Intel Compilers

Igor Vorobtsov, Technical Consulting Engineer

intel®

Agenda

- Key Updates
- Common Optimizations
  - High-Level Optimizations
  - InterProcedural Optimizations
  - Profile-Guided Optimizations
  - Vectorization
  - Auto-Parallelization
  - FP model
- New Compilers
  - What Is Data Parallel C++?
  - Intel® Compilers for OpenMP

Let's Get Started!

Key Updates

Key Knowledge for Intel® Compilers Going Forward

- New Underlying Back End Compilation Technology based on LLVM
- New compiler technology available in BETA today in oneAPI Beta for DPC++, C++ and Fortran
- Existing Intel proprietary “IL0” (ICC, IFORT) Compilation Technology compilers provided alongside new compilers
  - *CHOICE! Continuity!*
- *BUT Offload (DPC++ or OpenMP TARGET) supported only with new LLVM-based compilers*

C++ New Features – ICX

- What is this?
- Close collaboration with Clang*/LLVM* community
- ICX is Clang front-end (FE), LLVM infrastructure
  - PLUS Intel proprietary optimizations and code generation
- Clang FE pulled down frequently from open source, kept current
  - Always up to date in ICX
  - We contribute! Pushing enhancements to both Clang and LLVM
- Enhancements working with community – better vectorization, opt-report, for example

5

Packaging of Compilers

- Parallel Studio XE 2020 Production Compiler for Today
  - Drivers: icc, icpc, ifort
  - v19.1 Compiler versions; 19.1 branch

- oneAPI Base Toolkit(BETA) *PLUS* oneAPI HPC Toolkit(BETA)
  - Existing IL0 compilers ICC, ICPC, IFORT in HPC Toolkit
    - v2021.1 code base for IL0 compilers
  - ADDED! New compilers based on LLVM* framework
    - Drivers: icx, ifx and dpcpp
    - v2021.1 code base for LLVM-based compilers

Let's Get Started!

Common Optimizations

icc/ifort

intel®

What's New for Intel compilers 19.1?

icc/ifort

Advance Support for Intel® Architecture – Use Intel compiler to generate optimized code for Intel Atom® processor through Intel® Xeon® Scalable processor families

Achieve Superior Parallel Performance – Vectorize & thread your code (using OpenMP*) to take full advantage of the latest SIMD-enabled hardware, including Intel® Advanced Vector Extensions 512 (Intel® AVX-512)

What's New in C++

Initial C++20, and full C++ 17 enabled

- Enjoy advanced lambda and constant expression support
- Standards-driven parallelization for C++ developers

Initial OpenMP* 5.0, and full OpenMP* 4.5 support

- Modernize your code by using the latest parallelization specifications

What's New in Fortran

Substantial Fortran 2018 support

- Enjoy enhanced C-interoperability features for effective mixed language development
- Use advanced coarray features to parallelize your modern Fortran code

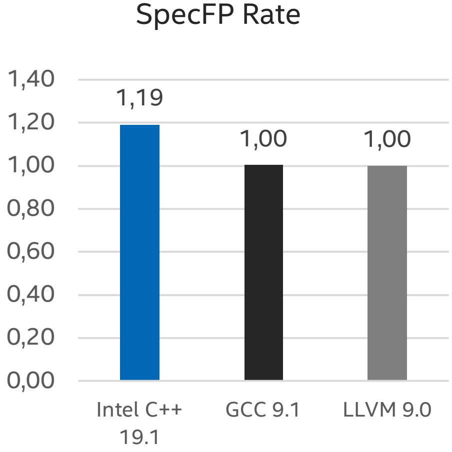
Initial OpenMP* 5.0, and substantial OpenMP* 4.5 support

- Customize your reduction operations by user-defined reductions

Intel® C++ Compiler Boosts Application Performance on Linux*

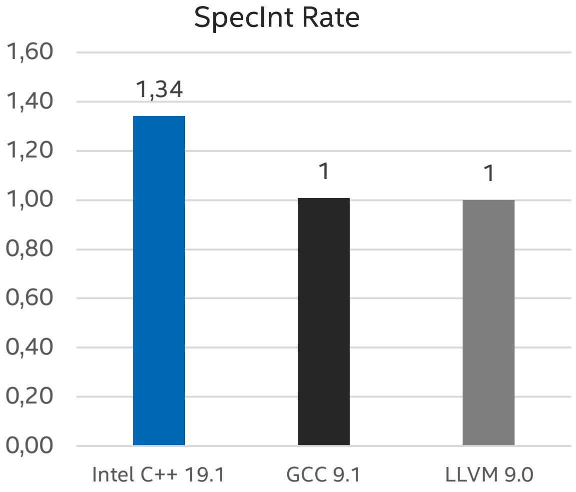
Relative geomean performance (FP Rate Base and FP Speed Base; higher is better)

Floating Point



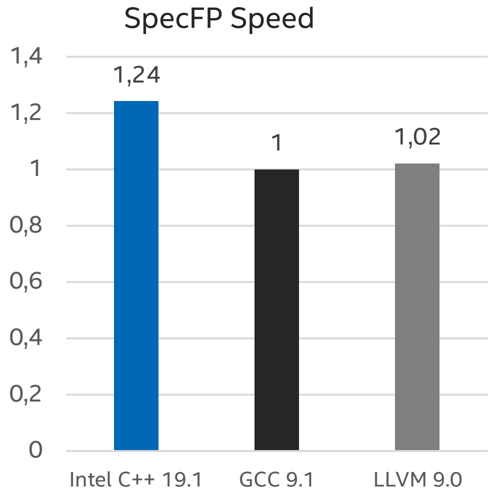
Estimated geometric mean of SPEC* CPU2017 Floating Point **RATE BASE** C/C++ benchmarks

