2022 RMACC HPC Symposium, August 4th, 2022

Migrating CUDA Code to SYCL with Intel oneAPI

Presenter: Shiquan Su, PhD, Intel Senior Technical Consulting Advisor

intel®

Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation. Learn more at intel.com or from the OEM or retailer.

Your costs and results may vary.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

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. https://software.intel.com/en-us/articles/optimization-notice

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. See backup for configuration details. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Outline:

- oneAPI introduction
- DPC++/SYCL introduction
- Intel® DPC++ Compatibility Tool (DPCT) introduction
- Code example: Rodinia Benchmark Suite
- DPCT demonstration

oneAPI introduction

Programming Challenges

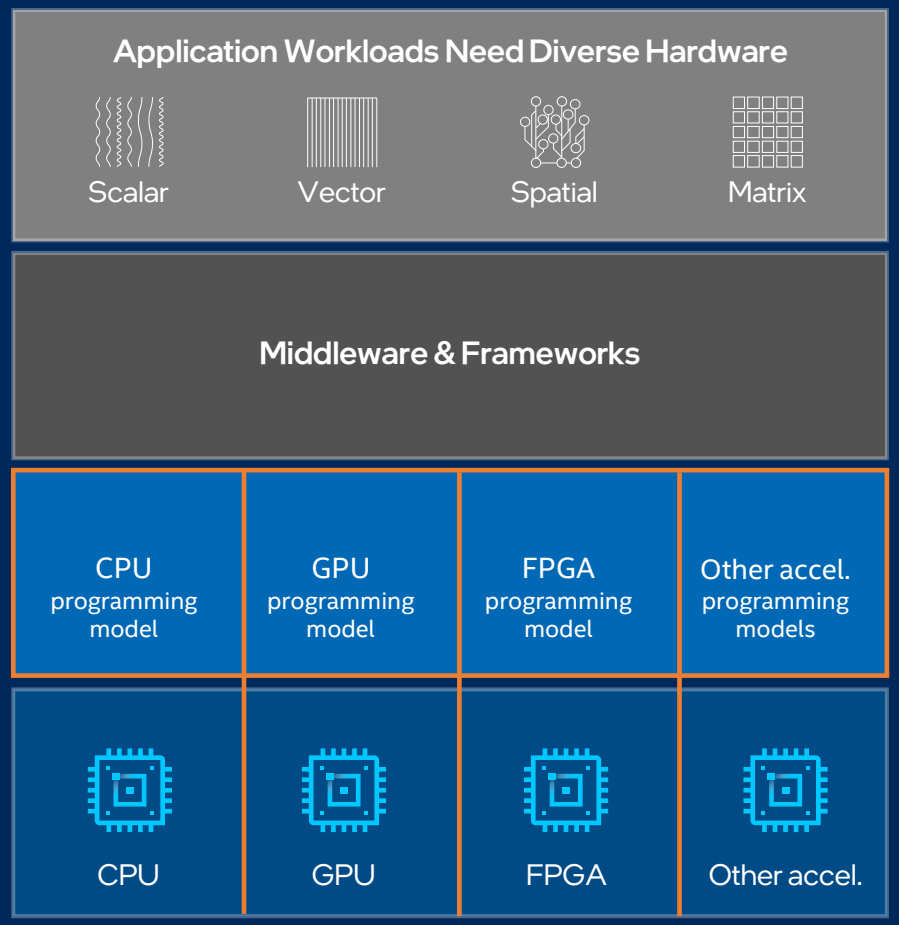
for Multiple Architectures

Growth in specialized workloads

Variety of data-centric hardware required

Separate programming models and toolchains for each architecture are required today

Software development complexity limits freedom of architectural choice



oneAPI

One Programming Model for Multiple Architectures and Vendors

Freedom to Make Your Best Choice

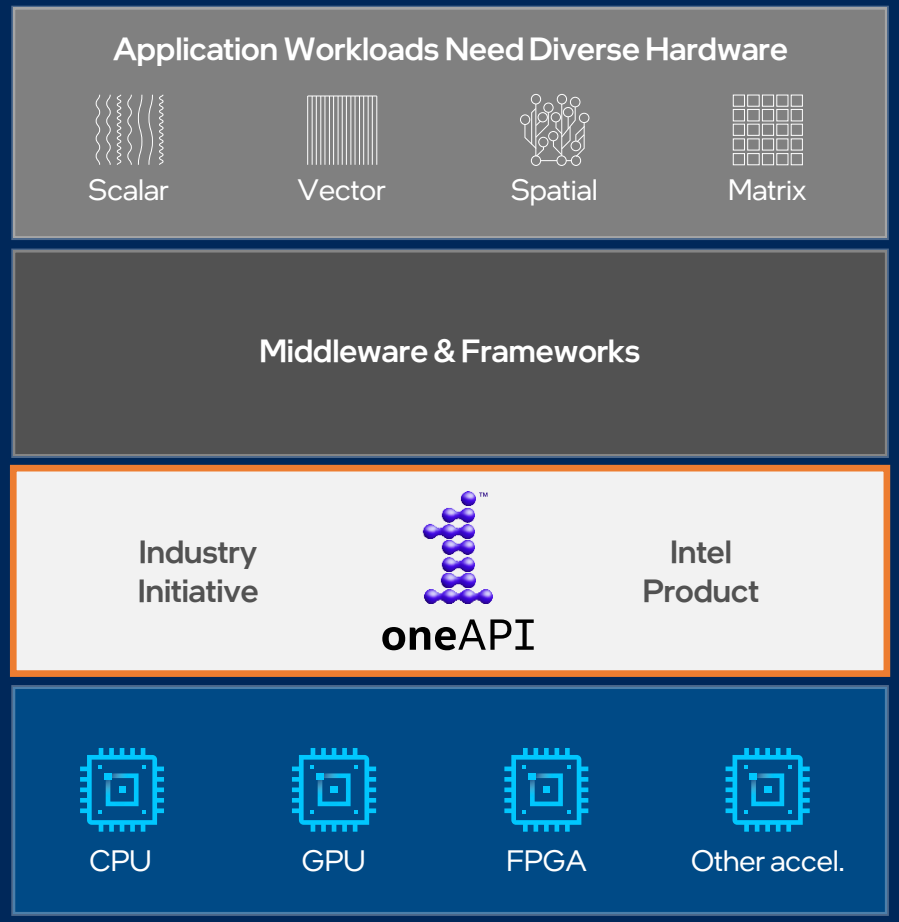
- Choose the best accelerated technology the software doesn't decide for you

