Intel® Compiler Introduction

Agenda

- Intel® Compilers overview
- High-Level Optimizations
- Optimization reports
- Intraprocedural Optimizations

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

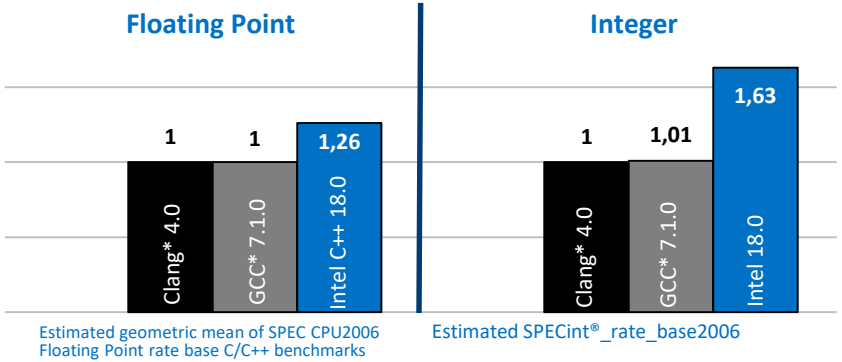
Intel® Compilers

As a software developer, I care about:		…and my challenges are:	Intel compilers offer:
	**Performance** - I develop applications that need to execute FAST	Taking advantage of the latest hardware innovations	Developers the full power of the latest x86-compatible processors and instruction sets
	**Productivity** - I need productivity and ease of use offered by compilers	Finding support for the leading languages and programming models	Support for the latest Fortran, C/C++, and OpenMP* standards; compatibility with leading compilers and IDEs
	**Scalability** - I develop and debug my application locally, and deploy my application globally	Maintaining my code as core counts and vector widths increase at a fast pace	Scalable performance without changing code as newer generation processors are introduced

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Boost application performance on Linux* Intel® C++ and Fortran Compilers

Boost C++ application performance on Linux* using Intel® C++ Compiler (higher is better)



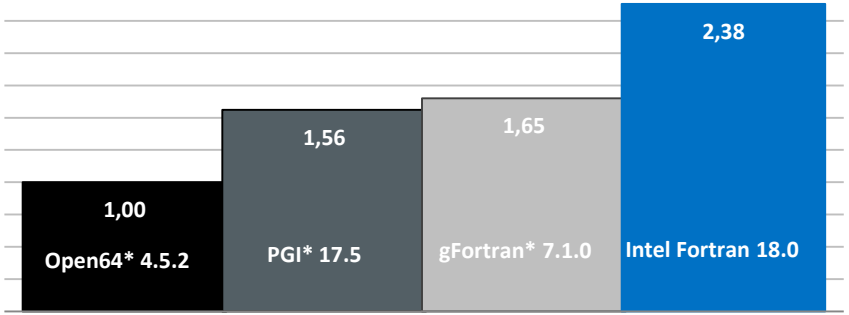
Relative geomean performance, SPEC* benchmark - higher is better

Configuration: Linux hardware: 2x Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz, 192 GB RAM, HyperThreading is on. Software: Intel compilers 18.0, GCC 7.1.0. PGI 15.10, Clang/LLVM 4.0. Linux OS: Red Hat Enterprise Linux Server release 7.2 (Maipo), kernel 3.10.0-514.el7.x86_64. SPEC* Benchmark (www.spec.org). SmartHeap 10 was used for CXX tests when measuring SPECint® benchmarks.
SPECint® rate base 2006 compiler switches: SmartHeap 10 were used for C++ tests. Intel C/C++ compiler 18.0: -m32 -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -qopt-mem-layout-trans=3 C++ code adds option -static . GCC 7.1.0: -m32 -Ofast -flto -march=core-avx2 -mfpmath=sse -funroll-loops . Clang 4.0: -m32 -Ofast -march=core-avx2 -flto -mfpmath=sse -funroll-loops C++ code adds option -fno-fast-math
SPECfp* rate base 2006 compiler switches: Intel C/C++ compiler 18.0: -m64 -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -qopt-mem-layout-trans=3 -auto-p32. C code adds option -static. Intel Fortran 18.0: -m64 -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -qopt-mem-layout-trans=3 -static. GCC 7.1.0: -m64 -Ofast -flto -march=core-avx2 -mfpmath=sse -funroll-loops. Clang 4.0: -m64 -Ofast -march=core-avx2 -flto -mfpmath=sse -funroll-loops
SPECint® speed base 2006 compiler switches: SmartHeap 10 were used for C++ tests. Intel C/C++ compiler 18.0: -m64 -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -auto-p32. C code adds options -static -parallel. GCC 7.1.0: -m64 -Ofast -flto -march=core-avx2 -mfpmath=sse -funroll-loops. C code adds options -ftree-parallelize-loops=40. Clang 4.0: -m64 -Ofast -march=core-avx2 -flto -mfpmath=sse -funroll-loops. C++ code adds options -fno-fast-math -fno-fast-math.
SPECfp® speed base 2006 compiler switches: Intel C/C++ compiler 18.0: -m64 -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -static -auto-p32. C code adds option -parallel. Intel Fortran 18.0: -m64 -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -static -parallel. GCC 7.1.0: -m64 -Ofast -flto -march=core-avx2 -mfpmath=sse -funroll-loops. C code adds option -ftree-parallelize-loops=40. Clang 4.0: -m64 -Ofast -march=core-avx2 -flto -mfpmath=sse -funroll-loops

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. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation

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 .

Boost Fortran application performance on Linux* using Intel® Fortran Compiler (higher is better)



Estimated relative geomean performance, Polyhedron* benchmark- higher is better

Configuration: Hardware: 2x Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz, 192 GB RAM, HyperThreading is on. Software: Intel Fortran compiler 18.0, PGI Fortran* 17.5, Open64* 4.5.2, gFortran* 7.1.0. Linux OS: Red Hat Enterprise Linux Server release 7.2 (Maipo), kernel 3.10.0-514.el7.x86_64 Polyhedron Fortran Benchmark (www.fortran.uk).
Linux compiler switches: Gfortran: -Ofast -mfpmath=sse -flto -march=haswell -funroll-loops -ftree-parallelize-loops=8. Intel Fortran compiler: -fast -parallel -xCORE-AVX2 -nostandard-realloc-lhs. PGI Fortran: -fast -Mipa=fast,inline -Msmartalloc -Mfprelaxed -Mstack_arrays -Mconcur=bind -tp haswell Open64: -march=bdver1 -mavx -mno-fma4 -Ofast -mso

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. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation

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 .

