Intel® DPC++ Compatibility Tool

Migrating CUDA Codes to DPC++

intel®

Agenda

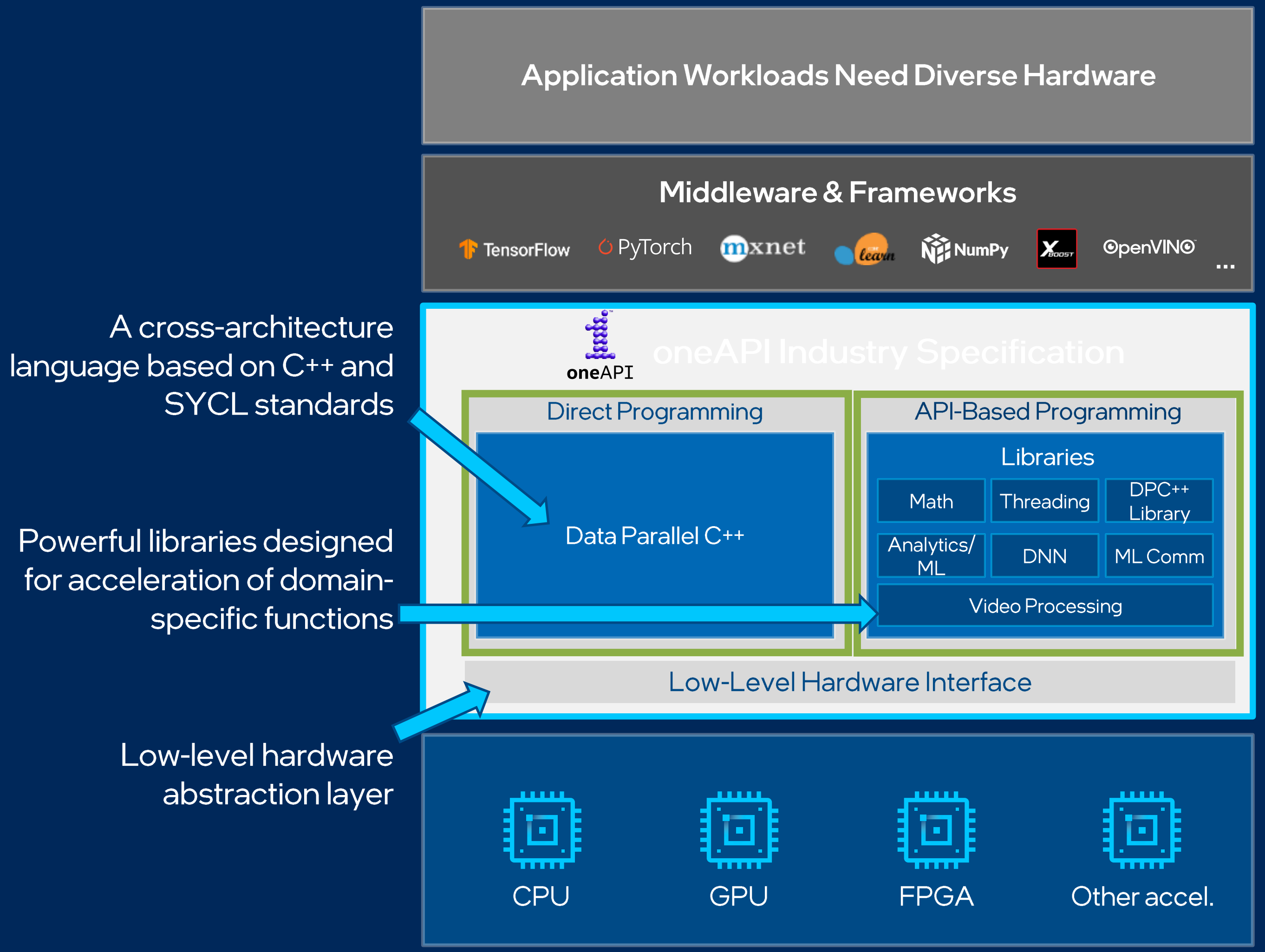
- oneAPI Brief Overview
- Intel® DPC++ Compatibility Tool Workflow
- Migration Flow and Vector-Add Example
- Demo Tutorial
  - Simple CUDA* File Project
  - Migrate Multi CUDA Files Project
- Best Known Methods for Migration
- Eclipse and Visual Studio Integration
- Key Takeaways

