Please help us to improve our Material

Provide your feedback here:

http://intel.ly/2aGgxTe

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Intel's Optimizing Compiler

Software Solutions Group
Intel® Corporation

Agenda

Intel's Optimizing Compiler: Brief Overview/Recap

Vector Programming

Intel® AVX-512

Consistency of Floating-Point Results

High Bandwidth Memory

Some Tips for Using OpenMP*

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Intel® Compiler: Brief Recap

Supports standards

- Fortran77, Fortan90, Fortran95, Fortran2003, most Fortran 2008
- Up to C99 (most of C11);   C++11 and much C++14
- Intel® Fortran (and C/C++) binary compatible with gcc, gdb, …
  - But not binary compatible with gfortran

Supports all instruction sets via vectorization (auto- and explicit)

OpenMP* 4.0 support, some 4.5, no user-defined reductions

Optimized math libraries   (including for KNL)

Many advanced optimizations

- With detailed, structured optimization reports

Drivers: icc, icpc, ifort

To set the environment: source compilervars.sh intel64

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Basic Optimizations with icc/ifort -O…

-O0 no optimization; sets -g for debugging

-O1 scalar optimizations

- Excludes optimizations tending to increase code size

-O2 **default** (except with -g)

- includes **auto-vectorization**; some loop transformations such as unrolling; inlining within source file;
- Start with this (after initial debugging at -O0)

-O3 more aggressive loop optimizations

- Including cache blocking, loop fusion, loop interchange, …
- May not help all applications; need to test

-qopt-report

- Generates compiler optimization reports in files *.optrpt

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Inter Procedural Optimization (IPO)

icc -ipo or ifort -ipo

Analysis & Optimization across function and source file boundaries, e.g.

- Function inlining; Interprocedural constant propagation; Alignment analysis; Disambiguation; Data & Function Layout; etc.

2-step process:

- Compile phase – objects contain intermediate representation
- "Link" phase – compile and optimize over all such objects
  - Fairly seamless: the linker automatically detects objects built with -ipo, and their compile options
  - May increase build-time and binary size
  - But can be done in parallel with -ipo[n], -ipo-jobs<n>
  - Entire program need not be built with IPO/LTO, just hot modules

Particularly effective for C++ apps with many smaller functions

Get report on inlined functions with -qopt-report-phase=ipo

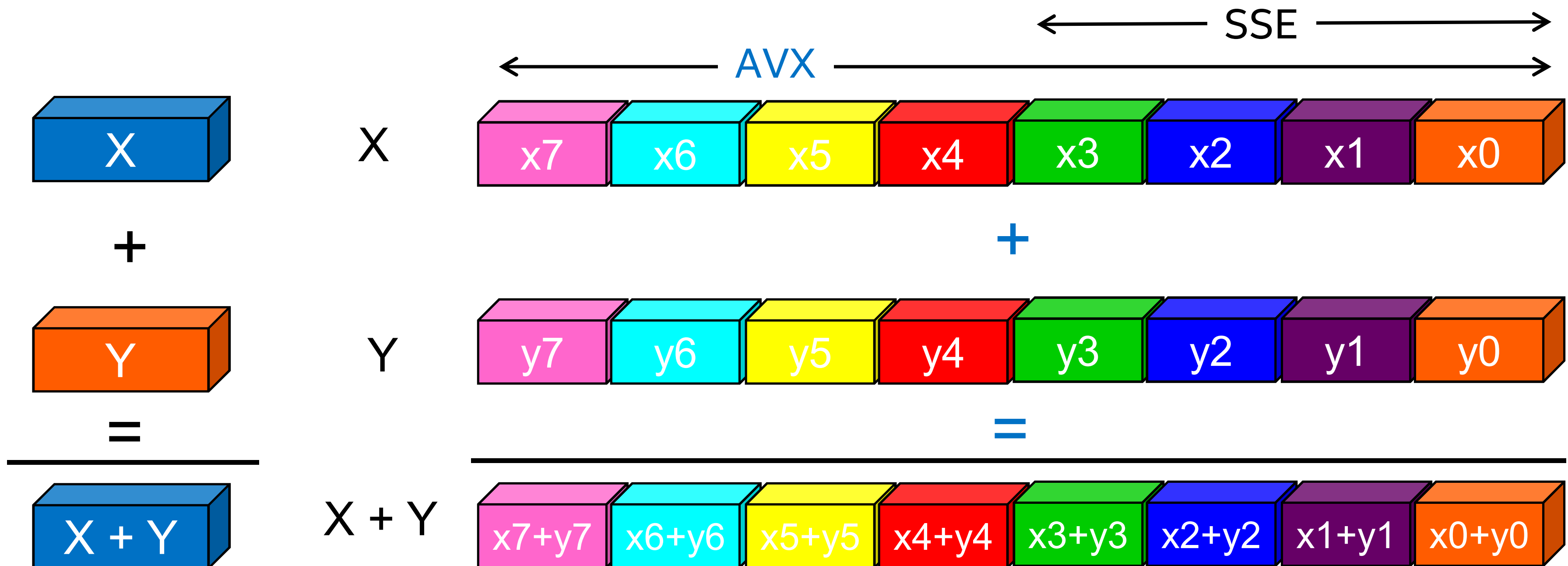
Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