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

General Compiler Features

Supports standards

- Full C11, Full C++14, Some C++ 17
  - use -std option to control, e.g. -std=c++14
- Full Fortran 2008, Some F2018

Intel® C/C++ source and binary compatible

- Linux: gcc
- Windows: Visual C++ 2013 and above

Supports all instruction sets via vectorization

- Intel® SSE, Intel® AVX, Intel® AVX-512

Full OpenMP* 4.5 support

Optimized math libraries

- libimf (scalar) and libsvml (vector): faster than GNU libm
- Driver links libimf automatically, ahead of libm
- Replace math.h by mathimf.h

Many advanced optimizations

- With detailed, structured optimization reports

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Activity #1: No optimization

1. `cd ~/day1/lab1`

`source /opt/intel/parallel_studio_xe_2019/psxevars.sh intel64`

Check source code file matmult_2.cpp:

`less matmult_2.cpp`

2. Compile with O0 :

`icc -O0 matmult_2.cpp f.cpp -o matmult_2_O0`

3. Benchmark

`./matmult_2_O0`

transposed matmul time = 109.098726 seconds

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

High-Level Optimization (HLO)

Compiler switches:

Disable optimization	-O0
Optimize for speed (no code size increase)	-O1
Optimize for speed (default)	-O2

-O2 (default), -O3

- O3 is suited to applications that have loops that do many floating-point calculations or process large data sets.
- Some of the optimizations are the same as at O2, but are carried out more aggressively. Some poorly suited applications might run slower at O3 than O2

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Activity #2: Base measurements

1. Compile with -O2 optimization.

```
icc -O2 -qopt-report=2 matmult_2.cpp f.cpp -o matmult_2_O2
```

2. Benchmark

```
./matmult_2_O2
```

3. Discover optimizations used: `-qopt-report[=n]`, where n – level of details

4. Check optreport file

```
less matmult_2.optrpt
```

[Intel® Xeon® Gold 6140, 4-Core 64-bit, 64-bit Intel® C++ Compiler 18.0]

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Activity #3

1. Use intraprocedural optimization (-ipo)

```
icc -O2 -ipo -qopt-report=2 matmult_2.cpp f.cpp -o
matmult2_ipo
```

2. Benchmark

```
./matmult2_ipo
```

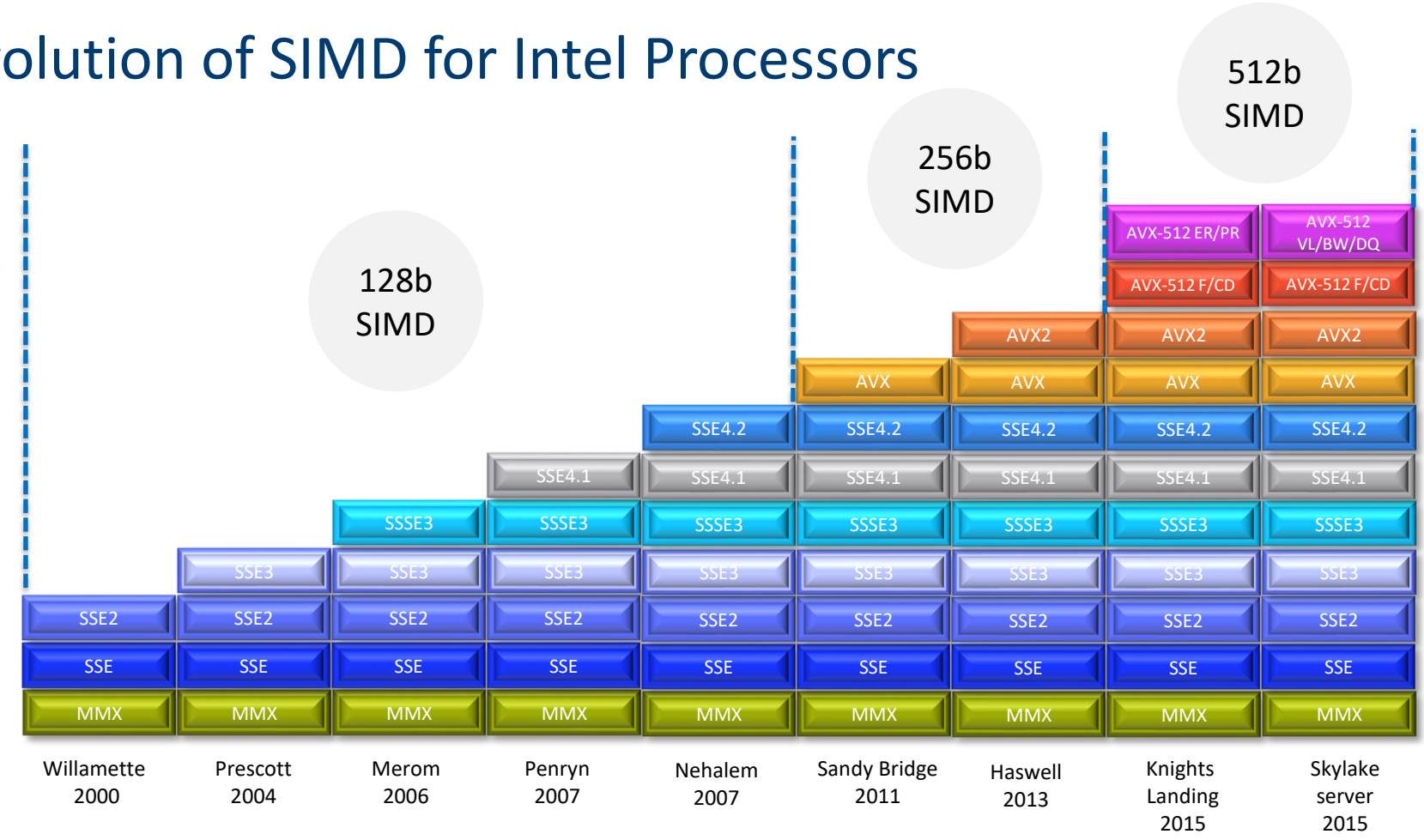
3. Check

```
less ipo_out.optrpt
```

F() is inlined

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Evolution of SIMD for Intel Processors



Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Activity #4

1. AVX2:

```
icc -O2 -xCORE-AVX2 -ipo matmult_2.cpp f.cpp -o matmult_2_AVX2

./matmult_2_AVX2
```

2. AVX512

```
icc -O2 -xCOMMON-AVX512 -ipo -qopt-zmm-usage=high matmult_2.cpp
f.cpp -o matmult_2_AVX512

./matmult_2_AVX512
```

Optimization Notice
Copyright © 2018, 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. For more complete information visit www.intel.com/benchmarks.

Copyright © 2018, 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

backup

Compiler Floating-Point (FP) Model

The Floating Point options allow to control the optimizations on floating-point data. These options can be used to tune the performance, level of accuracy or result consistency.

Accuracy
- Produce results that are “close” to the correct value
  - –Measured in relative error, possibly ulps (units in the last place)

Reproducibility
- Produce consistent results
  - –From one run to the next
  - –From one set of build options to another
  - –From one compiler to another
  - –From one platform to another

Performance
- Produce the most efficient code possible
  - –Default, primary goal of Intel® Compilers

These objectives usually conflict! Wise use of compiler options lets you control the tradeoffs.

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Why Results Vary I

Basic problem:

- FP numbers have **finite resolution** and
- **Rounding** is done for each (intermediate) result

Caused by algorithm:
Conditional numerical computation for different systems and/or input data can have unexpected results

Non-deterministic task/thread scheduler:
Asynchronous task/thread scheduling has best performance but reruns use different threads

Alignment (heap & stack):
If alignment is not guaranteed and changes between reruns the data sets could be computed differently (e.g. vector loop prologue & epilogue of unaligned data)

⇨ **User controls those (direct or indirect)**

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Why Results Vary II

Order of FP operations has impact on rounded result, e.g.

`(a+b)+c ≠ a+(b+c)`

$2^{-63} + 1 + -1 = 2^{-63}$ (mathematical result)

$(2^{-63} + 1) + -1 \approx 0$ (correct IEEE result)

$2^{-63} + (1 + -1) \approx 2^{-63}$ (correct IEEE result)

Constant folding: `X + 0` ⇨ `X` or `X * 1` ⇨ `X`

Multiply by reciprocal: `A/B` ⇨ `A * (1/B)`

Approximated transcendental functions (e.g. `sqrt(…)`, `sin(…)`, …)

Flush-to-zero (for SIMD instructions)

Contractions (e.g. FMA)

Different code paths (e.g. SIMD & non-SIMD or Intel AVX vs. SSE)

…

⇨ **Subject of Optimizations by Compiler & Libraries**

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Compiler Optimizations

Why compiler optimizations:

- Provide best performance
- Make use of processor features like SIMD (vectorization)
- In most cases performance is more important than FP precision and reproducibility
- Use faster FP operations (not legacy x87 coprocessor)

FP model of compiler limits optimizations and provides control about FP precision and reproducibility:

Default is "**`fast`**"

Controlled via:
Linux*: **`-fp-model`**

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Common Optimization Options

	Linux*
Disable optimization	-O0
Optimize for speed (no code size increase)	-O1
Optimize for speed (default)	-O2
High-level loop optimization	-O3
Create symbols for debugging	-g
Multi-file inter-procedural optimization	-ipo
Profile guided optimization (multi-step build)	-prof-gen -prof-use
Optimize for speed across the entire program (“prototype switch”) ***fast* options definitions changes over time!**	-fast same as: -ipo -O3 -no-prec-div -static -fp-model fast=2 -xHost)
OpenMP support	-qopenmp
Automatic parallelization	-parallel

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Intel® Compilers: Loop Optimizations

-O3

Loop optimizations:

- **Automatic vectorization**‡ (use of packed SIMD instructions)
- Loop interchange ‡ (for more efficient memory access)
- Loop unrolling‡ (more instruction level parallelism)
- Prefetching (for patterns not recognized by h/w prefetcher)
- Cache blocking (for more reuse of data in cache)
- Loop versioning ‡ (for loop count; data alignment; runtime dependency tests)
- Memcpy recognition ‡ (call Intel's fast memcpy, memset)
- Loop splitting ‡ (facilitate vectorization)
- Loop fusion (more efficient vectorization)
- Scalar replacement‡ (reduce array accesses by scalar temps)
- Loop rerolling (enable vectorization)
- Loop peeling ‡ (allow for misalignment)
- Loop reversal (handle dependencies)
- etc.

‡ **all or partly enabled at -O2**

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Compiler Reports - Optimization Report

- Enables the optimization report and controls the level of details
  - `-qopt-report[=n]`
  - When used without parameters, full optimization report is issued on stdout with details level 2
- Control destination of optimization report
  - `-qopt-report=<filename>`
  - By default, without this option, a <filename>.optrpt file is generated.
- Subset of the optimization report for specific phases only
  - `-qopt-report-phase[=list]`
    Phases can be:
    - all - All possible optimization reports for all phases (default)
    - loop - Loop nest and memory optimizations
    - vec - Auto-vectorization and explicit vector programming
    - par - Auto-parallelization
    - openmp - Threading using OpenMP
    - ipo - Interprocedural Optimization, including inlining
    - pgo - Profile Guided Optimization
    - cg - Code generation
    - offload - offload of data and/or execution to Intel® MIC Architecture or to Intel® Graphics Technology
      Note: “offload” does not report on optimizations for MIC, it reports on data that are offloaded.

