INTEL® PARALLEL STUDIO XE 2016 COMPOSER EDITION

What's New

August 2015
Significant New Features

New Directory Structure

OpenMP* 4.1 Extensions

C11 Standard Support

More C++14 Standard Support

Fortran 2008 Submodules and IMPURE ELEMENTAL

Further C Interoperability from Fortran 2015

Enhanced Uninitialized Variable Run-time Detection in Fortran
NEW FEATURES COMMON TO BOTH C++ AND FORTRAN

What's in Intel® Compiler 16.0
New Features Common to both C++ and Fortran

Operating System and IDE Support
New Directory Structure Layout
Licensing Changes
OpenMP* 4.1 Extensions
Improvements in Vectorization
Loop Blocking Pragma/Directive (BLOCK_LOOP)
OpenMP* 4.1 Extensions

Support for Features in OpenMP* 4.1 Technical Report 3

- Non-structured data allocation
  - `omp target [enter | exit ] data`

- Asynchronous offload
  - `nowait clause on omp task`

- Dependence (signal)
  - `depend clause on omp task`

- Map clause extensions
  - Modifiers `always and delete`

Available for C/C++ and Fortran
Improvements in Vectorization
Intel® Cilk™ Plus and OpenMP* 4.0

**simdlen** (i.e. `vectorlength`) and **safelen** for loops
- Usable with `#pragma simd` (Intel Cilk™ Plus) and `omp simd` (OpenMP*)

Array reductions
- Fortran only (available in Beta update)

User-defined reductions
- Supported for parallel in C/C++ for POD types. No support for Fortran, SIMD, or non-POD types (C++)

**omp-simd collapse** (N) clause
- Available in a Beta update

FP-model honoring for simd loops
Improvements in Vectorization

Ordered blocks in SIMD context

- `ordered with simd` specifies structured block simd loop or SIMD function that executes in order of loop iterations or sequence of SIMD function(s) calls.

```
#pragma omp ordered [simd]
structured code block
-- OR --
#pragma simdoff
structured code block
```

Adjacent gathers optimization

- Replace series of gathers with series of vector loads and sequence of permutes.

```
!$omp ordered [simd]
structured code block
!$omp end ordered
```
Improvements in Vectorization
Other internal improvements

Alignment analysis

- Information propagation improved
- \_assume\_aligned() fixed

Memory reference analysis

- Resolved all “subscript/dereference too complex” cases
- More convoluted cases optimized to use vector loads

Improvements for AVX512

- conflict/compress/expand idioms improved
Improvements in vectorization messages (Continued from 15.0)

- Removal of many vectorization failure messages
  - E.g. Subscript too complex, unsupported data type, loop structure

- Clarity on messages
  - Reference to function names, data variables, control structure
  - E.g. Function `<name>` was vectorized, not vectorized because of break statement

- Suggested actions for next steps
  - Try an option, pragma, clause to override current behavior
  - E.g. Use `fp_model=fast`, use `veclen` clause
Improvements in Vectorization
Other internal improvements

Improved optimization reports

Uniformity analysis and handling

- Scalar control flow and scalar computations
- Benefits to memory reference analysis

Local target control supported

- Vectorization properly targeted, e.g.

```c
#include <immintrin.h>

void foo1(float *y, float *a, float *b, int n) {
    if (_may_i_use_cpu_feature(_FEATURE_AVX2)) {
        for (int i=0; i < n; ++i)
            y[i] = a[i]*y[i] + b[i];  // use FMA
    } else {
        for (int i=0; i < n; ++i)
            y[i] = a[i]*y[i] + b[i];
    }
}
```
Intel® Advisor XE - Vectorization Advisor

Data Driven Vectorization Design

Have you:

- Recompiled with AVX2, but seen little benefit?
- Wondered where to start adding vectorization?
- Recoded intrinsics for each new architecture?
- Struggled with cryptic compiler vectorization messages?

Breakthrough for vectorization design

- What vectorization will pay off the most?
- What is blocking vectorization and why?
- Are my loops vector friendly?
- Will reorganizing data increase performance?
- Is it safe to just use pragma simd?

