WHAT'S NEW IN INTEL® FORTRAN 19.1

March 2019

Agenda

- New Features from Fortran 2018
- OpenMP 5.0 Features
- Other features

FEATURES FROM FORTRAN 2018

Obsolescent features

- **Arithmetic IF statement** is deleted in F2018. Standards warning message will be issued.

  IF (cond) LABEL1, LABEL2, LABEL3

- Referencing **intrinsic functions by their specific names** is obsolescent. Generic names are preferred. Standards warning will be issued.

- Labeled DO loops are now obsolescent. Standards warning for F2018 and later.

```
INTEGER I
DO 10, I = 1, 5
PRINT *, I
10 END DO
END
```

Obsolescent features

- Common blocks, EQUIVALENCE statements, and BLOCK DATA program units are obsolescent. Warning issued for F2018 and later.
- The nonblock forms of the do loop are deleted. This includes the shared termination forms of the do loop. Standards warning for F2018 and later.

```
Integer i, j
do 30, i = 1, 5
  do 30, j = 1, 3
    print *, "doing..."
30 continue
end
```

Diagnostics for non-standard procedures

Warnings issued for

- Any reference to an INTEL provided intrinsic procedure.
- USE of an INTEL provided intrinsic module.
- Any reference to a procedure in a standard defined intrinsic module which is not specified by the standard.

```
use, intrinsic :: iso_c_binding, only : c_random_non_standard_sub !non-standard
                                                                  !intrinsic

use, intrinsic :: mod                                    !non-standard module

use, non_intrinsic :: iso_c_binding, only : c_random_non_standard_sub !no warning
                                                                      !non-intrinsic use
```

Inquiry functions classified as Transformational

- The following Inquiry functions are now transformational functions.
  - IEEE_SUPPORT_FLAG, IEEE_SUPPORT_HALTING from the intrinsic module IEEE_ARITHMETIC.
  - IEEE_SUPPORT_ROUNDING from the intrinsic module IEEE_EXCEPTIONS.
  - C_ASSOCIATED, C_LOC, and C_FUNLOC from the intrinsic module ISO_C_BINDING.
- These IEEE functions are allowed in constant and specification expressions, ISO_C_BINDING functions are permitted in specification expressions.
- Example

```
SUBROUTINE sub (a, b)
  use, intrinsic :: ISO_C_BINDING
  use, intrinsic :: IEEE_ARITHMETIC
  INTEGER(KIND=C_INT) :: a
  TYPE(C_PTR) :: b, cptr => C_LOC (a)    !specification expression
  LOGICAL, parameter :: l1 = IEEE_SUPPORT_FLAG (IEEE_INEXACT, 4) !constant expression
  LOGICAL :: l2 = IEEE_SUPPORT_HALTING (IEEE_INEXACT, b) !specification expression
END
```

Enhancements to CMPLX and SIGN intrinsics

- CMPLX(X [, Y, KIND]) has two interfaces in F2018.
  CMPLX(X [, KIND]).
  -X shall be of type complex.
  -KIND (optional) shall be a scalar integer constant expression.
  CMPLX(X [, Y, KIND]).
  -X shall be of type integer or real, or a <boz-literal-constant>.
  -Y (optional) shall be of type integer or real, or a <boz-literal-constant>.
  -KIND (optional) shall be a scalar integer constant expression.

- When X is complex, first interface is used, which has no “Y” arg. So, the requirement that “no actual argument shall correspond to Y if the argument X is of type complex” is not needed.

- Therefore, in references to intrinsic CMPLX with a complex actual argument, **no keyword is needed** for the KIND argument.

- The arguments to the **SIGN function** can be of different kinds.

