Benchmarking the Performance of oneAPI on Heterogeneous Computing Platforms

oneAPI – 가속 컴퓨팅을 개발하기 위한 스마트한 방식

2021. 6. 18.
MOASYS

intel software

moasys

Contents

Heterogeneous Computing: Road to Exascales

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
5	**Selene** - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	555,520	63,460.0	79,215.0	2,646

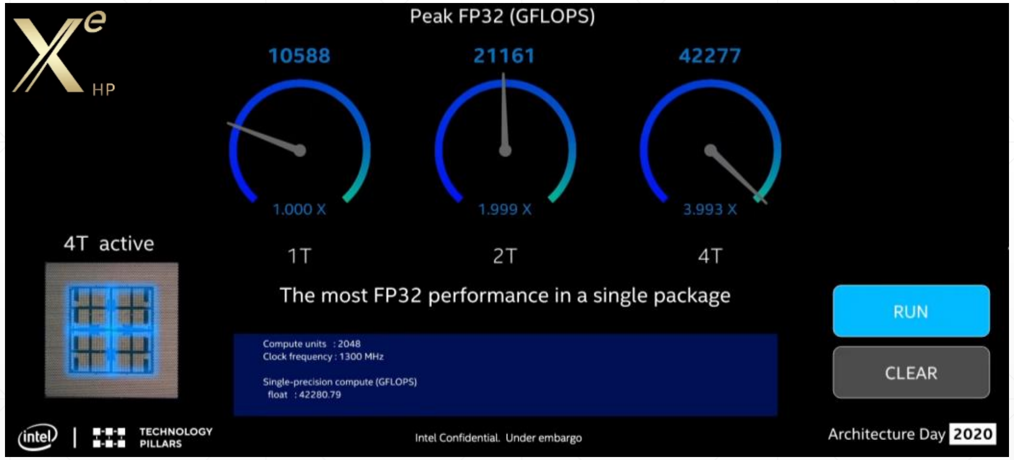
https://www.top500.org/lists/top500/2020/11/

2022- Xeon Scalable + XE "Ponte Vecchio" GPU (1 ExaFLOPS)

- Currently 6/10 supercomputers at Top 10 are heterogeneous platforms based on NVIDIA GPUs.
  - Aurora will be the one of the first *exascale* supercomputers in America powered by Intel's *Ponte Vecchio* architecture.
  - Intel® OneAPI allows researchers to harness the power of Aurora for data analytics, AI research, HPC applications.

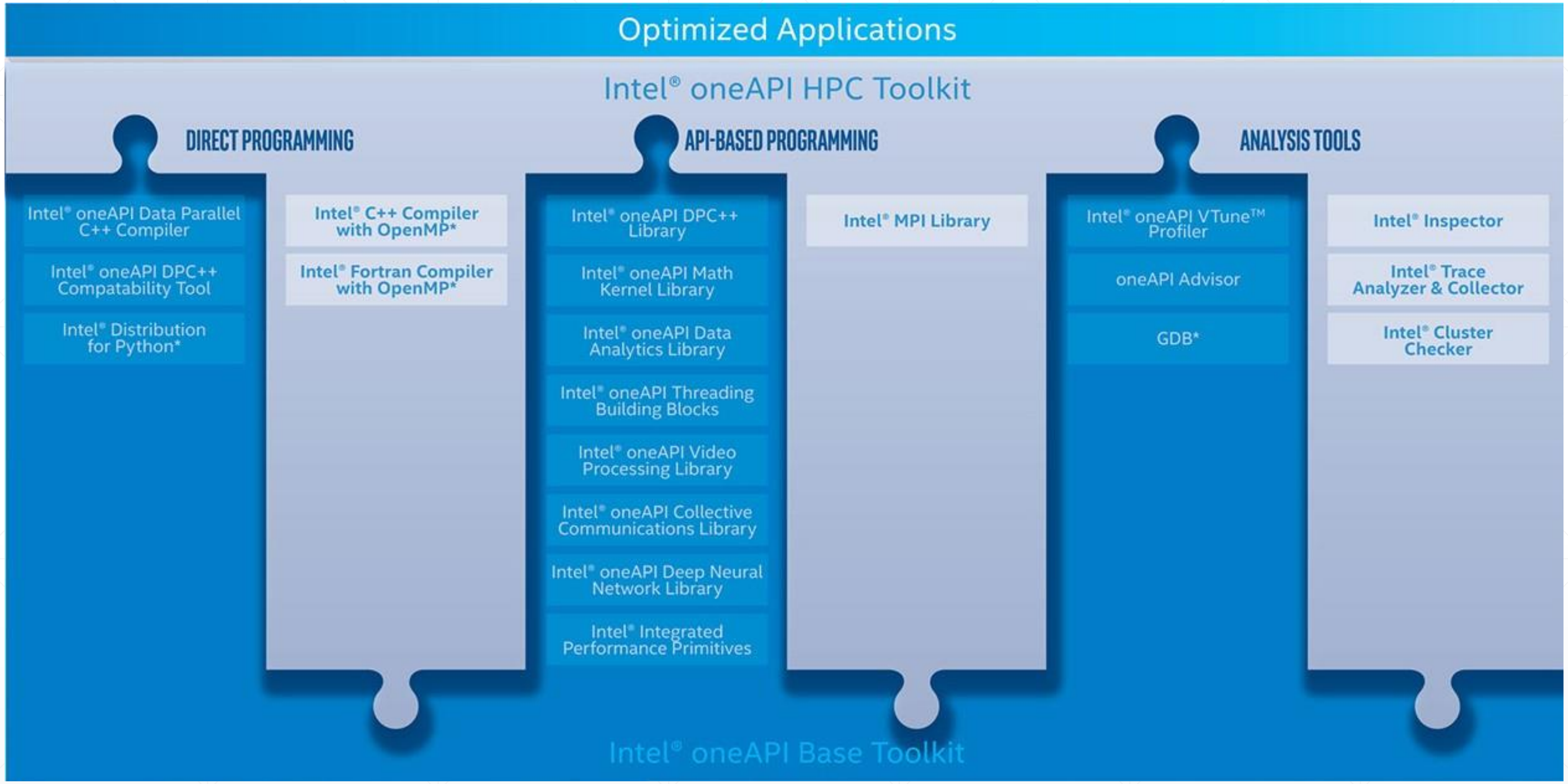
Intel Xe Architecture: Building the Foundation for Exascale Computing

µArchitecture		Packaging		Process
Xe HPC	PONTE VECCHIO	FOVEROS CO-EMIB	BASE TILE	Intel 10nm SuperFin
			COMPUTE TILE	Intel Next Gen & External
			RAMBO CACHE TILE	Intel 10nm Enhanced SuperFin
			Xe LINK I/O TILE	External
Xe HP	TBA	EMIB		Intel 10nm Enhanced SuperFin
Xe HPG	TBA	STANDARD		External
Xe LP	SG1 DG1 TIGER LAKE	STANDARD		Intel 10nm SuperFin



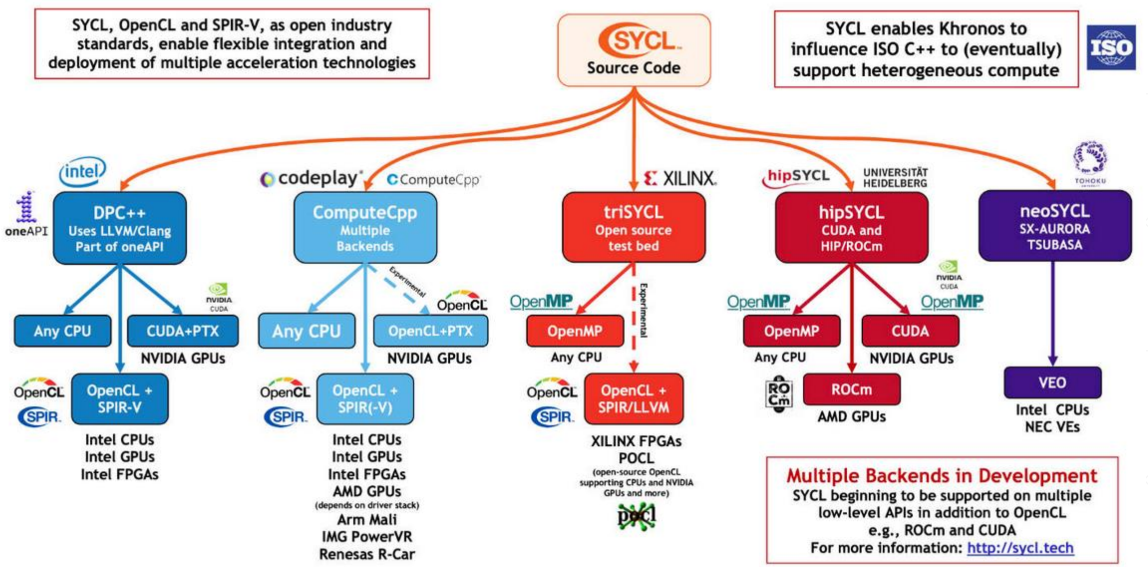
- Intel architecture day 2020:
  - https://newsroom.intel.com/wp-content/uploads/sites/11/2020/08/Intel-Architecture-Day-2020-Presentation-Slides.pdf
  - Xe-HP can scale up to 4 tiles with a peak FP32 performance of 42 Tflops

oneAPI: Driving a New Era of Accelerated Computing

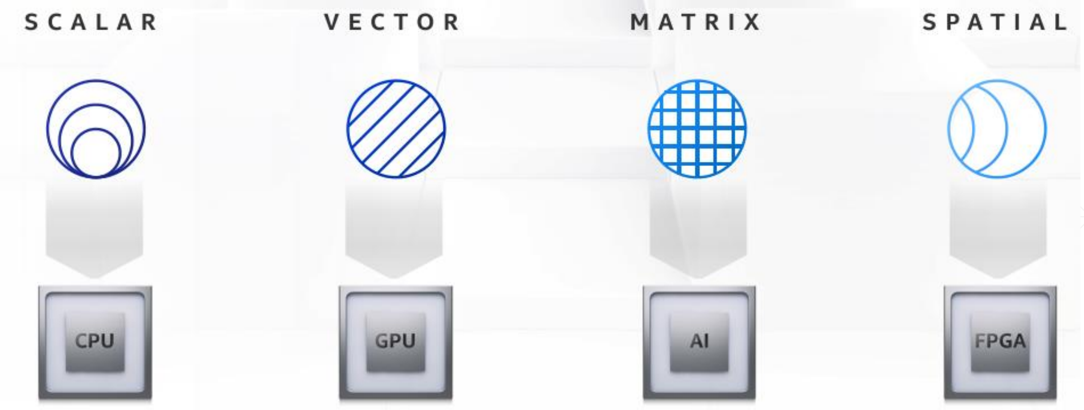


- Future-ready programming model provides freedom of choice
- Top performance for accelerated architectures
- Fast and efficient development
- Easy integration with legacy code

