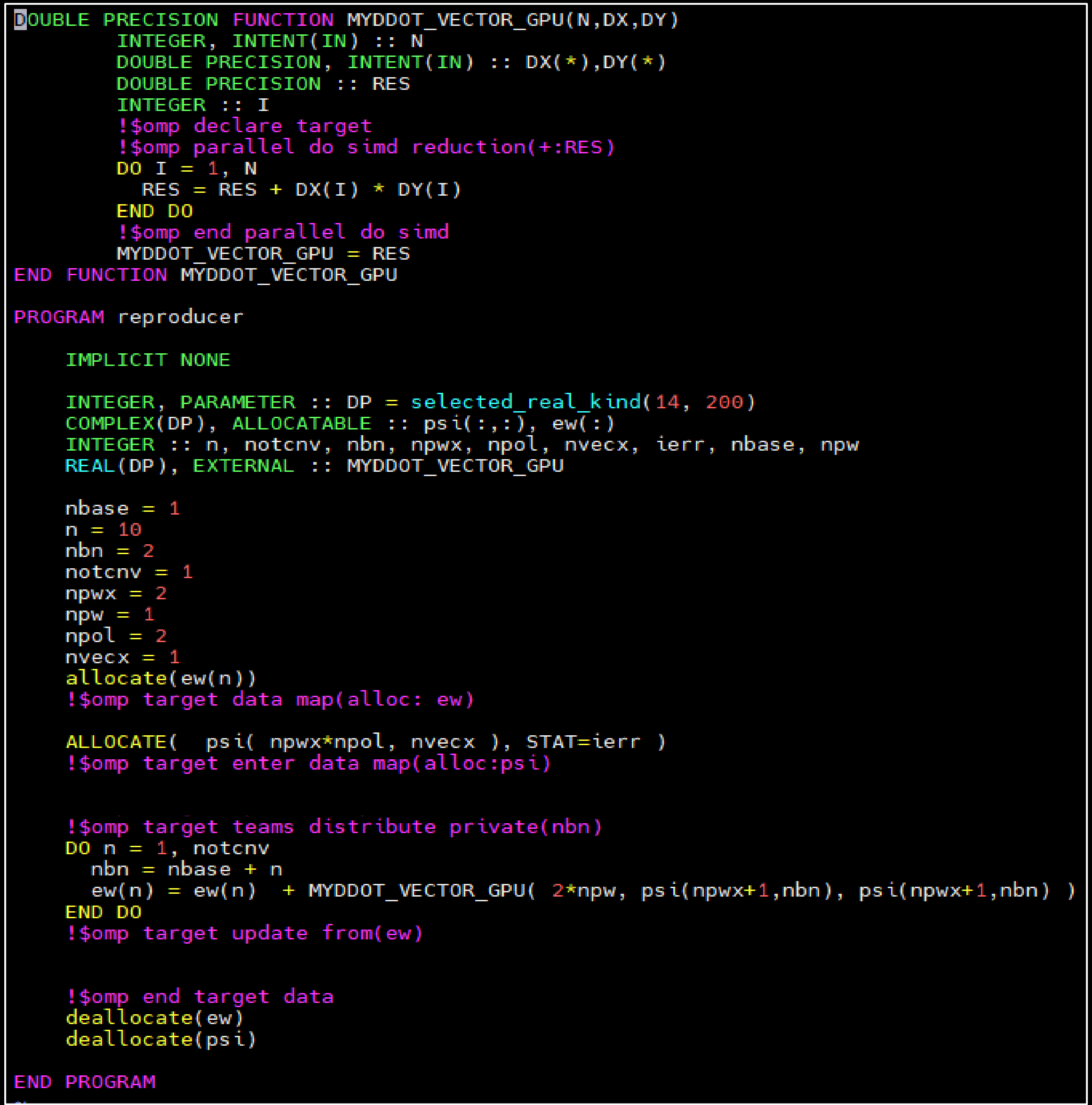


```fortran
DOUBLE PRECISION FUNCTION MYDDOT_VECTOR_GPU(N,DX,DY)
        INTEGER, INTENT(IN) :: N
        DOUBLE PRECISION, INTENT(IN) :: DX(*),DY(*)
        DOUBLE PRECISION :: RES
        INTEGER :: I
        !$omp declare target
        !$omp parallel do simd reduction(+:RES)
        DO I = 1, N
          RES = RES + DX(I) * DY(I)
        END DO
        !$omp end parallel do simd
        MYDDOT_VECTOR_GPU = RES
END FUNCTION MYDDOT_VECTOR_GPU
```



```
$make
ftn -fopenmp -c myddot.f90 -o myddot.o

        !$omp parallel do simd reduction(+:RES)
ftn-7212 ftn: WARNING MYDDOT_VECTOR_GPU, File = myddot.f90, Line = 7
  Variable "res" is used before it is defined.
ftn-7256 ftn: WARNING MYDDOT_VECTOR_GPU, File = myddot.f90, Line = 7
   An OpenMP parallel construct in a target region is limited to a single thread.

Cray Fortran : Version 15.0.0.3 (20220920162820_088e5928c3724749216ddb6b2fbbcd2152ed2bb8)
Cray Fortran : Thu Jan 05, 2023  15:58:21
Cray Fortran : Compile time:  0.0472 seconds
Cray Fortran : 13 source lines
Cray Fortran : 0 errors, 2 warnings, 0 other messages, 0 ansi
Cray Fortran : "explain ftn-message number" gives more information about each message.
ftn -fopenmp -c reproducer.f90 -o reproducer.o
ftn -fopenmp myddot.o reproducer.o -o reproducer.x
error: reproducer.f90:28:0: in function reproducer_$ck_L25_1 void (i64, i64, i64, i64, i64, i64): unsupported call
to variadic function myddot_vector_gpu_

make: *** [Makefile:8: reproducer] Error 1
```

