ABOUT INTEL COMPILER

INTEL COMPILER OPTIMIZATIONS

Wang Yang

Intel® Developer Products Division

Sep 2017

Intel Compilers

- Over 20 years developments
- Windows, Linux, Mac, Android, Embedded OS support
- Intel C++ Compiler 18.0 was just launched in September 2017
- Widely used in various industries

- Why use Intel compilers?

Why Use Intel® C++ Compilers?

Compatibility

- Provides language conformance
- C99, C++11 and C++ 14 new features support
- Compatible with MS(windows) & GNU (Linux) compiler
  - Compilation option support
  - Mixing and matching binary files created

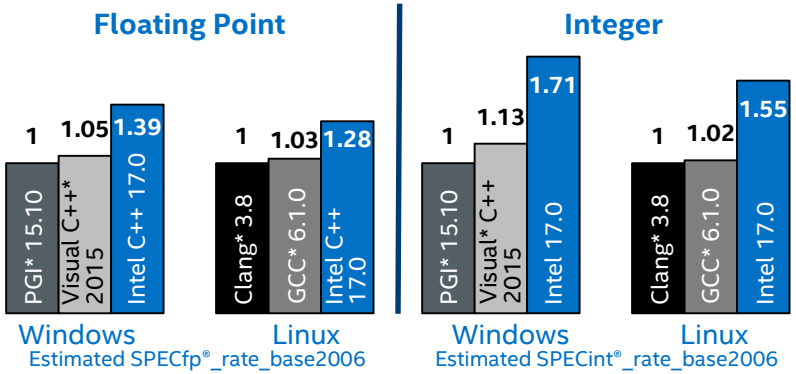
Why Use Intel® C++ Compilers?

Performance Gained in a variety of ways:

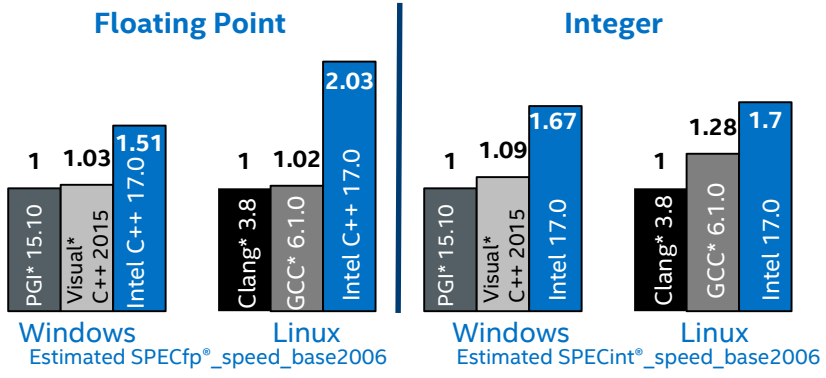
- New instructions enable new opportunities (SSE, AVX, AVX2, AVX512)
- Be on the cutting edge of new performance features
  - Latest Instructions support
  - Code generation tuned for latest micro-architecture
- Highly Optimized libraries
- Advanced optimization features and optimization reports

Intel® C++ Compilers-Performance Advantage as measured by SPEC*

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



Relative geomean performance, SPEC* benchmark - higher is better



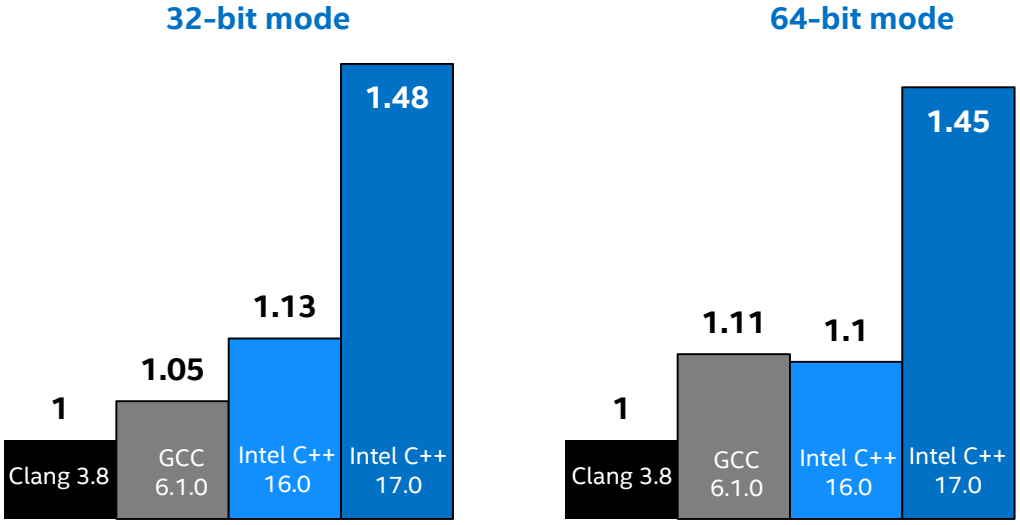
Relative geomean performance, SPEC* benchmark - higher is better

