oneAPI

PCCC HPCオープンソースソフトウェア普及部会ワークショップ「高性能クラスタ・プログラミング最前線」

Intel® oneAPI Base & HPC Toolkitsのご紹介

野村 昴太郎 / Kentaro Nomura
HPC ソフトウェア テクニカル セールス スペシャリスト
AXG & AI ソリューション & セールス グループ

HPC wire
2021 Readers' Choice Awards
Best HPC Programming Tool or Technology
oneAPI

intel®

Programming Challenges

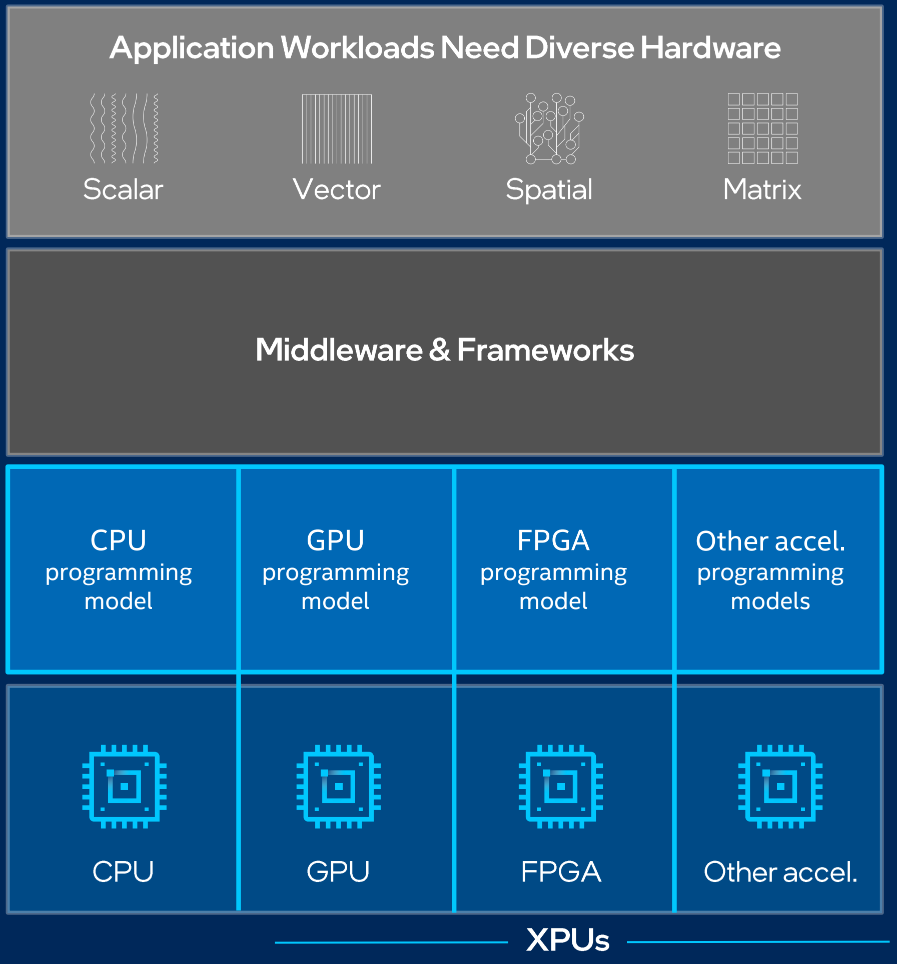
for Multiple Architectures

Growth in specialized workloads

Variety of data-centric hardware required

Requires separate programming models and toolchains for each architecture

Software development complexity limits freedom of architectural choice



oneAPI

One Programming Model for Multiple Architectures and Vendors

HPC wire
2021 Readers' Choice Awards
Best HPC Programming Tool or Technology

Freedom to Make Your Best Choice

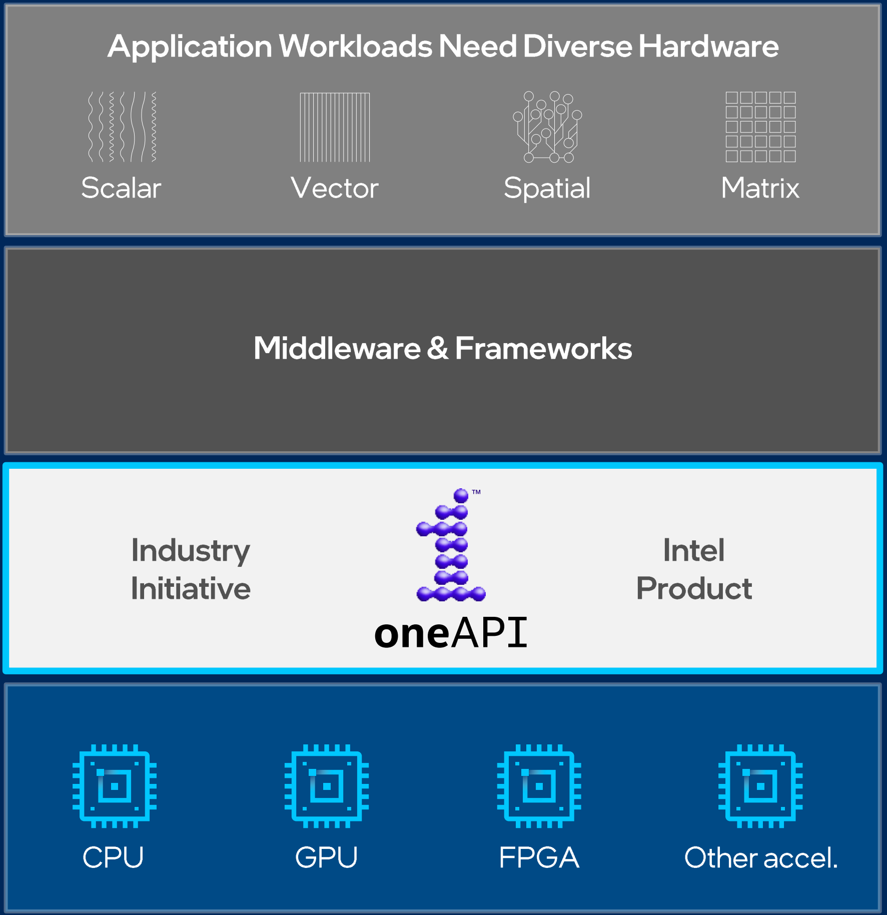
- Choose the best accelerated technology the software doesn't decide for you

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++, Python, SYCL, OpenMP, Fortran, and MPI