Integer



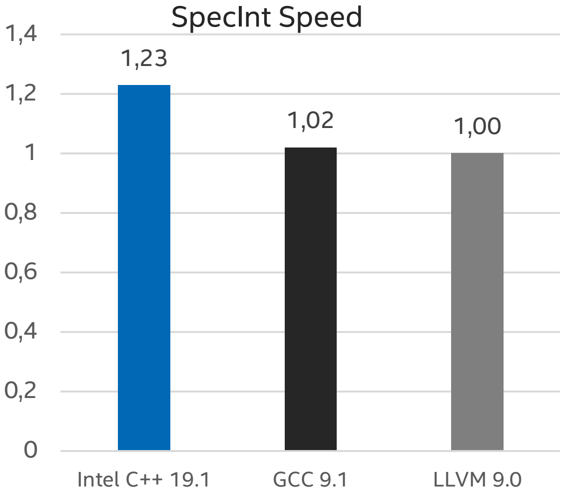
Estimated SPECint®**_rate_base2017**

Floating Point



Estimated geometric mean of SPEC* CPU2017 Floating Point **SPEED BASE** C/C++ benchmarks

Integer



Estimated SPECint®**_speed_base2017**

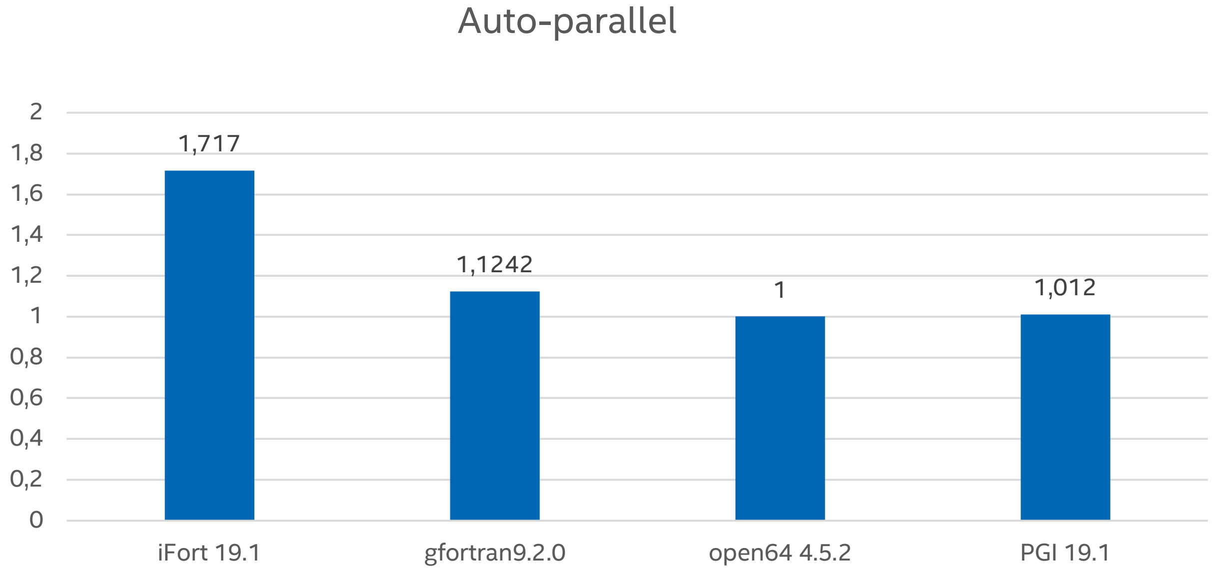
Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer. Performance results are based on testing as of Aug. 26, 2019 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure. 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 about performance and benchmark results, visit www.intel.com/benchmarks.

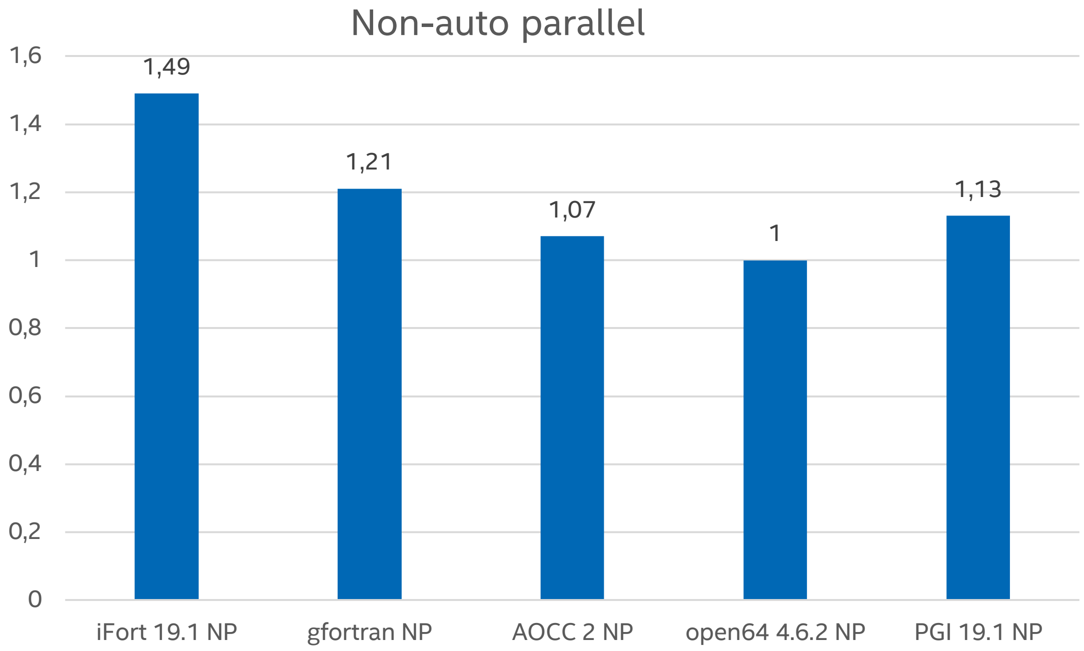
Configuration: Testing by Intel as of Aug. 26, 2019. Linux hardware: Intel® Xeon® Platinum 8180 CPU @ 2.50GHz, 384 GB RAM, HyperThreading is on. Software: Intel® C++ Compiler 19.1, GCC 9.1.0. Clang/LLVM 9.0. Linux OS: Red Hat* Enterprise Linux Server release 7.4 (Maipo), 3.10.0-693.el7.x86_64. SPEC* Benchmark (www.spec.org). SPECint®_rate_base_2017 compiler switches: qkmalloc was used for Intel C++ Compiler 19.1 SPECint rate test, jemalloc 5.0.1 was used for GCC and Clang/LLVM SPECint rate test. Intel® C Compiler / Intel C++ Compiler 19.1: -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-mem-layout-trans=4. GCC 9.1.0 -march=skylake-avx512 -mfpmath=sse -Ofast -funroll-loops -flto. Clang 9.0: -march=skylake-avx512 -mfpmath=sse -Ofast -funroll-loops –flto. SPECfp®_rate_base_2017 compiler switches: jemalloc 5.0.1 was used for Intel C++ Compiler 19.1, GCC and Clang/LLVM SPECfp rate test. Intel C/C++ compiler 19.1: -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -ffinite-math-only -qopt-mem-layout-trans=4. GCC 9.1.0: -march=skylake-avx512 -mfpmath=sse -Ofast -fno-associative-math -funroll-loops -flto. Clang 9.0: -march=skylake-avx512 -mfpmath=sse -Ofast -funroll-loops –flto. SPECint®_speed_base_2017 compiler switches: Intel C Compiler / Intel C++ Compiler 19.1: -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-mem-layout-trans=4 -qopenmp. GCC 9.1.0: -march=skylake-avx512 -mfpmath=sse -Ofast -funroll-loops -flto -fopenmp. Clang 9.0: -march=skylake-avx512 -mfpmath=sse -Ofast -funroll-loops -flto -fopenmp=libomp. SPECfp®_speed_base_2017 compiler switches: Intel C Compiler / Intel C++ Compiler 19.1: -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -ffinite-math-only -qopenmp. GCC 9.1.0: -march=skylake-avx512 -mfpmath=sse -Ofast -fno-associative-math -funroll-loops -flto -fopenmp. Clang 9.0: -march=skylake-avx512 -mfpmath=sse -Ofast -funroll-loops -flto -fopenmp=libomp. compiler switches: jemalloc 5.0.1 was used for Intel C++ Compiler 19.0 update 4, GCC and Clang/LLVM SPECfp rate test. Intel C/C++ compiler 19.1:-.

