# What's New in Intel® Fortran? Intel Fortran Composer 2011 XE Webinar

December 14th, 2010

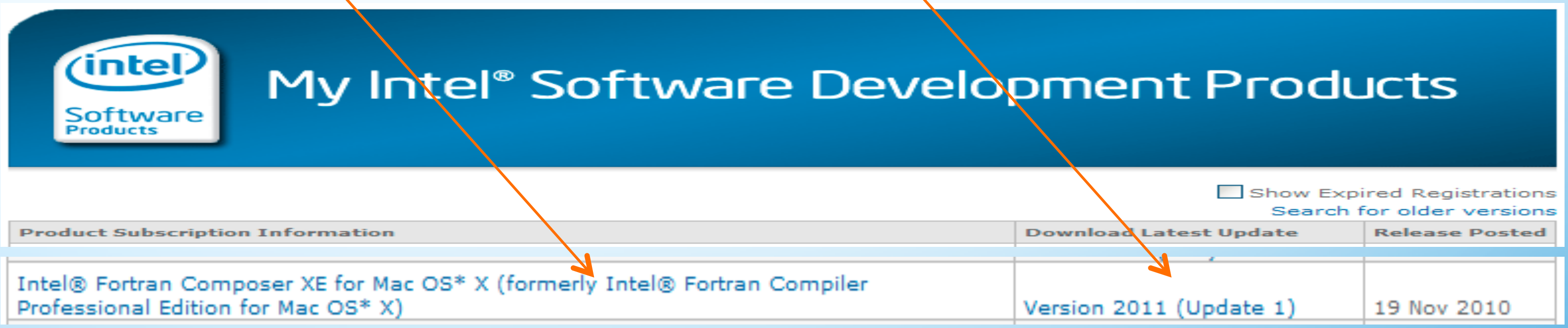
Steve Lionel, Ron Green

# Agenda

- What’s in a name? Changes in naming
- New utilities, new options, new installation dirs
- What’s New in Fortran standards features?
- A simple and quick look at using the Intel Fortran compiler’s Coarray Fortran (CAF) feature
- Question and Answer session

# An Obvious Change: Naming

- New names:
  - Intel® Visual Fortran Composer XE 2011 (Windows*)
  - Intel® Fortran Composer XE 2011 (Linux* and Mac OS* X)
- Replaces older "…Compiler Pro " naming
- Composer XE 2011 is our next major release ( 12.0 )
- Registration Center: Version numbers not as prominent – using "Update x"



My Intel® Software Development Products

□ Show Expired Registrations
Search for older versions

| Product Subscription Information | Download Latest Update | Release Posted |
|---|---|---|
| Intel® Fortran Composer XE for Mac OS* X (formerly Intel® Fortran Compiler Professional Edition for Mac OS* X) | Version 2011 (Update 1) | 19 Nov 2010 |

# Licensing

- Existing, CURRENT licenses for Compiler Pro will work for Intel Fortran Composer products
- Registration Center will offer to “upgrade” your existing licenses – this is FREE
- Intel C/C++ licenses MUST BE UPGRADED for Intel C/C++ Composer XE 2011 – upgrade is free
- All renewals will get upgraded licenses to Composer XE 2011 products

- Optional: Compiler customers can move to new “Intel® Parallel Studio XE 2011” products. These contain the Intel Fortran compiler, C++ compiler, libraries, AND new checking and performance tools

# Poll Question #1

# What’s ~~New~~ OLD in Intel Fortran: listing file

Back by popular demand: DEC Fortran style cross referenced listing file:

-list[=*filename*]  or   /list[:*filename*]

where *filename* is the name of the output file

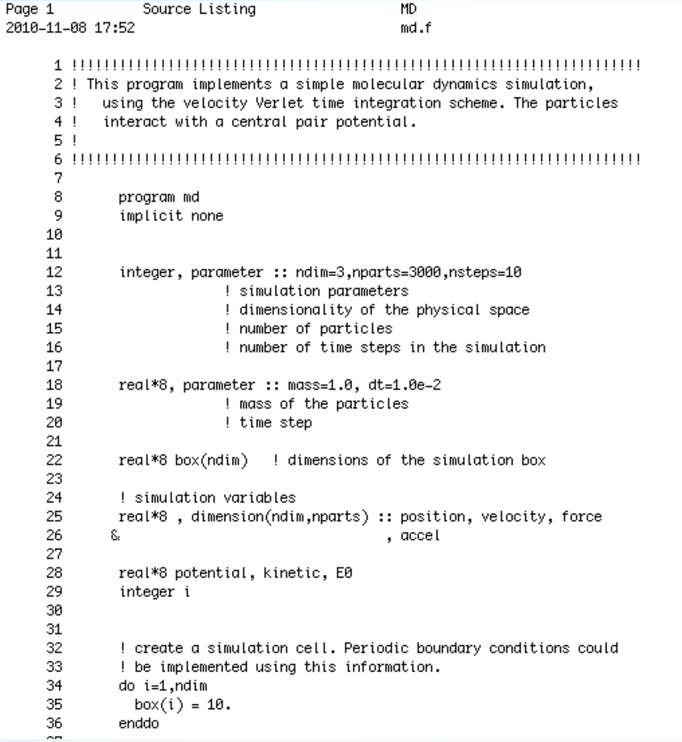
- if filename is not specified, the listing is saved in the name of the source file with extension .lst

default is –no-list  or /list-

The listing contains the following:

