oneAPI

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

Intel® DPC++ Compatibility Tool

March 2022

intel

All information provided in this deck is subject to change without notice.
Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

Agenda

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

Intel® DPC++ Compatibility Tool

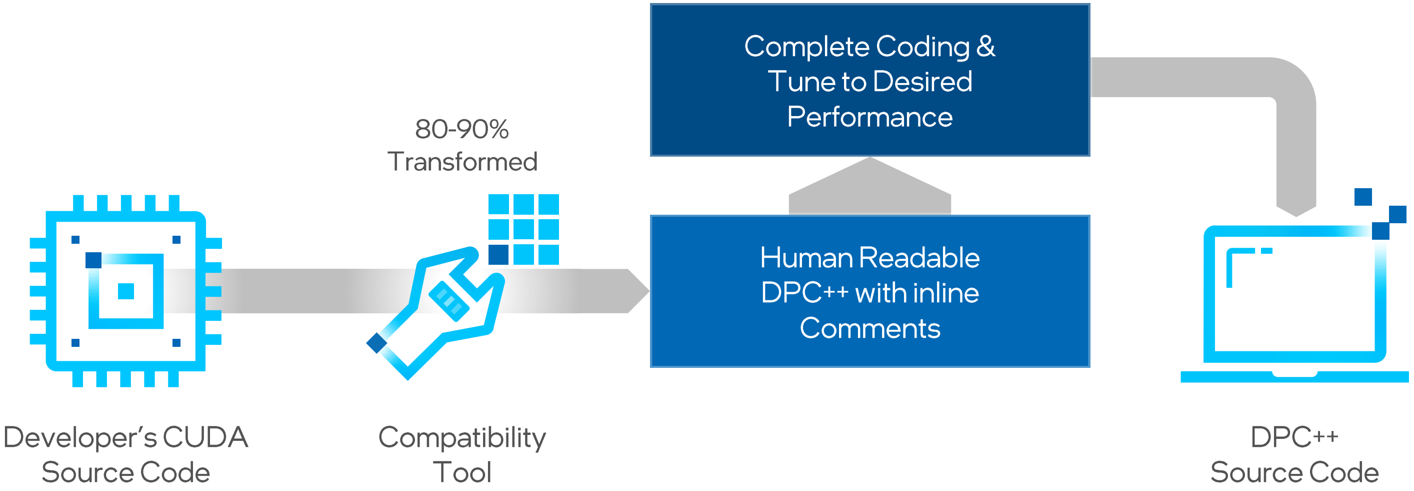
Minimizes Code Migration Time

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

~80-90% of code typically migrates automatically

Inline comments are provided to help developers finish porting the application

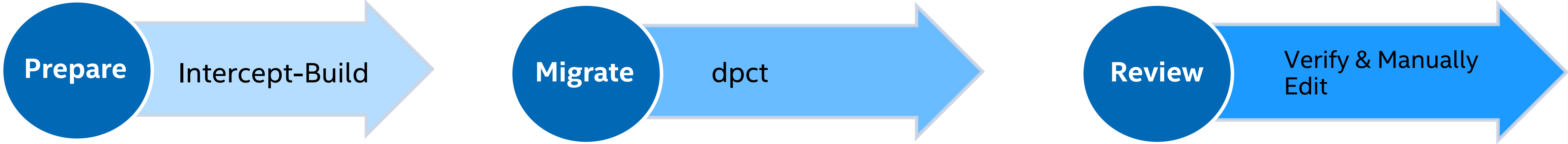
Intel DPC ++ Compatibility Tool Usage Flow



Complete Coding & Tune to Desired Performance
80-90% Transformed
Human Readable DPC++ with inline Comments
Developer's CUDA Source Code
Compatibility Tool
DPC++ Source Code

Intel® DPC++ Compatibility Tool

Migration of Large Code Bases



Prepare

Intercept-Build

Migrate

dpct

Review

Verify & Manually Edit

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

Migration example

```
#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()
{


  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

```
#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()
{
  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

intel.

