oneAPI

Cross-Architecture Programming for Accelerated Compute, Freedom of Choice for Hardware

Intel® DPC++ Compatibility Tool

September 2023

intel

Agenda

- Intel® DPC++ Compatibility Tool overview
- vecAdd Migration Example
- Project migration (Rodinia NW)

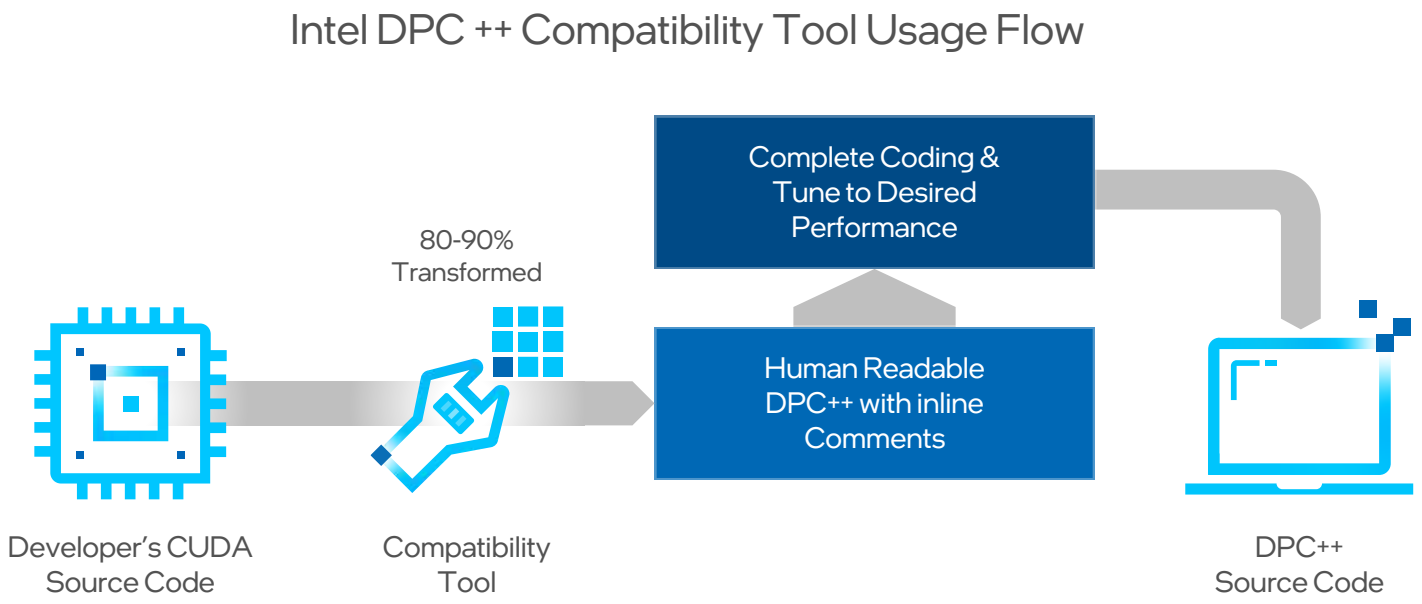
Intel® DPC++ Compatibility Tool

Minimizes Code Migration Time

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

~90-95% of code typically migrates automatically[1]

Inline comments are provided to help developers finish porting the application



[1]Intel estimates as of September 2021. Based on measurements on a set of 70 HPC benchmarks and samples, with examples like Rodinia, SHOC, PENNANT. Results may vary.