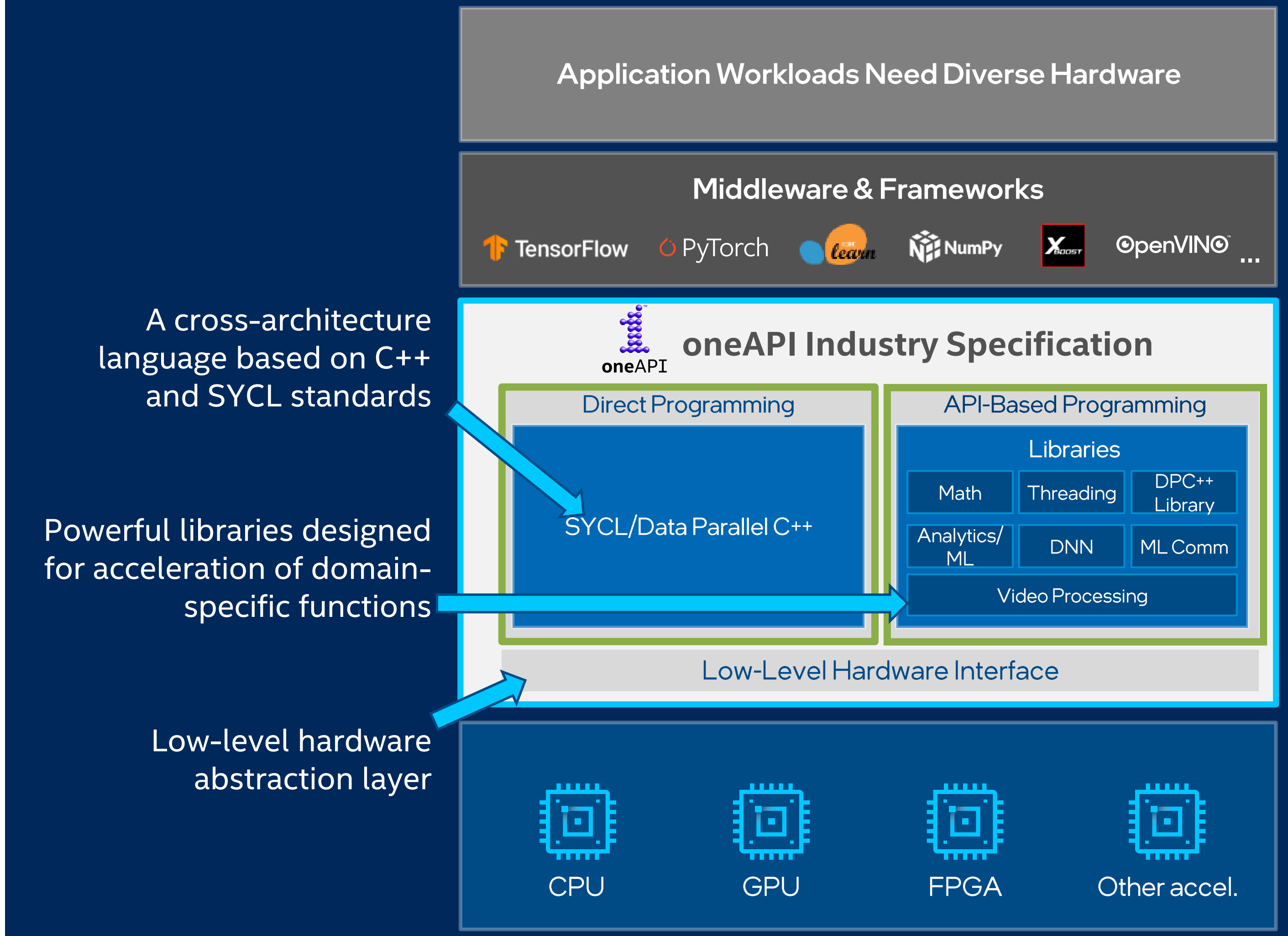


oneAPI Industry Initiative

Break the Chains of Proprietary Lock-in

Open to promote community and industry collaboration

Enables code reuse across architectures and vendors



Application Workloads Need Diverse Hardware
Middleware & Frameworks
TensorFlow
PyTorch
learn
NumPy
XGBoost
OpenVINO
...
oneAPI
oneAPI Industry Specification
A cross-architecture language based on C++ and SYCL standards
Powerful libraries designed for acceleration of domain-specific functions
Low-level hardware abstraction layer
Direct Programming
SYCL/Data Parallel C++
API-Based Programming
Libraries
Math
Threading
DPC++ Library
Analytics/ ML
DNN
ML Comm
Video Processing
Low-Level Hardware Interface
CPU
GPU
FPGA
Other accel.

oneAPI

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

Visit oneapi.com for more details

Data Parallel C++

oneAPI's implementation of SYCL

DPC++ = ISO C++ and Khronos SYCL and community extensions

Freedom of Choice: Future-Ready Programming Model

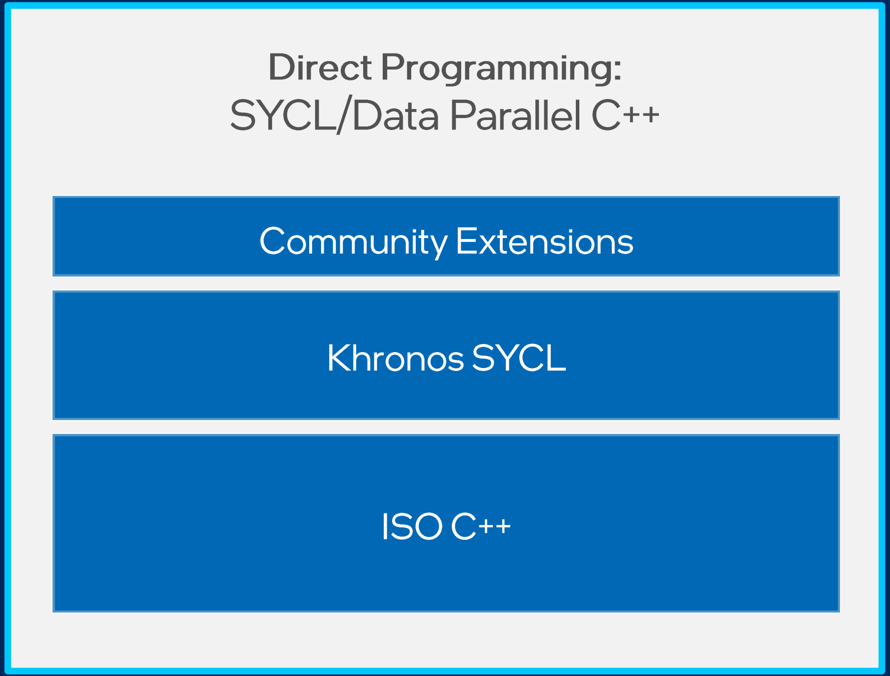
- Allows code reuse across hardware targets
- Permits custom tuning for a specific accelerator
- Open, cross-industry alternative to proprietary language

DPC++ = ISO C++ and Khronos SYCL and community extensions

- Designed for data parallel programming productivity
- Provides full native high-level language performance on par with standard C++ and broad compatibility
- Adds SYCL from the Khronos Group for data parallelism and heterogeneous programming

Community Project Drives Language Enhancements

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



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

Powerful oneAPI Libraries

Realize all the Hardware Value

Designed for acceleration of key domain-specific functions

Freedom of Choice

Pre-optimized for each target platform for maximum performance

oneAPI Math Kernel Library **oneMKL**	oneAPI Deep Neural Network Library **oneDNN**
oneAPI Video Processing Library **oneVPL**	oneAPI Data Analytics Library **oneDAL**
oneAPI Threading Building Blocks **oneTBB**	oneAPI Collective Communications Library **oneCCL**
oneAPI DPC++ Library **oneDPL**	Ray Tracing
oneAPI Image Processing Library* **oneIPL**	oneAPI Data Transformation Library* oneDTL