VECTOR PROGRAMMING

SIMD: Single Instruction, Multiple Data

```
for (i=0;  i<n;  i++)      z[i] = x[i] + y[i];
```

- Scalar mode
  - one instruction produces one result
  - E.g. vadd**ss**, (vadd**sd**)
- Vector (SIMD) mode
  - one instruction can produce multiple results
  - E.g. vadd**ps**, (vadd**pd**)



Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Guidelines for Writing Vectorizable Code

Prefer simple "for" or "DO" loops

Write straight line code. Try to avoid:

- function calls (unless inlined or SIMD-enabled functions)
- branches that can't be treated as masked assignments

Avoid dependencies between loop iterations

- Or at least, avoid read-after-write dependencies

Prefer arrays to the use of pointers

- Without help, the compiler often cannot tell whether it is safe to vectorize code containing pointers.
- Try to use the loop index directly in array subscripts, instead of incrementing a separate counter for use as an array address.
- Disambiguate function arguments, e.g. -fargument-noalias or "restrict"

Use efficient memory accesses

- Favor inner loops with unit stride
- Minimize indirect addressing a[i] = b[ind[i]]
- Align your data consistently where possible (to 16, 32 or 64 byte boundaries)

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Example for Vectorizable Code: Search Loop Pattern

```
void mysearch(float *a,float *b, int n)
{
    int i;
    for (i = 0; i < n; i++)
    {
         if  (a [i] < 0. )  break;
          c[i] = sqrt(a[i]) * b[i] ;
      }

      return;
}
```

```
void mysearch(float *a,float *b, int n)
{
    int i, ns;
    for (i = 0; i < n; i++)
    {
         if  (a [i] < 0. )  break;
     }
     ns = i;
     for (i = 0; i < ns; i++)
     {
         c[i] = sqrt(a[i]) * b[i] ;
      }

      return;
}
```

Left-hand side will not vectorize:

remark #15520: loop was not vectorized: loop with multiple exits cannot be vectorized unless it meets search loop idiom criteria

Right-hand side: Both loops will vectorize

- Simple search pattern of the first loop is recognized by the compiler
- Computationally intensive second loop will be vectorized

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Explicit SIMD (Vector) Programming

Modeled on OpenMP* for threading (explicit parallel programming)

#pragma omp simd <clauses> (for loops)

#pragma omp declare simd <clauses> (for functions and subroutines)

Enables reliable vectorization of complex loops that compiler can't auto-vectorize

- E.g. outer loops

Directives are commands to the compiler, not hints

- Programmer is responsible for correctness (like OpenMP threading)
  - E.g. PRIVATE and REDUCTION clauses
- Overrides all dependency and cost-benefit analysis

Incorporated in OpenMP 4.0 ⇒ portable

- -qopenmp or -qopenmp-simd to enable

```
void addit(double* a, double * b, int m, int n, int x)  {
#pragma omp simd    // If you know x<0
  for (int i = m; i < m+n; i++)  a[i] = b[i] + a[i-x];
}
```

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Loops Containing Function Calls

- Function calls can have side effects that introduce a loop-carried dependency, preventing vectorization
- Possible remedies:
  - Inlining (-ip or –ipo)
    - best for small functions
  - !$OMP SIMD directive (last resort)
    - Calls to the routine will be serialized
    - Remaining parts of the loop body will be vectorize
  - SIMD-enabled functions
    - Good for large, complex functions and in contexts where inlining is difficult
    - Two ways to vectorize:
      - Call from regular DO loop
      - Adding "ELEMENTAL" keyword allows SIMD-enabled function to be called with array section argument

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

SIMD-enabled Subroutine

Compiler generates SIMD-enabled (vector) version of a scalar subroutine that can be called from a vectorized loop:

```
subroutine test_linear(x, y)
!$omp declare simd (test_linear) linear(ref(x, y))
   real(8),intent(in)  :: x
   real(8),intent(out) :: y
   y = 1. + sin(x)**3
end subroutine test_linear
…
interface
…
do j = 1,n
    call test_linear(a(j), b(j))
enddo
```

Important because arguments passed by reference in Fortran

remark #15301: FUNCTION WAS VECTORIZED.

remark #15300: LOOP WAS VECTORIZED.