Realize all the Hardware Value

- Performance across CPU, GPUs, FPGAs, and other accelerators

Develop & Deploy Software with Peace of Mind

- Open industry standards provide a safe, clear path to the future
- Compatible with existing languages and programming models including C, C++, Python, SYCL, OpenMP, Fortran, and MPI



oneAPI Industry Initiative

Break the Chains of Proprietary Lock-in

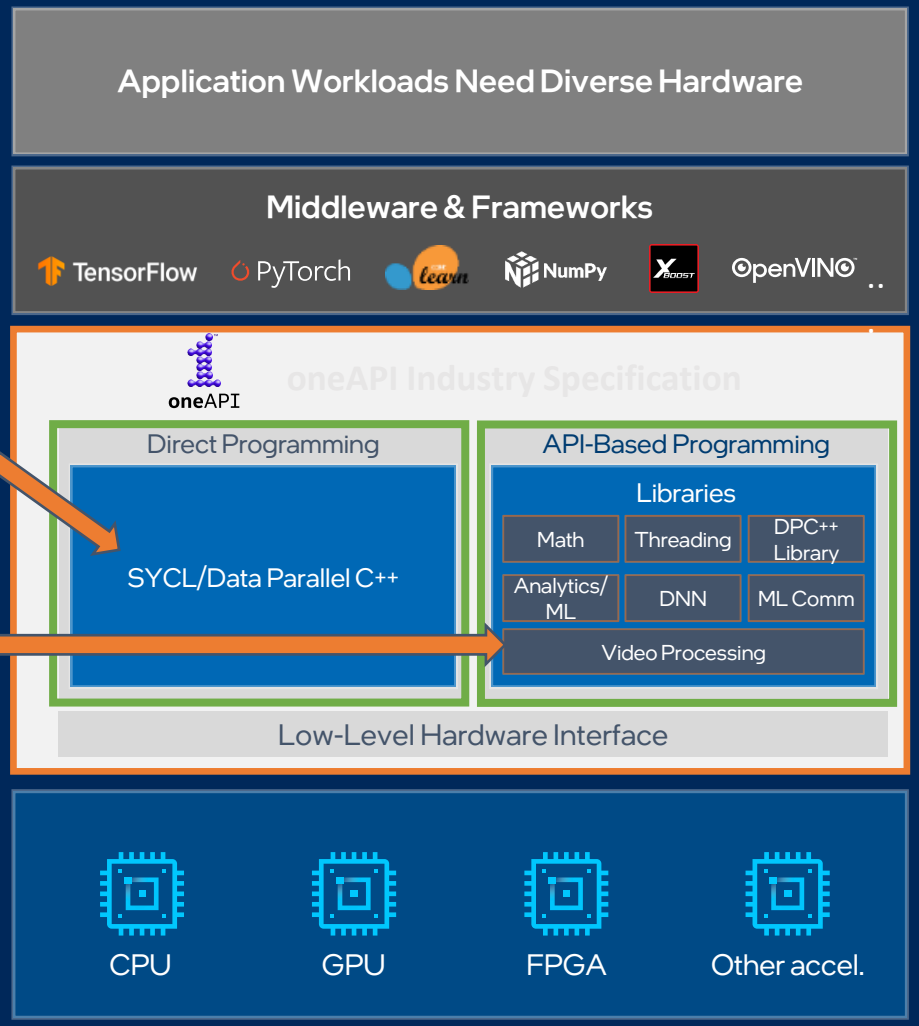
Open to promote community and industry collaboration

Enables code reuse across architectures and vendors

A cross-architecture language based on C++ and SYCL standards

Powerful libraries designed for acceleration of domain-specific functions

Low-level hardware abstraction layer



oneAPI

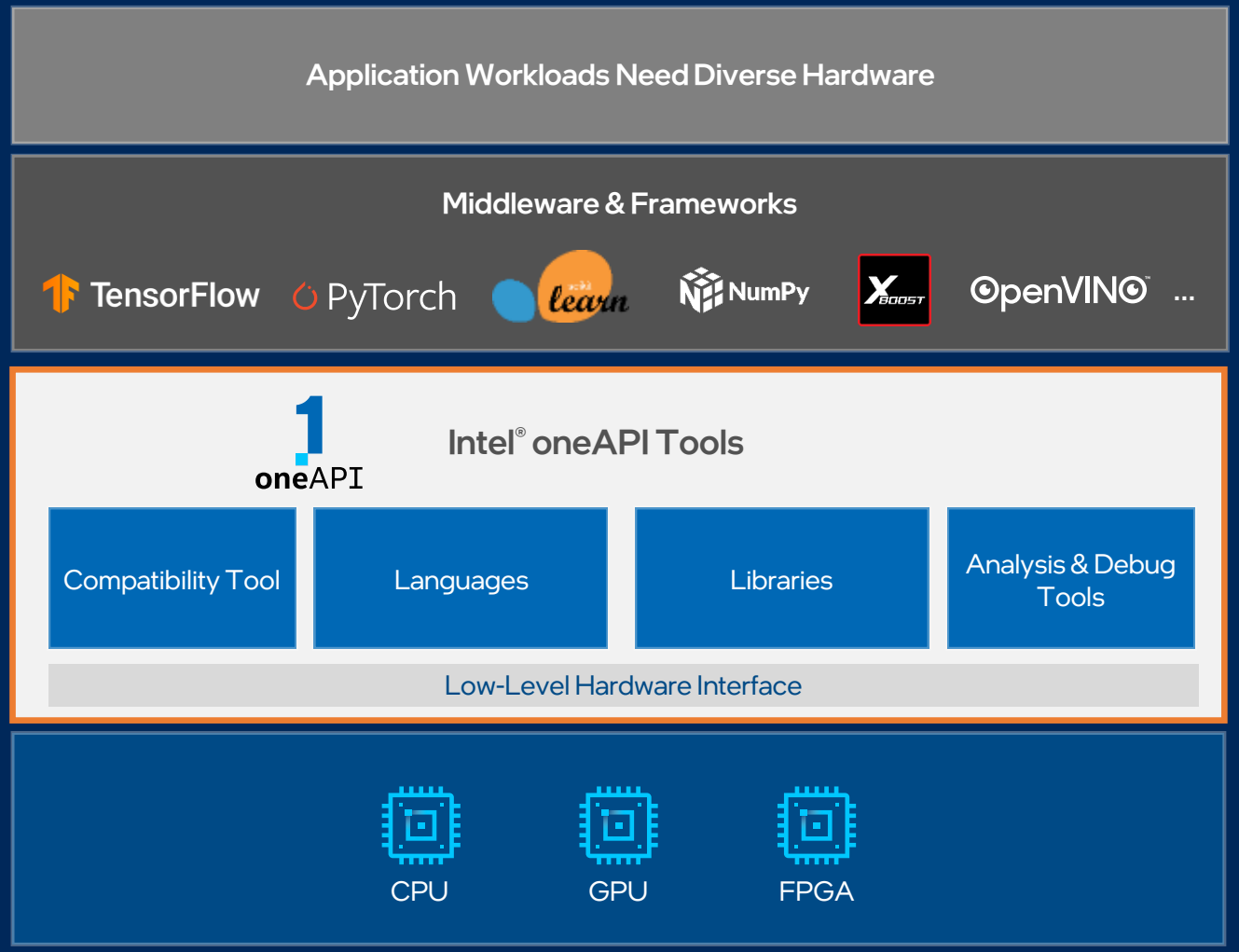
The productive, smart path to freedom for accelerated computing from the economic and technical burdens of proprietary programming models

