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
  M=(2,2)
  N=(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

CRAY_ACC_DEBUG=2

CRAY_ACC_DEBUG=3
Debugging with Cray compiler: –hlist=aimd

```fortran
program test
  integer :: i
  complex, pointer :: A(:)
  allocate(A(500))
  do i=1, 500
    A(i) = (0,0)
  enddo
  !$omp target teams distribute parallel do simd map(from:A)
  do i=1, 500
    A(i)= (2,2)
  enddo
  !$omp end target teams distribute parallel do simd
  write(*,*) "A(1)= ", A(1)
end program test
```

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

```
1. program test
  2. integer :: i
  3. complex, pointer :: A(:)
  4. allocate(A(500))
  5. do i=1, 500
  6.  A(i) = (0,0)
  7. enddo
  8. !$omp target teams distribute parallel do simd map(from:A)
  9. do i=1, 500
  10.   A(i)= (2,2)
  11. enddo
  12. !$omp end target teams distribute parallel do simd
  13. write(*,*) "A(1)= ", A(1)
end program test
```
Profiling OpenMP® offloading code on AMD GPUs

- After compiling the code, run it with “rocprof --hip-trace”
  
  ```
  $ ftn -hnoacc -fopenmp -omp -o ./test ./test.f90
  $ rocprof --hip-trace ./test
  ```

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

Used when VASP is compiled with OpenACC

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

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

```fortran
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)
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

```c
void hip_zgemm(void *ptr, int modeA, int modeB, int n, 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_create();
  rocblas_double_complex *alpha2 = reinterpret_cast<rocblas_double_complex>(alpha);
  rocblas_double_complex *B = reinterpret_cast<rocblas_double_complex>(B);
  rocblas_double_complex *C = reinterpret_cast<rocblas_double_complex>(C);
  rocblas_double_complex *alpha2 = reinterpret_cast<rocblas_double_complex>(alpha);
  rocblas_double_complex *beta = reinterpret_cast<rocblas_double_complex>(beta);
  rocblas_handle *handle = rocblas_handle_create();
  rocblas_double_complex alpha2 = reinterpret_cast<rocblas_double_complex>(alpha);
  rocblas_double_complex beta = reinterpret_cast<rocblas_double_complex>(beta);
  rocblas_double_complex *A = reinterpret_cast<rocblas_double_complex>(A);
  rocblas_double_complex *B = reinterpret_cast<rocblas_double_complex>(B);
  rocblas_double_complex *C = reinterpret_cast<rocblas_double_complex>(C);
  rocblas_handle *handle = rocblas_handle_create();
}
```
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 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

```c
MODULE wave_struct_def
  TYPE wavedos
    INTEGER_POINTER :: LMMAXX(::)
  END TYPE wavedos
  TYPE wavedos1
    INTEGER_POINTER :: LMMAXX(::) => NULL()
  END TYPE wavedos1
END MODULE wave_struct_def

program test
  USE wave_struct_def
  INTEGER :: i, j, k, n
  TYPE (wavedos) WDES
  TYPE (wavedos1) 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)
  !$OMP TARGET ENTER DATA MAP( TO: WDES%LMMAXX)
  ! use WDES / WDES%LMMAXX in different loops/directives
  WDES%LMMAXX => WDES$%LMMAXX
  !$OMP TARGET ENTER DATA MAP( TO: WDES1)
  !$OMP TARGET ENTER DATA MAP( TO: WDES1%LMMAXX)
  !$OMP TARGET TEAM DISTRIBUTED MAP( FROM: OUTPUT)
  do i=1, N
    OUTPUT(i) = WDES1$%LMMAXX(i)
  enddo
  !$OMP END TARGET TEAM DISTRIBUTED
  do i=1, N
    WRITE(*,*) " OUTPUT(", i, ", "); OUTPUT(i)
  enddo
  !$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)
end program test
```
Pointer aliasing (alternative methods)

---

**Launch a kernel**

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

TYPE wavedes1
  INTEGER, POINTER :: LMAXX(:) => 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%LMAXX(N))
ALLOCATE(OUTPUT(N))
do i=1, N
  WDES%LMAXX(i) = 1
  OUTPUT(i) = 0
enddo
!$OMP TARGET ENTER DATA MAP(TO:WDES)
!$OMP TARGET ENTER DATA MAP(TO:WDES%LMAXX)
!$OMP TARGET ENTER DATA MAP(TO:WDES%LMAXX)
! use WDES / WDES%LMAXX in different loops/directives

!$OMP TARGET
WDES%LMAXX => WDES%LMAXX
!$OMP END TARGET
---
```