Open Source CUDA* to SYCL* Code Migration Tool
SYCLomatic

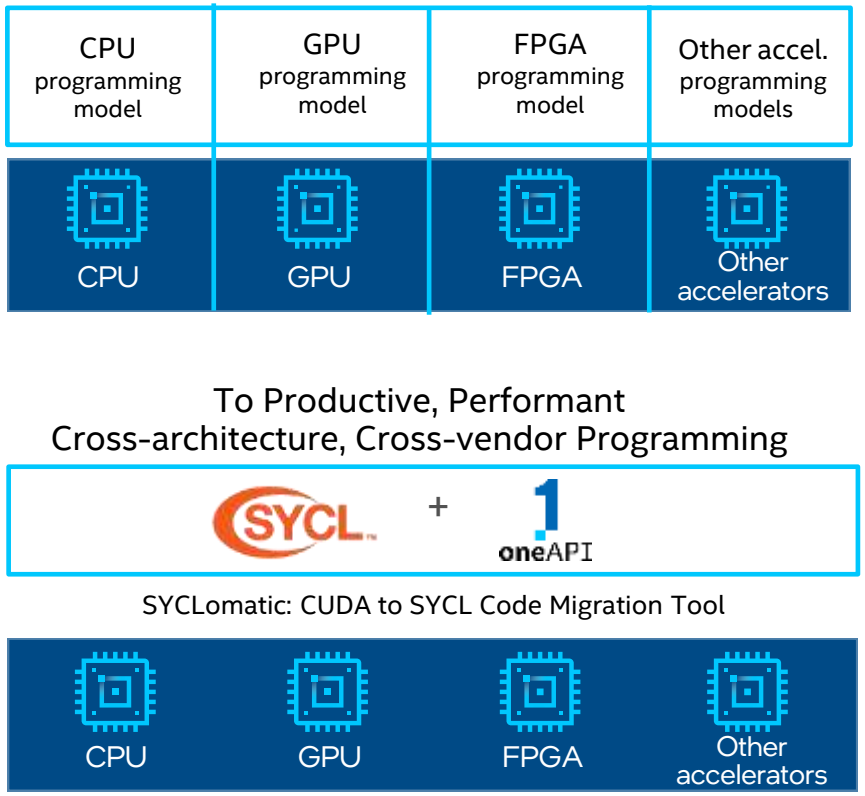
SYCL with oneAPI open, cross-architecture, standards-based programming

- Allows developers to expand investments across architectures
- Provides choice in hardware & freedom from proprietary, single-vendor lock-in

Intel is open sourcing a CUDA to SYCL migration tool: **SYCLomatic**

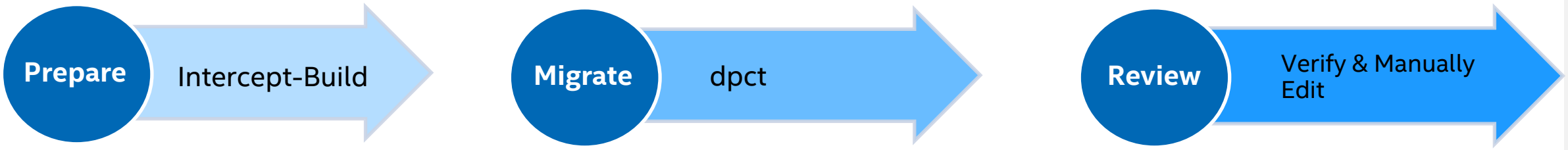
- A productive path to create single-source, portable code for hardware targets regardless of vendor
- Simplifies development while delivering performance, reduces time & costs for code maintenance
- A community to share, collaborate & contribute software technologies
- On GitHub:
  - github.com/oneapi-src/SYCLomatic
  - Use the tool, please provide feedback!