Configuration: Windows hardware: Intel(R) Xeon(R) CPU E3-1245 v5 @ 3.50GHz, HT enabled, TB enabled, 32 GB RAM; Linux hardware: Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, 256 GB RAM, HyperThreading is on.
Software: Intel compilers 17.0, Microsoft (R) C/C++ Optimizing Compiler Version 19.00.23918 for x86/x64, GCC 6.1.0. PGI 15.10, Clang/LLVM 3.8
Linux OS: Red Hat Enterprise Linux Server release 7.1 (Maipo), kernel 3.10.0-229.el7.x86_64. Windows OS: Windows 10 Pro (10.0.10240 N/A Build 10240).
SPEC* Benchmark (www.spec.org). SmartHeap libs 11.3 for Visual® C++ and Intel Compiler were used for SPECint® benchmarks.

Configuration: Windows hardware: Intel(R) Xeon(R) CPU E3-1245 v5 @ 3.50GHz, HT enabled, TB enabled, 32 GB RAM; Linux hardware: Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, 256 GB RAM, HyperThreading is on.
Software: Intel compilers 17.0, Microsoft (R) C/C++ Optimizing Compiler Version 19.00.23918 for x86/x64, GCC 6.1.0. PGI 15.10, Clang/LLVM 3.8
Linux OS: Red Hat Enterprise Linux Server release 7.1 (Maipo), kernel 3.10.0-229.el7.x86_64. Windows OS: Windows 10 Pro (10.0.10240 N/A Build 10240).
SPEC* Benchmark (www.spec.org).

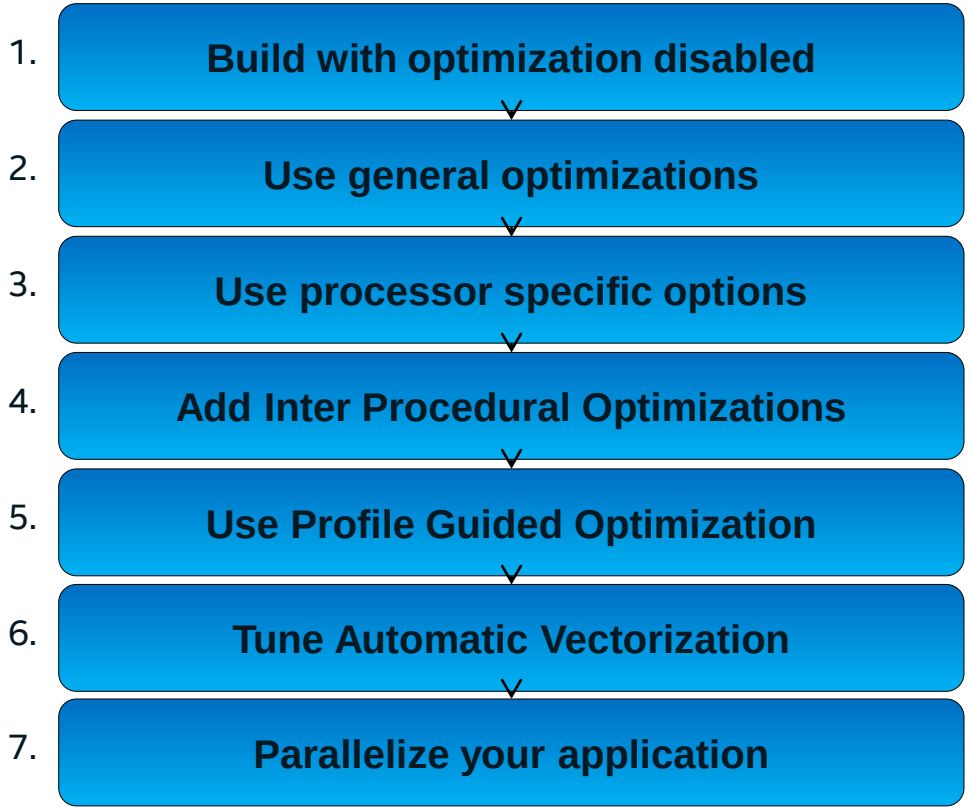
Intel® C++ Compilers deliver Impressive Performance on Embedded Applications powered by Intel® Xeon® processors