SIMD-enabled routine must have **explicit interface**

!$omp simd   not needed in simple cases like this

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

SIMD-enabled Subroutine

The LINEAR(REF) clause is very important

- In C, compiler places consecutive argument values in a vector register
- But Fortran passes arguments by reference
  - By default compiler places consecutive addresses in a vector register
  - Leads to a gather of the 4 addresses (slow)
  - LINEAR(REF(X)) tells the compiler that the addresses are consecutive; only need to dereference once and copy consecutive values to vector register
  - New in compiler version 16.0.1
- Same method could be used for C arguments passed by reference

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

How to Align Data (Fortran)

Align array on an "n"-byte boundary (n must be a power of 2)

```
!dir$ attributes align:n :: array
```

- Works dynamically allocated, automatic and static arrays (not in common)

For a 2D array, choose column length to be a multiple of n,
so that consecutive columns have the same alignment (pad if necessary)

`-align array64byte` compiler tries to align all array types

And tell the compiler…

```
!dir$ vector aligned   OR
!$omp simd aligned( var [,var…]:<n>)
```

- Asks compiler to vectorize, assuming all array data accessed in loop are aligned for targeted processor
  - May cause fault if data are not aligned

```
!dir$ assume_aligned array:n   [,array2:n2, …]
```

- Compiler may assume array is aligned to n byte boundary
- Typical use is for dummy arguments
- Extension for allocatable arrays v17

```
!DIR$ ASSUME_ALIGNED a(1):64, b(1):64, c(1):64, d(1):64
```

n=16 for Intel® SSE, n=32 for Intel® AVX, n=64 for Intel® AVX-512

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

How to Align Data (C/C++)

Allocate memory on heap aligned to n byte boundary:

```
void* _mm_malloc(int size, int n)
int posix_memalign(void **p, size_t n, size_t size)
void* aligned_alloc(size_t alignment, size_t size)     (C11)
#include <aligned_new>                                 (C++11)
```

Alignment for variable declarations:

```
__attribute__((aligned(n)))  var_name     or
__declspec(align(n))  var_name
```

And tell the compiler…

```
#pragma vector aligned
#pragma omp simd aligned( var [,var…]:<n>)
```

- Asks compiler to vectorize, overriding cost model, and assuming all array data accessed in loop are aligned for targeted processor
- May cause fault if data are not aligned

```
__assume_aligned(array, n)
```

- Compiler may assume array is aligned to n byte boundary

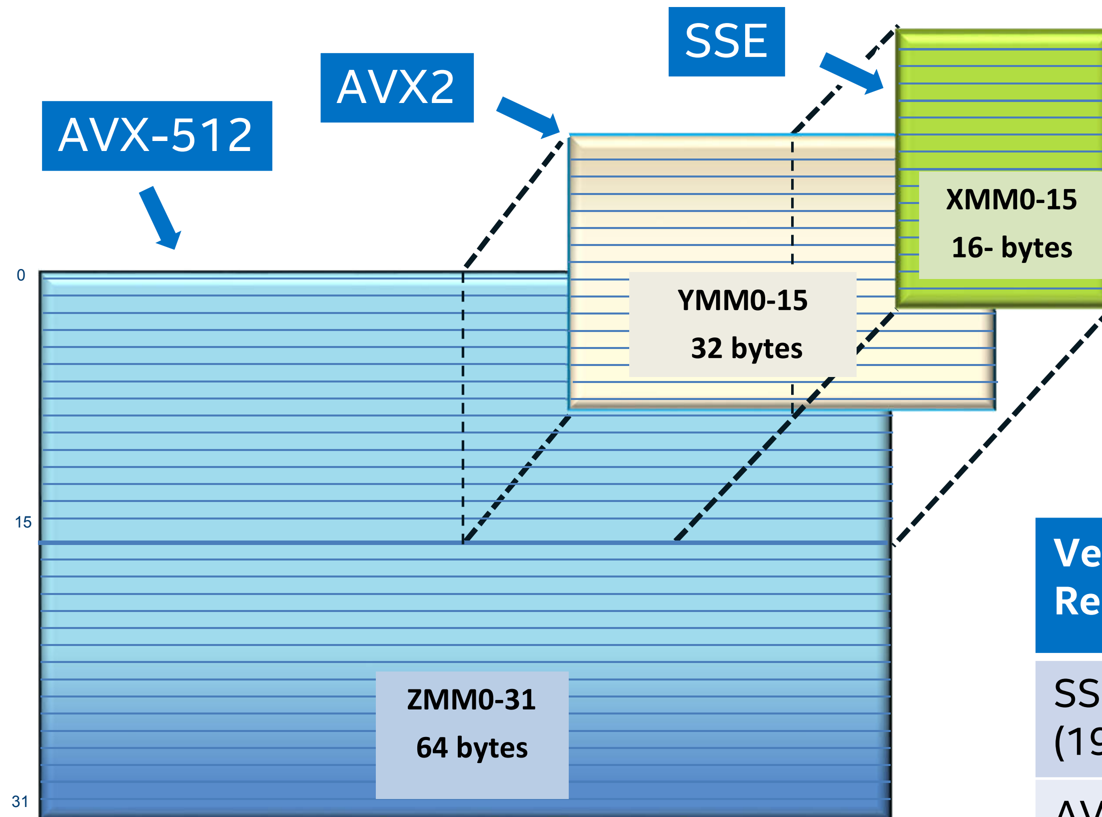
n=64 for Intel® Xeon Phi™ coprocessors, n=32 for Intel® AVX, n=16 for Intel® SSE