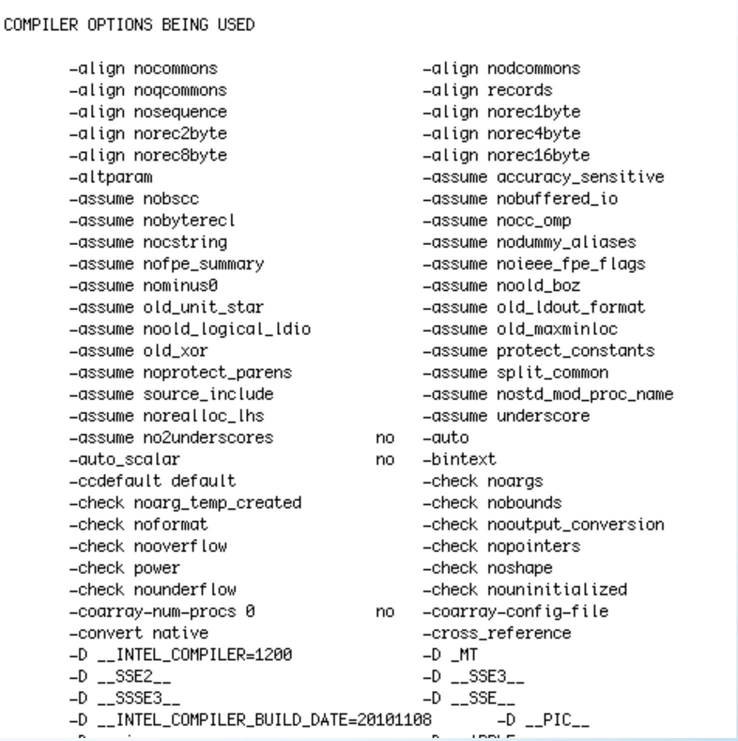
- The contents of files, including contents with INCLUDE statements, with line numbers
- A symbol list with a line number cross-reference for each
- A list of compiler options used for the current compilation
- list-line-len and list-page-len options for further control

# What’s ~~New~~ OLD in Intel Fortran: listing file



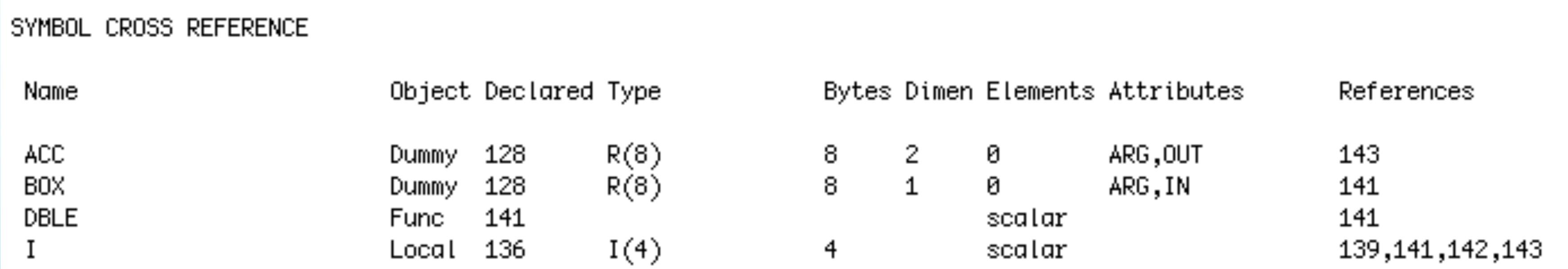
```
Page 1             Source Listing                    MD
2010-11-08 17:52                                     md.f

     1 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
     2 ! This program implements a simple molecular dynamics simulation,
     3 !   using the velocity Verlet time integration scheme. The particles
     4 !   interact with a central pair potential.
     5 !
     6 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
     7
     8       program md
     9       implicit none
    10
    11
    12       integer, parameter :: ndim=3,nparts=3000,nsteps=10
    13                    ! simulation parameters
    14                    ! dimensionality of the physical space
    15                    ! number of particles
    16                    ! number of time steps in the simulation
    17
    18       real*8, parameter :: mass=1.0, dt=1.0e-2
    19                    ! mass of the particles
    20                    ! time step
    21
    22       real*8 box(ndim)   ! dimensions of the simulation box
    23
    24       ! simulation variables
    25       real*8 , dimension(ndim,nparts) :: position, velocity, force
    26      &                                 , accel
    27
    28       real*8 potential, kinetic, E0
    29       integer i
    30
    31
    32       ! create a simulation cell. Periodic boundary conditions could
    33       ! be implemented using this information.
    34       do i=1,ndim
    35         box(i) = 10.
    36       enddo
```



```
COMPILER OPTIONS BEING USED

       -align nocommons                      -align nodcommons
       -align noqcommons                     -align records
       -align nosequence                     -align norec1byte
       -align norec2byte                     -align norec4byte
       -align norec8byte                     -align norec16byte
       -altparam                             -assume accuracy_sensitive
       -assume nobscc                        -assume nobuffered_io
       -assume nobyterecl                    -assume nocc_omp
       -assume nocstring                     -assume nodummy_aliases
       -assume nofpe_summary                 -assume noieee_fpe_flags
       -assume nominus0                      -assume noold_boz
       -assume old_unit_star                 -assume old_ldout_format
       -assume noold_logical_ldio            -assume old_maxminloc
       -assume old_xor                       -assume protect_constants
       -assume noprotect_parens              -assume split_common
       -assume source_include                -assume nostd_mod_proc_name
       -assume norealloc_lhs                 -assume underscore
       -assume no2underscores             no -auto
       -auto_scalar                       no -bintext
       -ccdefault default                    -check noargs
       -check noarg_temp_created             -check nobounds
       -check noformat                       -check nooutput_conversion
       -check nooverflow                     -check nopointers
       -check power                          -check noshape
       -check nounderflow                    -check nouninitialized
       -coarray-num-procs 0               no -coarray-config-file
       -convert native                       -cross_reference
       -D __INTEL_COMPILER=1200              -D _MT
       -D __SSE2__                           -D __SSE3__
       -D __SSSE3__                          -D __SSE__
       -D __INTEL_COMPILER_BUILD_DATE=20101108       -D __PIC__
```



```
SYMBOL CROSS REFERENCE

 Name                    Object Declared Type          Bytes Dimen Elements Attributes     References

 ACC                     Dummy  128      R(8)          8     2     0        ARG,OUT        143
 BOX                     Dummy  128      R(8)          8     1     0        ARG,IN         141
 DBLE                    Func   141                                scalar                  141
 I                       Local  136      I(4)          4           scalar                  139,141,142,143
```