Simplify Heterogeneous Development



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

tinyurl.com/intel-dpcpp-compatibility-tool

ZIB CUDA Code Migration to SYCL Delivers Performance across Architectures

Freedom of Choice

ZUSE INSTITUTE BERLIN

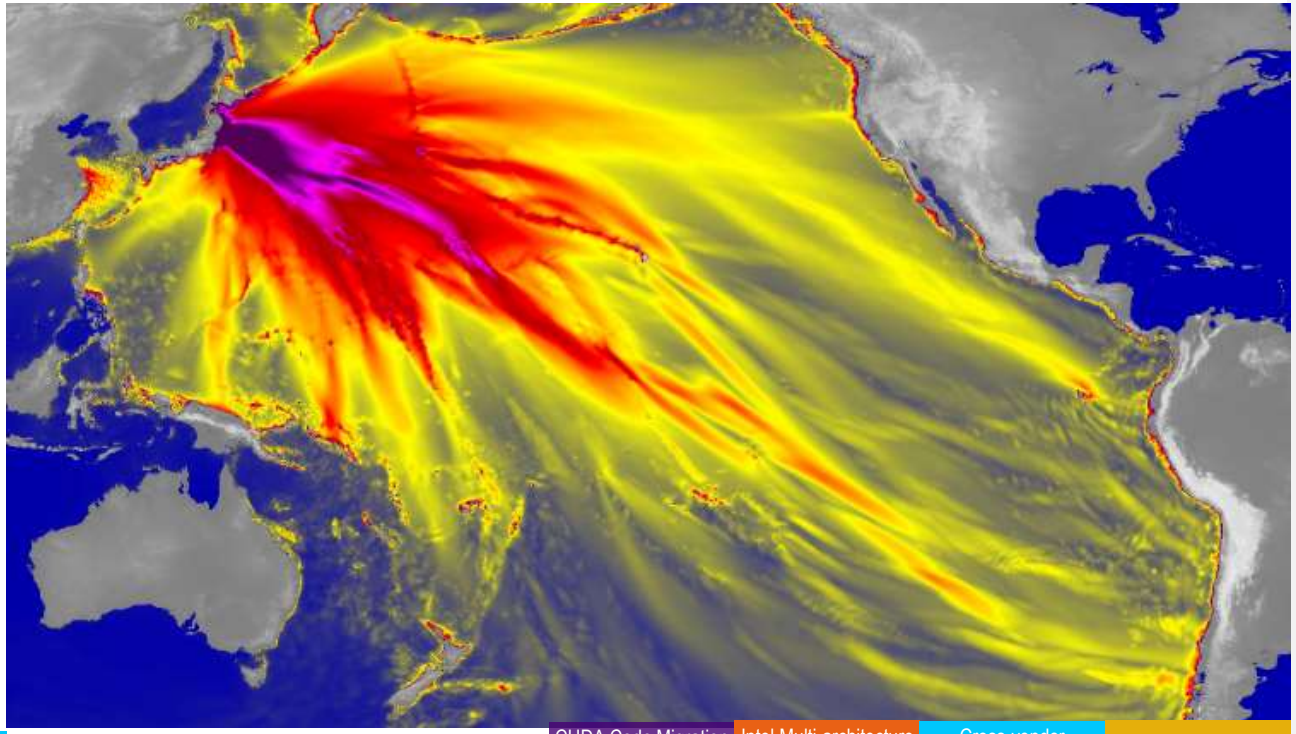
ZIB ported *easyWave* application from CUDA to SYCL/Data Parallel C++ (DPC++) delivering performance across multi-architectures

- Ported *easyWave* written in CUDA to DPC++ efficiently using the Intel® DPC++ Compatibility Tool
- Achieved strong performance across Intel CPU, GPU and FPGA architectures, and within 5% of CUDA performance on Nvidia P100

Podcast | Video


Intel® oneAPI Base Toolkit + FPGA Add-on

DPC++ is oneAPI's implementation in SYCL.



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

CUDA Code Migration to DPC++ for Single Source Code | Intel Multi-architecture Deployments (CPU, GPU, FPGA) | Cross-vendor Multi-architecture Deployments | Used Multiple tools in Intel® oneAPI Toolkits

github.com/christgau/easywave-sycl

For details see XPUG presentation: ZIB: oneAPI case study with the tsunami simulation EasyWave from CUDA to DPC++ and back to Nvidia GPUs and FPGAs – Configuration: Compute Domain: approx. 2000 x 1400 cells; 10 hours simulation time. Same code produces valid data on CPU, Intel GPUs, and FPGA. oneAPI performance evolution on DevCloud Coffee Lake Gen9.5 GT2 iGPU using code migrated from CUDA to Data Parallel C++ using the Intel® DPC++ Compatibility tool, build with open source Intel LLVM w/ CUDA support (contribution by Codeplay). Typical application run on Nvidia P100-SXM2-16GB shows migrated DPC++ code runs only 4% slower than CUDA code. Results: Same DPC++ code can target different platforms (almost) without modifications. • Performance is on par with architecture-specific CUDA code. For workloads and configurations visit www.Intel.com/PerformanceIndex.
Results may vary. Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

What's New?