*New element in v1.2 oneAPI Provisional Specification

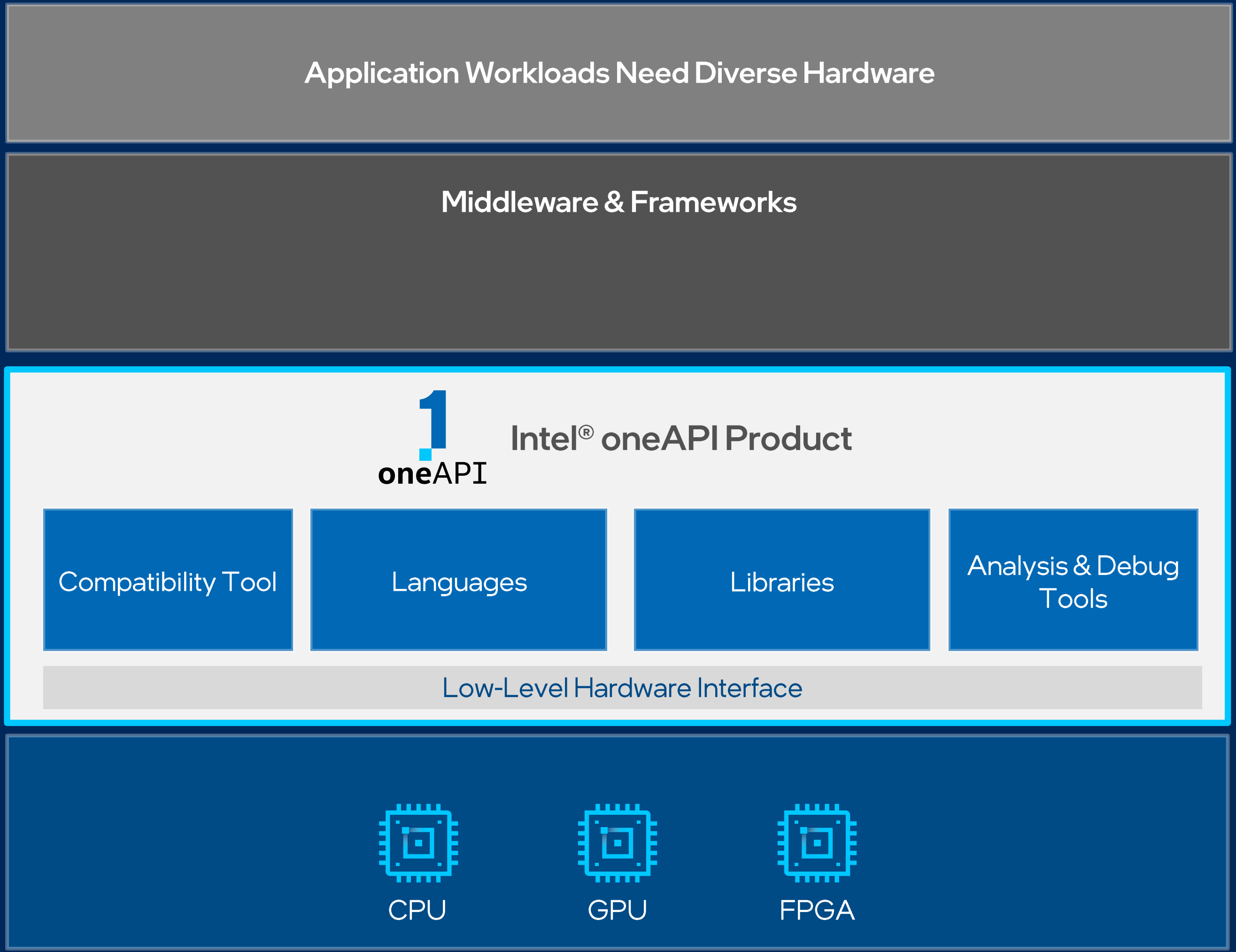
Intel® oneAPI Tools

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

Latest version is 2021.1



Available Now

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.

Intel® oneAPI Toolkits

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

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

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

Toolkit **powered by oneAPI**

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

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!

Intel® oneAPI Tools for HPC

Intel® oneAPI HPC Toolkit

Deliver Fast Applications that Scale

What is it?

A toolkit that adds to the Intel® oneAPI Base Toolkit for building high-performance, scalable parallel code on C++, SYCL, Fortran, OpenMP & MPI from enterprise to cloud, and HPC to AI applications.

Who needs this product?

- OEMs/ISVs
- C++, Fortran, OpenMP, MPI Developers

Why is this important?

- Accelerate performance on Intel® Xeon® and Core™ Processors and Intel® Accelerators
- Deliver fast, scalable, reliable parallel code with less effort built on industry standards

Intel® oneAPI Base & HPC Toolkits

Direct Programming	API-Based Programming	Analysis & debug Tools
Intel® C++ Compiler Classic	Intel® MPI Library	Intel® Inspector
Intel® Fortran Compiler Classic	Intel® oneAPI DPC++ Library oneDPL	Intel® Trace Analyzer & Collector
Intel® Fortran Compiler	Intel® oneAPI Math Kernel Library - oneMKL	Intel® Cluster Checker
Intel® oneAPI DPC++/C++ Compiler	Intel® oneAPI Data Analytics Library - oneDAL	Intel® VTune™ Profiler
Intel® DPC++ Compatibility Tool	Intel® oneAPI Threading Building Blocks - oneTBB	Intel® Advisor
Intel® Distribution for Python	Intel® oneAPI Video Processing Library - oneVPL	Intel® Distribution for GDB
Intel® FPGA Add-on for oneAPI Base Toolkit	Intel® oneAPI Collective Communications Library oneCCL	
	Intel® oneAPI Deep Neural Network Library - oneDNN	
	Intel® Integrated Performance Primitives – Intel® IPP	