# GAP – Guided Automatic Parallelization

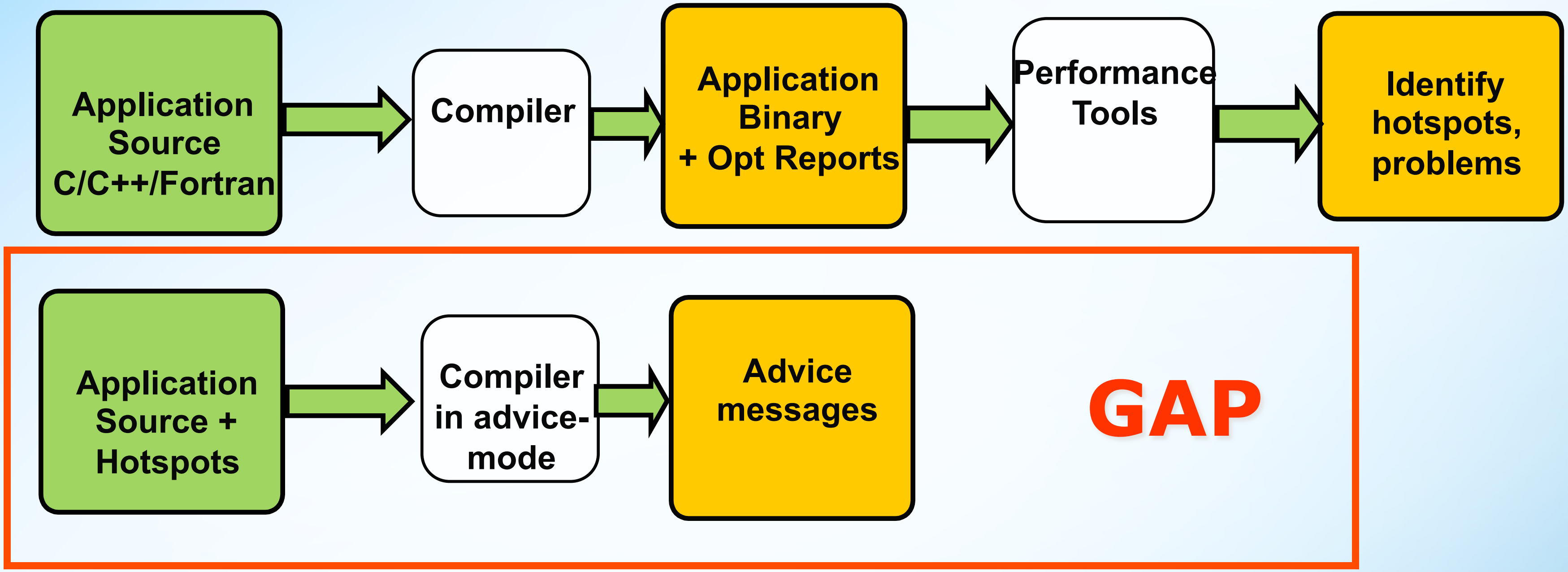
## Key design ideas:

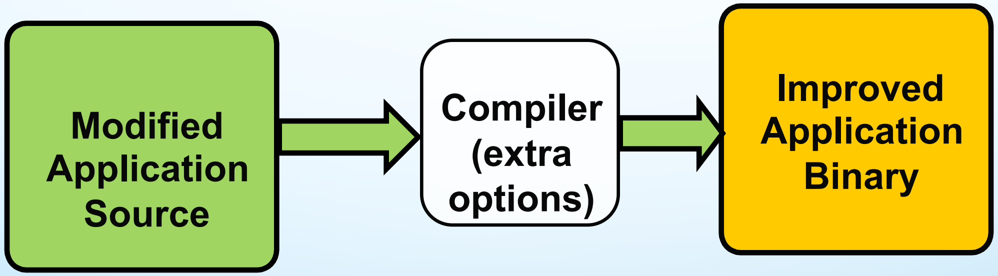
- Use compiler to help detect what is blocking optimizations – in particular vectorization, parallelization and data transformations – gives advice on how to change code, add directives, add compiler options
  - Extend diagnostic message for failed vectorization and parallelization by specific hints to fix problem
- Not a separate tool, part of the compiler