DPC++: A Unified Programming Model for Heterogeneous Computing



https://www.khronos.org/sycl/



https://software.intel.com/content/www/us/en/develop/tools/oneapi.html

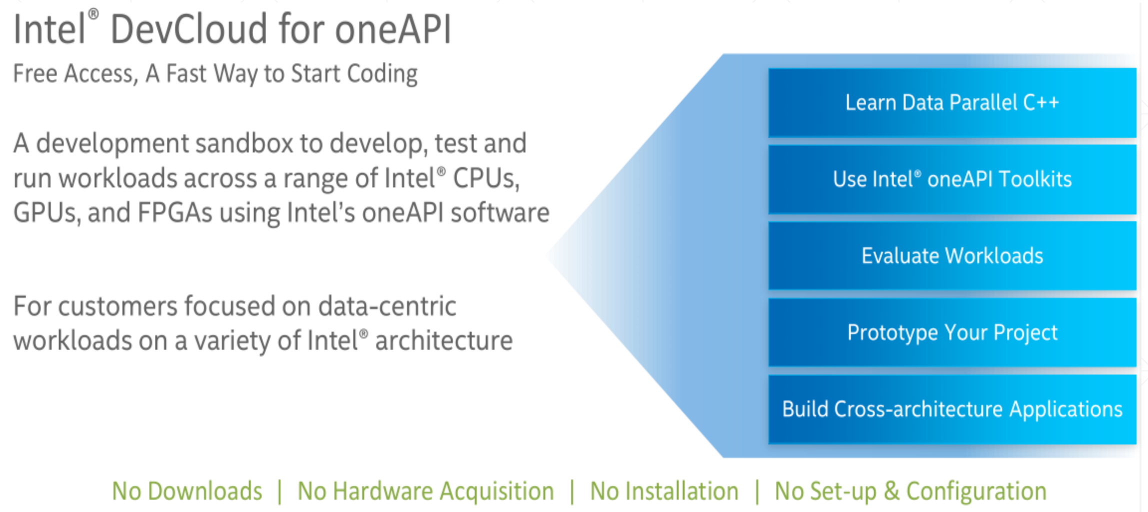
- DPC++ is an implementation of SYCL standards by Intel
- Intel has made many important contributions to expand the SYCL standards:
  - Unified Shared Memory (USM): now part of 2020 SYCL
  - Sub-groups and work-group collectives
  - Explicit SIMD
- **Does DPC++ support NVIDIA devices and CUDA performance libraries such as cuBLAS/cuRAND ?**
  - Codeplay's contribution to intel-llvm: https://devmesh.intel.com/projects/dpc-for-cuda
  - Codeplay's contribution to oneMKL interface: https://github.com/oneapi-src/oneMKL

Motivations

- Intel® DPC++:
  - An ambition solution to vendor-lock issues
  - Support for diverse class of accelerator devices: CPUs, GPUs, FPGAs and AI processors
  - *Write once run everywhere*
- Migration to DPC++:
  - Is the cost of migration justifiable ?
  - Is portability or performance more important ?
  - Does the high level abstraction of DPC++ have a negative impact on performance in comparison to native implementation ?
- A systematic evaluation of DPC++ and oneAPI for heterogeneous computing
  - Measurement of memory bandwidth utilization using the standard STREAM benchmark
  - Migration of well-established algorithms written in CUDA to DPC++ using Intel DPC++ Compatibility Tool (DPCT)
  - Performance comparison between migrated DPC++ codes and native implementations CUDA
  - Demonstration of oneMKL's interoperability with native performance library such as cuBLAS
  - Introduction to oneDNN execution model

Benchmark Methodology: Hardwares

- Intel DevCloud
  - Intel® Xeon® E-2176 / **UHD P630**
  - Intel® Core® i9-10920X / **Iris Xe Max**
- Software stacks
  - oneAPI 2021.2
- https://devcloud.intel.com/oneapi/



-
- Benchmark platforms:
  - Intel® Xeon® E5-2670 / **NVIDIA Tesla K40**
  - Intel® Xeon® Gold 6230 / **NVIDIA Tesla V100**
- Software stacks:
  - Intel LLVM compiler with CUDA backend
  - oneMKL interface
  - oneDNN

	Intel Core i9	UHD P630	Iris Xe Max	Tesla K40	Tesla V100
Clock	3.50 GHz	1100 MHz	1650 MHz	875 MHz	1530 MHz
Core/SS/SM*	12 (x2 HT)	3	12	15	80
Memory BW	94 GB/s	41.6 GB/s	68.26 GB/s	288 GB/s	900 GB/s
Single Prec	2.68 TF	422.4 GF	2.53 TF	5.04 TF	15,7 TF
Double Prec	1.34 TF	105.6 GF	n/a	1.68 TF	7.8 FT

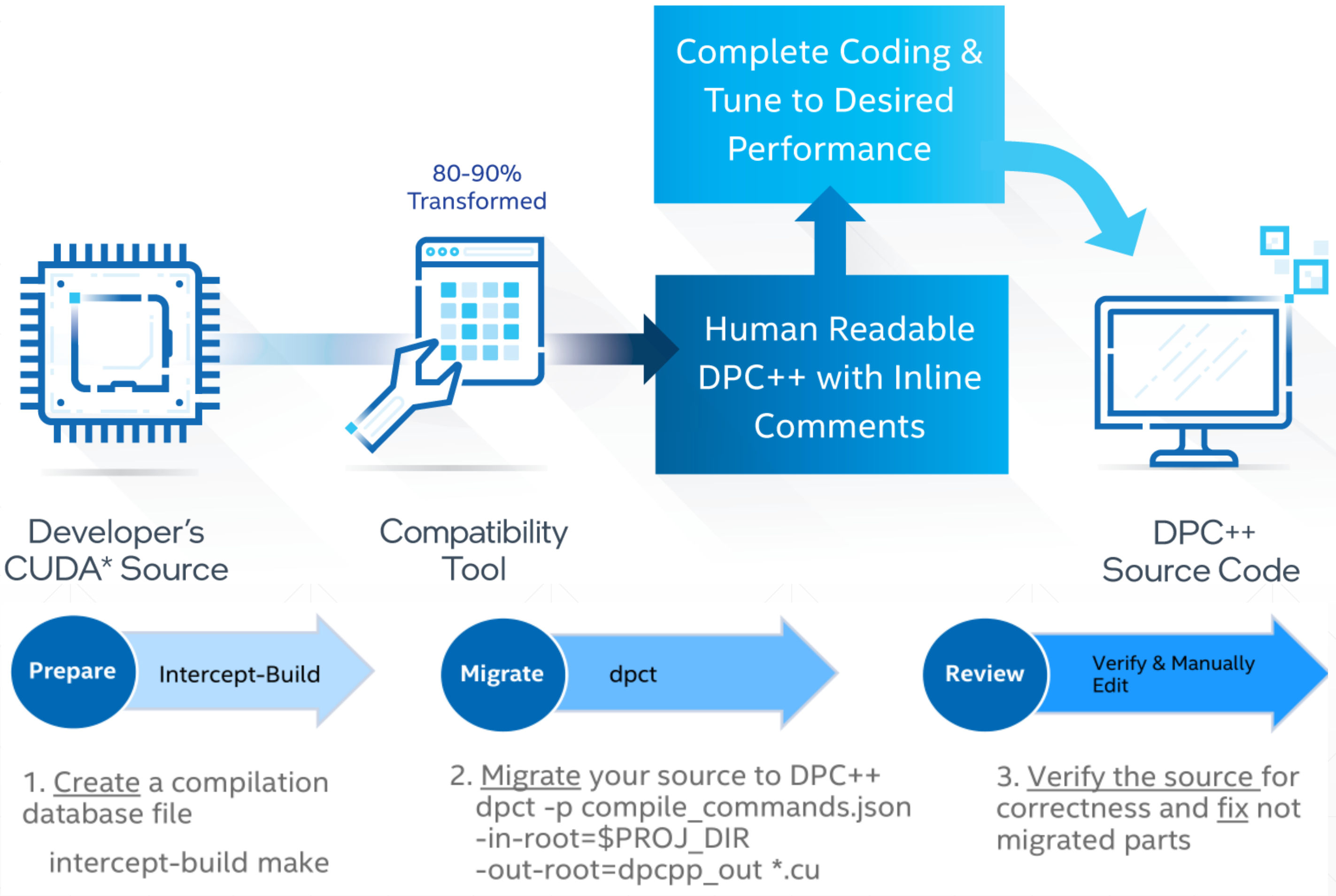
*SS: Sub Slice / SM: Streaming Multiprocessor

Xe Architecture for Machine Learning and AI

```
Device Name                                  Intel(R) Iris(R) Xe MAX Graphics [0x4905]
Max compute units                            96
Max clock frequency                          1650MHz
Preferred / native vector sizes
    char                                     16 / 16
    short                                     8 / 8
    int                                       4 / 4
    long                                      1 / 1
    half                                      8 / 8     (cl_khr_fp16)
    float                                     1 / 1
    double                                    1 / 1     (n/a)

Half-precision Floating-point support        (cl_khr_fp16)
    Denormals                                 Yes
    Infinity and NANs                         Yes
    Round to nearest                          Yes
    Round to zero                             Yes
    Round to infinity                         Yes
    IEEE754-2008 fused multiply-add           Yes
    Support is emulated in software           No

Single-precision Floating-point support      (core)
    Denormals                                 Yes
    Infinity and NANs                         Yes
    Round to nearest                          Yes
    Round to zero                             Yes
    Round to infinity                         Yes
    IEEE754-2008 fused multiply-add           Yes
    Support is emulated in software           No
    Correctly-rounded divide and sqrt operations  No

Double-precision Floating-point support      (n/a)
```

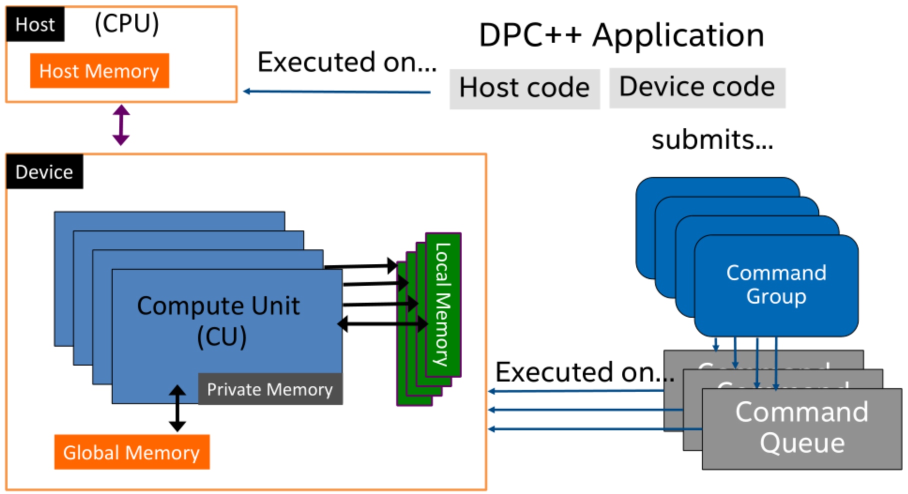
- Native supports for half and single precision
- Double precision available through software emulator
- Support for INT8 (16-bit) and INT4 (8-bit) integers for machine learning inference

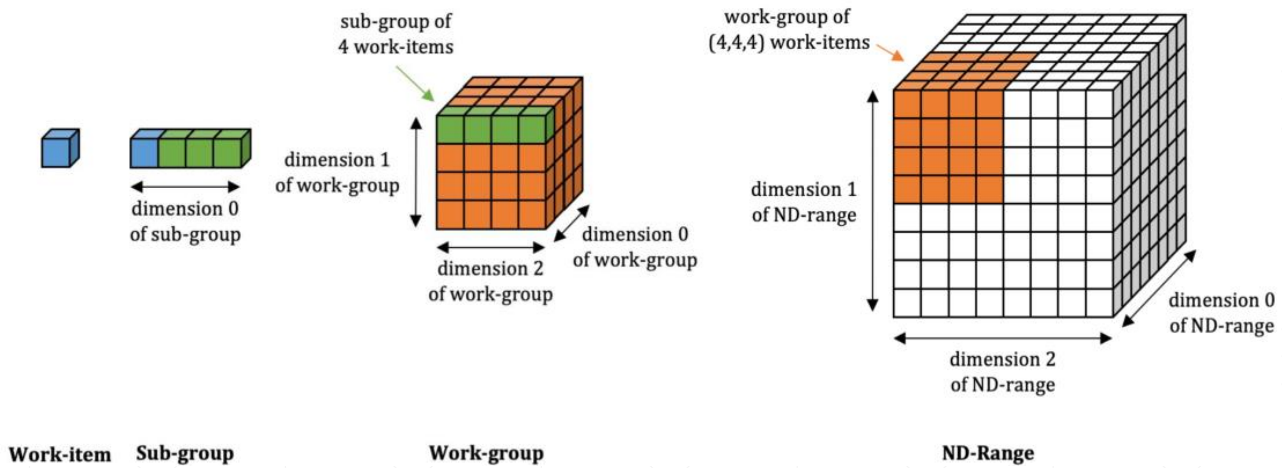
Intel® DPC++ Compatibility Tool



- Support for migrating CUDA codes to DPC++
  - Automatic migration of 80 ~ 90% of original source codes
  - Useful inline comments for troubleshooting and manually porting any not yet supported CUDA functions.