**Using target data construct**

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

TYPE wavedes1
  REAL, POINTER :: LMAXX(:) => NULL()
END TYPE wavedes1

END MODULE wave_struct_def

program test
use wave_struct_def
!!$omp requires unified_shared_memory
integer :: i, j, k, N, q
TYPE (wavedes) WDES
TYPE (wavedes1) WDES1
REAL, POINTER :: OUTPUT(:)
N=10
do q=1, 1000000
  ALLOCATE(WDES%LMAXX(N))
  ALLOCATE(OUTPUT(N))
  do i=1, N
    WDES%LMAXX(i) = 1
    OUTPUT(i) = 0
  enddo
  !$OMP TARGET ENTER DATA MAP(TO:WDES)
  !$OMP TARGET ENTER DATA MAP(TO:WDES%LMAXX)
  !$OMP TARGET ENTER DATA MAP(TO:WDES%LMAXX)
  !$OMP TARGET ENTER DATA MAP(TO:WDES%LMAXX)
  ! use WDES / WDES%LMAXX in different loops/directives
  !$OMP TARGET DATA USE_DEVICE_PTR(WDES)
  WDES%LMAXX => WDES%LMAXX
  !$OMP END TARGET DATA
  !$OMP TARGET DATA USE_DEVICE_PTR(WDES)
  WDES%LMAXX => WDES%LMAXX
  !$OMP END TARGET DATA
  !$OMP TARGET DATA USE_DEVICE_PTR(WDES)
  WDES%LMAXX => WDES%LMAXX
  !$OMP END TARGET DATA
  !$OMP TARGET EXIT DATA MAP(EXIT:WDES)
  !$OMP TARGET EXIT DATA MAP(EXIT:WDES)
  !$OMP TARGET EXIT DATA MAP(EXIT:WDES)
  deallocate(OUTPUT)
  deallocate(WDES%LMAXX)
enddo
end program test
```
Pointer mismatch in subroutine calls

```fortran
MODULE wave_struct_def
  TYPE wavespin
    COMPLEX(8), POINTER :: CPTWFP(:,:)
    REAL, POINTER :: PROJ(:,:)
  END TYPE wavespin
  TYPE wavefun1
    COMPLEX(8), POINTER, CONTIGUOUS :: CPTWFP(:) => NULL() 
    REAL, POINTER, CONTIGUOUS :: PROJ(:) => NULL() 
    COMPLEX(8), POINTER, CONTIGUOUS :: PROJ(:) => NULL() 
  END TYPE wavefun1
END MODULE wave_struct_def

program PointerAliasing
  use wave_struct_def
  TYPE (wavefun1), TARGET :: W1(10)
  INTEGER NIP, NSIM
  ALLOCATE(W1%CPTWFP(100,100))
  ALLOCATE(W1%PROJ(100,100))
  NSIM=10
  !$OMP TARGET ENTER DATA MAP(TO:W1)
  DO NIP=1, NSIM
    CALL NEWAV_W(R(W1(NP)))
  ENDDO
  DO NIP=1, NSIM
    CALL SETWAV(W1,NP), I)
    CALL EECP(W1(NP))
  ENDDO
END PROGRAM