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® Fortran Compiler Boosts Application Performance on Linux*

Polyhedron* Benchmark
Estimated relative geomean performance - higher is better





Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer. Performance results are based on testing as of Dec. 12, 2019 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure. 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.

Configuration: Testing by Intel as of Dec. 12, 2019. Hardware: Intel® Core™ i7-6700 CPU @ 3.40GHz, HyperThreading is on. RAM 8G. Software: Intel® Fortran Compiler 19.1, PGI Fortran* 19.1, Open64* 4.6.2, gFortran* 9.2.0. Red Hat Enterprise Linux Server release 7.4 (Maipo), Kernel 3.10.0-693.11.6.el7.x86_64 Polyhedron Fortran Benchmark (www.fortran.uk). Linux compiler switches: Auto-parallel: Gfortran:gfortran -Ofast -mfpmath=sse -flto -march=skylake -funroll-loops -ftree-parallelize-loops=8. Intel Fortran compiler: -fast -parallel -xCORE-AVX2 -nostandard-realloc-lhs. PGI Fortran: pgf95 -fast -Mipa=fast,inline -Msmartalloc -Mfprelaxed -Mstack_arrays -Mconcur –mp=bind. Open64: openf95 -march=auto -Ofast -mso -apo. Non-auto parallel (NP): ifort -fast -xCORE-AVX2 -nostandard-realloc-lhs. open64:openf95 -march=auto -Ofast -mso. gcc: gfortran -Ofast -mfpmath=sse -flto -march=native -funroll-loops. pgi: pgf95 -fast -Mipa=fast,inline -Msmartalloc -Mfprelaxed -Mstack_arrays. aocc: flang -flto -Wl,-mllvm -Wl,-function-specialize -Wl,-mllvm -Wl,-region-vectorize -Wl,-mllvm -Wl,-reduce-array-computations=3 -ffast-math -Wl,-mllvm -Wl,-inline-recursion=4 -Wl,-mllvm -Wl,-lsr-in-nested-loop -Wl,-mllvm -Wl,-enable-iv-split -O3 -flto -march=znver2 -funroll-loops -Mrecursive -mllvm -vector-library=LIBMVEC -z muldefs -lamdlibm -lflang -lamdlibm -lm

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

https://tinyurl.com/icc-user-guide

High-Level Optimizations

Basic Optimizations with icc -O…

-O0 no optimization; sets -g for debugging

-O1 scalar optimizations

excludes optimizations tending to increase code size

-O2 **default** for icc/icpc (except with -g)

includes **auto-vectorization**; some loop transformations, e.g. unrolling, loop interchange;
inlining within source file;
start with this (after initial debugging at -O0)

-O3 more aggressive loop optimizations

including cache blocking, loop fusion, prefetching, …
suited to applications with loops that do many floating-point calculations or process large data sets

InterProcedural Optimizations (IPO)

Multi-pass Optimization

`icc -ipo`

Analysis and optimization across function and/or source file boundaries, e.g.

- Function inlining; constant propagation; dependency analysis; data & code layout; etc.

2-step process:

- Compile phase – objects contain intermediate representation
- “Link” phase – compile and optimize over all such objects
- Seamless: linker automatically detects objects built with -ipo and their compile options
- May increase build-time and binary size
- But build can be parallelized with `-ipo=n`
- Entire program need not be built with IPO, just hot modules