oneAPI Industry Initiative

Break the Chains of Proprietary Lock-in

Open to promote community and industry collaboration

Enables code reuse across architectures and vendors



oneAPI

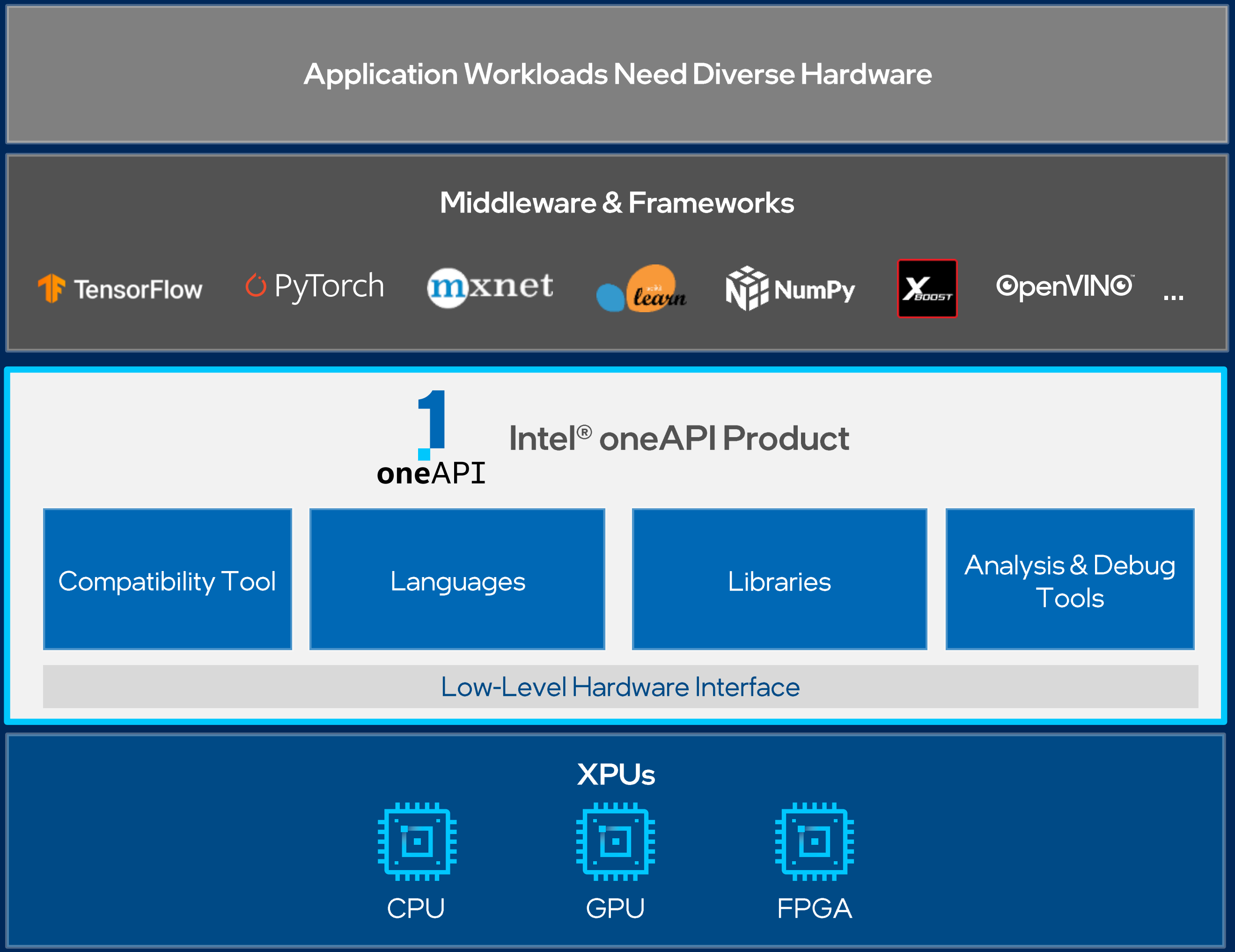
The productive, smart path to freedom for accelerated computing from the economic and technical burdens of proprietary programming models

Intel® oneAPI Product

Built on Intel's Rich Heritage of CPU Tools Expanded to XPUs

A complete set of advanced compilers, libraries, and porting, analysis and debugger tools

- Accelerates compute by exploiting cutting-edge hardware features
- Interoperable with existing programming models and code bases (C++, Fortran, Python, OpenMP, etc.), developers can be confident that existing applications work seamlessly with oneAPI
- Eases transitions to new systems and accelerators—using a single code base frees developers to invest more time on innovation



Available Now

Visit software.intel.com/oneapi for more details
Some capabilities may differ per architecture and custom-tuning will still be required. Other accelerators to be supported in the future.