DPC++ Execution Model and Thread Hierarchy





- *parallel_for<kernel_name>(range<1>{size}, [=](id<1> idx) {*
  - **range<>** class: iteration space for parallel execution
  - **id<>** class: index of an individual instance of kernel
  - Work-group size is automatically determined by OpenCL at runtime
- *parallel_for<kernel_name>(nd_range<3>{global_size, local_size})[=](nd_item<3> item) {*
  - **nd_range<>** class: grouped execution range including global and local execution range of each work-group
    - **global_size**: dimensions of the entire index space
    - **local_size**: dimensions of the work group
  - **nd_item<>** class: represent an individual instance of a kernel functions
  - Work-items within same work-group are schedule on one Compute Unit (CU)
  - Work-items within same sub-group are mapped to vector hardware with private local memory

Max Work Group Size: Intel vs. NVIDIA

```
Found Platform:
    info::platform::name is 'NVIDIA CUDA BACKEND'
    info::platform::vendor is 'NVIDIA Corporation'
    info::platform::version is 'CUDA 11.2'
    info::platform::profile is 'FULL PROFILE'
  Device: Tesla K40
    is_host(): No
    is_cpu(): No
    is_gpu(): Yes
    is_accelerator(): No
    info::device::vendor is 'NVIDIA Corporation'
    info::device::driver_version is 'CUDA 11.2'
    info::device::max_work_item_dimensions is '3'
    info::device::max_work_group_size is '1024'
    info::device::mem_base_addr_align is '4096'
    info::device::partition_max_sub_devices is '0'
```

```
Found Platform:
    info::platform::name is 'Intel(R) OpenCL HD Graphics'
    info::platform::vendor is 'Intel(R) Corporation'
    info::platform::version is 'OpenCL 3.0 '
    info::platform::profile is 'FULL_PROFILE'
  Device: Intel(R) UHD Graphics P630 [0x3e96]
    is_host(): No
    is_cpu(): No
    is_gpu(): Yes
    is_accelerator(): No
    info::device::vendor is 'Intel(R) Corporation'
    info::device::driver_version is '21.11.19310'
    info::device::max_work_item_dimensions is '3'
    info::device::max_work_group_size is '256'
    info::device::mem_base_addr_align is '1024'
    info::device::partition_max_sub_devices is '0'
```

```
Found Platform:
    info::platform::name is 'Intel(R) OpenCL'
    info::platform::vendor is 'Intel(R) Corporation'
    info::platform::version is 'OpenCL 2.1 LINUX'
    info::platform::profile is 'FULL_PROFILE'
  Device: Intel(R) Core(TM) i9-10920X CPU @ 3.50GHz
    is_host(): No
    is_cpu(): Yes
    is_gpu(): No
    is_accelerator(): No
    info::device::vendor is 'Intel(R) Corporation'
    info::device::driver_version is '2021.11.3.0.17_160000'
    info::device::max_work_item_dimensions is '3'
    info::device::max_work_group_size is '8192'
    info::device::mem_base_addr_align is '1024'
    info::device::partition_max_sub_devices is '24'
```

```
Found Platform:
    info::platform::name is 'NVIDIA CUDA BACKEND'
    info::platform::vendor is 'NVIDIA Corporation'
    info::platform::version is 'CUDA 11.2'
    info::platform::profile is 'FULL PROFILE'
  Device: Tesla V100
    is_host(): No
    is_cpu(): No
    is_gpu(): Yes
    is_accelerator(): No
    info::device::vendor is 'NVIDIA Corporation'
    info::device::driver_version is 'CUDA 11.2'
    info::device::max_work_item_dimensions is '3'
    info::device::max_work_group_size is '1024'
    info::device::mem_base_addr_align is '4096'
    info::device::partition_max_sub_devices is '0'
```

```
Found Platform:
    info::platform::name is 'Intel(R) OpenCL HD Graphics'
    info::platform::vendor is 'Intel(R) Corporation'
    info::platform::version is 'OpenCL 3.0 '
    info::platform::profile is 'FULL_PROFILE'
  Device: Intel(R) Iris(R) Xe MAX Graphics [0x4905]
    is_host(): No
    is_cpu(): No
    is_gpu(): Yes
    is_accelerator(): No
    info::device::vendor is 'Intel(R) Corporation'
    info::device::driver_version is '21.11.19310'
    info::device::max_work_item_dimensions is '3'
    info::device::max_work_group_size is '512'
    info::device::mem_base_addr_align is '1024'
    info::device::partition_max_sub_devices is '0'
```

```
Found Platform:
    info::platform::name is 'SYCL host platform'
    info::platform::vendor is ''
    info::platform::version is '1.2'
    info::platform::profile is 'FULL PROFILE'
  Device: SYCL host device
    is_host(): Yes
    is_cpu(): No
    is_gpu(): No
    is_accelerator(): No
    info::device::vendor is ''
    info::device::driver_version is '1.2'
    info::device::max_work_item_dimensions is '3'
    info::device::max_work_group_size is '1'
    info::device::mem_base_addr_align is '1024'
    info::device::partition_max_sub_devices is '1'
```

- For NVIDIA devices, max_work_group_size is fixed at 1024 (maximum number of thread per block)
- For Intel devices, max_work_group_size varies from device to device: Iris Xe Max (512) vs. Gen9 (256)
- The host device is emulation of an OpenCL device with no thread support

STREAM Benchmark: Heterogeneous Programming Approaches

```
template <class T>
void OMPStream<T>::triad()
{
  const T scalar = startScalar;
#ifdef OMP_TARGET_GPU
  int array_size = this->array_size;
  T *a = this->a;
  T *b = this->b;
  T *c = this->c;
  #pragma omp target teams distribute parallel for simd
#else
  #pragma omp parallel for
#endif
  for (int i = 0; i < array_size; i++)
  {
    a[i] = b[i] + scalar * c[i];
  }
}
```

OpenMP

```
template <typename T>
__global__ void triad_kernel(T * a, const T * b, const T * c)
{
   const T scalar = startScalar;

   const int i = blockDim.x * blockIdx.x + threadIdx.x;
   a[i] = b[i] + scalar * c[i];
}

template <class T>
void CUDAStream<T>::triad()
{
  triad_kernel<<<array_size/TBSIZE, TBSIZE>>>(d_a, d_b, d_c);
  check_error();
  cudaDeviceSynchronize();
  check_error();
}
```

NVIDIA CUDA

```
kernel void triad(
   global TYPE * restrict a,
   global const TYPE * restrict b,
   global const TYPE * restrict c)
 {
   const size_t i = get_global_id(0);
   a[i] = b[i] + scalar * c[i];
 }


template <class T>
void OCLStream<T>::triad()
{
  (*triad_kernel)(
    cl::EnqueueArgs(queue, cl::NDRange(array_size)), d_a, d_b, d_c
  );
  queue.finish();
}
```

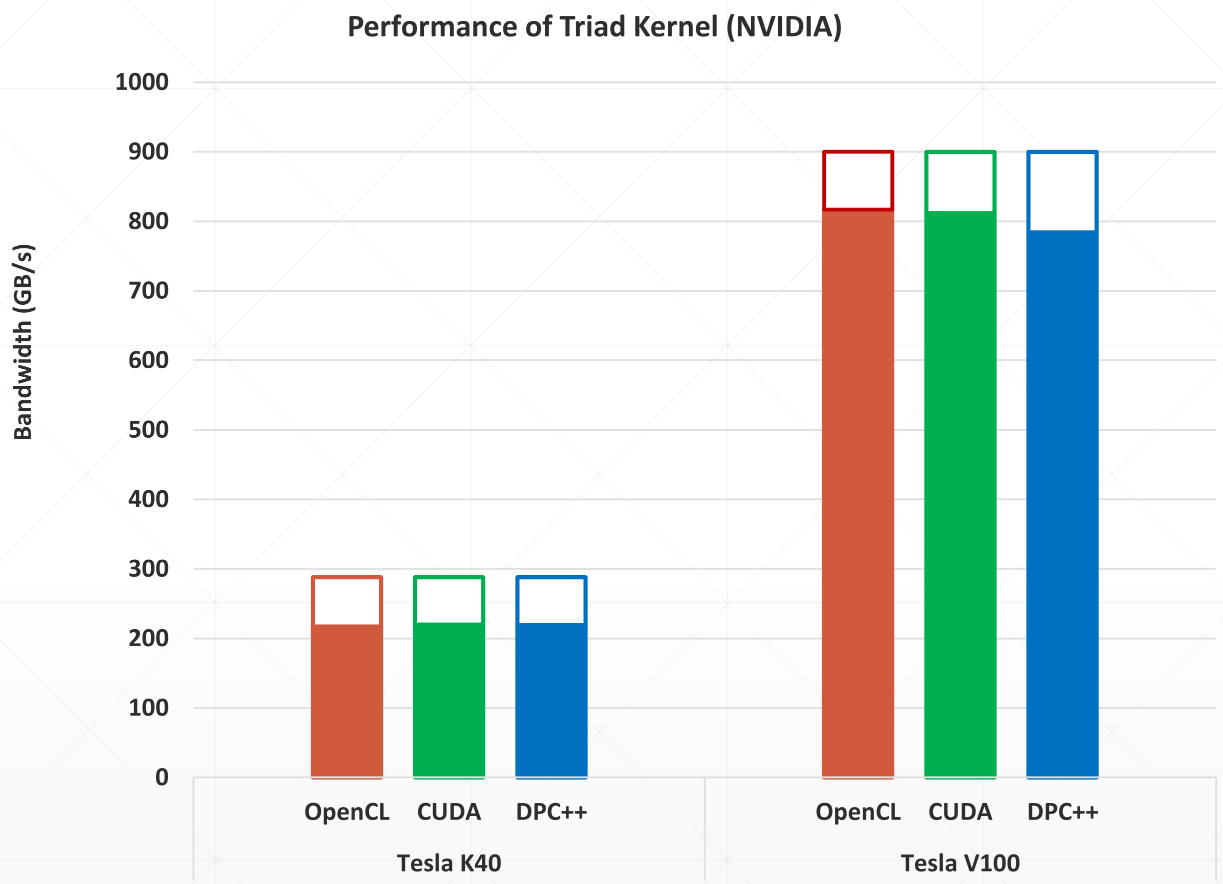
OpenCL

```
template <class T>
void SYCLStream<T>::triad()
{
  const T scalar = startScalar;

  queue->submit([&](handler &cgh)
  {
    auto ka = d_a->template get_access<access::mode::write>(cgh);
    auto kb = d_b->template get_access<access::mode::read>(cgh);
    auto kc = d_c->template get_access<access::mode::read>(cgh);
    cgh.parallel_for<triad_kernel>(range<1>{array_size}, [=](id<1> idx)
    {
       ka[idx] = kb[idx] + scalar * kc[idx];
    });
});
queue->wait();
}
```

oneAPI

- Triad kernel measures memory bandwidth associated with computation on 1d-vectors: $a = b + \gamma c$