## It is not:

- Automatic vectorizer or parallelizer
  - in fact, no code is generated to accelerate analysis
- GAP does not ask the programmer to change algorithms, transformation ordering or internal heuristics of compiler
  - It is restricted to changes applied to the program to be compiled

# Workflow with Compiler as a Tool


Application Source C/C++/Fortran
Compiler
Application Binary + Opt Reports
Performance Tools
Identify hotspots, problems
Application Source + Hotspots
Compiler in advice-mode
Advice messages
GAP



***Simplifies programmer effort in application tuning***

# GAP – How it Works (linux)

## Selection of most Relevant Switches

**Multiple compiler switches to activate and fine-tune guidance analysis**

- **Activate messages individually for vectorization, parallelization, data transformations or all three**

```
-guide[=level]
-guide-vec[=level]
-guide-par[=level]
-guide-data-trans[=level]
```

**Optional argument level=1,2,3,4 controls extend of analysis**

- **Control the source code part for which analysis is done**

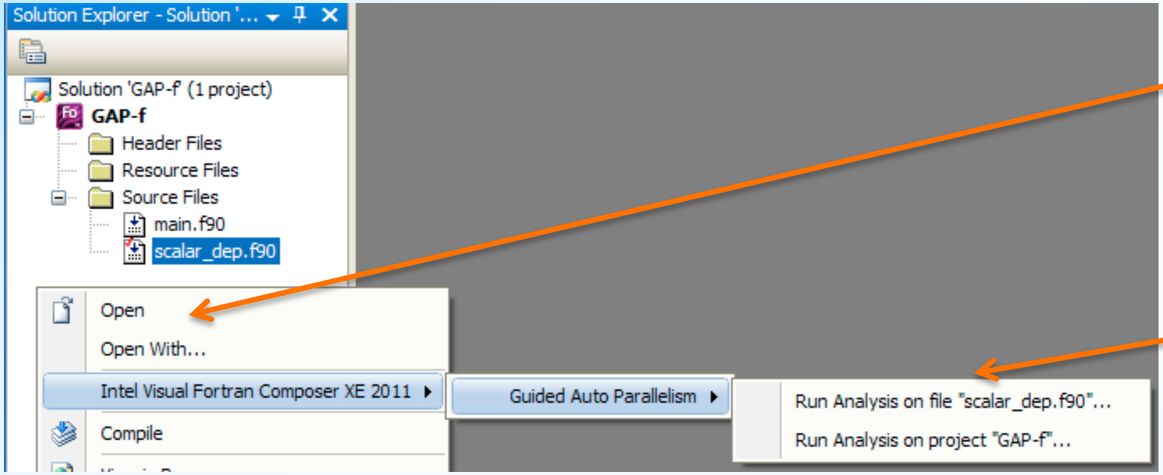
```
-guide-opts=<arg>
Samples:
-guide-opts="bar.f90,'module_1::func_solve`“
```

- **Control where the message are going**

```
-guide-file=<file_name>
```

# GAP – How it Works (Windows)

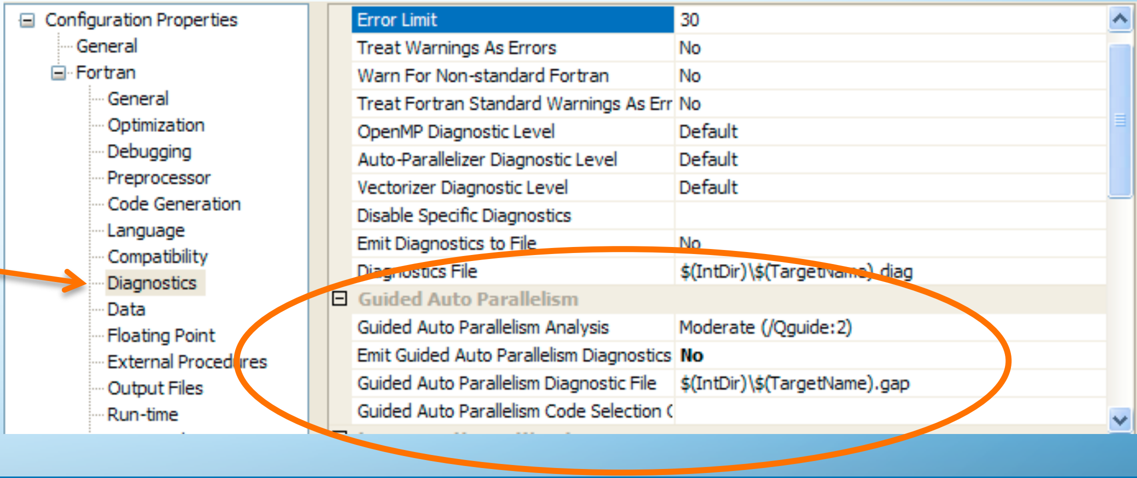
- Windows – right-click or Project Properties
  - GAP analysis appears in Output window


Solution Explorer - Solution '...
Solution 'GAP-f' (1 project)
GAP-f
Header Files
Resource Files
Source Files
main.f90
scalar_dep.f90
Open
Open With...
Intel Visual Fortran Composer XE 2011
Compile
Guided Auto Parallelism
Run Analysis on file "scalar_dep.f90"...
Run Analysis on project "GAP-f"...

