Data Parallel C++

Performance, Portability & Productivity

Rakshith Krishnappa, Developer Evangelist @Intel

rakshith.krishnappa@intel.com

Performance, Portability and Productivity

01/12/2022	Introduction to Performance, Portability and Productivity for HPC
02/09/2022	Optimization Best Practices using SYCL
03/09/2022	Optimization for Performance Portability across CPU and GPU
03/16/2022	Performance, Portability and Productivity – Mini Hackathon

What will you learn in this Learning Series

Session 1 (Jan 12th 2022)

- Understand why Performance, Portability and Productivity are important for HPC
- Identify an algorithm and implement using Math Kernel Library and check for performance on CPUs and GPUs
- Implement the same algorithm using basic SYCL programming
- Analyze results using Intel Advisor Roofline and Intel VTune Profiler
- We will do all this on Intel DevCloud on the following CPUs and GPUs:
  - Intel® Xeon® E-2176G Processor with **GEN9 GT2 Graphics**
  - Intel (R) Core (TM) i9-10920X **with Iris XE Max discrete Graphics**
  - Intel® Xeon® Gold 6128 Processor
  - Intel® Xeon® Platinum 8153 Processor

What will you learn in this Learning Series

Session 2 (Feb 9th 2022)

- Use SYCL features to tune the basic algorithm.
- Learn about using ND_Range kernels and impact of work-group size
- Use private memory and shared local memory to improve performance
- Analyze results using Intel Advisor Roofline and Intel VTune Profiler

What will you learn in this Learning Series

Session 3 (Mar 9th 2022)

- Optimize algorithm for Performance Portability across CPUs and GPUs
- Analyze results using Intel Advisor Roofline and Intel VTune Profiler and compare all algorithm implementations.
- Understand different accelerator hardware characteristics and further optimize algorithm
- Understand impact of accelerator Occupancy and impact of varying Work-group sizes
- Resources for more advanced tuning using the oneAPI GPU optimization guide

What will you learn in this Learning Series

Session 4 (Mar 16th 2022)

- Mini-Hackathon and working session, bring your own code, share and ask us any questions.
- We will have a bunch of Intel experts with hardware architecture , SYCL language experts, Performance tuning experts and Tools experts.
- We will have break-out rooms to work with Intel experts.