Declare target (alternative method)



```
DOUBLE PRECISION FUNCTION MYDDOT_VECTOR_GPU(N,DX,DY)
        INTEGER, INTENT(IN) :: N
        DOUBLE PRECISION, INTENT(IN) :: DX(*),DY(*)
        DOUBLE PRECISION :: RES
        INTEGER :: I
        !$omp declare target
        !$omp parallel do simd reduction(+:RES)
        DO I = 1, N
          RES = RES + DX(I) * DY(I)
        END DO
        !$omp end parallel do simd
        MYDDOT_VECTOR_GPU = RES
END FUNCTION MYDDOT_VECTOR_GPU

PROGRAM reproducer

    IMPLICIT NONE

    INTEGER, PARAMETER :: DP = selected_real_kind(14, 200)
    COMPLEX(DP), ALLOCATABLE :: psi(:,:), ew(:)
    INTEGER :: n, notcnv, nbn, npwx, npol, nvecx, ierr, nbase, npw
    REAL(DP), EXTERNAL :: MYDDOT_VECTOR_GPU

    nbase = 1
    n = 10
    nbn = 2
    notcnv = 1
    npwx = 2
    npw = 1
    npol = 2
    nvecx = 1
    allocate(ew(n))
    !$omp target data map(alloc: ew)

    ALLOCATE(  psi( npwx*npol, nvecx ), STAT=ierr )
    !$omp target enter data map(alloc:psi)


    !$omp target teams distribute private(nbn)
    DO n = 1, notcnv
      nbn = nbase + n
      ew(n) = ew(n)  + MYDDOT_VECTOR_GPU( 2*npw, psi(npwx+1,nbn), psi(npwx+1,nbn) )
    END DO
    !$omp target update from(ew)


    !$omp end target data
    deallocate(ew)
    deallocate(psi)

END PROGRAM
```

- To get around the error, we can define function in the same file as function call
  - It would be challenging to apply his workaround in the applications with many function/subroutine calls

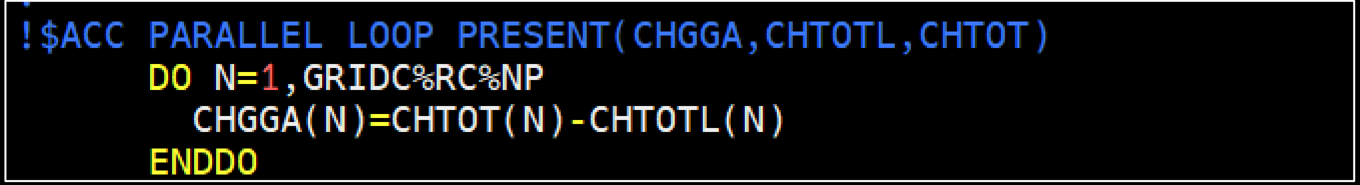
AMD
together we advance_

Content

- Introduction
- Debugging and profiling OpenMP offloading code in VASP
- OpenMP Offloading Challenges in VASP
  - Concurrent support for different directive-based paradigms
  - Enable/disable offloading in different code paths
  - Interface OMP offloading with ROCM libraries
- Compiler related challenges
  - Pointer aliasing
  - Pointer mismatch in subroutine calls
  - Atomic update
  - Declare target
- **Data management**
- Summary

Data management

- Data management is challenging in porting big applications like VASP
- The present clause in OpenACC is very helpful for data management in VASP
- In OpenMP offloading, omp_target_is_present can be used but it makes the code unmaintainable
- Present clause in OpenMP would be very useful for debugging and performance optimization



```
!$ACC PARALLEL LOOP PRESENT(CHGGA,CHTOTL,CHTOT)
      DO N=1,GRIDC%RC%NP
        CHGGA(N)=CHTOT(N)-CHTOTL(N)
      ENDDO
```

Summary

- Debugging and profiling OpenMP offloading code on AMD GPUs
- Discussed the challenges in adding OpenMP offloading support in VASP
- Compiler related challenges
  - Having a standard benchmark for capturing the compiler related issues would be helpful
- Data management
  - Having present clause in OMP offloading would be helpful to better deal with data management in big applications

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED ‘AS IS.” AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Third-party content is licensed to you directly by the third party that owns the content and is not licensed to you by AMD. ALL LINKED THIRD-PARTY CONTENT IS PROVIDED “AS IS” WITHOUT A WARRANTY OF ANY KIND. USE OF SUCH THIRD-PARTY CONTENT IS DONE AT YOUR SOLE DISCRETION AND UNDER NO CIRCUMSTANCES WILL AMD BE LIABLE TO YOU FOR ANY THIRD-PARTY CONTENT. YOU ASSUME ALL RISK AND ARE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY ARISE FROM YOUR USE OF THIRD-PARTY CONTENT.

AMD