Right-click file or project, Fortran Composer pop-up

Choose single-file or whole project

OR you can use the Project Properties, Fortran, Diagnostics property page



| | |
|---|---|
| Error Limit | 30 |
| Treat Warnings As Errors | No |
| Warn For Non-standard Fortran | No |
| Treat Fortran Standard Warnings As Err | No |
| OpenMP Diagnostic Level | Default |
| Auto-Parallelizer Diagnostic Level | Default |
| Vectorizer Diagnostic Level | Default |
| Disable Specific Diagnostics | |
| Emit Diagnostics to File | No |
| Diagnostics File | $(IntDir)\$(TargetName).diag |
| **Guided Auto Parallelism** | |
| Guided Auto Parallelism Analysis | Moderate (/Qguide:2) |
| Emit Guided Auto Parallelism Diagnostics | **No** |
| Guided Auto Parallelism Diagnostic File | $(IntDir)\$(TargetName).gap |
| Guided Auto Parallelism Code Selection ( | |

# GAP Sample Messages

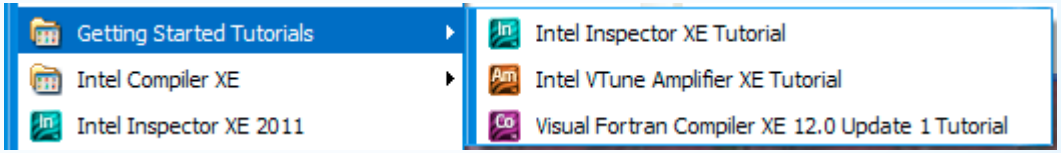
```
 GAP REPORT LOG OPENED ON Thu May 20 15:22:14 2010

C:\scalar_dep.f90(66): remark #30525: (PAR) If the trip count of the
  loop at line 66 is greater than 36, then use "!dir$ loop count min
  (36)" to parallelize this loop.
  [VERIFY]  Make sure that the loop has a minimum of 36 iterations.


C:\scalar_dep.f90(66): remark #30515: (VECT) Loop at line 66 cannot
  be vectorized due to
     conditional assignment(s) into the following variable(s): T.
  This loop will be vectorized
     if the variable(s) become unconditionally initialized at the top
  of every iteration.
      [VERIFY] Make sure that the value(s) of the variable(s) read in
  any iteration of the loop
     must have been written earlier in the same iteration.
```

# Also New: Tutorials and Samples, New Directory Structure

- Greatly enhanced and updated samples included in Composer XE
- Tutorials to help introduce new features ( GAP, for example )


Getting Started Tutorials
Intel Compiler XE
Intel Inspector XE 2011
Intel Inspector XE Tutorial
Intel VTune Amplifier XE Tutorial
Visual Fortran Compiler XE 12.0 Update 1 Tutorial

- New directory paths on Linux, Windows, Mac OS X
- Better integration of libraries
- Linux and Mac OS X: side-by-side installation of multiple versions BUT symbolic links give a 'default' path to tools that is not version dependent

# Also New Features in Both Intel Fortran and Intel C/C++

- Enhanced vectorization
  - Loops with mixed data types, conditionals
  - Support AVX instruction set (-[a]xAVX  or /Q[a]xAVX)
- SIMD directives
  - For example, require compiler to vectorize a loop
- Math library options (-fimf-precision /Qimf-precision)
  - High/low accuracy options   (tradeoff against performance)
  - Require consistent results on all processor types
- Matrix multiply idiom recognition   (-opt-matmul)
  - Replace by high performance library call

# Poll Question #2

# Fortran Language Features – What's New?

# What’s New in Intel Fortran

## Fortran 2003 implementation mostly complete

- Added in 12.0 ( not in 11.1 )
  - Complete type-bound procedures (GENERIC, OPERATOR,..)
  - FINAL procedures
- Remaining major features of F2003 not implemented:
  - User-defined derived type I/O
  - Parameterized derived types

## Fortran 2008 features

- Coarrays
- DO CONCURRENT
- CONTIGUOUS
- I/O enhancements
- New constants in ISO_FORTRAN_ENV
- New intrinsic functions
- Increase maximum rank from 7 to 31
  - F2008 requires only 15

# Fortran 2003 Pointer Bounds Specification and Remapping List on Pointer Assignment

- Fortran 2003 features
- Pointer assignment for arrays extended to allow specification of lower bounds:

```
real :: myarray(1:100,1:100)
real, pointer :: ptr(:,:)
ptr(0:,0:) => myarray
```

- Remapping of a rank-one array
  - ptr(1:n,1:n) => 1D_Array(1:n*n)

# Fortran 2003 FINALizers

- A derived type with ‘FINAL’ subroutines bound to it
- The FINAL subroutine(s) perform ‘clean-up’ when the object ceases to exist

```
module M
  type mytype
    : !declaration of mytype components
  contains
    FINAL :: mycleanup
  end type mytype
contains
  subroutine mycleanup(x)
  type(mytype) :: x
   !...deallocate data in object X
  end subroutine mycleanup
end module M
```

Note: Fortran 2003 does not have the equivalent to C++ constructor functions

# F2008 DO CONCURRENT

## A new Parallel Loop Construct

- Syntax uses elements of Fortran 90 FORALL

```
DO [,] CONCURRENT <forall-header>
```