Programming Challenges

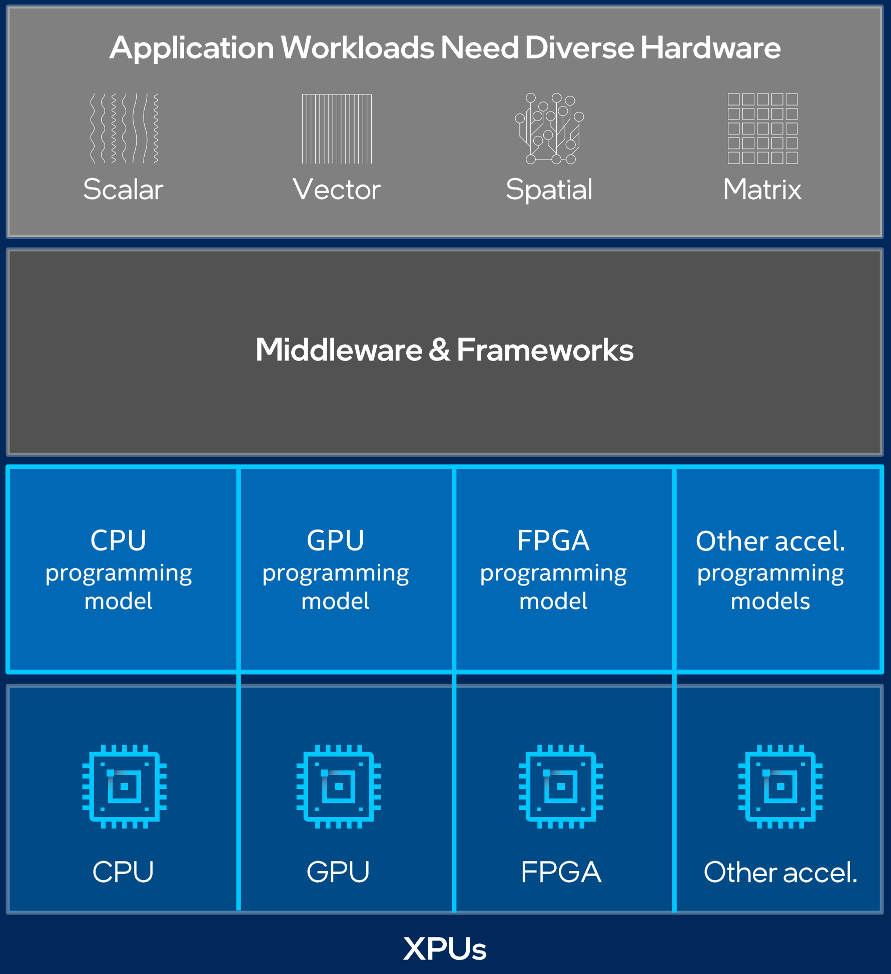
for Multiple Architectures

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



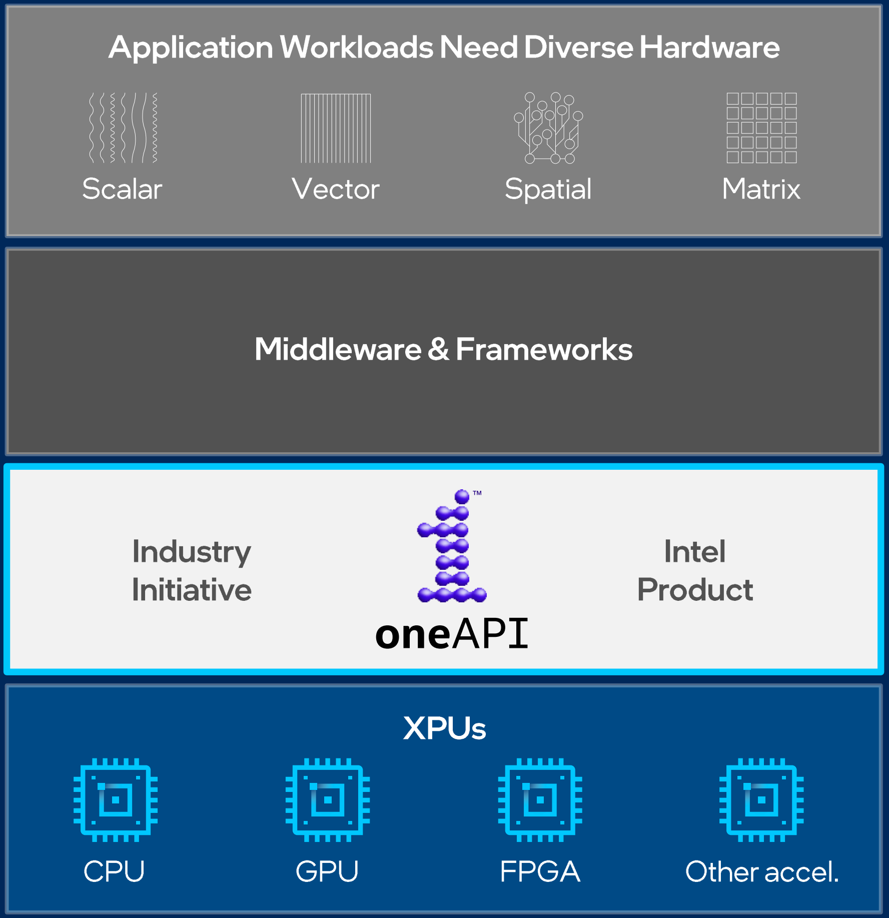
Introducing
oneAPI

Cross-architecture programming that delivers freedom to choose the best hardware

Based on industry standards and open specifications

Exposes cutting-edge performance features of latest hardware

Compatible with existing high-performance languages and programming models including C++, OpenMP, Fortran, and MPI



Data Parallel C++

Standards-based, Cross-architecture Language
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