Optional STAT= and ERRMSG= to intrinsics and constructs

- ATOMIC_DEFINE and ATOMIC_REF intrinsics can have STAT arguments.
- MOVE_ALLOC intrinsic and CRITICAL constructs can have STAT= and ERRMSG=. Image selectors can now have STAT= specifier.

```
REAL :: C[*], MY_COUNTER[*]
REAL, allocatable :: co_reg [:,:], co_reg_local[:,:]
INTEGER :: i, istatus
Character :: ch
C[1,STAT=i] = 4.0 !Image selector
CRITICAL(stat=istatus, errmsg = ch)
  MY_COUNTER[1] = MY_COUNTER[1] + 1
END CRITICAL
...
move_alloc(co_reg_local,co_reg,istatus,ch) !stat= and errmsg= arguments
```

SELECT RANK Construct

- Selects one of its constituent blocks for execution based on the **rank of an assumed rank** variable. Similar to Select Type construct which selects based on “Type”.
- An assumed rank variable is a dummy argument whose rank is inherited from the actual argument associated with it.
- The syntax is [name:] SELECT RANK ([assoc-name =>] selector)

  [rank-case-stmt
  block]...
  END SELECT [name]

  Each ***rank-case-stmt*** is one of the following:
  RANK (scalar-int-const-expr) [name]
  RANK (*) [name]
  RANK DEFAULT [name]
- A select rank construct selects at most one block to be executed. **If** the actual argument corresponding to the selector is an assumed-size array, a RANK (*) statement block is executed.
- Else, a RANK (*scalar-int-const-exp*) block that matches the rank of the selector is chosen. Otherwise, if there is a RANK (DEFAULT) statement, the block following that statement is executed.

SELECT RANK construct - Example

```
PROGRAM select_rank
 real :: a, b(5), d(5,5,5)
 call process_array(a) !Rank(0) block
 call process_array(b) !Rank(1) block
 call sub(d) !Rank (*) block
 contains
 SUBROUTINE sub (y)
  real :: y(*)       !Assumed-size
  call process_array(y)
 END SUBROUTINE
 SUBROUTINE process_array(x)
  real :: x(..) !Assumed rank array
  SELECT RANK(y=>x)
   RANK (0)
     y = 0
     print *, RANK(y) !Should print 0
   RANK (1)
     y(::2) = 1
     print *, RANK(y) !Should print 1
```

SELECT RANK construct

```
    RANK (*)
      do i=1,125
       y(i) = 3
      end do
      print *, RANK(y) !Should print 1
    RANK DEFAULT
      print *, 'Error: Unexpected rank ', RANK(y)
  END SELECT
 END SUBROUTINE process_array
END PROGRAM select_rank
```

Expected output is 0 1 1

- A branch from within a block of a SELECT RANK construct to the END SELECT statement or to any statement within the block or to outside the END SELECT statement is allowed.
- Branches to the END SELECT from outside the construct are not permitted. Branches to statement within another rank-case-stmt block is not allowed.

A GENERIC statement to declare generic interfaces

- GENERIC statement is an alternative to interface block.
- GENERIC [, <access-spec>] :: <generic-spec> => <specific-procedure-list>

```
Interface write(formatted)
   module procedure WriteSomeType
end interface
generic :: write(formatted) => WriteSomeType !Same as above interface block
-----------------------------------------------------------------------------------
module m1
 generic, public :: gen => mp1
contains
 subroutine mp
  generic :: gen=>mp2  !Combining generic from the host scope.
...
 end subroutine
...
-----------------------------------------------------------------------------------
 interface gen1
   module procedure mp1, mp2
 end interface
 generic :: gen1 => mp2   !Error, generic has duplicate specific procedures.
 generic :: gen1 => mp3   !This is okay
```

IMPLICIT NONE (EXTERNAL | TYPE)

- IMPLICIT NONE (TYPE) is the same as the existing IMPLICIT NONE.
- IMPLICIT NONE (EXTERNAL) means that any references to an external procedure must be to a name that is explicitly declared to have the EXTERNAL attribute. In other words, no implicit interfaces.
- If an IMPLICIT NONE or IMPLICIT NONE (TYPE) appears, there must be **no other IMPLICIT statements** in the scoping unit.
- No more than one IMPLICIT NONE statement shall appear in a scoping unit.
- -warn external would automatically turn on IMPLICIT NONE (EXTERNAL) similar to how -warn declarations turns on IMPLICIT NONE (TYPE).

```
subroutine foo
  implicit none (external)
  implicit real(i)                !This is okay
  REAL, EXTERNAL :: G
  REAL :: X, Y
  i = 0.0
  X = F (Y)        ! Invalid: F lacks the EXTERNAL attribute.
  X = G (Y)        ! Valid: G has the EXTERNAL attribute.
end subroutine
```