■ Intel® oneAPI **HPC** Toolkit +

■ Intel® oneAPI **Base** Toolkit

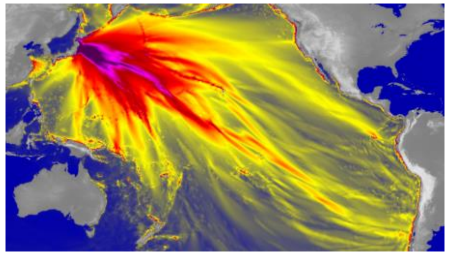
Learn More: intel.com/oneAPI-HPCKit

Deliver Fast HPC Applications that Scale

Customer Use Cases – Intel® oneAPI Base & HPC Toolkits

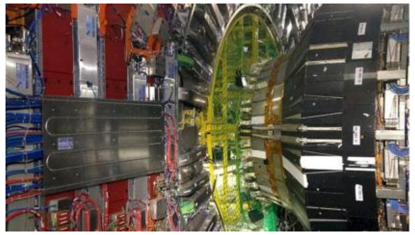
SAMPLE USE CASES & PROOF POINTS

Intel oneAPI tools help prepare code for Aurora.
Aurora, Argonne Leadership Computing Facility's Intel-HPE/Cray supercomputer, will be one of the U.S.'s 1st exascale systems

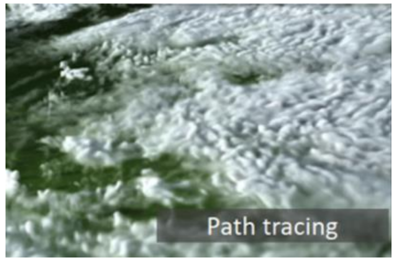


Zuse Institute Berlin (ZIB) ported the *easyWave* tsunami simulation application from CUDA to Data Parallel C++ (DPC++) **delivering performance on Intel CPUs, GPUs, FPGAs**, & Nvidia P100

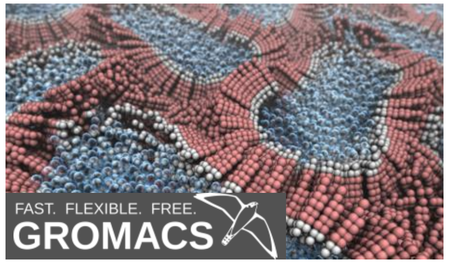
Accelerating Google Cloud for HPC
C2 provides great performance for HPC workloads: **40% higher performance/core.** Runs on Intel® Xeon® processors + AMD, optimized by Intel® oneAPI **Base** & **HPC** Toolkits. Video
Video | Podcast



Acceleration for HPC & AI Inferencing
CERN, SURFsara, and Intel are investigating approaches driving **breakthrough performance on simulations** used in scientific, engineering, and financial applications*.



Texas Advanced Computing Center (TACC)
Frontera Supercomputer Visualization & Filesystem Use Cases Show Value of Large Memory Fat Nodes on Intel® Xeon® processors & Intel® Optane Persistent Memory*



University of Stockholm/KTH
GROMACS, a simulation application used to design new drugs, was optimized by oneAPI. CUDA code was migrated to oneAPI to create new cross-architecture code targeting Intel CPUs and multiple accelerators.

Learn more: *oneAPI Discussions with HPC Thought Leaders* Video [2.20]