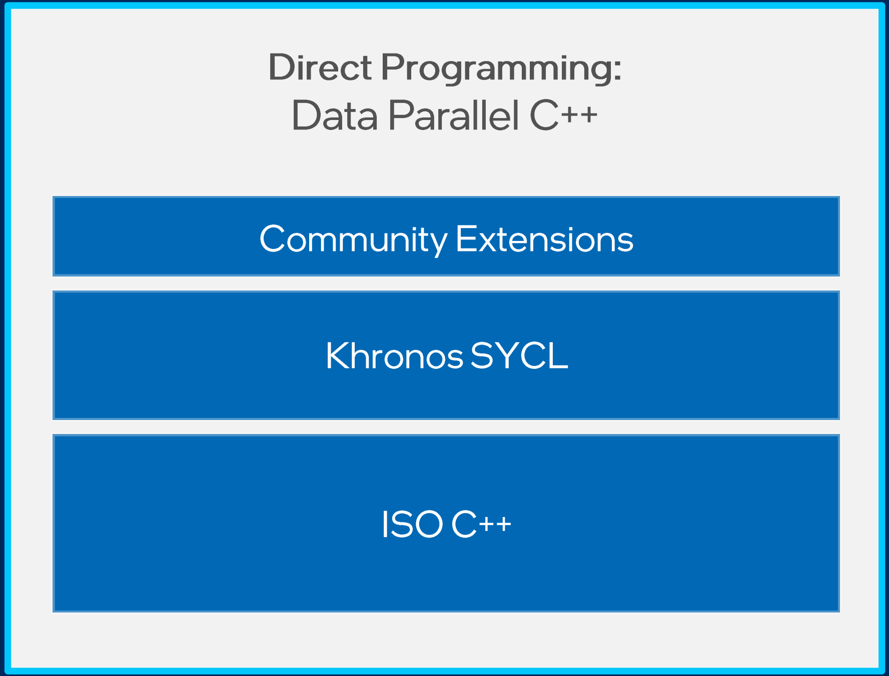
Based on C++ and SYCL

- 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 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.

Data Parallel C++

DPC++ Essentials Learning Path:

https://www.intel.com/content/www/us/en/developer/tools/oneapi/training/dpc-essentials.html

DPC++ Book:

https://www.apress.com/us/book/9781484255735

oneAPI GPU Optimization Guide:

https://www.intel.com/content/www/us/en/develop/documentation/oneapi-gpu-optimization-guide/top/intro.html

SYCL 2020

Specification:

https://www.khronos.org/registry/SYCL/specs/sycl-2020/pdf/sycl-2020.pdf

SYCL 2020 Specification

- The SYCL 2020 specification was released on February 9, 2021

- Several major features, including:
  - Unified shared memory
  - Reductions
  - Modern atomics
  - Sub-groups
  - Group algorithms (e.g. reductions, scans)
  - Extension and interoperability mechanisms

DPC++ Extensions now part of SYCL 2020

Many of DPC++ Extensions became part of SYCL 2020 specification:

- Unified Shared Memory (USM)
- Sub-Groups
- Reductions

DPC++ now adopts the SYCL 2020 syntax for the above features.

Unified Shared Memory in SYCL 2020

Unified Shared Memory(USM) is now part of SYCL 2020, No changes in USM syntax.

Type	Description	Accessible on Host?	Accessible on Device?
`sycl::malloc_device`	Allocations in device memory. Programmer must explicitly transfer data between host and device.	No	Yes
`sycl::malloc_host`	Allocations in host memory. Kernels can access these allocations directly.	Yes	Yes
`sycl::malloc_shared`	Allocations can migrate between host and device memory. Different implementations may provide different guarantees regarding whether allocations can be accessed by host and device concurrently.	Yes	Yes

Sub-Groups in SYCL 2020

Sub-Groups are now part of SYCL 2020 Specification

sub_group class

A sub-group handle can be obtained from an **`nd_item`** using `get_sub_group()`

Before

```
sycl::ONEAPI::sub_group sg = item.get_sub_group();

sycl::ext::oneapi::sub_group sg = item.get_sub_group();
```

Now

```
sycl::sub_group sg = item.get_sub_group();

auto sg = item.get_sub_group();
```

Sub-Groups in SYCL 2020

DPC++ Sub-Groups Shuffles are now part of SYCL 2020 Sub-Group "Group Algorithms" free functions instead of member functions.