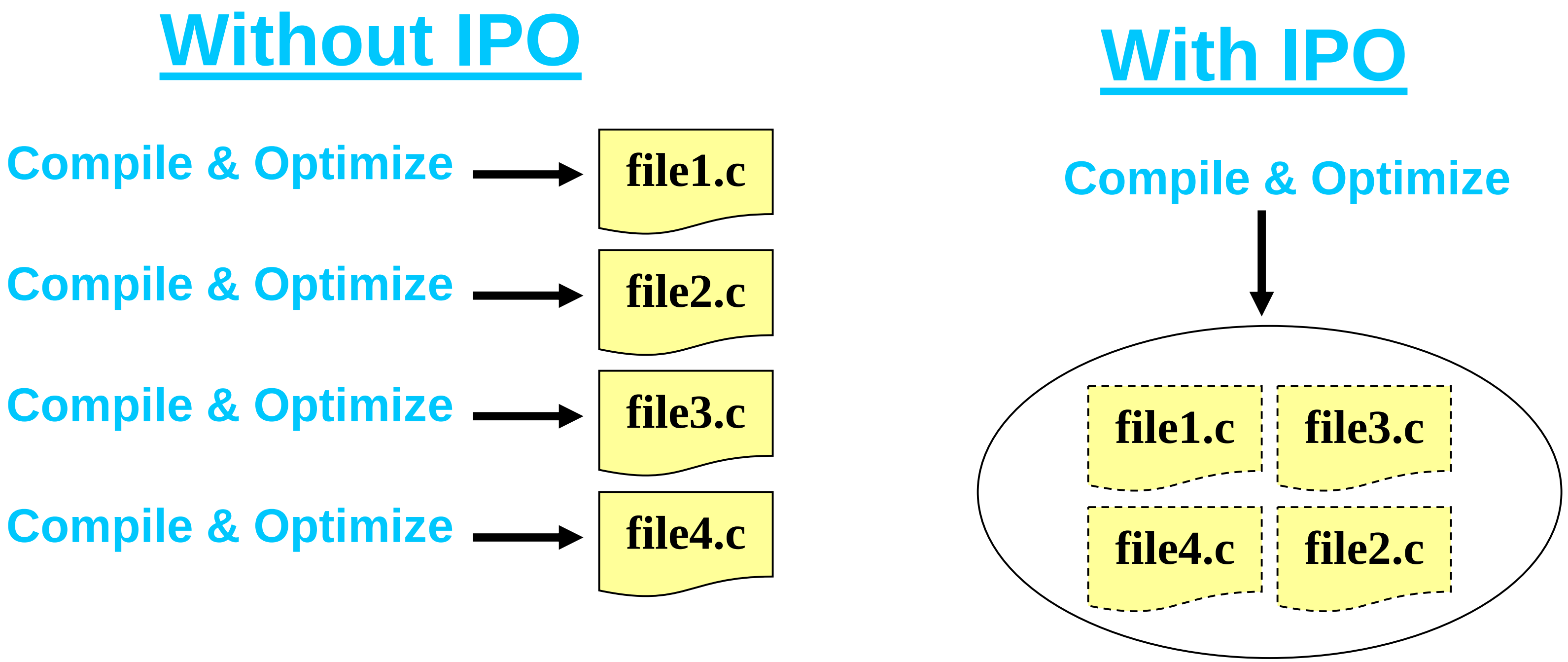
Particularly effective for applications with many smaller functions

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

InterProcedural Optimizations

Extends optimizations across file boundaries

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



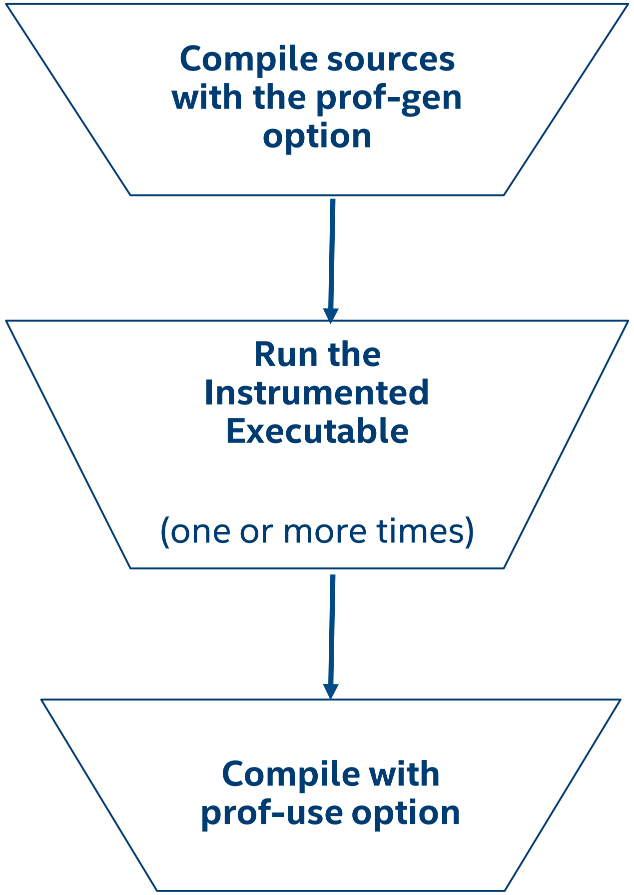
Profile-Guided Optimizations (PGO)

- 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



PGO Usage: Three-Step Process

Step 1

Compile + link to add instrumentation

`icc -prof-gen prog.c -o prog`

Instrumented executable:

`prog`

Step 2

Execute instrumented program

`./prog` (on a typical dataset)

Dynamic profile:
`12345678.dyn`

Step 3

Compile + link using feedback

`icc -prof-use prog.c -o prog`

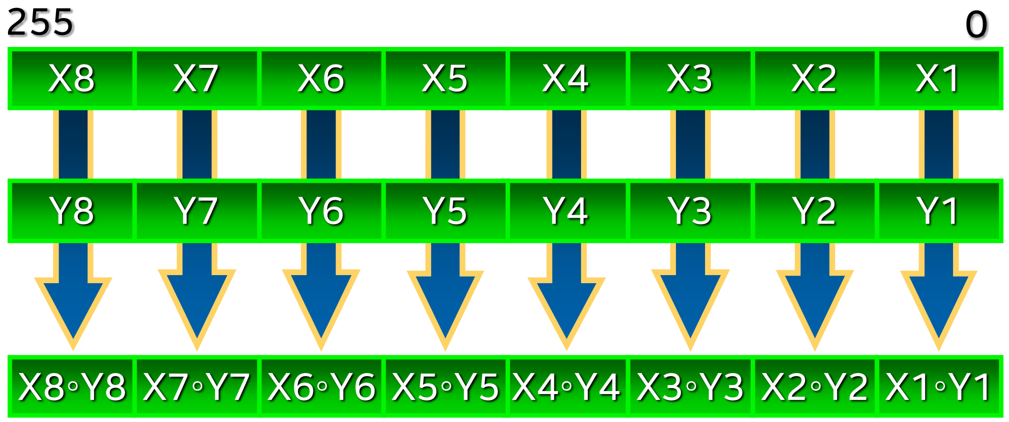
Merged .dyn files:
`pgopti.dpi`

Optimized executable:
`prog`

Math Libraries

- icc comes with Intel's optimized math libraries
  - libimf (scalar) and libsvml (scalar & vector)
  - Faster than GNU* libm
  - Driver links libimf automatically, ahead of libm
  - Additional functions (replace math.h by mathimf.h)
- Don't link to libm explicitly! 🚫 -lm 🚫
  - May give you the slower libm functions instead
  - Though the Intel driver may try to prevent this
  - gcc needs -lm, so it is often found in old makefiles