SUBROUTINE NEWAV_W(W1)
  use wave_struct_def
  TYPE (wavefun1), INTENT(INOUT) :: W1
  INTEGER MPLW
  MPLW=100
  ALLOCATE(W1%CR(MPLW))
  !$OMP TARGET ENTER DATA MAP(ALLOC:W1%CR)
  !$OMP TARGET W1%CR(1,1)
  !$OMP END TARGET
END SUBROUTINE

SUBROUTINE DELAV_W(W1)
  use wave_struct_def
  TYPE (wavefun1) :: W1
  !$OMP TARGET EXIT DATA MAP(DELETE:W1%CR)
  DEALLOCATE(W1%CR)
END SUBROUTINE

SUBROUTINE EECP(W1)
  use wave_struct_def
  TYPE (wavefun1) :: W1
  INTEGER MM
  COMPLEX(8), TARGET :: CE
  CE=0
  !$OMP TARGET TEAMs DISTIBUTE PARALLEL DO SIMD REDUCTION(+:CE)
  DO MM =1, 100
    CE=CE+W1%CR(MM)
  ENDDO
  !$OMP END TARGET TEAMs DISTIBUTE PARALLEL DO SIMD
  write(++,*) "ce = ", CE
END SUBROUTINE

SUBROUTINE SETWAV(W1,W1,I)
  use wave_struct_def
  TYPE (wavefun1), INTENT(IN) :: W
  TYPE (wavefun1), INTENT(INOUT) :: W1
  INTEGER I, NIR
  !$OMP TARGET EXIT DATA MAP(DELETE:W1%CPTWFP)
  !$OMP TARGET EXIT DATA MAP(DELETE:W1%PROJ)
  W1%CPTWFP= W%CPTWFP(:,I)
  W1%PROJ = W%PROJ(:,I)
  !$OMP TARGET ENTER DATA MAP(TO:W1%CPTWFP,W1%PROJ)
END SUBROUTINE
```
Pointer mismatch in subroutine calls (alternative method)

```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, only: W
  use wavefun, only: TARGET ! W(10)
  integer np,nsim
  allocate(w1(100,100)), allocate(w1(10,10))
  nsim=10
  !$OMP target enter data map(t0:W1)
  do np=1,nsim
    call setwav(w1(np))
  enddo
  do i=1,10
    call setwav(w1(np),i)
    call eccp(w1,nsim)
  enddo
end program
```

```fortran
SUBROUTINE NEWMAY_R(WI)
  use wave_struct_def
  type (wavespin), intent(inout) :: W
  integer npw

  npw=100
  allocate(w1cr(npw))
  !$OMP target enter data map(w1cr)
  !$OMP target warnings(none)
  !$OMP end target

  subroutine DELAYR_R(WI)
  use wave_struct_def
  type (wavefun), wi
  !$OMP target enter data map(delete:w1cr)
  !$OMP target warnings(none)
  !$OMP end target

  subroutine ECCP_R(WI, NP)
  use wave_struct_def
  type (wavefun), wi
  integer mm
  complex(8), target :: ce
  ce=0
  !$OMP target teams distribute parallel do simd reduction(+,ce)
  do mm=1,10
    ce=wi*mm+cr(mm)
  enddo
  !$OMP end target teams distribute parallel do simd
  write(*,*), "ce=", ce
end subroutine
```

```fortran
SUBROUTINE SETTAV_W(W, I)
  use wave_struct_def
  type (wavespin), intent(in) :: W
  type (wavefun), intent(inout) :: WI
  integer i,j
  !$OMP target enter data map(delete:w1cr)
  !$OMP target enter data map(delete:w1cproj)

  do i=1,mp
    w1cproj = w1cproj + c*cr(i)
  enddo
  !$OMP end target enter data map(w1cr,w1cproj)
end subroutine
```
Atomic update for complex(8)

Original code

```fortran
program test
  integer :: i,j,N,M,k2,k
  complex(8) :: B($1,42)$, C($1,42$), X