Intel® oneAPI Tools

Built on Intel's Rich Foundation of CPU Tools Expanded to Accelerators

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

Latest version is 2021.1

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.

oneAPI Ecosystem Support

Accrad Accelerated Radiology
AI SINGAPORE
allegro.ai
Ansys
Argonne NATIONAL LABORATORY
archanan
AsiaInfo 亚信科技
AUTODESK ARNOLD
Berkeley UNIVERSITY OF CALIFORNIA
BENTLEY
ATLAS EXPERIMENT
Atos
BittWare a molex company
Brightskies
Alibaba Cloud
中国石油集团东方地球物理勘探有限责任公司 BGP INC.,CHINA NATIONAL PETROLEUM CORPORATION
CHAOZGROUP
codeplay
DELL Technologies
CANONICAL
CERN openlab
DI TI CERTI FACE
CGG
FACULTY OF MATHEMATICS AND PHYSICS Charles University
CDAC
ICT 中国科学院计算技术研究所 INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES
大势智慧 DASPATIAL
CINECA
CINESITE
Durham University
FOUNDRY
EURECOM Sophia Antipolis
LIGO ARCHIVE
GIGASPACES
GeoEast
Google Cloud
ILLUMINATION MACGUFF
HCL
Hewlett Packard Enterprise
IISER PUNE
MAXON
inesc id lisboa
TÉCNICO LISBOA
Soda
Inria: National Institute for Research in Digital Science & Technology
KFBIO technology for health.
Laboratório Nacional de Computação Científica
MAX PLANCK COMPUTING & DATA FACILITY
MPCDF
URZ HEIDELBERG UNIVERSITY COMPUTING CENTRE
ILLINOIS
Lenovo
LOBACHEVSKY UNIVERSITY
LAIKA
lrz
Microsoft Azure
MEGWARE
mercenaries engineering
Guerilla
Northern Illinois University
UT-BATTELLE Oak Ridge National Laboratory
PHILIPS
QZ quantumz.io
SAP
MEGH COMPUTING
OLD DOMINION UNIVERSITY
SAMSUNG MEDISON
SankhyaSutra Labs
sas
SENAI CIMATEC FIEB Federação das Indústrias do Estado da Bahia
SUSE
TANGENTSTUDIOS
Taboola
rENIAC
TensorFlow
ZUSE INSTITUTE BERLIN
СБЕР Sberbank
SiPearl
Ospirent
TACC
Tech Mahindra
TECHNISCHE UNIVERSITÄT DARMSTADT
Tencent 腾讯
UNITED 联影 IMAGING
UNIVERSITY OF CAMBRIDGE
Indian Institute of Technology Delhi
UCDAVIS UNIVERSITY OF CALIFORNIA
CtG
UNIVERSITY OF OREGON
UNIVERSIDAD DE MÁLAGA
Ben-Gurion University of the Negev
Stockholm University
KTH VETENSKAP OCH KONST
UNIVERSIDAD COMPLUTENSE MADRID
THE UNIVERSITY OF TENNESSEE KNOXVILLE
University College London
University of Stuttgart Germany
THE UNIVERSITY OF UTAH

These organizations support the oneAPI initiative 'concept' for a single, unified programming model for cross-architecture development.
It does not indicate any agreement to purchase or use of Intel's products. *Other names and brands may be claimed as the property of others.

oneAPI: Open Accelerator Ecosystem

Freedom of Choice in Hardware Drives Productivity

Codeplay contribution to DPC++ brings SYCL support for NVIDIA GPUs

oneAPI oneDNN on Arm for A64FX Fugaku

Extending DPC++ with Support for Huawei AI Chipset

NERSC, ALCF, CODEPLAY PARTNER ON SYCL FOR NEXT-GENERATION SUPERCOMPUTERS

on Nvidia

ARGONNE, ORNL AWARD CODEPLAY CONTRACT TO STRENGTHEN SYCL SUPPORT FOR AMD GPUS

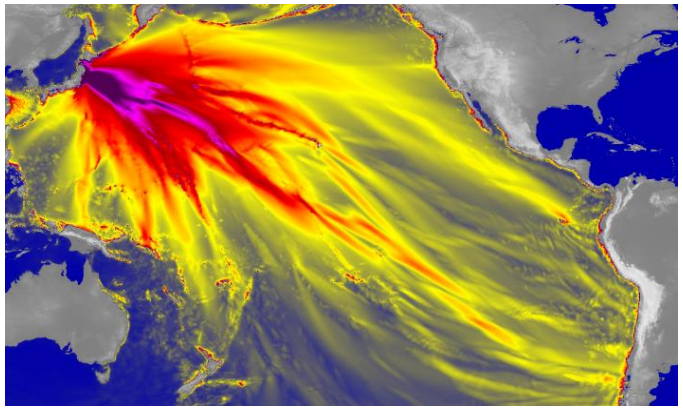
European exascale combines SiPearl's CPU RHEA with Intel's Xe GPU PVC

"So for us, the advantage of oneAPI is that we can increasingly write things in one way, but target multiple types of hardware. That means that we don't necessarily have to duplicate efforts for three or maybe even four accelerator targets."

– Erik Lindahl, GROMACS lead developer & biophysics professor, University of Stockholm

"If you like modern, standard C++ and you want to target GPUs or other accelerators, you will love SYCL!"

– Marcel Breyer, Researcher, University of Stuttgart



Visualization of *easyWave* tsunami simulation application - Courtesy Zuse Institute Berlin (ZIB)

Intel® oneAPI Toolkits

A complete set of proven developer tools expanded from CPU to Accelerators

oneAPI

Intel® oneAPI **Base** Toolkit

A core set of high-performance libraries and tools for building C++, SYCL and Python applications

Add-on **Domain-specific** Toolkits

Intel® oneAPI Tools for HPC
Deliver fast Fortran, OpenMP & MPI applications that scale

Intel® oneAPI Tools for IoT
Build efficient, reliable solutions that run at network's edge

Intel® oneAPI Rendering Toolkit
Create performant, high-fidelity visualization applications

Toolkits **powered by oneAPI**

Intel® AI Analytics Toolkit
Accelerate machine learning & data science pipelines end-to-end with optimized DL frameworks & high-performing Python libraries

Intel® Distribution of OpenVINO™ Toolkit
Deploy high performance inference & applications from edge to cloud

Latest version available 2022.2

DPCPP/SYCL introduction

Data Parallel C++

DPC++ = ISO C++ and Khronos SYCL

Parallelism, productivity, and performance for CPUs and accelerators

- Delivers accelerated computing by exposing hardware features
- Allows code reuse across hardware targets, while permitting custom tuning for specific accelerators
- Provides an open, cross-industry solution to single-architecture proprietary lock-in