Intel® DPC++ Compatibility Tool

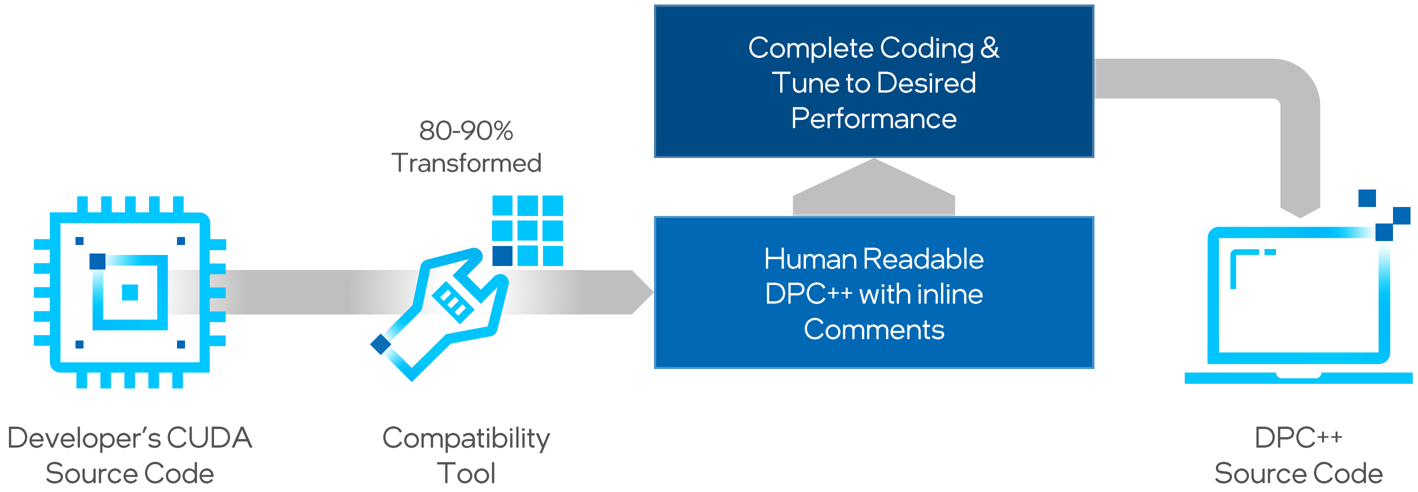
Minimizes Code Migration Time

Assists developers migrating code written in CUDA to DPC++ once, generating **human readable** code wherever possible

~80-90% of code typically migrates automatically

Inline comments are provided to help developers finish porting the application

Intel DPC ++ Compatibility Tool Usage Flow



Intel® oneAPI Base Toolkit

Accelerate Data-centric Workloads

A core set of core tools and libraries for developing high-performance applications on Intel® CPUs, GPUs, and FPGAs.

Who Uses It?

- A broad range of developers across industries
- Add-on toolkit users since this is the base for all toolkits

Top Features/Benefits

- Data Parallel C++ compiler, library and analysis tools
- DPC++ Compatibility tool helps migrate existing code written in CUDA
- Python distribution includes accelerated scikit-learn, NumPy, SciPy libraries
- Optimized performance libraries for threading, math, data analytics, deep learning, and video/image/signal processing

Intel® oneAPI Base Toolkit

Direct Programming

- Intel® oneAPI DPC++/C++ Compiler
- Intel® DPC++ Compatibility Tool
- Intel® Distribution for Python
- Intel® FPGA Add-on for oneAPI Base Toolkit

API-Based Programming

- Intel® oneAPI DPC++ Library oneDPL
- Intel® oneAPI Math Kernel Library - oneMKL
- Intel® oneAPI Data Analytics Library - oneDAL
- Intel® oneAPI Threading Building Blocks - oneTBB
- Intel® oneAPI Video Processing Library - oneVPL
- Intel® oneAPI Collective Communications Library oneCCL
- Intel® oneAPI Deep Neural Network Library - oneDNN
- Intel® Integrated Performance Primitives - Intel® IPP

Analysis & debug Tools

- Intel® VTune™ Profiler
- Intel® Advisor
- Intel® Distribution for GDB

Migrating Vector Add Example

Migrating Simple Example

- dpct [options] [<source0>... <sourceN>]
  - Ensure supported CUDA header files are available
    - May use –cuda-include-path
- Bult-in Usage Information
  - dpct --help

Vector-Add Example: Migration with Intel® DPC++ Compatibility Tool

CUDA

```
#include <cuda.h>
#include <stdio.h>
#define VECTOR_SIZE 256

__global__ void VectorAddKernel(float* A, float* B, float* C)
{
    A[threadIdx.x] = threadIdx.x + 1.0f;
    B[threadIdx.x] = threadIdx.x + 1.0f;
    C[threadIdx.x] = A[threadIdx.x] + B[threadIdx.x];
}

int main()
{
    float *d_A, *d_B, *d_C;

    cudaMalloc(&d_A, VECTOR_SIZE*sizeof(float));
    cudaMalloc(&d_B, VECTOR_SIZE*sizeof(float));
    cudaMalloc(&d_C, VECTOR_SIZE*sizeof(float));
```

DPC++

1

```
#include <CL/sycl.hpp>
#include <dpct/dpct.hpp>
#define VECTOR_SIZE 256
```