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

FP Model

FP model settings:

- **`precise`**: allows value-safe optimizations only
- **`source`**/**`double`**/**`extended`**: intermediate precision for FP expression eval.
- **`except`**: enables strict floating point exception semantics
- **`strict`**: enables access to the FPU environment disables floating point contractions such as fused multiply-add (fma) instructions implies "**`precise`**" and "**`except`**"
- **`fast[=1]`** (default):
  Allows value-unsafe optimizations compiler chooses precision for expression evaluation
  Floating-point exception semantics not enforced
  Access to the FPU environment not allowed
  Floating-point contractions are allowed
- **`fast=2`**: some additional approximations allowed
- **`consistent:`** consistent, reproducible floating-point results for different optimization levels or between different processors of the same architecture

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Interprocedural Optimizations (IPO)

Multi-pass Optimization

- Interprocedural optimizations performs a static, topological analysis of your application!
- ip: Enables inter-procedural optimizations for current source file compilation
- ipo: Enables inter-procedural optimizations across files
  - Can inline functions in separate files
  - Especially many small utility functions benefit from IPO

Linux*
-ip
-ipo

Enabled optimizations:

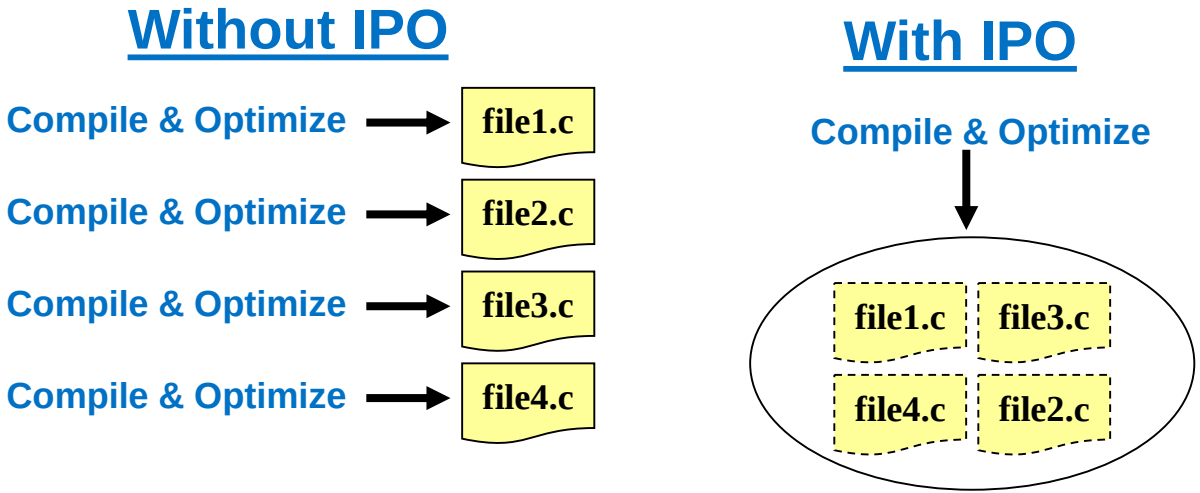
- Procedure inlining (reduced function call overhead)
- Interprocedural dead code elimination, constant propagation and procedure reordering
- Enhances optimization when used in combination with other compiler features
- Much of ip (including inlining) is enabled by default at option O2

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Interprocedural Optimizations

Extends optimizations across file boundaries

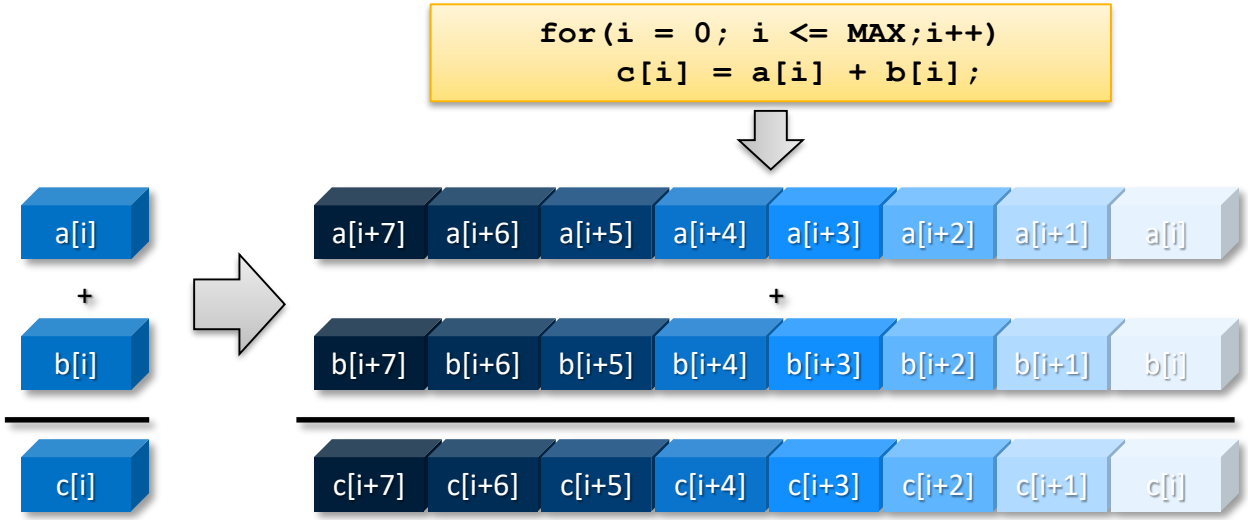
`-ip`	Only between modules of one source file
`-ipo`	Modules of multiple files/whole application



Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

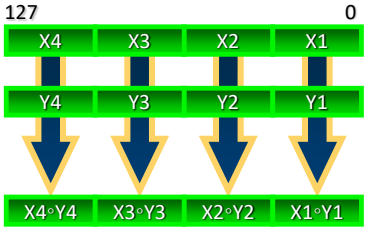
Vectorization of Code

- Transform sequential code to exploit vector processing capabilities (SIMD) of Intel processors
  - Manually by explicit syntax
  - Automatically by tools like a compiler