Coremark Pro* benchmarks running on Intel Xeon E3 processors



Configuration: Intel Xeon CPU E3-1245 v5 @ 3.50GHz, HT enabled, TB enabled. Software: Intel C++ compiler 17.0, Intel C++ compiler 16.0, GCC 6.1.0, Clang/LLVVM 3.8. Linux OS: Red Hat Enterprise Linux 7.2, Kernel 3.10.0-327.4.5.el7.x86_64, 32GB RAM. Coremark Pro* Benchmark (www.eembc.org).Compiler flags: Intel C++ 17.0: -O3 -no-prec-div -ipo -xCORE-AVX2; Intel C++ 16.0: -O3 -no-prec-div -ipo -xCORE-AVX2; GCC 6.1.0: -Ofast -mfpmath=sse -flto -march=native -funroll-loops -ffat-lto-objects; Clang/LLVM 3.8: -Ofast -mfpmath=sse -flto -march=native -funroll-loops –static. GCC and clang/LLVM 32bit modes have additional flag –m32. - Software & 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 & performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. 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 & 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 & Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804 .

General Steps of Optimization



Intel® VTune™ Amplifier is recommended for further performance profiling and analysis.

Processor specific optimizations

Processor specific extensions switches:

`/arch:<target>` (Microsoft compatible), `-march=<target>` (Linux*, OS X*)

- No Intel processor check
- No Intel specific optimizations
- Code runs only on <target>

`/tune:<target>` (Windows*), `-mtune=<target>`(Linux*, OS X*)

- Instruction generated that run fastest on the specific CPU
- **Can run on other processors**

`/Qx<target>,-x<target>`

- Intel specific optimizations
- Processor-check added to main-program
- Generates optimized code targeted for execution on the system you compile on

`/Qax<target>,-ax<target>`

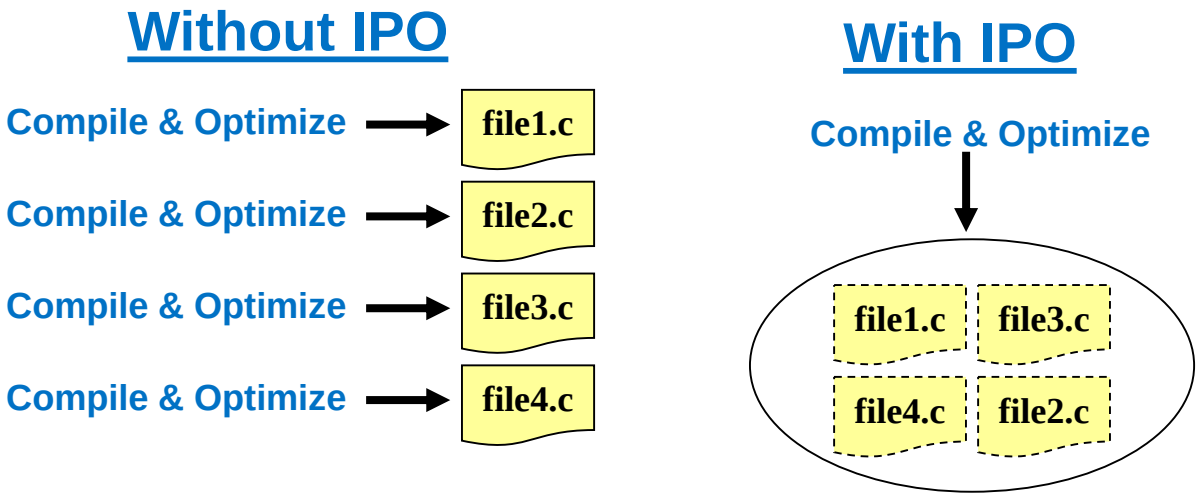
- Intel specific optimizations
- Autodispatch switch a for generating one or more additional optimized code paths