2

```
void VectorAddKernel(float* A, float* B, float* C, sycl::nd_item<3>
item_ct1)
{
    A[item_ct1.get_local_id(2)] = item_ct1.get_local_id(2) + 1.0f;
    B[item_ct1.get_local_id(2)] = item_ct1.get_local_id(2) + 1.0f;
    C[item_ct1.get_local_id(2)] =
        A[item_ct1.get_local_id(2)] + B[item_ct1.get_local_id(2)];
}
```

3

```
int main()
{
    dpct::device_ext &dev_ct1 = dpct::get_current_device();
    sycl::queue &q_ct1 = dev_ct1.default_queue();
    float *d_A, *d_B, *d_C;

    d_A = sycl::malloc_device<float>(VECTOR_SIZE, q_ct1);
    d_B = sycl::malloc_device<float>(VECTOR_SIZE, q_ct1);
    d_C = sycl::malloc_device<float>(VECTOR_SIZE, q_ct1);
```

https://github.com/oneapi-src/oneAPI-samples/tree/master/Tools/Migration/vector-add-dpct

Vector-Add Migration Example (continued)

CUDA | **DPC++**

4

```
VectorAddKernel<<<1, VECTOR_SIZE>>>(d_A, d_B, d_C);
```

```
q_ct1.submit([&](sycl::handler &cgh) {
    cgh.parallel_for(sycl::nd_range<3>(
                         sycl::range<3>(1, 1, VECTOR_SIZE),
                         sycl::range<3>(1, 1, VECTOR_SIZE)),
                     [=](sycl::nd_item<3> item_ct1) {
                       VectorAddKernel(d_A, d_B, d_C, item_ct1);
                     });
});
```

5

```
float Result[VECTOR_SIZE] = { };
cudaMemcpy(Result, d_C, VECTOR_SIZE*sizeof(float),
           cudaMemcpyDeviceToHost);
```

```
float Result[VECTOR_SIZE] = { };
q_ct1.memcpy(Result, d_C, VECTOR_SIZE * sizeof(float)).wait();
```

6

```
cudaFree(d_A);
cudaFree(d_B);
cudaFree(d_C);
```

```
sycl::free(d_A, q_ct1);
sycl::free(d_B, q_ct1);
sycl::free(d_C, q_ct1);
```

7

```
for (int i = 0; i < VECTOR_SIZE; i++) {
    if (i % 16 == 0) {
        printf("\n");
    }
    printf("%f ", Result[i]);
}

return 0;
}
```

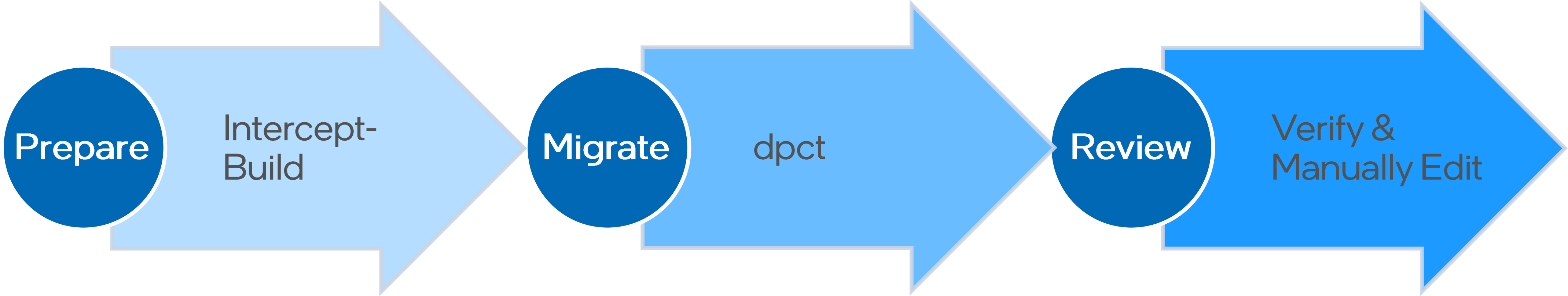
```
for (int i = 0; i < VECTOR_SIZE; i++) {
    if (i % 16 == 0) {
        printf("\n");
    }
    printf("%f ", Result[i]);
}

return 0;
}
```

https://github.com/oneapi-src/oneAPI-samples/tree/master/Tools/Migration/vector-add-dpct