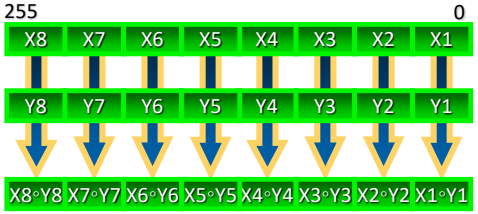


Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

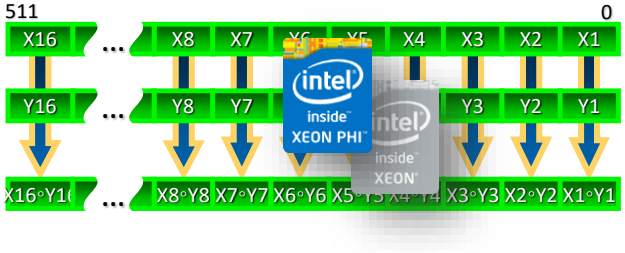
SIMD Types for Intel® Architecture



SSE
Vector size: **128 bit**
Data types:
8, 16, 32, 64 bit integer
32 and 64 bit float
VL: 2, 4, 8, 16



AVX
Vector size: **256 bit**
Data types:
8, 16, 32, 64 bit integer
32 and 64 bit float
VL: 4, 8, 16, 32

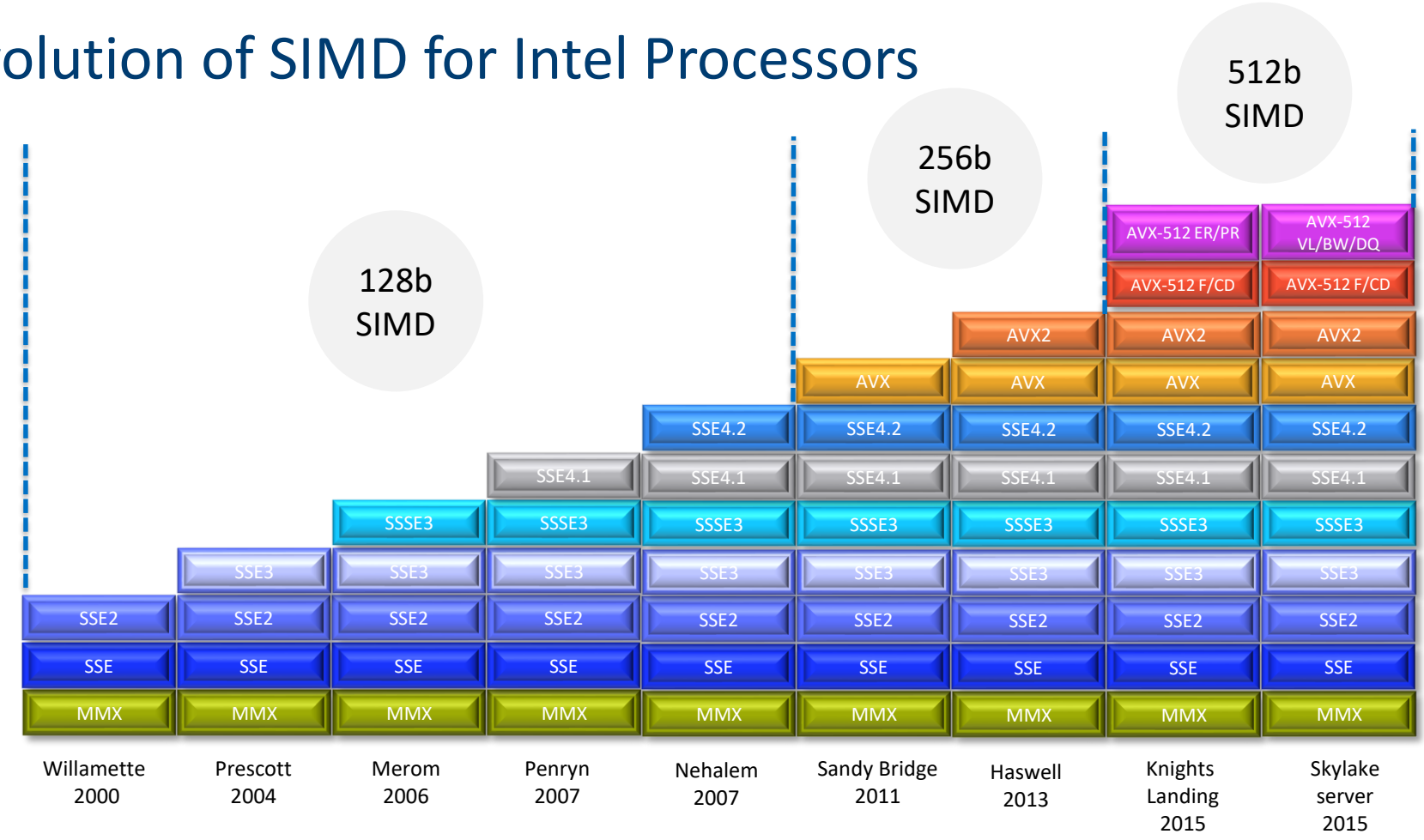


Intel® AVX-512
Vector size: **512 bit**
Data types:
8, 16, 32, 64 bit integer
32 and 64 bit float
VL: 8, 16, 32, 64