SIMD Types for Intel® Architecture



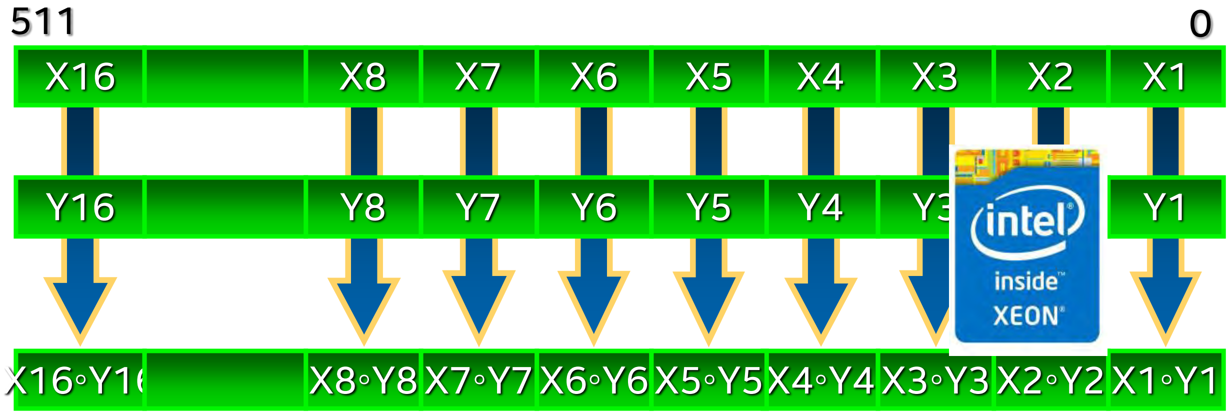
AVX

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**

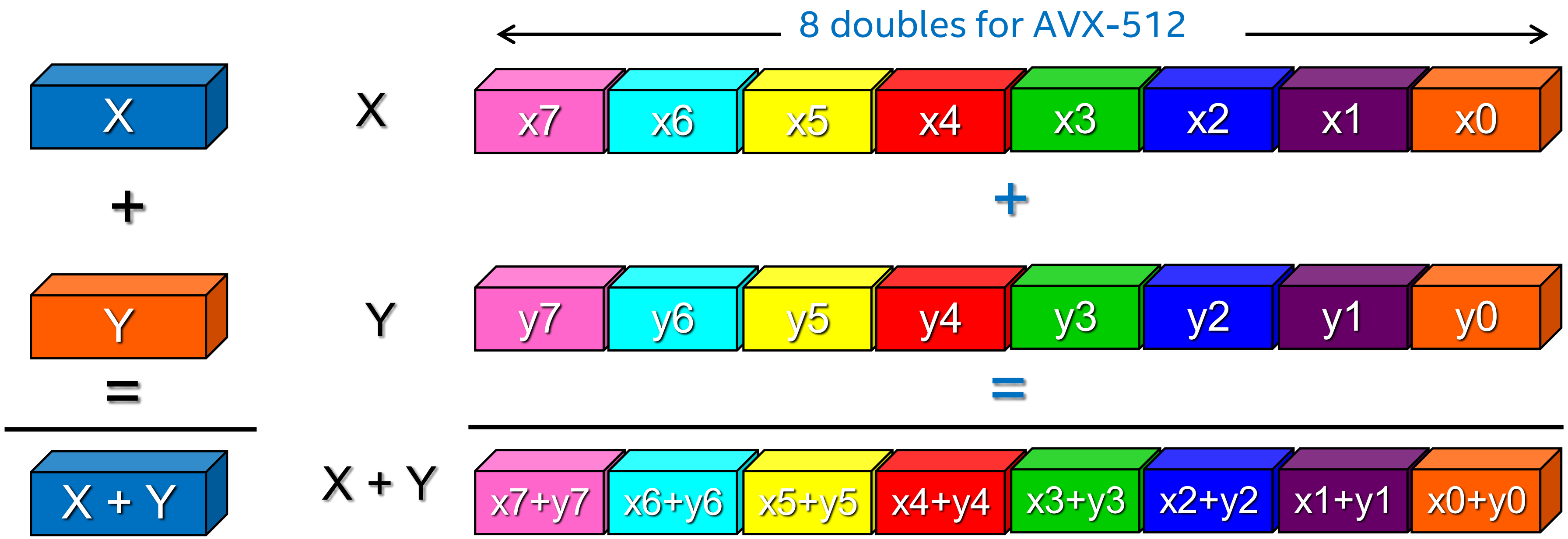
Data types:

- 8, 16, 32, 64 bit integer
- 32 and 64 bit float

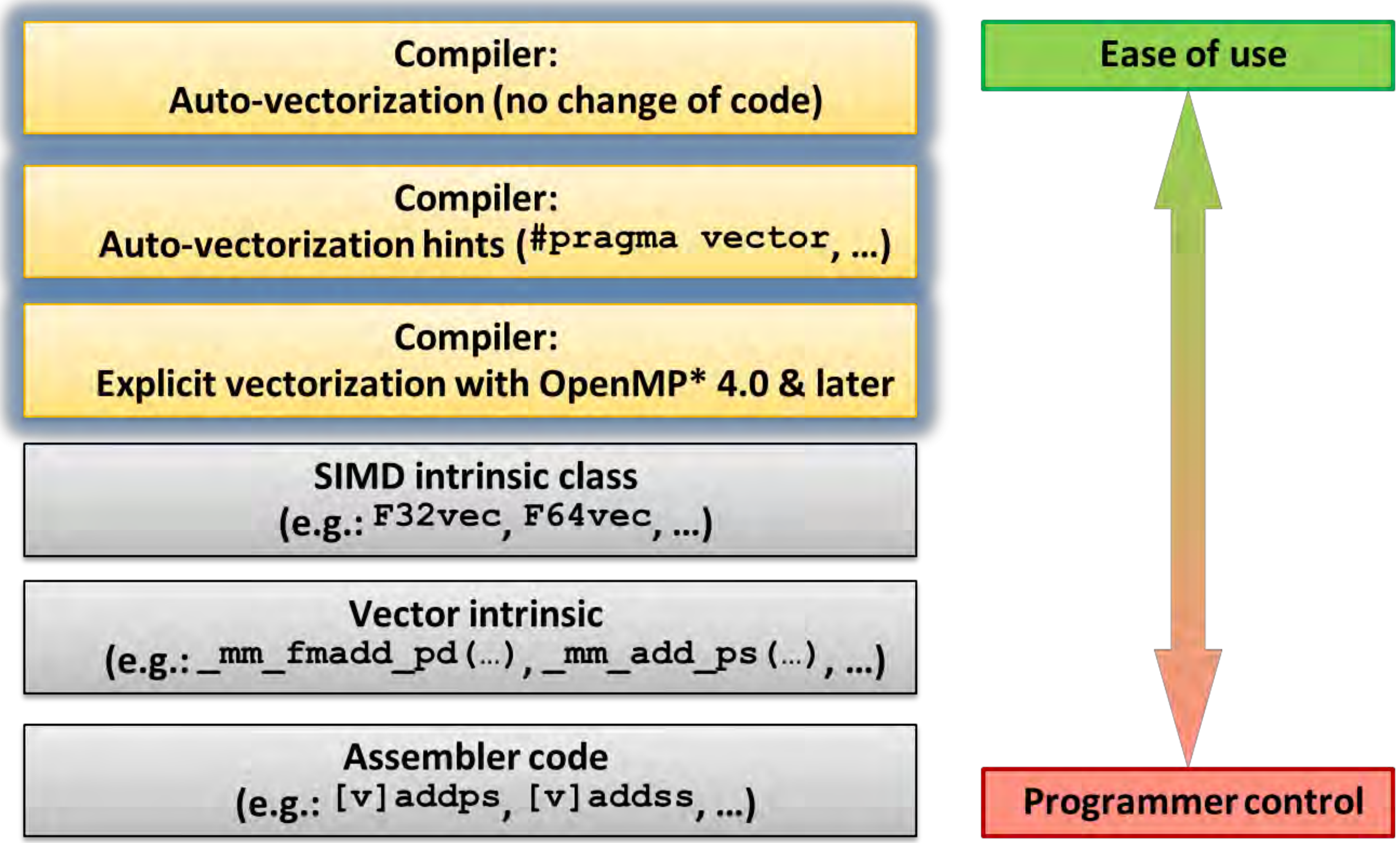
SIMD: Single Instruction, Multiple Data

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

- Scalar mode
  - one instruction produces one result
  - E.g. vaddss, (vaddsd)
- Vector (SIMD) mode
  - one instruction can produce multiple results
  - E.g. vaddps, (vaddpd)



Many ways to vectorize



Basic Vectorization Switches I

-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: CORE-AVX512, CORE-AVX2, CORE-AVX-I, AVX, SSE4.2, etc.
- Example:

```
icc -xCORE-AVX2 test.c
ifort –xSKYLAKE test.f90
```

Basic Vectorization Switches II

-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** (**/arch:sse2**)
- The baseline code path can be modified by **–m<code>** or **–x<code>**
- Example: `icc -axCORE-AVX512 -xAVX test.c`

  `icc -axCORE-AVX2,CORE-AVX512 test.c`

-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

- **-xHost**

Compiler Reports – Optimization Report

- -qopt-report[=n]: tells the compiler to generate an optimization report

  n: (Optional) Indicates the level of detail in the report. You can specify values 0 through 5. If you specify zero, no report is generated. For levels n=1 through n=5, each level includes all the information of the previous level, as well as potentially some additional information. Level 5 produces the greatest level of detail. If you do not specify n, the default is level 2, which produces a medium level of detail.

- -qopt-report-phase[=list]: specifies one or more optimizer phases for which optimization reports are generated.

  loop: the phase for loop nest optimization
  vec: the phase for vectorization
  par: the phase for auto-parallelization
  all: all optimizer phases

- -qopt-report-filter=string: specified the indicated parts of your application, and generate optimization reports for those parts of your application.

Optimization Report – An Example

$ icc -c -xcommon-avx512 **-qopt-report=3** -qopt-report-phase=loop,vec foo.c

Creates foo.optrpt summarizing which optimizations the compiler performed or tried to perform.
Level of detail from 0 (no report) to 5 (maximum).
-qopt-report-phase=loop,vec asks for a report on vectorization and loop optimizations only
Extracts:

```
LOOP BEGIN at foo.c(4,3)
Multiversioned v1
   remark #25228: Loop multiversioned for Data Dependence…
   remark #15300: LOOP WAS VECTORIZED
   remark #15450: unmasked unaligned unit stride loads: 1
   remark #15451: unmasked unaligned unit stride stores: 1
   ….  (loop cost summary)  ….
LOOP END

LOOP BEGIN at foo.c(4,3)
<Multiversioned v2>
   remark #15304: loop was not vectorized: non-vectorizable loop instance from multiversioning
LOOP END
```

```
#include <math.h>
void foo (float * theta, float * sth)  {
  int i;
  for (i = 0; i < 512; i++)
       sth[i] = sin(theta[i]+3.1415927);
}
```

Optimization Report – An Example

$ icc -c -xcommon-avx512 -qopt-report=4 -qopt-report-phase=loop,vec -qopt-report-file=stderr **-fargument-noalias** foo.c

report to stderr
instead of foo.optrpt

...