Migrating Needleman Wunsch and HydroC Examples

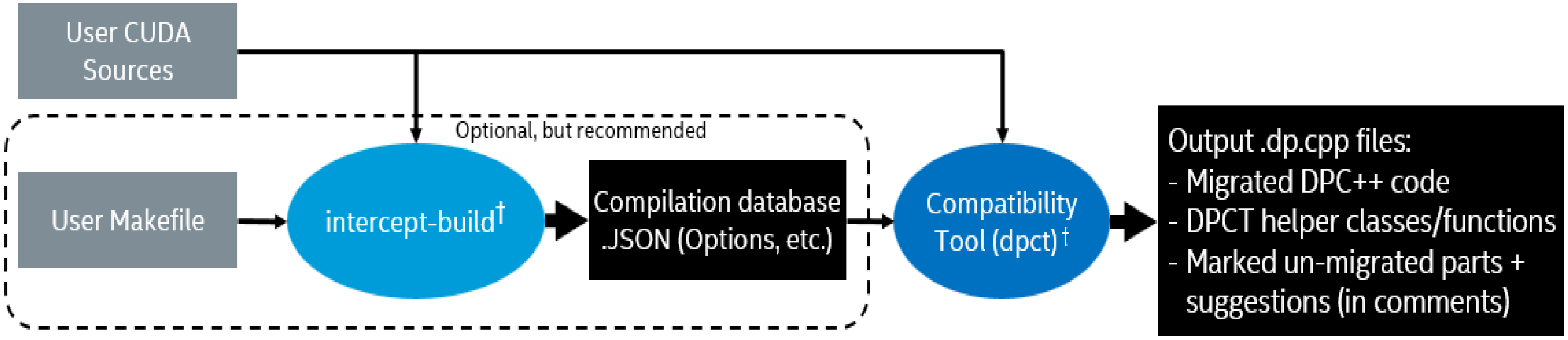
Migration Flow

Typical preparation steps for simple to complex projects



Intercept Build

- Use intercept-build to create a compilation database
  - For projects that use Make or Cmake
  - Keeps track of compilation options, settings, macro definitions, include paths, etc.
  - Creates a JSON file containing the build commands
- Run "make clean" before "intercept-build"



† Certain CUDA language header files may need to be accessible to the Intel® DPC++ Compatibility Tool

DPCT Basic Options

- dpct [options] [<source0>… <sourceN>]

DPCT Basic Options	
--in-root	Path to the root of the source tree to be migrated
--out-root	Path to root of generated files.
-p	Path to compile database JSON file
--process-all	Migraters/copies all files from --in-root directory to the --out-root directory, eliminating need to specify .cu files one by one
--extra-arg	Specify more Clang compiler options. e.g. dpct --extra-arg="-std=c++14" –extra-arg="-I…"
--format-style	Sets formatting style for output files. e.g. =llvm, =google, =custom (Uses .clang-format file)
--format-range	Code formatting applied to no code (=none), migrated code (=migrated), or all code (=all)

DPCT Recommended Options

- dpct [options] [<source0>... <sourceN>]

DPCT Options that Ease Migration/Debug	
--keep-original-code	Keep original CUDA code in the comments of generated DPC++ file. Allows easy comparison of original CUDA code to generated DPC++ code.
--comments	Insert comments explaining the generated code
---always-use-async-handler	Always create cl::sycl::queue with the async exception handler

- Many other options available use dpct --help

DPCT Namespace Usage

- DPCT namespace provides helper function and macros to assist the migration of input source code.
  - dpct::
- Implemented in header files (include/dpct)
- Intended to become part of your code.
- Examples: dpct_malloc, dpct_memcpy, get_buffer, get_default_queue, get_default_context
- Not recommended to use these when writing new DPC++ code

General Best Known Methods (BKMs)

- Migrate Incrementally
  - If you see *dpct* generate multiple errors when migrating a long list of CUDA source files in one run, do it one-by-one
- Start with a clean project - “make clean” before running “intercept-build make”
- Run *intercept-build make -k* to keep going when some targets can't be made when generating compilation database

Code Modifications Prior to Migration

- Ensure source files are syntactically correct
- Possibly needed due to differences between clang and nvcc
  1. Namespace qualification maybe needed in certain scenario with clang parser
  2. Additional forward class declarations may be needed by clang
  3. Space within the triple brackets of kernel innovacation are tolerated by nvcc but not clang
     - e.g. cuda_kernel<< <num_blocks, threads_per_block>> >(args…)
  - See [Compilation CUDA with clang](#) on llvm.org for more details.

Unified Shared Memory (USM) Usage

- DPC++ supports USM that allows pointer-based approach to manage host and device memory.
- USM produces less volume code compare to SYCL buffers
- The Compatibility Tool uses USM by default.
- May be trouble some for non-Intel compilers targeting non-Intel hardware

DPCT USM Option	
--usm-level	Sets Unified Shared Memory (USM) level. =Restricted: Use USM (default) =none: Uses helper functions and SYCL buffers