Illustrations: Xi, Yi & results 32 bit integer

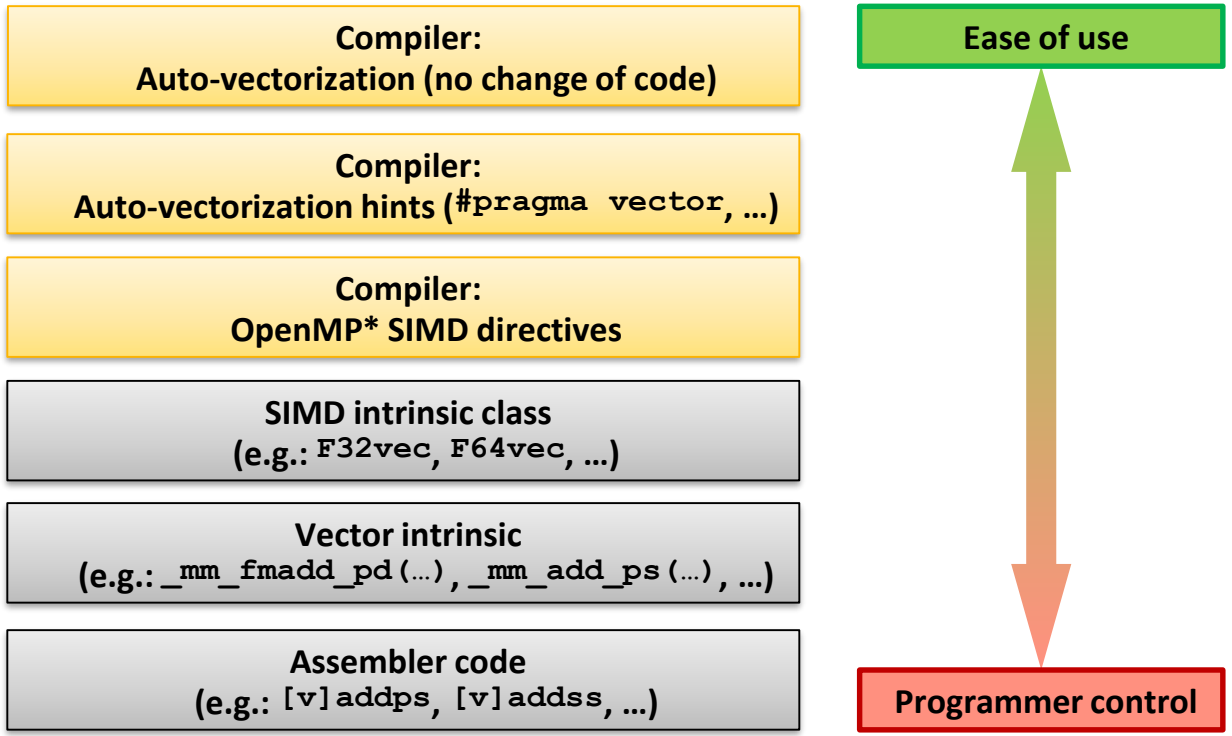
Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Evolution of SIMD for Intel Processors



Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Many Ways to Vectorize



Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Auto-vectorization of Intel Compilers


```
void add(double *A, double *B, double *C)
{
  for (int i = 0; i < 1000; i++)
  C[i] = A[i] + B[i];
}
```

```
subroutine add(A, B, C)
  real*8 A(1000), B(1000), C(1000)
  do i = 1, 1000
      C(i) = A(i) + B(i)
  end do
end
```

Intel® SSE4.2

```
.B2.14:
movups    xmm1, XMMWORD PTR [edx+ebx*8]
movups    xmm3, XMMWORD PTR [16+edx+ebx*8]
movups    xmm5, XMMWORD PTR [32+edx+ebx*8]
movups    xmm7, XMMWORD PTR [48+edx+ebx*8]
movups    xmm0, XMMWORD PTR [ecx+ebx*8]
movups    xmm2, XMMWORD PTR [16+ecx+ebx*8]
movups    xmm4, XMMWORD PTR [32+ecx+ebx*8]
movups    xmm6, XMMWORD PTR [48+ecx+ebx*8]
addpd     xmm1, xmm0
addpd     xmm3, xmm2
addpd     xmm5, xmm4
addpd     xmm7, xmm6
movups    XMMWORD PTR [eax+ebx*8], xmm1
movups    XMMWORD PTR [16+eax+ebx*8], xmm3
movups    XMMWORD PTR [32+eax+ebx*8], xmm5
movups    XMMWORD PTR [48+eax+ebx*8], xmm7
add       ebx, 8
cmp       ebx, esi
jb        .B2.14
...
```

Intel® AVX

```
.B2.15
vmovupd   ymm0, YMMWORD PTR [ebx+eax*8]
vmovupd   ymm2, YMMWORD PTR [32+ebx+eax*8]
vmovupd   ymm4, YMMWORD PTR [64+ebx+eax*8]
vmovupd   ymm6, YMMWORD PTR [96+ebx+eax*8]
vaddpd    ymm1, ymm0, YMMWORD PTR [edx+eax*8]
vaddpd    ymm3, ymm2, YMMWORD PTR [32+edx+eax*8]
vaddpd    ymm5, ymm4, YMMWORD PTR [64+edx+eax*8]
vaddpd    ymm7, ymm6, YMMWORD PTR [96+edx+eax*8]
vmovupd   YMMWORD PTR [esi+eax*8], ymm1
vmovupd   YMMWORD PTR [32+esi+eax*8], ymm3
vmovupd   YMMWORD PTR [64+esi+eax*8], ymm5
vmovupd   YMMWORD PTR [96+esi+eax*8], ymm7
add       eax, 16
cmp       eax, ecx
jb        .B2.15
```

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Basic Vectorization Switches I

Linux*, macOS*: **-x<code>**

- Might enable Intel processor specific optimizations
- Processor-check added to “main” routine:
  Application errors in case SIMD feature missing or non-Intel processor with appropriate/informative message

<code> indicates a feature set that compiler may target (including instruction sets and optimizations)

Microarchitecture code names: BROADWELL, HASWELL, IVYBRIDGE, KNL, KNM, SANDYBRIDGE, SILVERMONT, SKYLAKE, SKYLAKE-AVX512

SIMD extensions: COMMON-AVX512, MIC-AVX512, CORE-AVX512, CORE-AVX2, CORE-AVX-I, AVX, SSE4.2, etc.