New ATOMIC intrinsics

- Atomic subroutines are intrinsic subroutines that perform an action on their ATOM argument atomically. If it has an OLD argument, the value to be assigned to that argument is also determined atomically with the action performed on the ATOM argument. The evaluation or definition of any other argument is not performed atomically.
- STAT is an optional output argument. The STAT argument, if present, becomes defined with the value zero if no error condition occurs.
- If STAT is present and an error condition occurs, any INTENT(INOUT) or INTENT(OUT) argument becomes undefined. If the ATOM argument is on a failed image, STAT becomes defined with “STAT_FAILED_IMAGE” from ISO_FORTRAN_ENV, and an error condition occurs. If any other error condition occurs, STAT becomes defined with a processor dependent processor value other than the value of STAT_FAILED_IMAGE.
- If an error condition occurs and STAT is not present, error termination is initiated.

New ATOMIC intrinsics

ATOMIC_ADD (ATOM, VALUE [, STAT]) – Atomic Addition

If N[12] = 4 when the atomic operation performed by CALL ATOMIC_ADD (N[12], 10) is initiated; N[12] is 14 when the atomic operation is complete.

ATOMIC_AND (ATOM, VALUE [, STAT]) – Atomic Bitwise And – ATOM value becomes IAND (ATOM, INT (VALUE, ATOMIC_INT_KIND))

If N[4] = 29, after CALL ATOMIC_AND (N[4], 22) is initiated; N[4] is 20 when the atomic operation is complete.

ATOMIC_CAS (ATOM, OLD, COMPARE, NEW [, STAT]) – Atomic compare and swap- ATOM is of type integer and equal to COMPARE, or type logical and equivalent to COMPARE, it becomes defined with the value of NEW.

If N[5] = 12, CALL ATOMIC_CAS (N[5], OLD, 12, 2) - N[5] = 2 and OLD becomes 12.

If N[5] = 13, CALL ATOMIC_CAS (N[5], OLD, 12, 2) - N[5] is unchanged and OLD is 13.

ATOMIC_FETCH_ADD (ATOM, VALUE, OLD [, STAT]) – Atomic Fetch and Add

If N[4] = 7, CALL ATOMIC_FETCH_ADD (N[4], 8, M) - N[4] becomes 15 and M is 7.

New ATOMIC intrinsics

ATOMIC_FETCH_AND (ATOM, VALUE, OLD [, STAT]) – Atomic Fetch and bitwise AND

If N[4] = 23, CALL ATOMIC_FETCH_AND (N[4], 29, M) - N[4] becomes 21 and M is 23.

ATOMIC_FETCH_OR (ATOM, VALUE, OLD [, STAT]) - Atomic fetch and bitwise OR

If N[4] = 4, CALL ATOMIC_FETCH_OR (N[4], 9, M) - N[4] becomes 13 and M = 4.

ATOMIC_FETCH_XOR (ATOM, VALUE, OLD [, STAT]) – Atomic Fetch and bitwise XOR

If N[4] = 10, CALL ATOMIC_FETCH_XOR (N[4], 9, M) - N[4] = 3 and M becomes 10.

ATOMIC_OR (ATOM, VALUE [, STAT]) – Atomic bitwise OR

If N[4] = 9, CALL ATOMIC_OR (N[4], 10) - N[4] becomes 11.

ATOMIC_XOR (ATOM, VALUE [, STAT]) – Atomic bitwise XOR

If N[4] = 14, CALL ATOMIC_XOR (N[4], 10) - N[4] becomes 14.