Diagnostics Reference

Compatibility Tool highlights issues with migration and code comments

/path/to/file:20:1: warning: DPCT10XX:0: text of the warning

//source code line for which warning was generated

ID	Message	Detailed Help	Suggestions to Fix
`DPCT1000`	An error handling `if-stmt` was detected but could not be rewritten. See the details in the resulting file comments.	The CUDA* API return error codes that are consumed by the program logic. SYCL* uses exceptions to report errors and does not return the error code. When the error handling logic in the original code is simple (for example, a print error message and exit), the code is removed in the resulting Data Parallel C++ (DPC++) application. The expectation is that SYCL throws an exception, which is handled with the printing of an exception message and exiting (the exception handler is generated automatically by the Intel® DPC++ Compatibility Tool). This warning is generated when the Intel® DPC++ Compatibility Tool detects more complex error handling than it considers safe to remove.	Review the error handling `if-statement` and try to rewrite it to use an exception handler instead.
`DPCT1001`	The statement could not be removed. See the details in the resulting file comments.	The Intel® DPC++ Compatibility Tool was not able to remove the code in the `then` clause of `if-stmt`. See `DPCT1000`.	See `DPCT1000`.
`DPCT1002`	A special case error handling `if-stmt` was detected. You may need to rewrite this code.	See `DPCT1000`	See `DPCT1000`.

See Compatibility Tool – Diagnostics Reference

Code Review or Rewrite Needed

Diagnostic Reference

- Error code logic replaced with (*,0) code or commented out
- Equivalent DPC++ API not available
- CUDA Compute Capability-dependent logic
- Hardware-dependent API (clock())
- Migration not supported for some API
- Execution time measurement logic
- Handling built-in vector type conflicts
- Migration of cuBLAS API (Review arguments list)

Demo: Simple CUDA Project Migration

- Rodinia Benchmark Suite v3.1 – Introduction
- Setting/Verifying the Environment for Intel® DPC++ Compatibility Tool
- Demo
  - Planning for Migration
  - Compatibility Tool Options
  - Migrating Needleman Wunsch Application

http://rodinia.cs.virginia.edu/doku.php

https://software.intel.com/en-us/get-started-with-intel-dpcpp-compatibility-tool

https://software.intel.com/content/www/us/en/develop/documentation/intel-dpcpp-compatibility-tool-user-guide/top/migrate-a-project/migrate-a-project-on-linux.html

https://software.intel.com/en-us/intel-dpcpp-compatibility-tool-user-guide-usage-workflow-overview

Demo: HydroC - Multi CUDA Files Project Migration

- Setting/Verifying the Environment for Intel® DPC++ Compatibility Tool
- Demo
  - Planning for Migration; Understanding the Application File …
  - *intercept-build* Options
  - Compatibility Tool Options
  - Migrating HydroC Application

https://github.com/HydroBench/Hydro/tree/master/HydroC/cuHydroC_2DMpi/Src

https://github.com/HydroBench/Hydro

https://github.com/HydroBench/Hydro/blob/master/License.txt

SPECFEM3D_GLOBE

- SPECFEM3D_GLOBE simulates global and regional (continental-scale) seismic wave propagation
- Official repo: https://github.com/geodynamics/specfem3d_globe

github.com/AlDanial/cloc v 1.74 T=1.44 s (370.8 files/s, 156306.8 lines/s)

Language	files	blank	comment	code
Fortran 90	279	27677	41716	100021
C	81	3145	5405	20851
CUDA	88	1410	2286	10841
Ruby	61	554	192	4365
make	17	532	817	1887
C/C++ Header	5	284	370	995
C++	1	196	229	773
Markdown	1	31	0	102
SUM:	533	33829	51015	139835

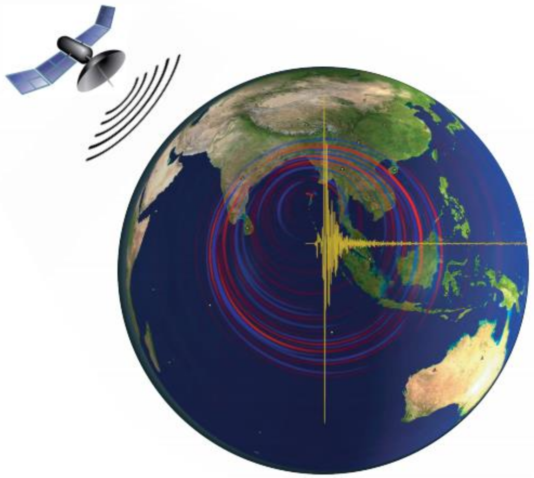


Image source: specfem3d_globe-manual.pdf

SPECFEM3D_GLOBE – Migration to DPC++

Prepare

```
$ git clone --recursive --branch devel https://github.com/geodynamics/specfem3d_globe.git
$ ./configure --with-cuda=cuda9    CUDA_LIB=${CUDA_ROOT}/lib64/   \
           CUDA_INC=${CUDA_ROOT}/include/  MPI_INC=${I_MPI_ROOT}/include/
$ intercept-build make -i
```