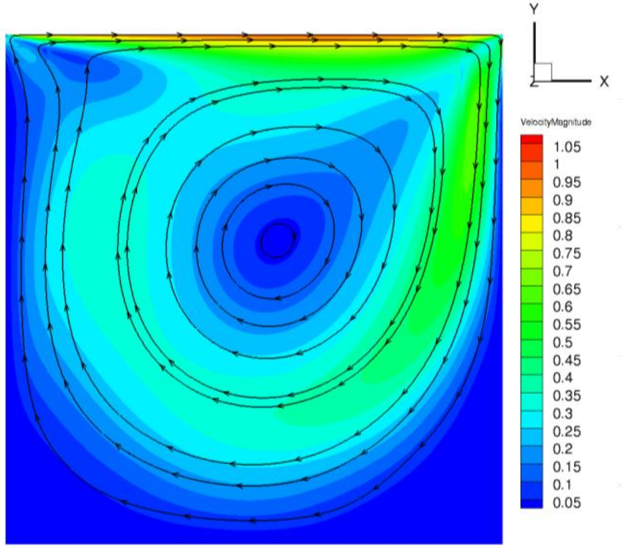
STREAM Benchmark

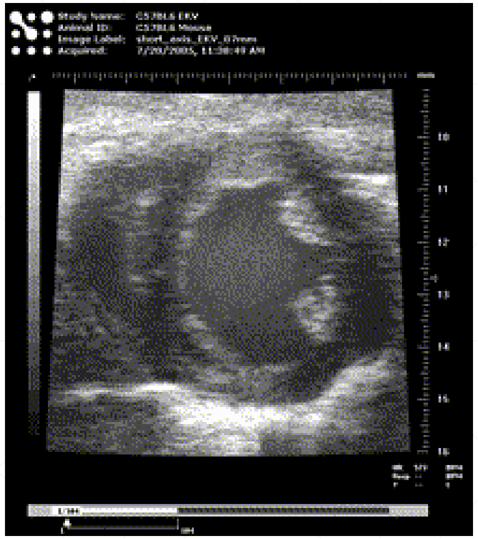


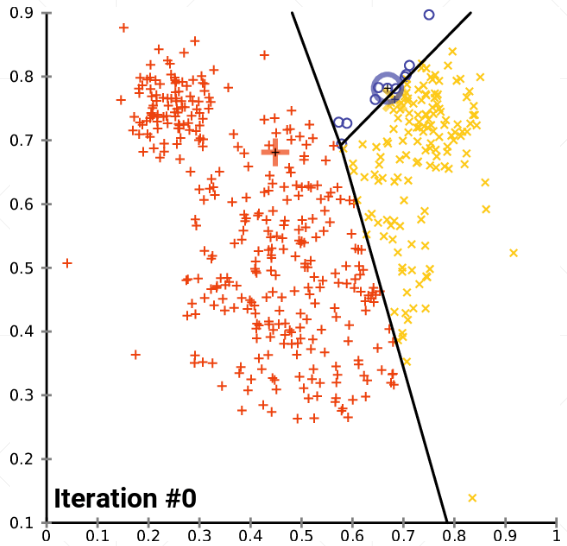
	V100 PCIe	V100 SXM2	V100S PCIe
GPU Architecture	NVIDIA Volta		
NVIDIA Tensor Cores	640		
NVIDIA CUDA® Cores	5,120		
Double-Precision Performance	7 TFLOPS	7.8 TFLOPS	8.2 TFLOPS
Single-Precision Performance	14 TFLOPS	15.7 TFLOPS	16.4 TFLOPS
Tensor Performance	112 TFLOPS	125 TFLOPS	130 TFLOPS
GPU Memory	32 GB /16 GB HBM2		32 GB HBM2
Memory Bandwidth	900 GB/sec		1134 GB/sec
ECC	Yes		
Interconnect Bandwidth	32 GB/sec	300 GB/sec	32 GB/sec
System Interface	PCIe Gen3	NVIDIA NVLink™	PCIe Gen3
Form Factor	PCIe Full Height/Length	SXM2	PCIe Full Height/Length
Max Power Comsumption	250 W	300 W	250 W
Thermal Solution	Passive		
Compute APIs	CUDA, DirectCompute, OpenCL™, OpenACC®		

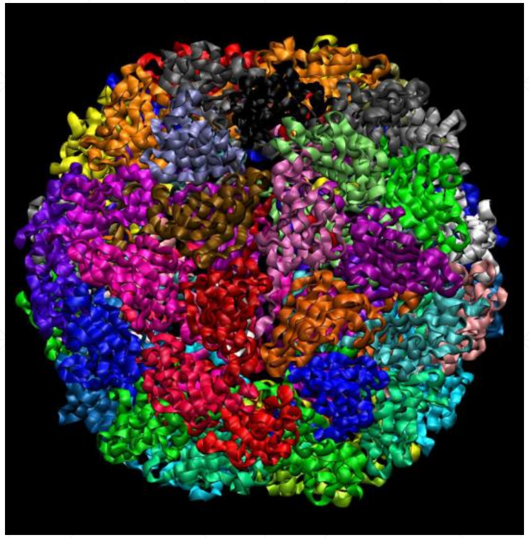
- DPC++ provides consistent performance in comparison with CUDA and OpenCL implementations.
- DPC++ archives 85% of theoretical bandwidth on both K40 and V100 devices.
- OpenCL/CUDA implementation: https://github.com/UoB-HPC/BabelStream

Representative Algorithms in Computation Sciences



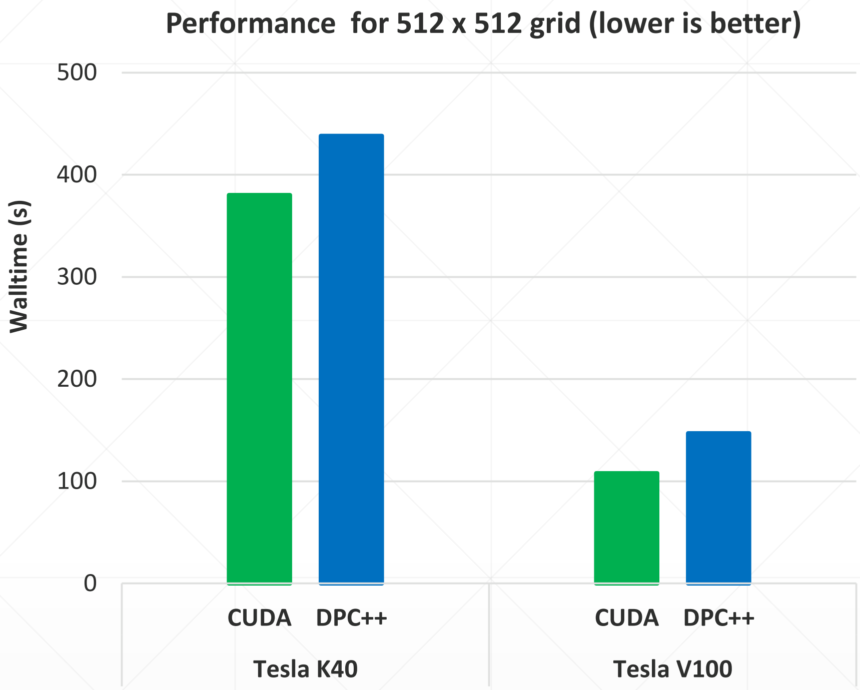


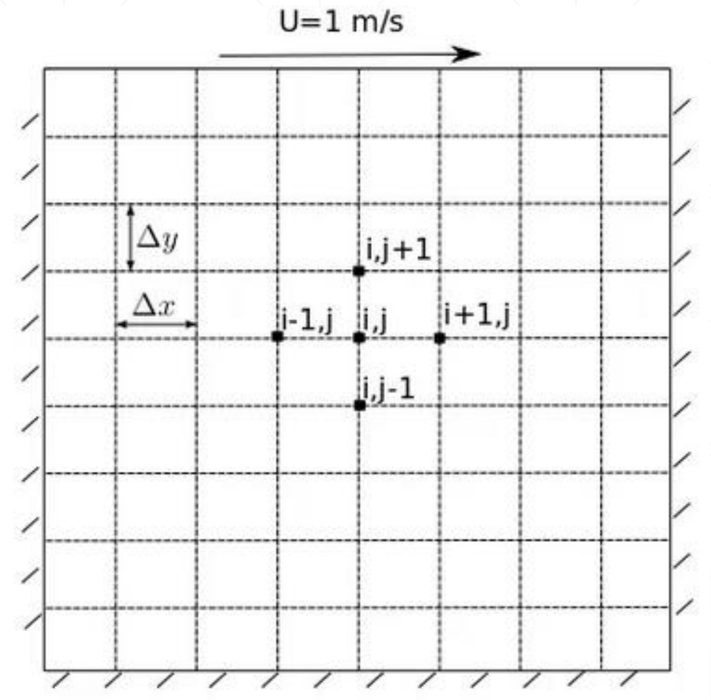




- Lid-driven cavity flow:
  - Standard benchmark problem in CFD: solving Nervier-Stokes equation using finite different method.
- Heart wall tracking:
  - Biomedical image processing: tracking the motion of the heart wall obtained form ultra sound
- K-mean clustering:
  - Important algorithm in unsupervised machine learnings: partition data points into cluster based on their features.
- GROMACS:
  - One of the most widely used codes in bio molecular simulations

Lid-Driven Cavity: Benchmarks on Heterogeneous Platforms





- Input set up:
  - 512 x 512 grid for finite different
- Algorithm description:
  - The kernel solves Navier-Stokes equation using finite difference on discrete lattice.
- DPC++/CUDA performs within 10 ~ 20% w.r.t native CUDA implementation
- CUDA implementation: https://github.com/kyleniemeyer/lid-driven-cavity_gpu

Lid-driven Cavity: Migration Result with DPCT

```
// block and grid dimensions for F
dim3 block_F (BLOCK_SIZE, 1);
dim3 grid_F (NUM / BLOCK_SIZE, NUM);
...
void calculate_F (const Real dt, const Real* u, const Real* v, Real* F)
{
  int row = (blockIdx.x * blockDim.x) + threadIdx.x + 1;

  int col = (blockIdx.y * blockDim.y) + threadIdx.y + 1;


  if (col == NUM) {
   // set boundary condition
  } else {
   // calculate components of F using finite difference
   }
}
...
```

```
#include <dpct/dpct.hpp>

// device and queue creation
dpct::device_ext &dev_ct1 = dpct::get_current_device();
sycl::queue        &q_ct1 = dev_ct1.default_queue();


// block and grid dimensions for F
sycl::range<3> block_F(1, 1, BLOCK_SIZE);
sycl::range<3> grid_F(1, NUM, NUM / BLOCK_SIZE);
...
void calculate_F (const Real dt, const Real* u, const Real* v, Real* F,
                  sycl::nd_item<3> item_ct1)
{
  int row = (item_ct1.get_group(2) * item_ct1.get_local_range().get(2)) +
            item_ct1.get_local_id(2) + 1;
  int col = (item_ct1.get_group(1) * item_ct1.get_local_range().get(1)) +
            item_ct1.get_local_id(1) + 1;

  if (col == NUM) {
     // set boundary condition
  } else {
     // calculate components of F using finite difference
  }
}
...
```