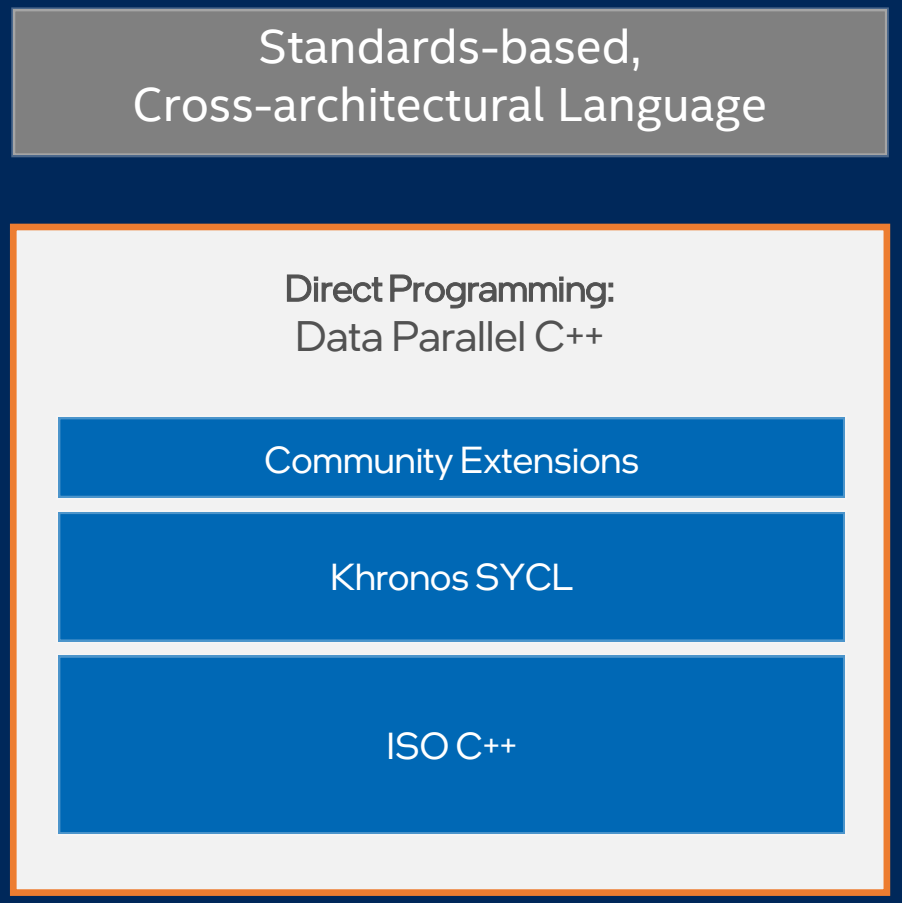
Based on C++ and SYCL

- Delivers C++ productivity benefits, using common, familiar C and C++ constructs
- Incorporates SYCL from the Khronos Group to support data parallelism and heterogeneous programming

Community Project to drive language enhancements

- Provides extensions to simplify data parallel programming
- Continues evolution through open and cooperative development

Apply your skills to the next innovation, not to rewriting software for the next hardware platform



The open source and Intel DPC++/C++ compiler supports Intel CPUs, GPUs, and FPGAs.
Codeplay announced a DPC++ compiler that targets Nvidia GPUs.

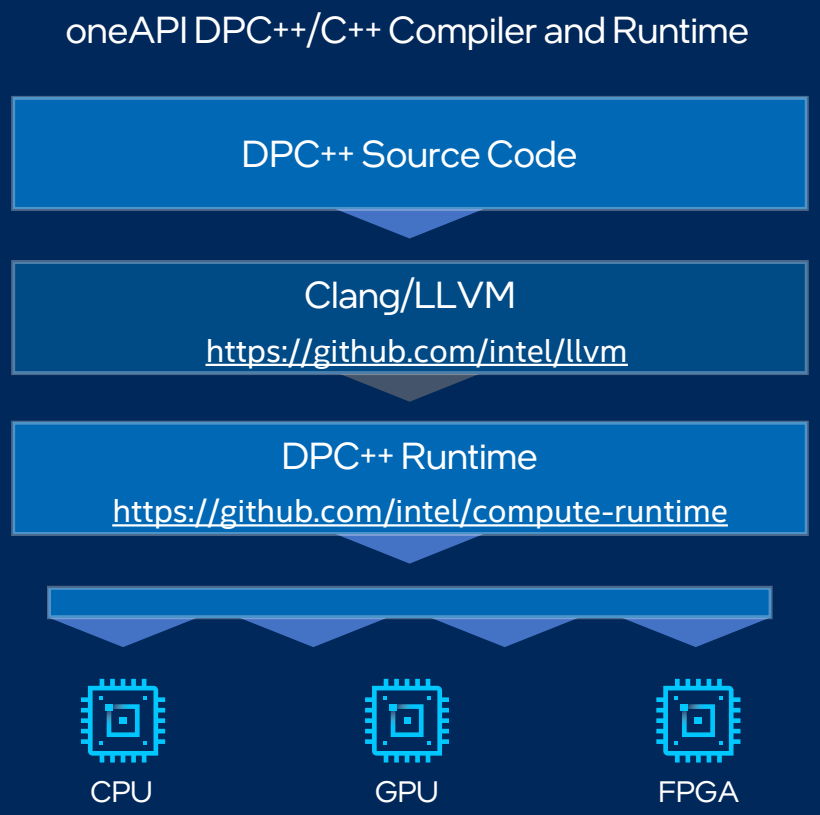
Intel® oneAPI DPC++/C++ Compiler

Parallel Programming Productivity & Performance

Compiler to deliver uncompromised parallel programming productivity and performance across CPUs and accelerators

- Allows code reuse across hardware targets, while permitting custom tuning for a specific accelerator
- Open, cross-industry alternative to single architecture proprietary language

Builds upon Intel's decades of experience in architecture and high-performance compilers