Migrate

```
$ dpct -p compile_commands.json
```

Review

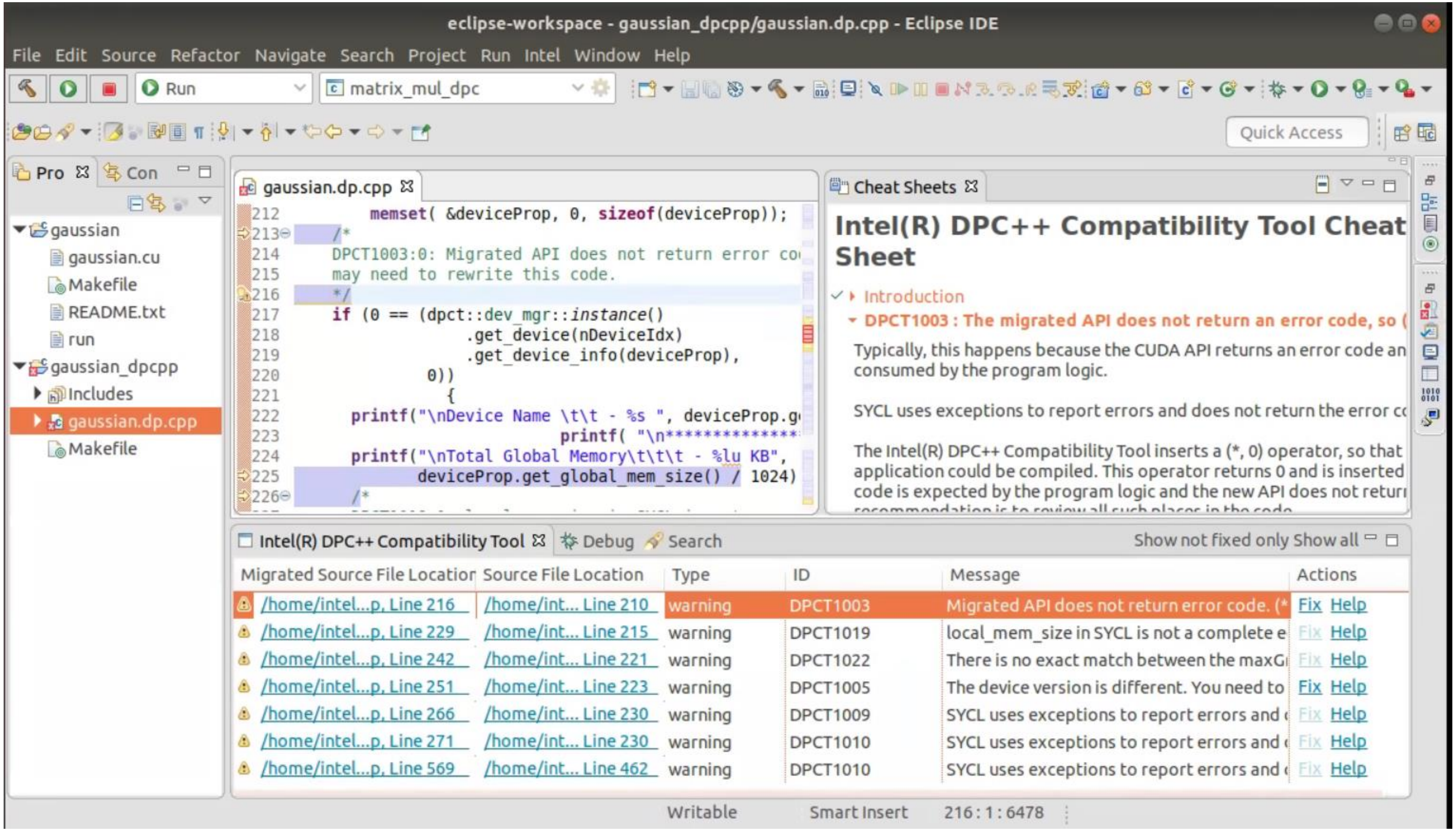
Review diagnostics messages using reference and manually edit
Address other not-so-obvious issues