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

INTEL AVX-512 INSTRUCTION SET ARCHITECTURE

Changes for SW development resulting from new Intel® AVX-512 ISA

AVX-512 - Greatly increased Register File

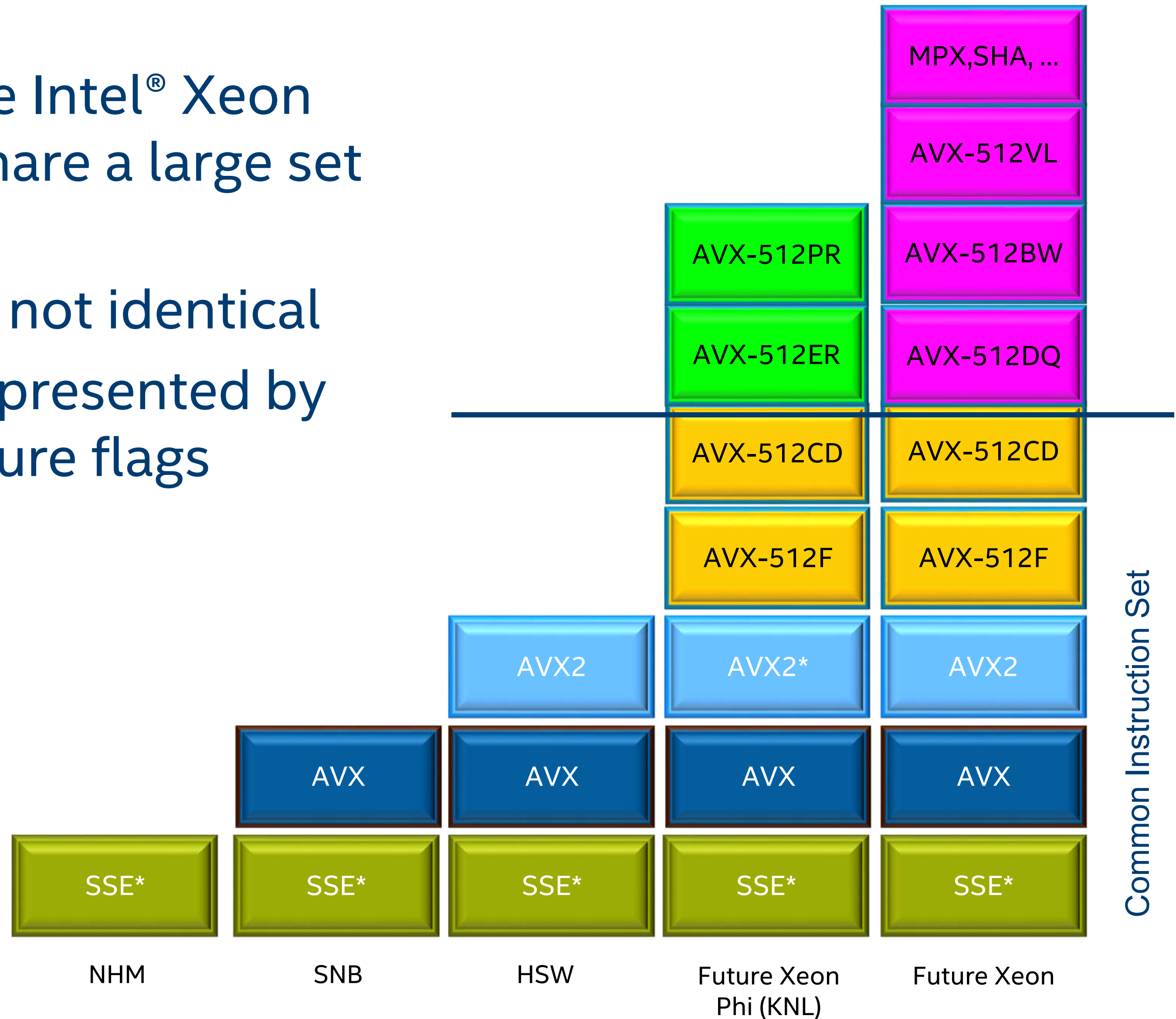


Vector Registers	IA32 (32bit)	Intel64 (64bit)
SSE (1999)	8 x 128bit	16 x 128bit
AVX and AVX-2 (2011 / 2013)	8 x 256bit	16 x 256bit
AVX-512 (2014 – KNL)	8 x 512bit	32 x 512bit

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

AVX-512 – KNL and future Xeon

- KNL and future Intel® Xeon architecture share a large set of instructions
  - – but sets are not identical
- Subsets are represented by individual feature flags (CPUID)



Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Intel® Compiler Switches Targeting Intel® AVX-512

Switch	Description
`-xmic-avx512`	KNL only
`-xcore-avx512`	Future Xeon only
`-xcommon-avx512`	AVX-512 subset common to both. Not a fat binary.
`-m, -march, /arch`	Not yet !
`-axmic-avx512 etc.`	Fat binaries. Allows to target KNL and other Intel® Xeon® processors
`-qoffload-arch=mic-avx512`	Offload to KNL coprocessor
`-mmic`	No – not for KNL

All supported in 16.0 and forthcoming 17.0 compilers

Binaries built for earlier Intel® Xeon® processors will run unchanged on KNL
Binaries built for Intel® Xeon Phi™ coprocessors will not.

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Optimization Improvements

Vectorization works as for other targets

- 512, 256 and 128 bit instructions available
- 64 byte alignment is best, like for KNC
- **New instructions can help**

```
for (int i; i <N; i++)  {
    if (a[i] > 0)   {
        b[j++]  = a[i];   // compress
        c[i]  = a[k++];  // expand
    }
}
```

- Cross-iteration dependencies by j and k