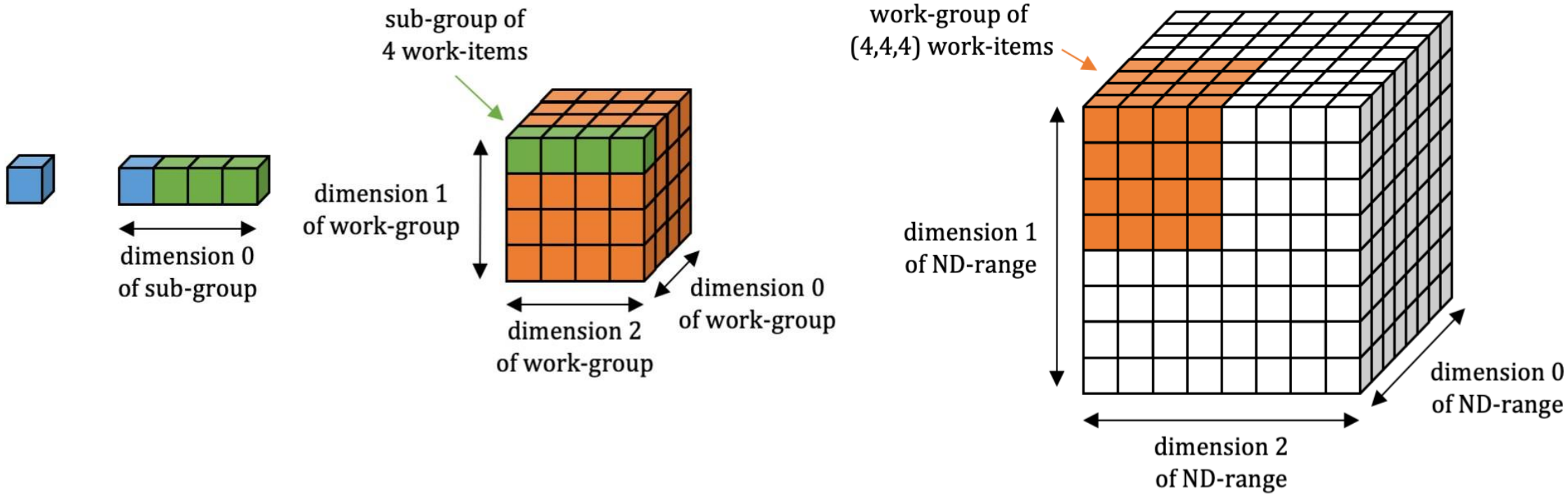
Code samples:

tinyurl.com/dpcpp-tests

tinyurl.com/oneapi-samples

There will still be a need to tune for each architecture.

DPC++ Thread Hierarchy and Mapping



Work-item

Cuda equivalent (CUDA thread)

Sub-group

Cuda equivalent (Warps)

Work-group

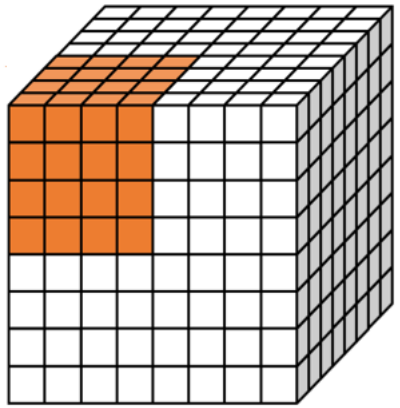
Cuda equivalent (Block)

ND-Range

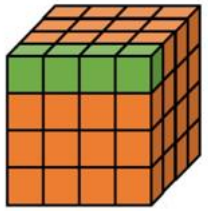
Cuda equivalent (Grid)

DPC++ Thread Hierarchy and Mapping

All work-items in a **work-group** are scheduled on one Compute Unit, which has its own local memory



All work-items in a **sub-group** are mapped to vector hardware



Anatomy of a DPC++ Application

```
#include <CL/sycl.hpp>
using namespace sycl;

int main() {
std::vector<float> A(1024), B(1024), C(1024);
// some data initialization
    {
      buffer bufA {A}, bufB {B}, bufC {C};
      queue q;
      q.submit([&](handler &h) {
          auto A = bufA.get_access(h, read_only);
          auto B = bufB.get_access(h, read_only);
          auto C = bufC.get_access(h, write_only);
          h.parallel_for(1024, [=](auto i){
              C[i] = A[i] + B[i];
          });
      });
    }
for (int i = 0; i < 1024; i++)
       std::cout << "C[" << i << "] = " << C[i] << std::endl;

}
```

Host code

Accelerator device code

Host code

Anatomy of a DPC++ Application

```
#include <CL/sycl.hpp>
using namespace sycl;

int main() {
std::vector<float> A(1024), B(1024), C(1024);
// some data initialization
    {
      buffer bufA {A}, bufB {B}, bufC {C};
      queue q;
      q.submit([&](handler &h) {
          auto A = bufA.get_access(h, read_only);
          auto B = bufB.get_access(h, read_only);
          auto C = bufC.get_access(h, write_only);
          h.parallel_for(1024, [=](auto i){
              C[i] = A[i] + B[i];
          });
      });
    }
for (int i = 0; i < 1024; i++)
       std::cout << "C[" << i << "] = " << C[i] << std::endl;

}
```

Application scope

Command group scope

Kernel scope

Application scope

Where is my “Hello World” code executed?

Device Selector

Get a device (any device):	`queue q (); // default_selector{}`

Create queue targeting a pre-configured classes of devices:	`queue q(cpu_selector{});`
	`queue q(gpu_selector{});`
	`queue q(intel::fpga_selector{});`
	`queue q(accelerator_selector{});`
	`queue q(host_selector{});` SYCL 1.2.1

Create queue targeting specific device (custom criteria):	`class custom_selector : public device_selector {`
	`  int operator()(…… // Any logic you want!`
	
	…
	`queue q(custom_selector{});`

default_selector

- DPC++ runtime scores all devices and picks one with highest compute power
- Environment variable

export SYCL_DEVICE_TYPE=GPU | CPU | HOST

new

export SYCL_DEVICE_FILTER={backend:device_type:device_num}

*will be available in oneAPI Update 1

tinyurl.com/dpcpp-env-vars for more details on env variables

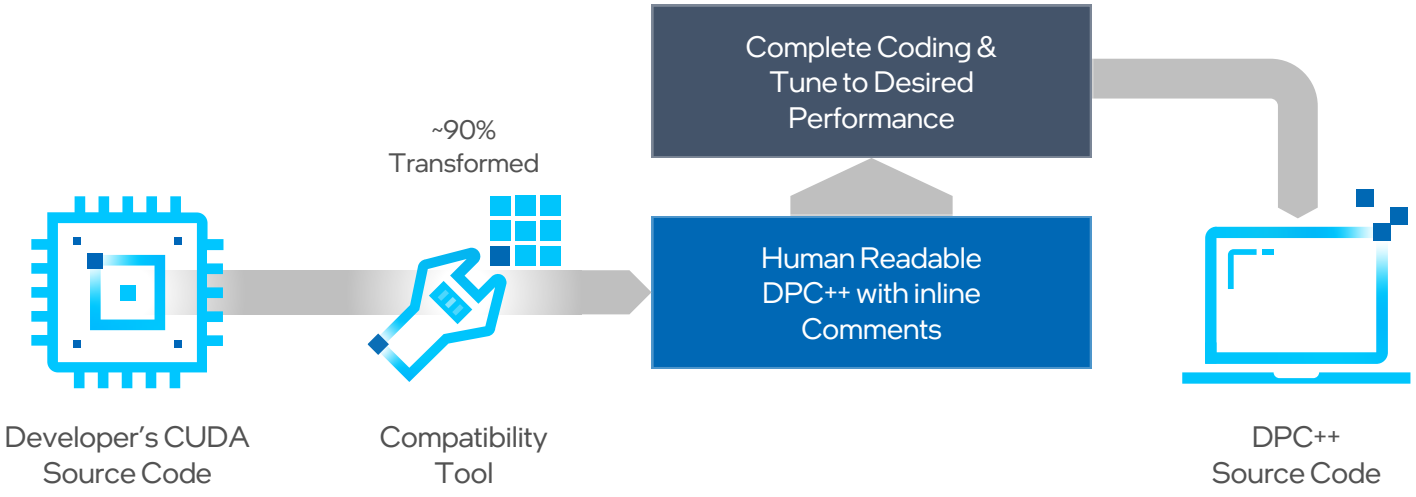
Intel® DPC++ Compatibility Tool (DPCT) introduction

Intel® DPC++ Compatibility Tool (DPCT)

Minimizes Code Migration Time

- Included in Intel® onAPI Base Toolkit
- Assist developers migrating code written in CUDA to DPC++, generating human readable code wherever possible
- 90%~95% of code typically migrates automatically
- Inline comments are provided to help developers finish porting the application
- Support windows* and Linux*, and well known IDEs.