`/QxHost, -xHost`

- generate instructions for the highest instruction set available on the compilation host processor

Interprocedural Optimizations

Extends optimizations across file boundaries



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

IPO - Interprocedural Optimizations

- 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

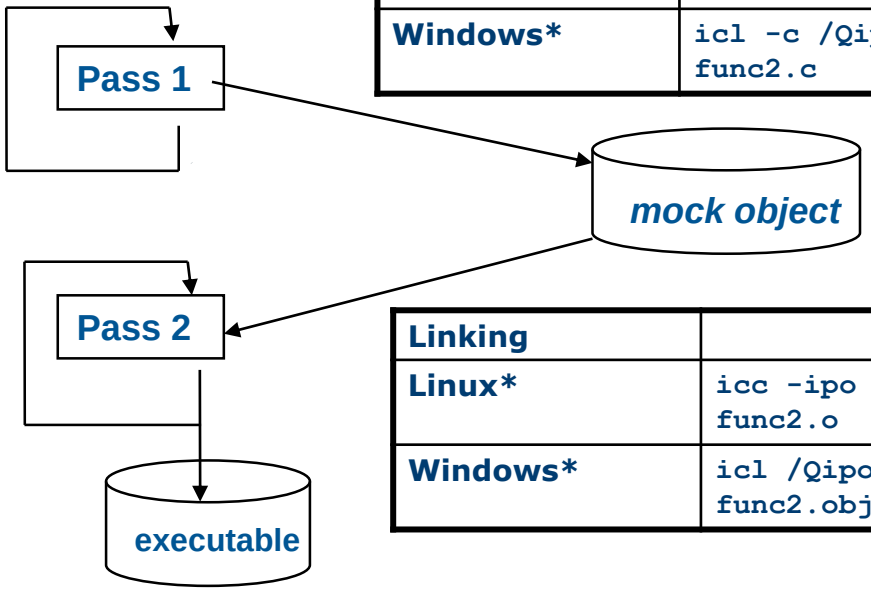
Windows*	Linux*
/Qip	-ip
/Qipo	-ipo

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

Interprocedural Optimizations (IPO)