Vectorization of compress/expand loops:

- Uses vcompress/vexpand on KNL

Convert certain gathers to vector loads

Can auto-generate Conflict Detection instructions (AVX512CD)

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Compress Loop with Intel® AVX2

```
nb = 0
 do ia=1, na               ! line 11
  if(a(ia) > 0.) then
   nb = nb + 1
   b(nb) = a(ia)
  endif
 enddo
```

With Intel® AVX2, does not auto-vectorize

- And vectorizing with SIMD would be too inefficient

```
ifort -c -xcore-avx2 -qopt-report-file=stderr -qopt-report=3 -qopt-report-phase=vec compress.f90
 …
 LOOP BEGIN at compress.f90(11,3)
  remark #15344: loop was not vectorized: vector dependence prevents vectorization.
                 First dependence is shown below. Use level 5 report for details
  remark #15346: vector dependence: assumed ANTI dependence between  line 13 and  line 13
 LOOP END
```

- C code behaves the same

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Compress Loop with Intel® AVX-512

Compile for KNL:

```
ifort -c -qopt-report-file=stderr -qopt-report=3 -qopt-report-phase=vec -xmic-avx512 compress.f90
…
LOOP BEGIN at compress.f90(11,3)
  remark #15300: LOOP WAS VECTORIZED
  remark #15450: unmasked unaligned unit stride loads: 1
  remark #15457: masked unaligned unit stride stores: 1
…
  remark #15478: estimated potential speedup: 14.040
  remark #15497: vector compress: 1
LOOP END

grep vcompress compress.s
      vcompressps %zmm4, -4(%rsi,%rdx,4){%k1}        #14.7 c7 stall 1
      vcompressps %zmm1, -4(%rsi,%r12,4){%k1}        #14.7 c5
      vcompressps %zmm1, -4(%rsi,%r12,4){%k1}        #14.7 c5
      vcompressps %zmm4, -4(%rsi,%rdi,4){%k1}        #14.7 c7 stall 1
```

Observed speed-up is substantial but depends on problem size, data layout, etc.