CUDA	DPC++	OpenCL
gridDim.{x, y, z}	nd_item::get_num_group({0,1,2})	get_num_group({0,1,2})
blockDim.{x, y, z}	nd_item::get_local_range({0,1,2})	get_local_size({0,1,2})
blockIdx.{x, y, z}	nd_item::get_group({0,1,2})	get_group_id({0,1,2})
threadIdx.{x, y, z}	nd_item::get_local_id({0,1,2})	get_local_id({0,1,2})

- DPCT always generates nd_range<3> kernels
  - Fastest index is the right most one
- For best portability across different devices:
  - `export SYCL_DEVICE_FILTER=<backend>:<type>`
  - `export SYCL_DEVICE_FILTER=opencl:gpu` **(Intel)**
  - `export SYCL_DEVICE_FILTER=cuda:gpu` **(NVIDIA)**

Lid-driven Cavity: Diagnostic Messages

```
main.cu:901:5: warning: DPCT1049:8:
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.
calculate_F <<<grid_F, block_F>>> (dt, u_d, v_d, F_d);
```

- Too large work group can leads to *illegal memory access* error
- For Intel devices, work group size varies UHD P630 (256), Iris Xe Max (512)

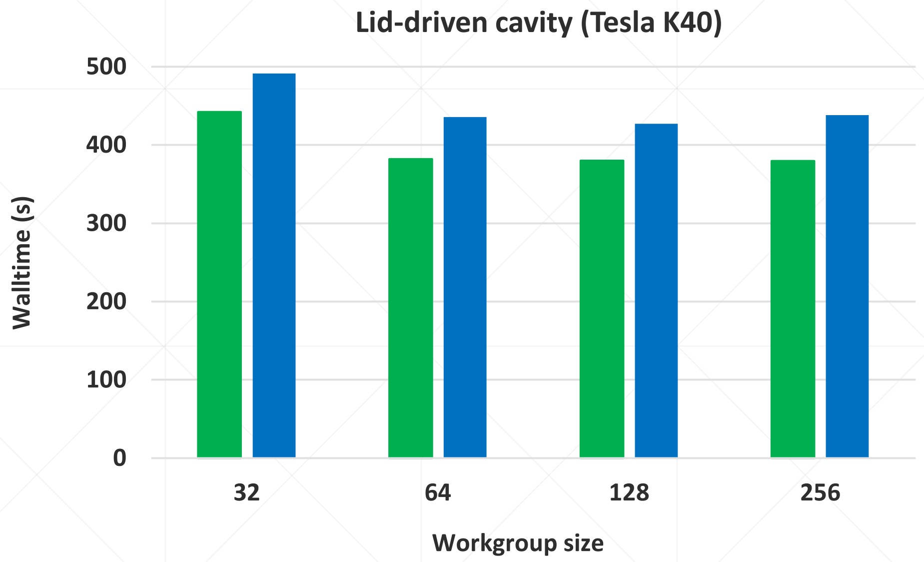
```
main.cu:714:5: warning: DPCT1065:7:
Consider replacing sycl::nd_item::barrier() with sycl::nd_item::barrier(sycl::access::fence_space::local_space)
for better performance, if there is no access to global memory.
__syncthreads();
```

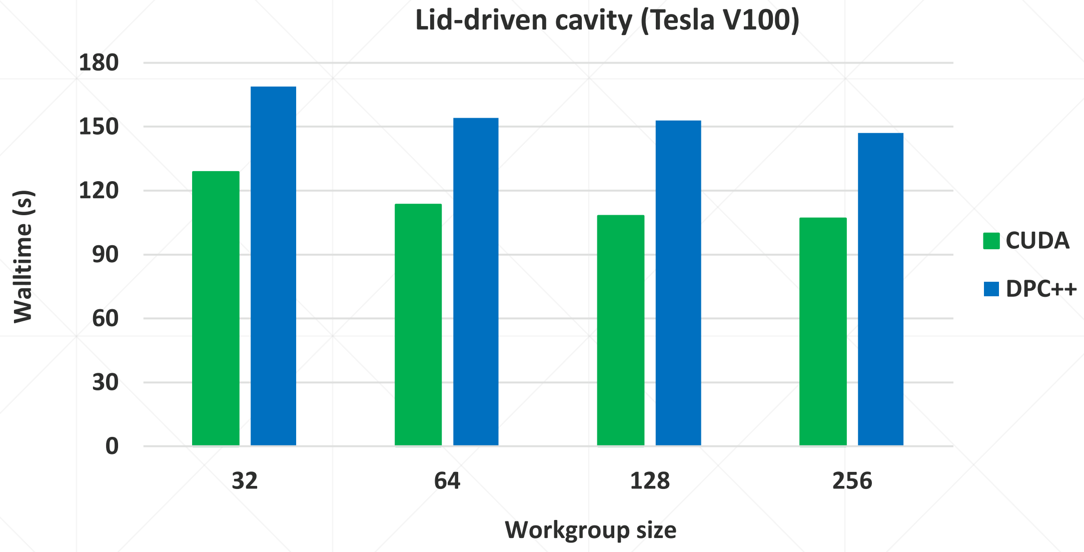
- Manual code review required to confirm memory access pattern

```
Main.cu: 1111:19 warning: DPCT1003:21:
Migrated API does not return error code. (*, 0) is inserted. You may need to rewrite this code.
checkCudaErrors (cudaDeviceReset());
```

- CUDA helpers function are not migrated and can be safely replaced with C++ exception handlers
- Device selection functions such as **cudaSetDevice()** has no equivalence in DPC++
- For portability and debug purpose:
  - `export SYCL_PI_TRACE=1`
  - `export SYCL_DEVICE_FILTER=opencl:gpu`

Lid-driven Cavity: Work Group Size Optimization



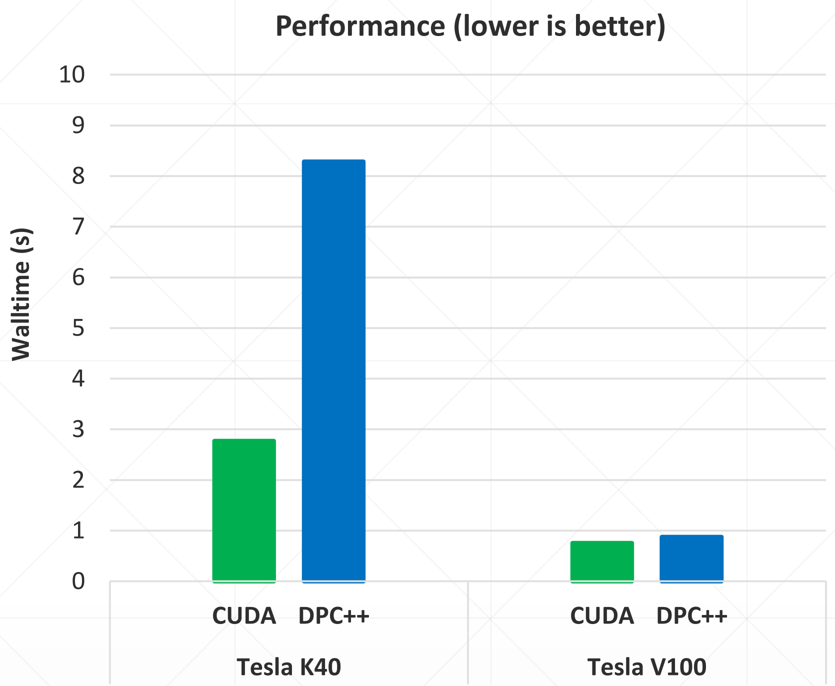


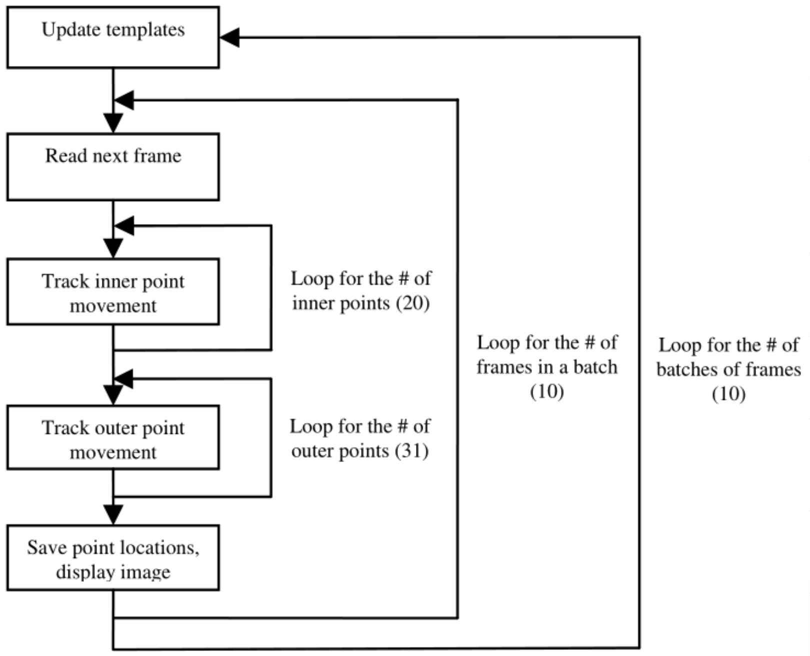

```
Device Name                                   Tesla K40
Device Vendor                                 NVIDIA Corporation
Device Topology (NV)                          PCI-E, 0000:86:00.0
Driver Version                                460.32.03
Max compute units                             15
Max clock frequency                           745Mhz
Compute Capability (NV)                       3.5
Device Partition                              (core)
    Max number of sub-devices                 1
    Supported partition types                 None
    Supported affinity domains                (n/a)
Max work item dimensions                      3
Max work item sizes                           1024x1024x64
Max work group size                           1024
Preferred work group size multiple (kernel)   32
Warp size (NV)                                32
```

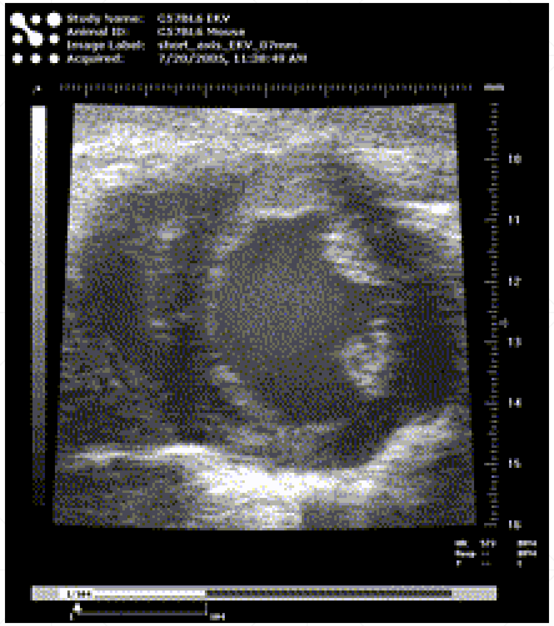
```
Device Name                                   Tesla V100-PCIE-32GB
Device Vendor                                 NVIDIA Corporation
Device Topology (NV)                          PCI-E, 0000:86:00.0
Driver Version                                460.32.03
Max compute units                             80
Max clock frequency                           1380MHz
Compute Capability (NV)                       7.0
Device Partition                              (core)
    Max number of sub-devices                 1
    Supported partition types                 None
    Supported affinity domains                (n/a)
Max work item dimensions                      3
Max work item sizes                           1024x1024x64
Max work group size                           1024
Preferred work group size multiple (kernel)   32
Warp size (NV)                                32
```