LOOP BEGIN at foo.c(4,3)

...

remark #15417: vectorization support: number of FP up converts: single precision to double precision 1
remark #15418: vectorization support: number of FP down converts: double precision to single precision 1
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 cost summary** ---
remark #15476: **scalar cost: 111**
remark #15477: **vector cost: 10.310**
remark #15478: **estimated potential speedup: 10.740**
remark #15482: vectorized math library calls: 1
remark #15487: **type converts: 2**
remark #15488: --- end vector 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 < 512; i++)
      sth[i] = sin(theta[i]+3.1415927);
}
```

Optimization Report – An Example

```
$ icc -S -xcommon-avx512 -qopt-report=4 -qopt-report-phase=loop,vec -qopt-report-file=stderr -fargument-noalias foo.c
LOOP BEGIN at foo2.c(4,3)
...
remark #15305: vectorization support: vector length 32
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 cost summary ---
  remark #15476: scalar cost: 109
  remark #15477: vector cost: 5.250
  remark #15478: estimated potential speedup: 20.700
  remark #15482: vectorized math library calls: 1
  remark #15488: --- end vector cost summary ---
  remark #25015: Estimate of max trip count of loop=32
LOOP END

$ grep sin foo.s
     call    __svml_sinf16_b3
```

```
#include <math.h>
void foo (float * theta, float * sth)  {
  int i;
  for (i = 0; i < 512; i++)
     sth[i] = sinf(theta[i]+3.1415927f);
}
```

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, 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
```

The -fp-model switch

-fp-model

- fast [=1] — allows value-unsafe optimizations (default)
- fast=2 — allows a few additional approximations
- precise — value-safe optimizations only
- source | double | extended — imply "precise" unless overridde
- except — enable floating-point exception semantics
- strict — precise + except + disable fma + don't assume default floating-point environment
- consistent — most reproducible results between different processor types and optimization options

-fp-model precise -fp-model source

- recommended for best reproducibility
- also for ANSI/ IEEE standards compliance, C++ & Fortran
- "source" is default with "precise" on Intel 64

Looking for best compiler options?

It depends!

- workload, hw, OS, compiler version, memory allocation, etc.
- take a look on benchmark results and options for reference:

SPECint®_rate_base_2017

-xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-mem-layout-trans=4

SPECfp®_rate_base_2017

-xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -ffinite-math-only -qopt-mem-layout-trans=4

SPECint®_speed_base_2017

-xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-mem-layout-trans=4 -qopenmp

SPECfp®_speed_base_2017

-xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -ffinite-math-only -qopenmp

Let's Get Started!

New compilers

icx/ifx/dpcpp

intel®

Intel® Compilers – Target & Packaging

Intel Compiler	Target	OpenMP Support	OpenMP Offload Support	Current Status (Sep 2020)	Release Q4'20	Included in oneAPI Toolkit
Intel® C++ Compiler, ILO (icc)	CPU	Yes	No	Production** + Beta	Production	HPC
Intel® oneAPI DPC++/C++ Compiler (dpcpp)	CPU, GPU, FPGA*	No	No	Beta	Production	Base
Intel® oneAPI DPC++/C++ Compiler (ICX)	CPU GPU*	Yes	Yes	Beta	Production	Base and HPC
Intel® Fortran Compiler, ILO (ifort)	CPU	Yes	No	Production** + Beta	Production	HPC
Intel® Fortran Compiler (ifx)	CPU, GPU*	Yes	Yes	Beta	Beta***	HPC

Cross Compiler Binary Compatible and Linkable!

*Intel® Platforms

**PSXE 2020 Production+oneAPI HPC Toolkit(BETA)

*** IFX will remain in BETA in 2021

What is Data Parallel C++?

The language is:

C++

+

SYCL* khronos.org/sycl/

+

Additional Features tinyurl.com/dpcpp-ext

What is Data Parallel C++?

The implementation is:

Clang

+

LLVM https://github.com/intel/llvm

+

Runtime https://github.com/intel/compute-runtime

Code samples:

tinyurl.com/dpcpp-tests

tinyurl.com/oneapi-samples

intel

DPC++ extensions

tinyurl.com/dpcpp-ext

tinyurl.com/sycl2020

Extension	Purpose	SYCL 2020
USM (Unified Shared Memory)	Pointer-based programming	✓
Sub-groups	Cross-lane operations	✓
Reductions	Efficient parallel primitives	✓
Work-group collectives	Efficient parallel primitives	✓
Pipes	Spatial data flow support	
Argument restrict	Optimization	
Optional lambda name for kernels	Simplification	✓
In-order queues	Simplification	✓
Class template argument deduction and simplification	Simplification	✓

IFX (Beta) Status, Setting Expectations

- Today and at GOLD end of 2020 will remain in BETA as it matures
  - IFX CORE Fortran LANGUAGE
    - F77, F90/95, a good subset of F03
    - Use **`-stand f03`** if you want warnings for features not in F2003
    - Use **`-stand f03 –warn errors`** options to abort if any F08 or above detected.
    - Much work needed in 2021 and beyond to implement rest of F03, then F08, then F18
  - IFX OpenMP Support
    - CPU OpenMP 3.x clauses mostly work
    - OFFLOAD: Small subset of offload – simple arrays, simple OpenMP TARGET MAP directives
    - Much work needed in 2021 and beyond to implement OpenMP offload

Fortran Strategy for Offload TODAY

- Utilize binary interoperability
- Core language CPU:
  - ifx to compile offload code, ifx for <= F03,
  - ifort for anything not compiling w/ IFX
  - link with ifx: offload needs ifx link
- OpenMP CPU: ifort or ifx for OpenMP cpu constructs
- OpenMP GPU TARGET offload:
  - ifx OMP5 offload or
  - ifx or ifort calling into C/C++ for OMP offload or DPCPP

Choices: ICX and ICC Classic

- Choice of ICC or ICX in oneAPI products
  - ICC for performance for CPU targets
  - ICX for offload and porting for future, or if you prefer superior Clang C++ language checking
  - ICX also available (with no offload) in PSXE 2020 via “icc -qnextgen”
- ICX used as basis for DPC++ Compiler
  - DPC++ extensions added, driver ‘dpcpp’ used instead of ‘icx/icc/icpc’
- ICX is needed for OpenMP 5 TARGET offload to Intel GPU targets
  - ICC Classic will not have OMP offload to GPUs

37

OpenMP with Intel® Compilers

- Drivers
  - icx (C/C++) ifx (Fortran)
- OPTIONS

-fiopenmp

- Selects Intel Optimized OMP
- -fopenmp for Clang* O.S. OMP
- -qopenmp NO!! rejected, only in ICC/IFORT