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Adjacent Gather Optimizations

Or "Neighborhood Gather Optimizations"

```
do j=1,n
      y(j) = x(1,j) + x(2,j) + x(3,j) + x(4,j) ....
```

- Elements of x are adjacent in memory, but vector index is in other dimension
- Compiler generates simd loads and shuffles for x instead of gathers
  - Before AVX-512: gather of x(1,1), x(1,2), x(1,3), x(1,4)
  - With AVX-512: SIMD loads of x(1,1), x(2,1), x(3,1), x(4,1) etc., followed by permutes to get back to x(1,1), x(1,2), x(1,3), x(1,4) etc.
  - Message in optimization report:

    remark #34029: adjacent sparse (indexed) loads optimized for speed
- Arrays of short vectors or structs are very common

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Histogramming with Intel® AVX2

```
!     Accumulate histogram of sin(x) in  h
      do i=1,n
           y      = sin(x(i)*twopi)
           ih     = ceiling((y-bot)*invbinw)
           ih     = min(nbin,max(1,ih))
           h(ih) = h(ih) + 1                 ! line 15
      enddo
```

With Intel® AVX2, this does not vectorize

- Store to **h** is a scatter
- **ih** can have the same value for different values of **i**
- Vectorization with a SIMD directive would cause incorrect results

```
ifort -c -xcore-avx2 histo2.f90 -qopt-report-file=stderr -qopt-report-phase=vec
LOOP BEGIN at histo2.f90(11,4)
  remark #15344: loop was not vectorized: vector dependence prevents vectorization.
                        First dependence is shown below. Use level 5 report for details
  remark #15346: vector dependence: assumed FLOW dependence between  line 15 and  line 15
LOOP END
```


Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Histogramming with Intel® AVX-512

Compile for KNL:

```
ifort -c -xmic-avx512 histo2.f90 -qopt-report-file=stderr -qopt-report=3 –S
…
LOOP BEGIN at histo2.f90(11,4)
  remark #15300: LOOP WAS VECTORIZED
  remark #15458: masked indexed (or gather) loads: 1
  remark #15459: masked indexed (or scatter) stores: 1
  remark #15478: estimated potential speedup: 13.930
  remark #15499: histogram: 2
LOOP END
```

Some remarks omitted

```
vpminsd     %zmm5, %zmm21, %zmm3              #14.7 c19
vpconflictd %zmm3, %zmm1                      #15.7 c21
vpgatherdd  -4(%rsi,%zmm3,4), %zmm6{%k1}      #15.15 c21
vptestmd    %zmm18, %zmm1, %k0                #15.7 c23
kmovw       %k0, %r10d                        #15.7 c27 stall 1
vpaddd      %zmm19, %zmm6, %zmm2              #15.7 c27
testl       %r10d, %r10d
…
 vpscatterdd %zmm2, -4(%rsi,%zmm3,4){%k1}     #15.7 c3
```

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Histogramming with Intel® AVX-512 (cont'd)

Observed speed-up between AVX2 (non-vectorized) and AVX512 (vectorized) can be large, but depends on problem details

- Comes mostly from vectorization of other heavy computation in the loop
  - Not from the scatter itself
- Speed-up may be (much) less if there are many conflicts
  - E.g. histograms with a singularity or narrow spike

Other problems map to this

- E.g. energy deposition in cells in particle transport Monte Carlos

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Intel® Software Development Emulator (SDE)

Intel Compilers Already Recognize Intel® AVX-512 and Will Generate KNL Code

Use Intel® Software Development Emulator (SDE) to Test Code

- Will test instruction mix, not performance
- Does not emulate hardware (e.g. memory hierarchy) only ISA

Use the SDE to Answer

- Is my compiler generating Intel® AVX-512/KNL-ready code?
- How do I restructure my code so that Intel® AVX-512 code is generated?

Visit *Intel Xeon Phi Coprocessor code named "Knights Landing" - Application Readiness*

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

FLOATING POINT CONSISTENCY

Dealing with FP consistency when moving from Intel® MIC or Intel Xeon to KNL

Floating-Point Reproducibility

-fp-model precise disables most value-unsafe optimizations (especially re-associations)

- The primary way to get consistency between different platforms or different optimization levels

Example of **disabled** optimizations:

- Vectorization of loops containing transcendental functions
- Fast, approximate division and square roots
- Flush-to-zero of denormals
- Vectorization of reduction loops
- Evaluation of constant expressions at compile time
- …

Does not prevent differences due to:

- Different implementations of math functions:
  - For consistency of math functions between KNL and Intel® Xeon® processors use
    -fimf-arch-consistency=true available for Xeon and KNL
- Use of fused multiply-add instructions (FMAs): see next slide