Intel DPC ++ Compatibility Tool Usage Flow



Intel® DPC++ Compatibility Tool

Migration of Large Code Bases

1. Create a compilation database file

   intercept-build make

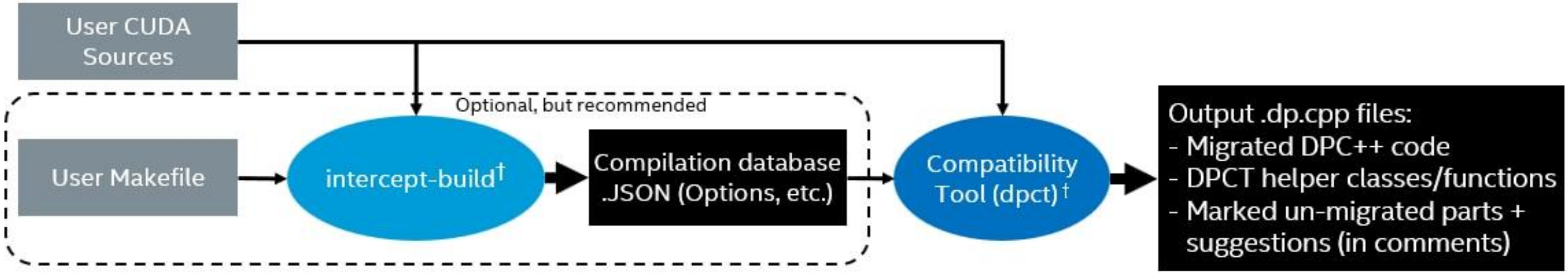
2. Migrate your source to DPC++

   dpct -p compile_commands.json -in-root=$PROJ_DIR -out-root=dpcpp_out *.cu

3. Verify the source for correctness and fix not migrated parts

https://tinyurl.com/intel-dpcpp-compatibility-tool

Diagram illustrate the workflow and the files generated when using the Compatibility Tool



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

Migration example

Header files

```cuda
#include <cuda.h>

#define VECTOR_SIZE 4
__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()
{
```

```sycl
#include <CL/sycl.hpp>
#include <dpct/dpct.hpp>
#define VECTOR_SIZE 4
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)];
}

int main()
{
```

Kernel Body

Mem Operation

```cuda
  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));
```

```sycl
  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);
```

Kernel Invoke

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

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

Mem Operation

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

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

Mem Operation

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

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

Main components in CUDA to SYCL Migration:

1. Header files
2. Memory operation
3. Kernel invocation
4. Thread/work item indexing
5. Synchronization
6. Error handling
7. Makefile

Header files

```
#include <cuda.h>
==>
#include <CL/sycl.hpp>
#include <dpct/dpct.hpp>
```

Memory operation

cudaMalloc

==>

sycl::malloc_device

- Convert to United Shared Memory(USM) approach in SYCL
- Can apply buffer approach later

Kernel invocation (1)

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

Kernel invocation (2)

```
needle_cuda_shared_1<<<dimGrid, dimBlock>>>(referrence_cuda, matrix_cuda
,max_cols, penalty, i, block_width);
```

==>

```
q_ct1.submit([&](sycl::handler &cgh) {
    sycl::accessor<int, 2, sycl::access_mode::read_write, sycl::access::target::local>
    ref_acc_ct1(sycl::range<2>(16, 16), cgh);
    cgh.parallel_for(sycl::nd_range<3>(dimGrid * dimBlock, dimBlock),
[=](sycl::nd_item<3> item_ct1) {
        needle_cuda_shared_1(referrence_cuda, matrix_cuda, max_cols, penalty, i,
block_width, item_ct1, ref_acc_ct1);});
});
```

(See more details about device memory handling in demo)

Kernel invocation (2)

```
needle_cuda_shared_1<<<dimGrid, dimBlock>>>(referrence_cuda, matrix_cuda
,max_cols, penalty, i, block_width);
```

==>

```
q_ct1.submit([&](sycl::handler &cgh) {
    sycl::accessor<int, 2, sycl::access_mode::read_write, sycl::access::target::local>
    ref_acc_ct1(sycl::range<2>(16, 16), cgh);
    cgh.parallel_for(sycl::nd_range<3>(dimGrid * dimBlock, dimBlock),
[=](sycl::nd_item<3> item_ct1) {
        needle_cuda_shared_1(referrence_cuda, matrix_cuda, max_cols, penalty, i,
block_width, item_ct1, ref_acc_ct1);});
});
```

(See more details about device memory handling in demo)

Kernel invocation (2)

```
needle_cuda_shared_1<<<dimGrid, dimBlock>>>(referrence_cuda, matrix_cuda
,max_cols, penalty, i, block_width);
```

==>

```
q_ct1.submit([&](sycl::handler &cgh) {
    sycl::accessor<int, 2, sycl::access_mode::read_write, sycl::access::target::local>
    ref_acc_ct1(sycl::range<2>(16, 16), cgh);
    cgh.parallel_for(sycl::nd_range<3>(dimGrid * dimBlock, dimBlock),
[=](sycl::nd_item<3> item_ct1) {
        needle_cuda_shared_1(referrence_cuda, matrix_cuda, max_cols, penalty, i,
block_width, item_ct1, ref_acc_ct1);});
});
```

(See more details about device memory handling in demo)

Thread/work item indexing

```
__global__ void VectorIncreKernel (float *A)
{ A[threadIdx.x] = threadIdx.x + 1.0f;  }
```

==>

```
void VectorIncreKernel (float *A, sycl::nd_item<1> item_ct1)
{ A[item_ct1.get_local_id(0)] = item_ct1.get_local_id(0) + 1.0f; }
```

- SYCL index the work item (thread) by N_Dimensional_Range item, as the counter part of the threadIdx in CUDA
- SYCL needs call get_local_id() method to retrieve the index

Synchronization

```
__syncthreads();
==>
item_ct1.barrier();
```

- DPCT1065:9: Consider replacing sycl::nd_item::barrier() with sycl::nd_item::barrier(sycl::access::fence_space::local_space) for better performance if there is no access to global memory.