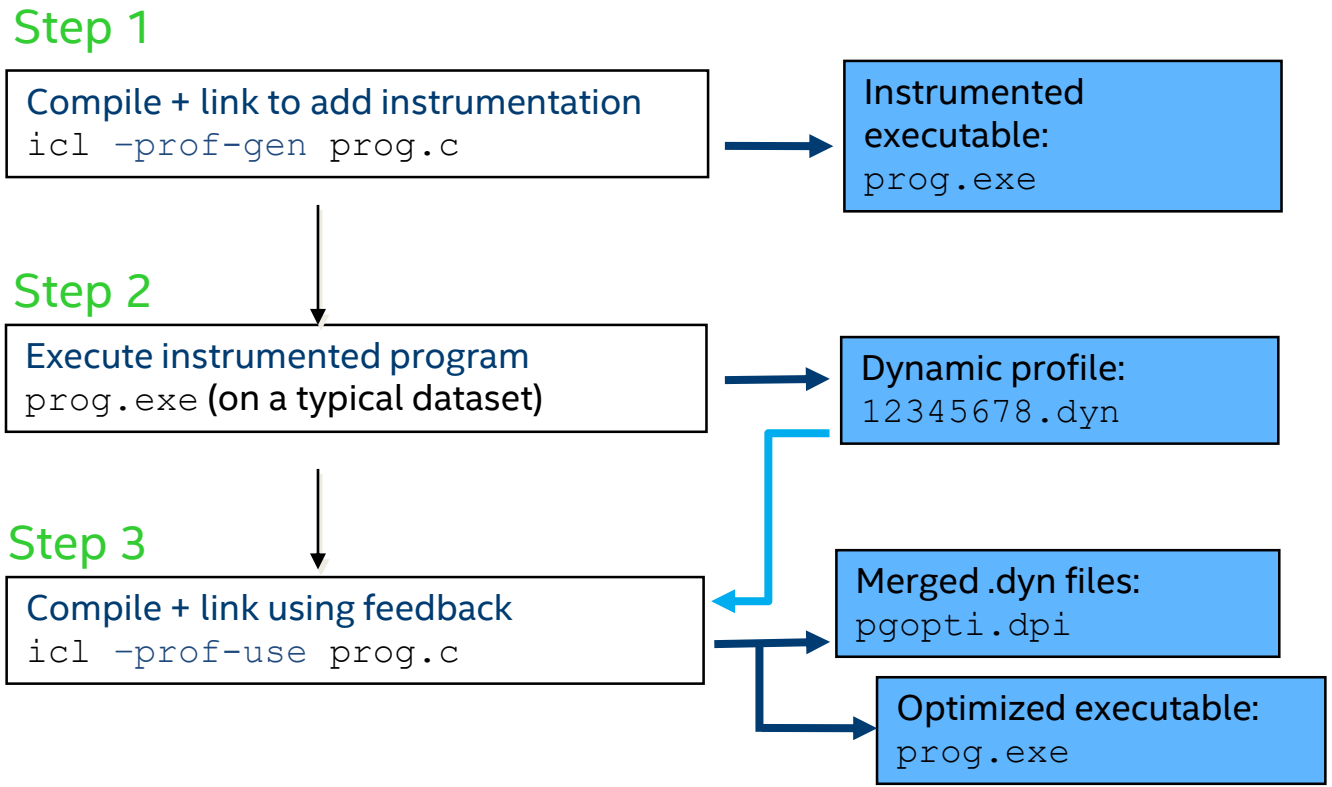
Usage: Two-Step Process



Compiling	
Linux*	`icc -c -ipo main.c func1.c func2.c`
Windows*	`icl -c /Qipo main.c func1.c func2.c`

Linking	
Linux*	`icc -ipo main.o func1.o func2.o`
Windows*	`icl /Qipo main.o func1.o func2.obj`

PGO - Profile-Guided Optimizations



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

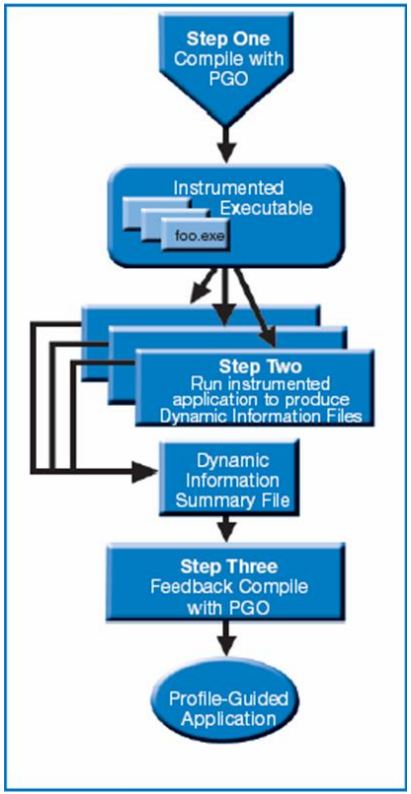
```
if (x > y)
        do_this();
else
        do that();
```

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

Use execution-time feedback to guide (final) optimization

Enhancements with PGO:

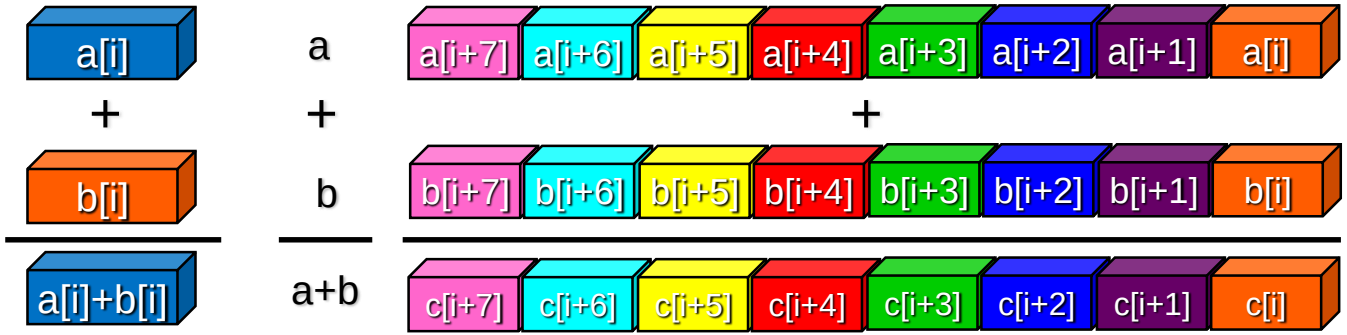
- More accurate branch prediction
- Basic block movement to improve instruction cache behavior
- Better decision of functions to inline (help IPO)
- Can optimize function ordering
- Switch-statement optimization
- Better vectorization decisions