Queue Selection

```
/// Util function to get the defualt 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”

oneAPI Development Example

ZUSE INSTITUTE BERLIN

ZIB ported *EasyWave* application from CUDA to DPC++ delivering performance across multi-architectures

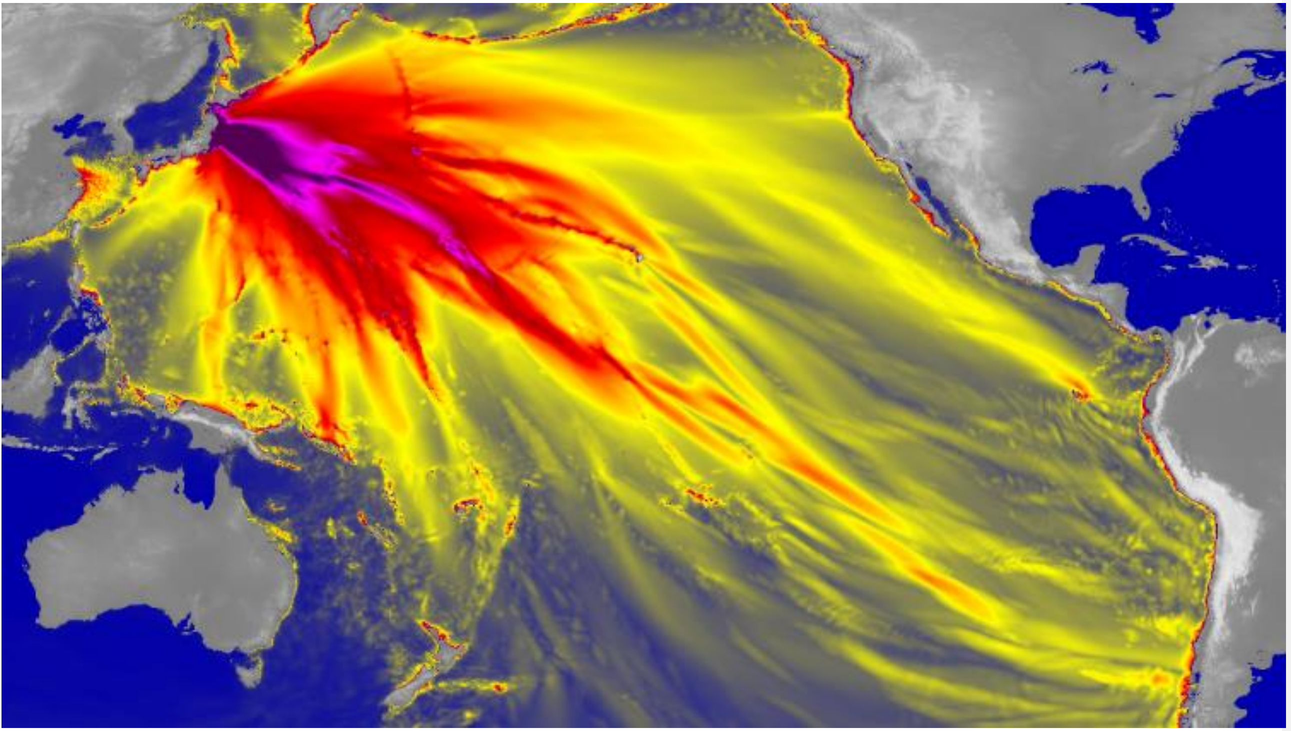
- Ported EasyWave written in CUDA to Data Parallel C++ efficiently using the Intel® DPC++ Compatibility Tool
- Achieved strong performance across Intel CPU, GPU and FPGA architectures, and within 5% of CUDA performance on Nvidia P100

oneAPI

intel. XEON

Xe

intel. STRATIX 10



Visualization of EasyWave tsunami simulation application
Courtesy Zuse Institute Berlin (ZIB)

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

easyWave - A Tsunami Simulations Application

Getting Started on DevCloud

- devcloud.intel.com
- qsub -I -l nodes=1:gpu:ppn=2 -d .
- sycl-ls (control devices via SYCL_DEVICE_FILTER)
- Compile and run simple vecAdd code
- export SYCL_PI_TRACE=1
- export SYCL_DEVICE_FILTER=level_zero

Demo

Pre-requisites on your own system:

- Get the latest oneAPI Base Toolkit:

  https://software.intel.com/content/www/us/en/develop/tools/oneapi/base-toolkit/download.html

- Set the environment, e.g. source /opt/intel/oneapi/setvars.sh

or use Intel DevCloud devcloud.intel.com

- git clone https://github.com/oneapi-src/oneAPI-samples.git
- cd oneAPI-samples/Tools/Migration/
- dpct --help

NOTE: You need to have CUDA* headers available to migrate the code

vecAdd

- *cd vector-add-dpct/src*
- *dpct --cuda-include-path=/home/u136312/include vector_add.cu*