- Improved conformance with SYCL 2020 specification
- Can use old build logs as input to start migration
- Support of some inline assembly in CUDA code
- Enhanced CUB API migration, Driver API migration in 2022
- --gen-build-script option to automatically create makefiles
- --no-incremental-migration to ignore existing migrated output
- Supported CUDA versions are: 8.0, 9.x, 10.1, 10.2, 11.0 - 11.8, 12.0, 12.1
- Improved support for cuFFT*, cuBLAS, cuSPARSE, Thrust, CUB, Driver API migration (partial)
- --use-custom-helper to customize the helper header files for migrated code
- --no-dpcpp-extensions to avoid specific DPC++ in a migrated code
- --assume-nd-range-dim to control dimensionality preferences

Migration example

```cuda
#include <cuda.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));



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



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

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

Header files

Kernel

Mem alloc

Kernel call

Mem copy

Mem free

```sycl
#include <sycl/sycl.hpp>
#include <dpct/dpct.hpp>
#define VECTOR_SIZE 256
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()
{
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);

  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);
    });
  });


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

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

tinyurl.com/cuda-sycl-equivalence

Queue Selection

```
/// Util function to get the default queue of current device in
/// dpct device manager.
static inline cl::sycl::queue &get_default_queue() {
  return dev_mgr::instance().current_device().default_queue();
}
```

```
cl::sycl::queue &default_queue() { return *_default_queue; }
```

```
 device_ext(const cl::sycl::device &base) : cl::sycl::device(base) {
#ifdef DPCT_USM_LEVEL_NONE
    _default_queue = new cl::sycl::queue(base, exception_handler);
#else
    _default_queue = new cl::sycl::queue(base, exception_handler,
                                         cl::sycl::property::queue::in_order());
#endif
    _queues.insert(_default_queue);
    _saved_queue = _default_queue;
  }
```

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
- Check that dpct recognized the input code as “valid”
  - default C++ std, macro definitions and include paths
- Start with a clean project - “make clean” before running “intercept-build make”

Demo: vecAdd

- *Download the oneAPI samples:* git clone https://github.com/oneapi-src/oneAPI-samples.git
- *cd oneAPI-samples/Tools/Migration/vector-add-dpct/src*
- *dpct vector_add.cu*
  - *Use the "--cuda-include-path=/your/cuda/path" option for non-default installation paths of CUDA*

```
NOTE: Could not auto-detect compilation database for file 'vector_add.cu' in '/home/mkirchha/Work/hlrs-workshop/oneAPI-
   samples/Tools/Migration/vector-add-dpct/src' or any parent directory.

The directory "dpct_output" is used as "out-root"
   Parsing: /home/mkirchha/Work/hlrs-workshop/oneAPI-samples/Tools/Migration/vector-add-dpct/src/vector_add.cu
   Analyzing: /home/mkirchha/Work/hlrs-workshop/oneAPI-samples/Tools/Migration/vector-add-dpct/src/vector_add.cu
   Migrating: /home/mkirchha/Work/hlrs-workshop/oneAPI-samples/Tools/Migration/vector-add-dpct/src/vector_add.cu

Processed 1 file(s) in -in-root folder "/home/mkirchha/Work/hlrs-workshop/oneAPI-samples/Tools/Migration/vector-add-
   dpct/src"

See Diagnostics Reference to resolve warnings and complete the migration:

https://software.intel.com/content/www/us/en/develop/documentation/intel-dpcpp-compatibility-tool-user-
   guide/top/diagnostics-reference.html
```