- Semantically there is a key difference to FORALL however :
  - No dependencies between the iterations of the loop body are permitted ( no “loop carried dependencies”)
- The semantics of DO CONCURRENT make it easier to parallelize
- Use option –parallel (/Qparallel) to get parallelization
- No requirement or guarantees that the loop will be parallelized
- Our implementation will execute the iterations in parallel using OpenMP*

# F2008 DO CONCURRENT

Example:

```
DO CONCURRENT (i=1:m)
    a(k+i) = a(k+i) + factor*a(l+i)
END DO
```

Different from FORALL, using DO CONCURRENT, the programmer guarantees, that the values of `m`, `k` and `l` will never cause `a(l+i)` to reference an element of the array defined on the LHS

in other words: the array sections `a(l+1:l+m)` and `a(k+1:k+m)` do not overlap

This allows compiler to generate very efficient parallel code.

# Fortran 2008 CONTIGUOUS Attribute

- An array attribute that tells the compiler that the data occupies a contiguous block
  - Allows compiler to make optimizations
  - Pointers and assumed-shaped arrays: useful to remove ambiguity when the compiler cannot determine if the object is contiguous or non-contiguous

```
real, pointer, contiguous :: ptr(:)
real, contiguous :: arrayarg(:,:)
```

- The POINTER target must be contiguous
- The actual argument corresponding to the assumed-shape array must be contiguous
- F08 intrinsic, logical return: `is_contiguous()`

```
IF ( is_contiguous(thisarray) ) THEN
   ptr => thisarray
```

# Fortran 2008 MOLD keyword for ALLOCATE

- ALLOCATE statement can give a polymorphic variable the type and shape of another object without copying the other object's values.

```
allocate ( polymorphvar, mold=srcvar )
```

Variable `polymorphvar` is allocated with the type and shape of `srcvar.` `polymorphvar` does not receive the values in the components of srcvar.

Also,  SOURCE= with polymorphic source not yet supported in ALLOCATE

# Fortran 2008 IO Additions

- NEWUNIT=<integer> keyword in OPEN finds a unit number that is not being used.  Simplifies bookkeeping of unit numbers

```
OPEN( NEWUNIT=iun, file=‘foo’, … )
```

!assigns an unused number to `iun`

- G0 and G0.d edit descriptors.  Can be used with multiple data types:
  - real or complex: acts like `esw.de`
    - e format with values w, d and e chosen by the processor
  - Integers, acts like `I0`
  - Logicals, acts like `L1`
  - Character, acts like `A`

# Fortran 2008 IO Additions
# Unlimited format item repeat count

- Asterisk preceding a list of edit descriptors
- Repeats the list indefinitely

```
real :: myarray(50) = 42.0
write(42, ‘( “myarray =“, *( g0, :, “,”))’) myarray
```

```
myarray=42.00000,42.00000,42.00000,42.00000,42.00000,4
  2.00000,42.00000,42.00000,42.00000,42.00000,42.00000
  ,42.00000,42.00000,…etc…
```

- This example writes 1 record, comma separated values
- This can be used with G0 edit descriptor to write output with various data types present

# F2008 Intrinsics

- Bessel, first kind
- Bessel, second kind
- Error functions
- GAMMA
- Euclidean distance
- Bit-wise comparisons
- Integer bit-wise shifts
- Bit masks
- Merge bits with mask
- Population count: return the number of 1 bits
- Parity of population
- Bit-wise exclusive-or (XOR) on array elements
- Bitwise reductions on array elements using AND or OR
- Number of leading or trailing 0 bits
- Storage size in bits

# Fortran 2008 Additions to ISO_FORTRAN_ENV

- CHARACTER_KINDS
  - Default integer array with the kind values supported for variables of type character
  - Size equals the number of kinds supported

- INTEGER_KINDS, REAL_KINDS
  - Similar to CHARACTER_KINDS, arrays of kind values for INTEGER and REAL data types

# Fortran 2008 Additions to ISO_FORTRAN_ENV

- INT8, INT16, INT32, and INT64
  - Default integer scalars, kind values for integers of storage size 8, 16, 32, and 64 bits
  - If there is no such type, -2 is return if there is a type of larger storage size or -1 otherwise

- REAL32, REAL64, and REAL128
  - Default integer scalars, the kind values for reals of storage size 32, 64, and 128 bits
  - If there is no such type, -2 is return if there is a type of larger storage size or -1 otherwise

# Poll Question #3

# Coarray Fortran Fundamentals

- Simple extension to Fortran to make Fortran into a robust and efficient parallel programming language
- Single-Program, Multiple-Data programming model
  - Single program is replicated a fixed number of times
  - Each program instance has it's own set of data objects – called an "IMAGE"
  - Each image executes asynchronously
  - Extensions to normal Fortran array syntax to allow images to reference data in other image(s)
- Part of the Fortran 2008 standard
- Shared-memory in Fortran Composer XE for Windows* and Linux*
- Distributed-memory supported in Linux only, Intel® Cluster Tools product line

# Compilation

- `ifort –coarray      !Linux*`
- `ifort /Qcoarray     !Windows*`

  along with other options. Enables compiling for CAF.  By default, executable will use as many cores (real and hyperthreaded) as are available.

```
ifort –coarray -coarray-num-images=x
ifort /Qcoarray /Qcoarray-num-images=x
```

along with other options.  Sets number of images to “x”.