*Uses Intel® oneAPI Rendering Toolkit

Key Tools for HPC Development

Intel® DPC++ Compatibility Tool

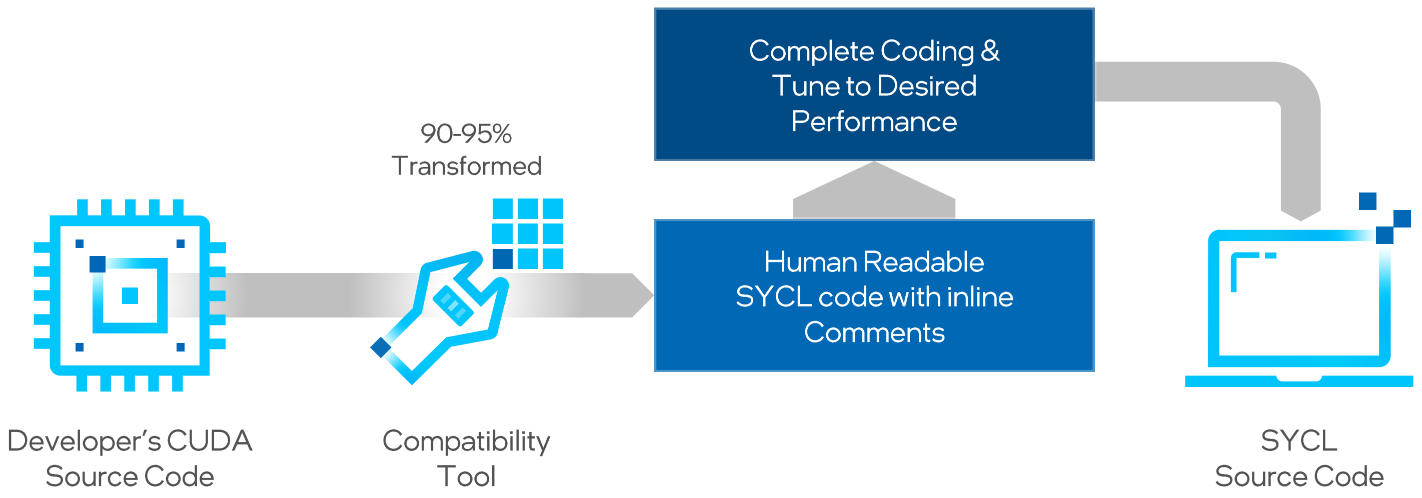
Minimizes Code Migration Time

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

~90-95% 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
90-95% Transformed
Human Readable SYCL code with inline Comments
Developer's CUDA Source Code
Compatibility Tool
SYCL Source Code

Intel® Compilers in Toolkits

Understanding your Intel Compiler Choices

Key Knowledge for Intel® Compilers Going Forward

New underlying back-end Compilation Technology based on LLVM

New compiler technology available in Intel® oneAPI Base & HPC Toolkit for DPC++/SYCL, C++, and Fortran

Existing Intel proprietary "IL0" (ICC, IFORT) Compilation Technology compilers provided alongside new compilers

- ***CHOICE! Continuity!***

BUT Offload (DPC++/SYCL or OpenMP TARGET) supported only with new LLVM-based compilers

Intel® Compilers – Target & Packaging

Intel Compiler	Driver	Target*	OpenMP Support	OpenMP Offload Support
Intel® C++ Compiler Classic	***icc***	CPU	Yes	No
Intel® oneAPI DPC++/C++ Compiler	***dpcpp***	CPU, GPU, FPGA	No	No
	icx	CPU GPU	Yes	Yes
Intel® Fortran Compiler Classic	***ifort***	CPU	Yes	No
Intel® Fortran Compiler	***ifx***	CPU, GPU	Yes	Yes

Cross-Compiler Binary Compatible and Linkable!

*Intel® Platforms

Data Parallel C++ (DPC++): oneAPI's implementation of the Khronos SYCL standard

Let's Get Started!

Data Parallel C++: oneAPI's implementation of SYCL

DPC++ = ISO C++ and Khronos SYCL and community extensions

Freedom of Choice: Future-Ready Programming Model

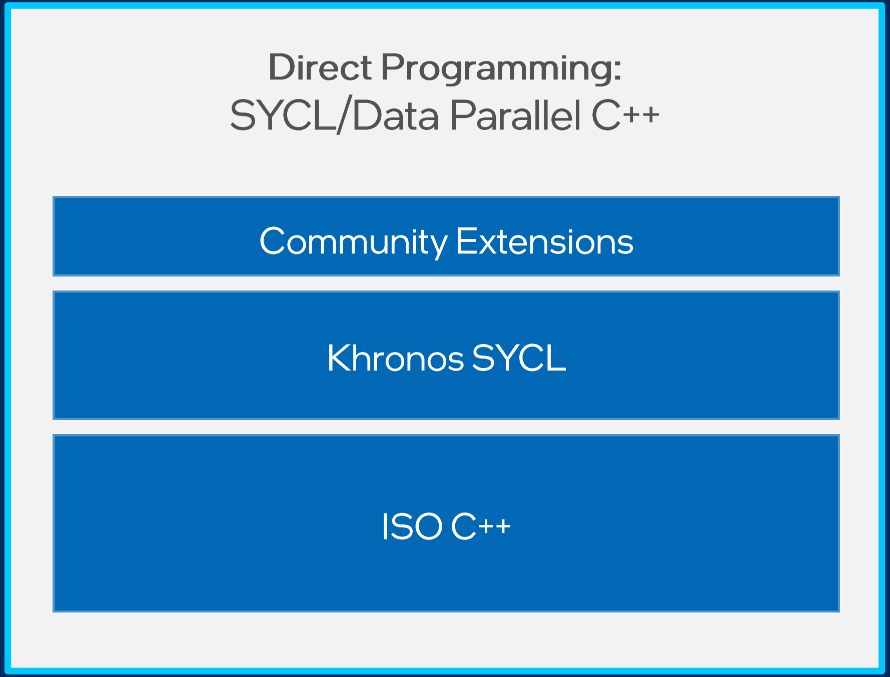
- Allows code reuse across hardware targets
- Permits custom tuning for a specific accelerator
- Open, cross-industry alternative to proprietary language