Error handling

DPCT1003:10: Migrated API does not return error code. (*, 0) is inserted. You may need to rewrite this code.

```
err_code = cudaDriverGetVersion(&version);
```

DPCT1009:13: SYCL uses exceptions to report errors and does not use the error codes. The original code was commented out and a warning string was inserted. You need to rewrite this code.

```
printf("Error \"%s\" checking driver version: %s.\n",
cudaGetErrorName(err_code), cudaGetErrorString(err_code));
```

Makefile

--gen-build-script (automatically generate Makefile)

```
CXX = nvcc
TARGET = needleman_wunsch_cu
SRCS = src/needle.cu
DEPS = src/needle_kernel.cu src/needle.h
==>
CXX = dpcpp
TARGET = needleman_wunsch_dpcpp
SRCS = src/needle.dp.cpp
DEPS = src/needle_kernel.dp.cpp src/needle.h
```

Best practice 1: Before executing the Compatibility Tool

- Ensure project source files are syntactically correct
- “make clean” before running “intercept-build make”
- For complex projects, use intercept-build command to create a compilation database
- Code modifications needed prior to migration due to differences between Clang and nvcc

Best practice 2: When migrating your CUDA project to DPC++ using the dpct executable

If you have trouble migrating all project source files at once, it may be helpful to migrate one file at a time incrementally.

https://www.intel.com/content/www/us/en/developer/articles/technical/intel-dpcpp-compatibility-tool-best-practices.html

- Unified Shared Memory (USM) Usage
- DPCT Helper Functions
- Timing Issues
- Common Runtime Issues

- Unified Shared Memory (USM) Usage
  - Unified Shared Memory (USM), supported in DPC++, is a feature that allows a pointer-based approach to manage host and device memory. When using the Compatibility Tool, the migrated DPC++ code will use USM as the default memory management method. When compared to using SYCL buffers, USM produces less volume of code and allows the `dpct` to support more memory-related APIs.

- DPCT Helper Functions
  - The Compatibility Tool will use helper functions and classes in migrated DPC++ code. Some examples of utility functions provided are memory management tasks such as `dpct_malloc`, `dpct_memcpy` and `get_buffer`, and device management tasks such as `get_default_queue` and `get_default_context`.
  - The associated files are located in `<dpcpp-ct installation directory>/latest/include/dpct`, the main header file is `dpct.hpp`, and the namespace is `dpct::`.
  - These DPCT helper functions are intended for migrated code only and not for any other purpose. If you write new DPC++ code, it’s not recommended to use these DPCT helpers.

- Timing Issues
  - Timing-related blocks may need manual editing, since calculation of time span is implementation-specific. Rewrite code containing any language-specific features and library dependencies to ensure equivalency.
  - For example, one potential issue involves profiling-related timer calls. If you're using timer functions from CUDA samples such as `sdkCreateTimer` or `sdkStartTimer`, you may need to reimplement those calls.
- Common Runtime Issues
  - **"OpenCL API failed. OpenCL API returns: -52 (CL_INVALID_KERNEL_ARGS)"**: This error indicates that some pointers are not properly set before execution on the device. Ensure all pointers are initialized to valid memory allocation or NULL before using them in parallel_for device execution.
  - **"OpenCL API failed. OpenCL API returns: -54 (CL_INVALID_WORKGROUP_SIZE)":** Different accelerators have hardware differences that limit the maximum number of work items in a workgroup. For example, NVIDIA* hardware often limits the workgroup size at 512 while Intel® Gen9 graphics is limited at 256. If this error is encountered, the workgroup size set at kernel launch needs to be adjusted according to hardware limits.
  - **"Caught asynchronous SYCL exception"**: Follow the error message where the exception was caught and make the necessary changes.

Code example Rodinia Code base

Rodinia

Rodinia is a Benchmark Suite for Heterogeneous Computing and represents collection of parallel programs which targets heterogeneous computing platforms written in several parallel languages such as CUDA, OpenCL, OpenMP.

In this project, the Intel's oneAPI tool DPCT is evaluated for porting the Rodinia Benchmark from CUDA to SYCL.

20 of 23 benchmarks are migrated to SYCL without much programming effort and few developer interventions.

Among the advantage of SYCL worthwhile to mention are the easiness to perform the parallel application on several devices such as CPU, GPUs without vendor restriction.

Rodinia: Accelerating Compute-Intensive Applications with Accelerators

A vision of heterogeneous computer systems that incorporate diverse accelerators and automatically select the best computational unit for a particular task is widely shared among researchers and many industry analysts; however, there are no agreed-upon benchmarks to support the research needed in the development of such a platform. There are many suites for parallel computing on general-purpose CPU architectures, but accelerators fall into a gap that is not covered by previous benchmark development. Rodinia is released to address this concern.

The Rodinia Benchmark Suite, version 3.1 (Version history) Rodinia is designed for heterogeneous computing infrastructures with OpenMP, OpenCL and CUDA implementations.

Applications	Dwarves	Domains	Parallel Model	Incre. Ver.
Leukocyte	Structured Grid	Medical Imaging	CUDA, OMP, OCL	â□□
Heart Wall	Structured Grid	Medical Imaging	CUDA, OMP, OCL	
MUMmerGPU	Graph Traversal	Bioinformatics	CUDA, OMP	
CFD Solver	Unstructured Grid	Fluid Dynamics	CUDA, OMP, OCL	
LU Decomposition	Dense Linear Algebra	Linear Algebra	CUDA, OMP, OCL	â□□
HotSpot	Structured Grid	Physics Simulation	CUDA, OMP, OCL	
Back Propogation	Unstructured Grid	Pattern Recognition	CUDA, OMP, OCL	
Needleman-Wunsch	Dynamic Programming	Bioinformatics	CUDA, OMP, OCL	â□□
Kmeans	Dense Linear Algebra	Data Mining	CUDA, OMP, OCL	

Needleman-Wunsch Algorithm

- The Needleman–Wunsch algorithm is an algorithm used in bioinformatics to align protein or nucleotide sequences.

- Needleman–Wunsch is a dynamic programing algorithm. essentially divides a large problem into a series of smaller problems, and it uses the solutions to the smaller problems to find an optimal solution to the larger problem.

- Needleman–Wunsch algorithm is widely used for optimal global alignment. The algorithm assigns a score to every possible alignment, and the purpose of the algorithm is to find all possible alignments having the highest score.

- Needleman–Wunsch algorithm has a stencil type of parallel patterns (no need of collective operation).