# Running (linux)

- Simple hello world:

```
program hello_image
   write(*,*) "Hello from image ", this_image(), &
     "out of ", num_images()," total images“
end program hello_image
```

```
ifort –coarray –o hello_image hello_image.f90
./hello_image
 Hello from image           1 out of            4  total images
 Hello from image           4 out of            4  total images
 Hello from image           2 out of            4  total images
 Hello from image           3 out of            4  total images
```

# Controlling the Number of Images, env var: FOR_COARRAY_NUM_IMAGES

- Environment variable can set number of images
- Environment variable overrides –coarray-num-images compiler option

```
Linux host>  export FOR_COARRAY_NUM_IMAGES=2
./hello_image
Window host> set FOR_COARRAY_NUM_IMAGES=2
hello_image.exe
 Hello from image 1 out of 2  total images
 Hello from image 2 out of 2  total images
```

# CAF Fundamentals:  Determining Number of Images, num_images()

- Intrinsic function num_images() returns an integer result, the total number of images in the CAF program:

```
$> cat hello_num_images.f90
program hello_num_images
  write(*,*) "Hello there are ", num_images()," total images"
end program hello_num_images

> ifort -coarray -coarray-num-procs=4 hello_num_images.f90
$> ./a.out
 Hello there are            4  total images
 Hello there are            4  total images
 Hello there are            4  total images
 Hello there are            4  total images
```

# Coarray Fundamentals:  this_image()

- Images have a logical ordering from 1 to N
- Integer function this_image() without an argument returns unique logical ordering from 1 to N
  - More complex image mappings possible: 2D, 3D, etc with arguments (topic discussed later)

```
$> cat hello_this.f90
program hello_this_image
  write(*,*) "Hello from image ", this_image()
end program hello_this_image
$> ifort -coarray -coarray-num-procs=4 hello_this.f90
$> ./a.out
Hello from image            1
Hello from image            3
Hello from image            2
Hello from image            4
```

- Remember, the images are inherently asynchronous

# CAF Fundamentals – Codimensions Declaration, A Simple Scalar Example

- A variable can be declared with a CODIMENSION

```
real, codimension[*] :: x
real :: y[*]
```

- X, Y are real scalar variables, codimension can be used to reference copies of X & Y on remote images
- Similar to assumed size array syntax, "[*]" means as many copies as there are images, one copy per image
  - "*" can ONLY be used on last codimension for the object
    - Ex:  [*,2] is illegal, but [2,*] is valid: means a 2D ordering of images. 20 images would have object with [2,10] codimension. 30 images would have object with [2,15] codimensions

# CAF Fundamentals – Codimensions Declaration, Coarray Examples

```
real ::  myarray(100)[*]
```

- A program with N images will have N copies of `myarray`, 1 per image
- Extent of `myarray` is 100, lower bound 1, upper bound 100 on each image
- Coarrays can have normal F08 attributes: ALLOCATABLE, POINTER, have multiple dimensions, be part of a derived type, etc.

```
real, allocatable :: a(:)[*], b(:)[*]

allocate( b(100)[*], b(100)[0:*] )

   **note that the brackets and cobounds are
  needed
```

# Some Advice Before Some Examples

- CAF behavior rule of thumb: when questioning the behavior of CAF ask “what would the Fortran semantics imply here” – follow Fortran rules
- The “[]” codimension syntax is a visual clue to where communication to remote images is performed (implies OVERHEAD, implies possible performance drops)
- There are many restrictions to where coarrays can be used: Simply put: any attempt to alias a coarray with a non-coarray object are prohibited:
  - Pointers that are not coarrays
  - Non-coarray dummy args passed coarrays
  - Passing coarray object to C or another language
  - COMMON, EQUIVALENCE, etc

# CAF Fundamentals – Codimensions Reference, A Simple Scalar Example

- Without specifying codimension, usual Fortran semantics:  X is the local image instance for X

```
x = 42.0   !refers to the local image’s variable instance
```

If you specify the codimension, it references a specific image’s copy of the variable:

```
x[3] = 42.0  !sets X on image 3 to 42.0
x = x[1]    !local X gets value of X from image 1
X[i] = x[j]  !image I’s value of X set to value from image J
```

- Objects referenced with square brackets “coindexed object”

# CAF Fundamentals - Codimensions

- Codimensions follow similar syntax and semantics as Fortran 90 array dimension syntax

[1:N]   codimensions 1 to N

[ -1:99, 0:100, -100:-1 ]  upper and lower bounds need not start at 1, can be negative, etc.

- Restriction:  Total number of dimensions PLUS codimensions <= 15

- Similar to array syntax, objects can have:
  - corank, cobounds, coextents

# CAF Fundamentals - Codimensions

- Intrinsic functions `lcobound()` and `ucobound()` return lower and upper cobounds
- UCOBOUND( coarray [,DIM, KIND] ) !upper
- LCOBOUND( coarray [,DIM, KIND] ) !lower

```
real, allocatable :: A[:,:,:]
integer :: lcb(3), ucb(3)
allocate( A[3:4,-1:6,*] )  !..assume 30 images

lcb = lcobound(A)
!...if images=30, lcb = (/ 3, -1, 1 /)
ucb = ucobound(A)
!...if images=30, ucb = (/ 4, 6, 2 /)
lcobound(A,DIM=2) == -1
```