Old DPC++ Compiler	Current DPC++ Compiler (SYCL 2020)
`sg.shuffle_down(x, 1);`	`sycl::shift_group_left(sg, x, 1);`
`sg.shuffle_up(x, 1);`	`sycl::shift_group_right(sg, x, 1);`
`sg.shuffle(x, id);`	`sycl::select_from_group(sg, x, id);`
`sg.shuffle_xor(x, mask);`	`sycl::permute_group_by_xor(sg, x, mask);`

Sub-Groups in SYCL 2020

DPC++ Sub-Groups Collective are now part of SYCL 2020 Sub-Group "Group Algorithms" with function name changes.

Old DPC++ Compiler	Current DPC++ Compiler (SYCL 2020)
`broadcast(sg, x, id);`	`group_broadcast(sg, x, id);`
`reduce(sg, x, op);`	`reduce_over_group(sg, x, op);`
`exclusive_scan(sg, x, op);`	`exclusive_scan_over_group (sg, x, op);`
`inclusive_scan(sg, x, op);`	`inclusive_scan_over_group (sg, x, op);`
`any_off(sg, x);`	`any_off_group(sg, x);`
`all_off(sg, x);`	`all_off_group(sg, x);`
`none_off(sg, x);`	`none_off_group(sg, x);`

Reductions in SYCL 2020

DPC++ introduced a dedicated abstraction for reduction kernels.

This is now part of SYCL 2020.

Before

```
q.parallel_for(nd_range<1>{N, B},
   sycl::ext::oneapi::reduction(sum, 0, sycl::ext::oneapi::plus<>()),
     [=](nd_item<1> it, auto& tmp) {
     int i = it.get_global_id(0);
     tmp += data[i];
 }).wait();
```

Now

```
q.parallel_for(nd_range<1>{N, B},
  sycl::reduction(sum, 0, sycl::plus<>()), [=](nd_item<1> it, auto& tmp) {
     int i = it.get_global_id(0);
     tmp += data[i];
 }).wait();
```

Reductions in SYCL 2020

Below example shows changes in reductions in SYCL2020 when using USM and Buffers

	Old DPC++	SYCL 2020
USM	`ext::oneapi::reduction(sum, ext::oneapi::plus<>())`	`sycl::reduction(sum, sycl::plus<>())`
Buffers	`ext::oneapi::reduction(sum_acc, ext::oneapi::plus<>())`	`sycl::reduction(sum_buf, h, sycl::plus<>())`

Reductions in SYCL 2020

SYCL 2020 specification extends kernel reductions even further by allowing multiple reductions in a single kernel.

```
auto reduction_min = sycl::reduction(min, sycl::minimum<>());
auto reduction_max = sycl::reduction(max, sycl::maximum<>());

q.parallel_for(nd_range<1>{N, B},
                  reduction_min, reduction_max,
                  [=](nd_item<1> it, auto& tmp_min , auto& tmp_max) {
     int i = it.get_global_id(0);
     tmp_min.combine(data[i]);
     tmp_max.combine(data[i]);
}).wait();
```

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

Run the tools in the Cloud

intel. DevCloud

oneAPI

oneAPI Available on Intel® DevCloud

A development sandbox to develop, test and run workloads across a range of Intel CPUs, GPUs, and FPGAs using Intel's oneAPI software.

Get Up & Running In Seconds!

software.intel.com/devcloud/oneapi

intel DevCloud

1 Minute to Code

No Hardware Acquisition

No Download, Install or Configuration

Easy Access to Samples & Tutorials

Support for Jupyter Notebooks, Visual Studio Code

Intel DevCloud

- Login to Intel DevCloud - devcloud.intel.com/oneapi
- Get Started -> Launch Jupyter Lab option
- Start Terminal and enter command to copy latest content
  - /data/oneapi_workshop/get_jupyter_notebooks.sh

Legal Notices and Disclaimers