```
NOTE: Could not auto-detect compilation database for file 'vector_add.cu' in '/home/u136312/oneAPI-samples/Tools/Migration/vector-add-dpct/src' or any parent directory.
The directory "dpct_output" is used as "out-root"
Processing: /home/u136312/oneAPI-samples/Tools/Migration/vector-add-dpct/src/vector_add.cu
/home/u136312/oneAPI-samples/Tools/Migration/vector-add-dpct/src/vector_add.cu:32:14: warning: DPCT1003:0: Migrated API does not return error code. (*, 0) is inserted. You may need to rewrite this code.
    status = cudaMemcpy(Result, d_C, VECTOR_SIZE*sizeof(float), cudaMemcpyDeviceToHost);
             ^
Processed 1 file(s) in -in-root folder "/home/u136312/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

vecAdd

- *dpct --cuda-include-path=/home/u136312/include --enable-ctad --out-root=test1 vector_add.cu*

  diff dpct_output/vector_add.dp.cpp test1/vector_add.dp.cpp

```
32,33c32,33
<       cgh.parallel_for(sycl::nd_range<3>(sycl::range<3>(1, 1, VECTOR_SIZE),
<                                          sycl::range<3>(1, 1, VECTOR_SIZE)),
---
>       cgh.parallel_for(sycl::nd_range(sycl::range(1, 1, VECTOR_SIZE),
>                                       sycl::range(1, 1, VECTOR_SIZE)),
```

- *dpct --cuda-include-path=/home/u136312/include --enable-ctad --out-root=test2 --keep-original-code vector_add.cu*

```
/* DPCT_ORIG __global__ void VectorAddKernel(float* A, float* B, float* C)*/
void VectorAddKernel(float *A, float *B, float *C, sycl::nd_item<3> item_ct1)
{
/* DPCT_ORIG     A[threadIdx.x] = threadIdx.x + 1.0f;*/
    A[item_ct1.get_local_id(2)] = item_ct1.get_local_id(2) + 1.0f;
/* DPCT_ORIG     B[threadIdx.x] = threadIdx.x + 1.0f;*/
    B[item_ct1.get_local_id(2)] = item_ct1.get_local_id(2) + 1.0f;
/* DPCT_ORIG     C[threadIdx.x] = A[threadIdx.x] + B[threadIdx.x];*/
    C[item_ct1.get_local_id(2)] =
        A[item_ct1.get_local_id(2)] + B[item_ct1.get_local_id(2)];
}  ...
  /* DPCT_ORIG     cudaMalloc(&d_A, VECTOR_SIZE*sizeof(float));*/
      d_A = sycl::malloc_device<float>(VECTOR_SIZE, q_ct1);
  /* DPCT_ORIG     cudaMalloc(&d_B, VECTOR_SIZE*sizeof(float));*/
      d_B = sycl::malloc_device<float>(VECTOR_SIZE, q_ct1);
  /* DPCT_ORIG     cudaMalloc(&d_C, VECTOR_SIZE*sizeof(float));*/
      d_C = sycl::malloc_device<float>(VECTOR_SIZE, q_ct1);

  /* DPCT_ORIG     VectorAddKernel<<<1, VECTOR_SIZE>>>(d_A, d_B, d_C);*/
      q_ct1.submit([&](sycl::handler &cgh) {
```

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

warning: DPCT1043:1: The version-related API is different in SYCL. An initial code was generated, but you need to adjust it.

warning: DPCT1009:2: 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.