- File vector_add.dp.cpp is generated in dpct_output directory

Demo: Rodinia NW

- *cd ~/dpct_demo/oneAPI-samples/Tools/Migration/rodinia-nw-dpct*
- *make clean*

1. *intercept-build make*

```
cat compile_commands.json
[
    {
        "command": "nvcc -c  -o needleman_wunsch_cu -D__CUDACC__=1 src/needle.cu",
        "directory": "/home/u136312/oneAPI-samples/Tools/Migration/rodinia-nw-dpct",
        "file": "/home/u136312/oneAPI-samples/Tools/Migration/rodinia-nw-
dpct/src/needle.cu"
    }
]
```

clang.llvm.org/docs/JSONCompilationDatabase.html

2. *dpct --cuda-include-path=/home/u136312/include -p compile_commands.json --in-root=. --out-root=migration*

```
warning: DPCT1065:0: 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.

…

warning: DPCT1043:12: The version-related API is different in SYCL. An initial code was generated, but you need to
adjust it.

…

warning: DPCT1101:20: 'BLOCK_SIZE' expression was replaced with a value. Modify the code to use the original expression,
provided in comments, if it is correct.
```

Addressing Warnings in Migrated Code

warning: DPCT1043:12: The version-related API is different in SYCL. An initial code was generated, but you need to adjust it.

warning: 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.

- remove unnecessary code processing error codes
- need to update the code with correct SYCL device API

needle.dp.cpp

```
int version = 0;
dpct::err0 err_code = 999;
/* ...dpct generated comments... */
err_code = DPCT_CHECK_ERROR(version = dpct::get_current_device().get_major_version());
if (err_code != 0)
/* ...dpct generated comments... */
    printf("Error \\"%s\\" checking driver version: %s.\\n",
    "cudaGetErrorName not supported" /*cudaGetErrorName(err_code)*/,
    "cudaGetErrorString not supported" /*cudaGetErrorString(err_code)*/);
else
    printf("CUDA driver version: %d.%d\n", version/1000, version%1000/10);
```

```
std::string version = dpct::get_current_device().get_info<sycl::info::device::version>();
printf("SYCL device version: %s\n", version.c_str());
```

Addressing Warnings in Migrated Code

Check warning DPCT1049:

```
 /*
DPCT1049:5: The workgroup size passed to the SYCL kernel may exceed the limit.
To get the device limit, query info::device::max_work_group_size. Adjust the
workgroup size if needed.
*/
        q_ct1.submit([&](sycl::handler &cgh) {
            sycl::range<2> temp_range_ct1(17 /*BLOCK_SIZE+1*/,
                                          17 /*BLOCK_SIZE+1*/);
            sycl::range<2> ref_range_ct1(16 /*BLOCK_SIZE*/, 16 /*BLOCK_SIZE*/);
```