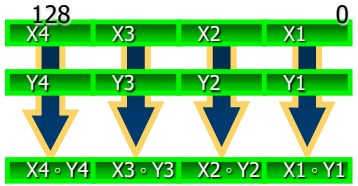
Auto-Vectorization: Single Instruction Multiple Data

- Scalar mode
  - one instruction produces one result
- SIMD processing
  - with SSE or AVX instructions
  - one instruction can produce multiple results

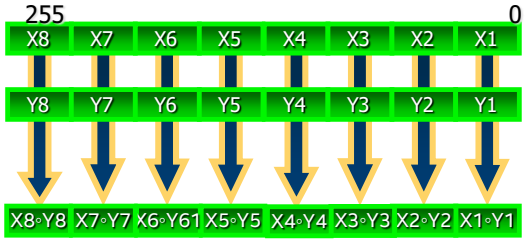
```
for (i=0;i<=MAX;i++)
    c[i]=a[i]+b[i];
```



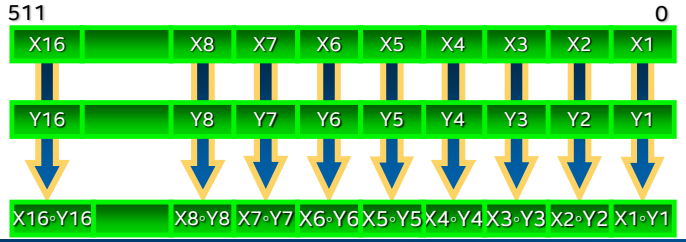
Vectorization is Achieved through SIMD Instructions & Hardware



Intel® SSE
Vector size: 128bit
Data types:
8,16,32,64 bit integers
32 and 64bit floats
VL: 2, 4, 8, 16



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



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

Xi, Yi & results 32-bit integer

Many Ways to Vectorize

Compiler:
Auto-vectorization (no change of code)

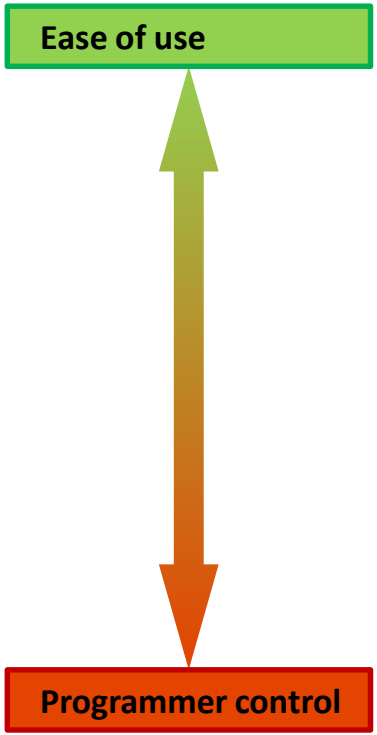
Compiler:
Auto-vectorization hints (`#pragma vector`, …)

Compiler:
Intel® Cilk™ Plus Array Notation Extensions

SIMD intrinsic class
(e.g.: `F32vec`, `F64vec`, …)

Vector intrinsic
(e.g.: `_mm_fmadd_pd(…)`, `_mm_add_ps(…)`, …)

Assembler code
(e.g.: `[v]addps`, `[v]addss`, …)