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

FMAs

Most common cause of differences between Intel® Xeon® processors and Intel® Xeon Phi™

- Not disabled by -fp-model precise
- Can disable for testing with -no-fma
- Or by function-wide pragma or directive:

  #pragma float_control(fma,off)

  !dir$ nofma

  - With some impact on performance
- -fp-model strict disables FMAs, amongst other things
  - But on KNC, results in non-vectorizable x87 code
- The fma() intrinsic in C should always give a result with a single rounding, even on processors with no FMA instruction

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

FMAs

Can cause issues even when both platforms support them
(e.g. Haswell and KNL)

- Optimizer may not generate them in the same places
  - No language rules
- FMAs may break the symmetry of an expression:

```
c = a;    d = -b;
result = a*b + c*d;       ( = 0  if no FMAs )
```

If FMAs are supported, the compiler may convert to either

```
result = fma(c, d, (a*b))    or    result = fma(a, b, (c*d))
```

Because of the different roundings, these may give results that are non-zero and/or different from each other.

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Bottom Line for FP consistency

To get consistent results between KNL and Intel Xeon processors, use

-fp-model precise -fimf-arch-consistency=true -no-fma

(you could try omitting -no-fma for Xeon processors that support FMA, but FMA's could still possibly lead to differences)

To come: One-Stop-Shop" in the 17.0 compiler:

-fp-model consistent

Note: even with this there will be the usual caveats: threaded reductions, math libraries, different compilation targets, SIMD directives…)

To get consistent results that are as close as possible between KNC and Intel® Xeon® processors or KNL, try

-fp-model precise -no-fma on both.

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Tips for Debugging OpenMP applications

Build with -qopenmp to generate threaded code

- but run a single thread, OMP_NUM_THREADS=1
- If it works, try Intel® Inspector XE to detect threading errors
- If still fails, excludes race conditions or other synchronization issues as cause

Build with -openmp-stubs -auto

- RTL calls are resolved; but no threaded code is generated
- allocates local arrays on the stack, as for OpenMP
- If works, check for missing FIRSTPRIVATE, LASTPRIVATE
- If still fails, eliminates threaded code generation as cause
- If works without -auto, implicates changed memory model
  - Perhaps insufficient stack size
  - Perhaps values not preserved between successive calls

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Additional Resources (Optimization)

Webinars:

https://software.intel.com/videos/getting-the-most-out-of-the-intel-compiler-with-new-optimization-reports
https://software.intel.com/videos/new-vectorization-features-of-the-intel-compiler
https://software.intel.com/articles/further-vectorization-features-of-the-intel-compiler-webinar-code-samples
https://software.intel.com/videos/from-serial-to-awesome-part-2-advanced-code-vectorization-and-optimization
https://software.intel.com/videos/data-alignment-padding-and-peel-remainder-loops

Vectorization Guide (C):

https://software.intel.com/articles/a-guide-to-auto-vectorization-with-intel-c-compilers/

Explicit Vector Programming in Fortran:

https://software.intel.com/articles/explicit-vector-programming-in-fortran

Initially written for Intel® Xeon Phi™ coprocessors, but also applicable elsewhere:

https://software.intel.com/articles/vectorization-essential

https://software.intel.com/articles/fortran-array-data-and-arguments-and-vectorization

Compiler User Forums at http://software.intel.com/forums

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Additional Resources

https://software.intel.com/articles/intel-xeon-phi-coprocessor-code-named-knights-landing-application-readiness

https://software.intel.com/sites/default/files/managed/4c/1c/parallel_mag_issue20.pdf

https://software.intel.com/articles/what-disclosures-has-intel-made-about-knights-landing

https://software.intel.com/articles/intel-software-development-emulator

https://github.com/memkind

https://software.intel.com/articles/consistency-of-floating-point-results-using-the-intel-compiler

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

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.

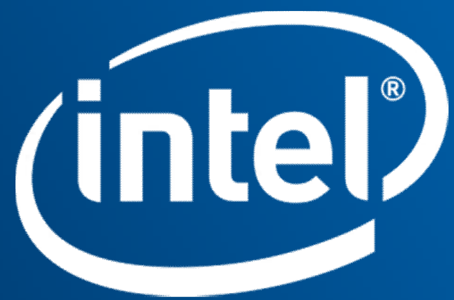
Copyright © 2016, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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

Copyright © 2016, Intel Corporation. All rights reserved.
brands may be claimed as the property of others.