- Preferred work group size is multiple is 32 (CUDA warp)

Heartwall Tracking: Benchmarks on Heterogeneous Platforms

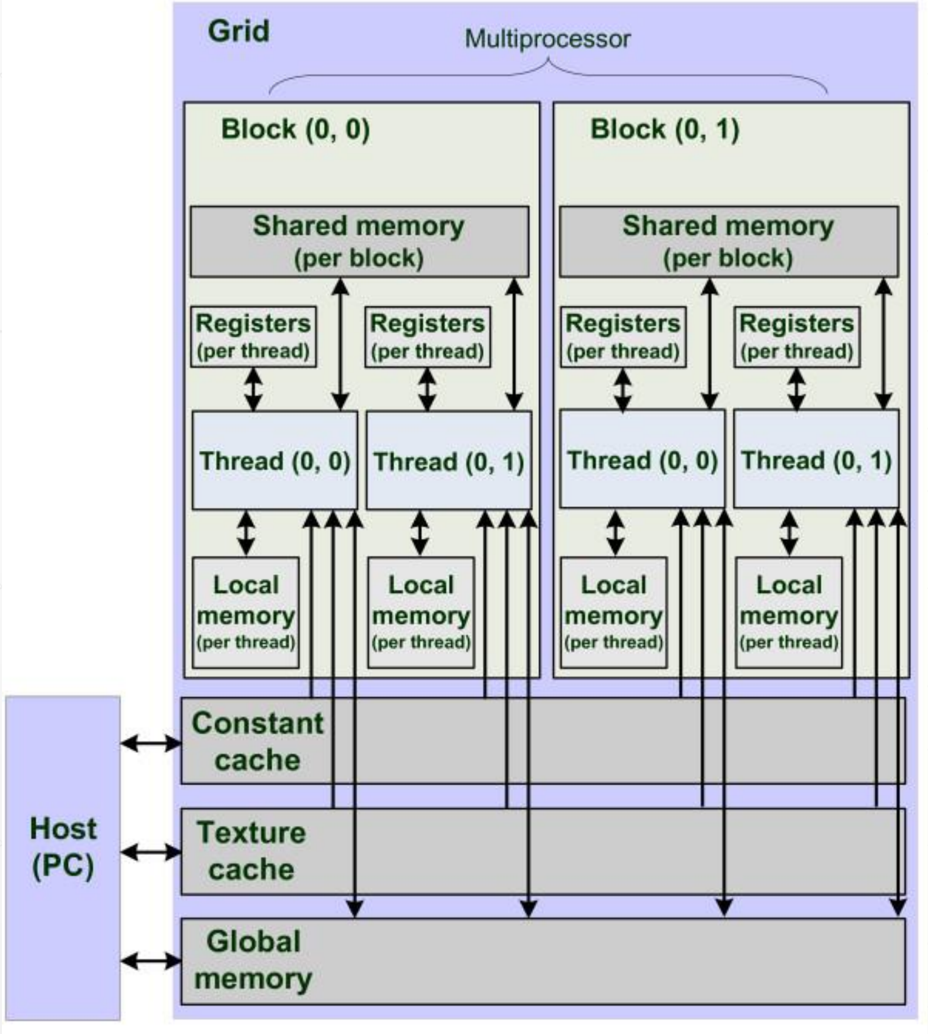






- Input setups:
  - Movement of mouse heart under stimulus: 104 frames, 656 x 744 pixels
- Description of algorithm:
  - Kernels performs images processing such as edge detection, noise reduction (SRAD), transformation and dilation (FP32)
  - Kernels performs Hough search to reconstruct shape of heart wall (FP32)
- DPC++/CUDA performs within 10% w.r.t native CUDA implementation for V100
- CUDA implementation: http://www.cs.virginia.edu/rodinia/doku.php?id=heart_wall

Heartwall Tracking: Migration CUDA Memory Model with DPCT



main.cu (Heartwall/CUDA)

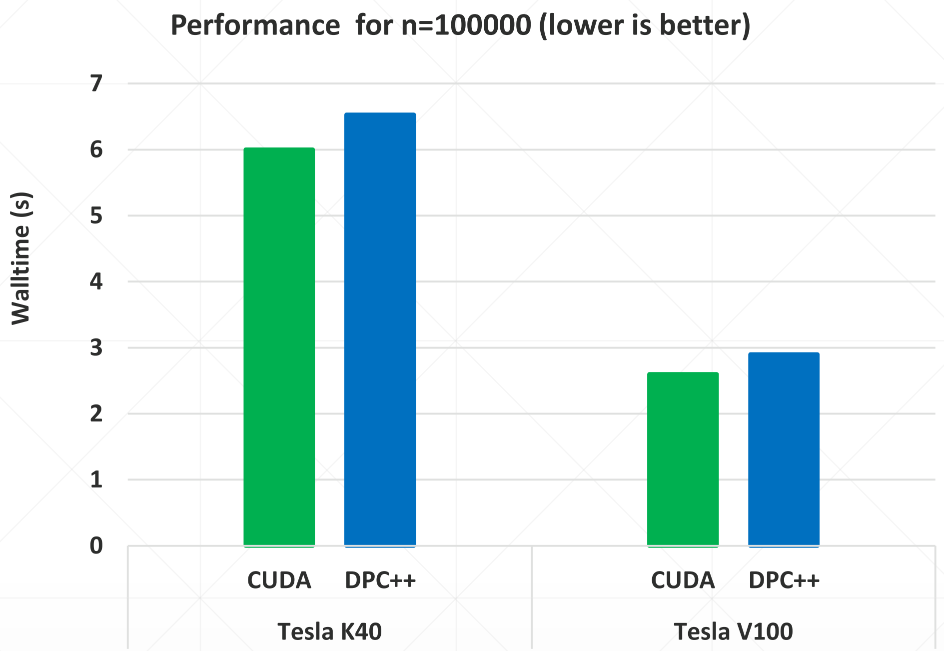
```
params_common_change common_change;
__constant__         params_common_change d_common_change;

params_common        common;
__constant__         params_common d_common;

params_unique        unique[ALL_POINTS];
__constant__         params_unique d_unique[ALL_POINTS];
```

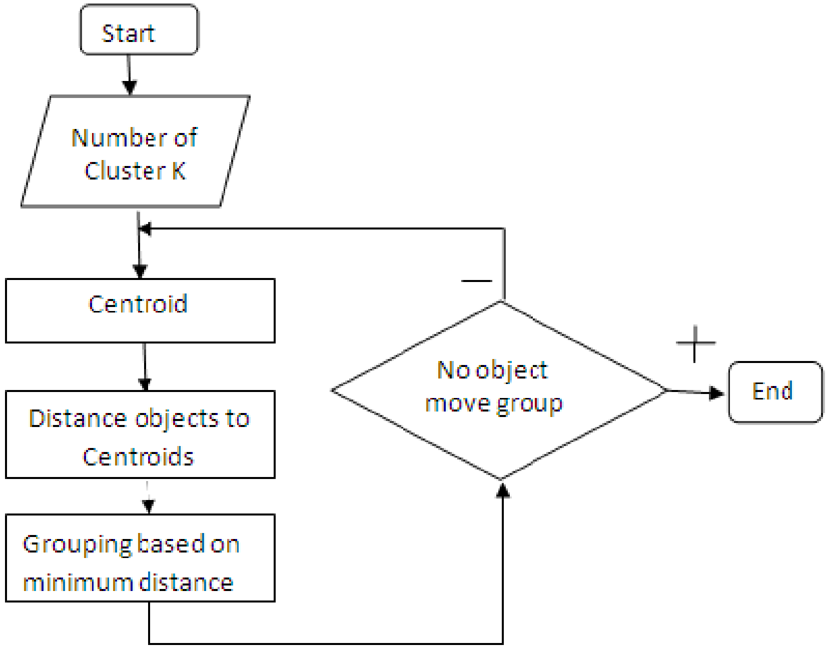
main.cpp (Heartwall/DPC++)

```
params_common_change                                 common_change;
dpct::constant_memory<params_common_change, 0> d_common_change;

params_common                                  common;
dpct::constant_memory<params_common, 0> d_common;

params_unique                                  unique[ALL_POINTS];
dpct::constant_memory<params_unique, 1> d_unique(ALL_POINTS);
```

- DPCT provides useful wrappers for NVIDIA’s memory models:
  - CUDA constant cache: **dpct::constant_memory**
  - CUDA texture cache: **dpct::image_wrapper**
- Migrated codes can be compiled with minimal change

K-Means Clustering: Benchmarks on Heterogeneous Platforms

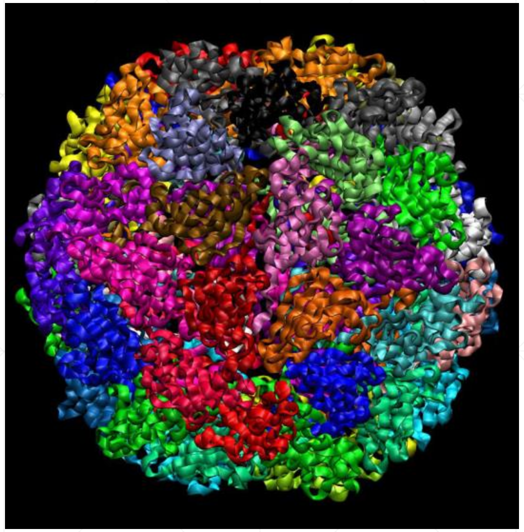


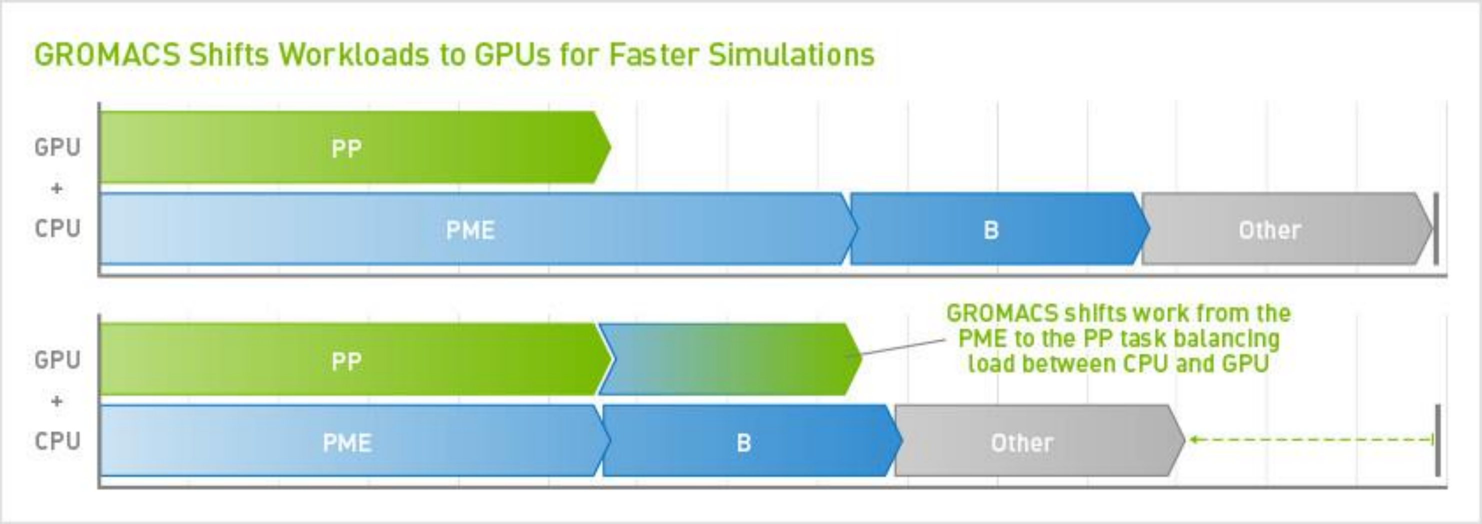


- Problem setup:
  - 100000 random generated data points <x_0, x_1, …x_n> (n = 34)
- Description of algorithm
  - Kernel calculates Euclidian distance between centroid and neighboring objects (FP32)
  - Kernel determines new Centroid and reassign neighboring objects within distance threshold to the newly defined cluster.
- DPC++/CUDA performs within 10% w.r.t native CUDA implementation
- CUDA implementation: http://rodinia.cs.virginia.edu/doku.php?id=kmeans