DPC++ = ISO C++ and Khronos SYCL and community extensions

- Designed for data parallel programming productivity
- Provides full native high-level language performance on par with standard C++ and broad compatibility
- Adds SYCL from the Khronos Group for data parallelism and heterogeneous programming

Community Project Drives Language Enhancements

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



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

```
#include <CL/sycl.hpp>
#include <iostream>
```

SYCL “Hello world”: Vector addition

```
void main() {
    using namespace cl::sycl;
    float A[1024], B[1024], C[1024];
    {
        buffer<float, 1> bufA { A, range<1> {1024} };
        buffer<float, 1> bufB { B, range<1> {1024} };
        buffer<float, 1> bufC { C, range<1> {1024} };

        queue myQueue;
        myQueue.submit([&](handler& cgh) {
            auto accA = bufA.get_access<access::read>(cgh);
            auto accB = bufB.get_access<access::read>(cgh);
            auto accC = bufC.get_access<access::write>(cgh);

            cgh.parallel_for<class vector_add>(range<1> {1024}, [=](id<1> i) {
                accC[i] = accA[i] + accB[i];
            });
        }).wait();
    }
    for (int i = 0; i < 1024; i++)
        std::cout << “C[“ << i << “] = “ << C[i] << std::endl;
}
```

intel.

```
#include <CL/sycl.hpp>
#include <iostream>

void main() {
    using namespace cl::sycl;
    float A[1024], B[1024], C[1024];
    {
        buffer<float, 1> bufA { A, range<1> {1024} };
        buffer<float, 1> bufB { B, range<1> {1024} };
        buffer<float, 1> bufC { C, range<1> {1024} };

        queue myQueue;
        myQueue.submit([&](handler& cgh) {
            auto accA = bufA.get_access<access::read>(cgh);
            auto accB = bufB.get_access<access::read>(cgh);
            auto accC = bufC.get_access<access::write>(cgh);

            cgh.parallel_for<class vector_add>(range<1> {1024}, [=](id<1> i) {
                accC[i] = accA[i] + accB[i];
            });
        });
    }
    for (int i = 0; i < 1024; i++)
        std::cout << "C[" << i << "] = " << C[i] << std::endl;
}
```

Create SYCL buffers using host pointers.

```
#include <CL/sycl.hpp>
#include <iostream>

void main() {
    using namespace cl::sycl;
    float A[1024], B[1024], C[1024];
    {
        buffer<float, 1> bufA { A, range<1> {1024} };
        buffer<float, 1> bufB { B, range<1> {1024} };
        buffer<float, 1> bufC { C, range<1> {1024} };

        queue myQueue;
        myQueue.submit([&](handler& cgh) {
            auto accA = bufA.get_access<access::read>(cgh);
            auto accB = bufB.get_access<access::read>(cgh);
            auto accC = bufC.get_access<access::write>(cgh);

            cgh.parallel_for<class vector_add>(range<1> {1024}, [=](id<1> i) {
                accC[i] = accA[i] + accB[i];
            });
        });
    }
    for (int i = 0; i < 1024; i++)
        std::cout << "C[" << i << "] = " << C[i] << std::endl;
}
```

Create a queue to submit work to a device (including host).

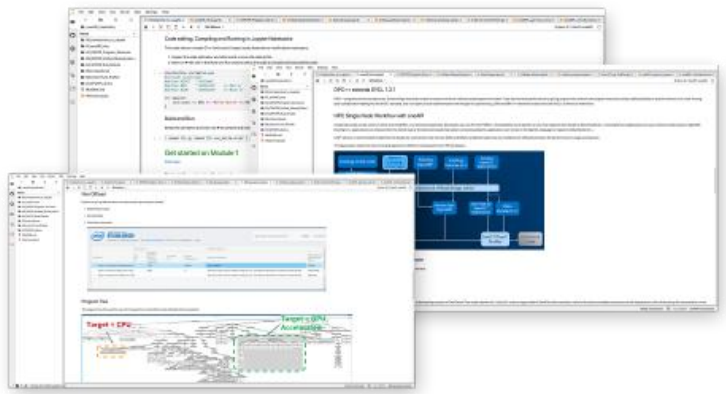
```
#include <CL/sycl.hpp>
#include <iostream>

void main() {
    using namespace cl::sycl;
    float A[1024], B[1024], C[1024];
    {
        buffer<float, 1> bufA { A, range<1> {1024} };
        buffer<float, 1> bufB { B, range<1> {1024} };
        buffer<float, 1> bufC { C, range<1> {1024} };

        queue myQueue;
        myQueue.submit([&](handler& cgh) {
            auto accA = bufA.get_access<access::read>(cgh);
            auto accB = bufB.get_access<access::read>(cgh);
            auto accC = bufC.get_access<access::write>(cgh);

            cgh.parallel_for<class vector_add>(range<1> {1024}, [=](id<1> i) {
               accC[i] = accA[i] + accB[i];
            });
        });
    }
    for (int i = 0; i < 1024; i++)
        std::cout << "C[" << i << "] = " << C[i] << std::endl;
}
```

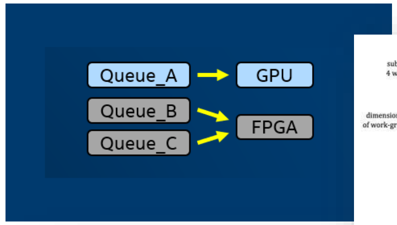
Read/write accessors create dependencies if other kernels or host access buffers.