  N=3000
  M=100
  do i=1, M
    do j=1, N/M
      C(j)=0
      enddo
    X=(1,1)
    !$omp target teams distribute map(tofrom:8) private(k,k2)
    do i=1, M
      !$omp parallel do
      do j=1, N/M
        k=(j*(N/M))
        k2=mod(k,4)+1
        C(k)=C(k2)+X
        enddo
      enddo
    !$omp end parallel do
    enddo
  !$omp end target teams distribute
  write(*,*) "B(1,1)=", B(1,1), " C(1,1)=", C(1,1)
end program test
```

Alternative

```fortran
program test
  integer :: i,j,N,M,k2,k
  complex(8) :: B($1,42$), C($1,42$), X

  N=3000
  M=100
  do i=1, M
    do j=1, N/M
      k=(i*(N/M))+j
      k2=mod(k,40)+1
      C(k)=C(k2)+X
    enddo
  enddo
  !$omp target teams distribute map(tofrom:8) private(k,k2)
  do i=1, M
    !$omp parallel do
    do j=1, N/M
      k=(i*(N/M))+j
      k2=mod(k,40)+1
      C(k)=C(k2)+X
    enddo
  enddo
  !$omp end parallel do
  enddo
  !$omp end target teams distribute
  write(*,*) "B(1,1)=", B(1,1), " C(1,1)=", C(1,1)
end program test
```

**Note:** The original code has been adapted to demonstrate the use of atomic updates in Fortran. The alternative code implements the same logic but with explicit atomic updates to ensure thread safety in multi-threaded environments. The changes highlight the importance of considering thread safety in complex operations involving shared data.
The overhead of subroutine call assuming there is no need for atomic update

Kernel time = 80 ms

Kernel time = 22 ms
Declare target

```
PROGRAM reproducer

IMPLICIT NONE

INTEGER, PARAMETER :: DP = selected_real_kind(14, 209)
COMPLEX(DP), ALLOCATABLE :: psi(:,:), ew(:)
INTEGER :: ntotcnv, nbn, npwx, npol, nvecx, ierr, nbase, npw
REAL(DP), EXTERNAL :: MYDDOT_VECTOR_GPU

nbase = 1
n = 10
nbn = 2
ntotcnv = 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)
!
!$omp target teams distribute private(nbn)
DO n = 1, ntotcnv
 nbn = nbase + n
 ew(n) = ew(n) + MYDDOT_VECTOR_GPU( 2*npw, psi(npwx+1,nbn), psi(npw+1,nbn) )
END DO
!$omp target update from(ew)
!
!$omp end target data
deallocate(ew)
deallocate(psi)

END PROGRAM
```

```
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
```

```
$make
fnp-openmp -c myddot.190 -o myddot.o
 !$omp parallel do simd reduction(+:RES)
 fnp-7212 fnp: WARNING MYDDOT_VECTOR_GPU, File = myddot.190, Line = 7
 Variable "res" is used before it is defined.
 fnp-7256 fnp: WARNING MYDDOT_VECTOR_GPU, File = myddot.190, Line = 7
 An OpenMP parallel construct in a target region is limited to a single thread.

Cray Fortran : Version 15.0.0.3 (20220920162820_088e5928c3724749216ddb52fbbcd2152ed2bb8)
Cray Fortran : Thu Jan 05, 2023 15:58:21
Cray Fortran : Compile time: 0.0472 seconds
Cray Fortran : 13 source lines
Cray Fortran : 13 source lines
Cray Fortran : 0 errors, 2 warnings, 0 other messages, 0 ansi
Cray Fortran : `explain fnp-message number` gives more information about each message.
 fnp -lopenmp -c reproducer.190 -o reproducer.o
 fnp -lopenmp myddot.o reproducer.o -o reproducer.x
error: reproducer.190:28:0: 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)

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
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
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.

© 2023 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, ROCm, and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.

The OpenMP name and the OpenMP logo are registered trademarks of the OpenMP Architecture Review Board