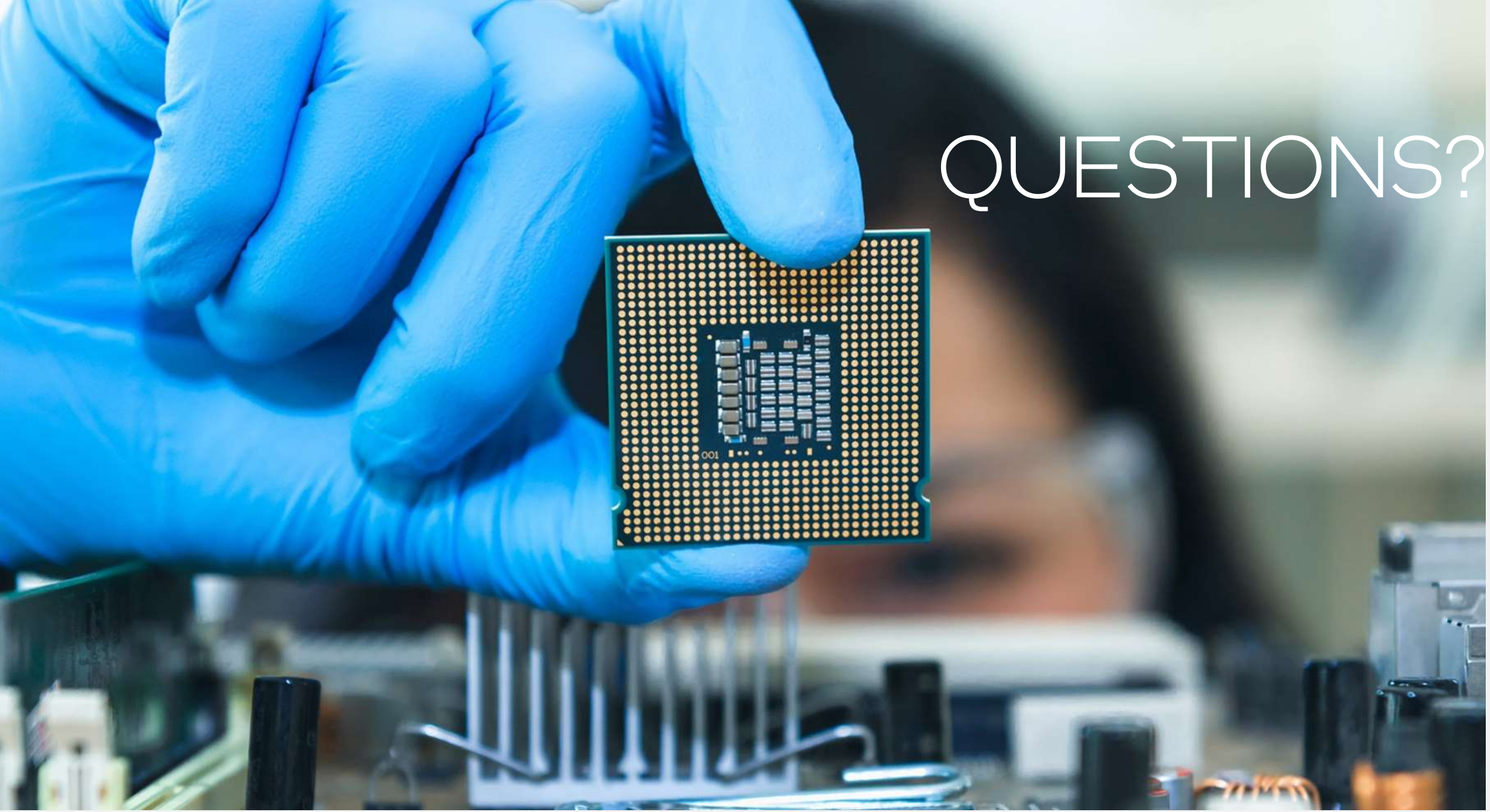
Once migration is completed, compile DPC++ code and run via make commands:

- *icpx -fsycl -o needleman_wunsch needle.dp.cpp*
- *./needleman_wunsch 4096 16*

```
WG size of kernel = 128

Start Needleman-Wunsch

Processing top-left matrix

Processing bottom-right matrix
```

Case Studies

- https://www.oneapi.io/events/devcon2021isc/
  - https://www.oneapi.io/event-sessions/experiences-with-adding-sycl-support-to-gromacs/
  - https://www.oneapi.io/event-sessions/application-optimization-with-cache-aware-roofline-model-and-intel-oneapi-tools/
  - https://www.oneapi.io/event-sessions/porting-namd-oneapi-dpc/
  - https://www.oneapi.io/event-sessions/evaluating-cuda-portability-with-hipcl-dpct/
- https://www.hlrn.de/doc/display/PUB/Joint+NHR@ZIB+-+INTEL+++oneAPI+Workshop
  - easyWave - A Tsunami Simulations Application
  - Ginkgo – a sparse linear algebra library for OneAPI Hardware


QUESTIONS?

Notices & Disclaimers

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

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

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

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


intel