Coarray collective intrinsic procedures

- Five new collective intrinsic subroutines. CO_BROADCAST, CO_MIN, CO_MAX, CO_REDUCE, and CO_SUM.
- Collective subroutines perform a calculation on a team of images, assign the result to one of the images or all of the images on the current team. Synchronization is not required.
- The collective subroutine when invoked is invoked by the same statement on all active images of the current team. Corresponding references to the subroutine participate in the same collective operation.
- If STAT argument is present, after successful execution STAT becomes 0. If the current team contains a stopped image, an error condition occurs and STAT becomes STAT_STOPPED_IMAGE.
- If the current team contains a failed image, an error condition occurs and STAT is STAT_FAILED_IMAGE. If any other error condition occurs, STAT is a positive value other than that of STAT_STOPPED_IMAGE or STAT_FAILED_IMAGE.
- In 19.1, “Teams” are not yet supported, the only team for collective intrinsic subroutines is the initial team with all images.

Coarray collective intrinsic procedures

CO_BROADCAST (A, SOURCE_IMAGE [, STAT, ERRMSG])

If R is [10, 20, 30, 40] on image 5, CALL CO_BROADCAST (R, 5), the value of R on all images of the current team becomes [10, 20, 30, 40].

CO_MAX (A, [,RESULT_IMAGE, STAT, ERRMSG])

If number of images is 2, and R is [5, 10, 20, 15] on image 1 and [10, 15, 20, 5] on image 2.

CALL CO_MAX (R), value of R is [10, 15, 20, 15] on both images if no error occurs.

CALL CO_MAX (R, 1) - R on image 1 is [10, 15, 20, 15], R on image 2 is undefined.

CO_MIN (A, [,RESULT_IMAGE, STAT, ERRMSG]) –

If number of images is 2, R is [5, 10, 20, 15] on image 1 and [10, 15, 20, 5] on image 2.

CALL CO_MIN (R) - R becomes [5, 10, 20, 5] on both images if no error occurs.

CALL CO_MIN(R, 2) - R on image 2 becomes [5, 10, 20, 5] while the value of R on image 1 is undefined.

Coarray collective intrinsic procedures

CO_REDUCE (A, OPERATION, [,RESULT_IMAGE, STAT, ERRMSG]) – Generalized reduction across images.

```
SUBROUTINE CO_ANY (VALUE)
  LOGICAL,INTENT(INOUT)  :: VALUE (:)
  CALL CO_REDUCE (VALUE, COMBINER)
  CONTAINS
    PURE FUNCTION COMBINER (OPND1, OPND2) RESULT (LOGICAL_SUM)
    LOGICAL                 :: LOGICAL_SUM
    LOGICAL,INTENT(IN)      :: OPND1, OPND2
    LOGICAL_SUM = OPN1 .OR. OPND2
  END FUNCTION COMBINER
END SUBROUTINE CO_ANY
```

If number of images is 2, R is [.T., .T., .F., .F.] on image 1 and [.T., .F., .T., .F.] on image 2, CALL CO_ANY (R) - R becomes [.T., .T., .T., .F.] on both images if no error occurs.

CO_SUM (A [, RESULT_IMAGE, STAT, ERRMSG])

If number of images is 2, R is [5, 10, 20, 15] on image 1 and [10, 15, 20, 5] on image 2. CALL CO_SUM (R, 2) - R becomes [15, 25, 40, 20] on image 2 and R is undefined on image 1, if no error occurs.

CALL CO_SUM (R) - R becomes [15, 25, 40, 20] on both images.

Locality of variable can be declared on the DO CONCURRENT statement

The locality of variables used in a DO CONCURRENT construct may be declared in a new clause on the DO CONCURRENT statement. Four localities are allowed: LOCAL, LOCAL_INIT, SHARED and DEFAULT. No variable list is given if DEFAULT is specified.

```
..
INTEGER :: N = 10
 REAL :: X, Y, Z, R
 X = 1.0
 Y = 2.0
 Z = 3.0
 R = 4.0
 DO CONCURRENT (INTEGER :: I = 1 : N) LOCAL (X, R) LOCAL_INIT (Y) SHARED (Z)
  X = I + 1.0      !! X HAS VALUE OF I + 1.0
  R = Y          !! R HAS VALUE OF 2.0 !local_init begins each iteration with value of outside variable
  IF (I == 1) THEN
    Z = 4.0                              !Shared, same variable as outside
  END IF
 END DO
 PRINT*, X, Y, Z, R         !! 1.0, 2.0, 4.0, 4.0
..
```