GROMACS-SYCL

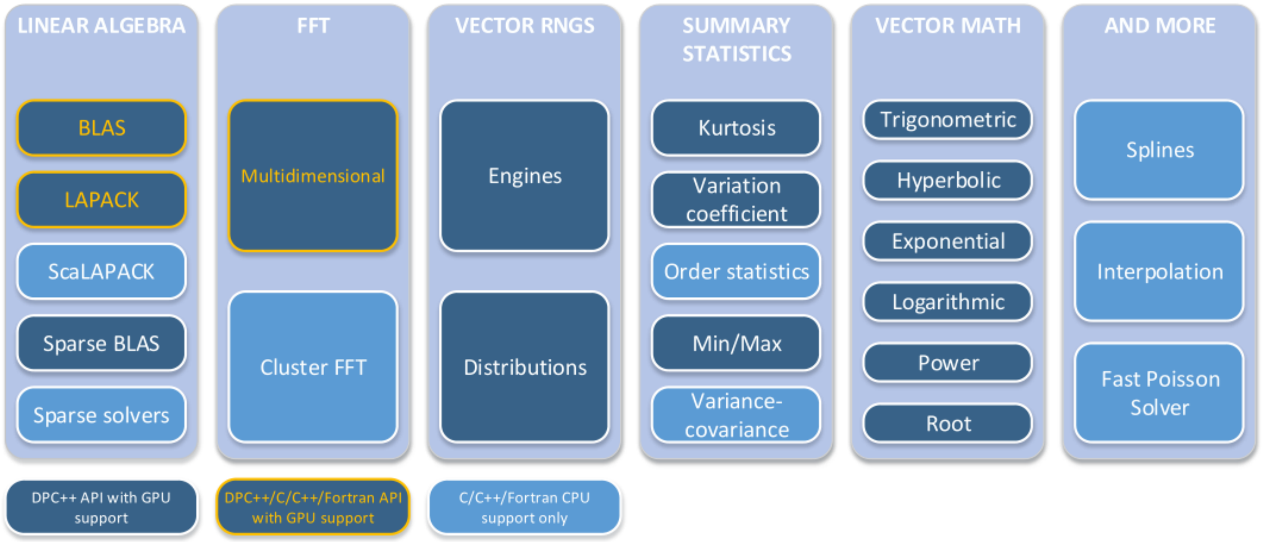
https://www.nvidia.com/en-sg/data-center/gpu-accelerated-applications/gromacs/





- Description of algorithms:
  - GPU kernels only Particle-Particle (PP) and Particle Mesh Ewald (PME)
  - A Fast CPU still required for optimal performance
- Status of SYCL port of GROMACS:
  - GROMACS was successfully built with DPCPP compilers
  - NVIDIA GPUs has received experimental support
  - Optimized version of SYCL kernels are slated to be released in 2022
  - Tested input: STMV virus (~ 1 million atoms), taken from Szilard *et al*, J.Chem.Phys 153, 134110 (2020)
  - Release: https://manual.gromacs.org/documentation/2021-sycl/download.html

oneMKL: Introduction



	Automatic offload	OpenMP offload	Manual offload
Invocation side	CPU		
Data location	CPU	GPU / CPU	GPU / CPU / shared
Interface	C/C++/Fortran	C/C++/Fortran + OpenMP	DPC++
EOY support	None	Most oneMKL GPU functionality	All oneMKL GPU functionality

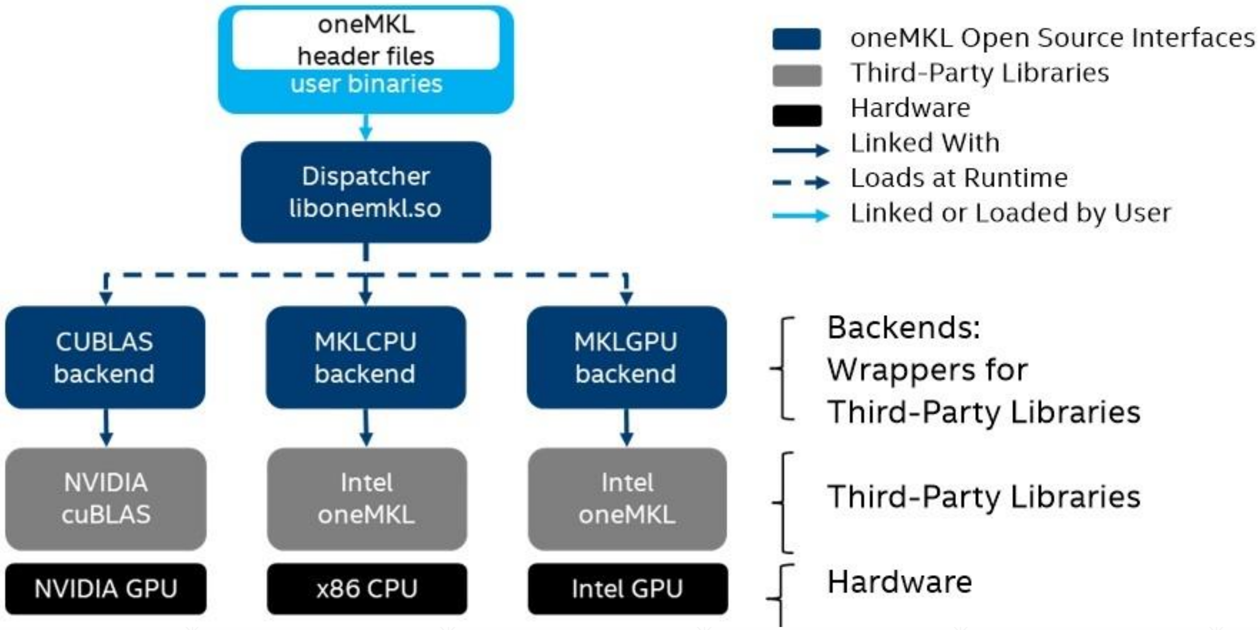
- oneAPI supports two models of off-loading:
  - OpenMP off-loading: C/C++/Fortran interface
  - DPC++ off-loading
- Off-loading interface support:
  - BLAS: full support for CPU and GPU (both USM and buffer)
  - LAPACK: major support for CPU and GPUs (both USM and buffer)
  - ScaLAPACK: only support for CPUs
  - Sparse BLAS/FFT/RNG: selective support for CPUs and GPUs

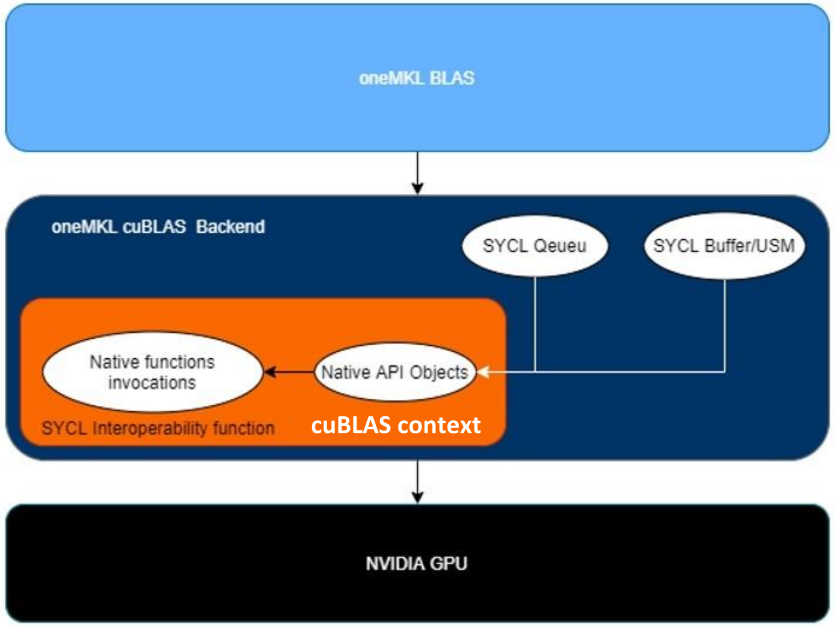
oneMKL Functionalities: oneAPI 2021.2

- **DPC++ supports several types of devices:**
  - Host device: Performs computations directly on the current CPU.
  - CPU device: Performs computations on a CPU using OpenCL™.
  - GPU device: Performs computations on a GPU.
  - Users can choose between USM or SYCL Buffer interfaces when applicable.

- **BLAS:**
  - Full support for BLAS level 1, 2 and 3 on all devices.
- **LAPACK**:
  - Major support on all devices.
  - Not yet available on GPUs: *gerqf* (RQ factorization), *{or,un}mrq* (least-square and eigen-value problems) *etc*
  - https://docs.oneapi.com/versions/latest/onemkl/lapack-functionality.html
- **Sparse BLAS:**
  - Selective support for CPU/GPU devices.
  - https://docs.oneapi.com/versions/latest/onemkl/sparse-blas-functionality.html
- **Discrete Fourier Transform(DFT):**
  - Full support for 1D, 2D and 3D real-to-complex (R2C) and complex-to-complex(C2C) transformations on all devices
  - https://docs.oneapi.com/versions/latest/onemkl/dft-functionality.html

OneMKL Interface: A Vendor-Neutral Path toward Math Acceleration





https://techdecoded.intel.io/resources/a-vendor-neutral-path-to-math-acceleration/

- oneMKL interfaces allows mapping oneMKL calls to 3rd party backends:
  - Automatic selection of backends at runtime based on SYCL device selection, i.e ***cpu_selector{}***/***gpu_selector{}***
    - **iGPU device queue** -> **Intel oneMKL backend selected**
    - **NVIDIA device queue** -> **NVIDIA cuBLAS backend selected**
    - **generic CPU device queue** -> **Netlib referece BLAS backend selected**
  - Automatically conversion of input parameters to suitable format for the 3rd party backends
  - Execution of BLAS function called on a dedicated device using native API objects
- There is plan to support for cuSparse and cuFTT in future.

oneMKL: SGEMM with Unified Shared Memory (USM)