Diagnostics Messages Breakdown

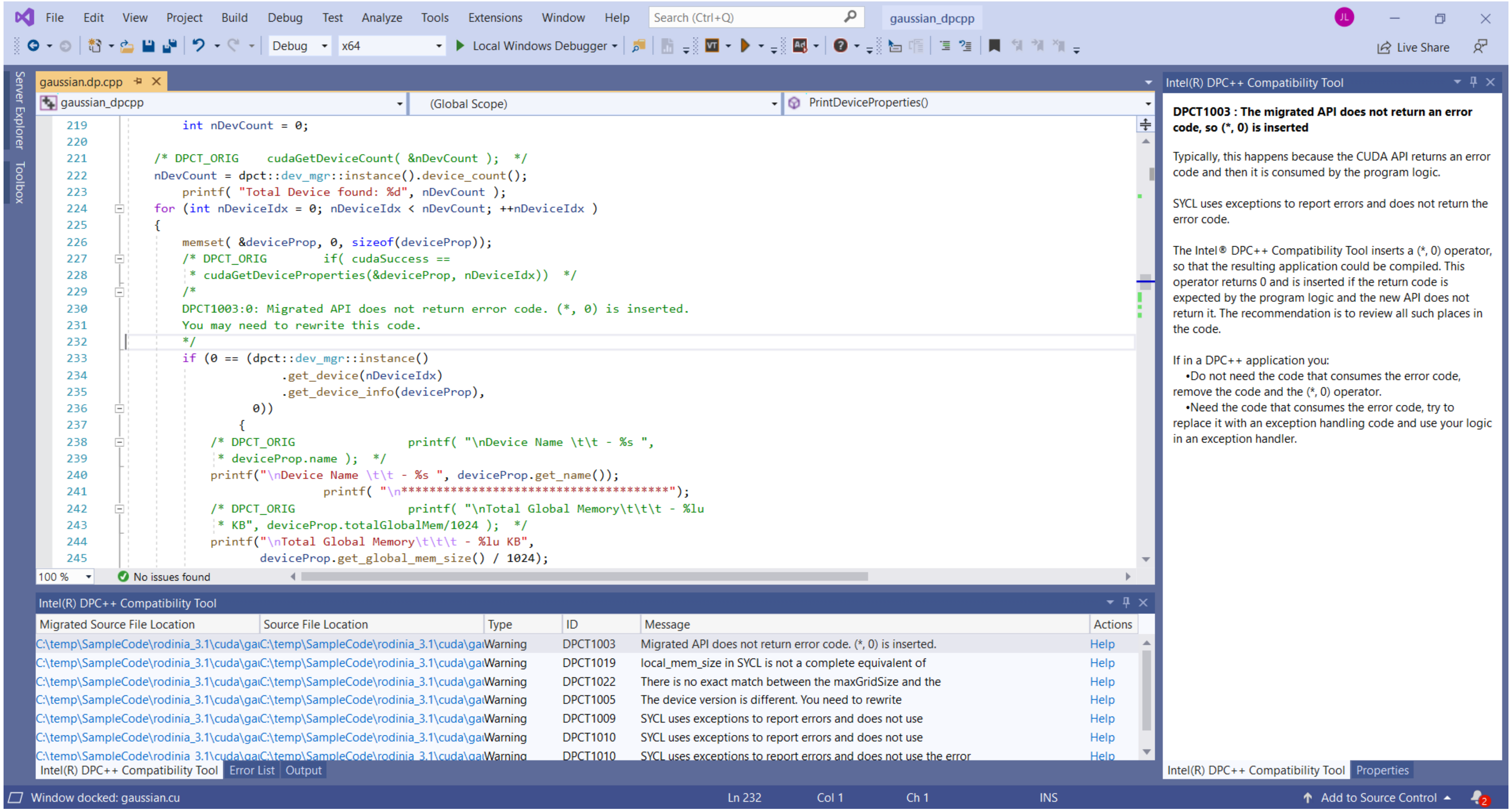
DPCT{diagnostics#} (count)	Summary
DPCT1000 (6), DPCT1001(6), DPCT1003 (111) , DPCT1009 (8), DPCT1010 (3) , DPCT1024 (2)	Different scenarios for error handling
DPCT1005 (4), DPCT1012 (4), DPCT1017 (10), DPCT1019 (1), DPCT1022 (1), DPCT1026 (5), DPCT1027 (3), DPCT1051 (4)	Unavailable equivalent API's in SYCL* (e.g. device versions, certain device properties, timing logic)
DPCT1039 (9)	Handling atomics (global atomics by default, local will need intervention)
DPCT1049 (59)	Validating use of work-group sizes

Using plugins with IDE

Eclipse: Gaussian



Visual Studio 2019: Gaussian



Summary

- OneAPI delivers a unified programming model to simplify development across diverse architectures
- Intel DPC++ Compatibility tool assists developers in migrating code written in CUDA to DPC++, increasing developer productivity
- DPC++ is an open specification for a portable, architecture-neutral language for expressing parallelism; it is based on industry standards

References

- Intel® DPC++ Compatibility Tool Jupyter Tutorial
- Intel® DPC++ Compatibility Tool
  - User Guide
  - Get Started Guide
  - Release Notes

Are You Ready to Try oneAPI?

1. Identify potential workloads/candidates for testing
   a. Download DPCT and migrate code to DPC++ on-prem, if applicable
   b. Test, tune and optimize your code or test samples in the Intel® DevCloud—a cloud-based development sandbox environment that gives you full access to the latest Intel® hardware and oneAPI software https://software.intel.com/devcloud/oneapi
2. Learn more at http://software.intel.com/oneapi the channel to documentation, downloads, access to Intel® Devcloud, and access to support forum

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.

The benchmark results reported herein may need to be revised as additional testing is conducted. The results depend on the specific platform configurations and workloads utilized in the testing, and may not be applicable to any particular user's components, computer system or workloads. The results are not necessarily representative of other benchmarks and other benchmark results may show greater or lesser impact from mitigations.

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.

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®