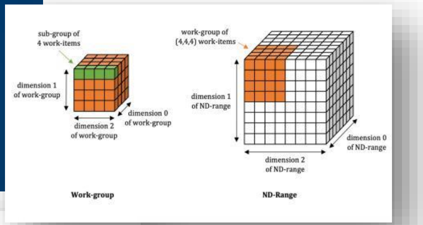
```
#include <CL/sycl.hpp>
#include <iostream>

void main() {
    using namespace cl::sycl;
    float A[1024], B[1024], C[1024];
    {
        buffer<float, 1> bufA { A, range<1> {1024} };
        buffer<float, 1> bufB { B, range<1> {1024} };
        buffer<float, 1> bufC { C, range<1> {1024} };

        queue myQueue;
        myQueue.submit([&](handler& cgh) {
            auto accA = bufA.get_access<access::read>(cgh);
            auto accB = bufB.get_access<access::read>(cgh);
            auto accC = bufC.get_access<access::write>(cgh);

            cgh.parallel_for<class vector_add>(range<1> {1024}, [=](id<1> i) {
                accC[i] = accA[i] + accB[i];
            });
        });
    }
    for (int i = 0; i < 1024; i++)
        std::cout << "C[" << i << "] = " << C[i] << std::endl;
}
```

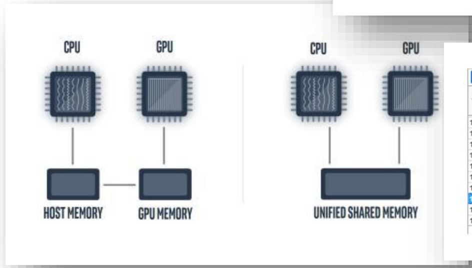
Write-buffer is now out-of-scope, so kernel completes, and host pointer has consistent view of output.

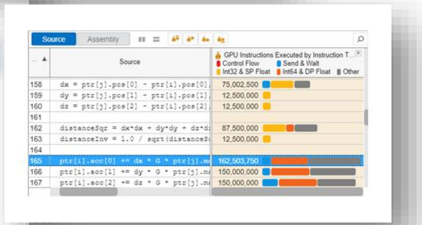
Data Parallel C++ (DPC++) Essentials Training






sub-group of 4 work-items
dimension 1 of work-group
dimension 2 of work-group
dimension 0 of work-group
Work-group
work-group of (4,4,4) work-items
dimension 1 of ND-range
dimension 2 of ND-range
dimension 0 of ND-range
ND-Range





...	Source	GPU Instructions Executed by Instruction T... (🟥 Control Flow, 🟦 Send & Wait, 🟨 Int32 & SP Float, 🟧 Int64 & DP Float, ⬛ Other)
158	dx = ptr[j].pos[0] - ptr[i].pos[0];	75,002,500
159	dy = ptr[j].pos[1] - ptr[i].pos[1];	12,500,000
160	dz = ptr[j].pos[2] - ptr[i].pos[2];	12,500,000
161		
162	distanceSqr = dx*dx + dy*dy + dz*d	87,500,000
163	distanceInv = 1.0 / sqrt(distanceS	12,500,000
164		
165	ptr[i].acc[0] += dx * G * ptr[j].m	162,503,750
166	ptr[i].acc[1] += dy * G * ptr[j].m	150,000,000
167	ptr[i].acc[2] += dz * G * ptr[j].m	150,000,000

Start Learning DPC++

Get hands-on practice with code samples in Jupyter Notebooks running on Intel® DevCloud

Learning Modules for Developing in DPC++

- DPC++ Program Structures
- DPC++ Unified Shared Memory
- DPC++ Sub-Groups
- Demonstration of Intel® Advisor
- Intel® VTune™ Profiler

software.intel.com/content/www/us/en/develop/tools/oneapi/training/dpc-essentials.html

Notices & Disclaimers

Texas Advanced Computing Center (TACC) Frontera references
Article: *HPCWire: Visualization & Filesystem Use Cases Show Value of Large Memory Fat Notes on Frontera*.
www.intel.com/content/dam/support/us/en/documents/memory-and-storage/data-center-persistent-mem/Intel-Optane-DC-Persistent-Memory-Quick-Start-Guide.pdf
software.intel.com/content/www/us/en/develop/articles/introduction-to-programming-with-persistent-memory-from-intel.html
wreda.github.io/papers/assise-osdi20.pdf

intel®