Example: `icc -xCORE-AVX2 test.c`

`ifort -xSKYLAKE test.f90`

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Basic Vectorization Switches II

Linux*, macOS*: **-ax<code>**

- Multiple code paths: baseline and optimized/processor-specific
- Optimized code paths for Intel processors defined by **<code>**
- Multiple SIMD features/paths possible, e.g.: **-axSSE2,AVX**
- Baseline code path defaults to **-msse2**
- The baseline code path can be modified by **-m<code>** or **-x<code>**
- Example: `icc -axCORE-AVX512 -xAVX test.c`

Linux*, macOS*: **-m<code>**

- No check and no specific optimizations for Intel processors:
  Application optimized for both Intel and non-Intel processors for selected SIMD feature
- Missing check can cause application to fail in case extension not available

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Supported Processor-specific Compiler Switches

Intel® processors only	Intel and non-Intel (-m also GCC)
-xsse2	-msse2 (default)
-xsse3	-msse3
-xssse3	-mssse3
-xsse4.1	-msse4.1
-xsse4.2	-msse4.2
-xavx	-mavx
-xcore-avx2	
-xcore-avx512	
-xHost	-xHost (-march=native)
Intel cpuid check	No cpu id check
Runtime message if run on unsupported processor	Illegal instruction error if run on unsupported processor

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Validating Vectorization Success I

Optimization report:

- Linux*: **`-qopt-report=<n>`**
  `n`: **`0`**, …, **`5`** specifies level of detail; **`2`** is default
- Prints optimization report with vectorization analysis

Optimization report phase:

- Linux*: **`-qopt-report-phase=<p>`**
  `<p>` is **`all`** by default; use **`vec`** for just the vectorization report

Optimization report file:

- Linux*: **`-opt-report-file=<f>`**
- **`<f>`** can be **`stderr`**, **`stdout`** or a file (default: *.optrpt)

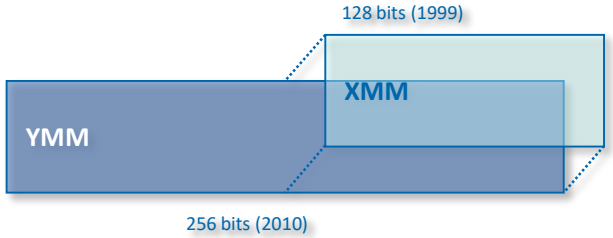
Assembler code inspection (Linux*: **`-S`**)

Intel® Advisor

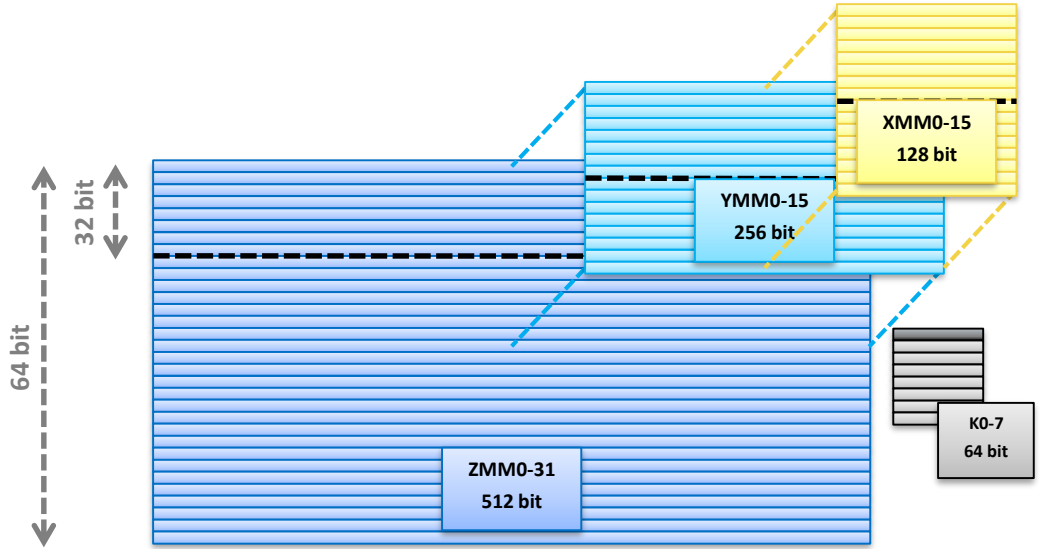
Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Intel® AVX and AVX-512 Registers

AVX is a 256 bit vector extension to SSE:



AVX-512 extends previous AVX and SSE registers to 512 bit:



OS support is required

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Tuning for Skylake SERVER

The Intel® Xeon® Processor Scalable Family is based on the server microarchitecture codenamed Skylake

Both Skylake and Knights Landing processors have support for Intel® AVX-512 instructions. There are three ISA options in the Intel® Compiler:

- `-xCORE-AVX512`: Targets Skylake, contains instructions not supported by Knights Landing
- `-xCOMMON-AVX512`: Targets both Skylake and Knights Landing
- `-xMIC-AVX512`: Targets Knights Landing, includes instructions not supported by Skylake

Many HPC applications benefit from aggressive ZMM usage

Most non-HPC apps degrade from aggressive ZMM usage

Intel® Compiler is conservative in its use of ZMM (512bit) registers

- To use with Skylake server add `-qopt-zmm-usage=high`

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Tuning for Skylake SERVER

- Compile with processor-specific option:

  Linux*, macOS*: **-xCORE-AVX512**

- New compiler option to enable a smooth transition from AVX2 to AVX-512

  Linux*, macOS*: **-qopt-zmm-usage=low|high**

  – Defines a level of zmm registers usage

  – Low is default for Skylake server

  – May control zmm usage with OpenMP* directive (simdlen) on code level

```
#include <math.h>
void foo(double *a, double *b, int size) {
  #pragma ivdep
  for(int i=0; i<size; i++) {
     b[i]=exp(a[i]);
  }
}
```

```
icpc -c -xCORE-AVX512 -qopenmp -qopt-report=5 foo.cpp
```

```
remark #15305: vectorization support: vector length 4
  …
remark #15321: Compiler has chosen to target XMM/YMM
vector. Try using -qopt-zmm-usage=high to override
  …
remark #15478: estimated potential speedup: 5.260
```