-fopenmp-targets=spir64

- Needed for OMP Offload
- Generates SPIRV code fat binary for offload kernels

Get Started with OpenMP* Offload Feature to GPU: tinyurl.com/intel-openmp-offload

Intel env Var LIBOMPTARGET_PROFILE

- OpenMP Standard ENV vars are accepted. Add to this list …
- export LIBOMPTARGET_PROFILE=T
  - performance profiling for tracking on GPU kernel start/complete time and data-transfer time.

```
GPU Performance (Gen9, export LIBOMPTARGET_PROFILE=T,usec)
… …
Kernel Name:
__omp_offloading_811_29cbc383__ZN12BlackScholesIdE12execute_partEiii_l368
iteration #0 ...
calling validate ... ok
calling close ...
execution finished in 1134.914ms, total time 0.045min
passed

LIBOMPTARGET_PROFILE:
-- DATA-READ: 16585.256 usec
-- DATA-WRITE: 9980.499 usec
-- EXEC-__omp_offloading_811_29cbc383__ZN12BlackScholesIfE12execute_partEiii_l368:
24048.503 usec
```

Debug RT env Var LIBOMPTARGET_DEBUG

- Export LIBOMPTARGET_DEBUG=1
  - Dumps offloading runtime debugging information. Its default value is 0 which indicates no offloading runtime debugging information dump.

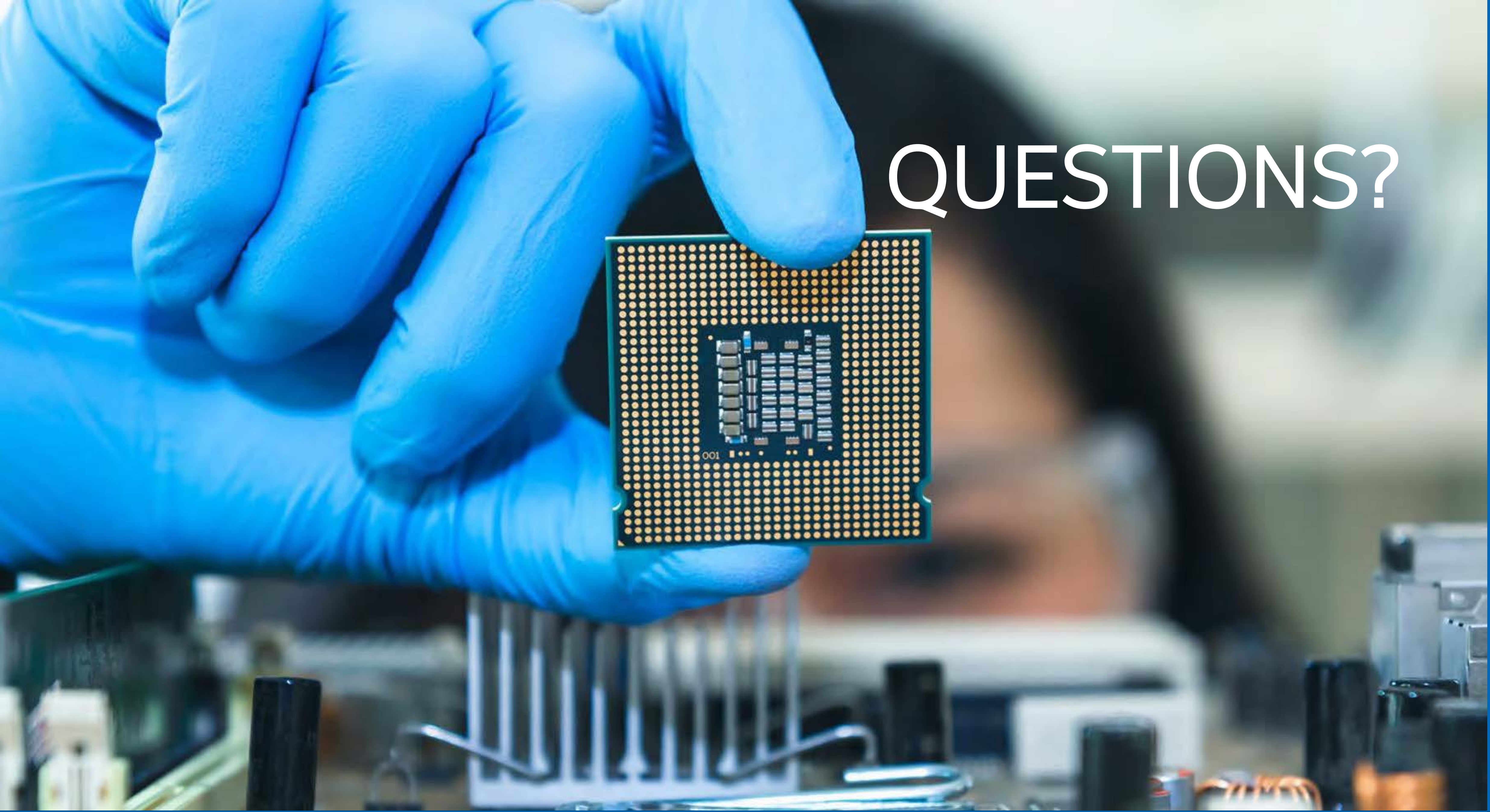
```
./matmul

Libomptarget --> Loading RTLs...
Libomptarget --> Loading library 'libomptarget.rtl.nios2.so'...
Libomptarget --> Loading library 'libomptarget.rtl.x86_64.so'...
Libomptarget --> Successfully loaded library 'libomptarget.rtl.x86_64.so'!
Libomptarget --> Loading library 'libomptarget.rtl.opencl.so'...

Target OPENCL RTL --> Start initializing OpenCL
Target OPENCL RTL --> cl platform version is OpenCL 2.1 LINUX
Target OPENCL RTL --> Found 1 OpenCL devices
Target OPENCL RTL --> Device#0: Genuine Intel(R) CPU 0000 @ 3.00GHz

…. AND MUCH MORE …
```

QUESTIONS?



Notices & Disclaimers

- This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice.
- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.
- 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.
- 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.
-

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