Demo

- obtain example at (google oneapi-samples):
  https://github.com/oneapi-src/oneAPI-samples/tree/master/Tools/Migration/rodinia-nw-dpct
  (2 .cu files, 1 .h file in src, and Makefile)

- prepare environment:
  export PATH=/usr/local/cuda/bin:$PATH
  which nvcc
  module load oneapi/2022.2.0
  which intercept-build
  which dpct

- intercept-build make
- cat compile_commands.json
- dpct -p compile_commands.json --in-root=. --out-root=. --process-all --keep-original-code --gen-build-script
- dpcpp needle.dp.cpp

  Review/handle warnings and errors, more info in "Diagnostics Reference" page of "Intel® DPC++ Compatibility Tool Developer Guide and Reference"
- review:

  https://github.com/oneapi-src/oneAPI-samples/blob/master/Tools/Migration/rodinia-nw-dpct/README.md

- dpcpp needle.dp.cpp
- ./a.out 4096 16
- redfine BLOCK_SIZE to 128 in needle.h

  dpcpp needle.dp.cpp

  ./a.out 4096 16

intel®

Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation. Learn more at intel.com or from the OEM or retailer.

Your costs and results may vary.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

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. https://software.intel.com/en-us/articles/optimization-notice

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. See backup for configuration details. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Backup Slides

Intel® oneAPI Toolkits Free Availability

Get Started Quickly

Code Samples, Quick-start Guides, Webinars, Training

software.intel.com/oneapi

Run the tools locally

- Downloads
- Repositories
- Containers

Or run the tools in intel DevCloud

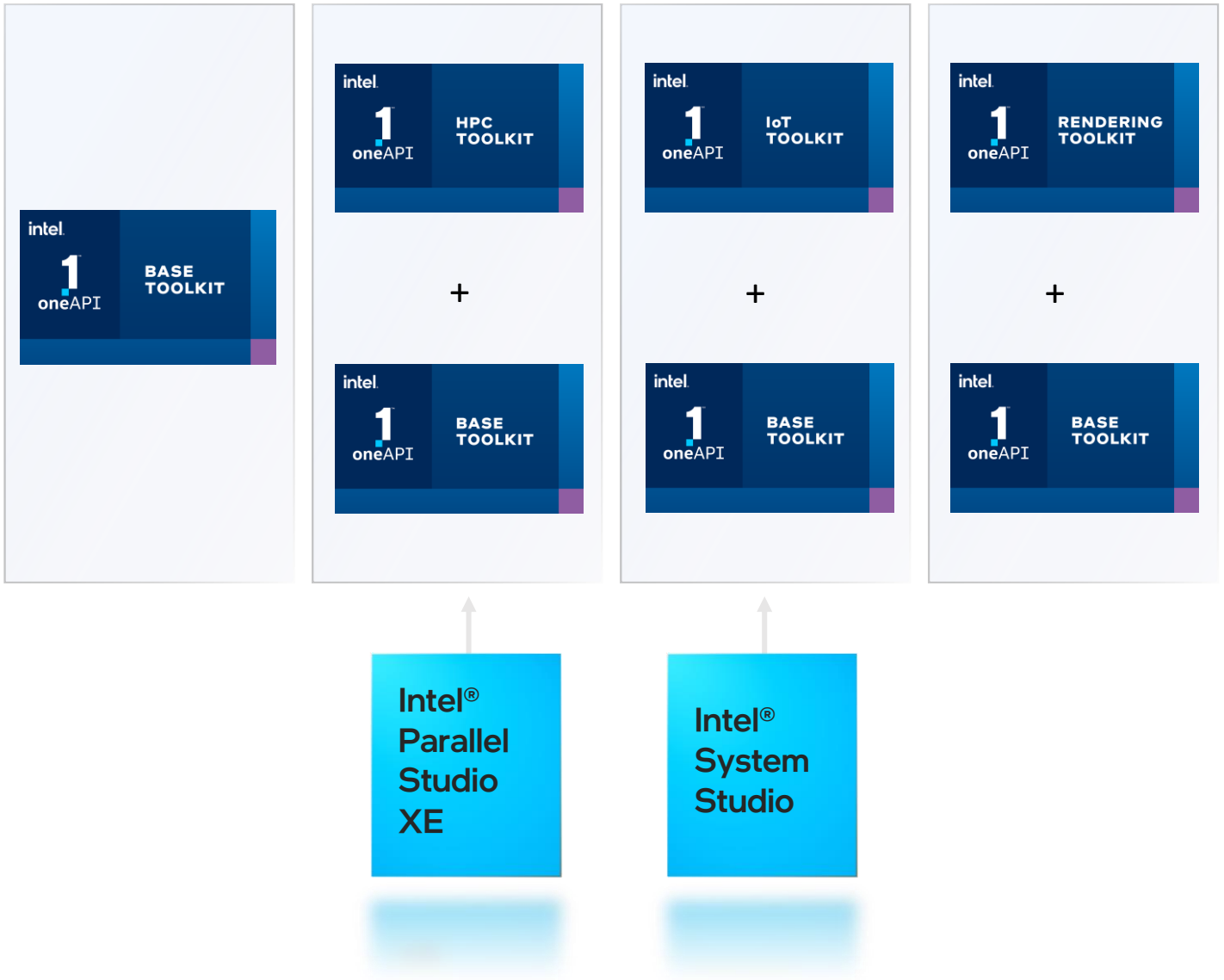
- 1 Minute to Code
- No Hardware Acquisition
- No Download, Install or Configuration
- Samples & Tutorials
- Supports Jupyter Notebooks, Visual Studio Code

Get Up & Running In Seconds!

Commercial Toolkits Deliver Priority Support

Next Generation of Commercial Intel® Software Development Products

- Worldwide support from Intel technical consulting engineers
- Prior commercial tool suites, Intel® Parallel Studio XE and Intel® System Studio, transition to oneAPI products



oneAPI Resources

software.intel.com/oneapi

Get Started

- software.intel.com/oneapi
- Documentation + dev guides
- Code Samples
- Intel® DevCloud

Industry Initiative

- oneAPI.io
- oneAPI open Industry Specification
- Open-source Implementations

Learn

- Training: Webinars & courses
- Intel® DevMesh Innovator Projects
- Summits & Workshops: Live & on-demand virtual workshops, community-led sessions
- Training by certified oneAPI experts worldwide for HPC & AI

Ecosystem

- Community Forums
- Intel® DevMesh Innovator Projects
- Academic Programs: oneAPI Centers of Excellence: research, enabling code, curriculum, teaching

Codeplay Launched

Data Parallel C++ Compiler for Nvidia GPUs

- Developers can retarget and reuse code between NVIDIA and Intel compute accelerators from a single source base
- Codeplay is the first oneAPI industry contributor to implement a developer tool based on oneAPI specifications
- They leveraged the DPC++ LLVM-based open source project that Intel established
- Codeplay is a key driver of the Khronos SYCL standard, upon which DPC++ is based
- More details in the Codeplay blog post
- Build DPC++ toolchain with support for NVIDIA CUDA:

tinyurl.com/dpcpp-cuda-be

tinyurl.com/dpcpp-cuda-be-webinar

Other News

Codeplay Brings NVIDIA GPU Support to Industry-Standard Math Library

Intel Open Sources the oneAPI Math Kernel Library Interface