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Profile-Guided Optimizations (PGO)

Static analysis leaves many questions open for the optimizer like:

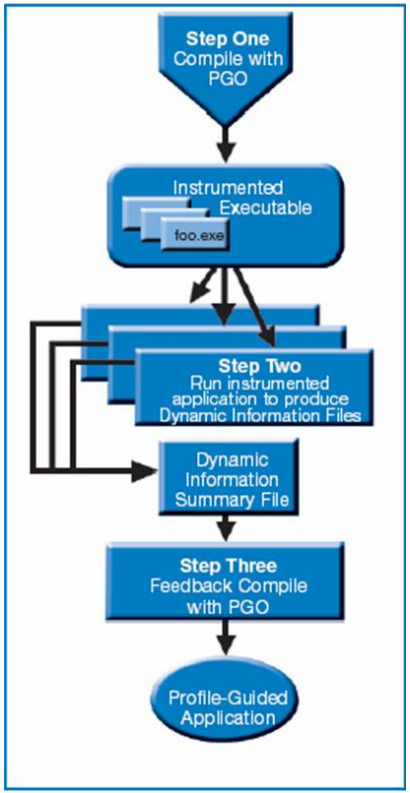
- How often is x > y
- What is the size of count
- Which code is touched how often

```
if (x > y)
        do_this();
 else
        do that();
```

```
for(i=0; i<count; ++I
do_work();
```

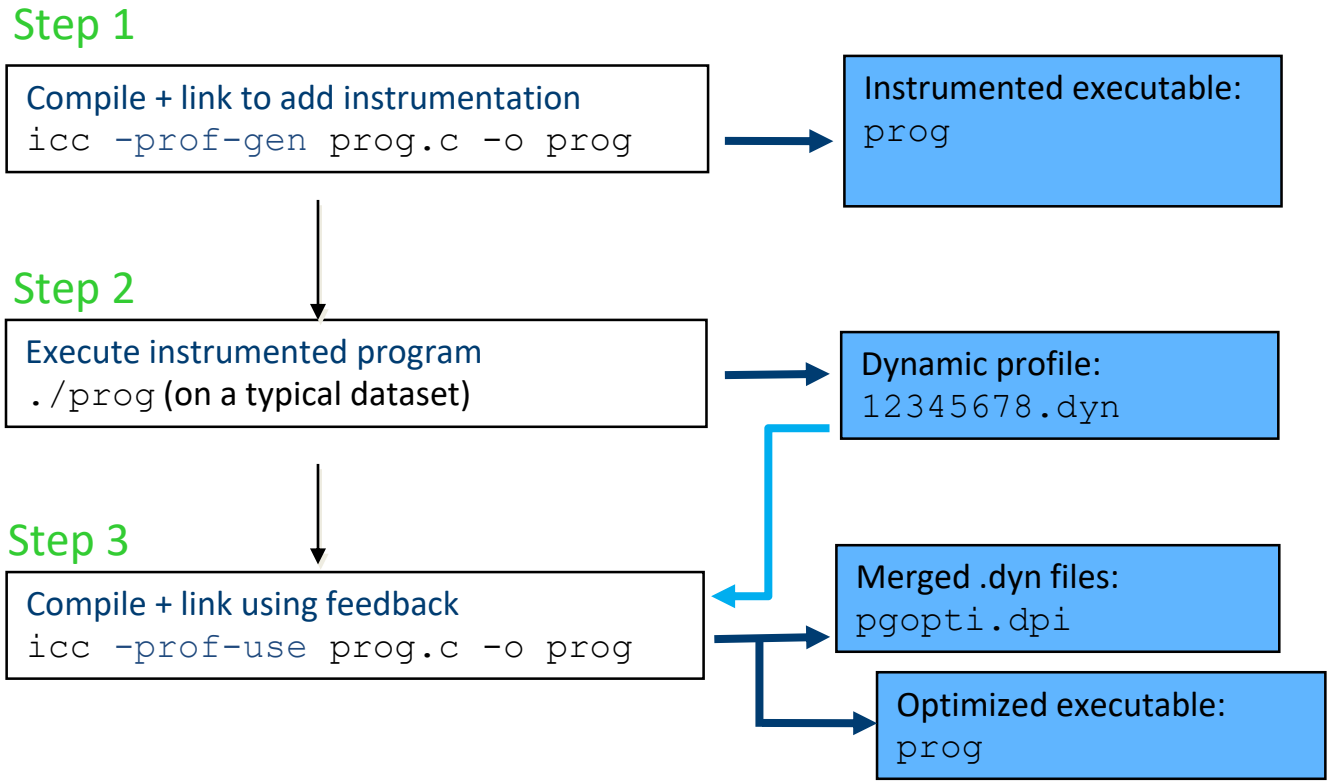
Use execution-time feedback to guide (final) optimization
Enhancements with PGO:

- More accurate branch prediction
- Basic block movement to improve instruction cache behavior
- Better decision of functions to inline (help IPO)
- Can optimize function ordering
- Switch-statement optimization
- Better vectorization decisions



Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

PGO Usage: Three Step Process



Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Auto-Parallelization

Based on OpenMP* runtime

Compiler automatically translates loops into equivalent multithreaded code with using this option:

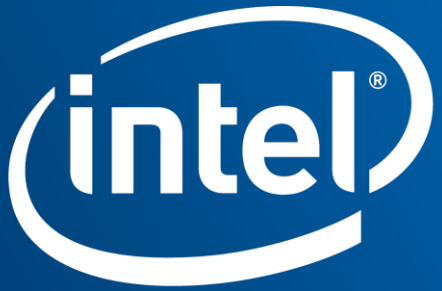
```
-parallel
```

The auto-parallelizer detects simply structured loops that may be safely executed in parallel, including loops implied by Intel® Cilk™ Plus array notation, and automatically generates multi-threaded code for these loops.

The auto-parallelizer report can provide information about program sections that were parallelized by the compiler. Compiler switch:

```
-qopt-report-phase=par
```

Optimization Notice
Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Software