Processor Specific Optimizations Enables Auto Vectorization

Feature	SIMD Extension	Compiler Options
Intel® Streaming SIMD Extensions 2 (Intel® SSE2) as available in initial Pentium® 4 or compatible non-Intel processors	sse2	-xSSE2
Intel® Streaming SIMD Extensions 3 (Intel® SSE3) as available in Pentium® 4 or compatible non-Intel processors	sse3	-xSSE3
Supplemental Streaming SIMD Extensions 3 (SSSE3) as available in Intel® Core™2 Duo processors	ssse3	-xSSSE3
Intel® SSE4.1 as first introduced in Intel® 45nm Hi-K next generation Intel Core™ micro-architecture	sse4.1	-xSSE4.1
Intel® SSE4.2 Accelerated String and Text Processing instructions supported first by Intel® Core™ i7 processors	sse4.2	-xSSE4.2
Like SSSE3 but optimizes for the Intel® Atom™ processor and Intel® Centrino® Atom™ Processor Technology	atom_ssse3	-xATOM_SSSE3
Like SSE4.2 but optimized for Intel® Atom™ processors that support Intel® SSE4.2 and MOVBE instructions.	atom_sse4.2	-xATOM_SSE4.2
Intel® Advanced Vector Extensions (Intel® AVX) as available in 2nd generation Intel® Core™ processor family	avx	-xAVX
Intel® Advanced Vector Extension (Intel® AVX) including instructions offered by the 3rd generation Intel® Core processor	core-avx-i	-xCORE-AVX-I
Intel® Advanced Vector Extension 2 (Intel® AVX2) as provided by a 4th generation Intel® Core processor	core-avx2	-xCORE-AVX2

Processor Specific Optimizations Enables Auto Vectorization

- Intel Compiler will auto vectorize the source if it can
- Use –O2 or –O3 plus relevant options
- Pros:
  - Minimal effort required
  - Maintainable – source code is not changed
  - Portable across Intel SIMD architectures
  - Optimal performance is possible in best cases
  - Scales forward!
- Cons:
  - Compiler is conservative; will not generate unsafe code

Guidelines for writing vectorizable Code

- an inner loop
- Straight-line code
- Avoid:
  - Function/subroutine calls (unless inlined)
  - Non-mathematical operators (sqrt, sin, exp,… OK)
  - Data-dependent loop exit conditions
    - Iteration count must be known at entry to loop
  - Loop carried data dependencies
    - Reduction loops are OK
  - Non-contiguous data (indirect addressing; non-unit stride) - inefficient

- Directives/pragmas can help:
  - #pragma ivdep

    ignores potential dependencies
  - #pragma vector always

    ignores efficiency heuristics
  - #pragma aligned

    assume data aligned
- Compiler can generate runtime alignment and dependency tests for simple loops (but less efficient)

SIMD Data Layout Template -

Improve productivity and boost C++ performance

- Quickly convert “Array of Structures” to “Structure of Arrays” representation.
- Increase productivity: Use predefined templates with minimal effort, and let SDLT do the vecorization for you.
- Improve performance: SDLT vectorizes your code by making memory access contiguous, which can lead to more efficient code and better performance.
- Seamless integration: SDLT follows the familiar Intel vector programming model.

```
struct Point3s
{
    float x;
    float y;
    float z;
    // helper methods
};

std::vector<Point3s> inputDataSet(count);
std::vector<Point3s> outputDataSet(count);

for(int i=0; i < count; ++i) {
  Point3s inputElement = inputDataSet[i];
  Point3s result = // transformation of inputElement that is independent of other iterations
                   // can keep algorithm high level using object helper methods
  outputDataSet[i] = result;
}
```

```
SDLT_PRIMITIVE(Point3s, x, y, z)

sdlt::soa1d_container<Point3s> inputDataSet(count);
sdlt::soa1d_container<Point3s> outputDataSet(count);

auto inputData = inputDataSet.const_access();
auto outputData = outputDataSet.access();

#pragma forceinline recursive
#pragma omp simd
for(int i=0; i < count; ++i) {
  Point3s inputElement = inputData[i];
  Point3s result = // transformation of inputElement that is independent of other iterations
                   // can keep algorithm high level using object helper methods
  outputData[i] = result;
}
```

Compiler Optimization Reports

- Enables the optimization report and controls the level of details
  - `/Qopt-report[:n], -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-file:<filename>, -qopt-report-file=<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], -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.

Get Intel Compilers

Find out more at: ***https://software.intel.com/en-us/c-compilers***

Academic license:

https://software.intel.com/en-us/qualify-for-free-software/student

Contact us: ***https://software.intel.com/en-us/forums/intel-c-compiler***

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.

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


intel®
experience
what's inside™