# CAF Fundamentals - Codimensions

- Mapping of objects with codimensions: 2D

```
real, codimension[2,*] :: x
```

- Mapping of X if program run with 6 images:

| | | |
|---|---|---|
| Image 1<br>X[1,1] | Image 3<br>X[1,2] | Image 5<br>X[1,3] |
| Image 2<br>X[2,1] | Image 4<br>X[2,2] | Image 6<br>X[2,3] |

Mapping of X if program run with 9 images

| | | | | |
|---|---|---|---|---|
| Image 1<br>X[1,1] | Image 3<br>X[1,2] | Image 5<br>X[1,3] | Image 7<br>X[1,4] | Image 9<br>X[1,5] |
| Image 2<br>X[2,1] | Image 4<br>X[2,2] | Image 6<br>X[2,3] | Image 8<br>X[2,4] | |

# CAF Fundamentals: Global Barrier Synchronization

- SYNC ALL statement global barrier: requires all images to join the synchronization point
- sync images( ) allows synchronization with a subset of images. The image set is an integer scalar holding an image index, an integer array of rank 1 holding distinct image indices, or an asterisk to indicate all images,
- Critical sections can be created, bounded by **`CRITICAL ; END CRITICAL`**
- SYNC MEMORY ensures any changed data that is held in temporary storage ( cache, registers ) or in transit between images is made visible to the other image

# Additional Synchronization

- LOCK and UNLOCK statements provide fine-grained control
- ERROR STOP stops execution on all images immediately with error code
- Implicit global synchronization at ALLOCATE, DEALLOCATE of coarrays
  - When coarray is allocated on one image, wait until all images allocate their copy.  Otherwise, one image could attempt to access unallocated coarray data on another image
  - Similar on DEALLOCATE:  Wait to remove the coarray data until all images synch and deallocate: otherwise, other images could try to access deallocated coarray data

# CAF Fundamentals - Input/Output

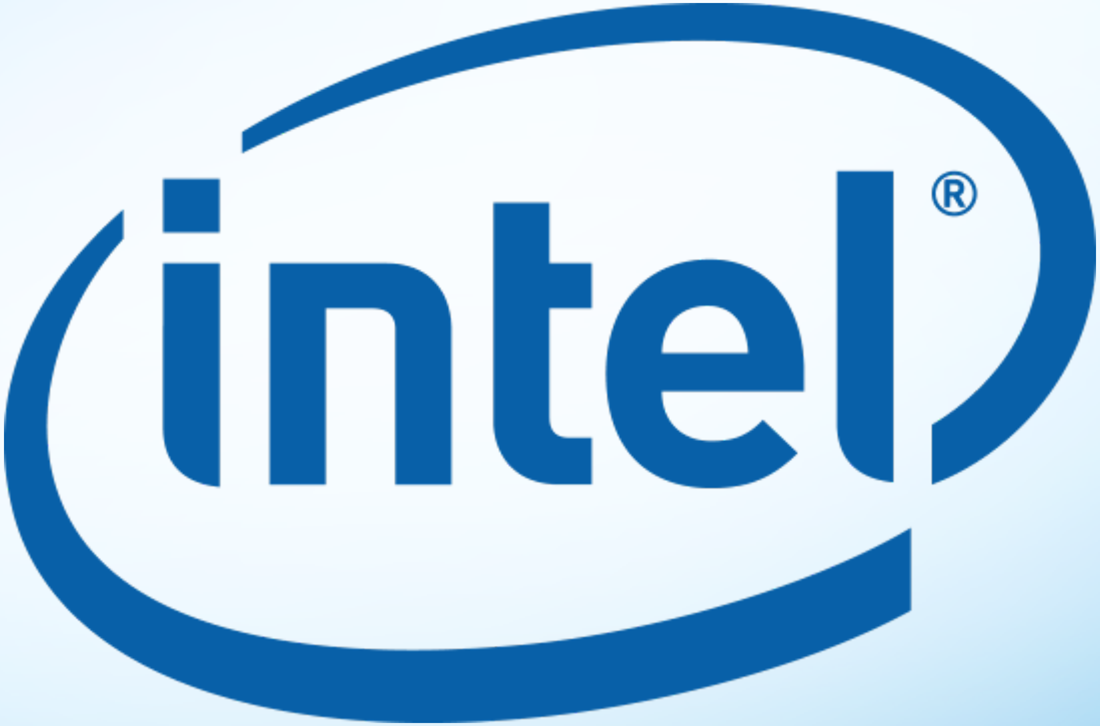
- Each image has its own set of connected units
- Default output unit is preconnected on all images
  - Assumption is that processor will merge the streams
- Default input unit is preconnected on image 1 only

# Further Reading

- Coarrays in the next Fortran Standard
  - ftp://ftp.nag.co.uk/sc22wg5/N1801-N1850/N1824.pdf
- The New Features of Fortran 2008
  - ftp://ftp.nag.co.uk/sc22wg5/N1801-N1850/N1828.pdf
- Fortran 2008 Standard (current draft)
  - http://j3-fortran.org/doc/standing/links/007.pdf

# Poll Question #4

# Questions and Answers Session



intel®

# Legal Disclaimer

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference www.intel.com/software/products.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

http://intel.com/software/products

# Optimization Notice

## Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel® Compiler User and Reference Guides" under "Compiler Options." Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

Intel® compilers, associated libraries and associated development tools 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 Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® 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.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20101101