…

warning: 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.
```

tinyurl.com/dpct-project-migration-linux

Rodinia NW

- *cp Makefile migration/*
- Replace the CUDA configurations in that new `Makefile` with the following for use with DPC++:

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

- Compilation out-of-box fails with an error similar to the following:

```
error: assigning to 'int' from incompatible type 'typename info::param_traits<info::device,
(device)4143U>::return_type' (aka 'basic_string<char>’)
```

- Need to address warnings first

Addressing Warnings in Migrated Code

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

warning: DPCT1043:1: The version-related API is different in SYCL. An initial code was generated, but you need to adjust it.

warning: DPCT1009:2: 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;
int err_code = 999;
/* ...dpct generated comments... */
err_code = (version = dpct::get_current_device().get_info<sycl::info::device::version>(), 0);
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*/);
```

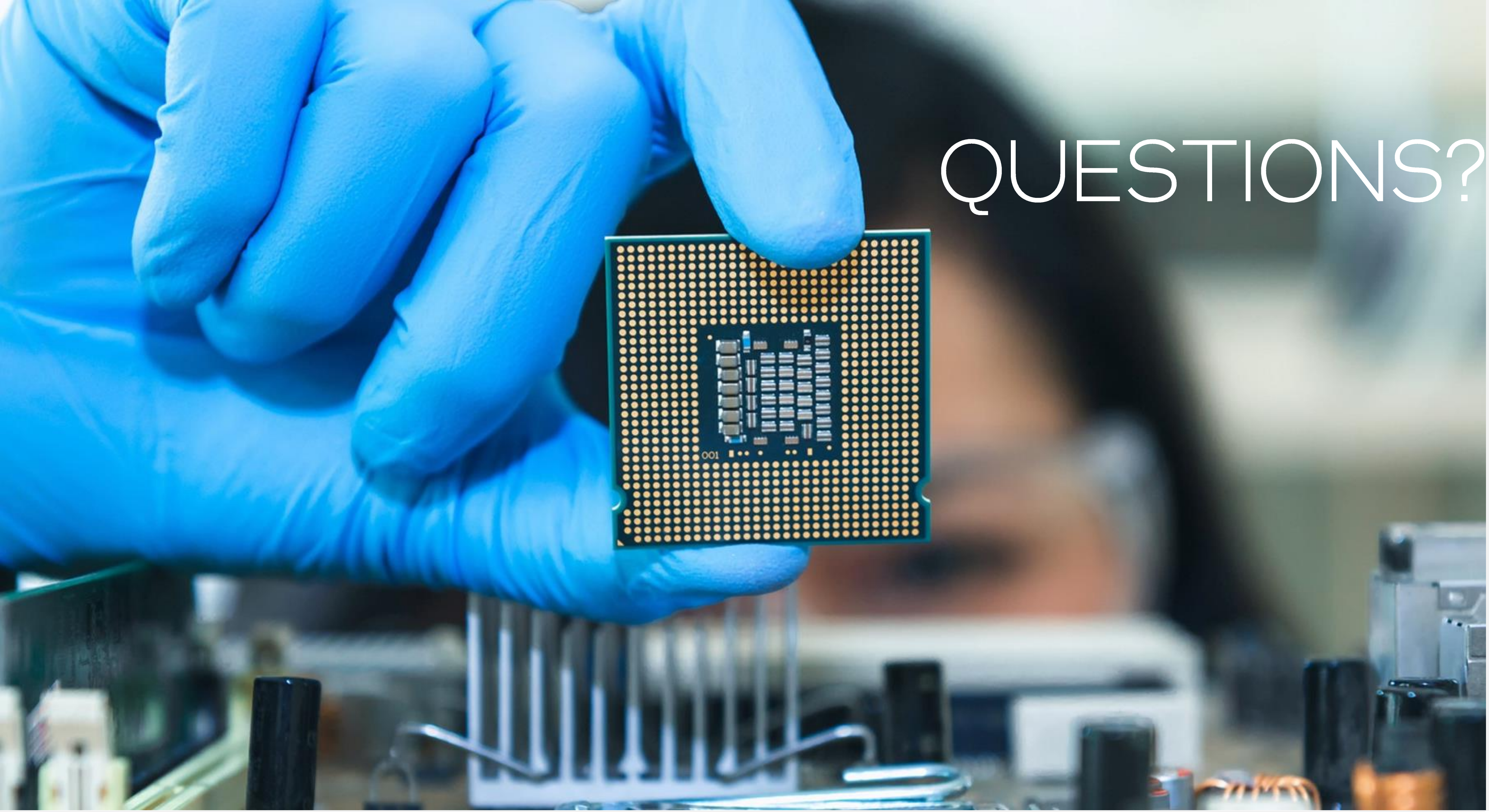
Once migration is completed, compile DPC++ code and run via make commands:

- *make*
- *make run*
- *./needleman_wunsch_dpcpp 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®