SUBNORMAL is now synonymous with DENORMAL

- Change in terminology from the old IEEE standard.
- Denormal/Denormalized had become subnormal.
- IEEE_DENORMAL becomes IEEE_SUBNORMAL in IEEE_FEATURES.
- In IEEE_ARITHMETIC,
  - IEEE_NEGATIVE_DENORMAL becomes IEEE_NEGATIVE_SUBNORMAL.
  - IEEE_POSITIVE_DENORMAL becomes IEEE_POSITIVE_SUBNORMAL.
  - IEEE_SUPPORT_DENORMAL becomes IEEE_SUPPORT_SUBNORMAL.
- For backwards compatibility F2018 requires that old names still work. So, old names continue to work and are synonymous with the new names.

The SIZE= specifier for non-advancing I/O

- Allow SIZE= without ADVANCE=.
- This would allow you to determine the number of characters transferred by data edit descriptors, exclusive of padding, even if advancing input is used.
- Example

```
150 FORMAT (F10.2, F10.2, I6)
READ (UNIT=20, FMT=150, SIZE=X, ADVANCE='NO', EOR=700) A, F, I
```

Changes to Edit descriptors

- D, E, EN, and ES edit descriptors can have a field width of zero.
- Adds new edit descriptor forms D0.d, E0.d, E0.dEe, EN0.d, EN0.dEe, ES0.d and ES0.dEe.
- Analogous to the F edit descriptor

- The exponent width e in a data edit descriptor may be zero.
- "e" in the "Ee" part of the E, EN, ES, and G edit descriptors can have the value zero.
- E0 requests the exponent width to be minimal.
- Analogous to a field width of zero.

New EX edit descriptor

- The EX edit descriptor provides hexadecimal-significand formatted output conforming to ISO/IEC/IEEE 60559:2011.
- Floating-point formatted input accepts hexadecimal-significand numbers conforming to ISO/IEC/IEEE 60559:2011 - numeric input values in the form 0x...

IEEE_NEXT_DOWN() and IEEE_NEXT_UP()

- IEEE_NEXT_DOWN(X) - Adjacent lower machine number. Result is the greatest value in the representation method of X that compares less than X. When X = -infinity the result is –infinity. When X is a NaN the result is a NaN. If X is a signaling NaN, IEEE_INVALID signals, otherwise, no exception is signaled.

  If IEEE_SUPPORT_SUBNORMAL (0.0) is true, the value of IEEE_NEXT_DOWN (+0.0) is the negative subnormal number with least magnitude.

- IEEE_NEXT_UP(X) - Adjacent higher machine number. It is the least value in the representation method of X that compares greater than X. When X is equal to +infinity, the result has the value +infinity. When X is a NaN the result is a NaN. If X is a signaling NaN, IEEE_INVALID_signals, otherwise, no exception is signaled.

  If IEEE_SUPPORT_INFO (X) is true, the value of IEEE_NEXT_UP (HUGE(X)) is +infinity.

- IEEE_SUPPORT_DATATYPE (X) must be true. IEEE_NEXT_DOWN (-HUGE(X)) and IEEE_NEXT_UP (HUGE(X)) must not be invoked if IEEE_SUPPORT_INF (X) is false.

IEEE_FMA (), IEEE_SIGNBIT(), new optional arg to IEEE_RINT()

- IEEE_FMA (A, B, C) - Fused multiply-add operation. When the result is in range, its value is equal to the mathematical value of (A * B) + C rounded to the representation mode of A according to the rounding mode. Example. The value of IEEE_FMA(TINY(0.0), TINY(0.0), 1.0), when the rounding mode is IEEE_NEAREST, is equal to 1.0; only the IEEE_INEXACT exception is signaled.