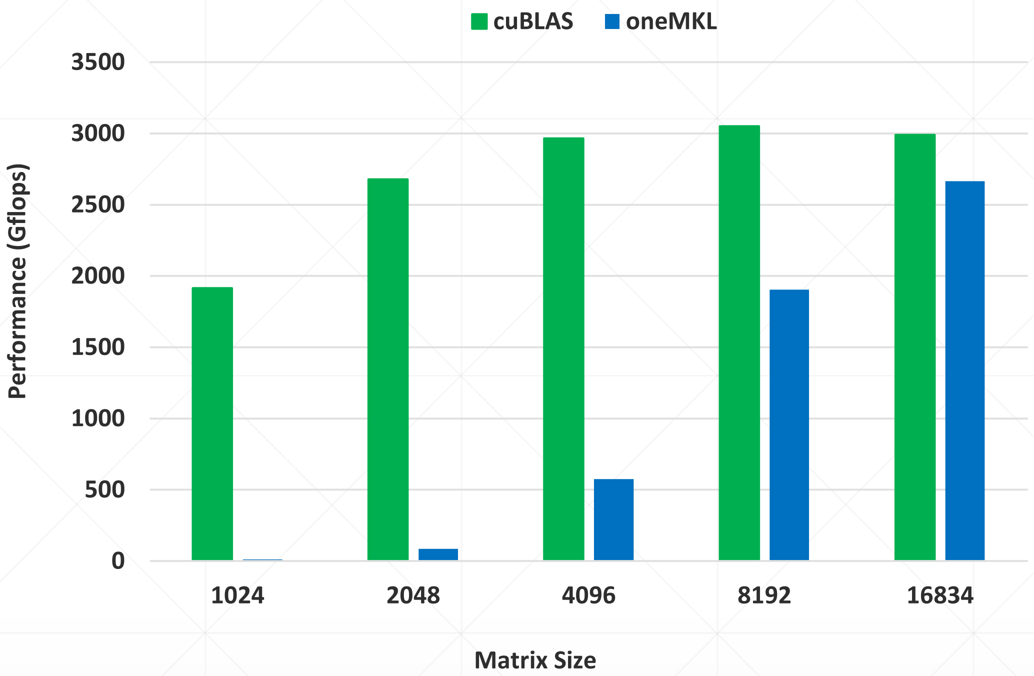
```
// GPU is selected implicitly


// host data
float* A = (float *) aligned_alloc(32, (m * k) * sizeof(float));
float* B = (float *) aligned_alloc(32, (k * n) * sizeof(float));
float* C = (float *) aligned_alloc(32, (m * n) * sizeof(float));

// device data
float *dA, *dB, *dC;
cudaMalloc((void**) &dA, (m * k) * sizeof(float));
cudaMalloc((void**) &dB, (k * n) * sizeof(float));
cudaMalloc((void**) &dC, (m * n) * sizeof(float));

// copy matrix to gpu
cublasSetMatrix(m, k, sizeof(float), A, ldA, dA, ldA);
cublasSetMatrix(k, n, sizeof(float), B, ldB, dB, ldB);
cublasSetMatrix(m, n, sizeof(float), C, ldC, dC, ldC);

// cublas context
cublasStatus_t status;
cublasHandle_t handle;
cublasCreate(&handle);

// cuda events
cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);

cudaEventRecord(start);
for (int i=0; i < LOOP; i++) {
    status = cublasSgemm(
        handle, CUBLAS_OP_N, CUBLAS_OP_N, m, n, k
        &alpha, dA, ldA, dB, ldB, &beta, dC, ldC
    );
}
cudaEventRecord(stop);
// copy data back to host
cublasGetMatrix(m, n, sizeof(float), dC, ldC, C, ldC);
cublasDestroy(handle);
```

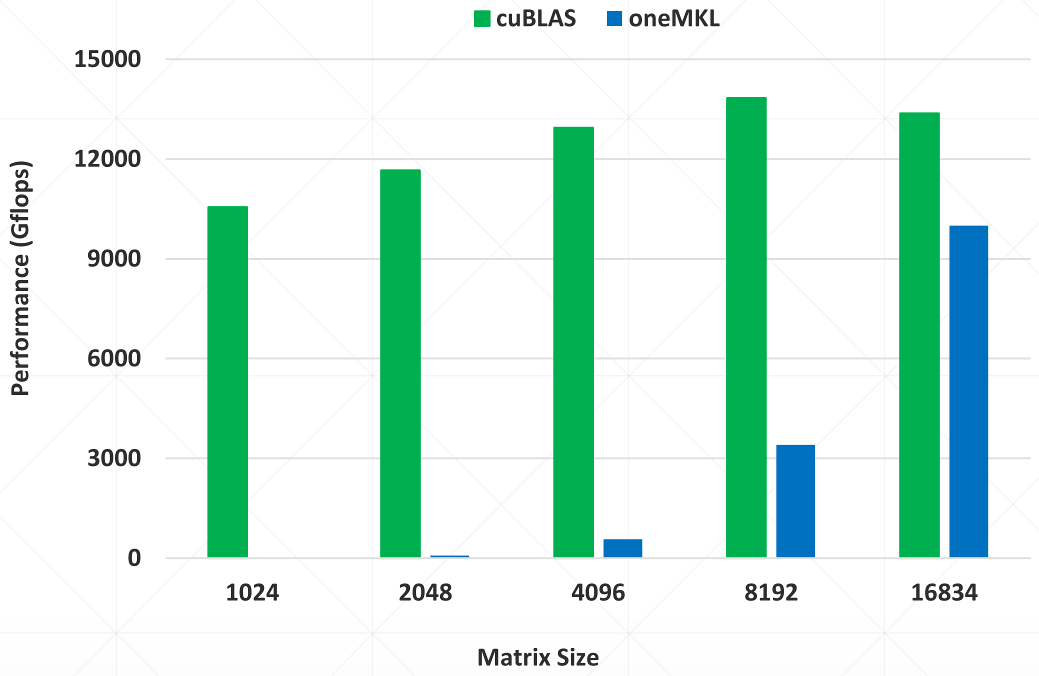
NVIDIA CUDA

```
// device is selected explicitly
sycl::queue dev_queue(sycl::gpu_selector{});

// USM is fully supported with BLAS{1,2,3}
float *A_USM = sycl::malloc_shared<float>(m * k, dev_queue);
float *B_USM = sycl::malloc_shared<float>(k * n, dev_queue);
float *C_USM = sycl::malloc_shared<float>(m * n, dev_queue);


// no explicit device pointer allocation




// no explicit data transfer from host to device




// cuBLAS context is wrapped under oneapi::mkl call









auto start = std::chrono::high_resolution_clock::now();
for (int i=0; i < LOOP; i++) {
   oneapi::mkl::blas::column_major::gemm(
          dev_queue, transA, transB, m, n, k,
          alpha, A_USM, ldA, B_USM, ldB, beta, C_USM, ldC
  ).wait();
}
auto end = std::chrono::high_resolution_clock::now();
// no explicit data transfer from device back to host
```

oneAPI

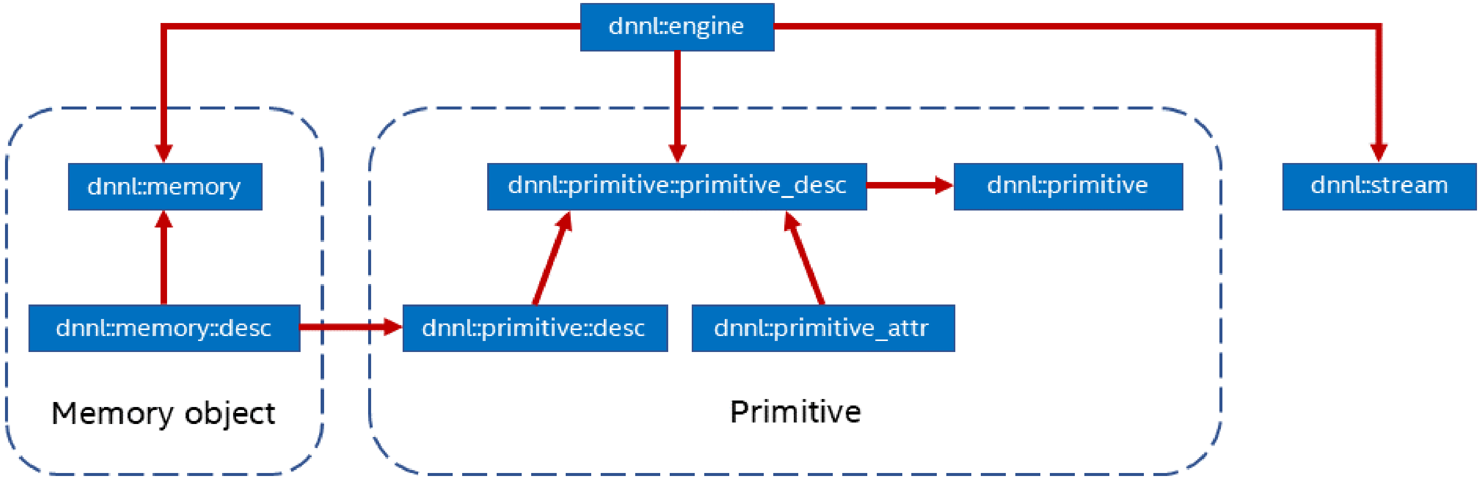
oneMKL: SGEMM Performance on NVIDIA Devices





- For Tesla K40:
  - Theoretical: 4.3 TFops / **SGEMM (cuBLAS)**: 3 TFlops (70% efficiency) / **SGEMM (oneMKL)**: 2.6 TFlops (60% efficiency)
- For Tesla V100:
  - Theoretical: 15 Tflops / **SGEMM(cuBLAS)**: 13.8 Tflops (90% efficiency) / **SGEMM (oneMKL)**: 10Tflops (66% efficiency)
- Reason for poor performance of oneMKL with matrices on NVIDIA devices:
  - **cuBLAS**: context creation is expensive but can be effectively excluded from *cudaEventRecord()*
  - **oneMKL**: context creation is wrapped under oneMKL and cannot be subtracted from *std::chrono* measurement
  - The overhead can be mitigated by using a sufficiently large matrix (N = 16834) where $t_{context} \ll t_{BLAS}$

oneDNN: Introduction



- oneDNN is built on the concept on a primitive (***dnnl::primitive***)
  - A primitive encapsulates a specific type of computation:
    - *Convolution*: forward, backward, or weight update for a batched convolution operation on 1D, 2D, or 3D spatial data with bias.
    - *Inner Product*: treat minibatch as a vector and computes its product with a weights 2D tensor producing a 2D tensor as an output.
    - *Matrix Multiplication*: computes the product of two 2D tensors with optional bias addition
    - *PReLU*: performs forward or backward operation on data tensor.
    - https://docs.oneapi.com/versions/latest/onednn/supported_primitives.html
- Engines (***dnnl::engine***) is abstraction for computational devices such as CPU or GPU
- Streams (***dnnl::stream***) encapsulate execution context such as OpenCL command queues
- Memory objects (***dnnl::memory***) encapsulates memory allocated on engine, tensor dimension, data type.
- Support for NVIDIA devices has been introduced into oneDNN recently.

Conclusions

- We performed systematic benchmarks of oneAPI and DPC++
  - Memory bandwidth: STREAM
  - Computational fluid dynamics: lid-driven cavity flow
  - Data mining/Machine learning: k-means clustering
  - Biomedical imaging: heart wall tracking
  - Biomolecular simulation: GROMACS-SYCL

- Portability vs. Performance:
  - DPC++ archive high level of portability with a acceptable performance penalty
  - DPC++ compatibility tools help provide seamless experiences migrating CUDA codes to DPC++

- oneMKL interoperability with cuBLAS
  - Performance of SGEMM on NVIDIA devices is respectable (~ 60%)