intel

Optimization Report Example

$ icc -c -qopt-report=5 –xavx **-fargument-noalias** foo.c

Begin optimization report for: foo
.....
LOOP BEGIN at foo.c(4,7)
 remark #15389: vectorization support: reference theta has **unaligned** access   [foo.c(5,20)]
  remark #15389: vectorization support: reference sth has **unaligned** access   [foo.c(5,11)]
  remark #15381: vectorization support: unaligned access used inside loop body
  remark #15305: vectorization support: **vector length 4**
  remark #15309: vectorization support: normalized vectorization overhead 0.043
  remark #15417: vectorization support: number of **FP up converts**: single precision to double precision 1   [foo.c(5,20)]
  remark #15418: vectorization support: number of **FP down converts**: double precision to single precision 1   [foo.c(5,11)]
  remark #15300: LOOP WAS VECTORIZED
  remark #15450: unmasked unaligned unit stride loads: 1
  remark #15451: unmasked unaligned unit stride stores: 1
  remark #15475: --- begin vector loop cost summary ---
  remark #15476: scalar loop cost: 114
  remark #15477: vector loop cost: 40.750
  remark #15478: estimated potential speedup: 2.790
  remark #15482: vectorized math library calls: 1
  remark #15487: type converts: 2
  remark #15488: --- end vector loop cost summary ---
  remark #25015: Estimate of max trip count of loop=32
LOOP END

```
#include <math.h>
void foo (float * theta, float * sth)   {
 int i;
 for (i = 0; i < 128; i++)
     sth[i] = sin(theta[i]+3.1415927);
}
```

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Optimization Report Example

$ icc -c -qopt-report=5 -fargument-noalias –xavx foo.c

Begin optimization report for: foo
LOOP BEGIN at foo.c(6,7)
  remark #15388: vectorization support: **reference theta has aligned access** [foo.c(7,20)]
  remark #15388: vectorization support: **reference sth has aligned access** [foo.c(7,11)]
  remark #15412: vectorization support: **streaming store** was generated for sth [foo.c(7,11)]
  remark #15305: vectorization support: **vector length 8**
  remark #15309: vectorization support: normalized vectorization overhead 0.013
  remark #15300: LOOP WAS VECTORIZED
  remark #15448: unmasked aligned unit stride loads: 1
  remark #15449: unmasked aligned unit stride stores: 1
  remark #15467: unmasked aligned streaming stores: 1
  remark #15475: --- begin vector loop cost summary ---
  remark #15476: scalar loop cost: 110
  remark #15477: vector loop cost: 9.870
  remark #15478: **estimated potential speedup: 11.130**
  remark #15482: vectorized math library calls: 1
  remark #15488: --- end vector loop cost summary ---
  remark #25015: Estimate of max trip count of **loop=250000**
LOOP END

```
#include <math.h>
void foo (float * theta, float * sth)  {
  int i;
  __assume_aligned(theta,32);
  __assume_aligned(sth,32);
For (I = 0; I < 2000000; i++)
    sth[i] = sinf(theta[i]+3.1415927f);
};
```

Estimated performance speedup from type conversion: 9.7
Additional performance gain due to alignment and streaming stores

Optimization Notice
Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Optimization Report Example

$ icc -c -qopt-report=5 -fargument-noalias –xmic-avx512 foo.c

Begin optimization report for: foo
LOOP BEGIN at foo.c(6,7)
  remark #15388: vectorization support: reference theta has **aligned** access   [foo.c(7,20)]
  remark #15388: vectorization support: reference sth has **aligned** access   [foo.c(7,11)]
  remark #15412: vectorization support: **streaming store** was generated for sth   [foo.c(7,11)]
  remark #15305: **vectorization support: vector length 16**
  remark #15309: vectorization support: normalized vectorization overhead 0.013
  remark #15300: LOOP WAS VECTORIZED
  remark #15448: unmasked aligned unit stride loads: 1
  remark #15449: unmasked aligned unit stride stores: 1
  remark #15467: unmasked aligned streaming stores: 1
  remark #15475: --- begin vector loop cost summary ---
  remark #15476: scalar loop cost: 111
  remark #15477: vector loop cost: 4.930
  remark #15478: **estimated potential speedup: 22.480**
  remark #15482: vectorized math library calls: 1
  remark #15488: --- end vector loop cost summary ---
  remark #25015: Estimate of max trip count of loop=125000
LOOP END

```
#include <math.h>
void foo (float * theta, float * sth)  {
  int i;
  __assume_aligned(theta,64);
  __assume_aligned(sth,64);
for (i = 0; i < 2000000; i++)
    sth[i] = sinf(theta[i]+3.1415927f);
}
```

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.