- IEEE_SIGNBIT (X) - Tests sign bit. Result is true if and only if the sign bit of X is nonzero. No exception is signaled even if X is a signaling NaN. Example. IEEE_SIGNBIT (-1.0) has the value true.

- IEEE_RINT (X [, ROUND]). If ROUND is present, result is the value of X rounded to an integer according to the mode specified by ROUND. Otherwise, the value of X is rounded to an integer according to the rounding mode. If the result is zero, the sign is that of X. Examples. If the rounding mode is round to nearest, the value of IEEE_RINT (1.1) is 1.0. The value of IEEE_RINT (1.1, IEEE_UP) is 2.0.

New IEEE_AWAY rounding mode; Optional RADIX argument to IEEE_GET_ROUNDING_MODE() and IEEE_SET_ROUNDING_MODE()

- The new IEEE_AWAY rounding mode is required for decimal formats, but not required for binary formats. This is similar to Fortran's I/O rounding mode "COMPATIBLE". IEEE_AWAY is a new named constant of IEEE_ROUND_TYPE.
- The following functions have new specifications.

  IEEE_GET_ROUNDING_MODE (ROUND_VALUE [, RADIX])
  IEEE_SET_ROUNDING_MODE (ROUND_VALUE [, RADIX])

- Optional RADIX argument added. The decimal rounding mode can be inquired and set independently of the binary rounding mode, using RADIX argument to the IEEE_GET_ROUNDING_MODE and IEEE_SET_ROUNDING_MODE respectively.
- RADIX must be an integer scalar with the value two or ten. If RADIX is present with the value ten, the rounding mode queried/set is the decimal rounding mode. Otherwise, it is the binary rounding mode.

IEEE_MAX|MIN_NUM[_MAG] intrinsics

- IEEE_MAX_NUM (X, Y) - Maximum numeric value.

  Example. The value of IEEE_MAX_NUM (1.5, IEEE_VALUE(IEEE_QUIET_NAN)) is 1.5.

- IEEE_MAX_NUM_MAG (X, Y) - Maximum magnitude numeric value.

  Example. The value of IEEE_MAX_NUM_MAG (1.5, -2.5) is -2.5.

- IEEE_MIN_NUM (X, Y) - Minimum numeric value.

  Example. The value of IEEE_MIN_NUM (1.5, IEEE_VALUE(IEEE_QUIET_NAN)) is 1.5.

- IEEE_MIN_NUM_MAG (X, Y) - Minimum magnitude numeric value.

  Example. The value of IEEE_MIN_NUM_MAG (1.5, -2.5) is 1.5.

- Must not be invoked if IEEE_SUPPORT_DATATYPE (X) has the value false.

Implement IEEE_QUIET|SIGNALING_COMPARE where COMPARE is EQ, GE, GT, LE, LT, or NE.

Add new functions for the quiet versions, IEEE_QUIET_EQ(X,Y), IEEE_QUIET_NE(X,Y), IEEE_QUIET_GE(X,Y), IEEE_QUIET_GT(X,Y), IEEE_QUIET_LE(X,Y), and IEEE_QUIET_LT(X,Y).

New functions for the SIGNALING versions. IEEE_SIGNALING_EQ, IEEE_SIGNALING_NE, IEEE_SIGNALING_GT, IEEE_SIGNALING_GE, IEEE_SIGNALING_LT, IEEE_SIGNALING_LE

Not to be invoked if IEEE_SUPPORT_DATATYPE (A) has the value false.

IEEE_MODES_TYPE to IEEE_ARITHMETIC, IEEE_SET_MODES() and IEEE_GET_MODES().