More Performance
Fewer Machine Dependencies
Loop Blocking Pragma/Directive

- **Syntax:**
  
  - **C++**
    
    ```
    #pragma block_loop [clause[,clause]...]  
    #pragma noblock_loop
    ```

  - **Fortran**
    
    ```
    !DIR$ BLOCK_LOOP [clause [[,]clause]...]  
    !DIR$ NOBLOCK_LOOP
    ```

- **BLOCK_LOOP** enables greater control over optimizations on specific DO/for loop inside a nested loop

- Uses loop blocking technique to separate large iteration counted loops into smaller iteration groups

- Smaller groups can increase efficiency of cache space use and augment performance

- Works seamlessly with other directives including SIMD
NEW FEATURES IN INTEL® C++ COMPILER 16.0
Intel® C++ Compiler 16.0 New Features

C11 and C++14 Standards Support

GNU* Compatibility

Microsoft* Compatibility

Other New Features and Enhancements

- Compile-time improvements
- SIMD Operator support
- Honoring Parentheses
- Intel® Cilk Plus™ Combined Parallel/SIMD loops
Intel® Cilk™ Plus

- New Combined Parallel/SIMD Loops

```
_Cilk_for_Simd (int i = 0; i < N; ++i)
    // Do something
```

or

```
#pragma simd
_Cilk_for (int i = 0; i < N; ++i)
    // Do something
```

- Combined loop yields both parallelism (using threads) and vectorization

- Behaves approximately like this pair of nested loops

```
_Cilk_for (int i_1 = 0; i_1 < N; i_1 += M)
    for_Simd (int i = i_1; i < i_1 + M; ++i)
        // Do something
```

- The chunk size, M, is determined by the compiler and runtime
NEW FEATURES IN
INTEL® FORTRAN COMPILER 16.0
New and Changed Features
Intel® Fortran Compiler 16.0

Submodules from Fortran 2008

IMPURE ELEMENTAL from Fortran 2008

Further C Interoperability from Fortran 2015

Other New Features

- ASYNCHRONOUS communication
- \(-\text{fpp-name} \) option
- VS2013 Shell
- Uninitialized Variable Run-time Detection
Submodules (F2008) – The Problem

module bigmod
... contains
subroutine sub1
...<implementation of sub1>

function func2
...<implementation of func2>

subroutine sub47
...<implementation of sub47>
...
end module bigmod

! Source source1.f90
use bigmod
... Call sub1

! Source source2.f90
use bigmod
...
  x = func2(...)

! Source source47.f90
use bigmod
...
call sub47
Submodules (F2008) – The Solution

Changes in the submodule do not force recompilation of uses of the module – as long as the interface does not change
IMPURE ELEMENTAL (F2008)

In Fortran 2003, ELEMENTAL procedures are PURE

- No I/O, no side-effects, can call only other PURE procedures

New IMPURE prefix allows non-PURE elemental procedures

- Can do I/O, call RANDOM NUMBER, etc.
Uninitialized Variable Run-time Detection

Uninitialized variable checking using [Q]init option is extended to local, automatic, and allocated variables of intrinsic numeric type

Example:

```fortran
4  real, allocatable, dimension(:) :: A
...
20  ALLOCATE(A(N))
...
49  do i = 1, N
50    Total = Total + A(I)
51  enddo
```

```
$ ifort -init=arrays,s nan -g -traceback sample.F90 -o sample.exe
$ sample.exe
```

```
  Image              PC                Routine            Line        Source
...
  sample.exe        00000000000402E12  MAIN__            50          sample.F90
  ...
  Aborted (core dumped)
```
Scalar Math Library Optimized

libimf (Linux*, OS X*) and libm (Windows*) Optimized for Intel® AVX2

- FMA instructions, in particular, lead to speed up
  - Both double precision and single precision
  - tan, sin, cos, exp, pow...

- Intel® AVX2 support detected at run-time and corresponding function version selected

- No special optimizations for Intel® AVX since increased vector width does not directly benefit scalar code

- Short Vector Math Library (libsvml) has vectorized function versions optimized for Intel® AVX2 and AVX-512
WRAP-UP
THANK YOU!
## Additional Product Information

Presentations may be arranged for other Intel® Parallel Studio XE 2016 Editions or products

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Composer Edition</td>
<td>Intel® Compiler (C++ / Fortran), Intel® MKL Math Library, Intel® TBB threading library, Intel® IPP media and data library</td>
</tr>
</tbody>
</table>
Legal Disclaimer & Optimization Notice

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.

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.

Copyright © 2015, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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