- A new type IEEE_MODES_TYPE in the IEEE_ARITHMETIC intrinsic module.
- IEEE_GET_MODES(MODES) and IEEE_SET_MODES(MODES) to query and set rounding modes. MODES must be scalar of type IEEE_MODES_TYPE.
- In IEEE_GET_MODES(MODES), MODES is an INTENT (OUT) argument that is assigned the value of the floating-point modes.
- In IEEE_SET_MODES(MODES), MODES must be a value that was assigned by a previous invocation of IEEE_GET_MODES.

```
USE, INTRINSIC :: IEEE_ARITHMETIC
TYPE(IEEE_MODES_TYPE) SAVE_MODES
...
CALL IEEE_GET_MODES(SAVE_MODES)                ! Save all modes.
CALL IEEE_SET_ROUNDING_MODE(IEEE_TO_ZERO))
... ! calculation with abrupt round-to-zero.
CALL IEEE_SET_MODES(SAVE_MODES)                ! Restore all modes.
```

Specifiers in Inquire statement standardized.

- The value returned from the INQUIRE statement for the RECL= parameter has been standardized.
- In F2008, If there is no connection, or if the connection is for stream access, the scalar-int-variable becomes undefined. In F2018, If there is no connection, the scalar-int-variable is assigned the value -1, and if the connection is for stream access the scalar-int-variable is assigned the value -2.
- Values for POS= and SIZE= in an INQUIRE statement for pending asynchronous operations have been standardized.
- For both POS= and SIZE=, if there are pending data transfer operations for the specified unit, the value assigned is computed as if all the pending data transfers had already completed.

OPENMP FEATURES

Inclusive/exclusive scan in SIMD/TARGET SIMD

- The scan directive specifies that either an inclusive scan or exclusive scan computation is to be performed by the loop for each list item in the clause.
- A list item that appears in the inclusive or exclusive clause must appear in a reduction clause with the inscan modifier on the enclosing loop, loop SIMD, or simd construct.

```
!$dir omp simd reduction(inscan, +: s)
do i = 1, n s += a(i)
        !$dir omp scan inclusive(s)
        b(i) = s
end do
do i = 1, n
        b(i) = s
        !$dir omp scan exclusive(s)
        s += a(i)
enddo
```

IF and NONTEMPORAL clauses on SIMD directive

- The Open MP standard includes new omp simd clauses

  nontemporal(list) and

  if([simd :]scalar-logical-expression)

- The nontemporal clause specifies that accesses to the storage locations to which the list items refer have low temporal locality across the iterations in which those storage locations are accessed.
- A list item cannot appear in more than one nontemporal clause.
- IF clause to OpenMP SIMD allows for conditional vectorization based on runtime evaluation of a logical expression.

OTHER FEATURES

New Compiler Options

- assume [no]old_inquire_recl - Determines the value of the RECL= specifier on an INQUIRE statement for an unconnected unit or a unit connected for stream access.
- check [no]udio_iostat - Determines whether standard conformance checking occurs when user defined derived type input/output procedures are executed.
- warn [no]externals - Determines whether warnings occur for any dummy procedures or procedure calls that have no explicit interface or have not been declared EXTERNAL.
- assume [no]old_ldout_zero - Determines the format of a floating-point zero produced by list-directed output. old_ldout_zero uses exponential format, no_old_ldout_zero uses fractional format

Difference in behavior 19.1 vs 19.0

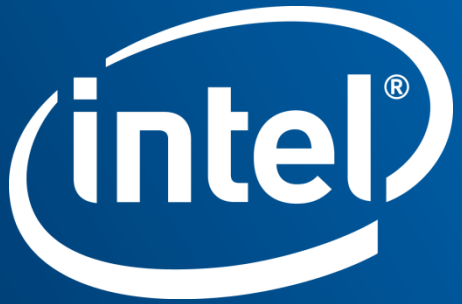
- PRIVATE or SEQUENCE statement to only appear after the declaration of any type parameters in a derived type declaration.
- The INQUIRE statement now uses realpath'/'GetFullPathNameA' and uses the resulting canonicalized file-paths if the calls succeeds.

  An example of the change is:

  Open file as "bar/foo".

  Before: Inquire whether "./bar/foo", "bar//foo" or "bar/../bar/foo" is open – it is not.

  After: Inquire whether "./bar/foo", "bar//foo" or "bar/../bar/foo" is open – it is.
- 19.1 complies with Interp 18/007 and makes C_F_PROCPOINTER IMPURE.


intel®

Legal Disclaimer & Optimization Notice

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804